Holo-box: Multi-Level Augmented Reality Glanceable Interfaces for Machine Shop Guidance using Eye and Hand Tracking Interactions

Grigorios Daskalogrigorakis

Masters Thesis
School of Electrical and Computer Engineering
Technical University of Crete
June 2022

Supervisor: Prof. Mania Katerina, Technical University of Crete
Committee: Assoc. Prof. McNamara Ann, Texas A&M University
Committee: Assoc. Prof. Samoladas Vasilis, Technical University of Crete

Abstract

In this work, we employ Serious Games and Augmented Reality as tools for manufacturing training inside a machine shop. First, we analyze the progress of modern manufacturing processes leading up to the 4th industrial revolution, commonly known as Industry 4.0. We tackle shortcomings in relation to the use of simulations and Serious Games as training tools as well as the use of Augmented Reality in manufacturing. Finally, we analyze Glanceable Augmented Reality and its technical challenges.

In the first part of this work, we propose a gamified approach to visualising manufacturing missions for the implementation of a Serious Game focused on G-code programming for milling and turning missions for undergraduate students as a standalone training system without the need of a supervisor. Our gamified manufacturing mission design was also translated for use through AR inside the real world machine shop.

Glanceable User Interfaces for Augmented Reality (AR) reveal virtual content "at a glance" to provide rapid information retrieval, often based on gaze interaction. They are ideal when the augmented content covers a small proportion of the view space, but when the size of virtual content grows, the potential to occlude the real-world increases provoking safety concerns.

In this work, we present *Holo-Box*, a novel interaction system for AR combining Glanceable interfaces and world-based 3D interfaces across three Levels-Of-Detail, including progressively more information and visuals. Holo-box uses a combination of eye-gaze and hand interactions, focusing on user safety. A 2D Glanceable interface facilitates rapid information retrieval at a glance, while extended 3D interfaces provide interaction with denser content and 3D objects. Holo-Box couples blink-based and gaze-based interactions to minimize errors arising from the Midas Touch Problem. While applicable across domains, the Holo-Box interface is designed and optimized for performing manufacturing tasks in the real world. We evaluated the Holo-Box interface using an object selection task of a manufacturing process. Participants completed subsequent tasks faster using Holo-Box, employing incrementally smaller LOD interfaces over time. The perceived accuracy of Holo-Box gaze-based inputs was high, even when the device's eye tracker accuracy was coarse.

1 Introduction

Training new personnel for manufacturing is often a tedious operation. Practice under real-world machinery is expensive, consuming vital system's resources and, in some cases, dangerous for the trainee [4]. G-code programs guide the machining process, written in a computer or machine controller and loaded to a machine. A safer alternative for training uses simulated factories or training environments. The concept of a digital factory can be defined as an interactive 3D environment supporting modeling, communications and operation of manufacturing [5]. Previous studies indicate that virtual training is effective as the trainee is guided through complex product manufacturing [6], [7]. While there are plenty of gamified virtual simulations that aid in manufacturing training and mainly focus on safety tasks [8] or manual operation [9], the task of creating a serious game focused on writing G-code is technically challenging. Potential trainees often have no previous programming knowledge. It is significant that they learn how to write efficient G-code programs, not just correct ones.

Augmented Reality (AR) provides an enhanced vision of the world by seamlessly integrating virtual computer-generated elements with real-world environments [10]. AR applications are prominent in complex, innovative workplaces. The manufacturing floors of modern machine shops are increasingly employing AR technology as the evolution of the industry moves towards the *industry 4.0 model* [11]. A limiting factor of AR Head-Mounted Displays (HMDs) stems from the narrow Field of View (FoV) available in current hardware. This narrow FoV restricts the amount of virtual content that can be viewed at one time which, in turn, occludes real-world elements, critical to the task at hand or the user's safety.

Workers employ AR to safely retrieve virtual information at any point eliminating the distraction and spatial limitations of traditional 2D screens[12]. AR applications need compelling, intuitive interfaces that do not impede or endanger the user to be effective in this space [13]. One approach is to use *Glanceable User Interfaces* that can retrieve information "at a glance" based on gaze detection when using an AR HMD with integrated eye-tracking [14]. Glanceable interfaces prove helpful in cases when the user is focused on a real-world task that directly requires keen attention, providing appropriate guidance without obstruction [15]. Glanceable interfaces are quick and convenient, but they suffer from three main drawbacks: (1) Limited Scalability: As the amount of AR content increases, Glanceable interfaces lose effectiveness due to the volume of data to be displayed and the potential to block real-world content [14][16]. (2) Viewing Angle: Problems can arise when viewing augmented content at an angle [17]. (3) Unintentional selection/interactions using gaze: This is the Midas Touch Problem (MTP) [18].

A Glanceable interface should be compact, enabling minimal information at a glance to avoid clutter, occlusion, and distraction [3]. When a large amount of data is necessary to guide the user through the task, AR elements occupy a large portion of the available visible space. Interactions with virtual content become less accurate and in workplaces such as machine shops, obstruction of the visual field is dangerous. Voice and gesture commands have been offered as an alternative to visual instruction. In a machine shop, however, AR interaction based on speech is not well-suited due to the high levels of noise, while mid-air hand gestures and wide controller motions can pose safety concerns [19].

In the first part of our work, we present an innovative desktop-based CNC machining training procedure, created as a serious game for CNC manufacturing training. The proposed serious game prepares the trainee in writing G-code and setting up virtual machines for completing milling and turning tasks. The serious game works as a standalone training system without the need for a supervisor or trainer to be present. The serious fame is focused on G-code training and basic machine setup as communicated by an engaging 3D environment and a streamlined training process. The serious game was developed in collaboration with the Micromachining and Manufacturing Modeling Lab (M3 lab) of the School of Production Engineering an Management of the Technical University of Crete. Technical development of the Serious game was accomplished by Grigorios Daskalogrigorakis and Salva Kirakosian. The M3 lab provided the 3D object models, 2D documentation in addition to theoretical guidance regarding manufacturing processes and manufacturing education. Salva Kirakosian was responsible for 2D and 3D design for the 3D virtual machine shop, while Grigorios Daskalogrigorakis was the gameplay designer and developer regarding the Gamification of the manufacturing process.

In the second part, we present Holo-box, a novel AR interface that combines Glanceable AR and world-based 3D interfaces employing a combination of eye-gaze and hand interactions. We

evaluate this in a machine shop taking workplace safety considerations into account, boosting productivity and accuracy in a manufacturing process. Holo-box utilizes a 3-Level-of-Detail (3-LOD) architecture providing varying levels of information to the user while performing tasks in the machine shop. Levels can be expanded at will, through glance, to reveal additional information (see Fig. 1). The Glanceable interfaces enable rapid information retrieval with minimal real-world occlusion. Incorporating an AR 3D interface enabled with hand-tracking allows for complex, accurate interactions in a controlled environment. Holo-box uses custom eye interaction logic to automate accurate and natural direct eye interaction through blinking. The interface is deliberately designed to accurately decode gaze, even when the accuracy of the eye tracker is low.

Figure 1: The proposed 3-LOD AR interface system.

Our work has the following contributions:

- We present Holo-Box, a novel 3-LOD interface system for AR combining both Glanceable and 3D interfaces [3]. Two initial LOD interfaces are controlled through glance (Glanceable interfaces), while the deepest level of LOD is a 3D world-based interface. The interface enables the user to view simple guidance information through the Glanceable interfaces or select to view the 3D interface to examine more detailed information.

- Holo-Box utilises a novel interaction system that utilizes a combination of eye-gaze blinking with a delay (dubbed blink-delay) for interacting with Glanceable interfaces to minimize unintended interactions. Objects remain selected and interactable even if the eye cursor exits their activation trigger unless another interactable object is selected. Glanceable interfaces are also visually adjusted to minimize errors in interaction, deliberately placing buttons on opposite sides of the interface. The 3D interfaces also allow hand tracking interactions with a delay (dubbed hand-delay). Users can engage interchangeably between hand-delay and blink-delay interactions; thus, interactable objects can be placed on any layout without restrictions [1].

- In the Machine shop Serious Game, we present a streamlined gamified presentation for manufacturing missions for milling and turning tasks [2]. The Serious Game was used for G-code education and received positive feedback from students. In addition, our proposed gamified presentation was adapted to AR for use in Holo-box to provide feedback on the real world [3]. The content shown in each LOD is adjusted so that each interface can be used to guide users with different levels of knowledge on its own. In the most compact Glanceable interface, we show only textual information, including the name and material of a given task. The second LOD level of the Glanceable interface offers additional text and 2D images. The subsequent 3D interface includes 3D objects and multiple interactable buttons. The Glanceable interfaces provide adequate guidance for experienced users, while the 3D interface provides additional information for inexperienced users, using 3D models relevant, in our case, to manufacturing tasks.

- We evaluated our work in Holo-box two times. In our first evaluation we found the issues of Holo-box and the limitations of our hardware and tracking, which were corrected for our final implementation [3]. In our second evaluation, we evaluate Holo-box by employing an object selection task in a pilot manufacturing scenario. The content displayed on all interfaces is used in presenting milling and turning manufacturing tasks in the form of gamified missions. Experiment results show that users complete missions faster and use more compact interfaces as they gain more experience. Our proposed interaction system compensates for the eye

tracker's inaccuracies resulting in higher perceived accuracy on blink-delay inputs than hand-delay.

Chapter 2 includes a low level analysis of Augmented reality including theoretical knowledge, analysis of the most well known AR Head-Mounted Displays (HMDs) and the various ways to develop AR applications. In addition, chapter 2 also includes a brief analysis of real world Machine shop manufacturing, the new Industry 4.0 model and the necessity in including AR into the manufacturing workspace.

Chapter 3 provides an analysis of previous work including the use of gamified environments in education and/or training and the use of virtual environments and AR in workspace education. Chapter 3 also covers related work regarding AR interface types used in multiple scenarios, including adaptive and Glanceable interfaces.

Chapter 4 covers the implementation of a Serious Game for manufacturing education in which we designed a gamified environment for machine shop education, which served as the core of our AR application. The Serious Game was developed for remote G-code education, without the need for a supervisor. In this game, we designed a gamified manufacturing mission presentation, which was also adapted for use in the real world using AR.

Chapter 5 covers the first part of our AR implementation which includes our proposed 3-LOD interface system. In this chapter, we discovered the issues and limitations of our first attempt, and we analyze the way to overcome them.

Chapter 6 covers the full implementation of our work, which extends the previous chapter and solves the issues present in the prototype with Chapter 7 analysing our work in a low level inside the Unity Engine. Chapter 8 covers the user study used to evaluate our work.

5 Holo-Box: Level-of-Detail Glanceable Interfaces for Augmented Reality

In this chapter we analyze the initial implementation of our work. The main motivation for this work was to combine the gamified mission format presented in G-code Machina [2] with Glanceable AR interfaces [14] in order to provide realtime guidance inside the manufacturing workspace using AR in a streamlined gamified manner.

Although this work had positive feedback when published [3] its usability was limited by issues regarding the Magic Leap's eye tracking accuracy. These issues were identified and corrected during the course of this work for the final implementation.

5.1 3-LOD architecture

Holo-box employs a three Level of Detail (LOD) interface combination, with each LOD revealing varying levels of information with varying levels of real world occlusion. Two of the three LODs are 2D display-fixed interfaces which are always visible at a fixed position of the user's visual field and use eye tracking for interactions with virtual objects. The third interface is a 3D world-based interface which receives hand-based inputs instead.

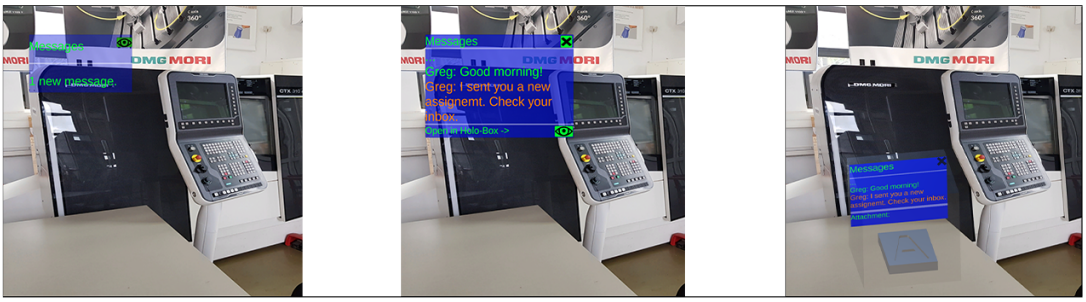


Figure 75: Holo-Box's 3 Levels of Detail interfaces shown in the context of a machine shop. (left) Simple Glanceable. (middle) Detailed Glanceable. (right) Holo-Box.

Initially, the user is presented with the first LOD interface and the usability of the 3-LOD system is as follows:

Level 1: Simple Glanceable interfaces (SG) (Fig.75 left), are visible by default while the user is focused on the real world, are compact and semi-transparent. Their purpose is to alert users to changes of the virtual content. Interaction is initiated by gaze-activated virtual buttons, using a small gaze-dwell delay. This short gaze-dwell initiates a transition to a Detailed Glanceable (DG) interface.

Level 2: Detailed Glanceable interfaces (DG) (Fig.75 center), provide more complex information compared to SG UIs such as lines of text or images. DG UIs are less transparent than SG. DG elements do not cover the entire FoV allowing for real-world spatial awareness. Using gaze-dwell as input, the user can revert back to the SG interface or progress to the more detailed Holo-box mode.

Level 3: Holo-boxes (HB) (Fig.75 right), are 3D interfaces summoned at will or established at predetermined locations. The HB is an isolated space where the user can freely interact with virtual content using direct input without environmental occlusion or safety hazards. HB interfaces include both 2D and 3D content scaled to the size of the HB itself and to the user's available space.

5.2 Machine Shop context

Holo-box is applied in the context of a machine shop. Users wear a Magic Leap while operating milling or turning machinery accessing virtual content through the Holo-box's UI. Augmented content may vary from notifications to visualizing complex 3D objects for manufacturing. Virtual interfaces use the presented three-tier non-intrusive LOD architecture while the user's focus is on the real world. Milling and turning tasks are presented gamified as "missions" consisting of a 3D or 2D object representation to manufacture as well as a textual description of the manufacturing

process [2]. By using Holo-box's architecture, missions are presented in a gamified manner as follows:

Figure 76: Simple Glanceable prototype

Level 1: SG (see Fig. 76) Notification that a new mission is available which may pop up at any time while the user is working. The "eye" button transitions to the DG interface.

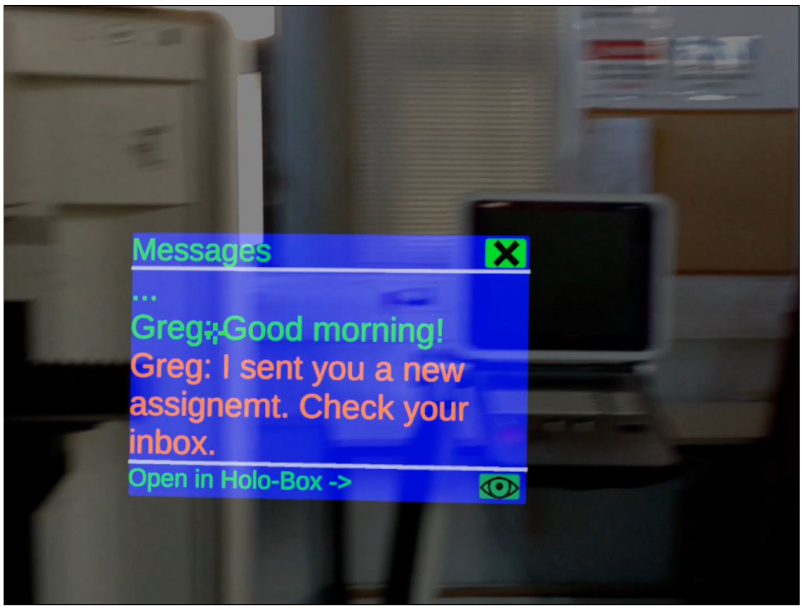


Figure 77: Detailed Glanceable prototype

Level 2: DG (see Fig. 77) Simple textual description of the mission, which is shown after pressing a button on the SG interface. The placeholder text will be replaced by a textual description of the given task. The "eye" button transitions to the HB interface. The "X" button reverts to the SG interface.

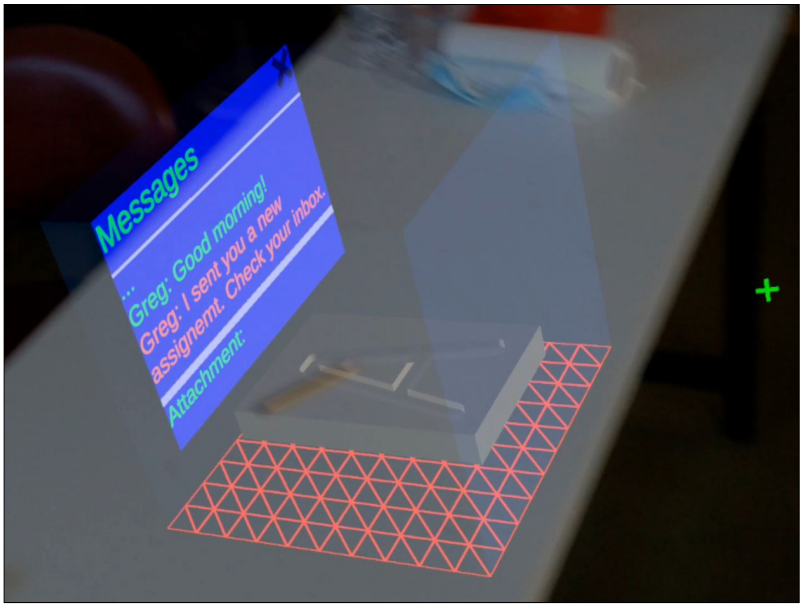


Figure 78: Holo-Box prototype

Level 3: HB (see Fig. 78) Detailed textual description of the request and an interactable 3D model of the requested object to manufacture, shown on demand after pressing a button on the DG interface at a Holo-box placed on the 3D space. The placeholder text is the same as the DG interface. By interacting with the 3D object by hovering their index finger over the 3D model the user can view additional information including measurements of the finished product as seen in the mechanical blueprint of the object (see Fig. 61).

5.3 Issues and limitations

During the implementation of this prototype we found certain issues or limitations in our proposed interface system, which we aimed to resolve during the course of our final implementation. These issues included the following:

Inaccurate eye tracking. For our implementation we utilised the Magic Leap One's on board eye tracker. The eye tracker calculates an estimated fixation point in 3D space, based on triangulating both eyes according to the Magic Leap's initial calibration data, in addition to a confidence value that grades the accuracy of the readings. In Unity, the fixation point is shown as a 3D point in space which changes values only when a new fixation point is calculated with a high confidence value, otherwise the fixation point remains stationary on its previous value. We translated this 3D point into a 2D cursor which only moved horizontally and vertically towards the fixation point and interacted with objects behind it regardless of depth. In addition, the cursor's movement was smoothed to follow the fixation point at a fixed speed in order to avoid sudden jumps and rapid flickering (as seen in Fig. 76 where the blue circle represents the fixation point and the green cross the cursor moving towards it).

In practice, the eye tracker was often unable to track the user's eye effectively resulting in either the cursor freezing in place making the user unable to interact with interfaces or the eye tracker detected false eye movements and moved onto interactable objects resulting in false interactions.

In order to solve this, we used the Magic Leap Control for two functions. First, the user could use the onboard trackpad to introduce a small offset from their fixation point by manually moving the target when the tracking failed. Second, we changed our passive gaze-delay interactions, where if the cursor remained on top of an interactable object for 1 second it would invoke an interaction, with an active interaction by pressing a controller button.

Through this change we understood that the eye tracking cursor should have some secondary form of movement to enhance its accuracy, and gaze-based interactions should be done through a form of direct interaction instead of happening passively under specific conditions.

For the final implementation, the Glanceable interfaces were unlocked from the user's field of view and instead were fixed to the user's surrounds at a set angle and a set distance. This way the user can use a combination of eye tracking and head movement to move the gaze-based cursor

in relation to the Glanceable interfaces. In addition, the Magic Leap's eye tracker also detects whether the user is blinking or not with one value for each eye. Thus, we replaced the clicking of a button on the controller with the blinking of one eye, which resulted in a direct form of interaction which only uses the user's eyes.

Confusing interaction switch between interfaces. As mentioned before, users were shown how to use gaze-based interactions with a delay for gaze-based interfaces, while the Holo-box used instantaneous hand-based interactions, which proved to be disorienting.

In order to solve this, for the final implementation both interaction types were combined into a universal system. Both forms of interaction used separate cursors, one for the eye-gaze and two for each index finger and an interaction occurred when the user either started blinking with the eye-gaze cursor on top of an object or when a hand cursor entered an object's trigger zone. Both actions performed an interaction with a small delay to minimise misinteractions and hand-based interactions took priority over gaze-based interactions if both occurred simultaneously to minimise conflicting cases. As a safety measure, hand-based interactions only work inside the Holo-box interfaces and not on Glanceable.

SG LOD is underused. The initial concept for the SG interface was to show a generic message notifying there was a change inside the AR interfaces, which provided no useful information without opening at least one additional interface.

For the finished implementation this was changed to show an informative message that was limited to a few words. The message informed the user of a new manufacturing task and included the name of the product to manufacture and the material to use (e.g. "12mm cog, copper"). Additional information on the given task are received through subsequent interfaces, but this amount of information could be enough to inform an experienced user that has manufactured the same object before and memorised it.

Additional machine shop information was needed. Even with the proposed 3-LOD interface system, the information given to the user was only relevant to the dynamic data of the given task. In the previous work of G-code Machina [2] the users first received a manufacturing task and then read the relevant documentation to gain additional knowledge required to complete a mission.

In our work, users did not need to read documentation regarding the G-code, but they had to find additional objects to complete their given task, such as manufacturing tools. Thus, the Holo-box was extended to have additional content than included a visualised documentation in addition to the given mission's description. This documentation could also include 3D objects which could be directly compared to their real world equivalents.

6 Holo-Box: Multi-LOD Glanceable Interfaces for Machine Shop Guidance using Blink and Hand Interaction

In this chapter we analyze the final design of our proposed Glanceable interface system Holo box developed using Unity 2020 and the Magic Leap One AR Head-mounted Display (HMD).

The contribution of this work is threefold:

- Holo-box is a novel, gaze-activated 3-LOD system that initially launches with a compact interface and progressively expands to reveal more information with each LOD. Two initial LOD interfaces are controlled through glance (Glanceable interfaces), while the deepest level of LOD is a 3D world-based interface. The interface enables the user to view simple guidance information through the Glanceable interfaces or select to view the 3D interface to examine more detailed information.

- Holo-box utilises a new interaction system that utilizes a combination of eye-gaze blinking with a delay (dubbed blink-delay) for interacting with Glanceable interfaces to minimize unintended interactions. Objects remain selected and interactable even if the eye cursor exits their activation trigger unless another interactable object is selected. Glanceable interfaces are also visually adjusted to minimize errors in interaction, deliberately placing buttons on opposite sides of the interface. The 3D interfaces also allow hand tracking interactions with a delay (dubbed hand-delay). Users can engage interchangeably between hand-delay and blink-delay interactions; thus, interactable objects can be placed on any layout without restrictions.

- The content shown in each LOD is adjusted so that each interface can be used to guide users with different levels of knowledge on its own. In the most compact Glanceable interface, we show only textual information, including the name and material of a given task. The second LOD level of the Glanceable interface offers additional text and 2D images. The subsequent 3D interface includes 3D objects and multiple interactable buttons. The Glanceable interfaces provide adequate guidance for experienced users, while the 3D interface provides additional information for inexperienced users, using 3D models relevant, in our case, to manufacturing tasks.

We evaluate Holo-box by employing an object selection task in a pilot manufacturing scenario. The content displayed on all interfaces is used in presenting milling and turning manufacturing tasks in the form of gamified missions. Experiment results show that users complete missions faster and use more compact interfaces as they gain more experience. Our proposed interaction system compensates for the eye tracker's inaccuracies resulting in higher perceived accuracy on blink-delay inputs than hand-delay.

The implementation can be separated into three chapters: (1) The interface design, (2) User interactions, and finally, (3) Adapting the content of the interfaces to take full advantage of the designed interface system.

Our proposed design aims to achieve three primary design goals (DG) that include:

- *(DG1) The system must be usable while working on a real world task* while minimizing potential safety risks, improving on information retrieval and safety practices of previous work in AR guidance in the manufacturing workspace [68].

- *(DG2) The interface must be able to show varying amounts of information in different LODs* based on whether the user's focus is on the real world or the virtual content. The user's focus is governed by direct interactions with virtual objects instead of automatically [17], reducing potential distractions from shifting interfaces while the user's focus is on a real-world task.

- *(DG3) The interfaces should achieve accurate interactions and the Midas Touch problem should be minimized when using gaze interaction.* To mitigate the impact of the Midas touch problem, a direct form of interaction, such as blinking, is preferred[73]. In dense interfaces where interactable objects are prone to overlap [71], a different interaction paradigm should be used such as hand-tracking. Speech control is not appropriate due to the noisy nature of the manufacturing workspace.

6.1 Interface design

Figure 79: Holo box. 3-LOD interface design. Level 1: Simple Glanceable - SG (Left), Level 2: Detailed Glanceable - DG (Middle), Level 3: Holo-Box - HB (Right)

Holo-box employs the same 3-LODs (e.g. Simple Glanceable (SG), Detailed Glanceable (DG) and Holo-Box (HB). See Fig. 79) mentioned in the prototype design, but the usability of each interface has changed to adapt to the issues and limitations mentioned in the previous chapter as well as the Design Goals mentioned in this chapter.

SG and DG are 2D display-fixed Glanceable interfaces that follow the user's head orientation and are always visible. HB is a 3D interactive interface which remains fixed at a predefined anchor location in the real world chosen by the user. The usability of each interface is as follows:

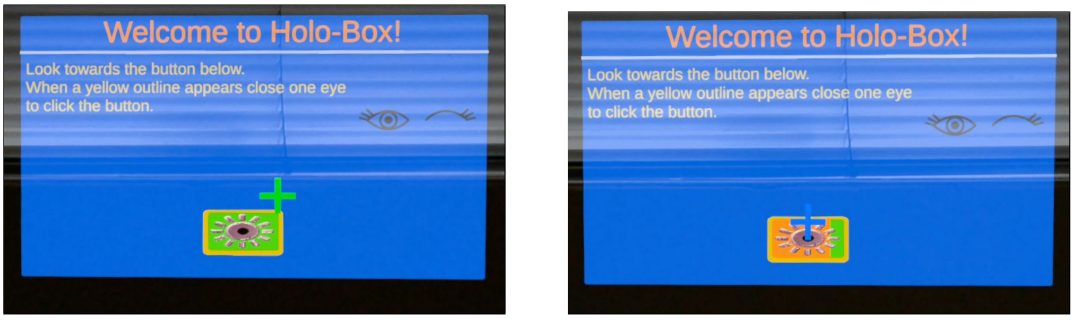
Simple Glanceable (SG): (Figure 79 left) The SG interface is a compact 2D interface that shows minimal information limited to a few words. SG minimizes real-world occlusion and is the first interface encountered to notify the user new data is present in the virtual interfaces. The SG interfaces include an interactable button that transitions to the DG interface.

Detailed Glanceable (DG): (Figure 79 middle) The DG interface provides more detailed information to the user, including a few lines of text as well as 2D media such as images. DG is larger than the SG interface, but it is still relatively small compared to the Magic Leap's FoV to not impede a large area of the available FOV. DG includes two interactable buttons, one to return to the SG and another that transitions to the HB. Locating the two buttons far from each other minimizes potential unintentional interactions.

Holo-Box (HB): (Figure 79 right) The HB interface is a 3D interface that appears in the real world. During the application setup process, the user manually places the HB on the real world, designated as a semi-transparent bounding box. Its position persists during the execution of the application or until the user manually repositions it. Upon HB activation through the DG interface, the content appears inside the predetermined position regardless of the user's position during the activation. The content of the HB includes complex 2D interfaces, 3D objects and any number of interactable objects. HB also allows for both blink-delay and hand-delay interactions; these can activate interchangeably.

The two Glanceable interfaces allow for rapid information retrieval "at a glance" with two levels of information. SG is compact, allowing for easier focus on the real world providing guidance, while DG triggers after the user interacts with the SG interface. Thus, we can assume the user wants to focus on the virtual content when the DG interface appears. SG and DG interfaces follow the user at a set distance and viewing angle as they move around the real world. On the other hand, HB is anchored on a specific real-world position and does not move. HB content is presented only on the inside of its bounding box. We assume that the user has placed it appropriately at a position where they can use hand tracking without risk of injury. This design fulfills DG1 and DG2.

6.2 User Interactions



(a) (b)

Figure 80: Eye tracking interaction. When one eye is closed the cursor turns blue and initiates the click.

Holo Box allows for two types of interaction, one that uses eye tracking (see Fig. 80) and one that uses hand tracking (see Fig. 81). Both interaction types are designed around using delay-based inputs, where the user targets an object using a cursor and remains on the target for a set amount of time before the interaction happens. To reduce gaze mis-interactions, the user must place the eye-tracking cursor and then blink with one eye to interact, thus, using "Blink-Delay" interactions.Hand tracking interactions on the other hand use different cursors which are placed on the users' index fingers on both hands.The user interacts with objects by positioning their fingers in a trigger zone which extends a few centimeters around the interactable object and remain inside the trigger for the same delay period for a "Hand-Delay" input. To avoid conflicting simultaneous interactions, hand tracking is prioritized over eye-tracking interaction. If both hands engage simultaneously, then the left-hand cursor is prioritized.

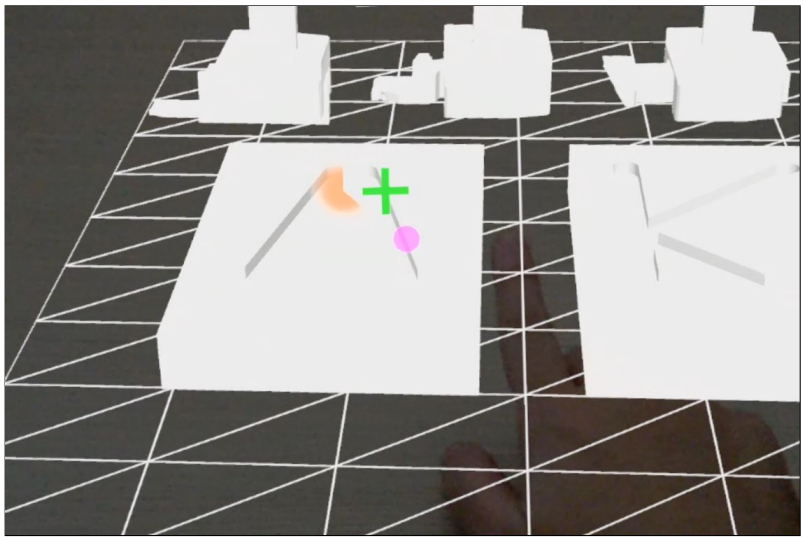


Figure 81: Right hand cursor (purple circle). The user's hand appears to the right of the cursor due to the Magic Leap camera's recording perspective.

Interactable objects in Holo-box operate using a universal state machine. An interactable object can be at one of three states: Idle, Highlighted or Clicked (see Fig. 82). Initially, all objects are in an idle state, showing their basic visuals. When a cursor enters the object's trigger zone, that object becomes highlighted, enabling an additional visual effect to show the transition. After a click is initiated by blinking on a Blink-Delayed input or immediately when highlighted on Hand-Delay inputs, a progressively filling visual effect is presented, showing the progress towards the interaction completion. After the delay period has passed, the click effect is triggered and the object returns to an idle state.

Figure 82: Button States

In Holo-box, to fulfill DG3, only one object can be at the highlighted and clicked states at all times shared across all interaction types. The highlighted object is deselected only after a click, or another object becomes highlighted and not when the user exits the trigger zone. When using Hand-Delay, the object changes from clicked to highlighted on exit. This way the user can select an object and even if their cursor moves outside the trigger zone due to tracking errors, the delayed click can still be performed by blinking if the cursor does not select a different object.

To fulfill design goal DG1, Hand-Delay interactions are disabled on the SG and DG interfaces, as these interfaces move and may be placed in a dangerous location for hand motions. Based on the design,the SG and DG interfaces are designed to minimize misinteractions. SG only comprises a single interactable button. DG includes two buttons placed on the top and bottom edges of the interface to reduce misinteractions when using Blink-Delay inputs. HB allows for both Hand-Delay and Blink-Delay inputs. As such, there are no restrictions on the amount of interactable objects as long as they are placed so that their trigger zones are not overlapping.

6.3 Machine shop Content Adaptation

A set of Glanceable interfaces were designed for this work, including information guiding the user to perform the experimental procedure. In the experiment, the users are given a set of instructions and they are tasked with selecting the correct object corresponding to a manufacturing task, including a **cutting tool**, a **cutting material** and a **manufactured product**.

For the Glanceable interfaces, each manufacturing task is presented in the form of a gamified "mission" [2]. A mission consists of a set of information required to perform a manufacturing task setup. This consists of which tool and material to use, a 2D blueprint of the finished product and numerical cutting parameters such as the cutting speed. Each mission has a predefined tool, finished product and manufacturing parameters, but the material can vary.

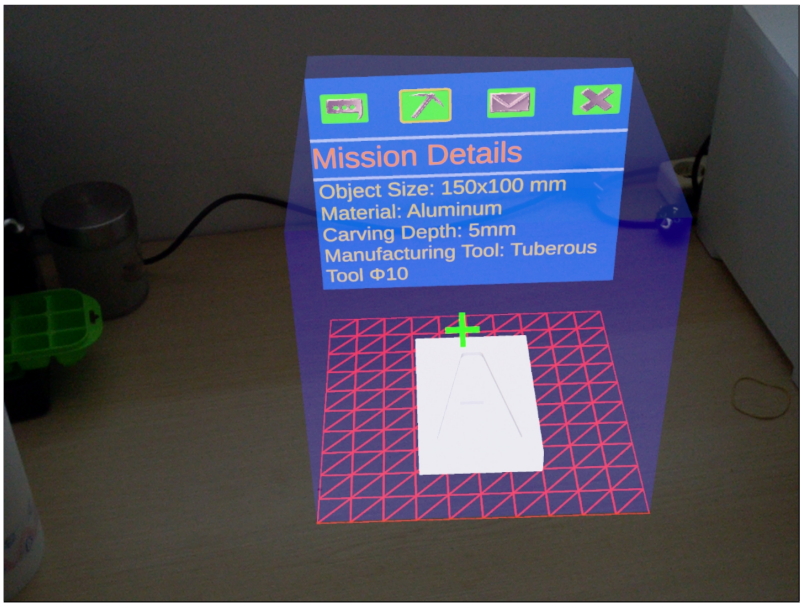
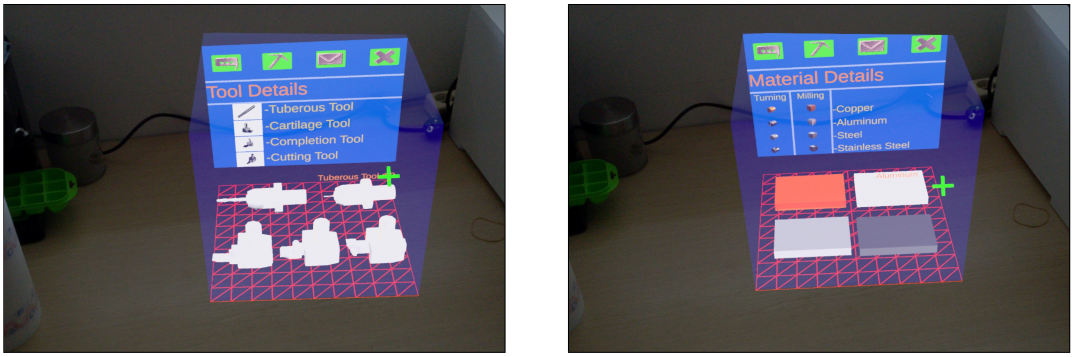


Figure 83: The Holo-box interface mission description.

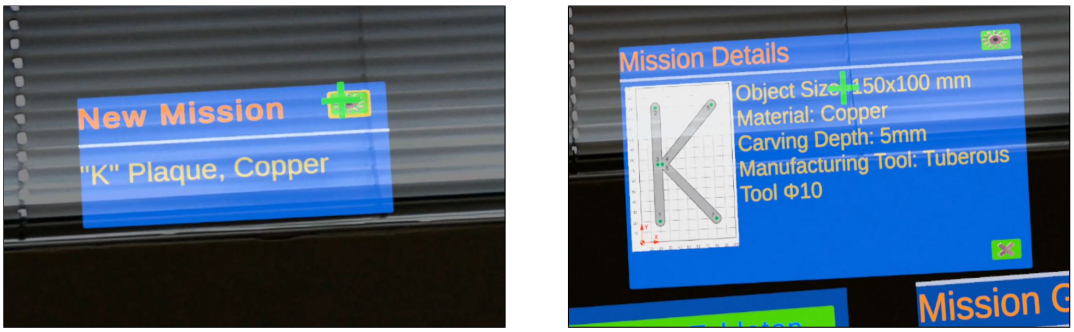
The presentation of the missions is adapted to allow efficient guidance with any of the three interfaces according to the user's expertise. The HB interface contains the information required to complete any mission. This includes mission information (see Fig. 83) as well as tool (see Fig. 84a) and material (see Fig. 84b) look-up tables with added 3D objects for better visualisation. The 3D models of the tools and materials use a modified version of the interaction logic that shows the object's name above the model when highlighted.



(a) (b)

Figure 84: Holo-Box interfaces. (a) Tools database, (b) Materials Database

The DG interface includes mission-specific information in a 2D interface (see Fig. 85b). The SG interface only contains the name of the mission as well as the correct material (see Fig. 85a). The SG interface is adequate for experienced users who have already completed a given mission and memorized it. To make the SG interface more efficient, each mission was given a different name, e.g., the "A plaque" corresponds to cutting the letter "A" on a square block.



(a) (b)

Figure 85: Simple Glanceable and Detailed Glanceable interfaces

6.4 Evaluation interface

The Evaluation Interface (EI) used in the evaluation is a modified version of the HB with the same functionality. This interface includes 2D interfaces with buttons used to operate the evaluation procedure, e.g., starting the evaluation or moving to the next mission providing feedback to the user (Figure 86). In addition, the interface includes 3D models of all available machining tools, materials and finished products which the user can select by interacting with them to complete a given training mission.

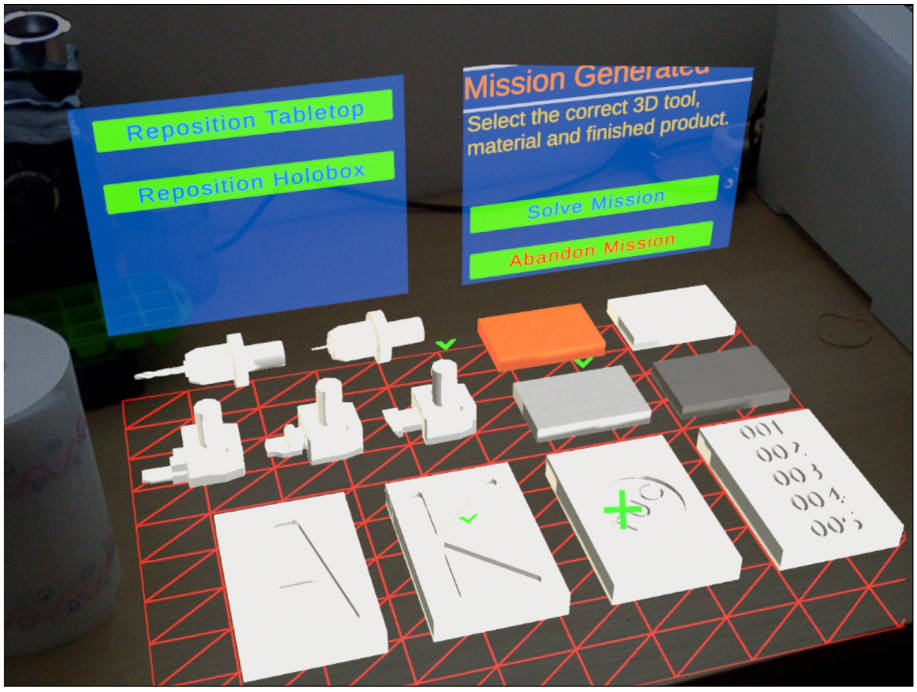


Figure 86: Evaluation Interface

7 Low level Unity Implementation

In this chapter we will analyze the implementation of our work at a low level. Our implementation was done exclusively through Unity, which was linked with Magic Leap's "The Lab" application for zero iteration debugging and a single click build and install.

7.1 Scene Hierarchy

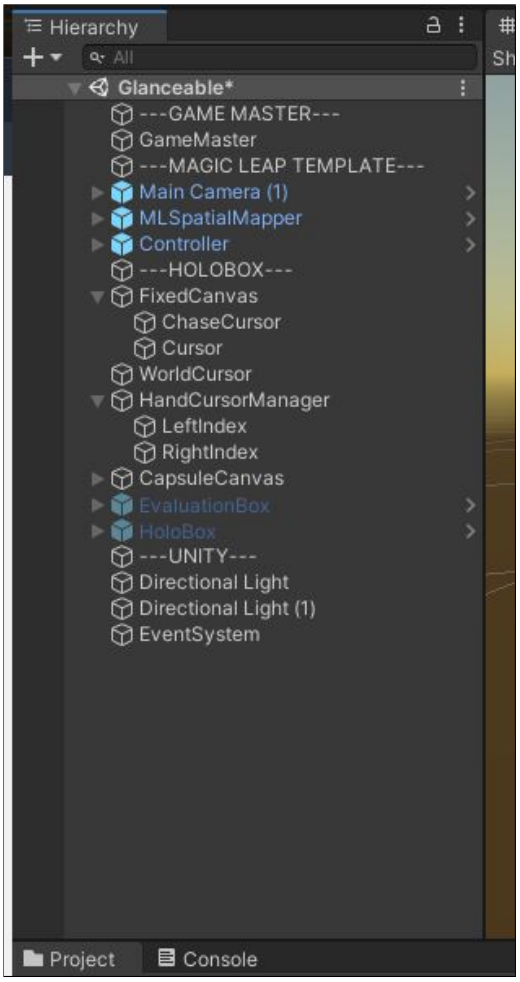


Figure 87: The implementation's scene hierarchy

Our implementation was done in a single virtual scene, dubbed Glanceable (see Fig. 87) as an internal name. The game objects inside our scene can be categorised in four categories:

- Magic Leap Template objects. These are standard pre-made object prefabs included in the Magic Leap Unity Template, which link our project with the Magic Leap One hardware appropriately.
- Unity standard objects. These are standard Unity objects present in any scene.
- The Game master. This game object holds the core logic of our work including global parameters and calls from and to other scripts.
- Holo-Box's game objects. These objects and their logic were fully implemented during this work and represent actual virtual objects inside the scene.

7.1.1 Magic Leap Template objects

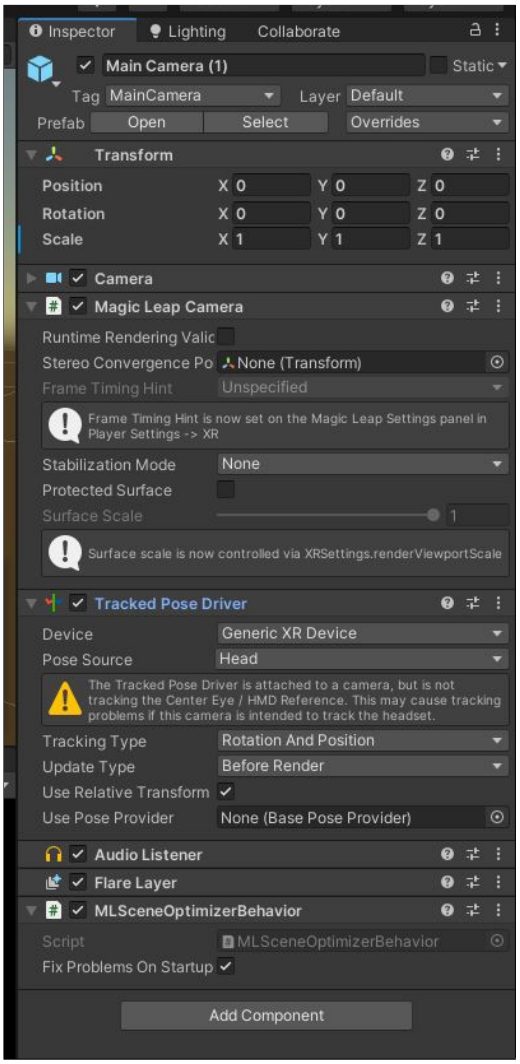


Figure 88: The Main Camera Object's inspector

Main Camera (see Fig. 88). The Main camera represents the user's eyes into the virtual scene. Through the *Magic Leap Camera*, *MLSceneOptimizerBehavior* and *Tracked Pose Driver* components the game object tracks the physical hardware and adapts virtual content as the user moves around the virtual scene to keep the real and virtual worlds aligned at all times.

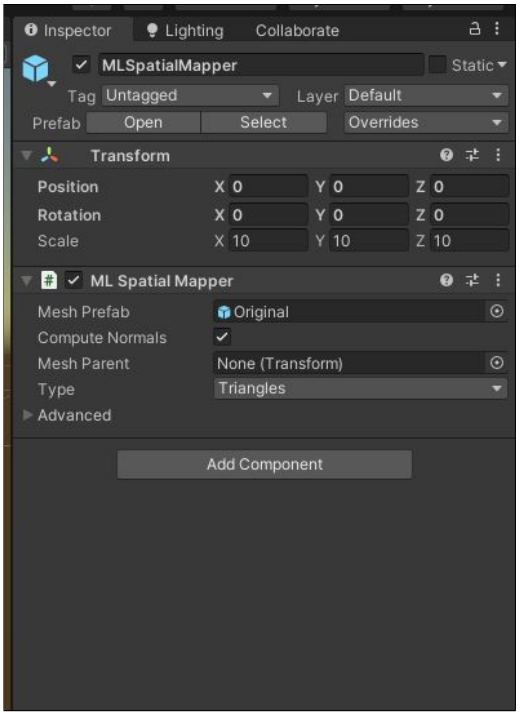


Figure 89: The ML Spatial Mapper Object's inspector

ML Spatial Mapper (see Fig. 89). This object is responsible for scanning the mesh of the real world and then creates a collection of polygonal meshes inside the virtual scene. These meshes allow us to visualise the scanned real world's shape and add occlusion and collision to these objects if necessary. The visuals and behavior of the scanned mesh objects are based on the *Mesh Prefab* field which specifies the game object to use as a template.

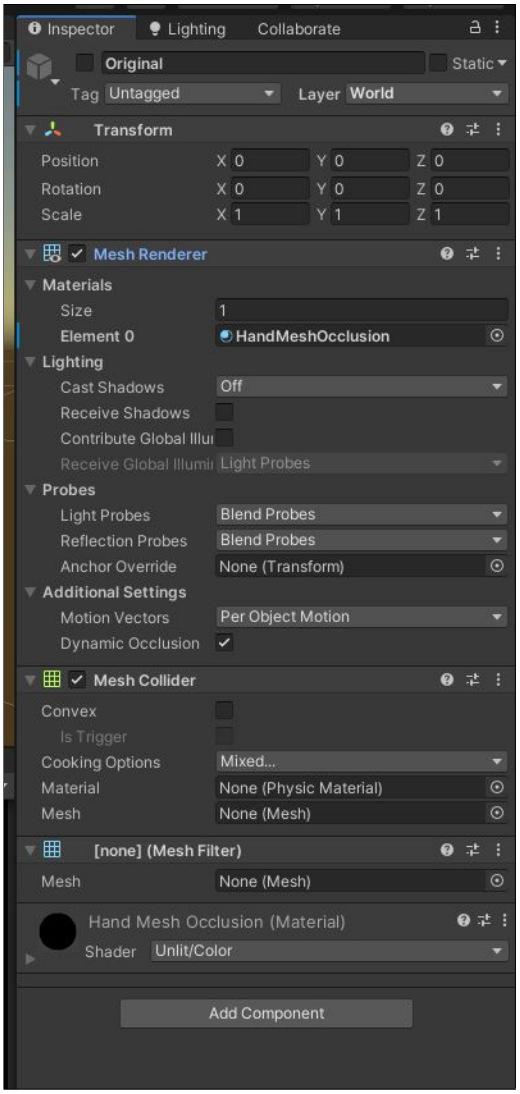


Figure 90: The ML Spatial Mapper's Mesh Template Object's inspector

The Mesh template object is provided by default along the Spatial mapper (see Fig. 90). By default, the mesh's objects are visualised as a multi-colored mesh visible to the users. In our case, we switched the default *Wireframe* material with the *HandMeshOcclusion* one, which has no visuals on its own, but occludes other objects behind it. This way, if a virtual interface intersects with a real object it will be partially occluded.

The object's mesh collider also allows our objects to be included in physics calculations, which in our case is necessary for the eye tracking, as the eye tracker calculates the user's fixation point by raycasting from the user's eyes towards the virtual world until it collides with a virtual object.

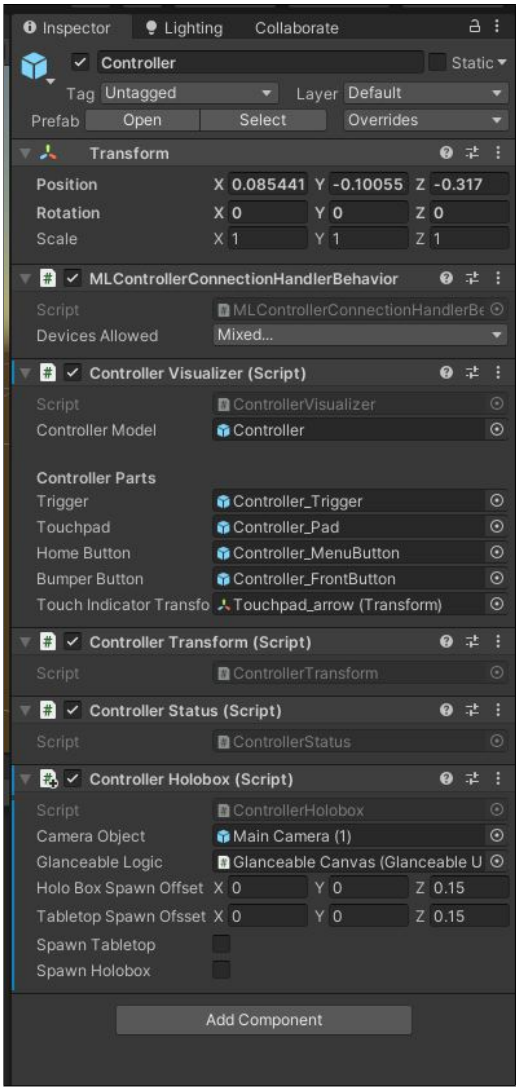


Figure 91: The Controller Object's inspector

Controller (see Fig. 91). The Controller is responsible for both tracking and visualising the Magic Leap Control. The *Controller Visualiser* script is responsible for visualising the controller as a virtual object inside the scene. The *Controller Transform*, *Controller Status* and *MLController-ConnectionHandlerBehavior* are responsible for tracking the controller's position. The *Controller Holobox* script allows us to read and use controller inputs in our application.

```
35      void Update()
36      {
37          CheckTrigger();
38      }
39
        1 reference
40      void OnButtonDown(byte controllerId, MLInput.Controller.Button button)
41      {
42          if(button == MLInput.Controller.Button.Bumper)
43          {
44              if (SpawnHolobox)
45              {
46                  GameObject box = Instantiate(GlanceableLogic.HoloBoxPrefab,
                      (transform.position + transform.forward * HoloBoxSpawnOffset.z
                      + transform.right * HoloBoxSpawnOffset.x + transform.up *
                      HoloBoxSpawnOffset.y), transform.rotation);
47                  GlanceableLogic.SetHoloBox(box);
48                  box.transform.GetChild(0).gameObject.SetActive(false);
49              }
50              else if (SpawnTabletop)
51              {
52                  GameObject box = Instantiate(GlanceableLogic.TabletopPrefab,
                      (transform.position + transform.forward * TabletopSpawnOfsset.z
                       + transform.right * TabletopSpawnOfsset.x + transform.up *
                      TabletopSpawnOfsset.y), transform.rotation);
53                  GlanceableLogic.SetTabletop(box);
54              }
55          }
56      }
57
        1 reference
58      void CheckTrigger()
59      {
60          if (controller.TriggerValue > 0.5f)
61          {
62              capsuleMove.SetGraphicsAngle(MainCamera.transform.eulerAngles.y);
63          }
64
65      }
```

Figure 92: The Controller Holobox script

The *Controller Holobox* script contains two key functions. The *Check Trigger* function is called on every frame and checks how far the trigger is pressed (0 = not pressed, 1 = fully pressed). If more than half the trigger is pressed, we rotate the *Capsule Canvas* which holds the Glanceable interfaces to match the user's forward angle. This way, if the user presses the trigger and then rotates their head they can reposition the Glanceable interfaces to their desired angle. The button and trigger values are received through the *MLController.Controller* class included in the *UnityEngine.XR.MagicLeap* package.

The *On Button Down* function is a callback method which is automatically executed by Unity every time a button press is detected from any input (including keyboard, mouse and all types of controllers when available). In our case, we only detect if the Magic Leap Control's bumper was pressed. The only times when we want to use this button are when we want to spawn the Evaluation Interface and the Holo-box, indicated by the *SpawnHolobox* and *SpawnTabletop* booleans.

In both cases, we Instantiate a new Game Object either of type *HoloBoxPrefab* or *TabletopPrefab*, both of which are initialized inside the *GlanceableLogic* script, which is a class of type *GlanceableUILogic* stored onto our *GlanceableCanvas* Game Object. The object is instantiated with the same position and rotation as the Magic Leap Control. Finally, the Glanceable logic's reference to the newly spawned object is updated, deleting a previous instance of the same interface if it existed previously and keeping the last one.

7.1.2 Unity Standard objects

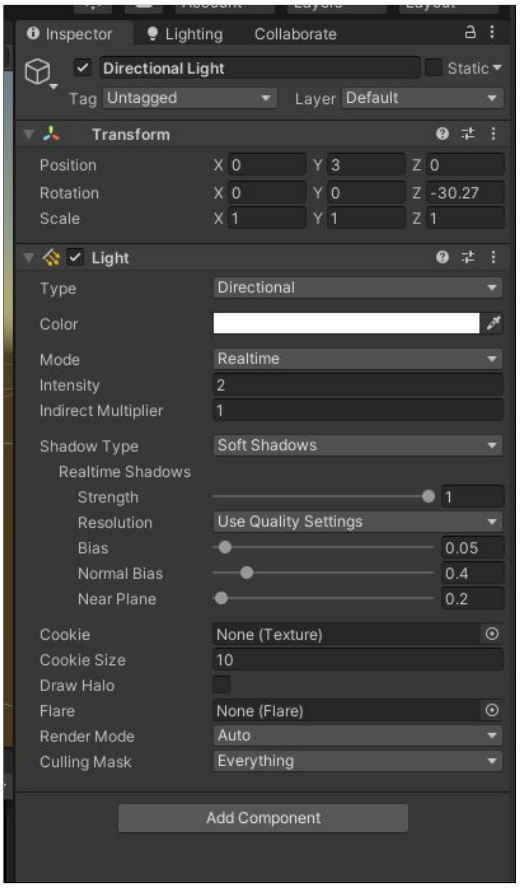


Figure 93: The Directional Light Object's inspector

Directional Light (see Fig. 93). The Directional light object provides a universal ambient lighting to all objects in the scene. This light simulates natural sunlight that lights every object in the scene from a set angle (identified by its rotation). In a standard 3D scene, this light would cast shadows after it collides with 3D objects and indirect light bounces would light up occluded areas with smaller intensity like the real world. In AR however, this process does not work as intended, as Magic Leap denotes lower intensity as object transparency, resulting in objects appearing fully visible on one side and transparent on the other side. To solve this, we added two Directional lights with different angles of rotation (X rot = 0^o for the first and X rot = 180^o for the second) and different intensity values (2 for the first and 1 for the second).

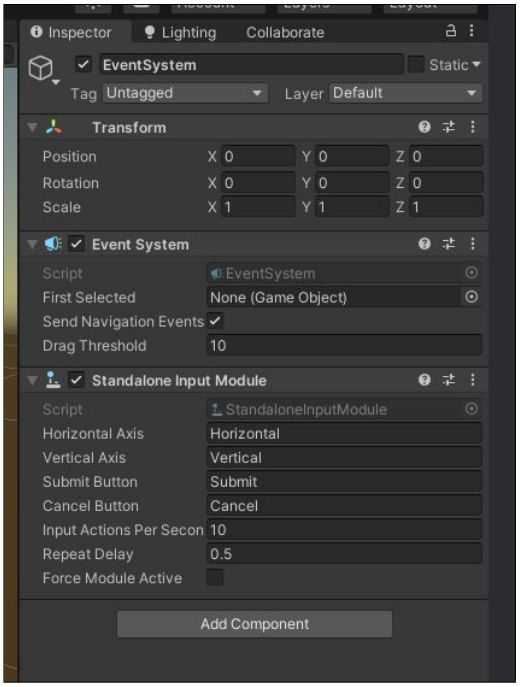


Figure 94: The Event System Object's inspector

Event System (see Fig. 94). Unity's event system is responsible for reading inputs and calling the appropriate functions in all available scripts such as the *OnButtonDown* function inside the *Controller Holobox* script as well as calling the *OnClick* functions on every Game Object with the *Button* component, which is used by all interactable objects inside Holo-box.

7.1.3 Game Master

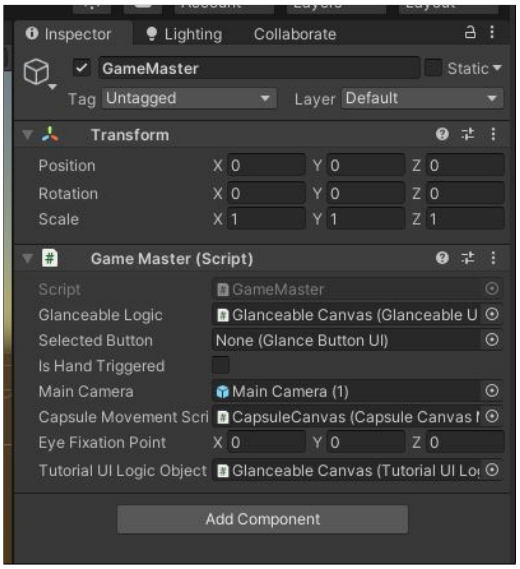


Figure 95: The Game Master Object's inspector

The Game master is an empty Game Object with no visual or functional components (see Fig. 95) that holds the *Game Master* script.

```
public class GameMaster : MonoBehaviour
{
    public static GameMaster instance;

    public GlanceableUILogic GlanceableLogic;
    public GlanceButtonUI SelectedButton;
    public bool isHandTriggered;
    public GameObject MainCamera;
    public CapsuleCanvasMovement CapsuleMovementScript;
    public Vector3 EyeFixationPoint;
    public TutorialUILogic TutorialUILogicObject;



    Unity Message | 0 references
    private void Awake()
    {
        if (instance == null)
        {
            instance = this;
        }
        else
        {
            Destroy(this.gameObject);
        }
    }


    6 references
    public void SwitchSelectedButton(GlanceButtonUI newButton)
    {
        if (newButton == SelectedButton)
            return;

        if (SelectedButton != null)
        {
            SelectedButton.DeselectButton();
        }
        SelectedButton = newButton;
        if (SelectedButton !=null)
            SelectedButton.SelectButton();
    }

    8 references
    public void ToggleIsHandTrigger(bool value)
    {
        isHandTriggered = value;
    }


}
```

Figure 96: The Game Master script

The Game master script holds a static reference to itself, making sure *On Awake* that this script's instance is unique. Any other script can access its public parameters through this instance using the *GameMaster.instance.parameter* or *GameMaster.instance.function()* format.

The Game master serves two functions. First, it holds public references to other objects and scripts which need to be accessed by other scripts easily as well as globally available values (*EyeFixationPoint*). Second, it manages the object selection logic, making sure only one object is selected at any given moment, stored in the *SelectedButton* parameter, and if the selected object was selected through the hand or eye cursor, stored in the *isHandTriggered* boolean parameter.

7.1.4 Holo-Box Game Objects

The Game objects we designed for Holo-box outside the Game Master fall into one of three categories:

- Cursors and their movement managers
- Interfaces with their Interface Logic
- Interface components

In the following chapters we will analyze the most important objects and scripts used by these objects.

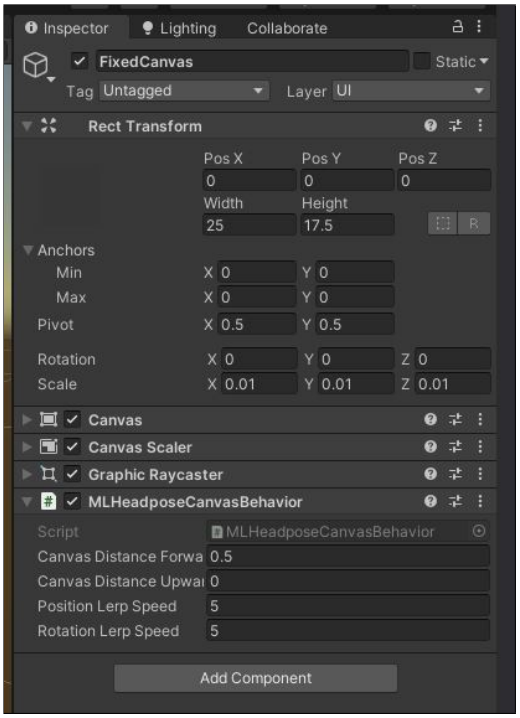


Figure 97: The Fixed Canvas Object's inspector

Fixed Canvas (see Fig. 97). The fixed canvas is a 2D panel that follows the user's viewpoint at a fixed distance using the *MLHeadposeCanvasBehavior* script provided by Magic leap. This script provides a smooth movement and the canvas's distance from the user can be adjusted using the *Canvas Distance Forward* parameter. As the Magic leap has a minimum draw distance of 0.37 meters in front of the user, we set this canvas's distance at a distance of 0.5 meters.

In the prototype implementation, this canvas held the Glanceable interfaces in addition to the eye tracking cursors, but in the final implementation it only holds the eye tracking cursors, which should always be visible in front of the user.

The fixed canvas has two children Game Objects (see Fig. 87) including the *Cursor* and the *ChaseCursor* in addition to the *WorldCursor* which is used for the same logic. The *Cursor* game object is a visual indicator that shows the realtime positon of the user's eye tracking position onto the Fixed Canvas. The *ChaseCursor* is a green cross cursor that follows the *Cursor* in a smooth motion along the Fixed Canvas. The *WorldCursor* is an invisible cursor that goes behind the *ChaseCursor* and collides with 3D objects behind it, activating the triggers of interactable objects selecting them. The *Cursor* and *ChaseCursor* objects are purely visual indicators while the *WorldCursor* has a Spherical collider with a radius of 1 millimeter.

```
void Update()
{
    /* Throw rays and reposition cursors*/
    ExtraCursor.transform.position = Camera.main.transform.position;

    //Cursor reposition Raycast hits only Layer 5(UI)
    RaycastHit rayHit;
    int layer = 1 << 5;

    viewpoint = MLEyes.FixationPoint;
    //Extra cursor is used to get a vector from the camera twards the fixation point
    ExtraCursor.transform.LookAt(viewpoint);
    viewpointStart = ExtraCursor.transform.position;

    if (Physics.Raycast(viewpointStart, ExtraCursor.transform.forward, out rayHit, 1000.0f, layer))
    {
        transform.position = rayHit.point;
    }

    ExtraCursor.transform.LookAt(transform.position);
    //Capsule Cursor. Goes through follow cursor. Hits only Layer 9 (World UI)
    int layer2 = 1 << 9;
    ExtraCursor.transform.LookAt(ChaseCursorObject.transform.position);

    if (Physics.Raycast(viewpointStart, ExtraCursor.transform.forward, out rayHit, 1000.0f, layer2))
    {
        TDMarker.transform.position = rayHit.point;
        Debug.DrawLine(viewpointStart, rayHit.point, Color.red);
    }
    Debug.DrawLine(viewpointStart, viewpoint, Color.white);

    /* Click button via blinking. Do not use if clicking usning hands */
    GameMaster gm = GameMaster.instance;
    if (!gm.isHandTriggered)
    {
        if ((MLEyes.LeftEye.IsBlinking && !MLEyes.RightEye.IsBlinking)
            || (!MLEyes.LeftEye.IsBlinking && MLEyes.RightEye.IsBlinking))
        {
            //if blinking was not on and became on in this frame start clicking
            if (!isBlinking)
            {
                ChaseCursorObject.GetComponent<Image>().color = CursorBlinkingColor;
                if (gm.SelectedButton != null)
                    gm.SelectedButton.DelayedClick();
            }
            isBlinking = true;
        }
        else
        {
            //if blinking was on and became off in this frame stop clicking
            if (isBlinking)
            {
                ChaseCursorObject.GetComponent<Image>().color = CursorIdleColor;
                if (gm.SelectedButton != null)
                    gm.SelectedButton.DelayedClickStop();
            }
            isBlinking = false;
        }
    }
}
```

Figure 98: The Eye Tracking Manager script

Eye tracking interactions are achieved through the *Eye Tracking Manager* script (see Fig. 98) which is placed on the *Cursor* Object. The eye tracker's fixation point is defined through the

MLEyes class included in the *UnityEngine.XR.MagicLeap* package using the *MLEyes.FixationPoint* parameter (line 40).

The script executes a process on every frame inside the *Update* function. First, we position an *ExtraCursor* object on the *MainCamera*'s position (line 34) and rotate the *ExtraCursor* to look towards the eye tracker's fixation point (line 42). Then we perform a raycast from the *ExtraCursor*'s forward vector which collides only with objects in the "UI" layer (line 36,38) which only includes the *Fixed Canvas* object and then reposition the *Cursor* object on the position the ray hit the *Fixed Canvas*.

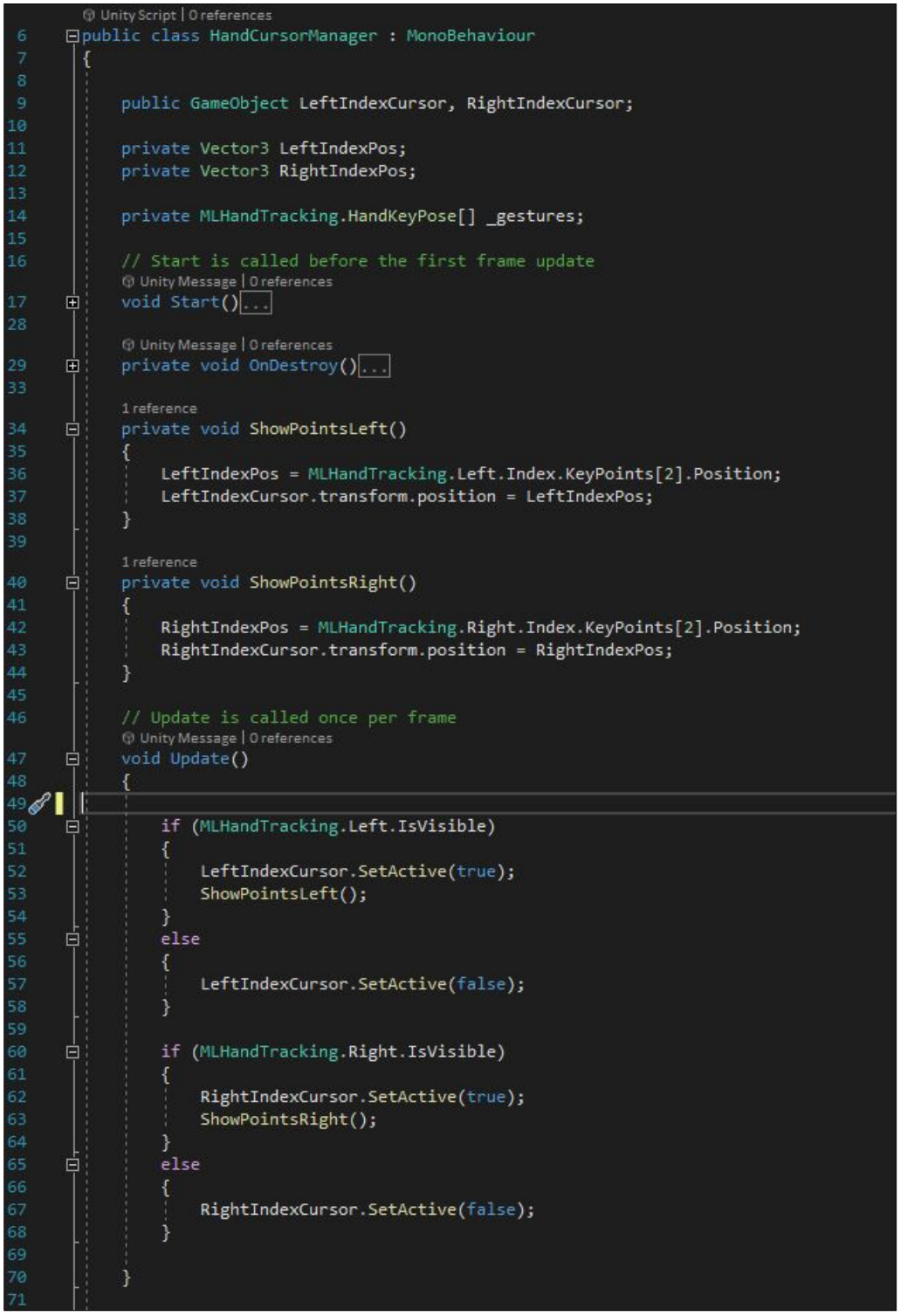
Next, we perform a second raycast and this time the *ExtraCursor* is looking towards the *ChaseCursor* which collides with objects inside the "World UI" layer which includes the Glanceable, Tutorial, Holo-box and Evaluation Interfaces and positions the *World cursor* on the hit position (lines 50-60).

Finally, at the end of the frame, we check the *MLEyes.Left.Eye.IsBlinking* and the *MLEyes.RightEye.IsBlinking* parameters and if the user is blinking with one of their eyes and the user has not selected an object with their hands then a *DelayedClick* is executed. Similarly, if the user was performing a *DelayedClick* using their eyes and they stop blinking the click is topped with the *DelayedClickStop* function (lines 62-89).

```
6   public class CursorChaseLogic : MonoBehaviour
7   {
8       public GameObject TargetObject;
9       public float LerpFactor = 0.8f;
10      public float SpeedFactor = 0.8f;
11      // Start is called before the first frame update
        Unity Message | 0 references
12      void Start()...
16
17      // Update is called once per frame
        Unity Message | 0 references
18      void Update()
19      {
20          transform.localPosition = Vector3.MoveTowards(transform.localPosition,
              TargetObject.transform.localPosition, SpeedFactor * Time.deltaTime);
21
22      }
23  }
24
```

Figure 99: The Chase Cursor Manager script

The *ChaseCursor* handles its own movement through a simple generic script (see Fig. 99) which sets a speed value and a target and then moves towards the target at every frame.



```
Unity Script | 0 references
public class HandCursorManager : MonoBehaviour
{

    public GameObject LeftIndexCursor, RightIndexCursor;

    private Vector3 LeftIndexPos;
    private Vector3 RightIndexPos;

    private MLHandTracking.HandKeyPose[] _gestures;

    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()...

    Unity Message | 0 references
    private void OnDestroy()...

    1 reference
    private void ShowPointsLeft()
    {
        LeftIndexPos = MLHandTracking.Left.Index.KeyPoints[2].Position;
        LeftIndexCursor.transform.position = LeftIndexPos;
    }

    1 reference
    private void ShowPointsRight()
    {
        RightIndexPos = MLHandTracking.Right.Index.KeyPoints[2].Position;
        RightIndexCursor.transform.position = RightIndexPos;
    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {

        if (MLHandTracking.Left.IsVisible)
        {
            LeftIndexCursor.SetActive(true);
            ShowPointsLeft();
        }
        else
        {
            LeftIndexCursor.SetActive(false);
        }

        if (MLHandTracking.Right.IsVisible)
        {
            RightIndexCursor.SetActive(true);
            ShowPointsRight();
        }
        else
        {
            RightIndexCursor.SetActive(false);
        }

    }

```

Figure 100: The Hand Cursor Manager script

Hand Cursor Manager (see Fig. 100). The Hand cursors consist of two Game Objects visualised as two small purple spheres (see Fig. 81) which follow the user's index fingers. The spherers have a radius of 0.5 millimeters, which is approximately the same size as the index finger's nail.

The *Hand Cursor Manager* script uses the *MLHandTracking* class from the *UnityEngine.XR.MagicLeap* package to receive tracking data from the Magic Leap's hand tracker. On every frame through the *Update* function the script checks if either hand is visible through the *MLHandTracking.Left.IsVisible* and *MLHandTracking.Right.IsVisible* parameters and if either hand is visible its cursor is enabled and positioned on the position where the index finger is tracked through the *MLHand-*

Tracking.Left.KeyPoints[2].Position and *MLHandTracking.Right.KeyPoints[2].Position* parameters. Magic leap Tracks 15 key points for every hand and *KeyPoints[2]* indicates the key point at the edge of the index finger. As an alternative, we also used *KeyPoints[0]* which is on the center of the palm which was easier to track but interactions with this point felt unnatural.

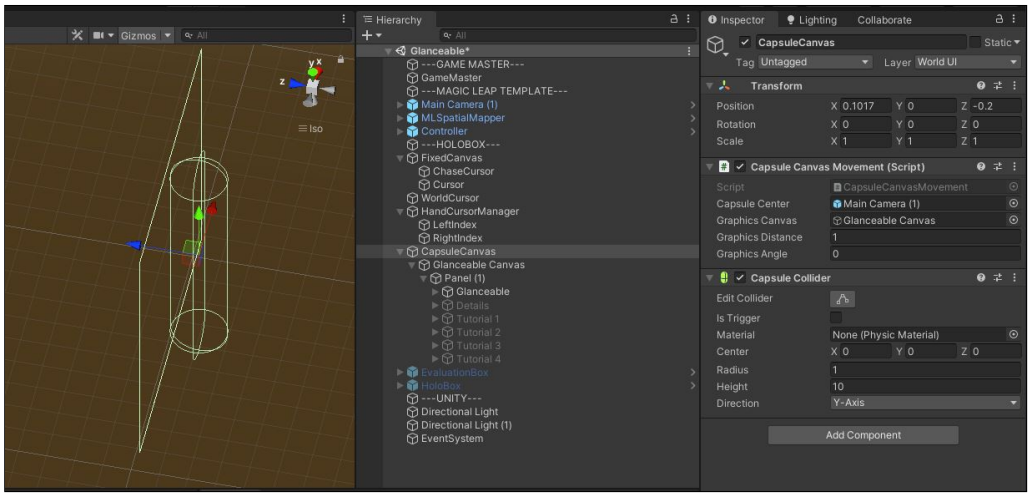


Figure 101: The Capsule Canvas in the Scene (left), Hierarchy (middle) and Inspector (right)

Capsule Canvas (see Fig. 101). The capsule canvas is a combination of two Game objects including the parent *CapsuleCanvas* which is a cylinder centered around the *Main Camera* and a 2D *Glanceable canvas* which rotates around the capsule's center along it's surface.

```
5   public class CapsuleCanvasMovement : MonoBehaviour
6   {
7       public GameObject CapsuleCenter;
8       public GameObject GraphicsCanvas;
9
10      public float GraphicsDistance = 1;
11      public float GraphicsAngle = 0;
12
13      // Start is called before the first frame update
            Unity Message | 0 references
14      void Start()
15      {
16          ResetGraphicsPosition();
17      }
18
19      // Update is called once per frame
            Unity Message | 0 references
20      void Update()
21      {
22          transform.position = CapsuleCenter.transform.position;
23      }
24
            1 reference
25      public void SetGraphicsAngle(float angle)
26      {
27          GraphicsAngle = angle;
28          ResetGraphicsPosition();
29      }
30
            0 references
31      public void SetGraphicsDistance(float distanceoffset)
32      {
33          GraphicsDistance += distanceoffset;
34          ResetGraphicsPosition();
35      }
36
            3 references
37      public void ResetGraphicsPosition()
38      {
39          Vector3 prevPos = GraphicsCanvas.transform.localPosition;
40          GraphicsCanvas.transform.localPosition = new Vector3(prevPos.x,
              prevPos.y, GraphicsDistance);
41          this.transform.localRotation = Quaternion.Euler(0, GraphicsAngle, 0);
42      }
43  }
44
```

Figure 102: The Capsule Canvas Movement script

The *Capsule Canvas Movement* script (see Fig. 102) contains functions that set the orientation and position of Glanceable interfaces. On every frame through the *Update* function, the capsule is set to follow the position of its *CapsuleCenter* parameter, which is set to be the position of the *Main Camera* (see Fig. 101). The *SetGraphicsAngle* function is called from the *Controller Holobox* script to adjust the rotation of the capsule canvas during runtime. Similarly, the *SetGraphicsDistance* function can be used to adjust the radius of the capsule canvas, moving the *Glanceable canvas* closer or further from the user.

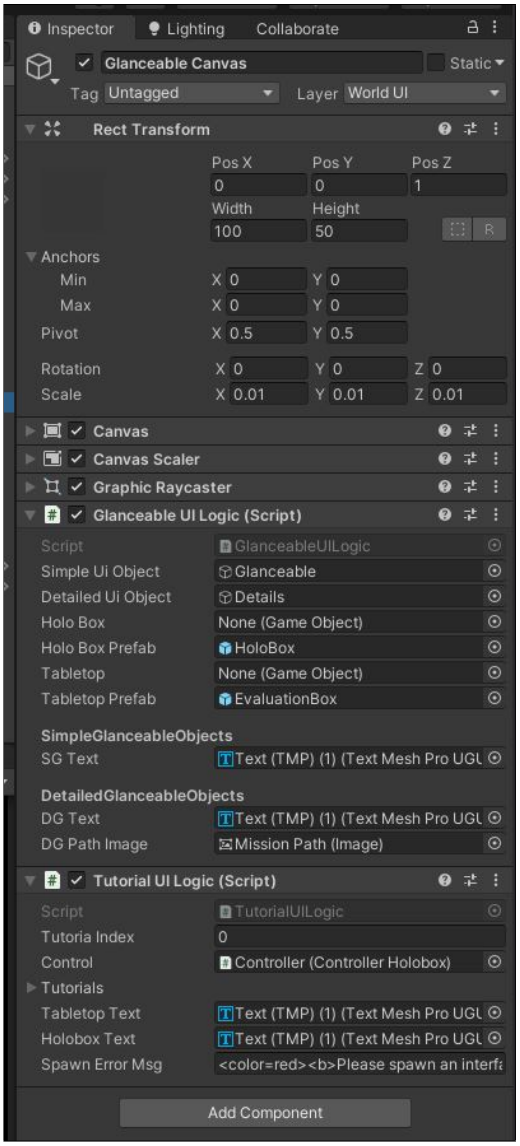
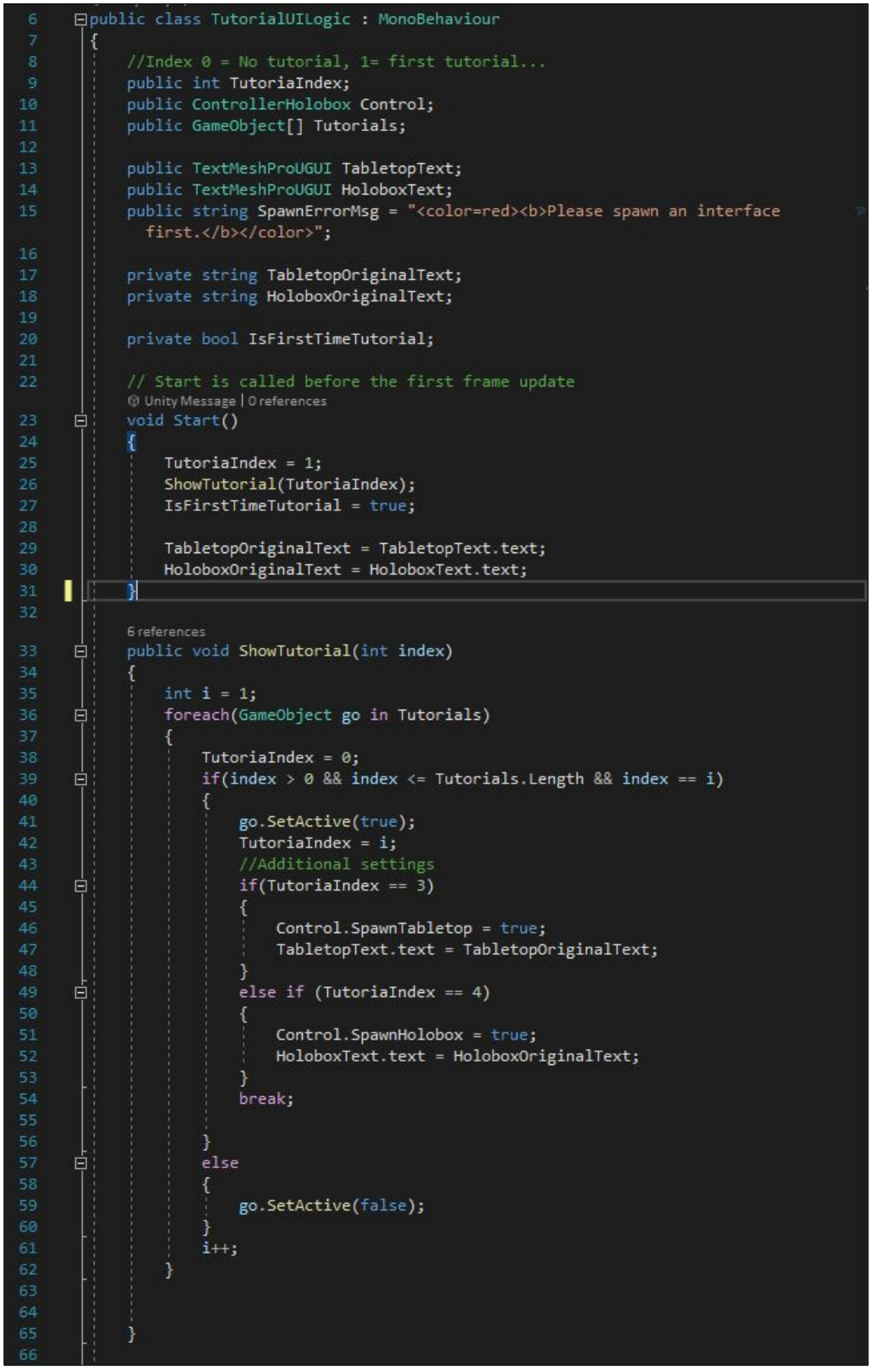


Figure 103: The Glanceable Canvas Object's inspector

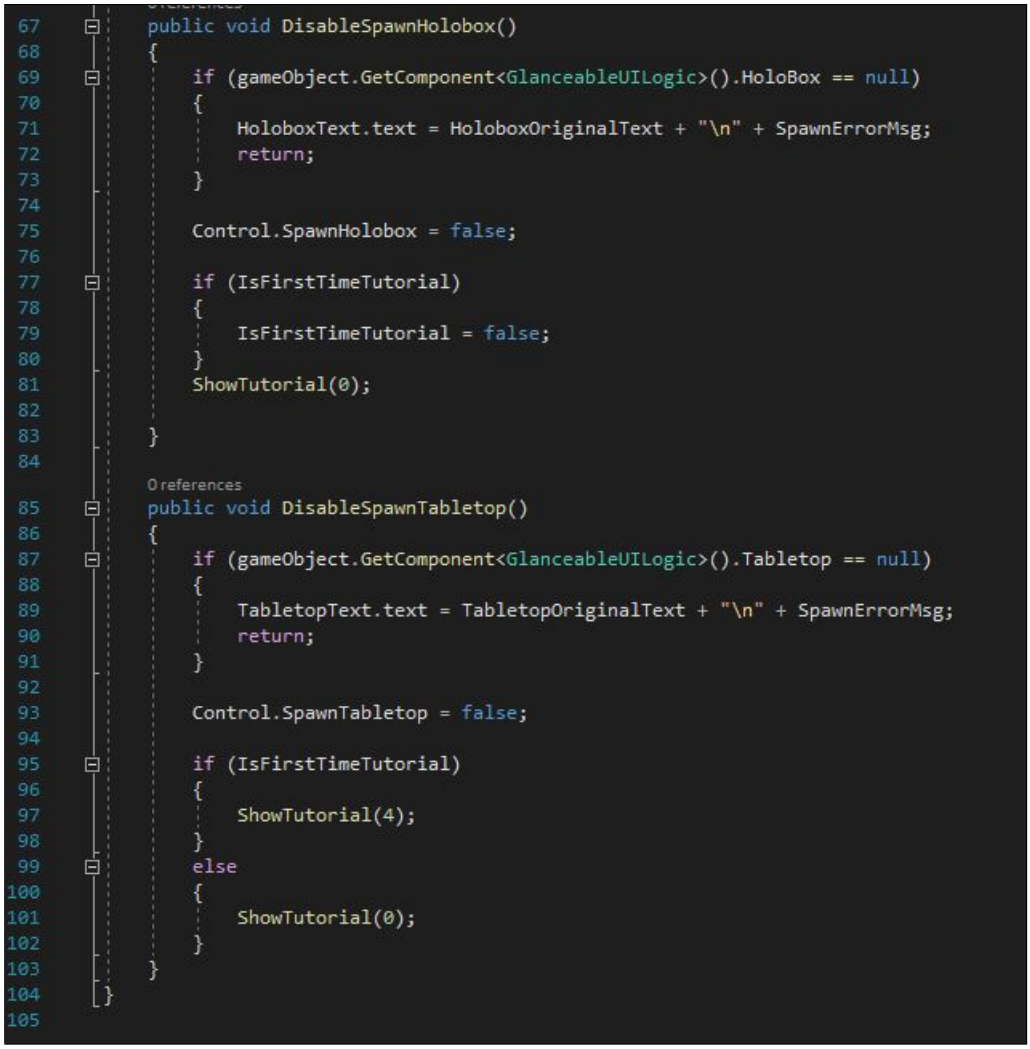
The *Glanceable Canvas* contains all 2D Glanceable interfaces including the *Glanceable* (Simple Glanceable interface), *Details* (Detailed Glanceable interface) and four *Tutorial interfaces* (see Fig. 101) In addition, the *Glanceable Canvas* has two scripts, the *Glanceable UI Logic* for managing the Glanceable interfaces and the *Tutorial UI Logic* script for managing the *Tutorial* interfaces.



```
public class TutorialUILogic : MonoBehaviour
{
    //Index 0 = No tutorial, 1= first tutorial...
    public int TutoriaIndex;
    public ControllerHolobox Control;
    public GameObject[] Tutorials;

    public TextMeshProUGUI TabletopText;
    public TextMeshProUGUI HoloboxText;
    public string SpawnErrorMsg = "<color=red><b>Please spawn an interface
      first.</b></color>";

    private string TabletopOriginalText;
    private string HoloboxOriginalText;

    private bool IsFirstTimeTutorial;

    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        TutoriaIndex = 1;
        ShowTutorial(TutoriaIndex);
        IsFirstTimeTutorial = true;

        TabletopOriginalText = TabletopText.text;
        HoloboxOriginalText = HoloboxText.text;
    }

    6 references
    public void ShowTutorial(int index)
    {
        int i = 1;
        foreach(GameObject go in Tutorials)
        {
            TutoriaIndex = 0;
            if(index > 0 && index <= Tutorials.Length && index == i)
            {
                go.SetActive(true);
                TutoriaIndex = i;
                //Additional settings
                if(TutoriaIndex == 3)
                {
                    Control.SpawnTabletop = true;
                    TabletopText.text = TabletopOriginalText;
                }
                else if (TutoriaIndex == 4)
                {
                    Control.SpawnHolobox = true;
                    HoloboxText.text = HoloboxOriginalText;
                }
                break;

            }
            else
            {
                go.SetActive(false);
            }
            i++;
        }


    }

```

Figure 104: The Tutorial UI Logic script. Initial tutorial execution

The *Tutorial UI Logic* is executed when the application starts through the *Start* function (see Fig. 104). On start, the logic shows the first tutorial interface which shows the first *Tutorial interface*. Each *Tutorial interface* includes a button with a script that is responsible for increasing the *TutorialIndex* and calling *ShowTutorial* again.



```
67      public void DisableSpawnHolobox()
68      {
69          if (gameObject.GetComponent<GlanceableUILogic>().HoloBox == null)
70          {
71              HoloboxText.text = HoloboxOriginalText + "\n" + SpawnErrorMsg;
72              return;
73          }
74
75          Control.SpawnHolobox = false;
76
77          if (IsFirstTimeTutorial)
78          {
79              IsFirstTimeTutorial = false;
80          }
81          ShowTutorial(0);
82
83      }
84
            0 references
85      public void DisableSpawnTabletop()
86      {
87          if (gameObject.GetComponent<GlanceableUILogic>().Tabletop == null)
88          {
89              TabletopText.text = TabletopOriginalText + "\n" + SpawnErrorMsg;
90              return;
91          }
92
93          Control.SpawnTabletop = false;
94
95          if (IsFirstTimeTutorial)
96          {
97              ShowTutorial(4);
98          }
99          else
100         {
101             ShowTutorial(0);
102         }
103     }
104 }
105
```

Figure 105: The Tutorial UI Logic script. Reposition interfaces

The third and fourth *Tutorial Interfaces* are responsible for spawning the *Evaluation Interface* and *Holo-Box interface* objects. Thus, when their buttons are pressed, they call the *DisableSpawnTabletop* and *DisableSpawnHolobox* functions accordingly (see Fig. 105). These functions check if the appropriate interface was spawned and then proceed to the next tutorial step. The *TutorialIndex* is also used as an indicator on the *Controller Holobox* script to determine which interface to spawn when the Magic Leap Control bumper is clicker. Additionally, the Glanceable, evaluation and Holo-box interfaces are disabled as long as the *TutorialIndex* is not zero.

These interfaces may also be activated again through the *Evaluation Interface* after the tutorial is completed in order to reposition the *Evaluaiton Interface* or *Holo-Box interface*. In this case, the tutorials will not be executed sequentially and instead after one tutorial step the *TutorialIndex* will be reset to zero indicating no *Tutorial interface* is enabled.

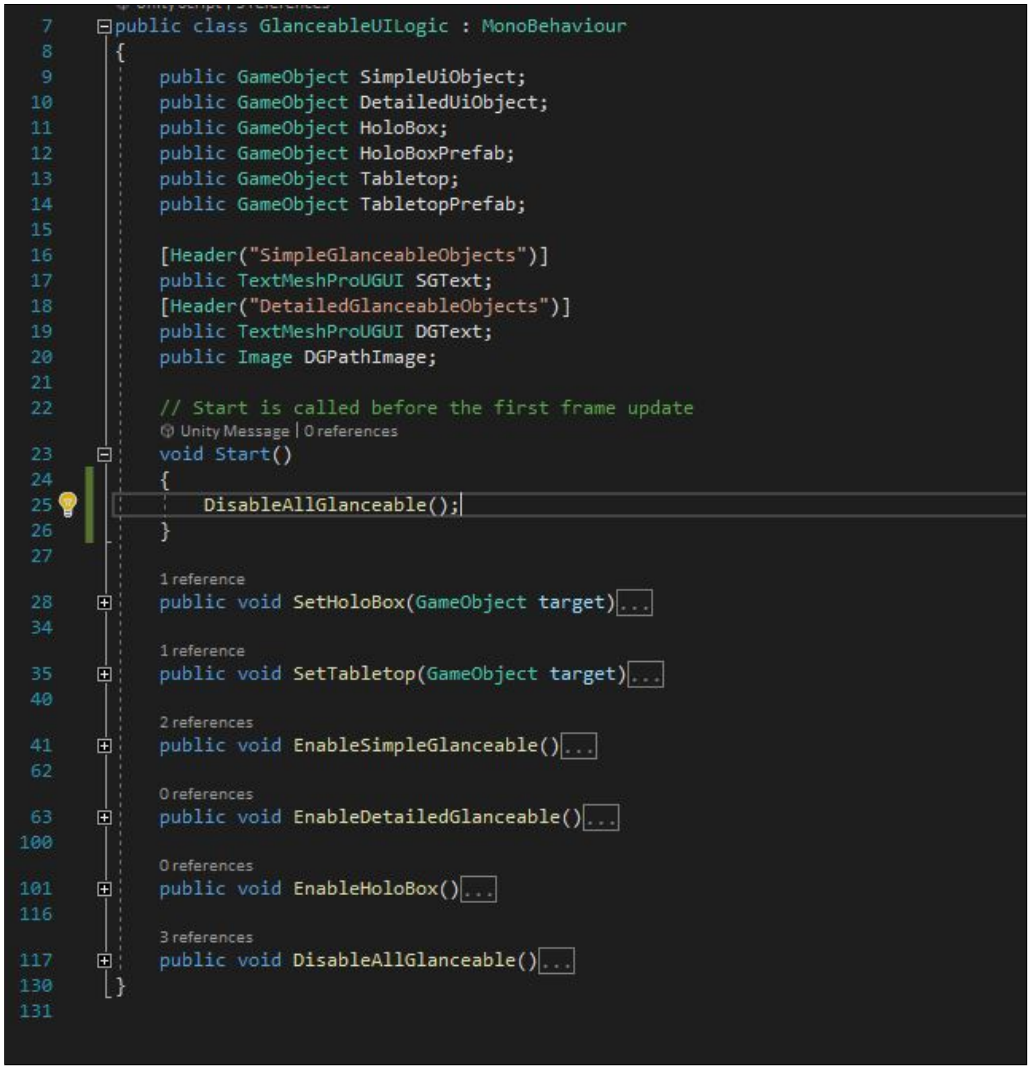


Figure 106: The Glanceable UI Logic script

The *Glanceable UI Logic* script handles all logic regarding enabling and disabling the Glanceable and Holo-box interfaces. At the start of the application's execution through the *Start* function, all three interfaces are disabled.

When the Magic Leap's Control's bumper is pressed through the *Controller Holobox* script the appropriate interface is instantiated and then the *SetHoloBox* or *SetTabletop* functions are called, destroying the previous instance of the object, if any, and then setting the *HoloBox* or *Tabletop* parameters to the new object.

The *EnableSimpleGlanceable*, *EnableDetailedGlanceable* and *EnableHoloBox* functions enable the appropriate interface while disabling the other two. These functions also set the objects and values that appear in each interface according to the selected mission's parameters including setting texts, the 2D image of the Detailed Glanceable interface and the 3D model of the Holo-box's finished product.

These three functions are called at the following times:

- *EnableSimpleGlanceable*: When a new mission is activated through the *Evaluation Box Logic*. When the Holo-box interface is closed using the "X" button.
- *EnableDetailedGlanceable* When the "eye" button on the Simple Glanceable interface is pressed.
- *EnableHoloBox* When the "eye" button on the Detailed Glanceable interface is pressed.

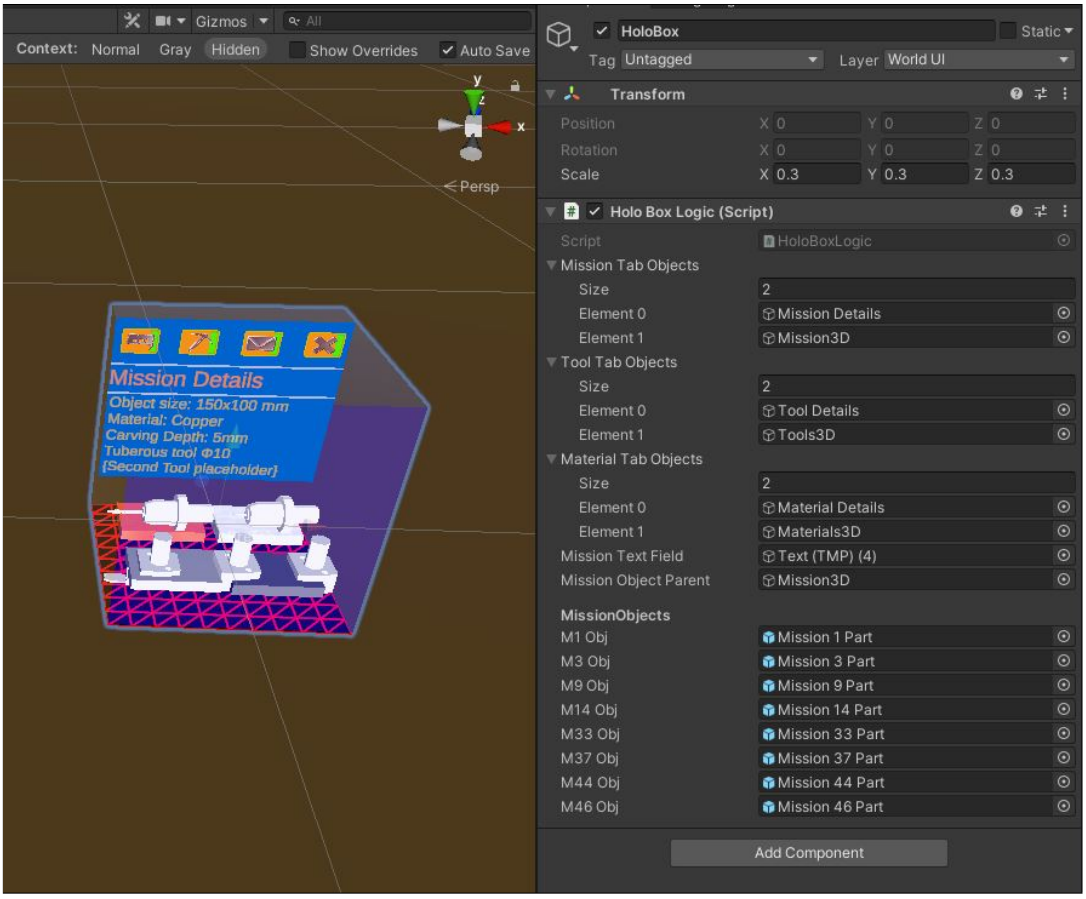


Figure 107: The Holo-Box Interface in the Scene (left) and Inspector (right)

Holo-Box Interface (see Fig. 107). The Holo-box holds multiple 2D and 3D objects which includes the following:

- Floor panel. This is indicated by a wireframe pattern at the bottom of the Holo-box
- Transparent box. This is the blue box which indicates the perimeter inside which virtual content will appear. The box is always visible at 95% transparency making it visible without occluding its contents.
- Back Panel. The back panel consists of two parts. On the top of the panel are four interactable buttons. The first three cycle between the three "tabs" of the Holo-Box interface including the *Mission Tab* (see Fig. 83), *Tool Tab* (see Fig. 84a) and *Materials Tab* (see Fig. 84a) with the fourth button closing the Holo-Box interface and returning to the Simple Glanceable. Below the buttons is a description whose content changes based on which tab is enabled.
- 3D tools. Five tools are placed inside the Holo-box, which are shown when the *Tool Tab* is enabled. Each tool includes a 3D text with its name which is placed above the 3D model and is shown when the tool is selected using eye or hand cursors.
- 3D Materials. Four Rectangular blocks indicated four different materials are placed inside the Holo-box, which are shown when the *Material Tab* is enabled. Each material includes a 3D text with its name which is placed above the 3D model and is shown when the material is selected using eye or hand cursors.
- 3D Finished Mission Product. Unlike the Tools and materials, which were manually placed so that their colliders don't overlap, Holo-box includes an array of references to all finished products inside the *MissionObjects* list. When the *Mission Tab* is shown, the appropriate finished product is instantiated during runtime.

```
Unity Message | 0 references
37  void OnEnable()
38  {
39      SwitchTab(1);
40  }
41
42
    0 references
43  public void CloseHoloBox()...
47
    4 references
48  public void SwitchTab(int option)...
69
    1 reference
70  public void Init(ActiveMissionData am)...
132
133 }
```

Figure 108: The Holobox UI Logic script

Initially, the Holo-box is enabled when it is spawned during the Tutorial. During that time, all the objects except the *Floor panel* and *Transparent box* are disabled. When the "eye" button of the Detailed Glanceable interface is pressed the content of the Holo-box is enabled, showing the *Mission Tab* (see Fig. 108) on the *OnEnable* function. The buttons that change the active tab execute the *SwitchTab* function with the appropriate parameter. If the Holo-box's "X" button is pressed, all tabs are disabled and the *CloseHoloBox* function is called, enabling the Simple Glanceable interface. When a new mission is given to the user, the *Init* function is called setting the content of the *Mission Tab* accordingly.

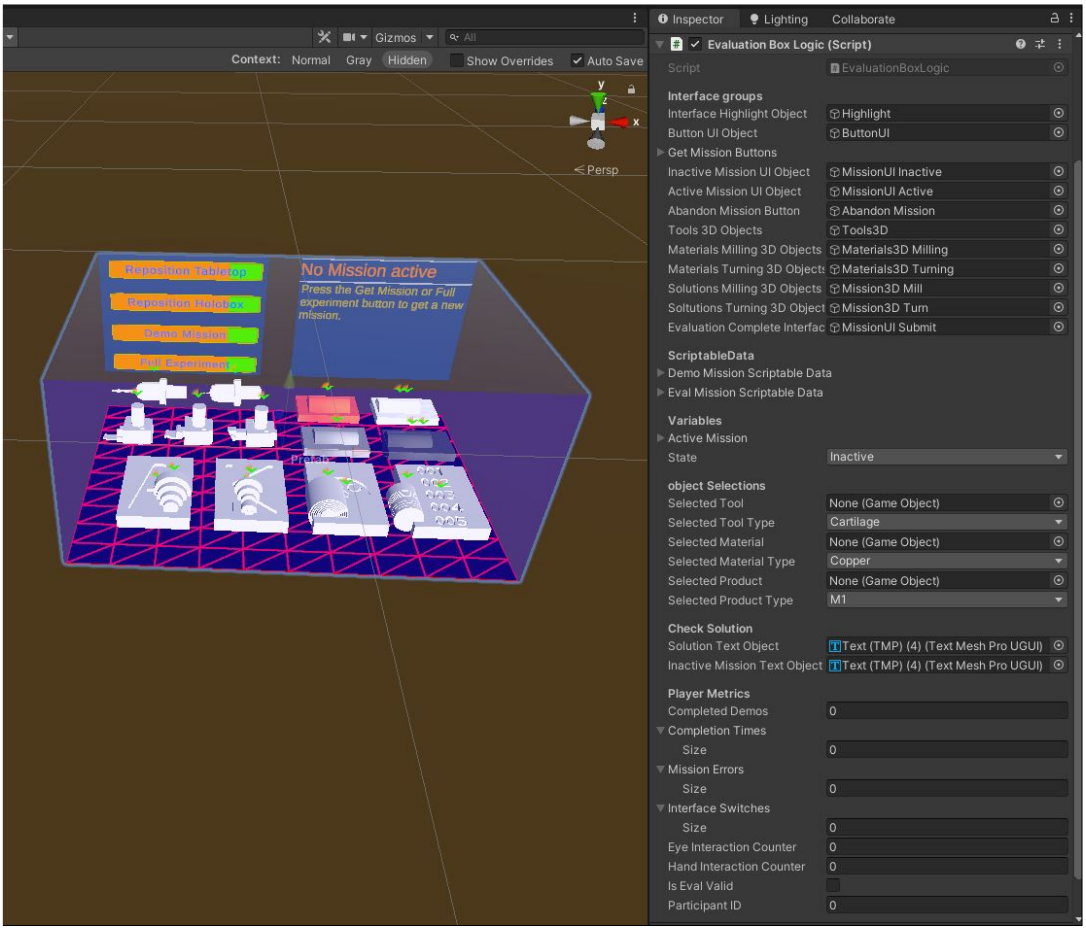


Figure 109: The Evaluation Interface in the Scene (left) and Inspector (right)

Evaluation Interface (see Fig. 109). The evaluation interface holds all the logic used for the execution of the user study outside the core logic of our proposed interface system.

The prefab of the *Evaluation Interface* includes the following Game Objects:

- Floor panel. This is indicated by a wireframe pattern at the bottom of the interface.
- Transparent box. This is the blue box which indicates the perimeter inside which virtual content will appear. The box is always visible at 95% transparency making it visible without occluding its contents.
- Left Back Panel. The Left back panel includes four buttons. The first two buttons include the *Reposition Tabletop* and *Reposition Holobox* buttons, which execute the appropriate logic from the *Tutorial UI Logic* and reposition one of the 3D interfaces. The last two buttons include the *Demo Mission* and *Full Experiment* buttons. The *Demo Mission* button generates a single mission, which can be abandoned at any time and if it is solved or abandoned it returns the Evaluation interface it its default state. The *Full experiment* on the other hand generates 10 missions, which are given to the user back to back and cannot be abandoned and when all 10 missions are completed the Evaluation interface returns to its default state.
- Right Back panel. The second back panel handles the per-mission execution. By default it includes a text stating that "No mission is active". If the user receives a *Demo Mission*, two buttons are added to the interface, one to *Solve Mission* and one to *Abandon mission*. Similarly, if the *Full Experiment* is pressed, only the *Solve mission* button is shown. When the *Solve mission* button is pressed, the Evaluation interface checks whether the use has selected the appropriate 3D objects and completes the mission if successful or provides textual feedback above the buttons if the user made an error.
- 3D tools. Five tools are placed inside the interface, which are shown when a mission is active. Each tool includes an interaction collider and two visual indicators above it, one circularly loading indicator showing that a *Delayed Click* is being executed to select the object and a green arrow to indicate which tool is selected, if any.
- 3D Materials. Four Rectangular or cylindrical objects that indicate four different materials are placed inside the interface, which are shown when a mission is active. The rectangular materials are shown for milling missions that use a rectangular stock material and the cylindrical models are used for turning missions. Each of the four materials is identified by its color and the same four colors are used in both cases. Each material includes an interaction collider and two visual indicators above it, one circularly loading indicator showing that a *Delayed Click* is being executed to select the object and a green arrow to indicate which material is selected, if any.
- 3D Finished product. Four Rectangular or cylindrical objects that indicate eight different Finished products are placed inside the interface, which are shown when a mission is active. The rectangular Finished products are shown for milling missions that use a rectangular stock material and the cylindrical models are used for turning missions. Each Finished product includes an interaction collider and two visual indicators above it, one circularly loading indicator showing that a *Delayed Click* is being executed to select the object and a green arrow to indicate which Finished product is selected, if any.

The Evaluation interface includes the *EvaluationBoxLogic* script that holds multiple variables separated into the following categories (see Fig. 109):

- Interface Groups. This includes references to all the objects ore sets of objects included in the interface.
- Scriptable Data. This includes two arrays of Scriptable Data objects, with each one holding all of the necessary information for each available mission. These are separated into *Demo Missions* and *Evaluation Missions* with three demo and four evaluation missions available.
- Variable Data. These include parameters which are used during runtime. These include the *Active mission* which is currently being executed and the *Tabletop State* enumeration which defines what objects are enabled inside the Evaluation interface at any moment.

- Check solution parameters. These define the text fields which are used to provide feedback after the *Check Solution* button is pressed.
- Player Metrics. These parameters include all the metrics that are written in a .CSV file in sequential format regarding a user's performance during the *Full Experiment*.

The *EvaluationBoxLogic* also includes multiple functions that are used during the execution of the application as follows:

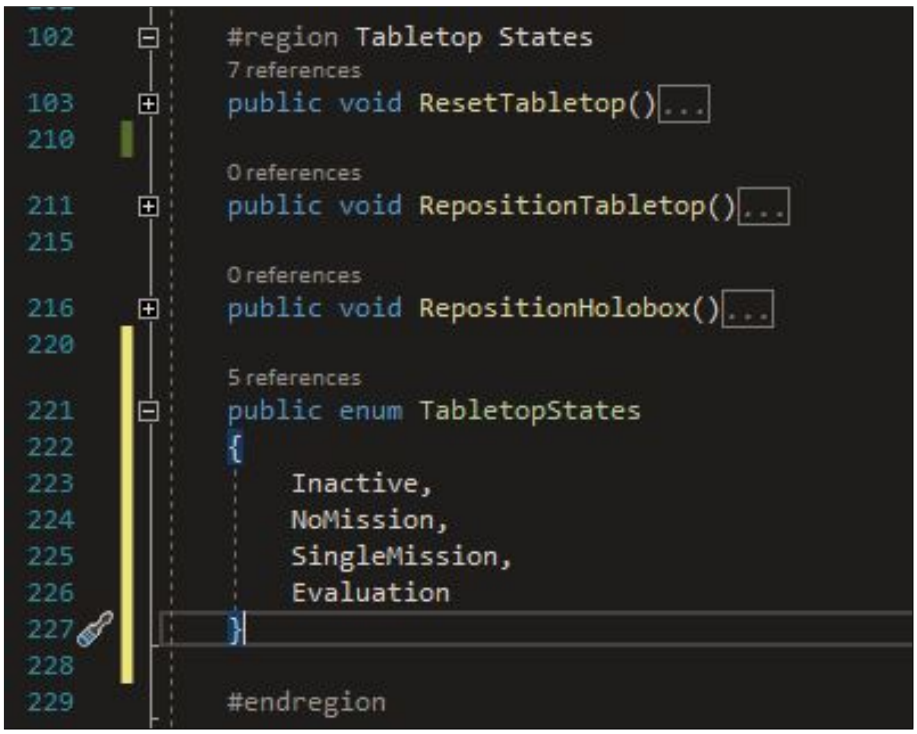


Figure 110: The Evaluation UI states functions

The *TabletopStates* enumeration defines the four distinct states the Evaluation interface is in at any moment. As long as any tutorial interface is active, the Evaluation Interface is in *Inactive* state. If no tutorial interfaces are active and there is no active mission, the Evaluation Interface is in *NoMission* state. The Evaluation Interface is in the *SingleMission* state if the Active mission is a *Demo mission* or in the *Evaluation* state otherwise.

The *RepositionTabletop* and *RepositionHolobox* functions are called when the same buttons are pressed on the Left Back Panel. The *ResetTabletop* is triggered when the *Tabletop State* changes to *NoMission*.

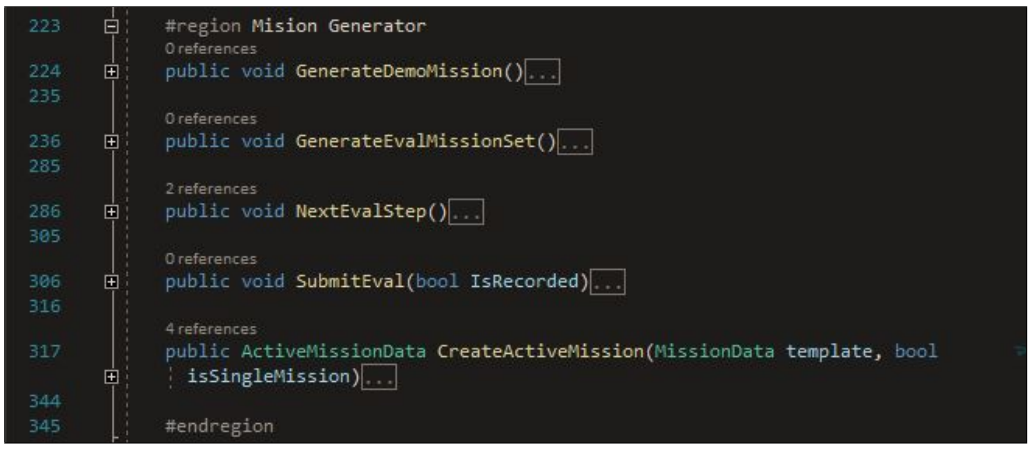


Figure 111: The Evaluation UI mission generator functions

The **GenerateDemoMission** and *GenerateEvalMission* functions are called when the the same buttons are pressed on the Left Back Panel. Both of these functions select one *Mission Data* Scriptable Object from either the *DemoMissionsScriptableData* or *EvalMissionsScriptableData* lists and call the *CreateActiveMission* function with the correct parameters. Similarly, for a *Full experiment* after the first mission is solved, for every subsequent mission the *NextEvalStep* function is called which selects the appropriate *Mission Data* Scriptable Object from the *EvalMissionsScriptableData* list and calls the *CreateActiveMission* function with the correct parameters. The *NextEvalStep* function also writes the *Player Metrics* to a .CSV file and returns the state to *NoMission* if the final mission of the evaluation is completed.

```
347     #region Item Selections
348
        1 reference
349     public void SetTool (GameObject Object, ToolIndexes Index)...
355
        1 reference
356     public void SetMaterial(GameObject Object, MaterialIndexes Index)...
362
        1 reference
363     public void SetProduct(GameObject Object, MissionIndexes Index)...
369
        2 references
370     public void DeselectTool()...
375
        2 references
376     public void DeselectMaterial()...
381
        2 references
382     public void DeselectProduct()...
387
388     #endregion
389
```

Figure 112: The Evaluation UI item selection functions

The *SetTool*, *SetMaterial* and *SetProduct* functions toggle the selection of the appropriate tool, material or finished product, deselecting the previous one if any and then enabling it's visual indicator that it is selected. Similarly, the *DeselectTool*, *DeselectMaterial* and *DeselectProduct* functions disable the visual indicator that an object is selected and setting the appropriate value to null. The Deselect functions are called when another object is set or when a mission is solved.

```
390     #region Mission Solution
391
        0 references
392     public void CheckSolution()...
426
        0 references
427     public void AbandonMission()...
432
433     #endregion
434
```

Figure 113: The Evaluation UI mission solution functions

The *CheckSolution* and *AbandonMission* functions are called when the the same buttons are pressed on the Right Back Panel. The *Check solution* function checks which 3D objects are selected in the evaluation interface and then writes feedback on the Right Back Panel if the user made an error, otherwise it records the appropriate data for the mission completion in the *Player metrics* and then returns the state to *NoMission* for a *Demo mission* or calls the *NextEvalStep* for a *Full experiment*. The *Abandon mission* on the other hand sets the *Active Mission* to null and sets the state to *NoMission*.

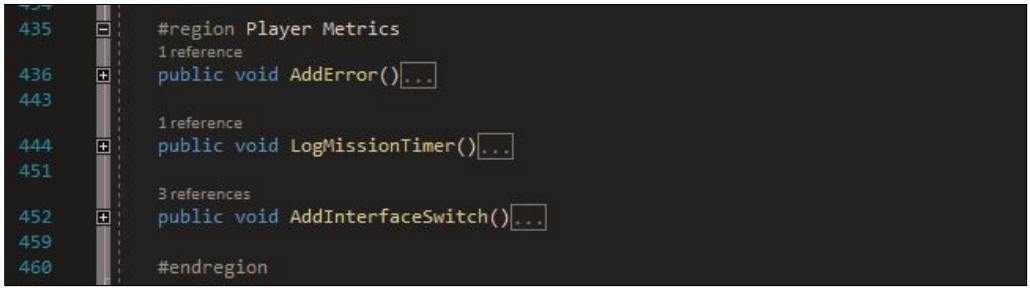


Figure 114: The Evaluation UI player metrics tracker functions

The *AddError*, *LogMissionTimer* and *AddInterfaceSwitch* functions are called from the *CheckSolution* function to record the three parameters which are recorded separately for every mission.

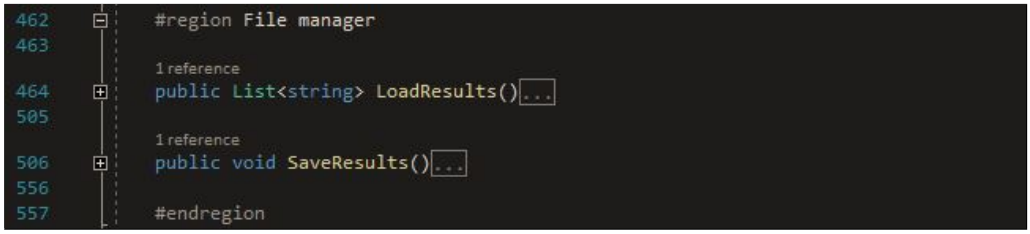


Figure 115: The Evaluation UI file manager functions

The *LoadResults* decrypts the results file from the .CSV format into a list of strings for every line, while the *SaveResults* takes a list of strings and encrypts it as a .CSV file. These functions are called on the *NextEvalStep* function after all the missions of a *Full Experiment* are completed.

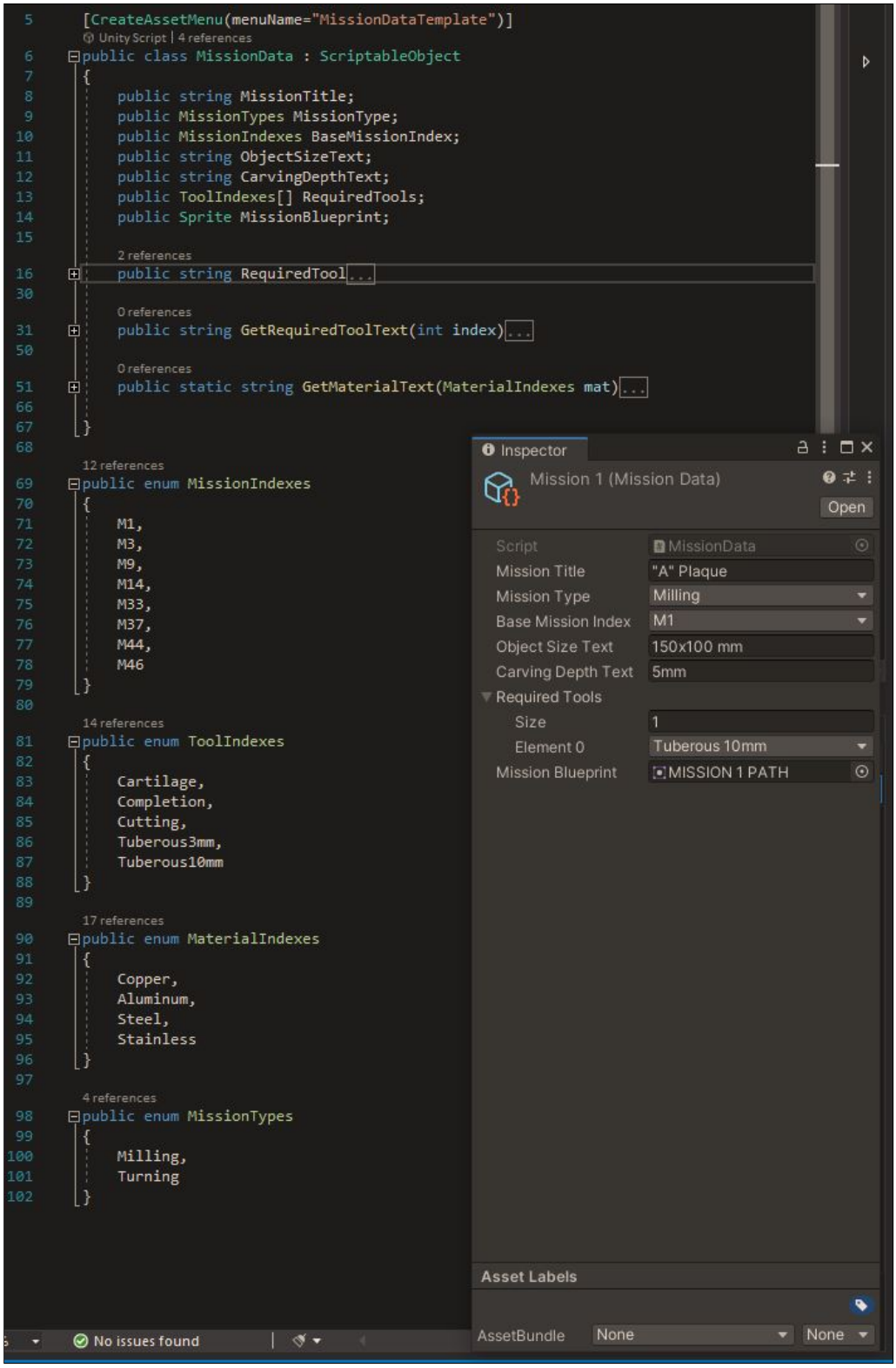


Figure 116: The Gamified Manufacturing Mission Scriptable Data Script and Inspector

Gamified Manufacturing Mission Scriptable Data (see Fig. 116). The Scriptable Data class is a special Data holder class type used by Unity. A Scriptable Data class defines a set of variables, an instance of which can be saved on the project's files on the disk as a preset set of values.

Our *MissionData* class includes all of the required information for any given mission. The *MissionType*, *BaseMissionIndex*, *ToolIndex* and the *Material Indexes* are defines as enumerations,

allowing for specific values, which can be selected through Unity's inspector as dropdown menus. Even though the *MaterialIndexes* are defined as an enumeration with four values, each mission does not have a predefined material, and the requested material is randomly selected out of the available four every time the *GenerateMission* function is called.

The tool, material, finished product and mission type enumerations are also used as identifiers of their 3D models in the Holo-box and Evaluation Interfaces to identify the correct objects to spawn during their initialization and to check if the correct objects are selected during the *CheckSolution* function.

The *MissionData* class also includes two converters that convert the tool and material enumerations into their text, which is then integrated into the 3-LOD interfaces. The *RequiredTools* are stored as an array if for an extension of this work a mission with multiple required tools is presented.

Outside the enumerations, a set of strings are also present including the *MissionTitle*, *ObjectSize* and *CarvingDepth* which are used on the interfaces as-is. Finally, the scriptable data also includes a sprite value which holds the 2D image shown in the Detailed Glanceable UI.