Immersive 3D Scene Prototyping in Virtual Reality

Jack Roper

MEng Computer Science

Supervisor: Anthony Steed

Special thanks to Daniele Giunchi

March 29, 2017

Abstract

As Virtual Reality provides a vastly different and more immersive user-experience than conventional desktop interaction, it also offers new ways of exploring how best to design and develop 3D virtual environments. With the recent release of consumer technology such as the HTC Vive and Oculus Rift, an increased number of VR applications and games are in development today; however, there remains disparity between how these applications are designed and how they are experienced by the end-user. To assist developers in prototyping a scene in VR, this project introduces a new 3D editor with the aim of maintaining immersion and experimenting with different approaches to optimising the creator's workflow. This includes the integration of features such as speech recognition, gesture detection, automatic object placement and web-enabled content search. User feedback is presented for assessing the effectiveness of these approaches, while further critical analysis is levied on the software-engineering aspects of the project. Future work includes objective comparison of the editor with similar tools — both desktop and VR, experiments that use the editor as a baseline for measurement, and an overall improvement and extension of the editor's features.

Contents

Chapter 1: Introduction

This section outlines the project's aims and goals against the backdrop of the overarching problem and its domain. It also discusses, at a high-level, the project's approach to solving this problem.

1.1 Motivation

This project's domain is centered within the field of virtual environments, overlapping with games industry areas such as level design and rapid prototyping. VR dramatically changes the way users interact with the virtual environment, demanding special concern from content creators interested in optimising the end-user's experience. In creating a VR application or game, developers may typically work in a desktop editor, using familiar interaction methods such as keyboard and mouse input. In order to experience the content as the user would, developers must then don their equipment and context switch from the desktop into virtual reality. To then make any necessary adjustments or add additional content, the developer or designer has to first disengage from virtual reality, before switching back to their working environment.

1.2 Problem Description

The problem with switching between desktop editing and virtual reality viewing is twofold:

1. It costs time and disrupts the creator's focus.
2. Creation and experience are fragmented, causing separation between designers and end-users.

Convenience is a paramount concern of modern technology, influencing the ways both individuals and businesses operate. Previews costing up to several minutes of physical and cognitive effort discourage designers from frequently assuming the end-user perspective, losing them the chance to gain formative feedback in return for their efforts. This problem is ubiquitous in the field of design, owing to its dependence on potentially many iterations where the emphasis is on exploration of ideas [23].

1.3 Aims

This project aims to overcome a fundamental difference in how VR applications are developed and how they are experienced by end users. In doing so, it offers designers an alternative perspective and method of interaction with the virtual environment. Concretely, these aims can be further broken down as the following:

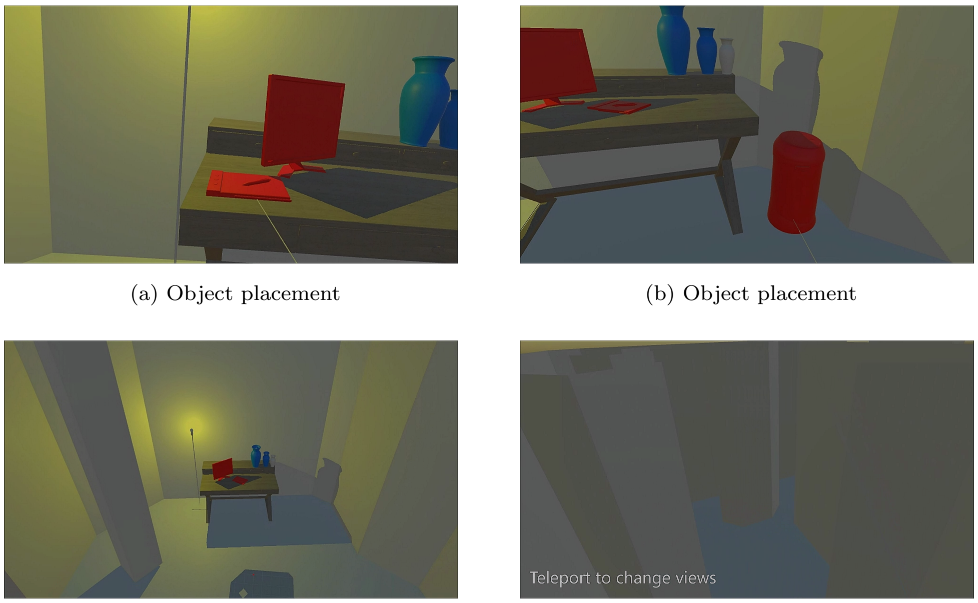
1. Support rapid prototyping with a workflow optimised for VR.
2. Facilitate the creation of virtual environments/scenes in VR.
3. Preserve immersion throughout this process so that users need not leave VR to prototype.

1.3.1 Rapid Prototyping

Traditional desktop editors are heavily optimised to take advantage of the interaction methods at their disposal. Certain tasks, such as those requiring great precision or keyboard-like input, are comparatively strenuous in VR. This project aims to integrate different approaches to overcoming this issue, using features such as speech input and gesture detection to complement user's editing experience and optimise their workflow for VR. Furthermore, the faster these scenes can be created, the more time is left for evaluation and revision [41]. In this way, this project makes no attempt to reproduce the feature-set of other editors, VR or otherwise; it is primarily concerned with the exploration and integration of VR-optimised approaches. This includes speech input, gesture detection, object recommendation and web-enabled object search from within the editor.

1.3.2 Scene Creation

In its simplest form, this aim entails the addition, manipulation and removal of 3D objects within a scene. However, scene topographies can vary, influencing how well a single editor can perform when trying to accommodate different types of environments. For example, it is common to make a distinction between interior and exterior scenes. The decision between these will not only have a dramatic impact on the types of objects likely to be used, but also on the technical constraints in operation [4].



(a) Object placement (b) Object placement

(c) Studio scene (d) Cave scene

Figure 1.1: Examples of scene creation

Games in particular are limited by their need for real-time performance and often have to make savings in some areas to avoid performance degradation. Draw distance, level of detail and the desired type of lighting are just a few characteristics that distinguish an interior scene from an exterior one.

For this project, the focus is on interior scenes; however, this is paired with emphasis on rapid prototyping, which typically relaxes concerns about performance and high fidelity visuals.

1.3.3 Immersion

It is critical that the user need not switch between VR and conventional desktop interaction while using the editor. Any such context switch works to the detriment of the project's stated aims and indicates a poor workflow that the user is trying to circumvent.

Of equal importance, is the definition of immersion in this case: it describes the capability of the technology to accurately provide a range of stimuli; it does not describe the psychological effect on the user [51]. If the technology supporting the editor fails to satisfy the requisite level of immersion, then the task may become difficult, boring or slow, as the experience fails to live up to the user's intuition.

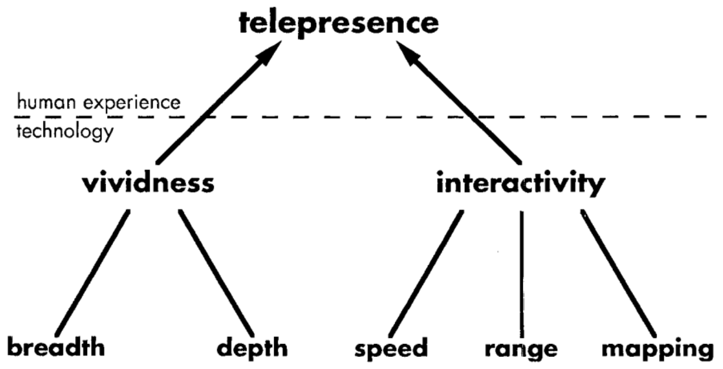


Figure 1.2: Achieving presence [55]

Presence, the psychological effect, is not a problem in this case, as believability constraints may obscure the best way to accomplish a design task. Being able to move through objects or switch views [23] allows the editor to be used earlier and more than a typical walk-through tool concerned with presence [6]. Figure 1.2 illustrates why this project's focus on interactivity and low-fidelity prototyping reduces its ability to encourage presence in the user. As a low-fidelity prototyping tool, it does not offer a rich or accurate representation of the environment (vividness). Although the editor makes no attempt to stylise objects on the user's behalf, the user's experience remains subject to the style and appearance of these objects. Styles can both diminish and heighten photo-realism, and in turn have an effect on the user's sense of presence [54].

1.4 Deliverable Goals

This project aims to deliver a complete system for prototyping scenes in Virtual Reality. To this end, the deliverables are separated into two components:

1. A desktop application for rapid and immersive scene prototyping in VR.

2. A web server that supplies 3D models to the desktop application.

Table 1.1 lists the initial set of user story epics used to capture the above goals. These stories are mostly targeted at specific kinds of users, addressing individual goals they would have in using the editor. As epics, these stories are large and encompass numerous sub-goals. Table 1.2 presents these stories.

Epic	Description
1	I want to prototype an environment in virtual reality.
2	As a designer, I want to arrange objects and move amongst them.
3	As a designer, I want to plan and visualise assets or layouts that do not exist yet.
4	As a developer, I want to add and observe simple object behaviours.
5	As a developer, I want my created levels to persist and be compatible with external tools such as Unity.
6	As an artist, I want to experiment with different material properties.

Table 1.1: Initial User Stories (Epics)

User Story	Description	Epic
1	Users can view the world in virtual reality.	1
2	Users can add/remove objects to/from levels.	2
3	Users can translate, rotate and scale objects.	2
4	Users can import custom 3D models for viewing.	2
5	Users can walk around the world.	2
6	Users can highlight areas and mark boundaries.	3
7	Users can switch between views.	3
8	Users can enable and customise collision and physics behaviour.	4
9	Users can define simple logical behaviours such as movement or spawning.	4
10	Users can save and load from a file.	5
11	Users can import levels into Unity as scenes.	5
12	Users can adjust material properties such as texture and colour.	6
13	Users receive recommendations based the objects they place in a scene.	2
14	Users can use speech to perform commands such as undo/redo.	1
15	Users can use search for models that they do not already have.	3

Table 1.2: User Stories (Stories)

Over the course of the project, these user stories were refined in accordance with changes in direction and scope refinement. For example, a greater emphasis was placed on optimising workflow, while stories related to the addition of logical behaviours became out of scope. Stories highlighted in red were removed over the course of this revision, while those in green were added in their place.

1.5 Approach

This project's approach can be split into two focal points: one providing a functional method of organising objects in a scene and representing the result as file; the other supports this process by attempting to optimise the user's workflow in VR.

1.5.1 Organising Objects

A scene is represented by a JSON formatted file, detailing the scene's object and component hierarchy. This includes a list of nested objects that together make up the scene. Objects can be one of two things:

1. 3D models stored as .obj files.

2. Other scene files.

This yields a recursive definition of scene, aiming to encourage modular use of scenes, which can be shared online. Figure 1.3 presents this model as a conceptual diagram.

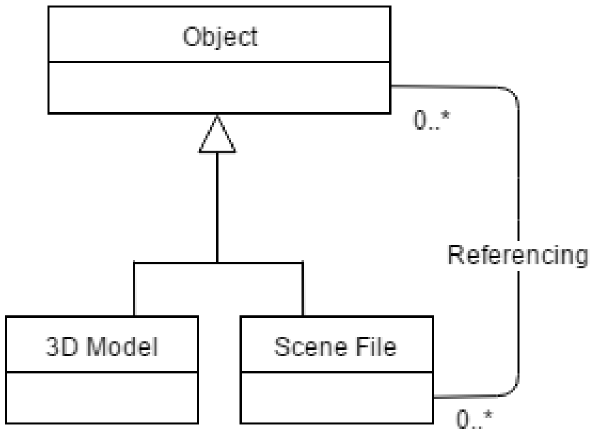
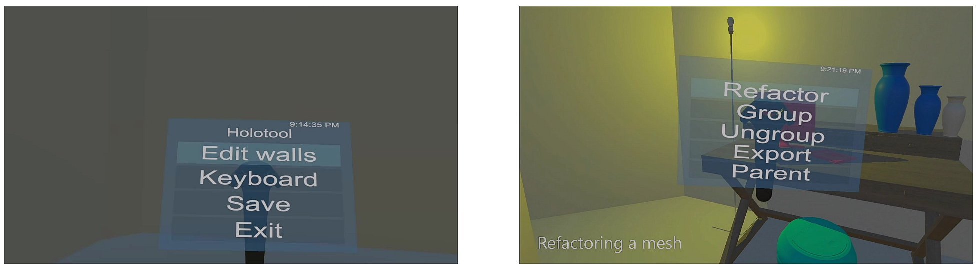


Figure 1.3: Scene Model

Scenes files are imported and the result displayed to the user. As the user interacts with a scene, they alter properties of objects such as position and size. These properties manifest themselves as components, with each component being imported and exported as required. Modifiable properties include:

- Object position, rotation and scale
- Object mesh (3D model)
- Object material (texture, colour etc.)
- Parent/child relationships
- Groups for refactoring multiple objects at once
- Whether or not the object is affected by gravity
- Collider type, size and offset

Note: While this list does not include everything one is likely to come across when producing a virtual environment, it represents a functional subset deemed suitable for the prototyping phase.



(a) Scene menu (b) Object/group menu

Figure 1.4: 3D menu interface

The user adjusts these properties through a combination of hand movement, speech and GUI controls (figure 1.4), where the GUI leverages the HTC Vive controller's array of input mechanisms. This mixture of input methods aims to map each task to the most appropriate mechanism. For example, moving and rotating an object using speech input would be a laborious task; however, it is natural fit for hand-based interaction.

1.5.2 Optimised Workflow

Aside from mapping tasks to the appropriate control mechanism, optimisations for VR include:

- **Modular inclusion of other scenes**
  Including other scenes allows users to re-use previous work, composing scenes with others. For example, one could design a classroom using several desks and chairs, having to arrange each individually. However, it would be easier to do so using multiple instances of a desk and chair scene.
- **Speech commands**
  Speech input allows us to free the users hands and adopt an intuitive interface for certain convenience commands such as undo/redo. Careful consideration of what the user is expected to say is needed to avoid over-taxing their memory and to reduce recognition error.
- **Object search via Web API**
  With the many editors available today, a question of how best to distribute content is raised. To avoid users having to rely on just what they already have, the project's web server allows users to search for, and download, objects that others have shared. This aims to encourage the kind of re-use and experimentation befitting a rapid prototyping system.
- **Object recommendation**
  Methods for reducing the need for menu interaction in VR are well worth considering, as they offer savings in time, as well as both cognitive and physical effort. To assist users with choosing the next object to add, the editor recommends objects of interest based on analysis of existing scenes.

- **Customisable gesture recognition**
  Users can define one-handed gesture patterns, which are learned by the editor and associated with some prototyping functionality.

Together, these measures aim to reduce the need to leave the editor at any point, as well as speeding up object interaction in the scene.

Chapter 2: Background

2.1 Related Work

The project was first inspired by a short-film titled World Builder (2007), illustrated in appendix A. This section is dedicated to comparing and contrasting this project with others, both commercial and academic in nature.

2.1.1 Unity Editor VR

Unity themselves are experimenting with an open system intended to place designers in the middle of their creations, closer to the user's perspective. Unity's editor already presents a well-polished interface not unlike the traditional editor.

Unity's editor offers some powerful tools such as animation recording, profiling in VR and the free manipulation of menus as objects. Appendix B presents how some of these features appear to the user. Both Unity and this project are concerned with the manipulation of objects and both have to tackle the problem with menus in VR. However, the editors differ in that Unity aims to replicate many features of its traditional desktop editor, while this project's focus lies entirely with (low-fidelity) prototyping. Recall that this project's third aim is to support rapid prototyping with a VR optimised workflow. To this end, this project incorporates convenience features such as: object snapping, procedural generation of wall geometry, gesture detection, speech input, web-provided object search and object recommendation. Such features are currently not included with Unity's editor, which uses higher-quality lighting and effects. It also provides fine-grain access to component variables, which are comparatively abstracted in this project.

2.1.2 Drawing and CAD in VR

A variety of VR-enabled drawing, modeling and sculpting tools have been released already. MakeVR presents a CAD engine, made accessible via a two-handed interface. Users interact with both primitive and complex shapes, modifying them via booleans, sweeps, deformation, and other CAD operations [22]. Similarly to this project, MakeVR (appendix C) allows users to construct scenes, but it also allows them to 3D model individual objects, affecting not just their position and scale etc., but their underlying geometry. Oculus Medium allows users to sculpt and paint models using clay-like modeling tools; Tilt brush is a painting application that enabled room-scale paintings in VR; 3D modeling in VR has also been sought for conducting engineering surveys [9].

CAD-like tools and this project's editor share relatively superficial similarities such as scene organisation, object manipulation and material adjustments. However, they accomplish these goals in very different ways. While VR modeling targets the underlying geometry, this project operates on objects that have already been modeled, only modifying geometry to generate walls procedurally. Ultimately, the tools work at different stages of the 3D content pipeline: MakeVR delivers models and scenes from scratch, while this project uses such models for prototyping higher-level scenes, where geometry can be mostly forgotten.

2.1.3 Alice

Alice is another VR framework intended to ease the process of rapid prototyping in VR [42]. Unlike the previous, commercial examples, Alice lets users write object-oriented programs that offer a lot of flexibility to users at runtime. Users can either grasp and manipulate objects, interactively evaluate program code fragments or manipulate GUI tools to effect change to the scene. A comparison between Alice and this project shows that both have the same aim regarding rapid prototyping; however, Alice's novel element exists in its interactively interpreted code editing, while this project aims to avoid users having to leave the VR editor for any reason.

2.2 Research

This section approaches some of the major topics of research visited over the course of the project. It is not an exhaustive list, including only that which was used to significantly direct and/or inspire the editor's development.

2.2.1 Scene Organisation

In the interest of both performance and design convenience, virtual environments are often broken up into logical groups of objects. A group can be referred to as a "scene", effectively partitioning the greater virtual environment. Scenes allow for reduced loading time, as only a subset of assets are required to load a particular scene. Isolated groups of objects like these form the basis for locale based travel and scalability techniques [2], decomposing a vast and inoperable virtual environment into manageable pieces for collaboration and/or performance benefits. Scenes are also useful for representing an aggregation of objects, allowing users to perceive and interact with the environment at a higher level of granularity [44].

Understanding the organisation of scenes involves studying methods of representation, such as scene graphs [50], as well as how they can be analysed and related to one another. Xu et al. [62] combine pattern mining and subspace clustering to yield a set of representative substructures called focal points. These focal points are comparable and deal well with hybrid scenes that are otherwise difficult to semantically label.

Scene organisation is of particular relevance to this project, as semantic analysis of a scene can support environment designers with recommendations or shortcuts based on the current or past state of their work. Ehinger et al. found that man-made scenes are often semantically defined by social function, while outdoor scenes may be sorted by terrain features [14]. Based on this, if the editor were able to understand that a particular arrangement of chairs and tables correlates with dining rooms, it could easily replicate this arrangement later or even discover similar ones to be recommended for use another time.

2.2.2 Locomotion

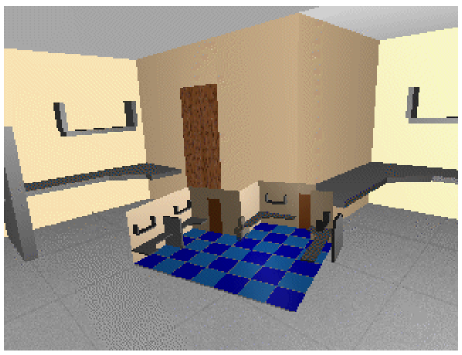
For the advantages of virtual reality to be fully realised, virtual environments must be able to extend beyond the non-virtual space's borders. Limits in tracking range and physical space serve to confine natural locomotion, which best conforms to the user's internal proprioceptive model. Advances in affordable room-scale tracking partially mitigate this issue; however, these do nothing to widen the physical space available: a problem that will persist in spite of tracking improvements.

Methods to achieve artificial locomotion include the use of joysticks, point or gaze-directed steering, as well as teleportation [51, 53]. The lattermost has become popular for its reduced likelihood to cause VR sickness, but also because it allows users to focus on the task at hand [7]. However, teleportation can reduce the user's spatial awareness [6] as it involves a disruptively sudden change in perspective.

For this project, joystick movement was quickly ruled out for over-burdening the hands and for its tendency to cause dizziness in users. Between point and gaze-directed steering, point presents the more convincing argument as gaze-direction is too restrictive in that it dictates where the user must look [6]. Finally, teleportation was selected for its simplicity and because it poses minimum risk to the user and so can be used for extended periods of time without pause.

2.2.3 World In Miniature

The introduction of virtual reality necessitated the application of new approaches for common tasks such as navigation and control. Metaphors involving a mouse and keyboard, the long dominant method of input for conventional computing, are largely incompatible with the demands of virtual reality when trying to offer place illusion (presence) [49]. This can be seen with locomotion, but the problem also affects object manipulation and viewing, where the need to adjust global scale and view angle is imperative [8, 35].


(a) World In Miniature [56]

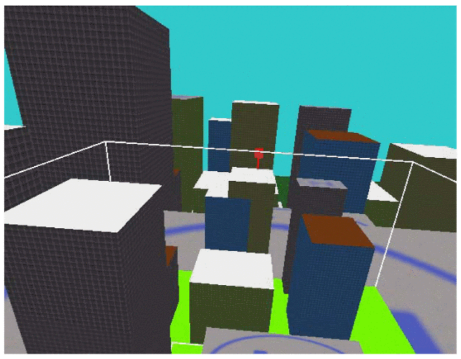

(b) Scaled Scrolling World In Miniature [59]

Figure 2.1: World In Miniature Models

A world in miniature (WIM) attempts to extend a standard virtual reality view with a smaller, possibly hand-held, representation of the environment. This WIM can be further scaled and scrolled to make interaction more efficient, as in figure 2.1. This project adopts the WIM concept in part, using it for large-scale interaction that would otherwise be too laborious to carry out manually.

Chapter 3: Context

3.1 Context Diagram

A high-level overview of the system context is presented in figure 3.1. It is intended to show which parts of the real world the system interacts with, briefly describing the phenomena shared between agents and how much of this falls within the project scope. The reader may refer to 6 for a detailed description of the implemented system's architecture. Through use of the twin peak model [38], requirements and architecture have been interleaved; however, project initiation first addresses the system context, independently of its implementation.

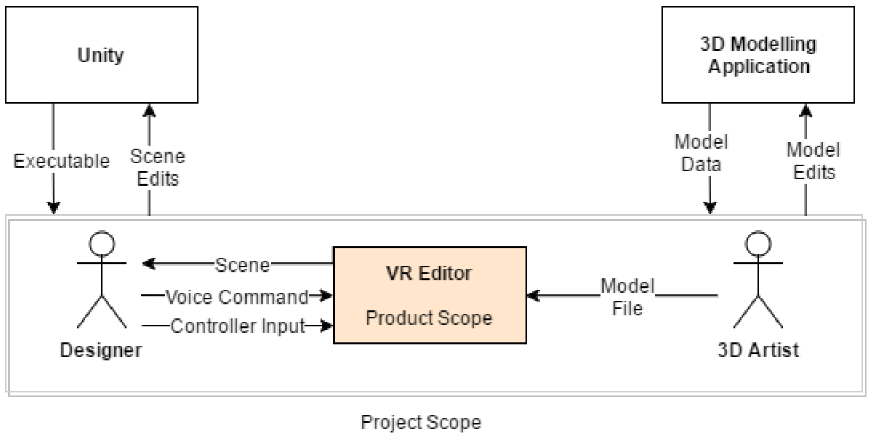


Figure 3.1: Context Diagram

In summary, the system has the following three key functions that operate on shared phenomena:

1. Interpretation of user commands (controller/voice)
2. Production of 3D scene environment viewable in VR
3. Acceptance of user-supplied 3D models

Specifically, the system comprises two separate components: a desktop application and web server. The desktop application fields commands from the user, producing a scene file that defines a virtual environment prototyped in VR. This scene could then be reused in another application, such as Unity, to transition from prototype to final product.

The web server is tasked with accepting 3D model file uploads from users and storing them online for use with the desktop application. Communicating via a RESTful API, the web server then delivers 3D models to the desktop application on request, filling any gaps in the local environments 3D model collection.

3.2 Stakeholders

In all software development, it is important to ascertain who the project's stakeholders are. To this end, appendix D illustrates a stakeholder onion model, where the system's primary users are located within the containing business layer.

3.2.1 Containing Business

The containing business layer consists of stakeholders that stand to functionally gain from the product.

- **Game/level designers**
  Many game and level designers that are working on VR applications use tools to produce content. This effort stands to benefit from an editor that empowers them to design and interact with content from the user's perspective.

- **3D artists**
  Similarly to designers, artists need a way of viewing their work. Ideally, they should be able to see it exactly as end-user would, making an editor that allows them to do so while experimenting with different materials or layouts attractive.

- **Researchers**

  Researchers interested in mixed reality will be able to use the editor as either a starting point for their work, or as a baseline provider when researching new methods of display and/or interaction.

3.2.2 Operational Area

The operational area includes stakeholders that directly operate and maintain the product.

- **Product maintainers**

  Maintainers are responsible for addressing bug reports related to the system. They operate directly within the product scope.

- **Web hosting provider**

  Any hosting provider shares in the success of its hosted applications, as increased traffic provides business and revenue opportunities for hosts.

3.2.3 Wider Environment

Members of the wider environment usually have indirect influence or interest in the product. In this case, they are not users of the product, but are instead connected to others that do use it.

- **Unity Technologies**

  As the engine used to build the editor, Unity has an interest in the editor's success, as it would increase awareness and potentially generate a source of income were the project to be deployed commercially.

- **Video game players**

  Video gamers have an interest in the tools used to make the games that they play, since these tools often influence the stylistic and mechanical nature of a game.

- **Virtual reality enthusiasts**

  Early adopters of VR are faced with a growing number of VR applications; however, against the myriad of general software applications available, this number remains comparatively small. Therefore, enthusiasts looking to experience VR products may have an interest in this project's outcome.

3.3 Constraints

Below are listed the major constraints affected the project, including evaluation of each one's measured significance.

Schedule

A major constraint for this project is the time available. Around fifteen weeks have been allotted for all stages of development, calling for a pragmatic outlook about which parts of the project will need to be prioritised.

Budget

There is no set budget for this project, encouraging the use of free or open source technologies to avoid cost where possible. Generally speaking, there is a large variety of free tools and academic licenses available, limiting the effect of this constraint to just the deployment view (where hosting and maintenance costs are harder to avoid).

Team size

As an individual project, there is only one active developer on this project. This is a major limiting factor and, like the schedule constraint, will require significant attention to avoid over-stretching the project's scope. Application of agile development principles and focusing on a minimum viable product (MVP) will help to mitigate this constraint.

Technology

As a virtual reality project aiming to leverage the recent resurgence in VR enthusiasm, the product must integrate with current technologies. Namely, the head-mounted display in use for this project is the HTC Vive, constraining choice of technologies to those that support it.

3.4 Risks

The project risks can be derived from the project goals and constraints. For each risk, there is an approximate measure of its likelihood and impact on the project's outcome.

Imprecision

If the editor fails to allows users to carry out tasks with the necessary precision, it may be deemed unsuitable for application to real projects or in industry. The likelihood of this threat is medium to low, depending largely on the users needs and the editor's chosen interaction methods. In mitigating the risk this poses, methods of locomotion and interaction have been chosen to minimise incompatibility with precise placement. Object snapping and speech-enabled undo/redo similarly aim to facilitate the editing process.

Virtual Reality Sickness

Sickness is a pressing concern for many users in VR. Discrepancies between what the user perceives and what the VR application shows can lead to dizziness, fatigue and/or sickness. Fortunately, this threat's likelihood can be kept low by avoiding methods that misalign visual perception with the user's proprioception. Using teleportation for locomotion helps to answer this risk.

Scope

As the editor is focused on exploring new methods of interaction in VR, the project's path may be influenced by the success and failure of these approaches. It is important to maintain a clear vision for the project and focus on a core set of beneficial features.

3.5 Technologies

Below is a brief discussion of the main technologies used for this project.

3.5.1 Virtual Reality HMD

Choosing between two of the major VR headset suppliers: Oculus and HTC, it was decided that the HTC Vive was preferable owing to the greater variety of input available at the time. Namely, the Vive includes two motion-tracked controllers with various buttons and triggers to support player input. Since then, Oculus has released a similar product called Oculus Touch, which satisfies the same goal. Ideally, there would be little distinction from this project's perspective, as typically such devices are unified behind a common interface such as OpenVR. However, this remains an ongoing process, motivating the use of a game engine that provides ready support for both devices.

3.5.2 Game Engine

Unity has fast become one of the most popular game engines available. Developers can download and work in Unity for free, making it an ideal engine given this project's budget constraint. Furthermore, Unity also represents a de facto standard in unifying support for various VR devices. It is simple to get underway with VR in Unity, which provides a camera metaphor familiar to those working in Computer Graphics or related fields.

3.5.3 Version Control

For managing changes to the project, a version control system is necessary. As a tried and tested version control system, git was selected for use with this project. For convenience and robustness, BitBucket is used to host a remote instance of the project repository.

3.5.4 Web Server and Online Storage

As the web server is intended to support the desktop application, it was important to choose a language and hosting provider that involved as little setup as possible. With this in mind, Heroku was chosen for its Platform-As-A-Service (PaaS) model. This eliminates the need for server configuration or complicated deployment: instead, deployment is automated behind a git push command. As one of Heroku's supported runtimes, the server is written in Node.js, acting as a proxy for an Amazon S3 bucket that provides storage for user file uploads.

3.5.5 SteamVR

SteamVR is a Unity package that enables VR developers to quickly get underway by offering head and hand tracking out of the box. It also integrates with Steam's chaperone system, which tries to prevent users from straying outside their predefined play area.

Chapter 4: Requirements

This section posits a list of functional and non-functional (quality) requirements, which serves as reference for the project's intended behaviour. The following requirements, written using the KAOS pattern, were derived from the user stories illustrated in figure 1.2 above. They will be used during acceptance testing to evaluate whether the end product satisfies the project's functional and non-functional objectives.

4.1 Functional Requirements

blocking mode A blocking mode includes any state where either the normal controls for performing an action are mapped to a different action or where action is not fit to perform.

[**R1**] **Req** Achieve[Object addition]
Given the user is not in a blocking mode
When the user triggers object addition
Then the last added object is again added to the scene
Or object search is initiated
Note: Adding the last object if possible is a convenience measure to avoid unnecessary menu interaction when object repetition is desired.
Architecturally significant: Yes: object addition can trigger more than one outcome and will need to account for potential delay in object search and/or load time.

[**R2**] **Req** Achieve[Object removal]
Given the user is not in a blocking mode
When the user triggers object removal
Then the indicated object is removed from the scene
Note: Object removal may not strictly remove the object (it may be disabled for quickly withdrawing the removal), but will ensure that it can no longer affect other objects in the scene or be visible to the user.
Architecturally significant: No: removal is accomplished simply by disabling/destroying an object. Any complexity can be attributed to other requirements that dictate how the object must be removed.

[**R3**] **Req** Achieve[Object transformation]
Given the user is not in a blocking mode
When the user attempts to grasp an object in VR
Then the object is anchored (position and rotation) to their grasping hand
And the object is scalable using both hands
And the object may be passed between hands
Note: Object transformation is well-suited for hand-based interaction.
Architecturally significant: Yes: transformation involves more than just the user's hands: target disambiguation and state management are both needed as well.

[**R4**] **Req** Achieve[Object loading]
Given objects are available for selection
When the user chooses an object to add
Then the object is loaded as a single 3D model

Or the object is loaded as a scene (group of objects)

And the object is visible in the scene

Note: Frame-rate is key in VR. Measures will have to be taken to avoid a significant drop in frame-rate when loading an object from disk.

Architecturally significant: Yes: loading a 3D model or scene file requires interaction with the file-system (and potentially untrusted input).

[R5] Req Achieve[Natural locomotion]

Given true

When the user physically moves in non-virtual space

Then the movement is reflected in virtual space

Note: This is mostly taken care of by SteamVR.

Architecturally significant: No: SteamVR simplifies this requirement, doing so as part of Unity's component system so that it is easy to integrate with any architecture.

[R6] Req Achieve[Artificial locomotion]

Given the user is not in a blocking mode

When the user elects to move to another position in VR

Then the user teleports instantly to that position, maintaining the same ground offset as before

Note:

Architecturally significant: No: teleportation can be implemented simply enough using raycasting, and it is easily decoupled from other actions.

[R7] Req Achieve[Persistent scenes]

Given a scene exists

When the user elects to save a scene

Then the scene is serialised to disk

And the scene can be deserialised in future sessions

Note: Without persistence, users could not recover their work or export it for use in another application.

Architecturally significant: Yes: serialisation and deserialisation necessitate interaction with the file-system and involves deep traversal of the scene hierarchy.

[R8] Req Achieve[Collision behaviour]

Given an object is designated as collidable

When the object is added to the scene

Then the object collides with other objects that are similarly collidable

Note: Collision behaviour must persist between sessions, so will need to be recorded in the scene file.

Architecturally significant: No: given an architecture that supports serialisation and deserialisation, the addition of persistent collision behaviour is not significant. Collision detection is implemented by Unity.

[R9] Req Achieve[Physics behaviour]
Given an object is designated as a physics object
When the object is added to the scene
Then the object is affected by gravity and its mass and drag are simulated
Note: Physics behaviour must persist between sessions as well.
Architecturally significant: No: given an architecture that supports serialisation and deserialisation, the addition of persistent physics behaviour is not significant.

[R10] Req Achieve[Unity scene import]
Given a scene file exists
When the scene file is selected for import
Then an import tool loads the scene into Unity for the user to then edit
Note: This is a separate application to the editor, implementable as an extension to Unity itself or as a standalone tool that converts a scene file to Unity's own format.
Architecturally significant: Yes: this requires interfacing with Unity and is a large component in its own right, dramatically impacting the architecture overview.

[R11] Req Achieve[Material adjustment]
Given an object exists and is targeted
When the user elects to change the texture or colour of the material
Then the texture is loaded and applied
Or the colour of the material is updated
Note:
Architecturally significant: No: any architectural complexity can be attributed to requirements that dictate how the material is updated. Unity materials can otherwise be updated easily enough without architectural demands.

[R12] Req Achieve[Object recommendation]
Given non-empty scenes exist
When the user adds an object to the scene
Then the editor uses saved scenes to recommend what objects to add next
Note: Object recommendation is non-trivial and can be done in a variety of ways that range from low-level geometry-analysis [40] to high-level observation of existing data. See 1.5.2 for discussion of object recommendation.
Architecturally significant: Yes: object recommendation is a convenience measure that, however implemented, requires semantic analysis in order to relate objects somehow. This requires a separate component dedicated to just this

task.

[R13] Req Achieve[Speech input]

Given speech commands have been defined

When the user speaks a command

Then the command is invoked by the editor

Note: See 1.5.2 for discussion of speech commands.

Architecturally significant: Yes: speech input can be triggered at the same time as other inputs and requires the editor to always be listening for activation.

[R14] Req Achieve[Undo/redo]

Given an undo-able/redo-able action has been performed

When the user elects to undo/redo the action

Then the action is undone/redone accordingly

Note:

Architecturally significant: Yes: implementing undo/redo behaviour requires that we record state both before and after an action has been performed. Also, queuing these undo/redo actions and chaining them together requires further state management of architectural significance.

[R15] Req Achieve[Local object search]

Given objects exist

When the user searches their machine for objects

Then the editor displays the objects found

Note: Search in VR requires input not-suited to hand-based interaction.

Architecturally significant: Yes: searching for objects requires interaction with the file-system and may need to account for inaccuracy in the search criteria.

[R16] Req Achieve[Web-based object search]

Given objects exist in online storage

When the user searches for objects via the web

Then the editor requests the results from the web-server and displays them to the user

Note:

Architecturally significant: Yes: web search involves interaction between two independent applications, which requires that a protocol be established and that the editor accounts for the usual problems of networking such as increased latency and connectivity problems.

4.2 Quality Requirements

Quality requirements are concerned with the non-functional behaviour of the system. Important ones are listed below.

4.2.1 Performance

Performance is important in VR for maintaining a consistent frame-rate and avoiding VR sickness. Therefore, certain measures must be taken to reduce impact of expensive operations, following a traditional game development approach where CPU and memory usage should remain as consistent as possible throughout the session.

[P1] Performance Req Achieve[Minimum of 30fps]
Given true
When the editor is running
Then the frame rate does not drop below 30fps
Note: Although this requirement is impossible to guarantee, owing to the application's dependence on a shared and likely non-real-time operating system, it is important to strive for a consistently high frame rate in VR. 30 frames per second is a fairly low a target to aim for; however, it is a conservative lower bound, with research indicating that 15 frames per second is the threshold for performance degradation [10].
Fitting criteria: *Throughput:* must render at least 30 frames per second. *Responsiveness:* user interaction with the world must not be interrupted due to a severe drop in frame rate.

[P2] Performance Req Achieve[Resource caching]
Given a resource, such as a mesh, texture or material, is used
When the same resource is needed again
Then it is retrievable from a cache
Note: Loading resources such as 3D models or textures from files is an expensive process. It is typical in game development to cache as much as possible to avoid sudden alterations in frame rate. This requirement lends itself to satisfying P1.
Fitting criteria: *Throughput:* the cache need only handle one request at a time. *Responsiveness:* the cache should prioritise popular resources to minimise time spent waiting for resources to load.

4.2.2 Availability

These requirements mostly target the web-application serving objects to clients using the editor.

[A1] Availability Req Achieve[Partial failure mitigation]
Given true
When either the editor or the web app fails
Then this failure does not propagate to the other component
Note: As the focus of the project is the editor, this requirement does not extend to failure in the object database or

object storage. A failure in either of these may lead to failure in the web app; however, this must not lead to failure in the editor as well.

Strategy: The editor's requests to the web app will be non-blocking, asynchronous requests. The web app will provide a stateless API, ensuring it is not affected by client-side failure.

[**A2**] **Availability Req** Avoid[Data loss]

Note: The database storing object details, as well as the object storage itself will require redundancy measures to avoid data loss in the event of failure.

Strategy: Data could be replicated using RAID, taking periodic backups when traffic is at its lowest.

[**A3**] **Availability Req** Avoid[Service downtime]

Note: Downtime should not exceed 0.01%, corresponding to 4 nines service level.

Strategy: Forecasted load balancing between deployed instances. Instances could be scaled up or down preemptively, or in response to demand.

4.2.3 Security

Security is not the focus of this project; nonetheless, standard measures should be taken to avoid disruption of service, as well as to protect users' data. Before addressing the security requirements, it is important to understand what assets this project has to protect.

User object files

Object files are uploaded by users and stored online for later retrieval. While these files are made publicly accessible via the API, attackers should not be able to circumvent the restrictions laid out by this interface. They should also not be able to disrupt access for other users.

Separately stored object details

While objects are stored in an Amazon S3 bucket, details of the objects and where to find them are stored in a separate database as in figure 6.4. If these details are compromised, it would interfere with the web app's service and prevent users from searching for objects.

User local machine

As the editor will be installed on users' machines, it is critical that it not introduce security vulnerabilities that an attacker can leverage to gain access to or disrupt normal operation of the machine.

Given the assets above, we can define a clear access control policy with table 4.1. Additionally, a further security requirement is listed below.

Asset	**System Administrator**	**User**
Object files	Full access with audit	Read access via API
Object details	Full access with audit	No access
User local machine	No access	-

Table 4.1: Access Control Policy

[S1] Security Req Achieve[Database query sanitation]

Given a database with tables exists

When a query is made to the database

Then this query is sanitised to prevent database corruption

Note: Database queries have long been a problem when performed in combination with untrusted user input. SQL injection continues to be a prevalent bug in commercial systems [20].

Strategy: Never directly use user input for querying the database. Ensure that any supplied input is processed to escape sensitive characters and prevent query manipulation.

4.2.4 Evolution

Given time and team-size constraints, this project aims to deliver an MVP-solution. Therefore, it is necessary to accommodate future expansion that either adds new features or improves on old ones.

New states

- Extending the editor's behaviour should be accomplishable with the addition of new states.
- This should not require any replication of existing state behaviour.
- New states may use and share dependencies with existing states and do so under the assumption that these dependencies are, or appear to be, purely functional and stateless with regards to their use.

Testing

- Tests should be provided to ensure that newly added functionality does not break or interfere with existing code.
- Instructions on how to run these tests is provided as part of the system manual appendix item.

Documentation

- Documentation should be provided to support the installation, operation and understanding of the editor.
- This report should be considered to partially serve that role. Further documents are submitted as appendix items.

Chapter 5: Design and Implementation

This chapter focuses on providing details of the editor and web application's implementation. It starts with an overview of pervasive design principles applied during the project. It then provides an account of how the project was implemented from start to finish.

5.1 Design Principles

This section addresses some of the overarching design principles that have been extensively applied during this project. These principles are extensively used to complement one another throughout the editor's codebase.

Inversion of control

Inversion of control is a method of decoupling a component from its dependencies by shifting the responsibility of dependency management to the component's constructor or caller. This often manifests itself as the dependency injection pattern [16], where object instances have their dependencies injected upon creation, or via injection methods or properties. This is very different from patterns such as the service locator where a dedicated service is called to resolve dependencies by the component. Inversion of control and dependency injection provide advantages for testing, as they allow implementations to be swapped out or mocked as necessary. They also improve code readability, as just by looking at the class constructor it is immediately obvious what the object needs to do its job.

Interface-based programming

To further increase component decoupling, it is useful to separate implementation from the interface it provides. Again, this is useful for testing as we can easily switch to a different, perhaps test-specific implementation, without needing to alter the dependent code. It likewise benefits extensibility since new implementations are easy to integrate as long as they implement the required interface. The vast majority of the editor's internal component interaction happens using well-defined interfaces such as these.

Composition over inheritance

Lastly, it is often advantageous to do away with a highly detailed and deep inheritance hierarchy; such structures can be difficult to update or even understand. Instead, we can apply inheritance purely in cases where subclasses share common data or behaviour, creating potentially many shallow but wide inheritance structures. Object definitions are instead presented as a composition of dependencies, using the well-defined interfaces discussed above.

5.2 Project Timeline

This section conveys the key stages of the project development timeline. Each stage includes a motivation, description and discussion of the design and implementation choices made.

5.2.1 State-based Architecture

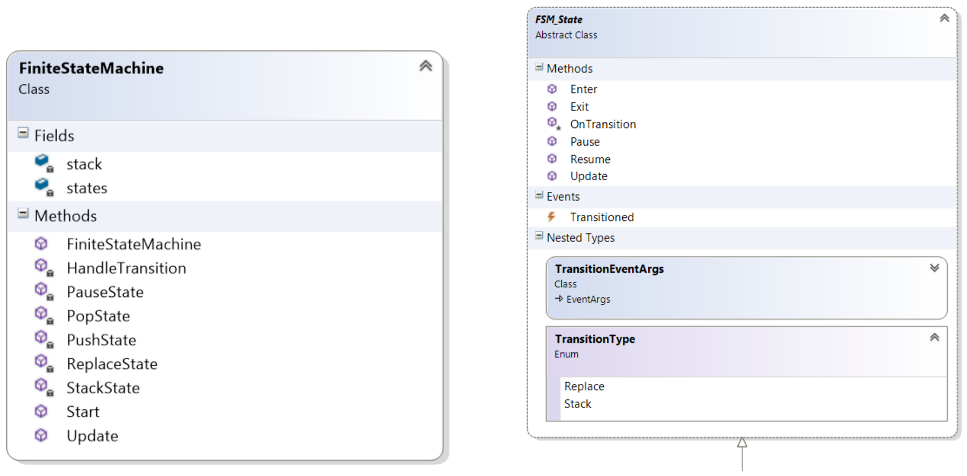
Motivation

The first task was to decide the architecture that would underpin interaction with the editor. Ad-hoc approaches tend to neglect extensibility: the ease with which a software system may be changed [34]. However, iterative development expects a partial-system to be available at the end of every stage [27]. Therefore, extensibility was the major focus of this decision, as the editor would effectively need to be extended after each iteration.

Description

Logically separating behaviours into distinct states is advantageous for the creation, debugging and testing of functionality. The State pattern applies this behavioural strategy, allowing an object to alter its behaviour according to some internal state [19].

For this project, almost all user-interaction is implemented within a specific state. *Navigation* (locomotion), *Transform* (object manipulation), *AddObject* and *Keyboard* are just a few examples. Each of these states inherits from an abstract base class, containing member variables and methods common to all states. The states themselves are then instantiated as objects, each having their dependencies injected upon creation. The state machine is a separate construct needed to house and act upon these states; it is the state machine that appears to change its behaviour dynamically at runtime according to whichever state is active.



(a) State Machine (b) Abstract State

Figure 5.1: State UML

Choices

- **Stack-based**

  Figure 5.1 indicates that the state machine operates like a stack. Originally, this was not the case: a state was either active or not, and the only way to enter a new state was to exit the previous one. While this is perfectly reasonable a lot of the time, it is convenient to be able to stack states in cases where the user wishes to accomplish more than one thing at a time. For instance, when the user selects a group of objects and wishes to export them as a scene, a filename must be provided via the keyboard user-interface. Using the stack, we can easily pause the underlying *Selection* state, temporarily ceding focus to the *Keyboard* state for the user to enter a name. Importantly, the *Selection* state's internal state is not lost, allowing it to continue from where it left off.

- **Transitions**

  Before rigorous application of dependency injection, transitioning to a new state was done on the fly, where the previous state was responsible for both creating and transitioning to the new one. This approach has two major flaws:

  1. State creation code is duplicated in lots of different places.

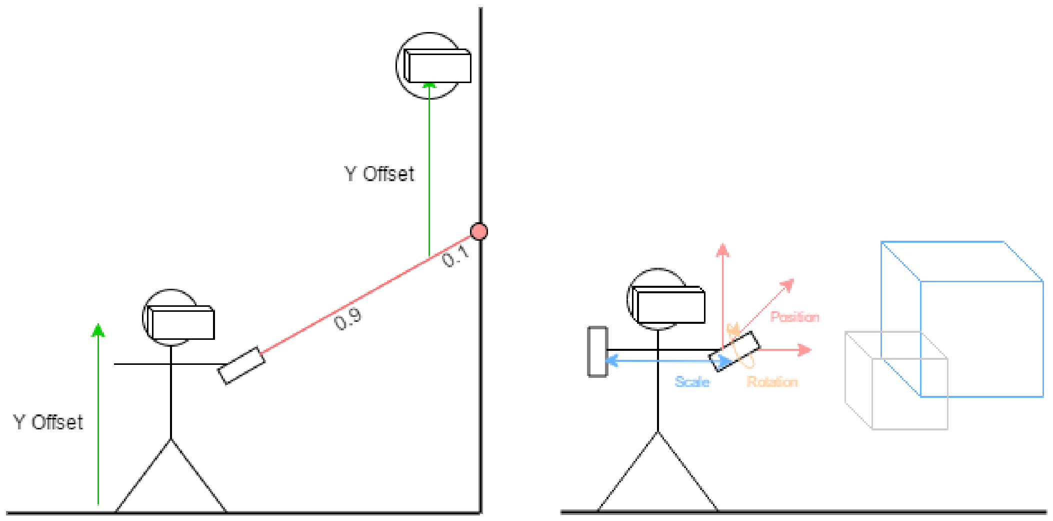
2. The previous state must intimately understand the new one, supplying everything it depends on to function properly, as well as the contextual parameters it needs to operate on.

Ideally, state-specific code should focus on its own purpose as much as possible. States should also not face the burden of knowing how to construct others. Adoption of dependency injection made this issue clear. The solution was to only create states once at the composition root (application entry point), storing them inside the state machine and using an event-based approach [15] to transitioning. This way, states can transition while only needing to know the type of the target state and whether the transition should stack on top of or replace the current state.

5.2.2 Navigation in VR

Motivation

Navigation is motivated by the user's need to traverse a scene, viewing and interacting with it from many different positions and angles. Importantly, the chosen method must not distract users from the primary task of designing the scene. Editor sessions may extend to long periods of time, making it important to account for VR sickness when choosing a method of locomotion.



(a) Navigation

(b) Transforming an object

Figure 5.2: *Navigation* and *Transform* states

Description

To avoid distraction and reduce the likelihood of users feeling sick after long period, teleportation offers a very efficient and conceptually simple approach to travel in VR. This is accomplished by having the user aim with one of the motion controllers and pressing a button mapped to the teleportation action. This simple point-and-click method is easy to pick up, although it is prone to disrupting spatial awareness over longer travel distances. Any offset between the ground and the user's view is preserved during teleportation to avoid users ending up with their heads on the ground. Similarly, to avoid users telporting inside of objects or walls, the anchor position is set 90% of the way along the teleportation ray. Figure 5.2 illustrates this process.

Choices

- **Gaze vs point direction**

  At first, teleportation was controlled by the user's gaze direction. While this succeeded in freeing up the users hands more, it merely shifted the burden to the user's head. After some experimentation and feedback from users, it became clear that relieving the user's head at the cost of one hand was a worthwhile compromise.

5.2.3 Speech Recognition

Motivation

In order to minimise the gesture and button vocabulary a user has to learn to use the editor, speech input offers a complementary method of extending the user's capabilities [5, 7]. The degree to which it is complementary depends on how speech is used to control the editor, so it is important to determine what type of actions are well-suited for voice input.

Description

Speech input is implemented using the Windows speech API provided by Unity. It exposes a *KeywordRecogniser* that allows one to supply a list of keywords to listen for, triggering an event when one is recognised. These events are distributed further to states by abstracting the *KeywordRecogniser*. This results in static events for commands such as undo and redo, making it easy for states to listen as and when they need.

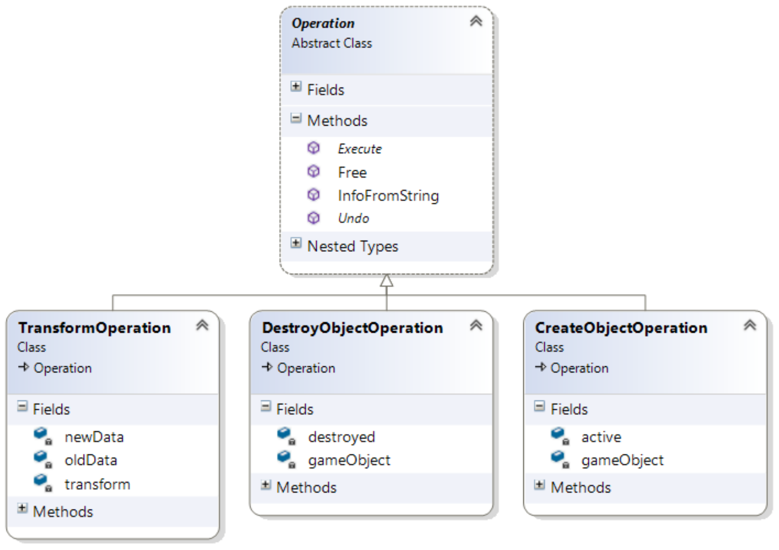


Figure 5.3: Undo/redo Operations

Speech input is inappropriate for some tasks, such as object manipulation. With this in mind, speech has only been sparingly applied to provide a quick and efficient interface for undoing or redoing an action. Supported actions are referred to as *Operations*, and these include those shown in figure 5.3. When an operation occurs, it is recorded in a circular buffer in case the user later wishes to undo it. Undo and redo are accomplished by the *Undo* and *Execute* methods respectively. The circular buffer, or *OperationMonitor*, stores a limited number of operations, overwriting old ones and throwing undone operations away when a brand new operation is added.

Choices

- **Singleton pattern**

  Originally, the class responsible for interacting with the *KeywordRecongiser* was implemented as a Singleton [19]. This ensured that there was only ever one instance and that this instance was globally accessible for easy distribution to other classes. After evaluation, this was noted as a bad decision: global access is rarely the answer to a problem, and the same end could be accomplished just by using static events anyway. In the end, the *SpeechManager* class was reverted to a normal class with some care being taken to make sure its resources are properly disposed of.

5.2.4 Procedural Geometry

Motivation

As the editor is intended for rapid prototyping, certain tasks should be performed automatically instead of leaving them up to the user. Manual construction of the scene's walls and floor is both laborious and non-trivial to implement, requiring users to interact with the underlying geometry in VR. Procedural generation of these walls offers savings in time and effort, while still allowing the users control of the scene layout.

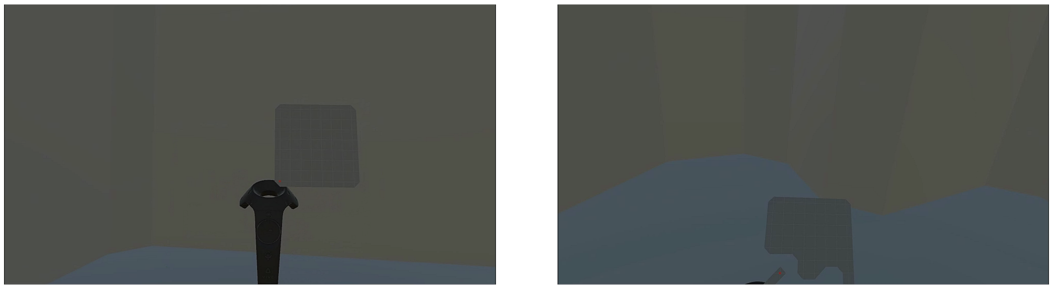

(a) Empty grid (b) Edited grid and geometry

Figure 5.4: Underlying grid supporting procedural wall generation

Description

To solve this problem, wall generation is first implemented using an extension of the marching squares algorithm, not unlike marching cubes [30]. Extending from Lague's implementation of marching squares [26], an underlying grid is used to spatially divide the scene floor (figure 5.4). Each cell is labeled on or off, indicating whether or not the cell is occupied by a wall. Polygons are generated by treating groups of four cells as the vertices of a square, where each of the sixteen different on/off combinations generates a distinct polygon. These polygons are finally combined to produce a floor; however, the editor advances on this, also generating wall polygons based on the combination of square vertices, saving time and memory compared to Lague's approach, which applies wall generation at a later stage. Finally, the grid is exposed to the user via a world-in-miniature-like interface, letting them decide which cells are occupied by walls.

Choices

- **Procedural**

  An alternative to using procedural generation would have been to allow the user to interact with the wall's vertices directly. In opting for procedural generation, scene walls are limited by the underlying grid resolution. Nevertheless, interactively producing complex shapes or walls with curvature is beyond the scope of this editor as a rapid prototyping tool. Procedural generation is more aligned with the project's goals for speed and effort, leaving more detailed approaches to the CAD tools discussed in section 2.1.2. Applications of marching cubes have been targeted at space planning and room design already [31].

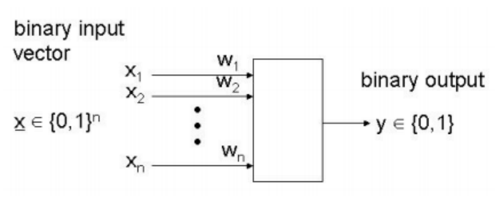
5.2.5 Gesture Detection

Motivation

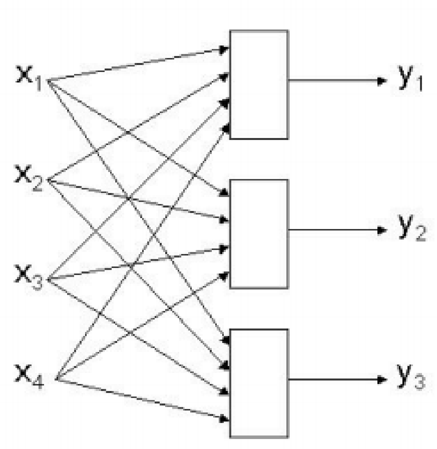
In addition to speech and button input, gesture provides another potential interface for the user. Gesturing in 3D boasts a theoretically infinite set of potential gestures; however, this is limited in reality by the physical capabilities of the user and the occupied space. More importantly, providing too rich a gesture vocabulary threatens to overwhelm users. This motivates the implementation of trainable, real-time gesture classifier [24], allowing users to create gestures and map them to actions in the editor as they see fit.

Description

The gesture classifier is implemented using the well-known Perceptron model [45], which extends from the McCullough and Pitts 1943 proposal of a neural model: the Binary Decision Neuron (BDN) [32].



(a) Binary Decision Neuron



(b) Perceptron: a net of BDNs

Figure 5.5: Neural Network Models

Figure 5.5 illustrates how a BDN computes a single, binary output from a weighted vector of inputs. It does this by summing these weighted inputs, outputting 1 if it exceeds some activation threshold, or 0 otherwise. A Perceptron may simply be viewed as a neural network of multiple neurons acting in the same way. In order to be of any use, the input weights must be adapted until the neural net successfully approximates the desired function. Very briefly, this is managed through a process called gradient descent, where the network is trained on pre-classified inputs. Since it knows the correct classification (output), the network can calculate its error over several successive iterations, adapting the weights to lower this error and converge on an optimal configuration. A Perceptron is only able to converge like this for linearly-separable problems [36].

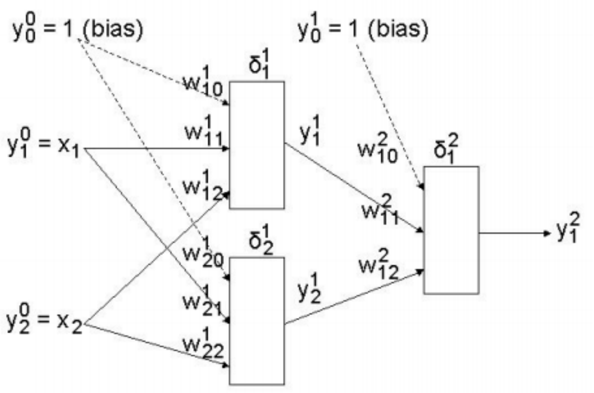


Figure 5.6: Multilayer Perceptron

To overcome the problem of linear-separability, we can add an additional hidden-layer, as in figure 5.6, turning it into the popular Multilayer Perceptron of today [46]. A similar process of error backpropagation is used to adapt weights backwards from the final output layer where the error is measured. Additionally, outputs are not binary, but continuous, outputting what can be thought of as the network's confidence when used to classify an input vector.

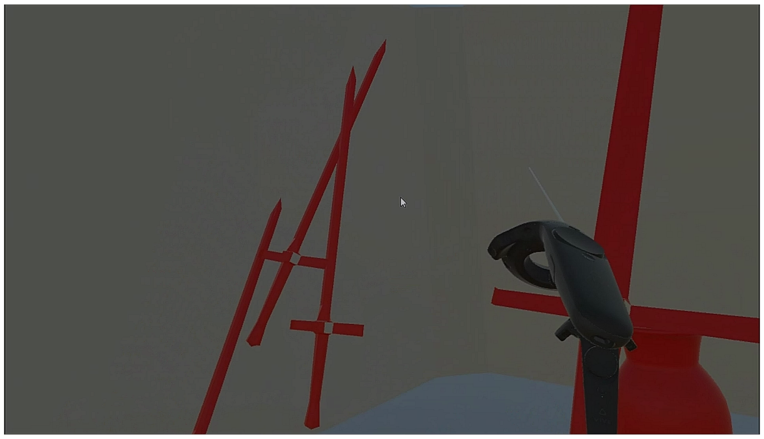


Figure 5.7: Duplicate object gesture

For detecting gestures, this model is directly applied using an input vector of 11 (x, y, z) coordinates. These coordinates are obtained by uniformly sampling the position of the user's hand while recording a gesture. After sampling enough positive and negative examples of a gesture, the classifier is trained using the backpropagation method described above, yielding a classifier that outputs a value between 0 and 1 to indicate its confidence that the input gesture is the same as the one it was trained on. A process of trial and error is necessary to decide the right configuration and training data needed to optimise the network; however, a universal approximation theorem states that only a single hidden layer is necessary [13]. Figure 5.7 illustrates how gesturing is used to duplicate an object held in the other hand.

Choices

- **Neural network approach**

  Gesture detection has been accomplished using many different approaches, including hidden Markov models [52, 28, 3], heuristics targeting specific gestures [58, 11], as well as artificial neural networks [37, 61]. While heuristics have been shown to be useful for rapid deployment and low cost [60, 25], the decision to use a neural network was borne out of desire for customisability. In order to classify a gesture, the network requires training; however, this allows for a range of gestures to be used and, in the future, the mapping from gesture to action can be left to the user's discretion. It is also worth noting that various types of artificial neural networks exist and that the Multilayer Perceptron was chosen as a proof of concept.

5.2.6 Object Recommendation

Motivation

In order to search for an object to use in VR, the user has to somehow discern and select this object from a possible list of others. In cases where they do not have an object in mind, they may have to browse this list seeking inspiration. This is expensive in terms of time and effort, particularly when the method of searching is taxing, as with keyboard input in VR. Object recommendation presents a possibility of predicting what object(s) the user may want to use next, potentially avoiding the search process entirely.

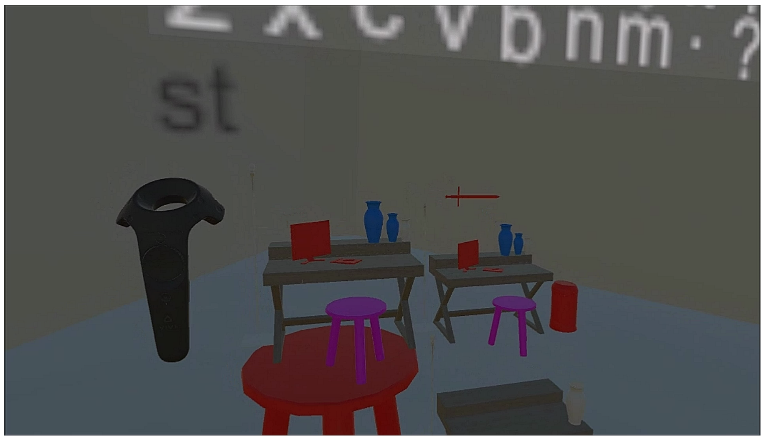


Figure 5.8: Trends within scenes contribute to object recommendation

Description

Object recommendation is achieved by indexing scenes at startup. By inspecting the objects used in each scene, the editor populates a database linking scenes to those objects. After an object is added to the scene, this database is queried to find other objects used alongside the last one added. The most popular are then provided as recommendations to the user. In practice, this is useful when you have a large number of scenes that share a trend in object relationships (figure 5.8). It avoids users having to repeat the effort of finding and selecting these objects for a new scene; however, it is limited by the number of scenes available for inspection and currently does not account for whether a recommendation is used or not, meaning recommendations tend to repeat if the same object is used many times. In the future, the recommendation engine may account for this, and it can also be extended to use information gathered from scenes that were shared online.

Choices

- **Semantics**
  Originally, object recommendation was based on semantic labeling of objects, where users would designate a model with qualities referring to their appearance, material and function. Similar objects will likely share similar labels, drawing a semantic connection usable for recommendations. Furthermore, object labels can be aggregated to describe the

containing scene, or, similarly, scene labels can be recursively applied to the comprising objects. While an interesting approach, labeling requires a significant amount of effort from the user, which contradicts the goals of an easy to use prototyping tool. Using existing scenes for recommendations allows users to avoid repeating themselves and can even be applied to aggregated user data to produce a statistical recommendation engine in the future.

5.2.7 Web-based Object Search

Motivation

A common problem faced by scene editors of all kinds is how to supply user content, referred to as objects throughout this project, for use within a scene. Matters are made worse when the user just wants to try the editor without necessarily having the requisite models or textures. This calls for an alternative option that allows users to start creating from scratch: web-based object search provides all users with access to a shared library of resources. In doing so, it greatly improves the editor's accessibility and ease of use, which in the end saves time and money during the prototyping stages.

Description

Web-based search is implemented using a web-application that allows users to upload their 3D models and scenes, sharing them with other people for use within their scenes. Uploaded files are stored as arbitrary blobs using Amazon S3, having first passed through the web-application, which records details about the model such as its name and retrieval location. Users can then search in exactly the same way as they would locally, prepending their search strings with a question mark (?) symbol. The editor's companion web-application exposes search via a RESTful API, returning formatted json results that fuzzily-match the search input. Fuzzy search is implemented for both local and web-based search using derivatives of the Lehvenstein distance metric [29], which can be expressed as the edit distance between strings. As the web-application functions separately from the editor and exposes a RESTful API, it offers a number of ways for research and other projects to utilise the crowd-sourced supply of data. Gamifying the sharing of objects is also a possible avenue of expansion that has shown promise in generating sources of data in fields such as natural language processing [39].

Choices

- **Search integration**

  When deciding how to integrate the web-application and editor, it was important that this interface be available for other projects to use as well. A RESTful API therefore made the most sense, as it relieves the web-application from having to manage state for potentially many different applications.

- **User identity**

  For simplicity and ease of use, users are not required to register or provide details about themselves to upload models. On the one hand, uploading is frictionless for anyone who has something they want to share; however, this does raise issues of ownership and severely limits the user's control after uploading an object. In the future, this balance will need to be re-addressed, but it was a necessary compromise given the scope of this project.

5.2.8 Automatic Object Placement

Motivation

VR offers a convenient and natural interface for when users want to directly handle objects themselves; however, it is important to accommodate situations where this is not the case. Sometimes, users care less about meticulous object placement, and more about quickly experimenting with a variety of objects in the scene. Automatic object placement offers a dramatic increase in placement speed, but at the cost of precision.

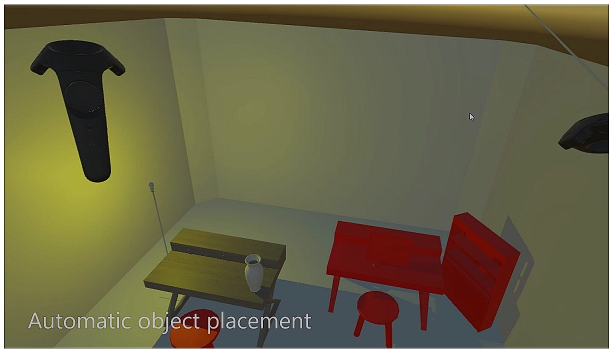


Figure 5.9: Automatically placed scenes

Description

Placement makes use of the underlying grid used by the procedural wall generation. It first takes the object, be it a model or a scene, and calculates a bounding box description in world space. This is then converted into grid-space by expressing the box in terms of grid cells. This is a fairly drastic simplification of object approximation and is largely the reason for automatic placement's imprecision. However, it has a major advantage in simplifying the task of avoiding collisions: we can now effectively steer the object's footprint, measured in grid cells, until it finds a space where it neither collides with the walls nor other objects. Steering behaviours have been widely applied to the movement of autonomous characters in games [43]. The same method is used here to steer away from spaces where collisions occur. Root level objects, those without parents, are treated independently for more flexible placement. Figure 5.9 demonstrates how two scenes were automatically placed in this way. Child objects have their relative positions preserved to prevent fragmentation of the imported scene.

Choices

- **Placement constraints**
  We can try to improve precision by applying constraints to the objects being placed: for example, a fireplace must be adjacent to a wall. Identifying focal points in a scene [62], before deriving spacial constraints between these points, might be one way to accomplish this. For this project, however, avoiding collision between objects was deemed more important as users would otherwise have to waste time correcting these mistakes, which defeats this feature's very purpose. A relatively simple extension of the current model would involve a more detailed representation of the object's shape — rather than using its bounding box for detecting collision within the grid.

Chapter 6: Architectural Views

The following architectural views offer a diagrammatic breakdown of the system. They were derived from the goals and requirements listed above and were used to guide the implementation, while undergoing revision when necessary.

6.1 Functional View

The functional view is represented with component diagrams that illustrate most of the components runtime interactions. The principles of DRY (Don't Repeat Yourself) programming and separation of concern have been applied to produce reusable, self-contained components that communicate via well-defined interfaces.

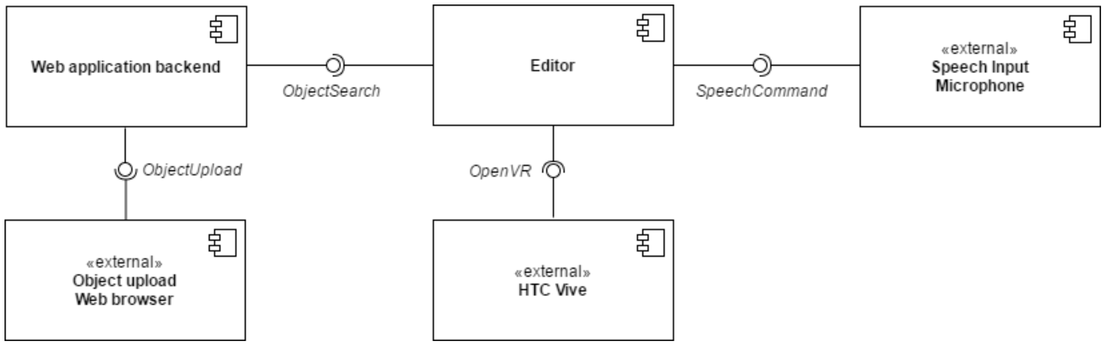


Figure 6.1: Overall system

Most of the functionality is contained within the *Editor* component; however, figure 6.1 demonstrates how we can treat the web application and editor as separate components whose interaction is clearly defined by the *ObjectSearch* interface

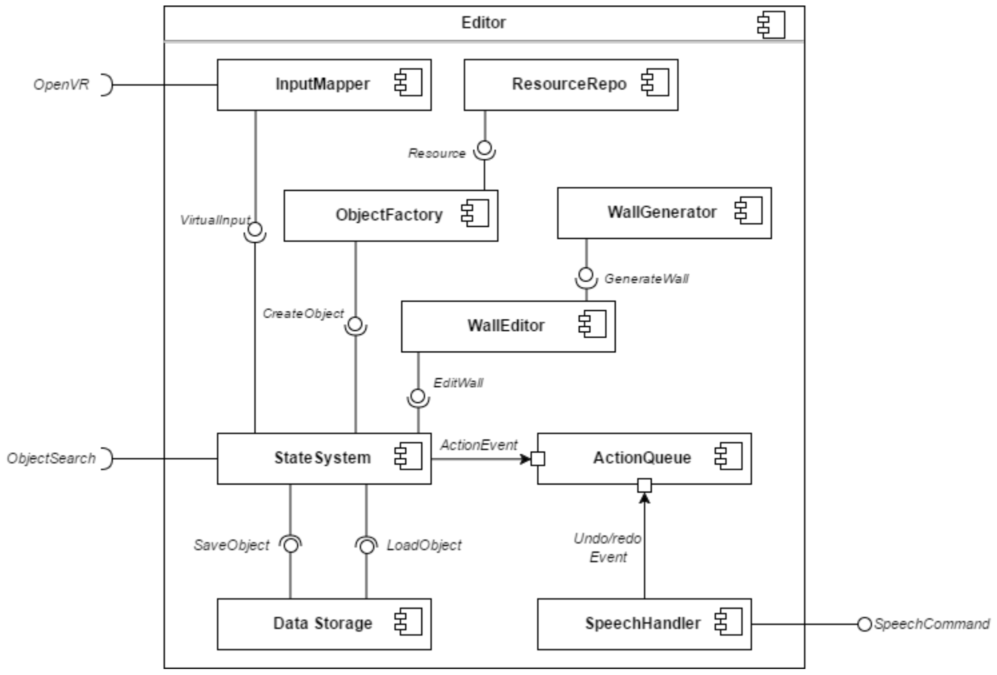


Figure 6.2: Editor component

The editor component, shown in figure 6.2, comprises two groups of components:

1. Those contained within the state system, which serves to logically separate implementation parts into distinct, non-overlapping states.
2. Those outside the state system, which mostly supply interfaces for use within the state system.

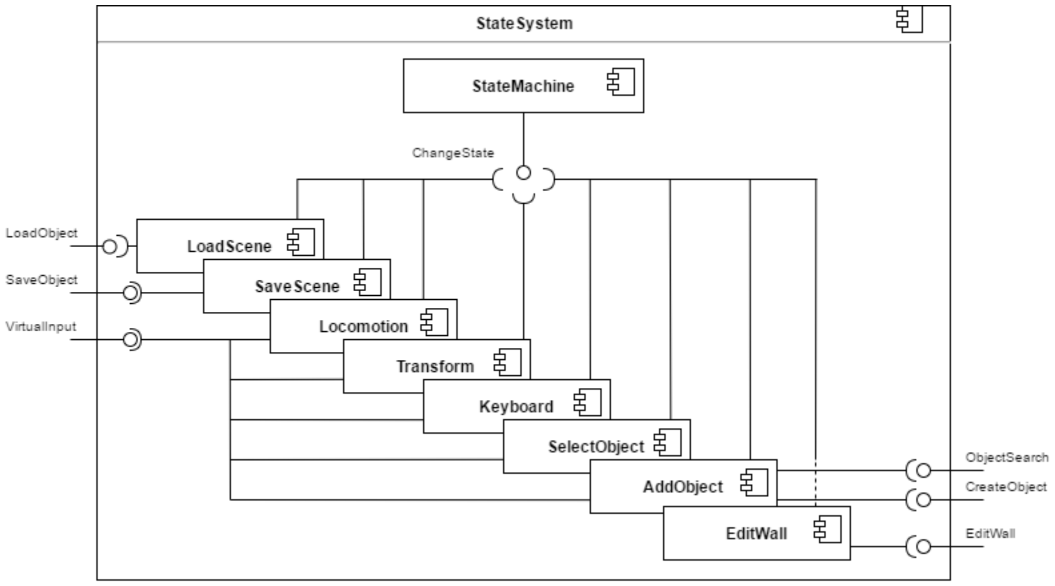


Figure 6.3: StateSystem component

Figure 6.3 illustrates the *StateSystem* component as a collection of states that each rely on the *StateMachine* for changing between one another. Taken out of the context, the diagram shows warning signs of over-reliance on interfaces such as *ChangeState* and *VirtualInput*. Normally, this would be to the detriment of cohesive data flow; however, since only one state can be active at any time, these signs are just an unfortunate consequence of the diagram and not reflected in reality.

6.2 Deployment View

The deployment view in figure 6.4 provides a concise but accurate look at how the system's components (artifacts) will be deployed. This includes the running instances (nodes) and their interaction protocols.

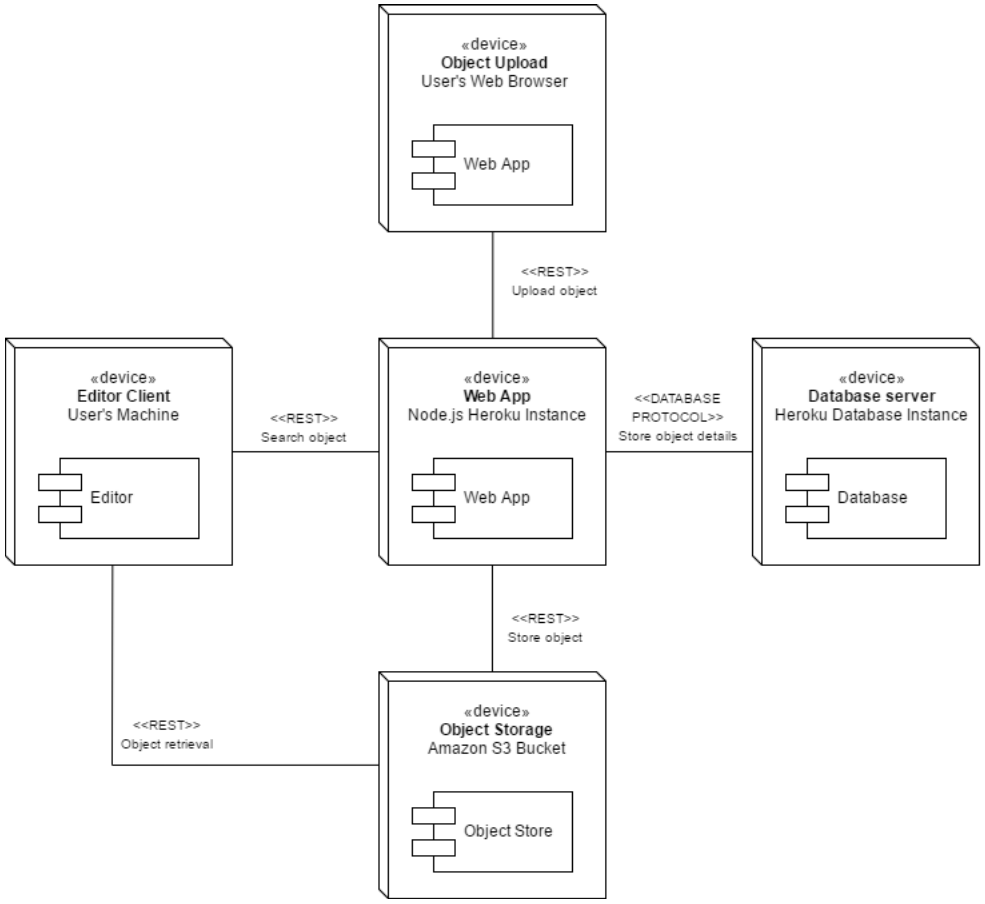


Figure 6.4: Deployment view

Editor Client

The editor client represents the core concern of this project. The user's machine runs the editor itself, which uses two REST APIs to access objects stored online:

1. The *Web App* API allows the editor to supply a search term (e.g. "chair"). It responds with details of online objects that are named similarly to this search term.

2. The *Object Storage* API lets the editor then retrieve objects using the details found by the *Web API* search.

Web App

The *Web App* facilitates search and upload of objects for client users. Upon object upload, it validates the file before using the *Object Storage* API provided by Amazon S3 to store the object online. Note that this storage is entirely separate from the database that records details of the stored objects, including names and retrieval URLs. These details are provided as search results to the editor.

Object Upload

The *Object Upload* component describes the client browser, which is responsible for uploading objects via the *Web App* API.

Database Server

The *Database Server* records details about stored objects, including filename and a URL that points to the object in question. The *Web App* acts as a proxy for this information when a user wishes to search for objects from the editor.

Object Storage

The *Object Storage* component describes the Amazon S3 instance responsible for actually storing the object file itself. It communicates with both the *Web App* and *Editor Client* for storing and retrieving objects respectively.

6.3 Scenarios

The following scenarios give more details about how object upload, search and retrieval take place.

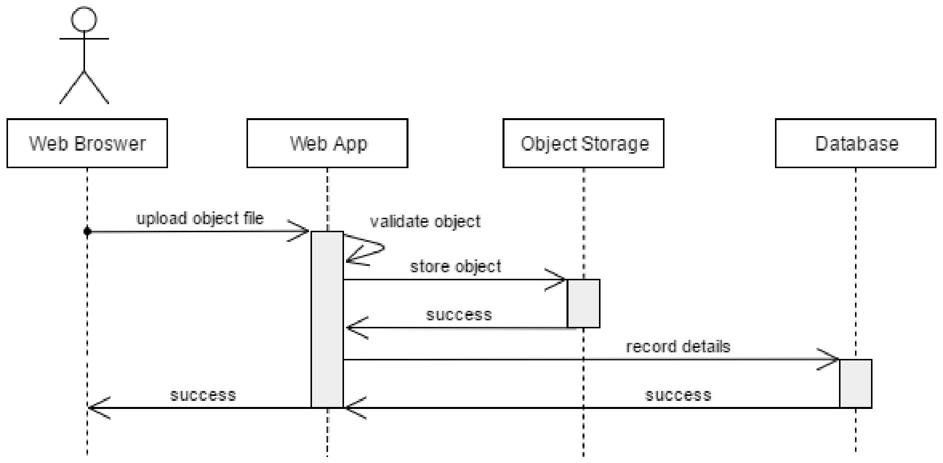


Figure 6.5: Object upload

Figure 6.5 demonstrates how a user submits a single object file for upload. The file's type is then validated, before it is sent to *Object Storage* for storing online. If the store request is successful, the details of the object are recorded in the *Database* to be used later when searching. If this too is successful, then the upload process is complete.

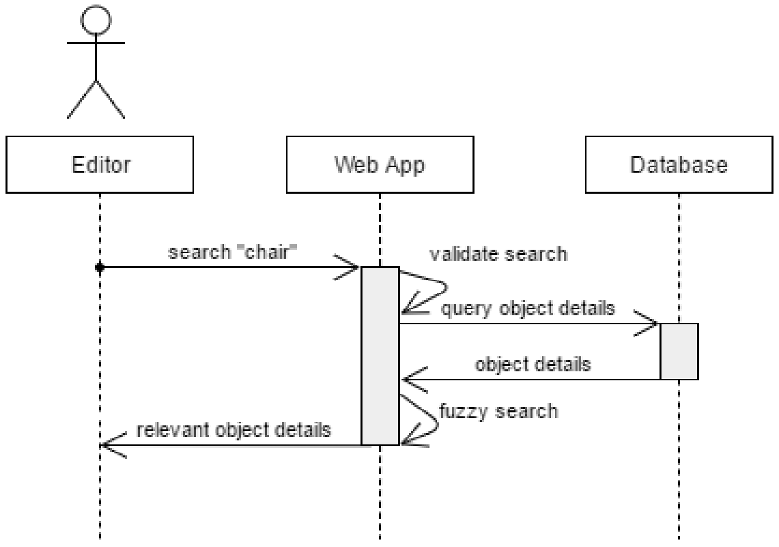


Figure 6.6: Object search

Figure 6.6 shows how a user can then search for objects, providing a search term that, after length and content validation, is used to perform a fuzzy search over all stored object details. Relevant objects (i.e. ones within a maximum distance from the search) are sent back to the editor.

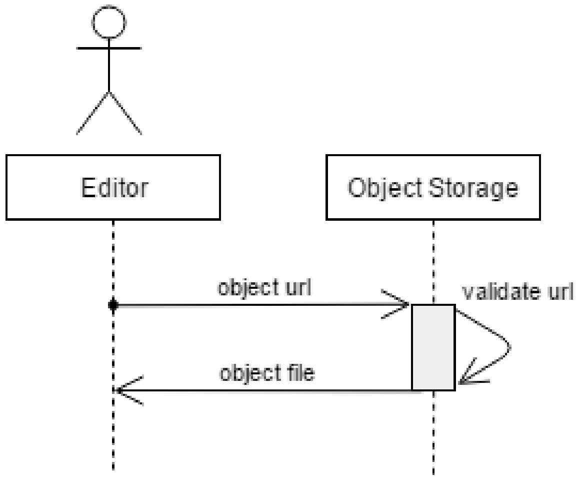


Figure 6.7: Object retrieval

Figure 6.7 briefly illustrates how a search result can be used. The object URL points to the stored object file in *Object Storage*. If it is valid, the object is returned for use within the editor.

Chapter 7: Testing

This chapter explains how testing was used to support development iterations throughout the project. Starting from an overall strategy description, the chapter then identifies key approaches to testing and how they were continuously applied from an early stage to monitor changes and extensions to the codebase.

7.1 Motivation

Testing offers numerous benefits in software engineering: tests can be used to make sure that breaking changes are identified quickly, in accordance with the well-established ”fail fast” principle [47]. This is particularly useful when working in large teams or on a large project where it is difficult to track and manage every part of the codebase. One could argue that this benefit is not fully realised in an individual project such as this one; however, tests can greatly benefit both maintenance and future work by providing a first line of verification when integrating new, or adjusting old, functionality.

7.2 Strategy

This project makes use of two different types of testing: unit and integration. Unit tests are applied at the method level, making independent assertions about their behaviour following calls with various parameters. These tests stem from the need to test code in isolation, attempting to guarantee the sanity of focal units in the codebase. However, they provide no evidence for the correctness of the overall system. Integration testing is applied at a higher-level, testing for successful component integration such that their combined functionality yields the expected result.

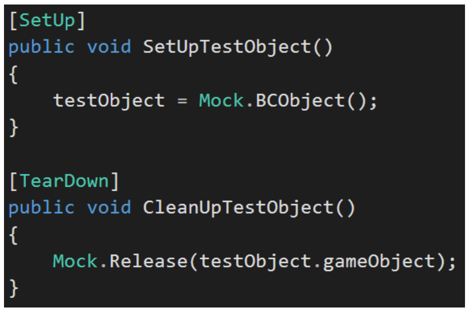
Some unit tests were also combined with simple fuzz testing, providing pseudo-random values as parameters in an attempt to discover hidden or unlikely errors. Integration tests have also been used to assert the correct state of a scene upon startup, providing a useful way to identity an incorrect setup or subtle problem that may otherwise have manifested as something else later on. This is particularly useful in the presence of less predictable peripherals such as the HMD and controllers.

7.3 Unit Testing

Unit tests have been used to identify breaking changes to the codebase, particularly resulting from unforeseen extensions. While they have been used to good effect (owing a lot to inversion of control making dependencies easier to mock) the overall code coverage is less than ideal due to time constraints. Nonetheless, a variety of tests have been constructed to support future completion of the test suite. Infrequently changing, and thus easily forgotten, parts of the codebase were prioritised for testing, as these fringe areas can often cost the most time to debug.

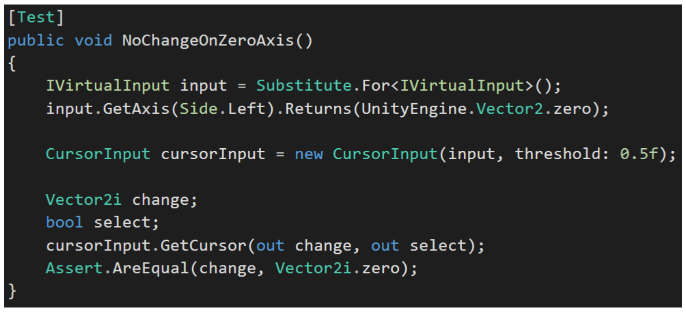
7.3.1 Dependency Management

Inversion of control promotes a style of dependency management that greatly supports testability, particularly when used alongside interface-based programming. Implementations can easily be switched out or mocked for test purposes, promoting good coverage of the class' internals.



```
[SetUp]
public void SetUpTestObject()
{
    testObject = Mock.BCObject();
}

[TearDown]
public void CleanUpTestObject()
{
    Mock.Release(testObject.gameObject);
}
```

(a) SetUp and TearDown attributes

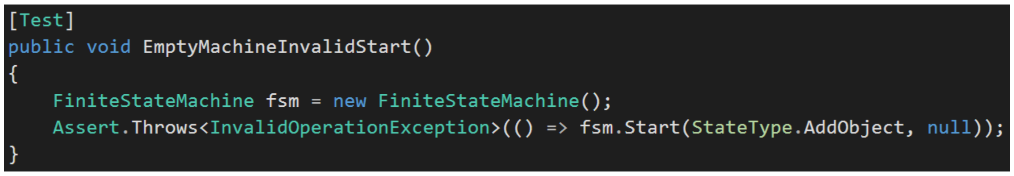


```
[Test]
public void NoChangeOnZeroAxis()
{
    IVirtualInput input = Substitute.For<IVirtualInput>();
    input.GetAxis(Side.Left).Returns(UnityEngine.Vector2.zero);

    CursorInput cursorInput = new CursorInput(input, threshold: 0.5f);

    Vector2i change;
    bool select;
    cursorInput.GetCursor(out change, out select);
    Assert.AreEqual(change, Vector2i.zero);
}
```

(b) Dependency substitution

Figure 7.1: Unit Test Dependency Management

Figure 7.1 demonstrates how inversion of control and interfaces are leveraged to substitute in test specific functionality. *IVirtualInput* is not the focus of the test on the right, making it suitable for mocking such that we are able to test *CursorInput* behaviour easily. In addition to inversion of control, some tests make use of special setup and tear-down attributes to designate behaviour that should execute before and after each test. This was an effective method of combining testing with Unity-specific resources that require proper releasing.



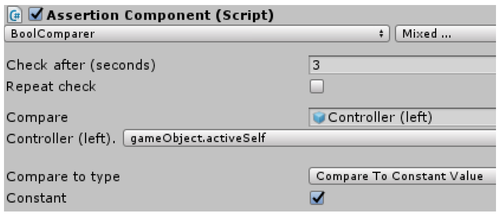
```
[Test]
public void EmptyMachineInvalidStart()
{
    FiniteStateMachine fsm = new FiniteStateMachine();
    Assert.Throws<InvalidOperationException>(() => fsm.Start(StateType.AddObject, null));
}
```

Figure 7.2: Throws Exception Test

Testing that exceptions are thrown as expected is also important, as it offers reassurance that defensive programming measures are working properly and as expected, even after changes. Figure 7.2 illustrates how the state system's expected exception conditions are tested.

7.4 Integration Testing

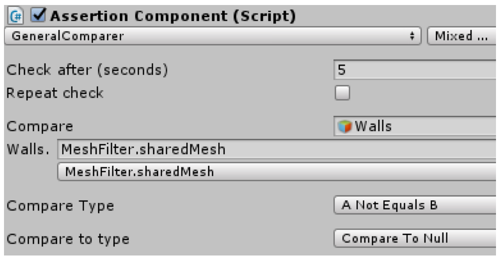
Integration tests monitor for broader failures that result from inter-component interaction. They are performed by running isolated test scenarios in Unity, checking the validity of components and the hierarchy at runtime using Unity's Assertion-Component.



Assertion Component (Script)
BoolComparer — Mixed ...

Check after (seconds)	3
Repeat check	☐
Compare	Controller (left)
Controller (left).	gameObject.activeSelf
Compare to type	Compare To Constant Value
Constant	✓

(a) Checking for Vive controller



Assertion Component (Script)
GeneralComparer — Mixed ...

Check after (seconds)	5
Repeat check	☐
Compare	Walls
Walls.	MeshFilter.sharedMesh
	MeshFilter.sharedMesh
Compare Type	A Not Equals B
Compare to type	Compare To Null

(b) Checking wall generation

Figure 7.3: Integration Tests

This assertion component integrates well with the Unity Editor and API, which is useful for testing scene-wide functionality that is tightly coupled with Unity constructs and not suitable for unit testing. Figure 7.3 demonstrates how after a fixed time period, assertions are made about the state of the scene, such as successful controller detection and wall generation. A similar approach is used to ensure that the game's entry object is active and ready at the start of a scene, as it is the source for nearly all object composition and nothing could follow otherwise.

As with this project's unit tests, the Unity Test Tools package was used to gain powerful features such as the Integration Test Runner environment, as well as the Assertion Components themselves.

7.5 Continuous Integration

Continuous integration [17] is a popular method of ensuring that a project gets the most out of its tests. By automatically building and testing either at regular intervals or in response to events, teams or individuals that integrate their work frequently are able to do so with confidence that problems are likely to be spotted early, provided they also have confidence in the test suite in place.

7.5.1 Git hooks

This project recognises the need to frequently build and test in an agile and iterative context. To do so, it uses version controlled, local git hooks to respond to git actions such as committing or pushing.

```
cwd = os.getcwd()
project = os.path.join(cwd, "Blue Comic")
results = os.path.join(cwd, "Test Results")
cmd = 'Unity -batchmode -projectPath "{}" -executeMethod UnityTest.Batch.RunIntegrationTests
subprocess.call(cmd, shell=True)
```

Figure 7.4: Git hook python script

Figure 7.4 illustrates a python script that triggers integration testing to run in batch mode. This particular hook is triggered just before every local commit, ensuring that any identifiable breaking changes never make their way into the repository's history. This is an alternative approach to how many popular and commercial CI providers handle events. As remote services, they cannot respond to local commits, instead relying on web hooks that listen for pushes to a remote repository. In this sense, local git hooks are more powerful and quick to act in the event of test failure. However, they suffer from having to be separately managed by every developer in the team, which is why they are version controlled before being moved into the necessary location within the repository. While this presents an inconvenience to large projects, it posed no problem to this one.

Chapter 8: Evaluation

This chapter is concerned with evaluating the successes and limitations of the project. This is done by revisiting the stated aims and requirements, critical discussion of the software product, as well as user-feedback-based evaluation of the editor overall.

8.1 Functional Evaluation

8.1.1 Goals

Recall this project's deliverable goals:

1. A desktop application for rapid and immersive scene prototyping in VR.
2. A web server that supplies 3D models to the desktop application.

Both these products have been delivered, with the web server exceeding its goal by providing a RESTful API that can easily be used in other projects interested in crowd-sourced 3D models and scenes. The editor aimed to offer a practical and immersive means of scene creation, focusing on the exploration of rapid prototyping features. The editor successfully delivers on this aim, showcasing numerous features geared towards a VR editing experience: including speech input, gesture detection, web-based search, object recommendation, modular scene inclusion and automatic object placement.

User Story	Description	Epic
1	Users can view the world in virtual reality.	1
2	Users can add/remove objects to/from levels.	2
3	Users can translate, rotate and scale objects.	2
4	Users can import custom 3D models for viewing.	2
5	Users can walk around the world.	2
8	Users can enable and customise collision and physics behaviour.	4
10	Users can save and load from a file.	5
11	Users can import levels into Unity as scenes.	5
12	Users can adjust material properties such as texture and colour.	6
13	Users receive recommendations based the objects they place in a scene.	2
14	Users can use speech to perform commands such as undo/redo.	1
15	Users can use search for models that they do not already have.	3

Table 8.1: User Stories Evaluation

Table 8.1 displays user stories, **not including those rejected at an early stage**. Green stories are implemented and functional within the editor; red stories indicate partial or no completion.

Note: this does not include additional prototyping features: gesture detection, procedural wall generation and automatic object placement. These were not identified as user stories early on but were nonetheless recognised as useful options for the editor as development unfolded. All three of these features are implemented and fully working as a testament to the project's iterative development process.

Looking only at the user stories above, the editor delivers an 83% user story completion rate. This is encouraging given the early stage at which these stories were created, and given the additional features and scope refinement that took place, the editor does well to satisfy its aims of delivering an immersive, rapid prototyping toolset.

8.1.2 Requirements

Treating this section as a type of contractual acceptance test, we examine the requirements defined in chapter 4 to see which of them are captured by the editor. This is best viewed as an early smoke-test carried out prior to more rigorous acceptance tests involving real users.

Functional Requirements

By inspection, 14 of the 16 functional requirements are satisfied by the editor and web-application deliverables, including all those deemed to be architecturally significant for having a large estimated influence on the product's implementation. R11 Achieve[Material adjustment] is not fully captured, as material changes are implemented as a proof of concept to demonstrate refactorisation of object groups. R10 was eventually deemed to be of low value and high cost, as it involved developing another separate tool that did not directly contribute to either the editor's immersive or rapid prototyping qualities. This reflects well on the editor's ability to fulfill its functional expectations, even greatly exceeding them in terms of the delivered feature set. Gesture detection, automatic object placement and procedural wall generation are also implemented and contribute a lot to the editor's rapid capabilities.

Quality Requirements

Performance requirements P1 and P2 are both satisfied under normal load. P1 is difficult to guarantee under all conditions, as the application's frame rate is subject to the environment it runs on. However, specific measures were taken to ensure that operations such as importing models from a file yield before execution falls below 30fps. P2 is implemented using a combination of caching and the Repository pattern, where resource access is mediated through a centralised caching and abstraction layer, known as the resource's repository. Availability requirements were defined speculatively with a production release in mind. Therefore, it is no surprise that A2 and A3 are not satisfied by the current MVP web-application prototype. A1 is satisfied, however, as the editor, web-app and object storage are all well isolated to avoid failure propagation. Security and Evolution requirements have all been satisfied by database query sanitation and documentation respectively. Between them, they promise a sustainable life for the editor.

8.2 Critical Evaluation

The following critical evaluation of the delivered products presents a review of the their strengths and weaknesses, as well as lessons learned that can be applied in the future. It also references some informal, formative feedback gathered after the implementation's completion.

8.2.1 Strengths

The editor's greatest strength is the variety of features it showcases for the benefit of rapid prototyping. While entire projects could be devoted to any one of them, this project sought to integrate and use them to form a cohesive editing experience. It successfully delivers on speech input, targeting it in such a way that it complements other features and avoids becoming a distraction or gimmick. Gesture detection uses a supervised learning approach to allow for customisable gesture recording and classification, adding an intuitive interface to the editor that users can freely customise for themselves. Additionally, object recommendation and automatic placement attempt to circumvent some of the worst parts of VR interaction, demonstrating the application's awareness of the user and creating opportunity for refinement in the future. Other features include procedural wall generation, object group refactorisation and web-based object search.

The editor's extensibility is also a strong proponent of its success: the state system represents a logical and efficient way to organise and implement editing behaviour. Future efforts could extend the editor's feature set without reference to anything except the state system documentation and an extension of the composition root (application entry point). This kind of extensibility is essential in large-scale or long-lived software applications, where the development team may change and functionality needs to be adapted.

Another strength lies in the web-application provision of a RESTful API that can be of assistance to a myriad of other projects interested in crowd-sourced models and scenes. In particular, Games With A Purpose (GWAP) demonstrate the promising application of crowd-sourcing methods to fields such as natural language processing [48, 57]. If a similar approach were taken for streaming content into editors, as with this project, it may reduce setup time and the cost of entry for content creators.

8.2.2 Limitations

A downside to including a wide range of significant features is the limited depth of their implementation. Simplifications and assumptions were necessary to deliver this many options, such as the restriction of speech input to just undo/redo behaviour. While this leaves some questions unanswered about how well a specifically targeted feature might integrate with VR rapid prototyping, it is also beneficial to act under constraints that force a reasoned and conservative application of a feature. Nevertheless, researchers interested in a thorough and exhaustive application of any of the mentioned features may be disappointed with the editor's focus on comparison and integration.

As another byproduct of the editor's scope, the overall user experience lacks a degree of polish that would be necessary for release. Despite being mappable, default controls involving the controllers' buttons can be confusing at times, and UI clues to do with selection and intent are sometimes absent. Not being able to teleport while holding or manipulating an object is another obvious point of contention. Fortunately, these problems can be mostly overcome with more development time and effort.

While code coverage is always an approximate measure of test completeness, it does not guarantee anything about the quality of testing. Nevertheless, this project's test coverage could be improved: detection of breaking changes can only happen when testing is appropriately sound and complete. Still, it is important to recognise the project's dual focus on research and software engineering. This led to the prioritisation of forming a cohesive testing strategy with examples rather than exhaustive coverage of the codebase.

8.2.3 Lessons Learned

The project's limitations largely stem from the imbalance between project feature set and resources available. This is a common problem in software development. In the future, it would be beneficial to perform a more comprehensive analysis of the cost-to-benefit ratio for a proposed feature. Furthermore, while iterative development saw to the timely delivery of a functional product, additional application of more recent lean startup practices may place greater emphasis on delivering a marketable product of value to users. Early and frequent user demonstration and feedback were sorely absent from this project and would have been invaluable in deciding which features held the most promise outright.

Early on in the project, some time was lost to a large, but highly successful, refactoring of the codebase. This removed many violations of inversion of control, replacing them with interface-based compositions of dependencies. This was a marked success; however, future projects would do well to adopt this approach from the very beginning to avoid a costly refactor later on.

8.2.4 User Feedback

Finally, formative feedback was gathered from a short, informal survey conducted over the course of five separate days following the editor's completion. A total of six users (friends, family and peers) tried the editor, having enough time to experience the core range of prototyping features available. Graphical results of these feedback sessions are presented in appendix E.

Specific feedback for wall editing tended to indicate that users struggled to understand the mapping from the hand-held grid to the actual scene. It was described as a nice feature to have, but one that was forgotten shortly after editing began. Speech input received a positive reception, owing to its ease of use and because it allowed users to change their minds while editing, using the undo/redo feature. Gesture control received mixed feedback as a few found it difficult to trigger gesture detection while others had much more success. Web search was acknowledged as a very useful feature, since none of the participants had easy access to 3D models without it. The downside was its integration with the keyboard, which users found tiresome to use repeatedly. Object recommendation and automatic placement were both described by the majority as easy to use but too simple in their current form.

These results indicate a lot of promise in the editor's features, particularly with speech input and web search.

Chapter 9: Future Work

This chapter briefly introduces possible avenues of future work that either utilise the editor for experimental or comparative studies, or involve further development of the editor itself. The RESTful API exposed by the web-application may be adopted by other projects as well.

9.1 User Studies

As the editor introduces a variety of different features for working in VR, it presents a use case for VR and HCI researchers interested in evaluating these features in greater depth, as well as those in need of a baseline measurement for comparison with their own approach.

9.1.1 Feature Evaluation

VR editors are often multi-modal and consequently call for research into preferred modes and their potential applications. An exploratory study can be applied to any, or even a combination, of the prototyping features presented by the editor, similarly to how Bolt does so for speech input [5]. Future work may also include evaluating the impacts on speed and effort, were a subset of the editor's features to be disabled. Although preliminary user feedback gathered at the end of this project indicates a preference for certain features, firm conclusions remain subject to more rigorous analysis in the future — involving a larger user sample that can account for factors such as the participant's level of expertise. A direct comparison of desktop editing to VR editing may also be carried out to ascertain whether or not the editor outperforms traditional approaches to rapid scene prototyping.

9.1.2 Research Baseline

Researchers of human computer interaction (HCI) may use the editor as a baseline for testing novel approaches to interaction in virtual or mixed reality systems. Discussion is already underway about a study interested in choosing the best paradigm for GUI presentation in immersive environments. The study will include a comparison of different modalities, such as:

1. Wrist GUI with hand selection.
2. Wrist GUI with head orientation selection.
3. Wrist GUI with eye selection.
4. Wrapping cylinder around the user, with swipe gesture selection.
5. Floating planes **(used in this project)**.

Possible results include a combination of objective and subjective evaluation, targeting user preference and speed/accuracy respectively. The study can also extend to mixed reality devices such as the Hololens for a wider spectrum of results.

9.1.3 RESTful API

The web-application's API allows anyone to upload and search for crowd-sourced 3D models and scenes. This is useful for both VR and non-VR research and applications, as the problem of sourcing content is ubiquitous across many fields.

9.2 Editor Expansion

The editor remains an MVP product and consequently offers a lot of opportunity for further development. Improvements to the user experience include reevaluating aspects of the control scheme and polishing the editor's visual appearance and UI elements. More importantly, its core set of features stands to be improved to offer greater utility than the current proof of concept allows.

9.2.1 Automatic Object Placement

Automatic object placement uses a dramatic approximation of the object's shape to avoid collision, and does not apply any constraints related to the object's rotation or scale. Further development may experiment with altering underlying grid's resolution for increased placement precision and density. Alternatively, it may separate automatic placement from the underlying grid entirely, using a continuous approach or machine learning to decide object positions and constraints.

9.2.2 Speech Input

Speech input may be extended to more than just undo/redo functionality, perhaps being integrated as a substitute for keyboard input. Input dictation was an early possibility for this project, but was omitted in the interest of time. A simple extension of the current system could make use of the same API's *DictationRecogniser* for detecting arbitrary speech input, not just keywords. This could then underpin a user-study's comparison of keyboard and speech input methods.

9.2.3 Gesture Detection

More advanced machine learning methods may be applied to gesture detection, including the use of concurrent neural networks that have often been applied to spatial problems such as this. Gestures are limited to object duplication at the moment, but this could be easily extended to other actions such as object addition, removal, grouping, parenting or repositioning.

9.2.4 RESTful API and Recommendation

Although not part of the editor itself, the API that supports web-based search can be expanded to include other data sources or more advanced functionality. Semantic labeling has been discussed above, and could be used to support object recommendation for getting started faster: users could simply choose a scene related to the label "office", using it to bootstrap their own.

9.2.5 Collaboration

A more ambitious expansion of the editor would be to include networked collaboration. This would add great value to the editor, further increasing its viability for application to industries such as game development. It introduces a new set of problems involving latency, ownership and performance. This is likely to involve a more significant overhaul of the editor, but is not beyond the realm of possibility.

Chapter 10: Conclusions

Scene prototyping presents a costly challenge to content creators, both VR and non-VR alike. The problem is exacerbated when designers are forced to frequently switch between conventional and immersive paradigms, discouraging iteration and efficient use of time. This project introduced an immersive editor aimed at solving this problem through the exploration and integration of VR-optimised features. It allows users to modularly compose scenes and share them with others via a separate web-application, which is integrated with the editor using a RESTful API. Users never have to leave the editor to prototype a scene, with their workflow supported by a novel combination of speech input, gesture detection, object recommendation, automatic placement and web-based object search.

The project's implementation was conducted iteratively as described, with evaluation demonstrating that the editor has achieved its stated aims. The editor and web-application satisfy nearly all of the requirements outlined in this report, exceeding them in terms of features delivered. However, the product remains, in parts, a proof of concept, and future work would do well to focus on improving the user experience and sophistication of core features such as gesture detection and automatic placement. The presented formative user feedback can be tentatively used to direct the course of further development, focusing on features deemed by users to be most helpful. The editor has implications for level design in industry, as well as VR and HCI researchers interested in a feature comparison or conducting user studies. Additionally, other projects may utilise the documented RESTful API for accessing crowd-sourced 3D models and scenes.

Bibliography

[1] Aily. Gridxz shader [source code]. `https://forum.unity3d.com/threads/wireframe-grid-shader.60071/`. Accessed: 2017-01-15.

[2] David B Anderson, John W Barrus, John H Howard, Charles Rich, Chia Shen, and Richard C Waters. Building multiuser interactive multimedia environments at merl. *IEEE MultiMedia*, 2(4):77–82, 1995.

[3] Derek Anderson, Craig Bailey, and Marjorie Skubic. Hidden markov model symbol recognition for sketch-based interfaces. In *AAAI fall symposium*, pages 15–21, 2004.

[4] Richard A Bartle. *Designing virtual worlds*. New Riders, 2004.

[5] Richard A Bolt. *"Put-that-there": Voice and gesture at the graphics interface*, volume 14. ACM, 1980.

[6] Doug A Bowman, David Koller, and Larry F Hodges. Travel in immersive virtual environments: An evaluation of viewpoint motion control techniques. In *Virtual Reality Annual International Symposium, 1997., IEEE 1997*, pages 45–52. IEEE, 1997.

[7] Doug A Bowman, Ernst Kruijff, Joseph J LaViola Jr, and Ivan Poupyrev. An introduction to 3-d user interface design. *Presence: Teleoperators and virtual environments*, 10(1):96–108, 2001.

[8] Jeff Butterworth, Andrew Davidson, Stephen Hench, and Marc T Olano. 3dm: A three dimensional modeler using a head-mounted display. In *Proceedings of the 1992 symposium on Interactive 3D graphics*, pages 135–138. ACM, 1992.

[9] Heng Chen. Three-dimensional modeling technology in virtual reality. In *International Conference on Information Computing and Applications*, pages 174–183. Springer, 2013.

[10] Jessie YC Chen and Jennifer E Thropp. Review of low frame rate effects on human performance. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 37(6):1063–1076, 2007.

[11] Mi Gyung Cho. A new gesture recognition algorithm and segmentation method of korean scripts for gesture-allowed ink editor. *Information Sciences*, 176(9):1290–1303, 2006.

[12] Valve Corporation. Steamvr [source code]. `https://www.assetstore.unity3d.com/en/#!/content/32647`. Accessed: 2016-10-10.

[13] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, 1989.

[14] Krista A Ehinger, Antonio Torralba, and Aude Oliva. A taxonomy of visual scenes: Typicality ratings and hierarchical classification. *Journal of Vision*, 10(7):1237–1237, 2010.

[15] Ted Faison. *Event-Based Programming*. Springer, 2006.

[16] Martin Fowler. Inversion of control containers and the dependency injection pattern. 2004.

[17] Martin Fowler and Matthew Foemmel. Continuous integration. *Thought-Works) http://www. thoughtworks. com/Continuous Integration. pdf*, page 122, 2006.

[18] Eric Foxlin et al. Motion tracking requirements and technologies. *Handbook of virtual environment technology*, 8:163–210, 2002.

[19] Erich Gamma. *Design patterns: elements of reusable object-oriented software.* Pearson Education India, 1995.

[20] William G Halfond, Jeremy Viegas, and Alessandro Orso. A classification of sql-injection attacks and countermeasures. In *Proceedings of the IEEE International Symposium on Secure Software Engineering*, volume 1, pages 13–15. IEEE, 2006.

[21] Istepaniuk. Stringdistance [source code]. `https://github.com/istepaniuk/StringDistance/`. Accessed: 2017-12-03.

[22] Jason Jerald, Paul Mlyniec, Arun Yoganandan, Amir Rubin, Dan Paullus, and Simon Solotko. Makevr: A 3d world-building interface. In *3D User Interfaces (3DUI), 2013 IEEE Symposium on*, pages 197–198. IEEE, 2013.

[23] HY Kan, Vincent G Duffy, and Chuan-Jun Su. An internet virtual reality collaborative environment for effective product design. *Computers in Industry*, 45(2):197–213, 2001.

[24] Juha Kela, Panu Korpipää, Jani Mäntyjärvi, Sanna Kallio, Giuseppe Savino, Luca Jozzo, and Sergio Di Marca. Accelerometer-based gesture control for a design environment. *Personal and Ubiquitous Computing*, 10(5):285–299, 2006.

[25] Sven Kratz and Michael Rohs. A $3 gesture recognizer: simple gesture recognition for devices equipped with 3d acceleration sensors. In *Proceedings of the 15th international conference on Intelligent user interfaces*, pages 341–344. ACM, 2010.

[26] Sebastian Lague. Unity procedural cave generation tutorial. `https://unity3d.com/learn/tutorials/projects/procedural-cave-generation-tutorial`. Accessed: 2016-10-16.

[27] Craig Larman. *Agile and iterative development: a manager's guide.* Addison-Wesley Professional, 2004.

[28] Hyeon-Kyu Lee and Jin-Hyung Kim. An hmm-based threshold model approach for gesture recognition. *IEEE Transactions on pattern analysis and machine intelligence*, 21(10):961–973, 1999.

[29] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.

[30] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *ACM siggraph computer graphics*, volume 21, pages 163–169. ACM, 1987.

[31] Carsten Maple. Geometric design and space planning using the marching squares and marching cube algorithms. In *Geometric Modeling and Graphics, 2003. Proceedings. 2003 International Conference on*, pages 90–95. IEEE, 2003.

[32] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[33] Mervill. Litjson [source code]. `https://github.com/Mervill/UnityLitJson/tree/master/Source`. Accessed: 2016-10-01.

[34] Josephine Micallef. Encapsulation, reusability and extensibility in object-oriented programming languages. 1987.

[35] Mark Mine. Isaac: A virtual environment tool for the interactive construction of virtual worlds. *UNC Chapel Hill Computer Science Technical Report TR95-020*, 1995.

[36] Marvin Minsky and Seymour Papert. Perceptrons. 1969.

[37] Kouichi Murakami and Hitomi Taguchi. Gesture recognition using recurrent neural networks. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 237–242. ACM, 1991.

[38] Bashar Nuseibeh. Weaving together requirements and architectures. *Computer*, 34(3):115–119, 2001.

[39] Iuliana-Elena Parasca, Andreas Lukas Rauter, Jack Roper, Aleksandar Rusinov, and Guillaume Bouchard Sebastian Riedel Pontus Stenetorp. Defining words with words: Beyond the distributional hypothesis. *ACL 2016*, page 122, 2016.

[40] Mark Pauly, Niloy J Mitra, Johannes Wallner, Helmut Pottmann, and Leonidas J Guibas. Discovering structural regularity in 3d geometry. In *ACM transactions on graphics (TOG)*, volume 27, page 43. ACM, 2008.

[41] R Pausch, T Burnette, AC Capehart, M Conway, D Cosgrove, R DeLine, J Durbin, R Gossweiler, S Koga, and J White. A brief architectural overview of alice, a rapid prototyping system for virtual environments. *IEEE Computer Graphics and Applications*, 1995.

[42] P Randy. A brief architectural overview of alice, a rapid prototyping system for virtual reality. *IEEE Computer Graphics and Applications*, 1995.

[43] Craig W Reynolds. Steering behaviors for autonomous characters. In *Game developers conference*, volume 1999, pages 763–782, 1999.

[44] David Roberts. Principles of collaborative virtual environments. Technical report, technical report, University of Salford Manchester, http://info. nicve. salford. ac. uk/web/files/uplink/Roberts_IIS2003. pdf, 2006-02-16.

[45] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[46] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[47] Jim Shore. Fail fast [software debugging]. *IEEE Software*, 21(5):21–25, 2004.

[48] Katharina Siorpaes and Martin Hepp. Games with a purpose for the semantic web. *IEEE Intelligent Systems*, 23(3), 2008.

[49] Mel Slater. Place illusion and plausibility can lead to realistic behaviour in immersive virtual environments. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 364(1535):3549–3557, 2009.

[50] Mel Slater, Anthony Steed, and Yiorgos Chrysanthou. *Computer graphics and virtual environments: from realism to real-time.* Pearson Education, 2002.

[51] Mel Slater, Martin Usoh, and Anthony Steed. Taking steps: the influence of a walking technique on presence in virtual reality. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 2(3):201–219, 1995.

[52] Thad Starner and Alex Pentland. Real-time american sign language recognition from video using hidden markov models. In *Motion-Based Recognition*, pages 227–243. Springer, 1997.

[53] Anthony Steed and Doug A Bowman. Displays and interaction for virtual travel. In *Human Walking in Virtual Environments*, pages 147–175. Springer, 2013.

[54] William Steptoe, Simon Julier, and Anthony Steed. Presence and discernability in conventional and non-photorealistic immersive augmented reality. In *Mixed and Augmented Reality (ISMAR), 2014 IEEE International Symposium on*, pages 213–218. IEEE, 2014.

[55] Jonathan Steuer. Defining virtual reality: Dimensions determining telepresence. *Journal of communication*, 42(4):73–93, 1992.

[56] Richard Stoakley, Matthew J Conway, and Randy Pausch. Virtual reality on a wim: interactive worlds in miniature. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 265–272. ACM Press/Addison-Wesley Publishing Co., 1995.

[57] Luis Von Ahn and Laura Dabbish. Designing games with a purpose. *Communications of the ACM*, 51(8):58–67, 2008.

[58] Andrew Wilson and Steven Shafer. Xwand: Ui for intelligent spaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 545–552. ACM, 2003.

[59] Chadwick A Wingrave, Yonca Haciahmetoglu, and Doug A Bowman. Overcoming world in miniature limitations by a scaled and scrolling wim. In *3D User Interfaces, 2006. 3DUI 2006. IEEE Symposium on*, pages 11–16. IEEE, 2006.

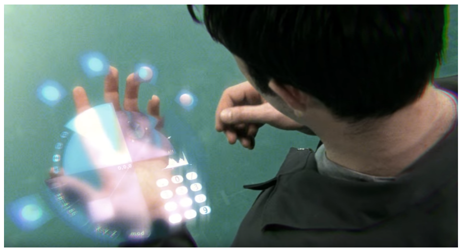
[60] Jacob O Wobbrock, Andrew D Wilson, and Yang Li. Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*, pages 159–168. ACM, 2007.

[61] Deyou Xu. A neural network approach for hand gesture recognition in virtual reality driving training system of spg. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, volume 3, pages 519–522. IEEE, 2006.

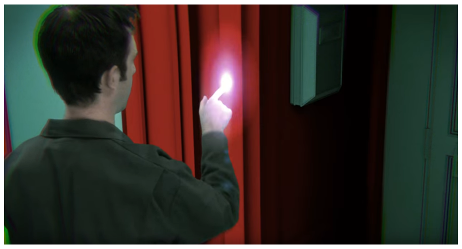
[62] Kai Xu, Rui Ma, Hao Zhang, Chenyang Zhu, Ariel Shamir, Daniel Cohen-Or, and Hui Huang. Organizing heterogeneous scene collections through contextual focal points. *ACM Transactions on Graphics (TOG)*, 33(4):35, 2014.

Appendix A: World Builder

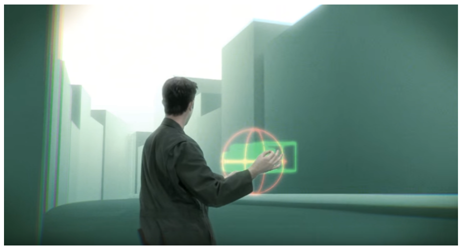
(a) Arm-mounted UI



(b) Interactive menu



(c) Altering material

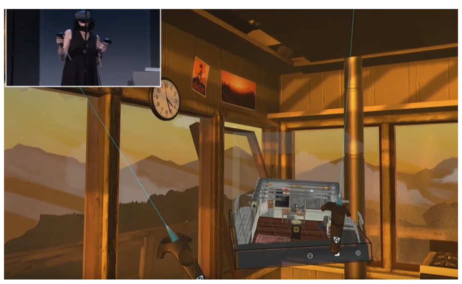


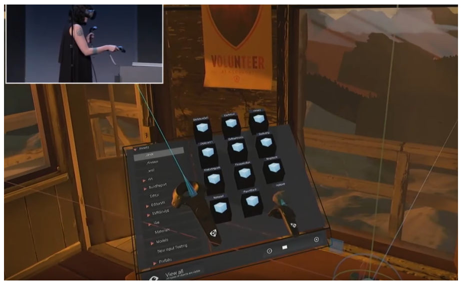
(d) Manipulating objects

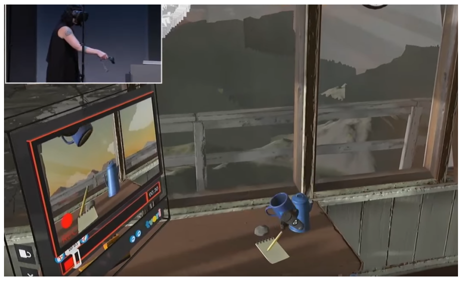
Figure A.1: World Builder

The protagonist exists in a highly-immersive virtual environment, where he uses a combination of gesture, touch and view-directed methods for interacting with the world.

Appendix B: Unity Editor VR


(a) Unity's 'chessboard' (WIM)


(b) Menu interaction with object preview


(c) Animation recording

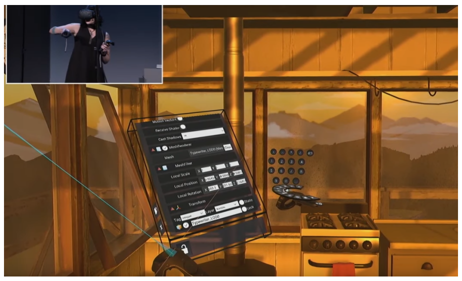
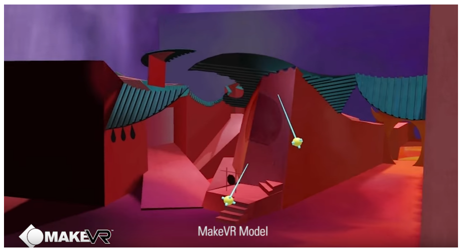
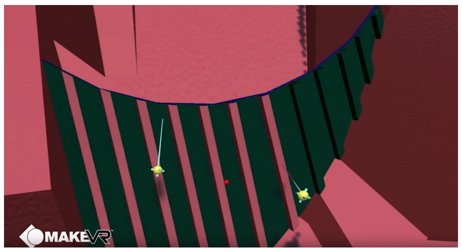

(d) Keyboard input

Figure B.1: Unity's Editor VR

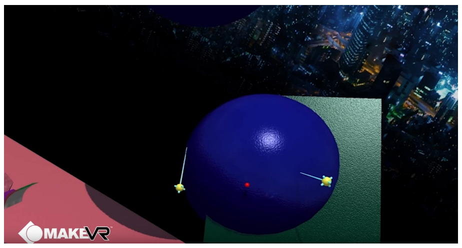
Appendix C: MakeVR



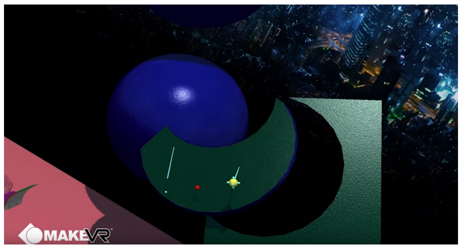
(a) Arranging a scene



(b) Altering object colour



(c) Geometric intersection 1



(d) Geometric intersection 2

Figure C.1: MakeVR CAD Engine

Appendix D: Stakeholder Onion Model

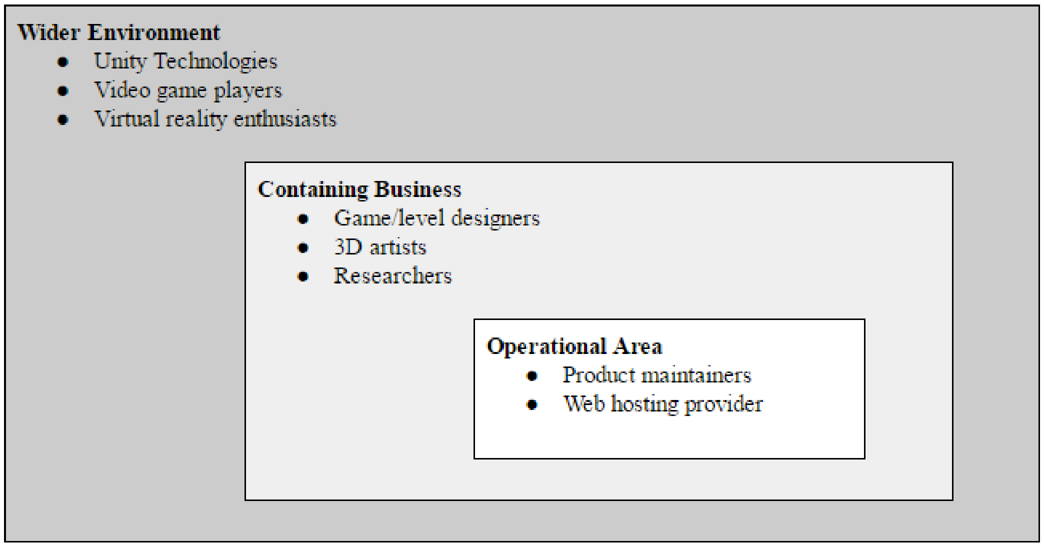
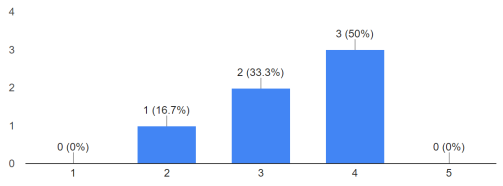
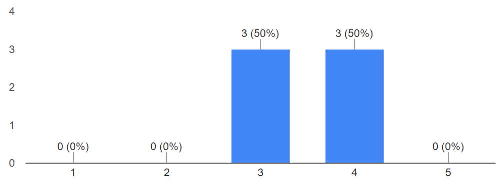


Figure D.1: Stakeholder Onion Model

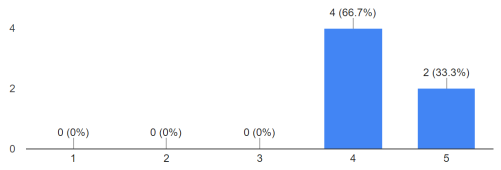
Appendix E: User Feedback



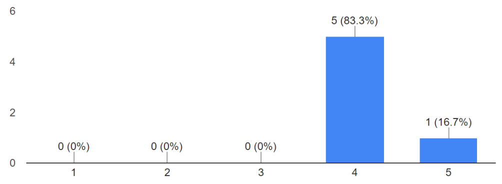
(a) How easy was wall generation to use?



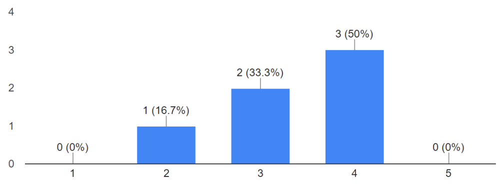
(b) How useful was wall generation for prototyping?



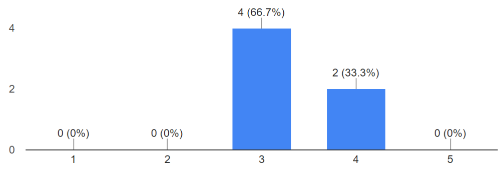
(c) How easy was speech input to use?



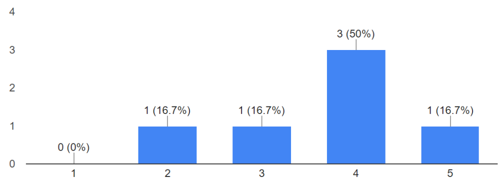
(d) How useful was speech input for prototyping?



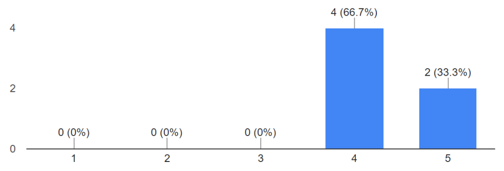
(e) How easy was gesture control to use?



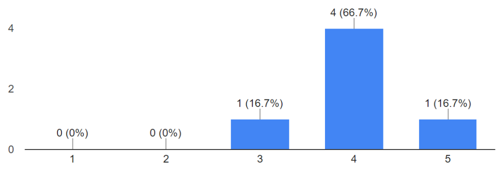
(f) How useful was gesture control for prototyping?



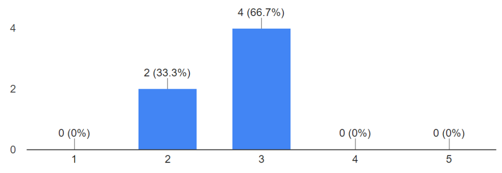
(g) How easy was web search to use?



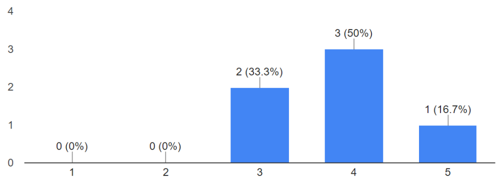
(h) How useful was web search for prototyping?



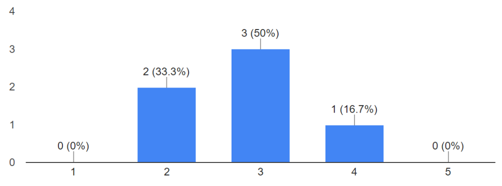
(i) How easy was object recommendation to use?



(j) How useful was object recommendation for prototyping?



(k) How easy was automatic placement to use?



(l) How useful was automatic placement for prototyping?

Appendix F: Video Demo

Figure F.1: Video demo: `https://youtu.be/4y6hHjxuHVI`

Appendix G: System Manual

G.1 Where is the project located?

The codebase is hosted as an online BitBucket git repository. This is accessible using via:

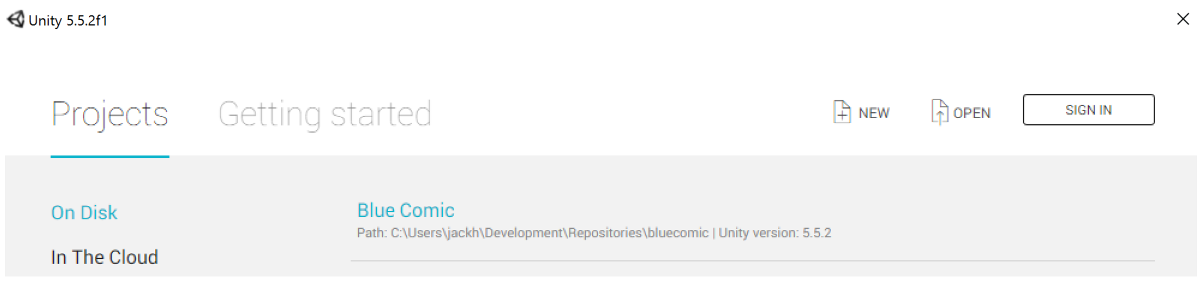
`https://bitbucket.org/zcabjro/bluecomic/`

G.2 Setup instructions

Firstly, you will need to clone the repository using either the command line or equivalent GUI application. On windows (with git installed: `https://git-scm.com/`), this can be done with:

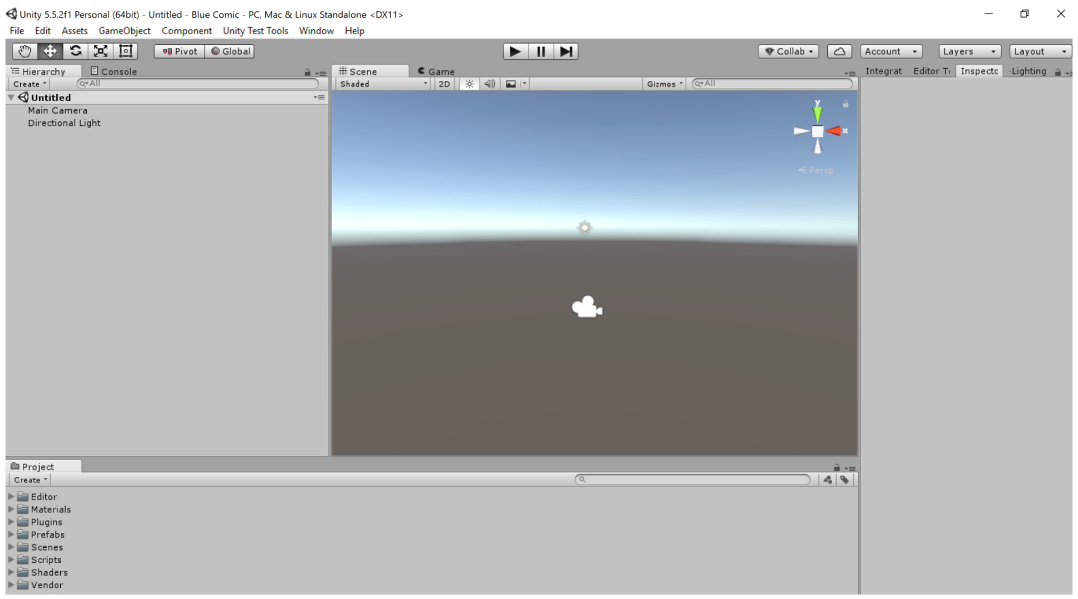
git clone `https://username@bitbucket.org/zcabjro/bluecomic.git`

You can now open the project in Unity. Version 5.5.2 was primarily used for this project; however, Unity updates are usually quite stable, only requiring an added re-import step if a newer version is used.

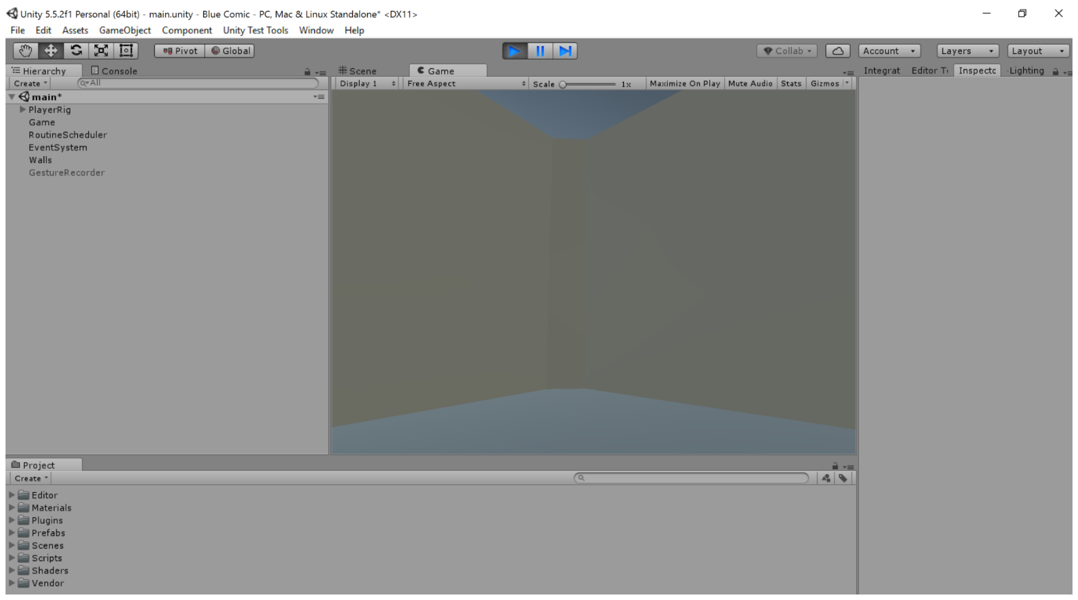


Path: C:\Users\jackh\Development\Repositories\bluecomic | Unity version: 5.5.2

You will need to select Open and navigate to *.../bluecomic/Blue Comic*


Unity 5.5.2f1 Personal (64bit) - Untitled - Blue Comic - PC, Mac & Linux Standalone <DX11>
File Edit Assets GameObject Component Unity Test Tools Window Help
Hierarchy
Console
Scene
Game
Untitled
Main Camera
Directional Light
Project
Editor
Materials
Plugins
Prefabs
Scenes
Scripts
Shaders
Vendor

This may open a new scene by default. This project comprises two scenes: *main* and *test*, where the latter is used to run scene-wide integration tests. Open *main* by selecting *File/Open Scene* and navigating to *Scenes/main.unity*.


Unity 5.5.2f1 Personal (64bit) - main.unity - Blue Comic - PC, Mac & Linux Standalone* <DX11>
File Edit Assets GameObject Component Unity Test Tools Window Help
Hierarchy
Console
Scene
Game
main*
PlayerRig
Game
RoutineScheduler
EventSystem
Walls
GestureRecorder
Project
Editor
Materials
Plugins
Prefabs
Scenes
Scripts
Shaders
Vendor

Simply pressing the *Play* button will run the scene inside Unity, triggering the start of the application and allowing you to preview/debug the scene.

IMPORTANT

You must have a HTC Vive set up and detectable by Unity in order for the editor to work properly. Although camera look is supported for the most basic of emulation, it was not feasible to develop a full VR emulation system for this project. Please consult the Vive's setup instructions for installing the necessary software, including SteamVR, as well as correctly positioning the Vive's Lighthouse trackers (`https://www.vive.com/uk/setup/`).

G.3 Project directory structure

Project resources are logically separated into directories with appropriate names.

Editor

Directories named "Editor" are special Unity directories whose contents are isolated within a separate, editor-specific assembly. This is for the benefit of writing editor extensions, as well as providing an ideal location for code that you don't want included within a build.

Materials

Unity materials are a composition of shaders and shader properties that determine how an object's visible surface is rendered. This directory contains all materials used by this project. Note though, that many materials are generated at runtime, using the MaterialRepository class to store them.

Plugins

"Plugins" is another special Unity directory that contains DLLs (dynamically linked libraries). This project relies on an SQLite database for indexing scenes at startup. The necessary DLLs are contained here.

Prefabs

A prefab is a Unity GameObject that has been serialised for repeated instancing. It is a convenient method of cloning objects for use within a scene. This project uses prefabs either for storing fundamental objects in the scene, or as customisable bases used during object creation. The BaseObject prefab is used for as the foundation for all BlueComic objects.

Scenes

Directory for storing Unity scene files.

Scripts

This is where all of this project's own code is stored. See the below documentation for details.

Shaders

This directory contains the custom shaders used by this project, including a surface shader for displaying a checkered grid, and another that supports GPU instancing for performance.

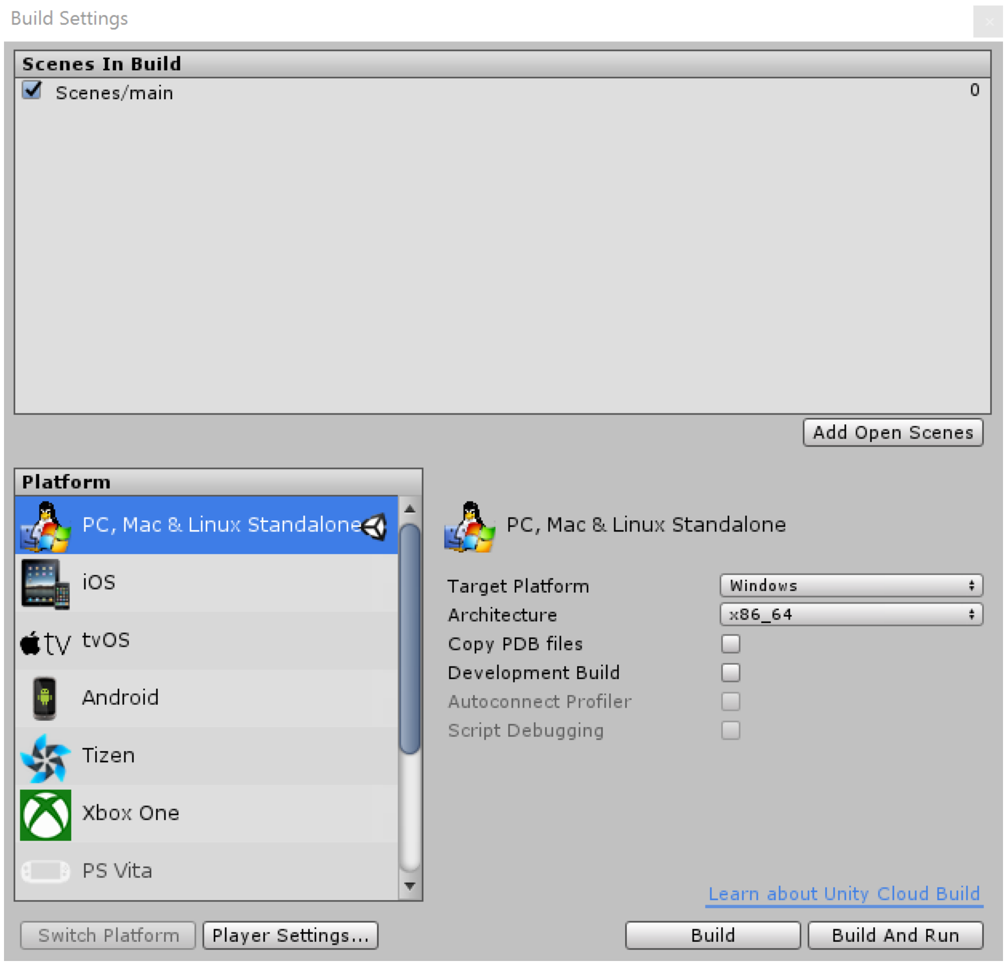
Vendor

This directory contains third-party source code, including:

- **LitJSON** (a JSON parsing library)
- **SteamVR** (supports the SteamVR features such as chaperone and input)
- **UnityTestTools** (necessary for testing in Unity)

G.4 Building the project

To build the project, select *File/Build Settings*



Build Settings

Scenes In Build

☑ Scenes/main 0

Target Platform: Windows

Architecture: x86_64

Copy PDB files: ☐

Development Build: ☐

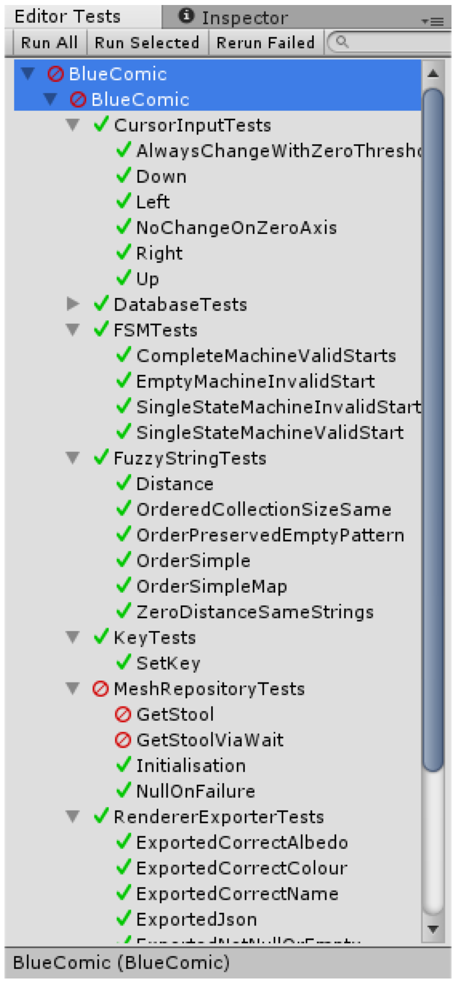
Autoconnect Profiler: ☐

Script Debugging: ☐

The project uses Windows specific Unity features so ensure that Standalone is selected as the platform on the left, and that Windows is chosen for *Target Platform*. It is a good idea to target 64bit architectures for performance in VR. Select *Build* to compile the executable.

G.5 Testing the project

Unit tests can be run by first selecting *Window/Editor Tests Runner*



```
Editor Tests    Inspector
Run All | Run Selected | Rerun Failed
▼ ⊘ BlueComic
  ▼ ⊘ BlueComic
    ▼ ✓ CursorInputTests
        ✓ AlwaysChangeWithZeroThresh
        ✓ Down
        ✓ Left
        ✓ NoChangeOnZeroAxis
        ✓ Right
        ✓ Up
    ▶ ✓ DatabaseTests
    ▼ ✓ FSMTests
        ✓ CompleteMachineValidStarts
        ✓ EmptyMachineInvalidStart
        ✓ SingleStateMachineInvalidStart
        ✓ SingleStateMachineValidStart
    ▼ ✓ FuzzyStringTests
        ✓ Distance
        ✓ OrderedCollectionSizeSame
        ✓ OrderPreservedEmptyPattern
        ✓ OrderSimple
        ✓ OrderSimpleMap
        ✓ ZeroDistanceSameStrings
    ▼ ✓ KeyTests
        ✓ SetKey
    ▼ ⊘ MeshRepositoryTests
        ⊘ GetStool
        ⊘ GetStoolViaWait
        ✓ Initialisation
        ✓ NullOnFailure
    ▼ ✓ RendererExporterTests
        ✓ ExportedCorrectAlbedo
        ✓ ExportedCorrectColour
        ✓ ExportedCorrectName
        ✓ ExportedJson
BlueComic (BlueComic)
```

Selecting *Run All* or *Run Selected* will run all the tests or the selected subset respectively. The usual green for pass, red for fail colour scheme is used. Note that tests are located inside an Editor folder to exclude them from the build process.

G.5.1 Integration

Integration tests are run as part of the test scene mentioned above. To run these manually, it is easier to select *Unity Test Tools/Integration Test Runner* and do the same as for unit tests.

Note that git hooks are included in the project's root directory, which include a python script that attempts to launch Unity using the command line prior to every commit. If successful, it runs integration tests and generates the results in the same directory. Ensure that the script is able to locate Unity by adding it to your PATH environment variable and also make sure that you have python installed and pointed to by the script.

To enable this process, you will need to replace the *hooks* directory (inside your *.git* repository directory) with the *hooks* directory located at project's root directory.

G.6 Code Documentation

Please refer to `https://zcabjro.bitbucket.io` for live documentation concerning the editor's codebase, as well as the web server's RESTful API. An offline copy of this documentation submitted accompanies this report.

G.7 Running the Web Server

In order for the web server to properly function as an object upload and retrieval mechanism, it is necessary to deploy it live with the necessary Amazon S3 credentials configured as environment variables. Hosting and configuration of S3 falls outside the scope of this report; however, to run the web server locally, one need only navigate to the root directory and, with node.js installed, run the command: *npm start*

Appendix H: User Manual

See next page. For a video walkthrough/demonstration of how to use the editor, please visit: `https://youtu.be/4y6hHjxuHVI`.

Setup Instructions

Required

- HTC Vive with accompanying lighthouse trackers and motion controllers.
- Installation of HTC Vive software (https://www.vive.com/uk/setup/)
- Read/write directory containing the built project. Inside, it should look like this:

BlueComic.exe	The executable used to run the editor.
BlueComic_Data	Data files used solely by the application. **Do not modify this directory.**
Data	User data, containing 3D models and textures for use within the editor.

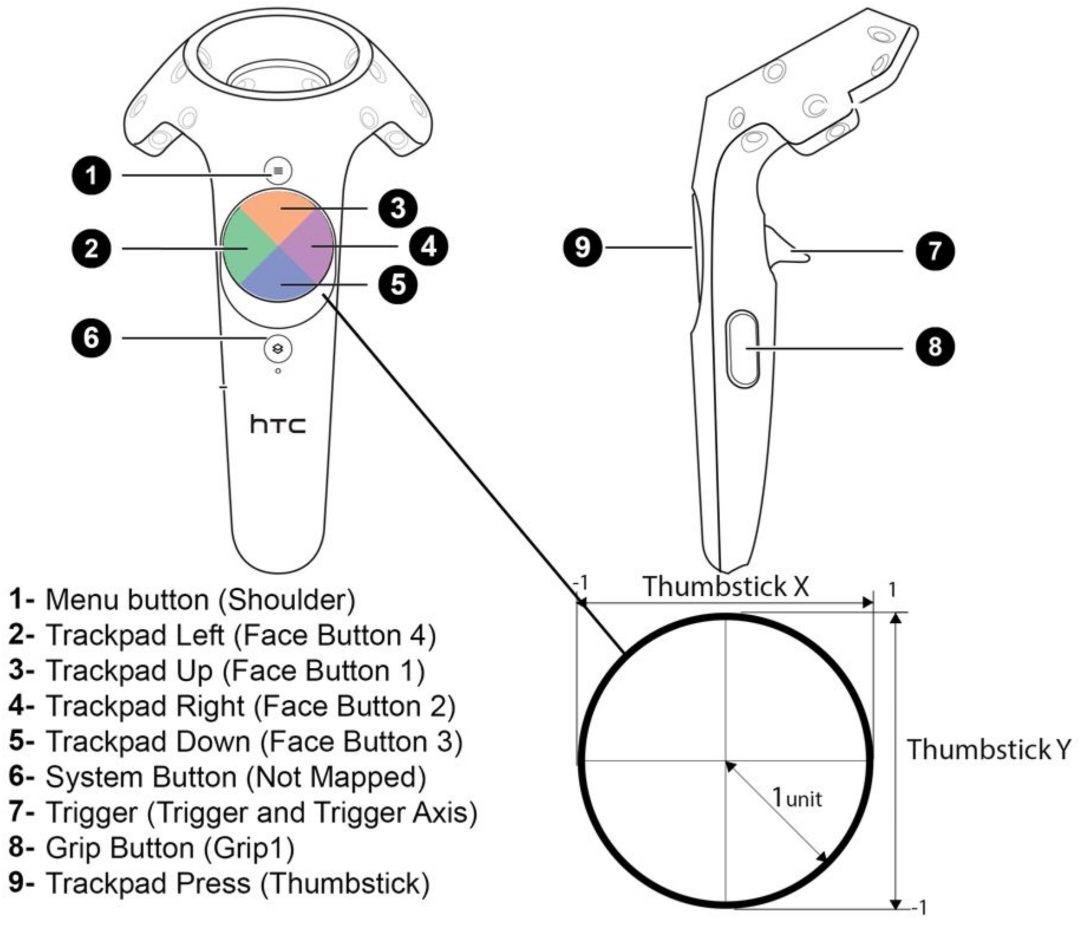
The *Data* directory, as noted above, is available for the user to add/remove resources to/from. Inside, it should look like this:

models	Directory for storing .obj files for importing 3D models.
textures	Directory for storing .png files for importing textures.
speech	Not used.

Note
The *Data* directory and its subdirectories are automatically added if they are not found by the editor, so it is not necessary to add these directories yourself unless you have models/textures you wish to import. If, instead, you want to use the Blue Comic Web service for importing 3D models, then it is fine to forget about this folder during setup.

There is one other location of importance used by the editor, which is used to store scene files and database information. This is *AppData/LocalLow/JackRoper/Blue Comic*

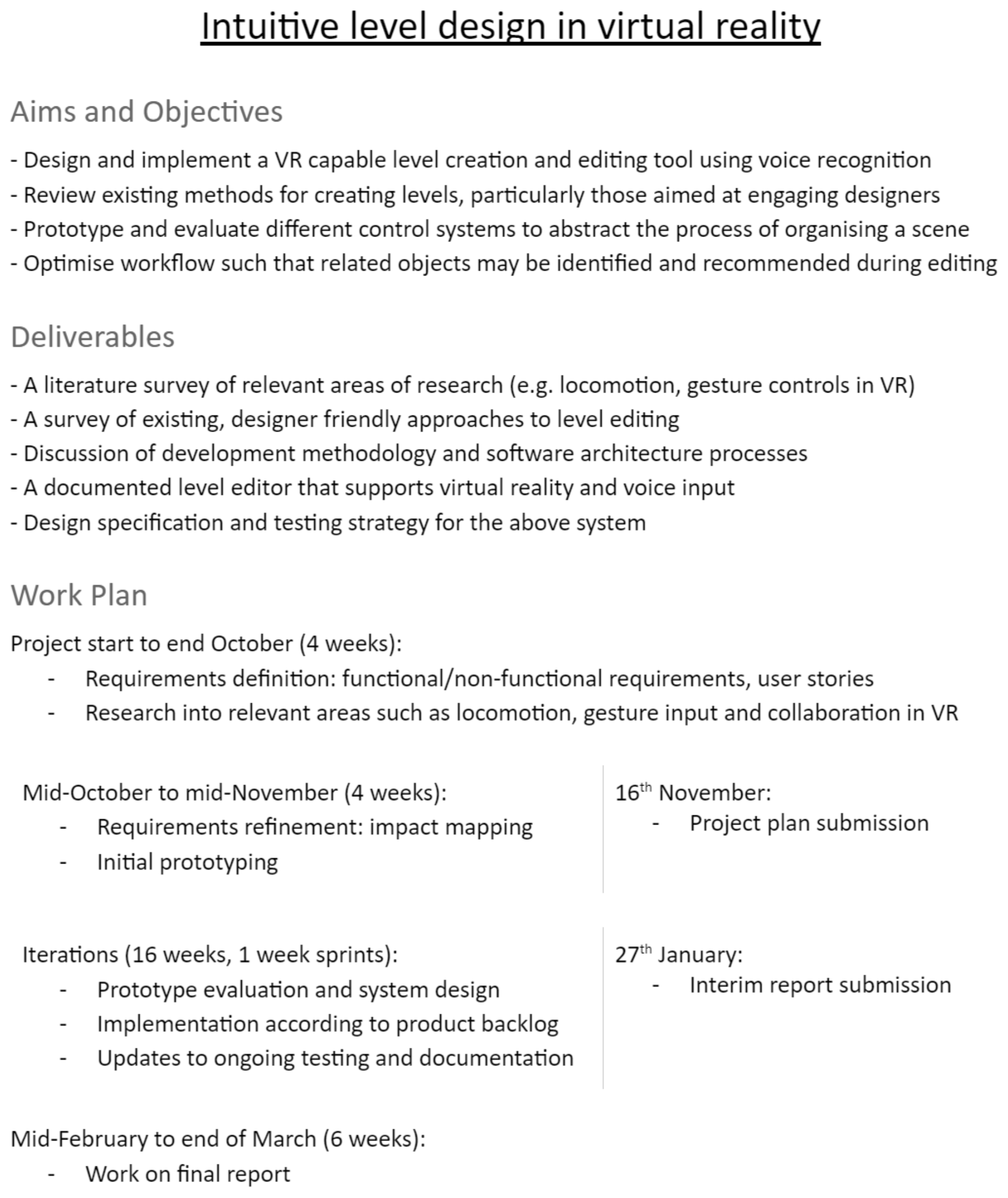
Controls


1
2
3
4
5
6
7
8
9
htc
1- Menu button (Shoulder)
2- Trackpad Left (Face Button 4)
3- Trackpad Up (Face Button 1)
4- Trackpad Right (Face Button 2)
5- Trackpad Down (Face Button 3)
6- System Button (Not Mapped)
7- Trigger (Trigger and Trigger Axis)
8- Grip Button (Grip1)
9- Trackpad Press (Thumbstick)
-1
Thumbstick X
1
Thumbstick Y
1 unit
-1

Controller	Input	State	Action	Description
Right	9	Navigation	Teleport	The default state is navigation. Aim the right controller and press down on the trackpad to teleport.
Right	7	Navigation Keyboard Input Grab	Add/remove object Press key Gesture	The right trigger is used to add an object to the scene, or for keyboard input. If pressed when the right grip button is being held, the target object is removed. While grabbing an object, holding the right trigger down and moving the controller will signal a gesture for gesture commands. By default, gesturing horizontally to the right will duplicate the held object.
Right	8	General Navigation Navigation	Back Grab/scale object Remove object	The right grip button is generally used to cancel an action (e.g. adding an object, keyboard input). By holding it down and pressing the right trigger, the target

				object is removed. It is also used to pick up and manipulate an object when pressed and held in an object’s vicinity. Grabbed objects can be scaled by holding the same button on the other controller and adjusting the distance between controllers.
Left	7	Navigation	Toggle menu	The left trigger is used to toggle the left hand menu.
Left	8	Navigation	Grab/scale object	The left grip button is used to pick up and manipulate an object when pressed and held in an object’s vicinity. Grabbed objects can be scaled by holding the same button on the other controller and adjusting the distance between controllers.
Left	9	Menu	Select/navigate	Pressing the left trackpad’s center selects the highlighted menu item. Pressing its top or bottom navigates the menu items.

Appendix I: Project Plan



Intuitive level design in virtual reality

Aims and Objectives

- Design and implement a VR capable level creation and editing tool using voice recognition
- Review existing methods for creating levels, particularly those aimed at engaging designers
- Prototype and evaluate different control systems to abstract the process of organising a scene
- Optimise workflow such that related objects may be identified and recommended during editing

Deliverables

- A literature survey of relevant areas of research (e.g. locomotion, gesture controls in VR)
- A survey of existing, designer friendly approaches to level editing
- Discussion of development methodology and software architecture processes
- A documented level editor that supports virtual reality and voice input
- Design specification and testing strategy for the above system

Work Plan

Project start to end October (4 weeks):

- Requirements definition: functional/non-functional requirements, user stories
- Research into relevant areas such as locomotion, gesture input and collaboration in VR

Mid-October to mid-November (4 weeks):

- Requirements refinement: impact mapping
- Initial prototyping

16th November:

- Project plan submission

Iterations (16 weeks, 1 week sprints):

- Prototype evaluation and system design
- Implementation according to product backlog
- Updates to ongoing testing and documentation

27th January:

- Interim report submission

Mid-February to end of March (6 weeks):

- Work on final report

Appendix J: Interim Report

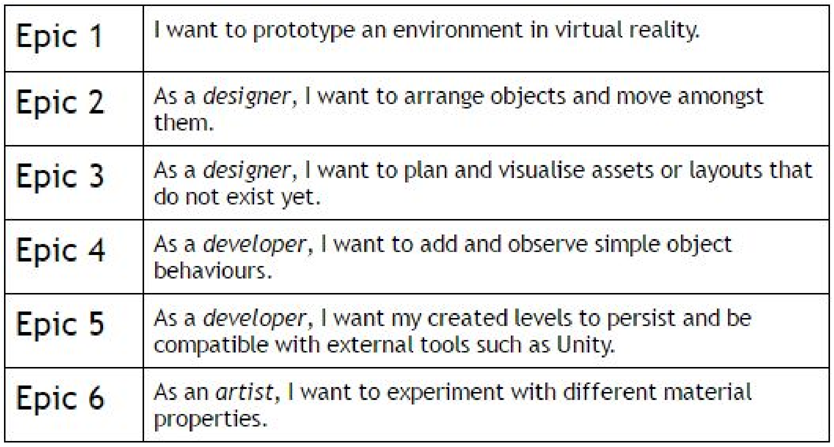
See next page.

Name	Jack Roper
Previous title	Intuitive level design in virtual reality
Current title	Immersive scene prototyping in virtual reality
Supervisor	Anthony Steed
Supervisor signature	

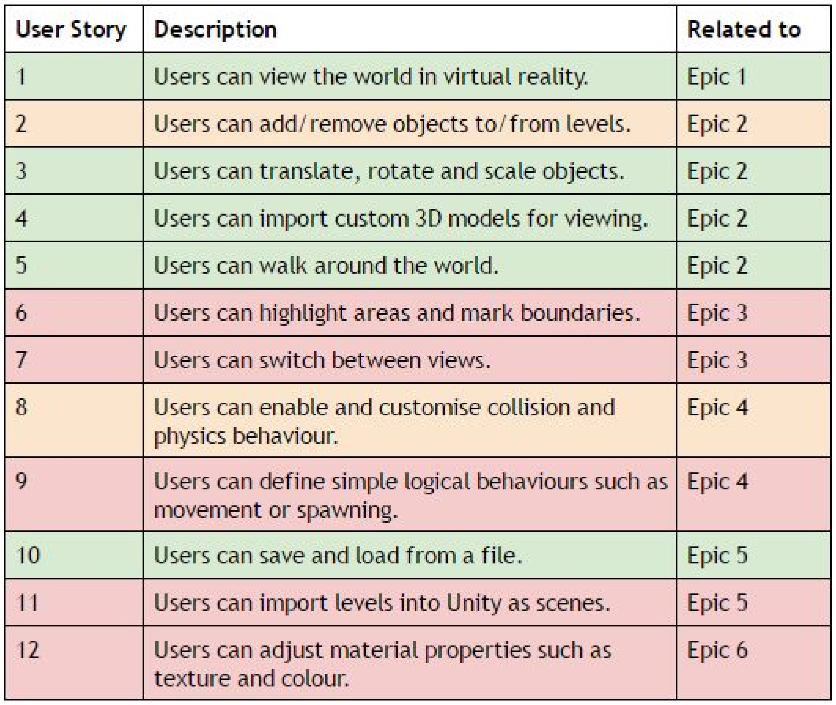
1 Progress

Below is a description of user-stories created at the start of the project. Section 1.3 discusses progress that goes beyond the stories listed.

Green = Complete
Orange = Work in progress
Red = Outstanding



Epic 1	I want to prototype an environment in virtual reality.
Epic 2	As a *designer*, I want to arrange objects and move amongst them.
Epic 3	As a *designer*, I want to plan and visualise assets or layouts that do not exist yet.
Epic 4	As a *developer*, I want to add and observe simple object behaviours.
Epic 5	As a *developer*, I want my created levels to persist and be compatible with external tools such as Unity.
Epic 6	As an *artist*, I want to experiment with different material properties.



User Story	Description	Related to
🟩 1	Users can view the world in virtual reality.	Epic 1
🟧 2	Users can add/remove objects to/from levels.	Epic 2
🟩 3	Users can translate, rotate and scale objects.	Epic 2
🟩 4	Users can import custom 3D models for viewing.	Epic 2
🟩 5	Users can walk around the world.	Epic 2
🟥 6	Users can highlight areas and mark boundaries.	Epic 3
🟥 7	Users can switch between views.	Epic 3
🟧 8	Users can enable and customise collision and physics behaviour.	Epic 4
🟥 9	Users can define simple logical behaviours such as movement or spawning.	Epic 4
🟩 10	Users can save and load from a file.	Epic 5
🟥 11	Users can import levels into Unity as scenes.	Epic 5
🟥 12	Users can adjust material properties such as texture and colour.	Epic 6

1.1 Complete

From the user's perspective, and with reference to the above stories, they are currently able to:

- [1] Start the application using a HTC Vive.
- [2] Remove objects from a scene using the controller as input.
- [3] Translate and rotate objects via grab, while scaling by adjusting the distance between hands.
- [4] Supply 3D models (.obj) and textures for use in a scene definition.
- [5] Navigate a default scene consisting of simple, editable geometry (see 1.3.3), where navigation uses room-scale tracking as well as a teleportation metaphor for long distances.
- [10] Save/load to/from a scene definition file (json format).

1.2 Work in progress

- [2] Adding objects remains a WIP, as the user-interface for selection of the desired object remains unfinished. Currently, users are able to provide input via a virtual keyboard in the world, which in turn uses a fuzzy string algorithm to compute a shortlist of objects with similar names to the search term. This will also be extended to allow for searching of an online database of models, as well as integrating with keyword descriptions of objects so that a name need not be known in advance. Much of the scaffolding for these meta-tags is already complete, including a database and meta-file management.
- [8] Customisation of physics properties is currently limited to the scene definition file where users can mark objects as physics objects, as well as specifying the type of collider. For this task's completion, I intend to allow users to toggle physics and adjust the collider dimensions where a box collider has been used in place of a mesh collider (currently the box is set using the bounds of the 3D model).

1.3 Further discussion of work done

1. Speech input was integrated from an early stage, allowing both keyword recognition and user-dictation, providing a viable substitute for tasks most affected by the lack of keyboard and mouse (e.g. text input, undo/redo).
2. Gesture recognition is another possible means for responding to user input. One example is a gesture that duplicates an object in the scene. To this end, I have programmed a rudimentary multilayer perceptron algorithm that uses backpropagation to learn from patterns of 3D coordinates. I hope to use it in combination with some other input to increase its accuracy and utility.
3. User actions have been implemented to allow undo/redo for easier and more rapid prototyping.
4. Users are able to edit walls and create corridors, walkways and similar structures using a simplified world-in-miniature metaphor. By flipping cells in a grid on/off, a marching squares algorithm generates a low-poly mesh for walls and the floor beneath them.
5. To assist users in accomplishing their tasks, they may interact with a world-space user-interface attached to their left-arm. Through this interface they can trigger modes such as wall editing or keyboard input.
6. Much of the codebase has undergone a process of design and refactorisation to support inversion of control and dependency injection. By default, Unity presents an unfriendly environment for testing, as code will commonly depend on Unity's own static APIs and routines. Through heavy use of interfaces, favouring composition over inheritance, much of the project's code can be completely mocked and tested in isolation from Unity. It is an ongoing effort to ensure that new and old code alike adheres to this approach.

1.4 Outstanding

1. Selection and manipulation of sub-regions/areas within a scene is a high priority task likely to be implemented within the next week. With it, users will not only be able to envision and label areas within their scene, but also group objects to make them reusable later on.

2. Switching between views was an early task that I had in mind that allowed designers to momentarily step away from the user's perspective. However, this perspective is the main reason for prototyping in VR and so I have chosen to focus on other tasks that enrich this perspective further.

3. Adding simple behaviours, such as spawning and pathfinding, is considered to be a stretch goal as it demands strict and separate consideration about how users will prescribe such behaviours and whether the necessary architectural baggage is worth the expense (e.g. a pathfinding solution based on A* might require a grid definition, which may not fit with the user's scene as a pathfinding solution must often be tailored to the application that uses it.

4. Importing a scene into Unity for continued prototyping remains a priority task. It is currently blocked by other outstanding features as the resulting import will largely depend on currently incomplete scope of object definitions and behaviour.

5. Adjustment of material properties is another high priority task - one that will likely be accomplished using a palette of loaded materials and textures to choose from.

Appendix K: Code Listing

The following code listing is not complete. A sample was chosen to offer variety. Some formatting has had to be corrected to accommodate the reading constraints of this document. The reader is asked to view multi-line statements with this constraint in mind.

```
using System.Collections.Generic;
using UnityEngine;

namespace BlueComic
{
  namespace Gesture
  {
    /// <summary>
    /// Supported gestures.
    /// </summary>
    public enum GestureType
    {
      Duplicate
    }

    /// <summary>
    /// Interface for recording and recognising gestures.
    /// </summary>
    public interface IGestureRecogniser
    {
      /// <summary>
      /// Is the recogniser currently recording
      /// </summary>
      bool IsRecording { get; }

      /// <summary>
      /// Add a classifier that recognises a specific gesture type.
      /// </summary>
      /// <param name="gesture">Gesture type.</param>
      /// <param name="classifier">Classifier that recognises the gesture.</param>
      void AddGesture(GestureType gesture, Classifier classifier);

      /// <summary>
      /// Start recording a gesture.
      /// </summary>
      void StartRecording();

      /// <summary>
      /// Called once per frame to record gesture sample.
      /// </summary>
      void Update();

      /// <summary>
      /// Stop recording a gesture.
      /// </summary>
      void StopRecording();

      /// <summary>
      /// Check if a gesture has been recognised by a registered classifier.
      /// </summary>
      /// <param name="gesture">Recognised gesture type.</param>
      /// <returns>Whether or not a gesture was recognised.</returns>
      bool Recognise(out GestureType gesture);
    }

    /// <summary>
    /// See IGestureRecogniser for details of public methods.
    /// </summary>
    public class GestureRecogniser : IGestureRecogniser
    {
      private struct GestureClassifier
      {
        public GestureType Gesture { get { return gesture; } }
        public Classifier Classifier { get { return classifier; } }
        private readonly GestureType gesture;
        private readonly Classifier classifier;

        public GestureClassifier(GestureType gesture, Classifier classifier)
        {
          this.gesture = gesture;
          this.classifier = classifier;
        }
      }

      private List<GestureClassifier> classifiers = new List<GestureClassifier>();
      private List<Vector3> samples = new List<Vector3>();
      private IView view;
      private ITransformHandle gesturer;
      private double minConfidence;

      public bool IsRecording { get { return recording; } }
      private bool recording = false;

      public GestureRecogniser(IView view, ITransformHandle gesturer,
double minConfidence)
      {
        this.view = view;
        this.gesturer = gesturer;
        this.minConfidence = minConfidence;
      }
```

```
public void AddGesture(GestureType gesture, Classifier classifier)
{
  classifiers.Add(new GestureClassifier(gesture, classifier));
}

public void StartRecording()
{
  samples.Clear();
  recording = true;
}

public void Update()
{
  if (recording)
  {
    Vector3 sample = view.LookHandle.ToLocalPosition(gesturer.Position);
    samples.Add(sample);
  }
}

public void StopRecording()
{
  recording = false;
}

public bool Recognise(out GestureType gesture)
{
  double[] pattern = NormaliseSamples();
  gesture = default(GestureType);
  double highestConfidence = -1.0;

  foreach (GestureClassifier gestureClassifier in classifiers)
  {
    double confidence = gestureClassifier.Classifier.Classify(pattern);
    if (confidence >= minConfidence && confidence > highestConfidence)
    {
      gesture = gestureClassifier.Gesture;
      highestConfidence = confidence;
    }
  }

  return highestConfidence != -1.0;
}
```

```
      private double[] NormaliseSamples()
      {
        double[] pattern = new double[33];
        int step = samples.Count / 11;
        int i = 0;
        int sample = 0;
        for (; sample < step * 11; i += 3, sample += step)
        {
          Vector3 s = samples[sample] - samples[0];
          pattern[i] = s.x;
          pattern[i + 1] = s.y;
          pattern[i + 2] = s.z;
        }
        return pattern;
      }
    }
  }
}
```

```
using System;
using System.Threading;

namespace BlueComic
{
  /// <summary>
  /// Facotry for creating classifiers with predefined configurations.
  /// </summary>
  public static class ClassifierFactory
  {
    /// <summary>
    /// Classifier configuration.
    /// </summary>
    public enum Config
    {
      Gesture
    }

    /// <summary>
    /// Container for passing training arguments to separate thread for training.
    /// </summary>
    private struct TrainingArgs
    {
      public readonly Classifier Classifier;
      public readonly double[][] TrainData;
      public readonly int[] Targets;
      public readonly double TrainRate;
      public readonly double TargetError;
      public readonly int MaxEpochs;
      public readonly int MaxRestarts;
      public readonly Action<Classifier> Callback;

      public TrainingArgs(
        Classifier classifier,
        double[][] trainData,
        int[] targets,
        double trainRate,
        double targetError,
        int maxEpochs,
        int maxRestarts,
        Action<Classifier> callback)
      {
        Classifier = classifier;
        TrainData = trainData;
        Targets = targets;
        TrainRate = trainRate;
        TargetError = targetError;
        MaxEpochs = maxEpochs;
        MaxRestarts = maxRestarts;
        Callback = callback;
      }
    }

    /// <summary>
    /// Create a gesture classifier using preset configuration. Classifier is
trained using the supplied training data and classifications.
    /// </summary>
    /// <param name="config">Config to use for classifier network.</param>
    /// <param name="trainData">Training data to train classifier with.</param>
    /// <param name="targets">Classifications used for supervised learning/error
backpropagation.</param>
    /// <param name="callback">Callback for classifier delivery.</param>
    public static void CreateGestureClassifier(Config config, double[][] trainData,
int[] targets, Action<Classifier> callback)
    {
      TrainingArgs trainingArgs;
      switch (config)
      {
        case Config.Gesture:
          trainingArgs = new TrainingArgs(
            classifier: new Classifier(numInputs: 33, numHiddenNeurons: 11),
            trainData: trainData,
            targets: targets,
            trainRate: 0.5,
            targetError: 0.05,
            maxEpochs: 3000,
            maxRestarts: 20,
            callback: callback
          );
          break;
        default:
          throw new ArgumentException("Unhandled config: " + config);
      }

      new Thread(Create).Start(trainingArgs);
    }

    private static void Create(object obj)
    {
      if (!(obj is TrainingArgs))
      {
        throw new ArgumentException("Object is not of type " +
          typeof(TrainingArgs).FullName);
      }

      TrainingArgs args = (TrainingArgs)obj;
      Classifier classifier = args.Classifier;
      classifier.Train(args.TrainData, args.Targets, args.TrainRate,
args.TargetError, args.MaxEpochs, args.MaxRestarts);
      args.Callback(classifier);
    }
  }

  /// <summary>
  /// Classifier class for classifying arbitrary input vectors.
  /// </summary>
  [Serializable]
  public class Classifier
  {
    private BDN[] hiddenNeurons;
    private BDN outputNeuron;

    public double Error
    {
      get
      {
        return error;
      }
    }
    private double error;

    /// <summary>
    /// Create new classifier network.
    /// </summary>
    /// <param name="numInputs">Size of input vector.</param>
    /// <param name="numHiddenNeurons">Number of neurons in the hidden
layer.</param>
    public Classifier(int numInputs, int numHiddenNeurons)
    {
      CreateNeurons(numInputs, numHiddenNeurons);
    }

    private void CreateNeurons(int numInputs, int numHiddenNeurons)
    {
      hiddenNeurons = new BDN[numHiddenNeurons];
      for (int i = 0; i < hiddenNeurons.Length; i++)
      {
        hiddenNeurons[i] = new BDN(numInputs);
      }
      outputNeuron = new BDN(numHiddenNeurons);
    }

    private void InitNeurons(Random random)
    {
      for (int i = 0; i < hiddenNeurons.Length; i++)
      {
        hiddenNeurons[i].InitWeights(random);
      }
      outputNeuron.InitWeights(random);
    }

    /// <summary>
    /// Train the classifier using the supplied data, classifications and training
parameters. See ClassifierFactory for convenient construction instead.
    /// </summary>
    /// <param name="trainData">Training data to train on.</param>
    /// <param name="targets">Classifications of training data to use for
training.</param>
    /// <param name="trainRate">How dramatically network responds to error. Smaller
means longer training time but smoother convergence.</param>
    /// <param name="targetError">Acceptable margin of error.</param>
    /// <param name="maxEpochs">Maximum number of iterations to reach targetError
before restarting.</param>
    /// <param name="maxRestarts">Maximum number of restarts before declaring
failure.</param>
    /// <returns>Final error of the network using supplied training data.</returns>
    /// <remarks>
    /// Final error should not be relied upon for measuring accuracy of classifier,
as overfitting may occur.
    /// Use a separate testing set of data for better measuring.
    /// </remarks>
    public double Train(double[][] trainData, int[] targets, double trainRate,
double targetError, int maxEpochs, int maxRestarts)
    {
      // Random used for initialising input weights
      Random random = new Random();

      // Initialise neurons for first training attempt
      InitNeurons(random);

      int epoch = 0;
      int restarts = 0;
      error = 0.0;
      do
      {
        // For each training pattern, run a forward and backward pass
        for (int i = 0; i < trainData.Length; i++)
        {
          double[] pattern = trainData[i];
          double patternTarget = targets[i];

          // Forward pass
          double[] hiddenNeuronActivations = new double[hiddenNeurons.Length];
          double[] hiddenNeuronOutputs = new double[hiddenNeurons.Length];
          for (int j = 0; j < hiddenNeurons.Length; j++)
          {
            BDN hiddenNeuron = hiddenNeurons[j];
            double activation = hiddenNeuron.Activate(pattern);
            hiddenNeuronActivations[j] = activation;
            hiddenNeuronOutputs[j] = Sigmoid(activation);
          }
          double outputNeuronActivation =
outputNeuron.Activate(hiddenNeuronOutputs);
          double outputNeuronOutput = Sigmoid(outputNeuronActivation);

          // Add squared pattern error to total error
          error += Math.Pow(patternTarget - outputNeuronOutput, 2);

          // Backward pass
          double outputDelta = (patternTarget - outputNeuronOutput) *
SigmoidDeriv(outputNeuronActivation);
```

```
          outputNeuron.AdjustWeights(trainRate, outputDelta, hiddenNeuronOutputs);
          for (int j = 0; j < hiddenNeurons.Length; j++)
          {
            BDN hiddenNeuron = hiddenNeurons[j];
            double hiddenDelta = outputDelta * outputNeuron.Weight(j + 1) *
SigmoidDeriv(hiddenNeuronActivations[j]);
            hiddenNeuron.AdjustWeights(trainRate, hiddenDelta, pattern);
          }
        }

        // Square root the average error for a conservative estimate
        error = Math.Sqrt(error / trainData.Length);

        if (++epoch >= maxEpochs)
        {
          // If we run out of attempts, finish now
          if (restarts >= maxRestarts)
          {
            return error;
          }

          // Otherwise, restart the network for another attempt
          InitNeurons(random);
          epoch = 0;
          restarts += 1;
        }

      } while (error > targetError);

      return error;
    }

    /// <summary>
    /// Classify the supplied input vector.
    /// </summary>
    /// <param name="pattern">Input vector to classify.</param>
    /// <returns>Confidence that the pattern is like the one learned from
training.</returns>
    public double Classify(double[] pattern)
    {
      double[] hiddenNeuronActivations = new double[hiddenNeurons.Length];
      double[] hiddenNeuronOutputs = new double[hiddenNeurons.Length];
      for (int j = 0; j < hiddenNeurons.Length; j++)
      {
        BDN hiddenNeuron = hiddenNeurons[j];
        double activation = hiddenNeuron.Activate(pattern);
        hiddenNeuronActivations[j] = activation;
        hiddenNeuronOutputs[j] = Sigmoid(activation);
      }
      double outputNeuronActivation = outputNeuron.Activate(hiddenNeuronOutputs);
      double outputNeuronOutput = Sigmoid(outputNeuronActivation);
      return outputNeuronOutput;
    }

    private static double Sigmoid(double x)
    {
      return 1 / (1 + Math.Exp(-x));
    }

    private static double SigmoidDeriv(double x)
    {
      double fx = Sigmoid(x);
      return fx * (1 - fx);
    }
  }

  /// <summary>
  /// Binary Decision Neuron: computation model that uses weighted inputs and
activation function to produce an ouput.
  /// Similar to how a neuron behaves.
  /// </summary>
  [Serializable]
  public class BDN
  {
    private double[] weights;

    /// <summary>
    /// New BDN instance with supplied input vector size.
    /// </summary>
    /// <param name="numInputs">Size of input vector.</param>
    public BDN(int numInputs)
    {
      this.weights = new double[1 + numInputs];
    }

    /// <summary>Get weight value for input at supplied index.</summary>
    /// <param name="index">Input index.</param>
    /// <returns>Weight value applied to specified input.</returns>
    public double Weight(int index)
    {
      return weights[index];
    }
```

```
/// <summary>
/// Randomly initialise BDN weights.
/// </summary>
/// <param name="random"></param>
public void InitWeights(Random random)
{
  for (int i = 0; i < weights.Length; i++)
  {
    weights[i] = 0.2 * random.NextDouble() - 0.1;
  }
}

/// <summary>
/// Adjust weights according to training rate and error/delta.
/// </summary>
/// <param name="trainRate">How dramatic an adjustment.</param>
/// <param name="delta">Error/delta measured during training.</param>
/// <param name="inputs">Inputs to the BDN.</param>
public void AdjustWeights(double trainRate, double delta, double[] inputs)
{
  weights[0] += trainRate * delta; // bias weight
  for (int i = 0; i < inputs.Length; i++)
  {
    weights[i + 1] += trainRate * delta * inputs[i];
  }
}

/// <summary>
/// Attempt to activate the BDN.
/// </summary>
/// <param name="inputs">Input vector.</param>
/// <returns>Activation value.</returns>
public double Activate(double[] inputs)
{
  return WeightedSum(inputs);
}

private double WeightedSum(double[] inputs)
{
  double sum = weights[0]; // bias weight
  for (int i = 0; i < inputs.Length; i++)
  {
    sum += inputs[i] * weights[i + 1];
  }
```

```
      return sum;
    }
  }
}
```

```
using Istepaniuk.StringDistance;
using System;
using System.Linq;
using System.Collections.Generic;

namespace BlueComic
{
  namespace Utility
  {
    /// <summary>
    /// Helper for calculating string distances from a supplied pattern.
    /// </summary>
    public static class FuzzyString
    {
      private static DamerauLevenshteinDistanceCalculator calculator = new
DamerauLevenshteinDistanceCalculator();

      /// <summary>
      /// Calculate the string distance between two strings.
      /// </summary>
      /// <param name="a">First string.</param>
      /// <param name="b">Second string.</param>
      /// <returns>String distance.</returns>
      public static int Distance(string a, string b)
      {
        return calculator.Distance(a, b);
      }

      /// <summary>
      /// Orders a collection of strings by their distance to a supplied pattern.
      /// </summary>
      /// <param name="dictionary">Collection to order.</param>
      /// <param name="pattern">Pattern string to calculate distance from.</param>
      /// <returns>Distance-ordered collection of strings.</returns>
      public static IEnumerable<string> Search(IEnumerable<string> dictionary,
string pattern)
      {
        return dictionary.OrderBy((string s) => calculator.Distance(Truncate(s,
pattern), pattern));
      }
      /// <summary>
      /// Orders a collection of items by their distance to a supplied pattern using
their string mapping.
      /// </summary>
      /// <typeparam name="T">Type of item mapped from.</typeparam>
      /// <param name="items">Collection to order.</param>
      /// <param name="mapper">Mapping from item to string.</param>
      /// <param name="pattern">Pattern string to calculate distance from.</param>
      public static void Order<T>(ref IEnumerable<T> items, Func<T, string> mapper,
string pattern)
      {
        items = items.OrderBy((item) =>
           calculator.Distance(Truncate(mapper(item), pattern), pattern)
        );
      }

      private static string Truncate(string s, string pattern)
      {
        if (s.Length > pattern.Length)
          return s.Substring(0, Math.Min(s.Length, pattern.Length));

        return s;
      }
    }
  }
}
```

```
using BlueComic.Utility;
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

namespace BlueComic
{
  /// <summary>
  /// Describes a mesh/model
  /// </summary>
  public struct MeshDescriptor
  {
    /// <summary>
    /// Mesh/model name
    /// </summary>
    public string Name { get { return name; } }
    private readonly string name;

    /// <summary>
    /// Create a mesh descriptor
    /// </summary>
    /// <param name="name">Mesh/model name</param>
    public MeshDescriptor(string name)
    {
      if (name == null || name == "")
      {
        throw new ArgumentException("Mesh name cannot be null or empty");
      }
      this.name = name;
    }

    public override int GetHashCode()
    {
      return Utils.GetHashCode(name);
    }
    public override bool Equals(object obj)
    {
      if (!(obj is MeshDescriptor))
      {
        return false;
      }
      MeshDescriptor other = (MeshDescriptor)obj;
      return name == other.name;
    }
  }

  /// <summary>
  /// Interface for repository of mesh resources.
  /// </summary>
  public interface IMeshRepository
  {
    /// <summary>
    /// Retrieve a mesh from the repository.
    /// </summary>
    /// <param name="desc">Mesh description.</param>
    /// <param name="callback">Callback for delivering mesh.</param>
    void GetMesh(MeshDescriptor desc, Action<Mesh> callback);
    /// <summary>
    /// Retrieve a mesh from the repository.
    /// </summary>
    /// <param name="desc">Mesh description.</param>
    /// <param name="callback">Callback for delivering mesh.</param>
    /// <returns>Enumerator for waiting on.</returns>
    IEnumerator WaitForMesh(MeshDescriptor desc, Action<Mesh> callback);
  }

  /// <summary>
  /// Cachee repository of used meshes.
  /// </summary>
  public class MeshRepository : IMeshRepository
  {
    private Dictionary<MeshDescriptor, Mesh> meshes =
      new Dictionary<MeshDescriptor, Mesh>();

    private IRoutineScheduler scheduler;

    public MeshRepository(IRoutineScheduler scheduler)
    {
      if (scheduler == null)
        throw new ArgumentNullException("Scheduler cannot be null.");

      this.scheduler = scheduler;
    }

    /// <summary>
    /// Create a mesh repository, adding the supplied meshes immediately.
    /// </summary>
    /// <param name="desc">Descriptions of the meshes to be added.</param>
    /// <param name="mesh">Meshes to add.</param>
    public MeshRepository(IRoutineScheduler scheduler, MeshDescriptor[] descriptors,
Mesh[] meshes)
    {
      if (scheduler == null)
        throw new ArgumentNullException("Scheduler cannot be null.");
      if (descriptors.Length != meshes.Length)
        throw new ArgumentException("Descriptors and meshes are not of equal
length.");

      this.scheduler = scheduler;

      for (int i = 0; i < descriptors.Length; i++)
      {
        if (this.meshes.ContainsKey(descriptors[i]))
          throw new ArgumentException(string.Format("Duplicate descriptor: {0}",
descriptors[i]));

        this.meshes[descriptors[i]] = meshes[i];
      }
    }
```

```
    /// <summary>
    /// Retrieve a mesh from the repository.
    /// </summary>
    /// <param name="desc">Mesh description.</param>
    /// <param name="callback">Callback for delivering mesh (null on
failure).</param>
    public void GetMesh(MeshDescriptor desc, Action<Mesh> callback)
    {
      // Check cached meshes
      Mesh mesh;
      if (meshes.TryGetValue(desc, out mesh))
      {
        callback(mesh);
      }
      else
      {
        scheduler.Schedule(LoadMesh(desc, callback));
      }
    }

    /// <summary>
    /// Retrieve a mesh from the repository.
    /// </summary>
    /// <param name="desc">Mesh description.</param>
    /// <param name="callback">Callback for delivering mesh (null on
failure).</param>
    /// <returns>Enumerator for waiting on.</returns>
    public IEnumerator WaitForMesh(MeshDescriptor desc, Action<Mesh> callback)
    {
      // Check cached meshes
      Mesh mesh;
      if (meshes.TryGetValue(desc, out mesh))
      {
        callback(mesh);
      }
      else
      {
        yield return LoadMesh(desc, callback);
      }
    }

    private IEnumerator LoadMesh(MeshDescriptor desc, Action<Mesh> callback)
    {
      // Load mesh from file
      Mesh mesh = null;
      string filePath = Path.Normalise(
        Path.GetUserData(Path.UserDataDir.Models),
        desc.Name
      );
      yield return ObjImporter.ImportFile(filePath, m => mesh = m);

      // Last second check for existing mesh that may have been added asynchronously
      Mesh existing;
      if (meshes.TryGetValue(desc, out existing))
      {
        // Use the existing one if so
        mesh = existing;
      }
      else if (mesh != null)
      {
        // Set mesh name for export
        mesh.name = desc.Name;
        // Cache mesh
        meshes.Add(desc, mesh);
      }

      callback(mesh); // NOTE: Null if model not found
      yield break;
    }
  }
}
```

```
using UnityEngine;

/// <summary>
/// Singleton implementation of Unity's MonoBehaviour.
/// </summary>
/// <typeparam name="T"></typeparam>
public class SingletonMonoBehaviour<T> : MonoBehaviour where T :
SingletonMonoBehaviour<T>
{
  protected static T instance;

  protected virtual void Awake()
  {
    if (instance != this && instance != null)
    {
      Destroy(gameObject);
      return;
    }
    instance = this as T;
  }

  protected virtual void OnDestroy()
  {
    if (instance == this)
    {
      instance = null;
    }
  }
}
```

```
using UnityEngine;
using Valve.VR;

namespace BlueComic
{
  /// <summary>
  /// Class for getting virtualised input state from VR devices
  /// </summary>
  public class VRInput : Input, IVirtualInput
  {
    private SteamVR_TrackedObject leftController;
    private SteamVR_TrackedObject rightController;

    public VRInput(SteamVR_TrackedObject leftController,
                   SteamVR_TrackedObject rightController)
    {
      this.leftController = leftController;
      this.rightController = rightController;
    }

    /// <summary>
    /// Checks if grab input was detected, outputting the side if so.
    /// </summary>
    /// <param name="side">Side of the grab input.</param>
    /// <returns>Whether or not grab input was detected.</returns>
    public override bool GetGrabDown(out Side side)
    {
      if (GetButton(VirtualButton.LeftGrab, InputState.Down))
      {
        side = Side.Left;
        return true;
      }

      side = Side.Right;
  return GetButton(VirtualButton.RightGrab, InputState.Down);
}

/// <summary>
/// Gets the hand position for the requested side
/// </summary>
/// <param name="side">The hand of interest</param>
/// <returns>Hand position</returns>
public override Vector3 GetHandPosition(Side side)
{
  return side == Side.Left
    ? leftController.transform.position
    : rightController.transform.position;
}

/// <summary>
/// Gets a value representing axis input
/// </summary>
/// <param name="side">The axis of interest</param>
/// <returns>Axis input</returns>
public override Vector2 GetAxis(Side side)
{
  SteamVR_Controller.Device controller = side == Side.Left
    ? GetLeftController()
    : GetRightController();
  return controller != null
    ? controller.GetAxis(EVRButtonId.k_EButton_SteamVR_Touchpad)
    : Vector2.zero;
}

/// <summary>
/// Gets a value representing input velocity
/// </summary>
/// <param name="side">The input side of interest</param>
/// <returns>Input velocity</returns>
public override Vector3 GetVelocity(Side side)
{
  SteamVR_Controller.Device controller = side == Side.Left
    ? GetLeftController()
    : GetRightController();
  return controller != null
    ? controller.velocity
    : Vector3.zero;
}
```

```
/// <summary>
/// Gets a value representing input angular velocity
/// </summary>
/// <param name="side">The input side of interest</param>
/// <returns>Input angular velocity</returns>
public override Vector3 GetAngularVelocity(Side side)
{
  SteamVR_Controller.Device controller = side == Side.Left
    ? GetLeftController()
    : GetRightController();
  return controller != null
    ? controller.angularVelocity
    : Vector3.zero;
}

/// <summary>
/// Checks if a virtual input is in a given state
/// </summary>
/// <param name="vButton">The virtual input to check</param>
/// <param name="state">The state to check for</param>
/// <returns></returns>
public override bool GetButton(VirtualButton vButton, InputState state)
{
  bool result = false;
  switch (this[vButton])
  {
    case RawButton.RightTrigger:
      result = GetTrigger(GetRightController(), state);
      break;
    case RawButton.RightGrip:
      result = GetGrip(GetRightController(), state);
      break;
    case RawButton.RightTouchpad:
      result = GetTouchpad(GetRightController(), state);
      break;
    case RawButton.LeftTrigger:
      result = GetTrigger(GetLeftController(), state);
      break;
    case RawButton.LeftGrip:
      result = GetGrip(GetLeftController(), state);
      break;
    case RawButton.LeftTouchpad:
      result = GetTouchpad(GetLeftController(), state);
      break;
    case RawButton.Unmapped:
      Debug.LogWarning(vButton + " is not mapped. Defaulting to false");
      result = false;
      break;
    default:
      throw new System.Exception(
        vButton +
        " mapped to unhandled input: " +
        this[vButton]
      );
  }
  return result;
}
/// <summary>
/// Tries to obtain a reference to the rightmost VR controller
/// </summary>
/// <returns></returns>
private SteamVR_Controller.Device GetRightController()
{
  return rightController.index != SteamVR_TrackedObject.EIndex.None
    ? SteamVR_Controller.Input((int)rightController.index)
    : null;
}
/// <summary>
/// Tries to obtain a reference to the leftmost VR controller
/// </summary>
/// <returns></returns>
private SteamVR_Controller.Device GetLeftController()
{
  return leftController.index != SteamVR_TrackedObject.EIndex.None
    ? SteamVR_Controller.Input((int)leftController.index)
    : null;
}

/// <summary>
/// Checks if the controller's trigger is in a given state
/// </summary>
/// <param name="controller">The controller to check</param>
/// <param name="state">The state to check for</param>
/// <returns></returns>
private static bool GetTrigger(SteamVR_Controller.Device controller,
                               InputState state)
{
  bool result = false;
  if (controller != null)
  {
    switch (state)
    {
```

```
        case InputState.Down:
          result = controller.GetPressDown(EVRButtonId.k_EButton_SteamVR_Trigger);
          break;
        case InputState.Held:
          result = controller.GetPress(EVRButtonId.k_EButton_SteamVR_Trigger);
          break;
        case InputState.Up:
          result = controller.GetPressUp(EVRButtonId.k_EButton_SteamVR_Trigger);
          break;
      }
    }
    return result;
  }
  /// <summary>
  /// Checks if the controller's grip is in a given state
  /// </summary>
  /// <param name="controller">The controller to check</param>
  /// <param name="state">The state to check for</param>
  /// <returns></returns>
  private static bool GetGrip(SteamVR_Controller.Device controller, InputState
state)
  {
    bool result = false;
    if (controller != null)
    {
      switch (state)
      {
        case InputState.Down:
          result = controller.GetPressDown(EVRButtonId.k_EButton_Grip);
          break;
        case InputState.Held:
          result = controller.GetPress(EVRButtonId.k_EButton_Grip);
          break;
        case InputState.Up:
          result = controller.GetPressUp(EVRButtonId.k_EButton_Grip);
          break;
      }
    }
    return result;
  }
  /// <summary>
  /// Checks if the controller's touch pad is in a given state
  /// </summary>
  /// <param name="controller">The controller to check</param>
  /// <param name="state">The state to check for</param>
  /// <returns></returns>
```

```
    private static bool GetTouchpad(SteamVR_Controller.Device controller,
                                    InputState state)
    {
      bool result = false;
      if (controller != null)
      {
        switch (state)
        {
          case InputState.Down:
            result =
controller.GetPressDown(EVRButtonId.k_EButton_SteamVR_Touchpad);
            break;
          case InputState.Held:
            result = controller.GetPress(EVRButtonId.k_EButton_SteamVR_Touchpad);
            break;
          case InputState.Up:
            result = controller.GetPressUp(EVRButtonId.k_EButton_SteamVR_Touchpad);
            break;
        }
      }
      return result;
    }
  }
}
```

```
namespace BlueComic
{
  namespace Operations
  {
    /// <summary>
    /// Interface for using operation monitor.
    /// </summary>
    public interface IOperationMonitor
    {
      /// <summary>
      /// Record an operation.
      /// </summary>
      /// <param name="operation">Operation to record.</param>
      void Record(Operation operation);

      /// <summary>
      /// Undo last operation.
      /// </summary>
      void UndoLast();

      /// <summary>
      /// Redo last undone operation.
      /// </summary>
      void RedoLast();
    }

    /// <summary>
    /// Class for recording, undoing and redoing operations according to Command
pattern.
    /// </summary>
    public class OperationMonitor : IOperationMonitor
    {
      private Operation[] operations;
      private int index = -1;
      private int count = 0;
      private int undoCount = 0;

      public OperationMonitor(int maxRecords)
      {
        operations = new Operation[maxRecords];
      }

      public void Record(Operation operation)
      {
        if (operation == null)
          throw new System.ArgumentNullException("Operation cannot be null");

        count++;
        undoCount = 0;
        index = (index + 1) % operations.Length;
        if (operations[index] != null)
        {
          operations[index].Free();
        }
        operations[index] = operation;
        operation.Execute();
      }

      public void UndoLast()
      {
        if (count > 0)
        {
          count--;
          undoCount++;
          operations[index].Undo();
          index = index > 0 ? index - 1 : operations.Length - 1;
        }
      }

      public void RedoLast()
      {
        if (undoCount > 0)
        {
          undoCount--;
          count++;
          index = index < operations.Length - 1 ? index + 1 : 0;
          operations[index].Execute();
        }
      }
    }
  }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using UnityEngine.Windows.Speech;

namespace BlueComic
{
  namespace Speech
  {
    /// <summary>
    /// Class for recognising speech input and exposing events for listeners
interested in speech commands.
    /// </summary>
    public class SpeechManager : IDisposable
    {
      /// <summary>
      /// Whether or not speech recognition is actively listening.
      /// </summary>
      public bool IsListening { get { return keywordRecogniser.IsRunning; } }

      /// <summary>
      /// Event for when the user says "select".
      /// </summary>
      public static event Action OnSelect;
      /// <summary>
      /// Event for when the user says "cancel".
      /// </summary>
      public static event Action OnCancel;
      /// <summary>
      /// Event for when the user says "undo".
      /// </summary>
      public static event Action OnUndo;
      /// <summary>
      /// Event for when the user says "redo".
      /// </summary>
      public static event Action OnRedo;

      /// <summary>
      /// Keyword to event mapping.
      /// </summary>
      private Dictionary<string, Action> keywordActions =
        new Dictionary<string, Action>()
        {
          { "select", delegate() { if(OnSelect != null) { OnSelect(); } } },
          { "cancel", delegate() { if(OnCancel != null) { OnCancel(); } } },
          { "undo", delegate() { if(OnUndo != null) { OnUndo(); } } },
          { "redo", delegate() { if(OnRedo != null) { OnRedo(); } } },
        };

      private KeywordRecognizer keywordRecogniser;

      /// <summary>
      /// Creates a new instance for speech recongition.
      /// </summary>
      public SpeechManager()
      {
        keywordRecogniser = new KeywordRecognizer(keywordActions.Keys.ToArray(),
                                                  ConfidenceLevel.Low);
        keywordRecogniser.OnPhraseRecognized +=
          KeywordRecogniser_OnPhraseRecognized;
      }

      /// <summary>
      /// Start listening for speech events.
      /// </summary>
      public void StartListening()
      {
        if (!keywordRecogniser.IsRunning)
        {
          keywordRecogniser.Start();
        }
      }
      private void KeywordRecogniser_OnPhraseRecognized(
        PhraseRecognizedEventArgs args)
      {
        Action keywordAction;
        if (keywordActions.TryGetValue(args.text, out keywordAction))
        {
          keywordAction.Invoke();
        }
      }

      /// <summary>
      /// Dispose of the SpeechManager's internal resources.
      /// </summary>
      public void Dispose()
      {
        keywordRecogniser.Dispose();
      }
    }
  }
}
```

```
using System;

namespace BlueComic
{
  namespace State
  {
    /// <summary>
    /// Base class for editor states.
    /// </summary>
    public abstract class FSM_State
    {

      /// <summary>
      /// Type of transition.
      /// </summary>
      public enum TransitionType
      {
        /// <summary>
        /// Exit current state and immediately enter new state.
        /// </summary>
        Replace,
```

```
        /// <summary>
        /// Pause current state and stack and enter new state.
        /// </summary>
        Stack
      }

      /// <summary>
      /// Transition arguments container.
      /// </summary>
      public class TransitionEventArgs : EventArgs
      {
        public static TransitionEventArgs Pop { get { return pop; } }
        private static readonly TransitionEventArgs pop =
          new TransitionEventArgs(StateType.Noof, TransitionType.Replace, null);

        /// <summary>
        /// Transition to state.
        /// </summary>
        public StateType State { get { return state; } }

        /// <summary>
        /// Transition type.
        /// </summary>
        public TransitionType Transition { get { return transition; } }

        /// <summary>
        /// Contextual arguments to new state.
        /// </summary>
        public object Args { get { return args; } }

        private readonly StateType state;
        private readonly TransitionType transition;
        private readonly object args;
        public TransitionEventArgs(StateType state, TransitionType transition,
object args)
        {
          this.state = state;
          this.transition = transition;
          this.args = args;
        }
      }

      /// <summary>
      /// Event for handling event transition.
      /// </summary>
      public event EventHandler<TransitionEventArgs> Transitioned;
      protected virtual void OnTransition(TransitionEventArgs e)
      {
        if (Transitioned != null)
        {
          Transitioned(this, e);
        }
      }

      /// <summary>
      /// When state first entered on the stack.
      /// </summary>
      /// <param name="stateArgs">Contextual state args.</param>
      public virtual void Enter(object stateArgs) { }

      /// <summary>
      /// When state leaves the stack.
      /// </summary>
      public virtual void Exit() { }

      /// <summary>
      /// Called once per frame for current state.
      /// </summary>
      public virtual void Update() { }

      /// <summary>
      /// When state is paused due to new state being pushed onto stack.
      /// </summary>
      public virtual void Pause() { }

      /// <summary>
      /// When state is resumed as previous state is popped from stack.
      /// </summary>
      public virtual void Resume() { }
    }
  }
}
```

```
using BlueComic.Gesture;
using BlueComic.Operations;
using UnityEngine;

namespace BlueComic
{
  namespace State
  {
    /// <summary>
    /// State for recording and classifying gestures.
    /// </summary>
    public class GestureState : FSM_State
    {
      public class Args
      {
        public GameObject TargetObject { get { return targetObject; } }
        private readonly GameObject targetObject;

        public Args(GameObject targetObject)
        {
          this.targetObject = targetObject;
        }
      }

      private IView view;
      private IGestureRecogniser gestureRecogniser;
      private IVirtualInput input;
      private OperationMonitor operationMonitor;
      private GameObject targetObject;

      public GestureState(IView view, IGestureRecogniser gestureRecogniser,
IVirtualInput input, OperationMonitor operationMonitor)
      {
        this.view = view;
        this.gestureRecogniser = gestureRecogniser;
        this.input = input;
        this.operationMonitor = operationMonitor;
      }

      public override void Enter(object stateArgs)
      {
        Args args = (Args)stateArgs;
        this.targetObject = args.TargetObject;

        gestureRecogniser.StartRecording();
        gestureRecogniser.Update();
      }

      public override void Update()
      {
        if (input.GetButton(VirtualButton.Gesture, InputState.Held))
        {
          gestureRecogniser.Update();
        }
        else
        {
          gestureRecogniser.StopRecording();
          GestureType gesture;
          if (gestureRecogniser.Recognise(out gesture))
          {
            ProcessGesture(gesture);
          }

          OnTransition(TransitionEventArgs.Pop);
        }
      }

      private void ProcessGesture(GestureType gesture)
      {
        switch (gesture)
        {
          case GestureType.Duplicate:
            GameObject duplicate = Object.Instantiate(
              targetObject,
              targetObject.transform.position + targetObject.transform.right,
              targetObject.transform.rotation
            );
            CreateObjectOperation createObjectOperation =
              new CreateObjectOperation(duplicate);
            operationMonitor.Record(createObjectOperation);
            break;
        }
      }
    }
  }
}
```

```
using System.Collections.Generic;
using UnityEngine;

namespace BlueComic
{
  namespace State
  {
    /// <summary>
    /// State type/id.
    /// </summary>
    public enum StateType
    {
      LoadScene,
      SaveScene,
      Navigation,
      AddObject,
      TransformObject,
      EditWalls,
      Keyboard,
      Gesture,
      Selection,

      /// <summary>
      /// Number of state types.
      /// </summary>
      Noof
    }

    /// <summary>
    /// Class following State pattern. Stored and switches between states.
    /// </summary>
    public class FiniteStateMachine
    {
      private FSM_State[] states;
      private Stack<FSM_State> stack = new Stack<FSM_State>();

      /// <summary>
      /// FSM constructor for organising state behaviours.
      /// NOTE: Supplied states must follow the ordering defined by StateType enum.
      /// </summary>
      /// <param name="states">States of the FSM (ordered by StateType)</param>
      public FiniteStateMachine(params FSM_State[] states)
      {
        this.states = states;
        foreach (FSM_State state in states)
        {
          state.Transitioned += HandleTransition;
        }
      }

      /// <summary>
      /// Start the state machine using the supplied start state and arguments.
      /// </summary>
      /// <param name="startState">State to start in.</param>
      /// <param name="args">Arguments for the start state.</param>
      public void Start(StateType startState, object args)
      {
```

```
  if (startState != StateType.Noof)
  {
    PushState((int)startState, args);
  }
}

/// <summary>
/// Update the state machine's current state.
/// </summary>
public void Update()
{
  if (stack.Count > 0)
  {
    stack.Peek().Update();
  }
}

private void HandleTransition(object sender, FSM_State.TransitionEventArgs e)
{
  if (e.State == StateType.Noof)
  {
    PopState();
  }
  else
  {
    int toState = (int)e.State;
    Debug.Assert(toState >= 0 && toState < states.Length);
    switch (e.Transition)
    {
      case FSM_State.TransitionType.Stack:
        StackState(toState, e.Args);
        break;
      case FSM_State.TransitionType.Replace:
        ReplaceState(toState, e.Args);
        break;
    }
  }
}

private void StackState(int state, object args)
{
  PauseState();
  PushState(state, args);
}
```

```
      private void ReplaceState(int state, object args)
      {
        if (stack.Count > 0)
        {
          stack.Pop().Exit(); // No resumption, just a straight swap
        }
        PushState(state, args);
      }

      private void PushState(int state, object args)
      {
        if (state < 0 || state >= states.Length)
          throw new System.InvalidOperationException("Cannot push out of range
state.");

        FSM_State s = states[state];
        stack.Push(s);
        s.Enter(args);
      }

      private void PauseState()
      {
        if (stack.Count > 0)
        {
          stack.Peek().Pause();
        }
      }

      private void PopState()
      {
        switch (stack.Count)
        {
          case (0):
            break;
          case (1):
            stack.Pop().Exit();
            break;
          default:
            stack.Pop().Exit();
            stack.Peek().Resume();
            break;
        }
      }
    }
  }
}
```

```
using System;
using UnityEngine;
using UnityEngine.UI;

/// <summary>
/// BlueComic is the project namespace. All editor code is contained under
BlueComic.
/// </summary>
namespace BlueComic
{
  /// <summary>
  /// Event args container for a tool selection event.
  /// </summary>
  public class ToolEventArgs : EventArgs
  {
    /// <summary>
    /// Selected tool type/identifier.
    /// </summary>
    public ToolType ToolType { get { return toolType; } }
    private readonly ToolType toolType;
    public ToolEventArgs(ToolType toolType)
    {
      this.toolType = toolType;
    }
  }


  /// <summary>
  /// Interface for working with the holotool UI.
  /// </summary>
  public interface IHolotool
  {
    /// <summary>
    /// Event for when a menu item is selected.
    /// </summary>
    event EventHandler<ToolEventArgs> ToolSelected;

    /// <summary>
    /// Event for when holotool is toggled on or off.
    /// </summary>
    event Action<bool> Toggled;

    /// <summary>
    /// Update the holotool UI.
    /// </summary>
    void Update();

    /// <summary>
    /// Switch menu group for paginated UI.
    /// </summary>
    /// <param name="menu"></param>
    void SwitchMenu(Menu menu);

    /// <summary>
    /// Force holotool toggled state.
    /// </summary>
    /// <param name="active"></param>
    void ForceActive(bool active);
  }

  /// <summary>
  /// Interface for getting trackpad input as menu cursor input.
  /// </summary>
  public interface ICursorInput
  {
    /// <summary>
    /// Get trackpad touch position in terms of cursor change. Outputs whether
trackpad was pressed or not.
    /// </summary>
    /// <param name="change"></param>
    /// <param name="select"></param>
    void GetCursor(out Vector2i change, out bool select);
  }

  /// <summary>
  /// See ICursorInput.
  /// </summary>
  public class CursorInput : ICursorInput
  {
    private IVirtualInput input;
    private float threshold;

    public CursorInput(IVirtualInput input, float threshold)
    {
      this.input = input;
      this.threshold = Mathf.Max(0.0f, threshold);
    }

    /// <summary>
    /// Poll the cursor for changes.
    /// </summary>
  /// <param name="change">Touch position on the trackpad.</param>
  /// <param name="select">Outputs whether trackpad was pressed or not.</param>
  public void GetCursor(out Vector2i change, out bool select)
  {
    select = input.GetButton(VirtualButton.Select, InputState.Down);
    Vector2 axis = input.GetAxis(Side.Left);
    change.x = Mathf.Abs(axis.x) >= threshold ? (int)Mathf.Sign(axis.x) : 0;
    change.y = Mathf.Abs(axis.y) >= threshold ? (int)Mathf.Sign(axis.y) : 0;
  }
}

/// <summary>
/// Interface for working with holotool input.
/// </summary>
public interface IHolotoolInput : ICursorInput
{
  /// <summary>
  /// Check if menu was toggled.
  /// </summary>
  /// <returns>True if menu was toggled, false otheriwse.</returns>
  bool GetToggle();
}

/// <summary>
/// See IHolotoolInput.
/// </summary>
public class HolotoolInput : CursorInput, IHolotoolInput
{
  private IVirtualInput input;

  public HolotoolInput(IVirtualInput input, float threshold) : base(input,
threshold)
  {
    this.input = input;
  }

  /// <summary>
  /// Check if menu was toggled.
  /// </summary>
  /// <returns>True if menu was toggled, false otheriwse.</returns>
  public bool GetToggle()
  {
    return input.GetButton(VirtualButton.ToggleHolotool, InputState.Down);
  }
}
```

```
/// <summary>
/// Interface for affecting menu tools.
/// </summary>
public interface ITool
{
  /// <summary>
  /// Name of the tool (displayed in the menu).
  /// </summary>
  string Name { get; set; }

  /// <summary>
  /// Type/identifier.
  /// </summary>
  ToolType ToolType { get; }

  /// <summary>
  /// Select this tool.
  /// </summary>
  void Select();

  /// <summary>
  /// Enable/disable this tool.
  /// </summary>
  /// <param name="active">Whether tool should be active in the menu or
not.</param>
  void SetActive(bool active);
}

/// <summary>
/// Tool types/identifiers.
/// </summary>
public enum ToolType
{
  EditWalls,
  Keyboard,
  Save,
  Exit,
  Refactor,
  ChangeMesh,
  ChangeMaterial,
  Group,
  Ungroup,
  Export,
  Parent
}
```

```
  /// <summary>
  /// Factory class for creating tools.
  /// </summary>
  public class ToolFactory
  {
    private GameObject toolPrefab;

    /// <summary>
    /// Create a new factory that uses clones the supplied prefab for creating 
tools.
    /// </summary>
    /// <param name="toolPrefab">Prefab to clone for each tool.</param>
    public ToolFactory(GameObject toolPrefab)
    {
      this.toolPrefab = toolPrefab;
    }

    /// <summary>
    /// Create a tool.
    /// </summary>
    /// <param name="parent">Parent to attach tool to.</param>
    /// <param name="name">Name for display.</param>
    /// <param name="toolType">Tool type/identifier.</param>
    /// <returns></returns>
    public Tool Create(Transform parent, string name, ToolType toolType)
    {
      GameObject toolObj = UnityEngine.Object.Instantiate(toolPrefab);
      toolObj.transform.SetParent(parent);
      toolObj.transform.localPosition = Vector3.zero;
      toolObj.transform.localRotation = Quaternion.Euler(Vector3.zero);
      toolObj.transform.localScale = Vector3.one;
      Selectable selectable = toolObj.GetComponentInChildren<Selectable>();
      Text nameText = selectable.GetComponentInChildren<Text>();
      Tool tool = new Tool(nameText, toolType, selectable);
      tool.Name = name;
      return tool;
    }
  }

  /// <summary>
  /// Class for managing a menu tool item.
  /// </summary>
  public class Tool : ITool
  {
    /// <summary>
    /// Display name of the tool.
    /// </summary>
    public string Name { get { return name.text; } set { name.text = value; } }

    /// <summary>
    /// Type/identifier of the tool.
    /// </summary>
    public ToolType ToolType { get { return toolType; } }
    private readonly Text name;
    private readonly ToolType toolType;

    private readonly Selectable selectable;

    /// <summary>
    /// Create a new tool instance that uses the supplied Selectable for selection.
    /// </summary>
    /// <param name="name">Display name.</param>
    /// <param name="toolType">Type/id.</param>
    /// <param name="selectable">Selectable to use for selection.</param>
    public Tool(Text name, ToolType toolType, Selectable selectable)
    {
      this.name = name;
      this.toolType = toolType;
      this.selectable = selectable;
    }

    /// <summary>
    /// Select this tool.
    /// </summary>
    public void Select()
    {
      //selectable.Select();

      var eventSystem = UnityEngine.EventSystems.EventSystem.current;
      if (eventSystem != null)
      {
        eventSystem.SetSelectedGameObject(null);
        eventSystem.SetSelectedGameObject(selectable.gameObject);
      }
    }

    /// <summary>
    /// Enable/disable this tool in the menu.
    /// </summary>
    /// <param name="active">Whether tool should be active in the menu or
not.</param>
    public void SetActive(bool active)
    {
      selectable.gameObject.SetActive(active);
    }
  }

  /// <summary>
  /// Menu layers.
  /// </summary>
  public enum Menu
  {
    Default = 0,
    Selection = 1,
  }

  /// <summary>
  /// See IHolotool.
  /// </summary>
  public class Holotool : IHolotool
  {
    public event EventHandler<ToolEventArgs> ToolSelected;
    public event Action<bool> Toggled;

    private IHolotoolInput input;
    private GameObject ui;

    private ITool[][] tools = new ITool[3][];

    private int pageIndex = 0;
    private int toolIndex = 0;

    public Holotool(IHolotoolInput input, GameObject ui, ITool[] tools, ITool[]
selectionTools, ITool[] refactorTools)
    {
      this.input = input;
      this.ui = ui;
      this.tools[0] = tools;
      this.tools[1] = selectionTools;
      this.tools[2] = refactorTools;

      // Only default tools active at start
      SetPageActive(0, true);
      SetPageActive(1, false);
      SetPageActive(2, false);
```

```
      ToolSelected += delegate (object sender, ToolEventArgs e)
      {
        if (e.ToolType == ToolType.Refactor)
          UpdateMenu(2);
      };

      if (ui.activeSelf)
      {
        Toggle();
      }
    }

    /// <summary>
    /// Switch the holotool UI to the supplied menu type.
    /// </summary>
    /// <param name="menu">Menu to switch to.</param>
    public void SwitchMenu(Menu menu)
    {
      int index = (int)menu;
      Debug.Assert(index >= 0 && index < tools.Length, "Page index is out of
range!");
      if (index >= 0 && index < tools.Length)
      {
        UpdateMenu(index);
      }
    }

    /// <summary>
    /// Highlights the menu item at the supplied index.
    /// </summary>
    /// <param name="index">Item to highlight.</param>
    private void UpdateMenu(int index)
    {
      SetPageActive(pageIndex, false);
      SetPageActive(index, true);
      pageIndex = index;
      toolIndex = 0;
      Highlight();
    }

    private void SetPageActive(int page, bool active)
    {
      for (int i = 0; i < tools[page].Length; i++)
      {
        tools[page][i].SetActive(active);
```

```
  }
}

/// <summary>
/// Update the holotool, checking for cursor input.
/// </summary>
public void Update()
{
  // Enable/disable holotool ui
  if (input.GetToggle())
  {
    Toggle();
  }

  // Holotool selection
  if (ui.activeSelf)
  {
    Vector2i change;
    bool select;
    input.GetCursor(out change, out select);
    if (select)
    {
      UpdateCursor(change);
    }
  }
}

private void UpdateCursor(Vector2i change)
{
  if (change != Vector2i.zero)
  {
    // Update tool cursor position
    toolIndex = toolIndex - change.y >= 0
      ? (toolIndex - change.y) % tools[pageIndex].Length
      : tools[pageIndex].Length - 1;
    Highlight();
  }
  else if (ToolSelected != null)
  {
    // Raise event for tool selection
    ToolEventArgs args =
      new ToolEventArgs(tools[pageIndex][toolIndex].ToolType);
    ToolSelected(this, args);
```

```
        // Toggle off if we're on the default tools
        if (pageIndex == 0)
          Toggle();
      }
    }

    /// <summary>
    /// Force holotool to be actice/inactive (ignores toggling).
    /// </summary>
    /// <param name="active">Enable or disable.</param>
    public void ForceActive(bool active)
    {
      ui.SetActive(active);
      Highlight();
    }

    private void Toggle()
    {
      ui.SetActive(!ui.activeSelf);

      if (Toggled != null)
      {
        Toggled(ui.activeSelf);
      }

      // Reset to default page on toggle
      UpdateMenu(0);

      Highlight();
    }

    private void Highlight()
    {
      tools[pageIndex][toolIndex].Select();
    }
  }
}
```

```
using System;
using System.Collections;
using UnityEngine;
using Object = UnityEngine.Object;

namespace BlueComic
{
```

```
/// <summary>
/// Factory for creating objects.
/// </summary>
public class ObjectFactory : IObjectFactory
{
  private IMaterialRepository materialRepository;
  private IMeshRepository meshRepository;
  private IRoutineScheduler scheduler;
  private GameObject baseObject;
  private BCObjectRefactorer refactorer;
  private ISceneDescriptor sceneDescriptor;

  public ObjectFactory(
    IMaterialRepository materialRepository,
    IMeshRepository meshRepository,
    IRoutineScheduler scheduler,
    GameObject baseObject,
    BCObjectRefactorer refactorer,
    ISceneDescriptor sceneDescriptor)
  {
    this.materialRepository = materialRepository;
    this.meshRepository = meshRepository;
    this.scheduler = scheduler;
    this.baseObject = baseObject;
    this.refactorer = refactorer;
    this.sceneDescriptor = sceneDescriptor;
  }

  /// <summary>
  /// Create object from supplied data, attaching to supplied parent.
  /// </summary>
  /// <param name="objectData">Object data to create from.</param>
  /// <param name="parent">Parent to attach to.</param>
  /// <param name="callback">Callback for object delivery.</param>
  public void Create(ObjectData objectData, Transform parent, Action<GameObject>
callback)
  {
    if (callback == null)
      throw new ArgumentNullException("Callback cannot be null.");

    // Load mesh
    meshRepository.GetMesh(new MeshDescriptor(objectData.Alias), (Mesh mesh) =>
    {
      // Signal failure if mesh is null
      if (mesh == null)
      {
        callback(null);
      }
      else
      {
        // Load material
        materialRepository.GetMaterial(
          new MaterialDescriptor("", "255,0,0,255"), (Material mat) =>
        {
          // Signal failure if material is null
          if (mat == null)
          {
            callback(null);
          }
          else
          {
            GameObject gameObject = CreateObject(
              objectData.Alias,
              parent,
              mesh,
              mat
            );
            callback(gameObject);
          }
        });
      }
    });
  }

  /// <summary>
  /// Create object from supplied data, attaching to supplied parent.
  /// </summary>
  /// <param name="objectData">Object data to create from.</param>
  /// <param name="parent">Parent to attach to.</param>
  /// <param name="callback">Callback for object delivery.</param>
  /// <returns>Enumerator for creating over multiple calls.</returns>
  public IEnumerator WaitForCreate(ObjectData objectData, Transform parent,
Action<GameObject> callback)
  {
    if (callback == null)
      throw new ArgumentNullException("Callback cannot be null.");

    // Load mesh
    Mesh mesh = null;
    yield return scheduler.Schedule(
      meshRepository.WaitForMesh(
        new MeshDescriptor(objectData.Alias), (Mesh m) => mesh = m));
      if (mesh != null)
      {
        Material mat = null;
        yield return scheduler.Schedule(
          materialRepository.WaitForMaterial(
           new MaterialDescriptor("", "255,255,255,255"), (Material m) => mat = m));

        if (mat != null)
        {
          GameObject gameObject = CreateObject(objectData.Alias, parent, mesh, mat);
          callback(gameObject);
          yield break;
        }
      }

      // If mesh or material are null, signal failure to the caller
      callback(null);
    }

    private GameObject CreateObject(string name, Transform parent, Mesh mesh,
Material mat)
    {
      // Create object
      BCObject obj = NewObject(name, parent, active: false);
      refactorer.RegisterObjects(obj);

      // Add grab functionality
      IGrabbable grabbable = obj.gameObject.AddComponent<GrabObject>();
      obj.Initialise(grabbable);

      // Add mesh
      obj.SetMesh(mesh);

      // Add material
      obj.SetMaterial(mat);

      // Add default collider
      AssignDefaultCollider(obj.gameObject, mesh.bounds.center, mesh.bounds.size);

      obj.gameObject.SetActive(true);
      return obj.gameObject;
    }

    private void AssignDefaultCollider(GameObject gameObject, Vector3 center,
Vector3 size)
    {
      // Add the default box collider
      BoxCollider boxCollider = gameObject.AddComponent<BoxCollider>();
      boxCollider.center = center;
      boxCollider.size = size;
      boxCollider.isTrigger = true;
    }

    private BCObject NewObject(string name, Transform parent, bool active)
    {
      // Create object
      GameObject gameObject = Object.Instantiate(baseObject);
      gameObject.SetActive(active);
      gameObject.name = name;
      gameObject.transform.SetParent(parent);

      BCObject bcObject = gameObject.GetComponent<BCObject>();
      Debug.Assert(bcObject != null, "BCObject missing!");
      return bcObject;
    }
  }
}
```

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;

namespace BlueComic
{
  /// <summary>
  /// Implementation of Web GET request HTTP client.
  /// </summary>
  public class Web
  {
    private IRoutineScheduler scheduler;

    public Web(IRoutineScheduler scheduler)
    {
      this.scheduler = scheduler;
    }
    /// <summary>
    /// GET request to the supplied URL.
    /// </summary>
    /// <param name="url">Request URL.</param>
    /// <param name="callback">Callback for response delivery.</param>
    public void Get(string url, Action<string> callback)
    {
      UnityWebRequest www = UnityWebRequest.Get(url);
      scheduler.Schedule(Send(www, callback));
    }

    /// <summary>
    /// GET request to the supplied URL using the supplied params.
    /// </summary>
    /// <param name="url">Reuest URL</param>
    /// <param name="parameters">Request params.</param>
    /// <param name="callback">Callback for response delivery.</param>
    public void Get(string url, IDictionary<string, string> parameters,
Action<string> callback)
    {
      url += "?";
      foreach (var param in parameters)
      {
        url += string.Format("{0}={1}&", param.Key, param.Value);
      }

      Get(url, callback);
    }

    private static IEnumerator Send(UnityWebRequest www, Action<string> callback)
    {
      yield return www.Send();

      if (www.isError)
      {
        Debug.LogErrorFormat("Web request error: {0}", www.error);
      }
      else
      {
        callback(www.downloadHandler.text);
      }
    }
  }
}
```