ΔΥΝΑΜΙΚΗ ΟΠΤΙΚΟΠΟΙΗΣΗ 3D ΑΝΤΙΚΕΙΜΕΝΩΝ ΜΕΣΩ ΕΠΑΥΞΗΜΕΝΗΣ ΠΡΑΓΜΑΤΙΚΟΤΗΤΑΣ

του

ΣΦΥΡΑΚΗΣ ΜΥΡΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

υποβληθείσα σε μερική εκπλήρωση των απαιτήσεων για το πτυχίο

ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΕΛΛΗΝΙΚΟ ΜΕΣΟΓΕΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

2023

Εγκεκριμένη από:

Δρ. Βιδάκης Νικόλαος

DYNAMIC VISUALIZATION OF 3D OBJECTS THROUGH AUGMENTED REALITY

by

SFYRAKIS MYRON

A THESIS

submitted in partial fulfillment of the requirements for the degree

BACHELOR OF INFORMATICS ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

SCHOOL OF ENGINEERING

HELLENIC MEDITERRANEAN UNIVERSITY

2023

Approved by:

Dr. Vidakis Nikolaos

Περίληψη

Η επαυξημένη πραγματικότητα (AR) είναι μια τεχνολογία που επικαλύπτει ψηφιακές πληροφορίες και στοιχεία στον φυσικό κόσμο, ενισχύοντας την εμπειρία του χρήστη στον πραγματικό κόσμο. Έχει διάφορες εφαρμογές, συμπεριλαμβανομένων των βιομηχανιών ψυχαγωγίας, εκπαίδευσης, τυχερών παιχνιδιών, λιανικής και διαφήμισης. Αν και μπορεί να βρίσκεται ακόμη στο αρχικό της στάδιο, σίγουρα κλιμακώνεται σε δημοτικότητα και οι προγραμματιστές το χρησιμοποιούν ακόμη πιο συχνά κάθε μέρα και σε όλο τον κόσμο. Ένα από τα ατυχή, αλλά λογικά πράγματα για το AR, είναι η έλλειψη εργαλείων, που βοηθούν τους προγραμματιστές και όχι μόνο, να εξοικειωθούν περισσότερο με την τεχνολογία και τελικά να ξεκινήσουν εύκολα και γρήγορα την ανάπτυξη εφαρμογών που έχουν να κάνουν με εμπειρίες επαυξημένης πραγματικότητας.

Για όλους τους προαναφερθέντες λόγους, χρειάστηκε να υλοποιηθεί ένα εργαλείο με το όνομα 3D Visualization in AR. Βοηθά κάθε χρήστη που θέλει να δημιουργήσει μια εμπειρία AR που θα οπτικοποιήσει, με μοναδικό τρόπο, κάθε προσαρμοσμένο αντικείμενο που επιθυμεί. Η κύρια συμπεριφορά των προσαρμοσμένων αντικειμένων που υποστηρίζει αυτήν τη στιγμή το εργαλείο, είναι η συμπεριφορά τροχιάς, αλλά μπορεί εύκολα να επεκταθεί για να υποστηρίξει περισσότερους τύπους συμπεριφορών. Το σύνολο εργαλείων αποτελείται από δύο διαφορετικούς custom editors στο περιβάλλον Unity, όπου ο πρώτος είναι υπεύθυνος για την επιλογή της συμπεριφοράς που θα χρησιμοποιηθεί. Ο δεύτερος custom editor, ο οποίος είναι ο κύριος, απαιτεί από τον χρήστη να συμπληρώσει ορισμένες ιδιότητες που θα καθορίσουν τη συμπεριφορά τροχιάς του προσαρμοσμένου αντικειμένου, όπως η ταχύτητα τροχιάς, η απόσταση, η τροχιά γύρω από το αντικείμενο κ.λπ.

Λέξεις κλειδιά: Επαυξημένη πραγματικότητα, AR, 3D, οπτικοποίηση, εργαλείο, αλληλεπίδραση χρήστη-υπολογιστή (HCI)

Abstract

Augmented Reality (AR) is a technology that overlays digital information and elements onto the physical world, enhancing the user's real-world experience. It has various applications, including entertainment, education, gaming, retail, and advertising industries. Although it might still be at its early stage, it is surely escalating into popularity, and developers even more frequently use everyday use and across the world. One of the unfortunate, but logical things about AR, is the lack of tools, which help the developers and not only, to get more familiar with the technology and eventually, start easily and quickly the development of applications that have to do with augmented reality experiences.

For all the aforementioned reasons, a toolset named 3D Visualization in AR needed to be implemented. It helps any user who wants to build an AR experience that will visualize, in a unique way, any custom object that he desires. The main behavior of the custom objects that the toolset currently supports is the orbit behavior, but it can easily be extended to support more types of behaviors. The toolset is made of two different custom editors in the Unity environment, where the first one is responsible for the selection of the behavior to be used. The second custom editor, which is the main one, requires the user's input to fill in some properties that will define the custom object's orbit behavior, such as the orbit speed, the distance, the orbit around object, etc.

Keywords: augmented reality, AR, 3D, visualization, tool, human-computer interaction (HCI)

Acknowledgements

Primarily, I would like to extend my sincere gratitude to my supervising committee, Dr. Nikolaos Vidakis, and Ilias Logothetis, for their invaluable guidance and support throughout my research journey. Their wealth of knowledge and expertise in the field has been an immense source of inspiration, and I am deeply grateful for the time and effort they have dedicated to my growth as a scholar.

Secondly, I would also like to express my heartfelt thanks to my family, who have been my rock and source of encouragement throughout this process. Their unwavering love and support have allowed me to pursue my dreams and achieve this important milestone. I am incredibly grateful to my parents, who have always believed in me and encouraged me to strive for excellence. Their sacrifices and dedication have been instrumental in shaping me into the person I am today, and I will be forever grateful for their unconditional love and support.

I would also like to acknowledge the support of my friends and colleagues from Nile Lab, who have provided me with a network of encouragement and motivation throughout this journey. Special thanks to Harris, Dimitris, and Sakis because this journey may not have ended like this without them. Finally, I would like to express my appreciation to all who have played a role in this thesis, directly or indirectly. Your contributions have been deeply appreciated, and I am grateful for everything you have done to make this a successful experience.

Table of Contents

List of Figures

Chapter 1 - Introduction

Augmented Reality (AR) and Virtual Reality (VR) are powerful technologies that are increasingly being used in various aspects of our daily lives. AR enables the user to overlay virtual content onto the real world [1], while Mixed Reality (MR) blends the real and virtual worlds [2]. VR, on the other hand, provides a fully immersive experience in a virtual environment [3]. The combination of all the aforementioned technologies that use a mix of both real and virtual word, belong to a bigger category named Extended Reality (XR) [4].

These technologies start to be increasingly popular nowadays, since they enhance the real world by overlaying digital information on top of it, making it easier for us to understand and interact with the environment. They make all the experiences way more interactive, by providing users with the ability to manipulate virtual objects in real-time. An also mayor benefit of their use, is the fact that can be an effective tool for education and learning [5], making complex concepts easier to understand and allowing students to engage in interactive experiences.

As concerns the developing side of the XR community, quite many tools and software products, such as ZeusAR [6] and EduARdo [7], have been published all this time. Each one of them contributes on their own way, making the developing process of XR applications way easier and more accessible to any new, to the XR world, developer. This way, the development of XR applications has become more accessible, allowing for a greater diversity of experiences to be created, and providing new opportunities for growth and innovation in the field.

1.1 Motivation

After researching all the previously mentioned tools, the lack of dynamic addition of custom objects in any scene during the development phase of an application that uses XR and especially AR was clear. Most of the already existing ones, do not have any future of visualizing objects with custom movement behavior. That is why the decision to create the proposed toolset was set. The toolset would help developers to build their own AR experiences, by just interacting with the Graphical User Interface (GUI) of the toolset, saving them a lot of time instead of doing this process manually by writing code.

1.2 Objective

The main goal is to create a toolset in Unity Game Engine [8] that consists of two custom editors. The first one, is responsible for the selection of the object's movement behavior, where the second one, is responsible for the essential settings, according to the selected behavior from the previous editor. The user will have the option to either create a new custom object or update an already existing one from active scene's hierarchy. Either way, the user must set all the required settings through the toolset's GUI, which easily guides him from the beginning of the custom object's creation process, until the end.

The main custom object's behavior to be implemented will be the orbit behavior, which consists of two kinds of rotations. Firstly, by rotating an object around itself, and secondly, by orbiting a custom object around another at the same time. The basic settings of this behavior is setting each custom object's orbit speed, scale, center position (anchor point), name, information and more that will be later analyzed more in-depth. The proposed toolset is an essential part of the EduARdo [7] system and the current report is heavily based on the already published academic paper, especially in the evaluation part of the toolset.

1.3 Outline

The current thesis report consists of seven main chapters:

Chapter 1 – Introduction*:* A short mention of the technologies to be used, similar work's weaknesses, and the final objective of the current thesis project.

Chapter 2 – Background Work*:* A literature review about the technologies to be used.

Chapter 3 – Similar Work*:* A mention of other systems and tools alike this thesis.

Chapter 4 – Software Design*:* An overview of the struct and architecture.

Chapter 5 – System Implementation*:* An in-depth explanation and breakdown of the whole process during the developing and 3D modeling phase.

Chapter 6 – Experiment Setup*:* An evaluation of the toolset.

Chapter 7 – Conclusion and Future Work*:* Final thoughts and potential future developments.

Chapter 2 - Background Work

2.1 Augmented Reality

Augmented Reality (AR) is a rapidly growing technology that has garnered significant attention in recent years. AR enhances the real world by overlaying digital information and graphics on it, providing users with a more immersive and interactive experience. This technology has been applied in various domains, including education, gaming, and healthcare, among others.

In education, AR has the potential to revolutionize the way students learn, providing them with interactive and engaging experiences [9], [10]. For example, AR can be used to create educational content that is more interactive [11], allowing students to explore complex concepts in a more intuitive and direct manner. Additionally, AR can be used to provide students with instant access to information, without the need for additional devices such as a computer or smartphone.

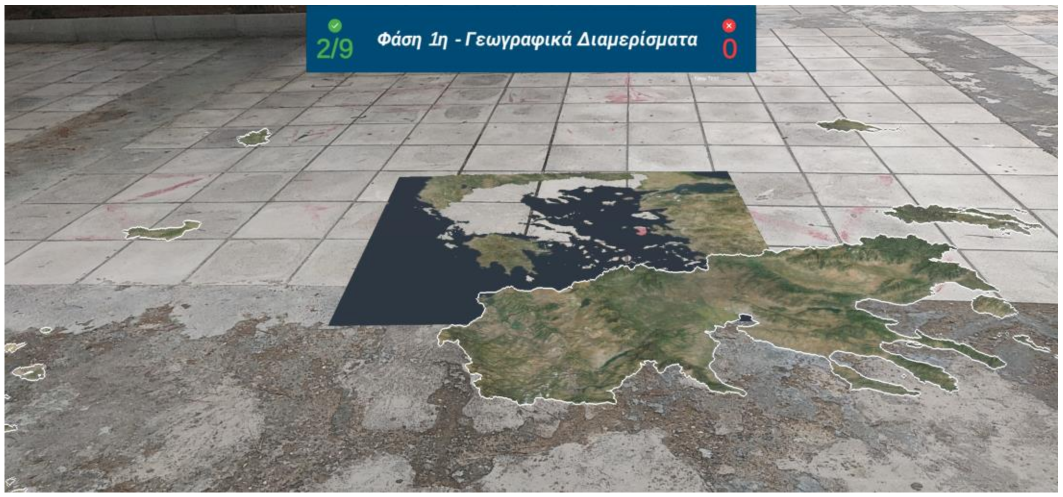


Figure 1. 3D Geography Course using AR: The Case of the Map of Greece [12].

In gaming, AR has been used to create immersive experiences that blend the real and virtual worlds. AR games can be played in the real world, with virtual elements being overlayed on the physical environment, making the gaming experience more interactive and engaging. Moreover, AR games can be played on smartphones, eliminating the need for specialized gaming devices [13].

In healthcare, AR has been used to provide medical professionals with a more comprehensive view of their patients [14]. For example, AR can be used to overlay medical images, such as x-rays or MRI scans, on the real world, providing medical professionals with a better understanding of their patients' conditions. Additionally, AR can be used to provide medical professionals with real-time information about their patients, helping them to make more informed decisions.

In conclusion, AR is a rapidly growing technology that has the potential to revolutionize various domains, including education, gaming, retail, and healthcare. This technology has the potential to provide users with more immersive and interactive experiences and has the potential to improve efficiency and productivity in various industries. With the increasing popularity of AR, it is likely that we will see more innovative and innovative applications in the future, providing new opportunities for growth and innovation in the field.

2.2 Virtual Reality

Virtual Reality (VR) is a technology that creates a simulated environment, either based on real-life or computer-generated environments [2]. It allows users to experience and interact with this environment as if they were present within it. VR provides a sense of presence and immersion using head-mounted displays (HMDs) and other sensory inputs such as audio [15], haptic feedback, and other controllers.

VR technology has been used in a variety of fields, including education, entertainment, and health care. In education, VR has been used to create immersive learning experiences, providing students with direct simulations and interactive scenarios that allow them to apply what they have learned. In entertainment, VR has been used to create immersive gaming experiences, allowing players to be fully immersed in a virtual environment. In health care, VR has been used to create virtual simulations for medical procedures and surgeries, providing training and experience for medical professionals.

Several VR systems have been developed and commercialized, including the Oculus Rift, HTC Vive, and PlayStation VR. These systems have been designed to deliver high-quality, immersive VR experiences, with advancements in hardware and software technology continually improving the VR experience.

Figure 2. VR experience with HTC Vive. [16]

2.3 Mixed Reality

Mixed Reality (MR) is a technology that blends the real and virtual worlds to create a hybrid environment [1], [2]. MR refers to a continuum that ranges from AR, which enhances the real world with virtual objects, to VR, which replaces the real world with a virtual one. It provides a seamless blend of the physical and digital worlds, allowing users to interact with virtual objects as if they were real. This is achieved by using various technologies such as AR, VR, and projection mapping, which combine to create an immersive experience that blurs the line between the physical and virtual worlds.

MR has a wide range of applications in fields such as entertainment, education, and industry. In entertainment, MR has been used to create immersive gaming experiences that allow players to interact with virtual objects in the real world. In education, MR has been used to create interactive and immersive learning experiences, providing students with firsthand simulations and interactive scenarios that allow them to apply what they have learned. In industry, MR has been used to create virtual prototypes, allowing designers and engineers to evaluate and refine their designs before going into production.

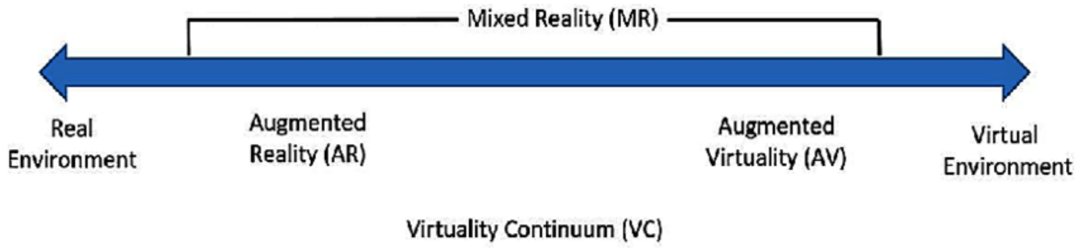


Figure 3. Virtuality Continuum (VC) – MR spectrum [2].

2.4 Extended Reality

Extended Reality (XR) is a broad term that encompasses various technologies that aim to enhance or extend human perception and experience. XR refers to a spectrum of realities that includes AR, VR, and MR, which range from augmenting the real world with virtual elements to completely immersing users in virtual environments. [17]

XR technologies have been rapidly evolving in recent years and have found applications in many areas, including entertainment, education, and industry. In entertainment, XR has been used to create immersive gaming experiences, providing players with new and exciting ways to interact with virtual objects. In education, XR has been used to create interactive and engaging learning experiences, providing students with direct simulations and interactive scenarios that help them to better understand complex concepts. In industry, XR has been used to create virtual prototypes, allowing designers and engineers to evaluate and refine their designs before going into production.

The potential of XR is vast, and its ability to enhance human perception and experience is being explored in various fields, including psychology, neuroscience, and medicine. XR has the potential to revolutionize the way we interact with the world, and its continued development will lead to new and exciting applications in the future.

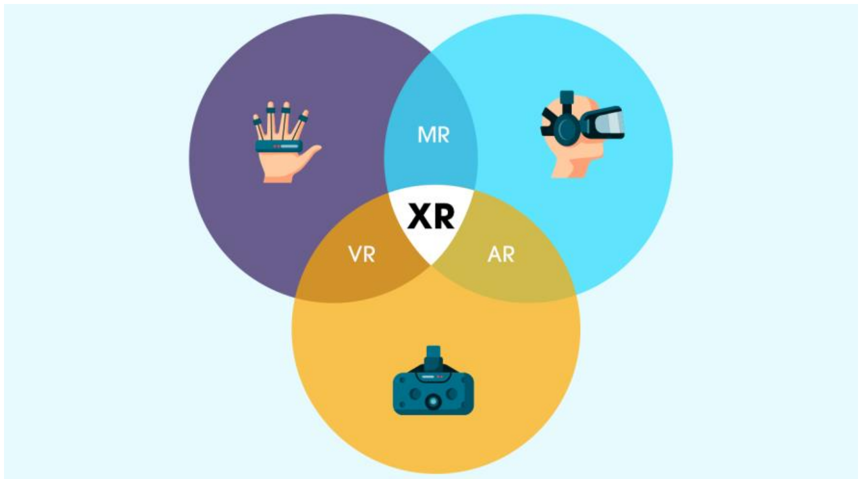


Figure 4. Extended Reality spectrum visualization [18].

Chapter 3 - Similar Work

Augmented Reality technology has seen a significant increase in popularity and adoption in recent years, with a wide range of applications across various industries. As mentioned previously, AR is a technology that allows for the overlay of virtual objects on the real world, creating an immersive and interactive experience for users. It comes with significant use in in gaming, where players can use AR-enabled devices to interact with virtual game elements in the physical world. Also, in education, by allowing the creation of interactive educational content, making learning more engaging and effective and more.

The use of AR technology has rapidly expanded in recent years and its applications are only set to increase in the future. As the technology continues to evolve and improve, we can expect to see even more innovative uses for AR in a wide range of industries. Further down, it will be a short mention of today's state of art in AR tools and applications already published, by pointing out each system's key points and objectives.

3.1 ZeusAR

ZeusAR is a software system for developing Augmented Reality Serious Games (ARSGs) that involves three phases: analysis, configuration, and generation [6]. The analysis phase involves examining the standard project structure of a typical serious game and identifying the actions that AR content can be added to. The configuration phase involves configuring the AR features and library to be integrated into the game structure, and the generation phase involves inserting the AR code and necessary files into the game structure. The ZeusAR process is implemented through a software architecture and is not dependent on specific software development technologies or programming languages.

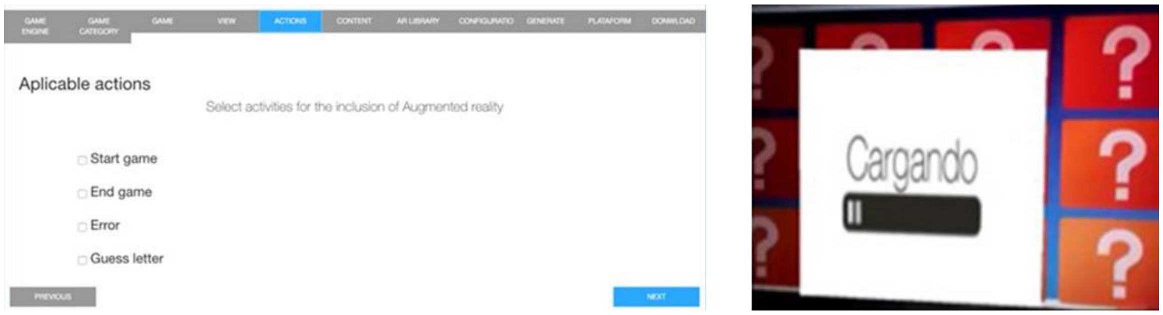


Figure 5. Left side - the ZeusAR toolset. Right side – the generated AR game [6].

3.2 A Pervasive Augmented Reality Serious Game

The game is a racing game where the objective is to start the car and move around the track without colliding with objects [19]. It is a pervasive augmented reality serious game aimed at enhancing entertainment through a multimodal tracking interface. The research focuses on designing and implementing user-friendly interfaces that can be used by a wide range of users, including people with disabilities. Users can interact with the game through a pinch glove, a Wiimote, tangible objects, and I/O controls of a UMPC (keyboard/mouse). Initial evaluation results show that multimodal-based interaction in serious games can be beneficial.

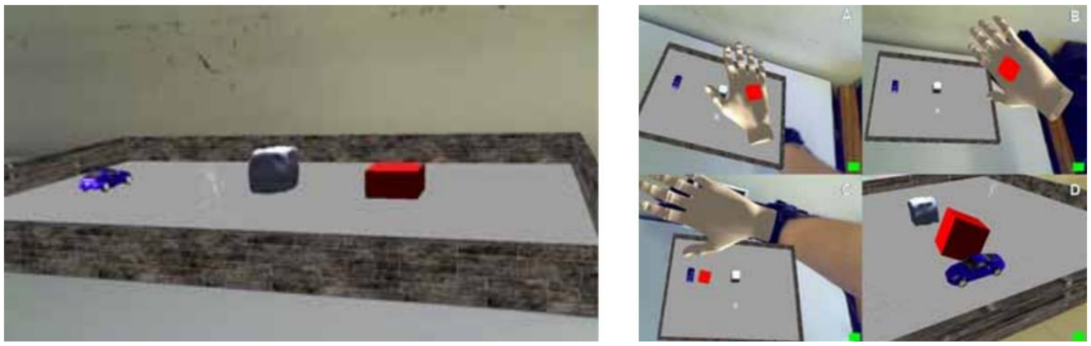


Figure 6. Pervasive Augmented Reality Serious Game – AR Racing Game [19].

3.3 AR-Maze

AR-Maze is a cutting-edge educational tool that leverages augmented reality to **instruct** children about computational thinking **[20]**. It offers real-time feedback on the physical world while maintaining a fun and cost-effective learning environment. Children can create their own programs by arranging programming blocks and testing their code using a mobile device. With AR-Maze, they will have the opportunity to learn essential programming concepts such as parameters, loop logic, debugging, and more. The tool has been developed, and a preliminary user study has been conducted to evaluate its effectiveness and guide future design improvements. The aim of AR-Maze is to provide children with an enjoyable and intuitive way to learn **programming.**

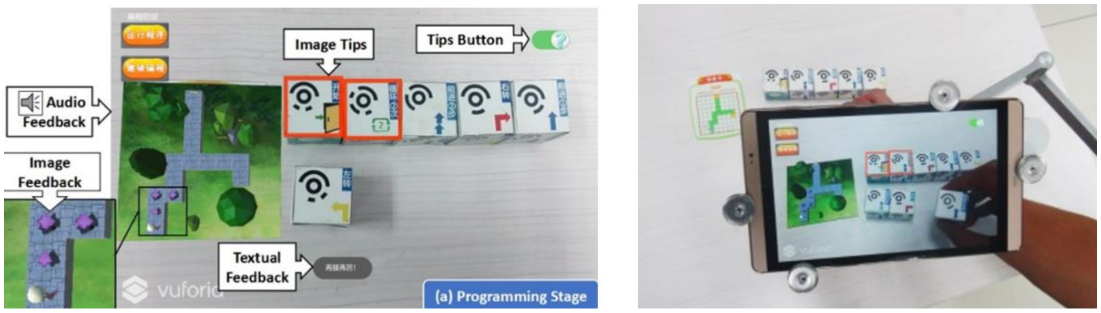


Figure 7. AR-Maze programming tool using augmented reality technology [20].

3.4 ComposAR

ComposAR is a tool aimed at making AR and MR application creation accessible to a broad audience [21]. It is unique in that it offers a combination of visual programming and scripting options, as well as real-time testing. Being written in Python, the user interface and runtime can be easily modified and additional modules from external sources can be integrated into the authoring process. ComposAR distinguishes itself from other AR authoring tools by offering multiple levels of interaction.

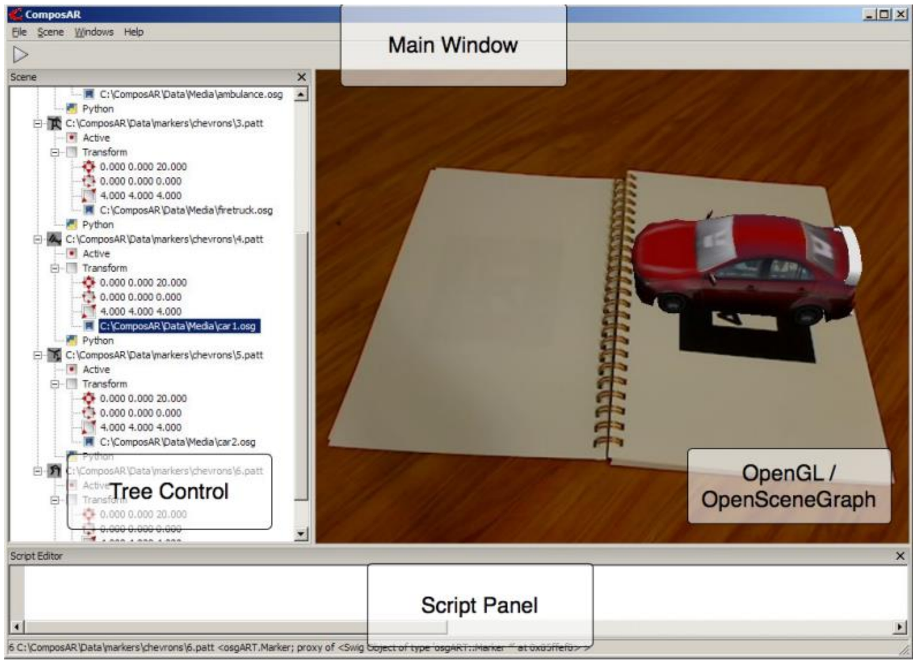
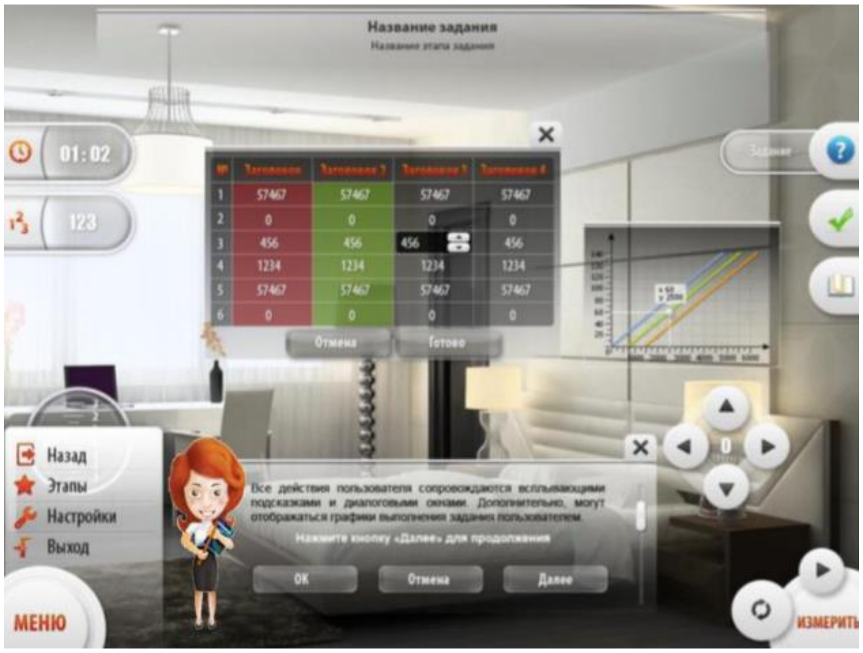


Figure 8. ComposAR interface components [21].

3.5 Interactive Educational Content Based on Augmented Reality and 3D Visualization

This tool uses AR and 3D Visualization, specifically for educational reasons, since its goal is to motivate students from secondary education to be more creative and conduct research on their subjects, by interacting with the final application in an immersive way [22]. The produced interactive education content (ICE) consists of several software and hardware resources, such as texts, audio, video, virtual 3D objects and more. To make the ICE even more appealing to the students, a few 2D animated virtual persons were added into the User Interface (UI).



№	Заголовок 1	Заголовок 2	Заголовок 3	Заголовок 4
1	57467	57467	57467	57467
2	0	0	0	0
3	456	456	456	456
4	1234	1234	1234	1234
5	57467	57467	57467	57467
6	0	0	0	0

Все действия пользователя сопровождаются всплывающими подсказками и диалоговыми окнами. Дополнительно, могут отображаться графики выполнения задания пользователем.

Нажмите кнопку «Далее» для продолжения

Figure 9. ICE example for laboratory work [22].

Chapter 4 - Software Design

The whole thesis project is implemented in C# with the help of Unity Game Engine. Since Unity gives the option to create a custom editor and add any functionalities the developer wants, it supports all the features required for the toolset's implementation. Of course, all the actions to be made, are well-combined with AR scenes that are under development. This way, the use of the toolset will help the developers to create even faster as many custom objects they need according to their use case applications. Before the implementation of the toolset, it will be necessary to analyze and design the components, classes and methods that will be further developed.

4.1 Main Components

The main components of the toolset will be visualized in Unified Modeling Language (UML) component diagram [23], [24]. It is a visual representation of the components, interfaces, inputs/outputs, and relationships in a software system, used for design and communication. In Figure 10, you can see all the essential components for the implementation of the toolset, which will be described down below.

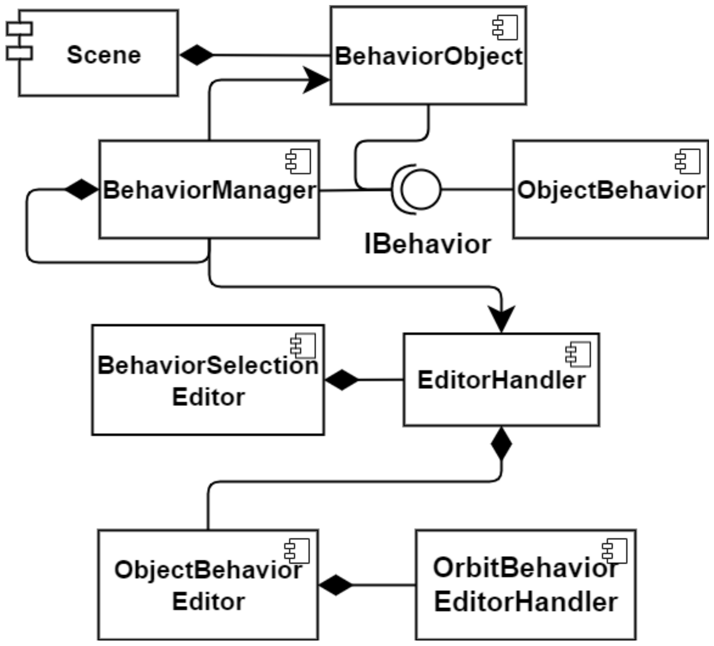


Figure 10. UML Component Diagram.

The whole design of the toolset consists of nine main components:

1. Scene*:* The active scene in Unity Game Engine, where all the custom objects to be created with the toolset's help, will be instantiated and used.

2. Behavior Object*:* The script of a game object that keeps all the necessary data, properties and information of any object created from the toolset. Its name is "Custom Object" in the actual implementation.

3. Behavior Manager: It is responsible for the management of all the available behaviors that the toolset supports.

4. IBehavior*:* An interface implemented by all the objects created with the toolset and by all the implemented behaviors mechanics.

5. Object Behavior*:* The implementation of each movement behavior to be used in the toolset.

6. Behavior Selection Editor*:* The custom editor to be used inside Unity's environment. It is responsible to get user's input, that have to do with the preferred behavior of the object to be created.

7. Editor Handler*:* It is responsible for the switch from one editor to another.

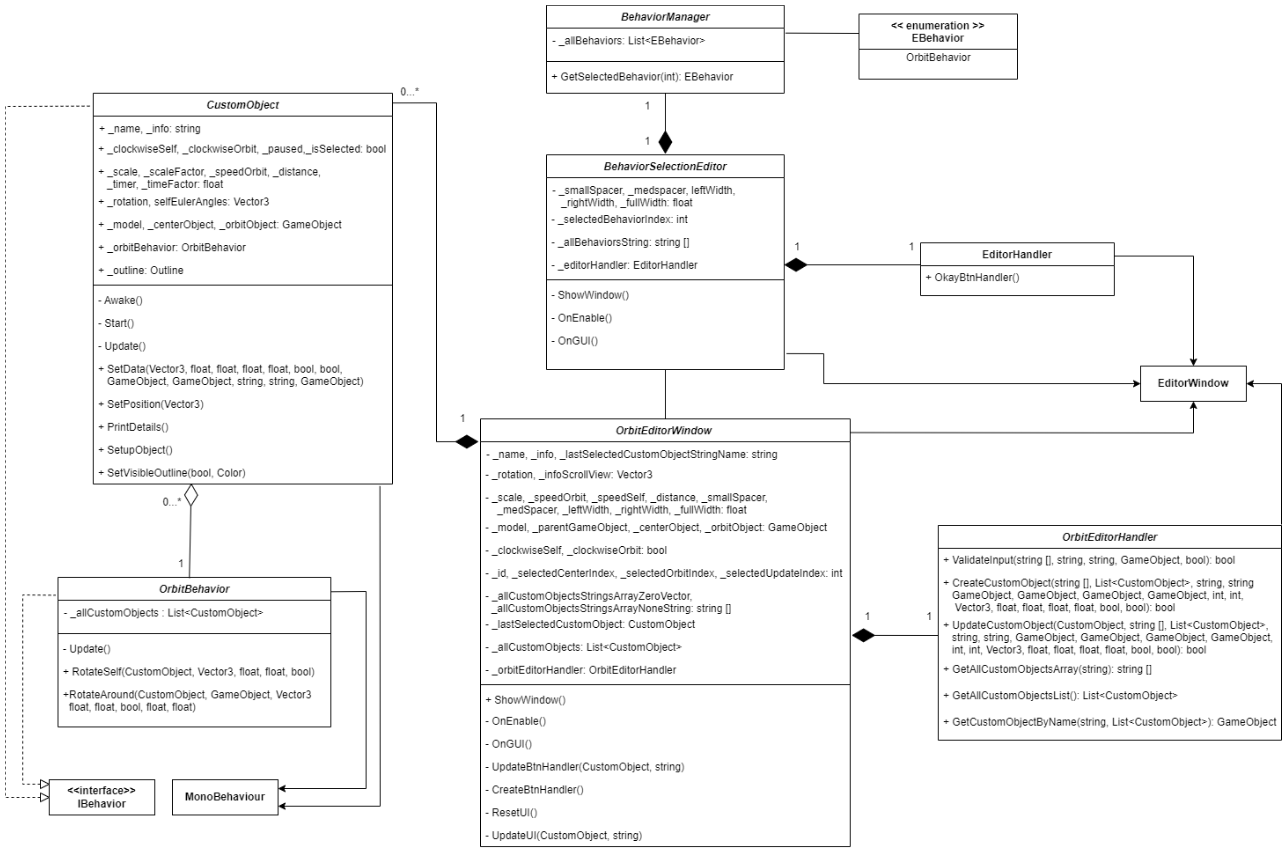
8. Object Behavior Editor*:* The specific custom editor, which is enabled after the preferred behavior from the user. It includes all the essential settings and data used for the custom object's behavior to be created. Its name is "Orbit Editor Window" in the actual implementation of the toolset.

9. Orbit Behavior Editor Handler*:* It is responsible for all the functionalities and processes required for the creation of the custom objects with orbit behavior as selection.

4.2 Classes and Methods

UML class diagrams are a type of diagram used in software engineering to model the structure of a system by representing the classes, objects, and relationships between objects [24]. They can be used for requirements gathering and analysis, design, implementation, and maintenance and evolution, providing a visual representation of the structure of a system and making it easier to understand and communicate the design. UML class diagrams typically include class names, attributes, and methods, and can represent real-world objects, abstract concepts, or software components, helping to ensure that the implementation matches the design.

In Figure 11, it is in-depth visualized all the classes, attributes, methods, and relationships of the proposed toolset. An object-oriented approach was followed, making the developing process much clearer and easier. This way, the final result of the toolset's implementation is extendable and effortlessly understandable from users/developer who want to use it for their own immersive AR experiences. Each class, represented in the class diagram below, is already explained in section 4.1. Each one of them, has their own attributes, methods, and relationships, which are also included in the class diagram.



BehaviorManager
- _allBehaviors: List<EBehavior>
+ GetSelectedBehavior(int): EBehavior
<< enumeration >>
EBehavior
OrbitBehavior
1
1
BehaviorSelectionEditor
- _smallSpacer, _medspacer, leftWidth, _rightWidth, _fullWidth: float
- _selectedBehaviorIndex: int
- _allBehaviorsString: string []
- _editorHandler: EditorHandler
- ShowWindow()
- OnEnable()
- OnGUI()
1
1
EditorHandler
+ OkayBtnHandler()
EditorWindow
CustomObject
0...*
+ _name, _info: string
+ _clockwiseSelf, _clockwiseOrbit, _paused,_isSelected: bool
+ _scale, _scaleFactor, _speedOrbit, _distance, _timer, _timeFactor: float
+ _rotation, selfEulerAngles: Vector3
+ _model, _centerObject, _orbitObject: GameObject
+ _orbitBehavior: OrbitBehavior
+ _outline: Outline
- Awake()
- Start()
- Update()
+ SetData(Vector3, float, float, float, float, bool, bool, GameObject, GameObject, string, string, GameObject)
+ SetPosition(Vector3)
+ PrintDetails()
+ SetupObject()
+ SetVisibleOutline(bool, Color)
0...*
1
OrbitBehavior
- _allCustomObjects : List<CustomObject>
- Update()
+ RotateSelf(CustomObject, Vector3, float, float, bool)
+RotateAround(CustomObject, GameObject, Vector3 float, float, bool, float, float)
<<interface>>
IBehavior
MonoBehaviour
1
OrbitEditorWindow
- _name, _info, _lastSelectedCustomObjectStringName: string
- _rotation, _infoScrollView: Vector3
- _scale, _speedOrbit, _speedSelf, _distance, _smallSpacer, _medSpacer, _leftWidth, _rightWidth, _fullWidth: float
- _model, _parentGameObject, _centerObject, _orbitObject: GameObject
- _clockwiseSelf, _clockwiseOrbit: bool
- _id, _selectedCenterIndex, _selectedOrbitIndex, _selectedUpdateIndex: int
- _allCustomObjectsStringsArrayZeroVector, _allCustomObjectsStringsArrayNoneString: string []
- _lastSelectedCustomObject: CustomObject
- _allCustomObjects: List<CustomObject>
- _orbitEditorHandler: OrbitEditorHandler
+ ShowWindow()
- OnEnable()
- OnGUI()
- UpdateBtnHandler(CustomObject, string)
- CreateBtnHandler()
- ResetUI()
- UpdateUI(CustomObject, string)
1
1
OrbitEditorHandler
+ ValidateInput(string [], string, string, GameObject, bool): bool
+ CreateCustomObject(string [], List<CustomObject>, string, string GameObject, GameObject, GameObject, GameObject, int, int, Vector3, float, float, float, float, bool, bool): bool
+ UpdateCustomObject(CustomObject, string [], List<CustomObject>, string, string, GameObject, GameObject, GameObject, GameObject, int, int, Vector3, float, float, float, float, bool, bool): bool
+ GetAllCustomObjectsArray(string): string []
+ GetAllCustomObjectsList(): List<CustomObject>
+ GetCustomObjectByName(string, List<CustomObject>): GameObject

Figure 11. UML Class Diagram.

4.3 Toolset's Sequence

UML sequence diagrams are a type of diagram used in software engineering to represent the interactions between objects or components in a system over time [24], [25]. They depict the order in which messages are sent and received between objects, and are used to model the behavior of a system. A UML sequence diagram shows the objects involved in the interaction, along with the messages they exchange, arranged in a timeline format. The messages are represented as arrows pointing from the sender object to the receiver object, and can include the name of the message, the parameters passed, and any return values.

Firstly, the user has to access the toolset through Unity’s toolbar and the first custom editor will appear. After that, he needs to select one from all the available object behaviors, that are supported from the toolset (only the orbit behavior is currently implemented). Then, a second custom editor (the selected behavior editor) will be shown in the screen, waiting for the user to give all the essential input, to configure the desired custom object that will be created or updated. After that, with just one click of a button, the new custom object is created in Unity’s hierarchy of the currently active scene. In Figure 12, it is pictured the UML sequence diagram of the toolset, where all the steps and possible scenarios according to user’s actions, are in-depth described and visualized.

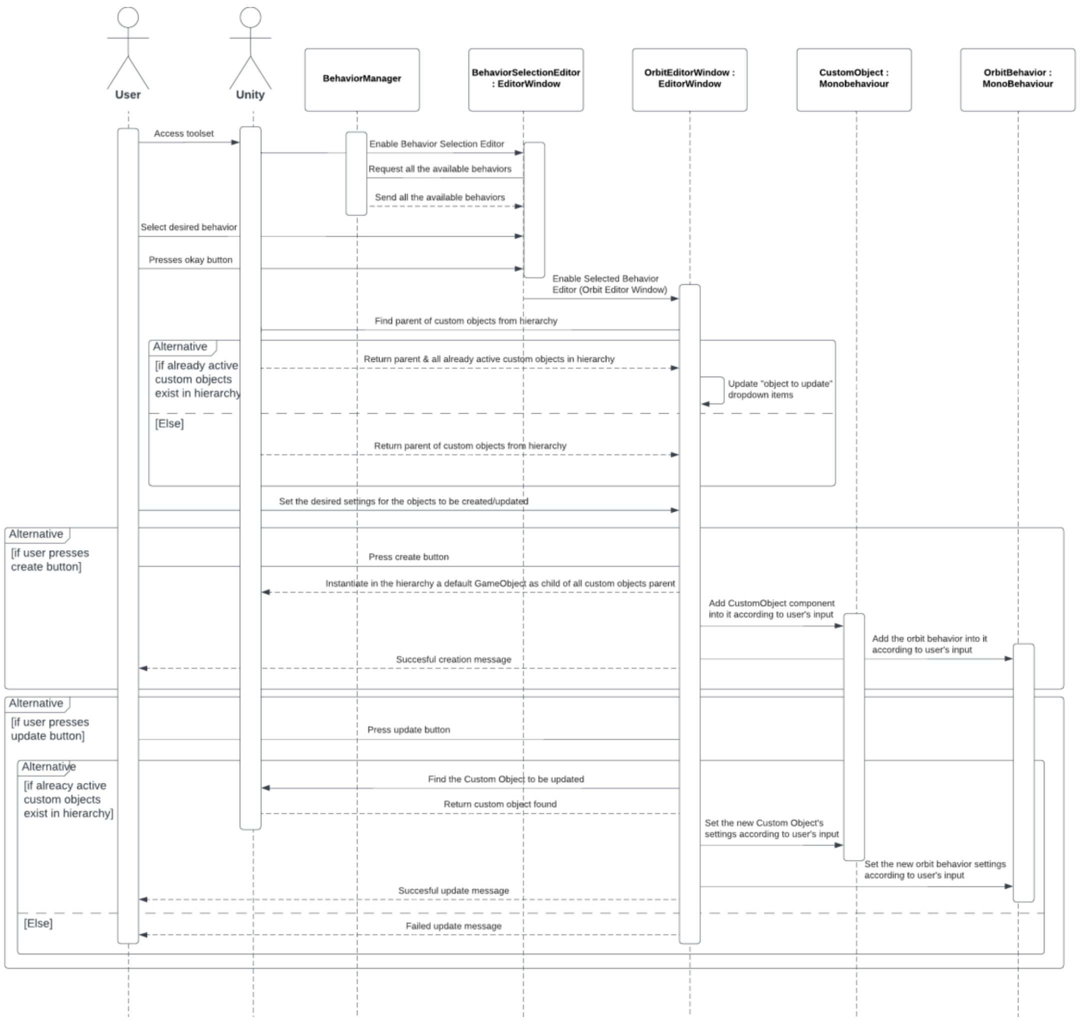

User
Unity
BehaviorManager
BehaviorSelectionEditor : EditorWindow
OrbitEditorWindow : EditorWindow
CustomObject : Monobehaviour
OrbitBehavior : MonoBehaviour
Access toolset
Enable Behavior Selection Editor
Request all the available behaviors
Send all the available behaviors
Select desired behavior
Presses okay button
Enable Selected Behavior Editor (Orbit Editor Window)
Find parent of custom objects from hierarchy
Alternative
[if already active custom objects exist in hierarchy
Return parent & all already active custom objects in hierarchy
Update "object to update" dropdown items
[Else]
Return parent of custom objects from hierarchy
Set the desired settings for the objects to be created/updated
Alternative
[if user presses create button]
Press create button
Instantiate in the hierarchy a default GameObject as child of all custom objects parent
Add CustomObject component into it according to user's input
Add the orbit behavior into it according to user's input
Succesful creation message
Alternative
[if user presses update button]
Press update button
Alternative
[if alreacy active custom objects exist in hierarchy]
Find the Custom Object to be updated
Return custom object found
Set the new Custom Object's settings according to user's input
Set the new orbit behavior settings according to user's input
Succesful update message
[Else]
Failed update message

Figure 12. UML Sequence Diagram.

Chapter 5 - System Implementation

5.1 Technologies and Frameworks

5.1.1 Unity

Unity is a cross-platform game engine developed by Unity Technologies. It is widely used in the game industry to create 2D and 3D games for a variety of platforms, including PC, console, mobile, and web [8]. Unity is also used in various non-game contexts, such as architecture, engineering, and film. In addition to its traditional game development capabilities, Unity strongly supports AR and VR development. Unity is a comprehensive game development platform that includes a powerful game engine, a visual scripting system, a 2D and 3D graphics engine, audio and animation tools, and a range of other features. It is designed to be user-friendly and easy to use, making it accessible to both novice and experienced game developers.

One of the main features of Unity is its game engine, which is responsible for rendering graphics, handling input, and managing game logic. The Unity game engine is built on top of the Mono .NET framework, which provides a range of features such as garbage collection, threading, and networking support. The game engine is optimized for performance and scalability, making it suitable for developing games of all sizes and complexity. In addition to the game engine, Unity also includes a visual scripting system called UnityScript, which is a node-based system that allows developers to create and modify game logic without writing code. It is designed to be user-friendly and easy to use, making it accessible to developers who may not have programming experience.

Unity also includes a powerful 2D and 3D graphics engine that allows developers to create high-quality graphics and special effects. The graphics engine is based on the Shaderlab language, which enables developers to create custom shaders and materials. Unity also includes a range of built-in lighting and shadowing systems and supports various rendering techniques, such as deferred rendering and screen space reflections.

Unity also provides a range of tools and features for developing AR applications. These tools include support for ARKit and ARCore, the AR platforms supplied by Apple and Google, respectively. In addition to these platform-specific tools, Unity also offers a range of cross-platform AR tools, such as the AR Foundation package, which allows developers to build AR applications that can be deployed to multiple platforms. Unity's AR tools also support features such as image tracking, object recognition, and 3D object placement. These tools make it easy for developers to build AR experiences that are interactive, engaging, and immersive.

In addition to its graphics engine, Unity includes various audio and animation tools. The audio tools allow developers to create and manage sound effects and music, while the animation tools allow developers to create and edit character and object animations. Unity also includes a range of built-in physics and AI systems, networking, and multiplayer gameplay support.

5.1.2 Blender

Blender is a free and open-source 3D computer graphics software tool for creating animated films, visual effects, art, 3D printed models, motion graphics, interactive 3D applications, and computer games [27]. Blender is developed and maintained by the Blender Foundation, a non-profit organization that provides open-source software tools for the 3D graphics community. Blender is a comprehensive 3D graphics software suite that includes a range of tools for modeling, shading, texturing, animating, and rendering 3D graphics. It is designed to be user-friendly and easy to use, making it accessible to both novice and experienced 3D graphics artists.

One of the main features of Blender is its 3D modeling toolset, which allows users to create and manipulate 3D shapes and objects. The modeling toolset includes features like mesh modeling, curve modeling, sculpting, and texture mapping. It has many advanced features, such as multiresolution sculpting, dynamic topology sculpting, and parametric modeling. These tools allow users to create 3D models, from simple shapes to complex, detailed models.

In addition to its modeling tools, Blender also includes a range of shading and texturing tools. These tools allow users to apply materials and textures to their 3D models and control the appearance of the models in the final rendered image. Blender's shading and texturing tools include support for a variety of techniques, such as bump mapping, specular mapping, and ambient occlusion. These tools allow users to create natural, high-quality materials and textures for their 3D models.

Blender also includes a powerful animation toolset that allows users to create and edit character and object animations. The animation toolset includes keyframe animation, inverse kinematics, and constraints. It also consists of a non-linear animation editor, which allows users to create complex animations by manipulating a series of keyframes. These tools enable users to create a wide range of animations, from simple character movements to complex, multi-layered animations.

Also, Blender includes a highly efficient rendering engine that allows users to create high-quality, photorealistic images and animations. The rendering engine supports various features, such as global illumination, ambient occlusion, and subsurface scattering. It also includes support for a variety of rendering techniques, such as path tracing, bidirectional path tracing, and final gathering. These features allow users to create highly realistic, lifelike 3D graphics.

In addition to its core features, Blender also includes a range of tools and features that are useful for 3D graphics artists. These include support for compositing, video editing, and 2D animation, as well as a range of third-party add-ons and plugins that can extend the software's functionality. Overall, Blender is a comprehensive and powerful 3D graphics software suite that is widely used in a variety of industries, including film and video, game development, architecture, and product design. It is free and open source, making it an affordable and accessible option for 3D graphics artists of all skill levels.

5.2 Tool Implementation

5.2.1 SOLID

SOLID is a mnemonic acronym that stands for five principles of object-oriented software design [26]. Robert C. Martin first proposed these principles in his book "Agile Software Development, Principles, Patterns, and Practices." The SOLID principles are intended to guide software designers in developing software systems that are easy to maintain and extend over time. They are widely recognized as best practices in the field of software engineering and are often cited as fundamental principles of good design.

The first principle of SOLID is the *Single Responsibility Principle (SRP).* This principle states that a class should have only one reason to change. In other words, a class should have a single, well-defined responsibility and not be responsible for multiple unrelated tasks. Adhering to the SRP can help reduce a software system's complexity and make it easier to understand and maintain.

The second principle of SOLID is the *Open/Closed Principle (OCP).* This principle states that software components should be open for extension but closed for modification. In other words, a member should be designed so that it can be extended to support new functionality without requiring changes to its existing code. This can help to reduce the risk of introducing new bugs or breaking existing functionality when adding new features to a software system.

The third principle of SOLID is the *Liskov Substitution Principle (LSP).* This principle states that objects of a subclass should be able to be used in the same way as objects of the parent class. In other words, a subclass should be a substitution for its parent class without altering the correctness of the program. Adhering to the LSP can help to ensure that a software system is well-structured and maintainable over time.

The fourth principle of SOLID is the *Interface Segregation Principle (ISP).* This principle states that clients should not be forced to depend on interfaces they do not use. In other words, interfaces should be designed to minimize the number of methods a client needs to be aware of. Adhering to the ISP can help reduce a software system's complexity and make it easier to understand and maintain.

The fifth and final principle of SOLID is *the Dependency Inversion Principle (DIP).* This principle states that high-level modules should not depend on low-level modules, but rather both should depend on abstractions. In other words, software components should depend on abstractions rather than concrete implementations. This can help to reduce the coupling between components and make a software system more flexible and easier to maintain.

In summary, the SOLID principles are a set of best practices for object-oriented software design. They are intended to guide software designers in developing software systems that are easy to maintain and extend over time. The SOLID principles are widely recognized as key principles of good design and are often cited as best practices in software engineering. Adhering to the SOLID principles can help to ensure that a software system is well-structured, maintainable, and flexible over time.

5.2.2 Unity Editors Implementation

Custom Unity Editors specialized in AR can be used to help build custom 3D objects for AR experiences. These custom editors provide a user-friendly interface for creating, modifying, and integrating 3D models, animations, and other AR assets. The editor's features can be tailored to fit the specific requirements of AR development, such as intuitive placement and scaling of objects in the real-world environment. Using these custom Unity Editors, students can learn how to build and design interactive AR experiences and gain hands-on experience with AR development tools. Creating custom 3D objects within the editor helps students fully realize their creative visions and add a unique touch to their AR projects. In conclusion, custom Unity Editors specialized in AR provide a valuable resource for students to build, experiment and create immersive AR experiences.

This section will analyze the two main custom editors of the toolset in more depth. First and foremost, the user should be able to select the behavior of the object that he wants to create. After that, according to the user's selection, the second editor, which is the main one, is shown, and helps the user to create or update a custom object ready to be used in an AR experience, by changing some values in a friendly and interactive way.

To access the tool, after it is imported via Unity's package manager, the user must go to Tools -> 3D Visualization in AR, and the Behavior Selection Editor will open.

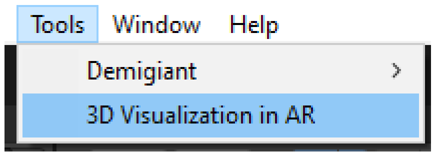
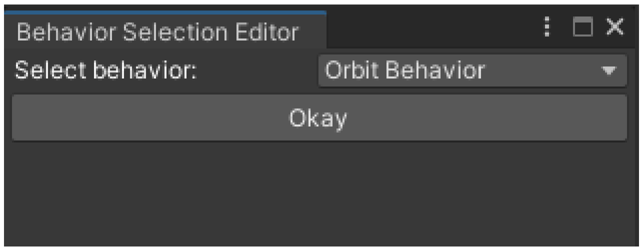


Figure 13. How to access the toolset.

5.2.2.1 Behavior Selection Editor

In this editor, the user selects the behavior of the custom objects they want to create. In the current version of the toolset, the only implemented behavior is the orbit behavior, in which custom objects are moving around in the AR space by orbiting around specific other custom objects or around themselves.



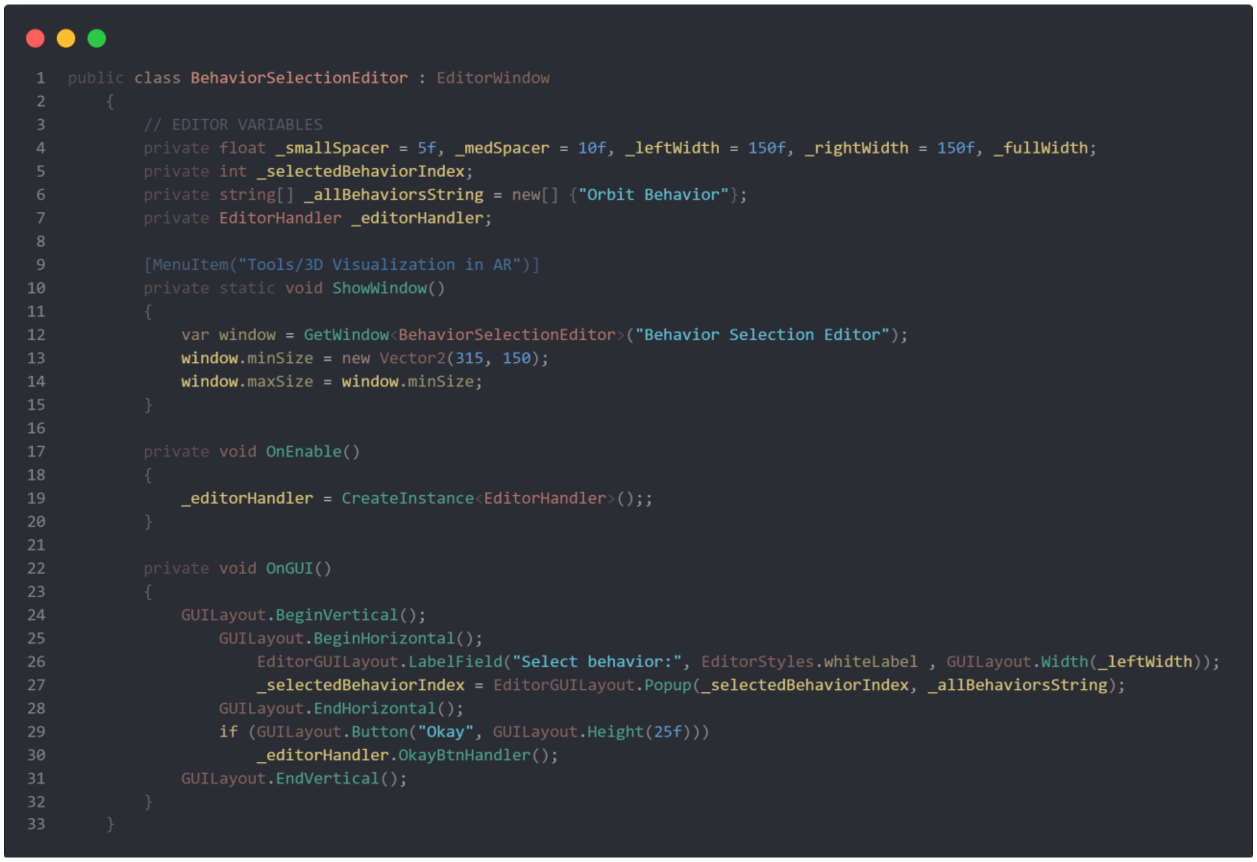
Behavior Selection Editor

Select behavior: Orbit Behavior

Okay

Figure 14. Behavior Selection Editor.

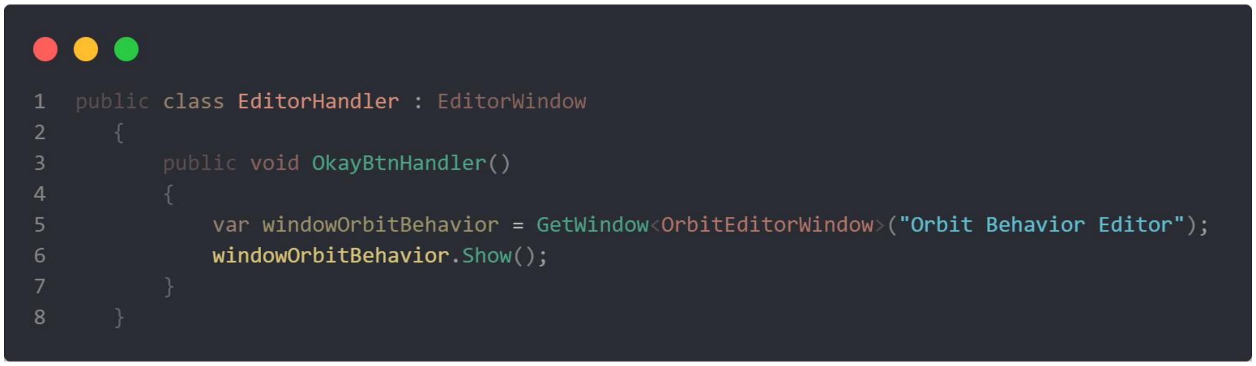
As concerns the coding part, it’s a simple Graphical User Interface (GUI) that includes a simple label, a drop-down menu, and a button to confirm the userdesirable behavior of the objects.



```
public class BehaviorSelectionEditor : EditorWindow
    {
        // EDITOR VARIABLES
        private float _smallSpacer = 5f, _medSpacer = 10f, _leftWidth = 150f, _rightWidth = 150f, _fullWidth;
        private int _selectedBehaviorIndex;
        private string[] _allBehaviorsString = new[] {"Orbit Behavior"};
        private EditorHandler _editorHandler;

        [MenuItem("Tools/3D Visualization in AR")]
        private static void ShowWindow()
        {
            var window = GetWindow<BehaviorSelectionEditor>("Behavior Selection Editor");
            window.minSize = new Vector2(315, 150);
            window.maxSize = window.minSize;
        }

        private void OnEnable()
        {
            _editorHandler = CreateInstance<EditorHandler>();;
        }

        private void OnGUI()
        {
            GUILayout.BeginVertical();
                GUILayout.BeginHorizontal();
                    EditorGUILayout.LabelField("Select behavior:", EditorStyles.whiteLabel , GUILayout.Width(_leftWidth));
                    _selectedBehaviorIndex = EditorGUILayout.Popup(_selectedBehaviorIndex, _allBehaviorsString);
                GUILayout.EndHorizontal();
                if (GUILayout.Button("Okay", GUILayout.Height(25f)))
                    _editorHandler.OkayBtnHandler();
            GUILayout.EndVertical();
        }
    }
```

Figure 15. Behavior Selection Editor code snippet.

The EditorHandler class is responsible for showing the selected behavior editor according to the user’s choice. Since the only implemented behavior is the Orbit Behavior, it just shows the OrbitEditorWindow, which will be analyzed in depth down below.

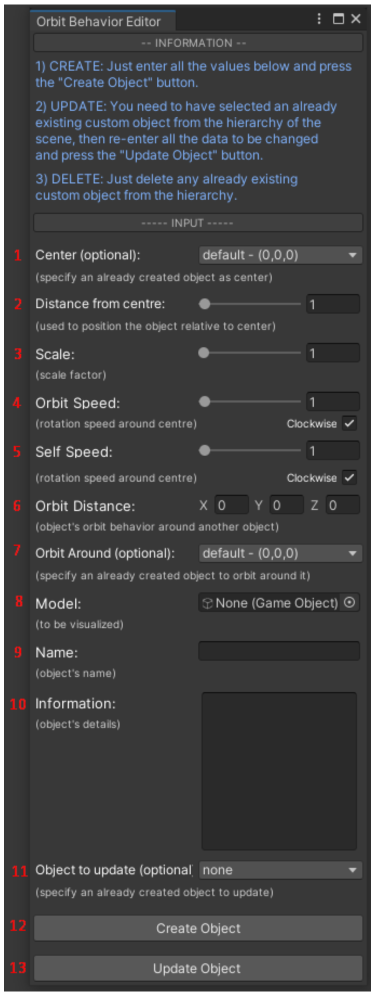


```
public class EditorHandler : EditorWindow
    {
        public void OkayBtnHandler()
        {
            var windowOrbitBehavior = GetWindow<OrbitEditorWindow>("Orbit Behavior Editor");
            windowOrbitBehavior.Show();
        }
    }
```

Figure 16. Editor Handler code snippet.

5.2.2.2 Selected Behavior Editor (Orbit Behavior Editor)

This editor is responsible for the creation of custom objects, according to the user's input. More specifically, it is assembled by thirteen required or not required fields, and each of them is a major property of the final movement of the custom object in the AR field.



Orbit Behavior Editor

-- INFORMATION --

1) CREATE: Just enter all the values below and press the "Create Object" button.

2) UPDATE: You need to have selected an already existing custom object from the hierarchy of the scene, then re-enter all the data to be changed and press the "Update Object" button.

3) DELETE: Just delete any already existing custom object from the hierarchy.

----- INPUT -----

1 Center (optional): default - (0,0,0)
(specify an already created object as center)

2 Distance from centre: 1
(used to position the object relative to center)

3 Scale: 1
(scale factor)

4 Orbit Speed: 1
(rotation speed around centre) Clockwise ✓

5 Self Speed: 1
(rotation speed around centre) Clockwise ✓

6 Orbit Distance: X 0 Y 0 Z 0
(object's orbit behavior around another object)

7 Orbit Around (optional): default - (0,0,0)
(specify an already created object to orbit around it)

8 Model: None (Game Object)
(to be visualized)

9 Name:
(object's name)

10 Information:
(object's details)

11 Object to update (optional none
(specify an already created object to update)

12 Create Object

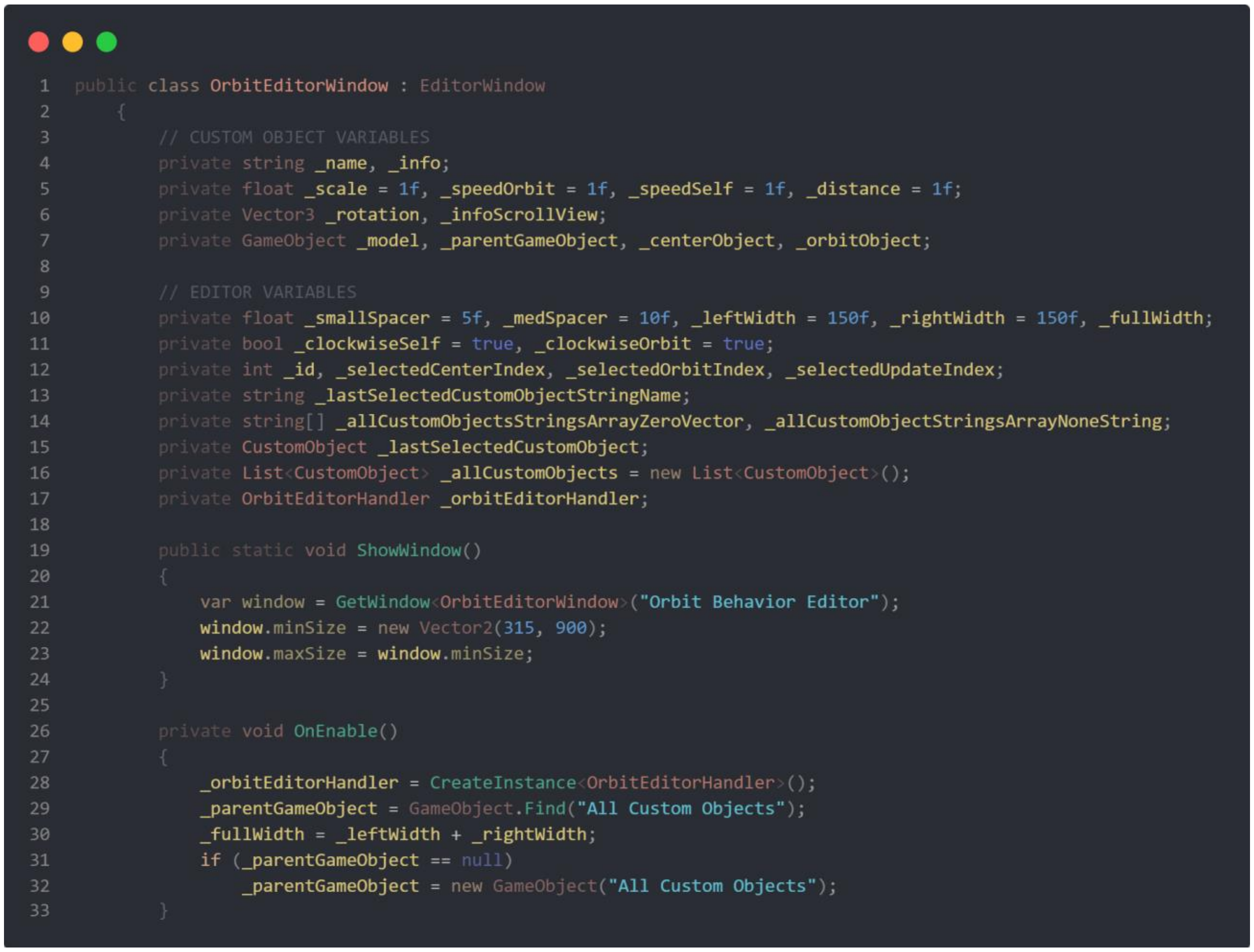
13 Update Object

Figure 17. Orbit Behavior Editor.

All the fields are described as follows:

1. **Center** *(optional):* The user specifies the anchor point of the orbit. By default, it is set to (0,0,0), which is the center of the whole composition of custom objects.
2. **Distance from center:** The user specifies the distance from the center. It is basically an offset in the X-Axis.
3. **Scale:** The user specifies the scale of the custom object.
4. **Orbit Speed:** The user specifies how fast or slow will the custom object be orbiting around the Orbit Around value, given in the seventh field of the editor. By default, the orbit around the Center direction will be clockwise, unless the user unchecks the "Clockwise" checkbox.
5. **Self Speed:** The user specifies how fast or slow will the custom object be rotating or spinning around itself. The spin direction will default to clockwise unless the user unchecks the "Clockwise" checkbox.
6. **Orbit Distance:** The user specifies the orbit behavior of the custom object, specifically for the X, Y, and Z axis. This way, the custom object can perform even a non-circular orbit.
7. **Orbit Around** *(optional):* The user specifies an already created custom object to orbit around it. By default, it is set to the center of the whole composition (0,0,0).
8. **Model**: The user specifies the model of the custom object to be created. It supports Unity's Game Object as input.
9. **Name**: The user specifies the name of the custom object to be created.
10. **Information**: The user specifies any information he wants about the custom object.
11. **Object to update** *(optional):* The user specifies an already existing custom object from the hierarchy, that he wants to update and apply the latest changes.
12. **Create Object Button:** The user clicks on it when he has finished with all the required input fields. After that, the new custom object is created in the hierarchy, and it is ready to be used for any use case.
13. **Update Object Button:** The user clicks on it when he has finished with the new input process. It is required to have already specified which already created custom object wants to apply the changes. This action is made, as previously mentioned, from the eleventh field named *Object to update*.

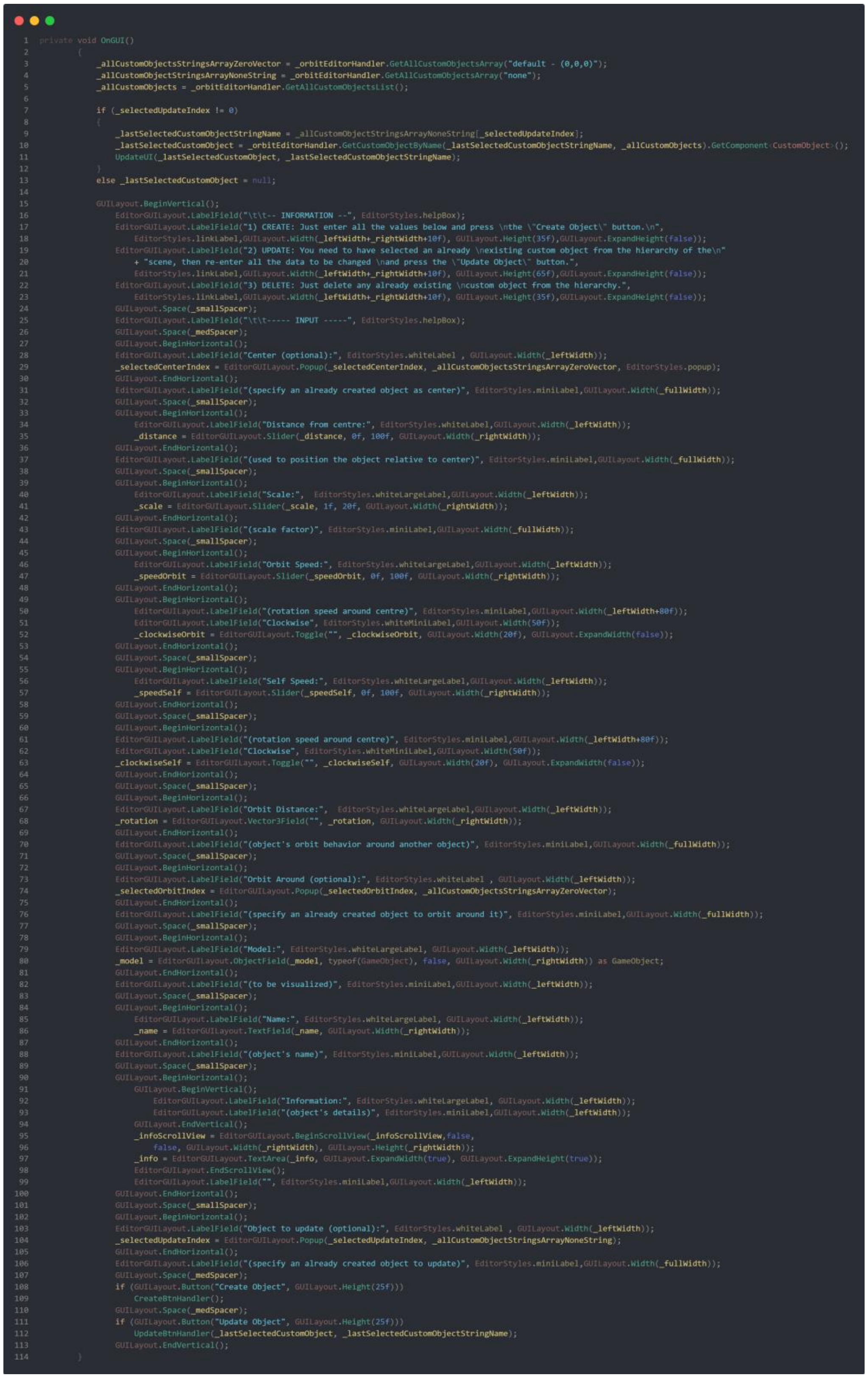
The programming part of the orbit editor window was a bit tricky since it involved the creation of the UI (labels, float fields, dropdowns, and more). In Figure 18, you can see, with more details, all the necessary variables, the ShowWindow function, and finally, the OnEnable function, where the setup of the functionalities is made, and the tool is ready to proceed with the user's input.



```
public class OrbitEditorWindow : EditorWindow
    {
        // CUSTOM OBJECT VARIABLES
        private string _name, _info;
        private float _scale = 1f, _speedOrbit = 1f, _speedSelf = 1f, _distance = 1f;
        private Vector3 _rotation, _infoScrollView;
        private GameObject _model, _parentGameObject, _centerObject, _orbitObject;

        // EDITOR VARIABLES
        private float _smallSpacer = 5f, _medSpacer = 10f, _leftWidth = 150f, _rightWidth = 150f, _fullWidth;
        private bool _clockwiseSelf = true, _clockwiseOrbit = true;
        private int _id, _selectedCenterIndex, _selectedOrbitIndex, _selectedUpdateIndex;
        private string _lastSelectedCustomObjectStringName;
        private string[] _allCustomObjectsStringsArrayZeroVector, _allCustomObjectStringsArrayNoneString;
        private CustomObject _lastSelectedCustomObject;
        private List<CustomObject> _allCustomObjects = new List<CustomObject>();
        private OrbitEditorHandler _orbitEditorHandler;

        public static void ShowWindow()
        {
            var window = GetWindow<OrbitEditorWindow>("Orbit Behavior Editor");
            window.minSize = new Vector2(315, 900);
            window.maxSize = window.minSize;
        }

        private void OnEnable()
        {
            _orbitEditorHandler = CreateInstance<OrbitEditorHandler>();
            _parentGameObject = GameObject.Find("All Custom Objects");
            _fullWidth = _leftWidth + _rightWidth;
            if (_parentGameObject == null)
                _parentGameObject = new GameObject("All Custom Objects");
        }
```

Figure 18. Orbit Editor Window code snippet – Variables, ShowWindow & OnEnable functions.

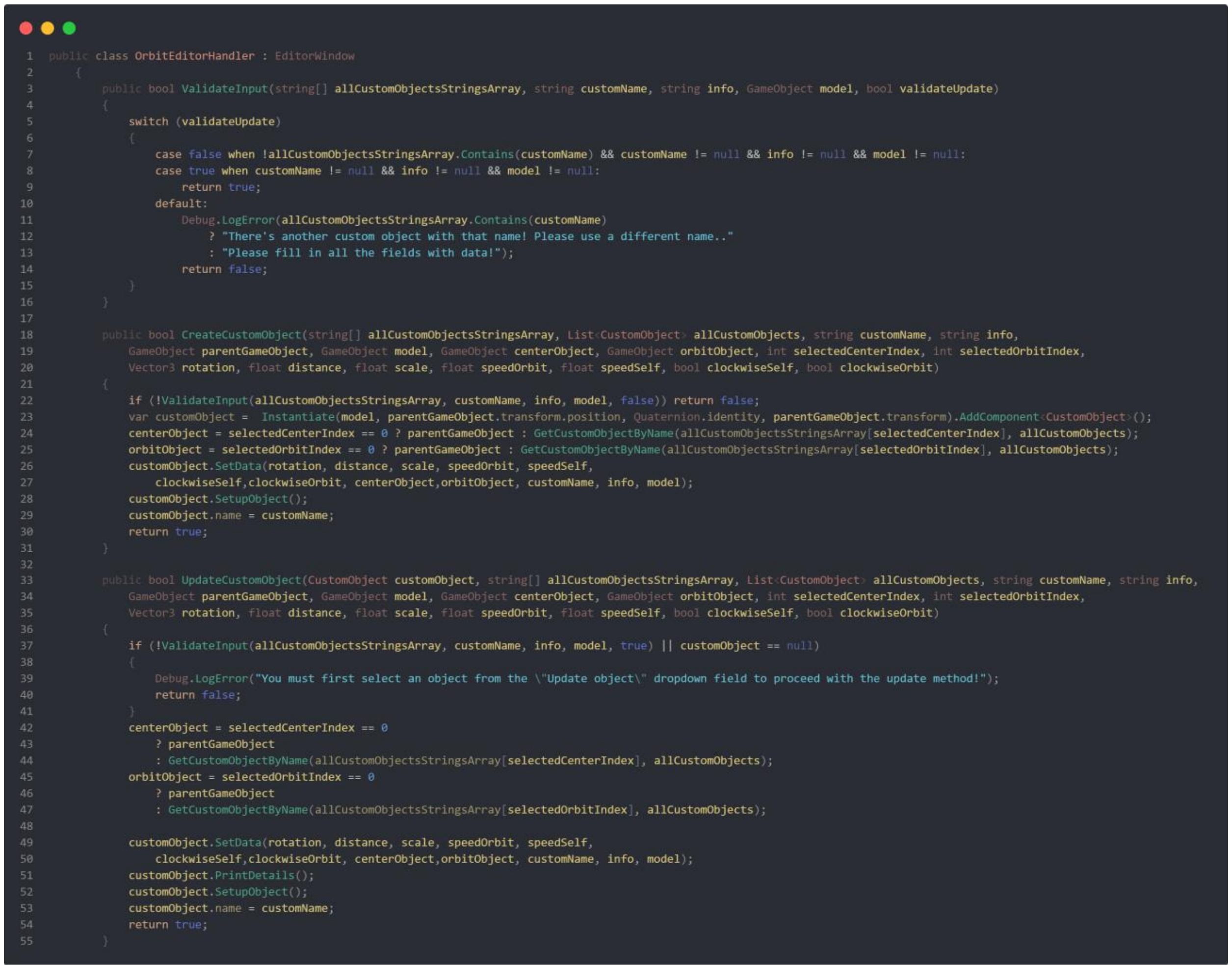
Most of the lines of code in the Orbit Editor Window script, have to do with the OnGUI function, where all the UI components are declared. More in depth, you can see all the code that assembles the final UI of the toolset, in Figure 19.



```
private void OnGUI()
        {
            _allCustomObjectsStringsArrayZeroVector = _orbitEditorHandler.GetAllCustomObjectsArray("default - (0,0,0)");
            _allCustomObjectStringsArrayNoneString = _orbitEditorHandler.GetAllCustomObjectsArray("none");
            _allCustomObjects = _orbitEditorHandler.GetAllCustomObjectsList();

            if (_selectedUpdateIndex != 0)
            {
                _lastSelectedCustomObjectStringName = _allCustomObjectStringsArrayNoneString[_selectedUpdateIndex];
                _lastSelectedCustomObject = _orbitEditorHandler.GetCustomObjectByName(_lastSelectedCustomObjectStringName, _allCustomObjects).GetComponent<CustomObject>();
                UpdateUI(_lastSelectedCustomObject, _lastSelectedCustomObjectStringName);
            }
            else _lastSelectedCustomObject = null;

            GUILayout.BeginVertical();
                EditorGUILayout.LabelField("\t\t-- INFORMATION --", EditorStyles.helpBox);
                EditorGUILayout.LabelField("1) CREATE: Just enter all the values below and press \nthe \"Create Object\" button.\n",
                    EditorStyles.linkLabel,GUILayout.Width(_leftWidth+_rightWidth+10f), GUILayout.Height(35f),GUILayout.ExpandHeight(false));
                EditorGUILayout.LabelField("2) UPDATE: You need to have selected an already \nexisting custom object from the hierarchy of the\n"
                    + "scene, then re-enter all the data to be changed \nand press the \"Update Object\" button.",
                    EditorStyles.linkLabel,GUILayout.Width(_leftWidth+_rightWidth+10f), GUILayout.Height(65f),GUILayout.ExpandHeight(false));
                EditorGUILayout.LabelField("3) DELETE: Just delete any already existing \ncustom object from the hierarchy.",
                    EditorStyles.linkLabel,GUILayout.Width(_leftWidth+_rightWidth+10f), GUILayout.Height(35f),GUILayout.ExpandHeight(false));
                GUILayout.Space(_smallSpacer);
                EditorGUILayout.LabelField("\t\t----- INPUT -----", EditorStyles.helpBox);
                GUILayout.Space(_medSpacer);
                GUILayout.BeginHorizontal();
                EditorGUILayout.LabelField("Center (optional):", EditorStyles.whiteLabel , GUILayout.Width(_leftWidth));
                _selectedCenterIndex = EditorGUILayout.Popup(_selectedCenterIndex, _allCustomObjectsStringsArrayZeroVector, EditorStyles.popup);
                GUILayout.EndHorizontal();
                EditorGUILayout.LabelField("(specify an already created object as center)", EditorStyles.miniLabel,GUILayout.Width(_fullWidth));
                GUILayout.Space(_smallSpacer);
                GUILayout.BeginHorizontal();
                    EditorGUILayout.LabelField("Distance from centre:", EditorStyles.whiteLabel,GUILayout.Width(_leftWidth));
                    _distance = EditorGUILayout.Slider(_distance, 0f, 100f, GUILayout.Width(_rightWidth));
                GUILayout.EndHorizontal();
                EditorGUILayout.LabelField("(used to position the object relative to center)", EditorStyles.miniLabel,GUILayout.Width(_fullWidth));
                GUILayout.Space(_smallSpacer);
                GUILayout.BeginHorizontal();
                    EditorGUILayout.LabelField("Scale:",  EditorStyles.whiteLargeLabel,GUILayout.Width(_leftWidth));
                    _scale = EditorGUILayout.Slider(_scale, 1f, 20f, GUILayout.Width(_rightWidth));
                GUILayout.EndHorizontal();
                EditorGUILayout.LabelField("(scale factor)", EditorStyles.miniLabel,GUILayout.Width(_fullWidth));
                GUILayout.Space(_smallSpacer);
                GUILayout.BeginHorizontal();
                    EditorGUILayout.LabelField("Orbit Speed:", EditorStyles.whiteLargeLabel,GUILayout.Width(_leftWidth));
                    _speedOrbit = EditorGUILayout.Slider(_speedOrbit, 0f, 100f, GUILayout.Width(_rightWidth));
                GUILayout.EndHorizontal();
                GUILayout.BeginHorizontal();
                    EditorGUILayout.LabelField("(rotation speed around centre)", EditorStyles.miniLabel,GUILayout.Width(_leftWidth+80f));
                    EditorGUILayout.LabelField("Clockwise", EditorStyles.whiteMiniLabel,GUILayout.Width(50f));
                    _clockwiseOrbit = EditorGUILayout.Toggle("", _clockwiseOrbit, GUILayout.Width(20f), GUILayout.ExpandWidth(false));
                GUILayout.EndHorizontal();
                GUILayout.Space(_smallSpacer);
                GUILayout.BeginHorizontal();
                    EditorGUILayout.LabelField("Self Speed:", EditorStyles.whiteLargeLabel,GUILayout.Width(_leftWidth));
                    _speedSelf = EditorGUILayout.Slider(_speedSelf, 0f, 100f, GUILayout.Width(_rightWidth));
                GUILayout.EndHorizontal();
                GUILayout.Space(_smallSpacer);
                GUILayout.BeginHorizontal();
                EditorGUILayout.LabelField("(rotation speed around centre)", EditorStyles.miniLabel,GUILayout.Width(_leftWidth+80f));
                EditorGUILayout.LabelField("Clockwise", EditorStyles.whiteMiniLabel,GUILayout.Width(50f));
                _clockwiseSelf = EditorGUILayout.Toggle("", _clockwiseSelf, GUILayout.Width(20f), GUILayout.ExpandWidth(false));
                GUILayout.EndHorizontal();
                GUILayout.Space(_smallSpacer);
                GUILayout.BeginHorizontal();
                EditorGUILayout.LabelField("Orbit Distance:",  EditorStyles.whiteLargeLabel,GUILayout.Width(_leftWidth));
                _rotation = EditorGUILayout.Vector3Field("", _rotation, GUILayout.Width(_rightWidth));
                GUILayout.EndHorizontal();
                EditorGUILayout.LabelField("(object's orbit behavior around another object)", EditorStyles.miniLabel,GUILayout.Width(_fullWidth));
                GUILayout.Space(_smallSpacer);
                GUILayout.BeginHorizontal();
                EditorGUILayout.LabelField("Orbit Around (optional):", EditorStyles.whiteLabel , GUILayout.Width(_leftWidth));
                _selectedOrbitIndex = EditorGUILayout.Popup(_selectedOrbitIndex, _allCustomObjectsStringsArrayZeroVector);
                GUILayout.EndHorizontal();
                EditorGUILayout.LabelField("(specify an already created object to orbit around it)", EditorStyles.miniLabel,GUILayout.Width(_fullWidth));
                GUILayout.Space(_smallSpacer);
                GUILayout.BeginHorizontal();
                EditorGUILayout.LabelField("Model:", EditorStyles.whiteLargeLabel, GUILayout.Width(_leftWidth));
                _model = EditorGUILayout.ObjectField(_model, typeof(GameObject), false, GUILayout.Width(_rightWidth)) as GameObject;
                GUILayout.EndHorizontal();
                EditorGUILayout.LabelField("(to be visualized)", EditorStyles.miniLabel,GUILayout.Width(_leftWidth));
                GUILayout.Space(_smallSpacer);
                GUILayout.BeginHorizontal();
                    EditorGUILayout.LabelField("Name:", EditorStyles.whiteLargeLabel, GUILayout.Width(_leftWidth));
                    _name = EditorGUILayout.TextField(_name, GUILayout.Width(_rightWidth));
                GUILayout.EndHorizontal();
                EditorGUILayout.LabelField("(object's name)", EditorStyles.miniLabel,GUILayout.Width(_leftWidth));
                GUILayout.Space(_smallSpacer);
                GUILayout.BeginHorizontal();
                    GUILayout.BeginVertical();
                        EditorGUILayout.LabelField("Information:", EditorStyles.whiteLargeLabel, GUILayout.Width(_leftWidth));
                        EditorGUILayout.LabelField("(object's details)", EditorStyles.miniLabel,GUILayout.Width(_leftWidth));
                    GUILayout.EndVertical();
                    _infoScrollView = EditorGUILayout.BeginScrollView(_infoScrollView,false,
                        false, GUILayout.Width(_rightWidth), GUILayout.Height(_rightWidth));
                    _info = EditorGUILayout.TextArea(_info, GUILayout.ExpandWidth(true), GUILayout.ExpandHeight(true));
                    EditorGUILayout.EndScrollView();
                    EditorGUILayout.LabelField("", EditorStyles.miniLabel,GUILayout.Width(_leftWidth));
                GUILayout.EndHorizontal();
                GUILayout.Space(_smallSpacer);
                GUILayout.BeginHorizontal();
                EditorGUILayout.LabelField("Object to update (optional):", EditorStyles.whiteLabel , GUILayout.Width(_leftWidth));
                _selectedUpdateIndex = EditorGUILayout.Popup(_selectedUpdateIndex, _allCustomObjectStringsArrayNoneString);
                GUILayout.EndHorizontal();
                EditorGUILayout.LabelField("(specify an already created object to update)", EditorStyles.miniLabel,GUILayout.Width(_fullWidth));
                GUILayout.Space(_medSpacer);
                if (GUILayout.Button("Create Object", GUILayout.Height(25f)))
                    CreateBtnHandler();
                GUILayout.Space(_medSpacer);
                if (GUILayout.Button("Update Object", GUILayout.Height(25f)))
                    UpdateBtnHandler(_lastSelectedCustomObject, _lastSelectedCustomObjectStringName);
                GUILayout.EndVertical();
        }
```

Figure 19. Orbit Editor Window code snippet – OnGUI function.

As concerns the actual validation, create and update operations of the editor, all of them are included in the Orbit Editor Handler script. It is responsible for the correct validation of the user’s input each time the create or the update button is pressed. In the Create Custom Object function, all the necessary actions are implemented, and the result is that a new Game Object is instantiated in Unity’s hierarchy, with all the data that the user gave as input through the toolset. As of the Update Custom Object, the same logic is also applied here, with the difference of not having to create a new custom object from scratch, but changing the properties of an already created custom object that is active in Unity’s hierarchy.

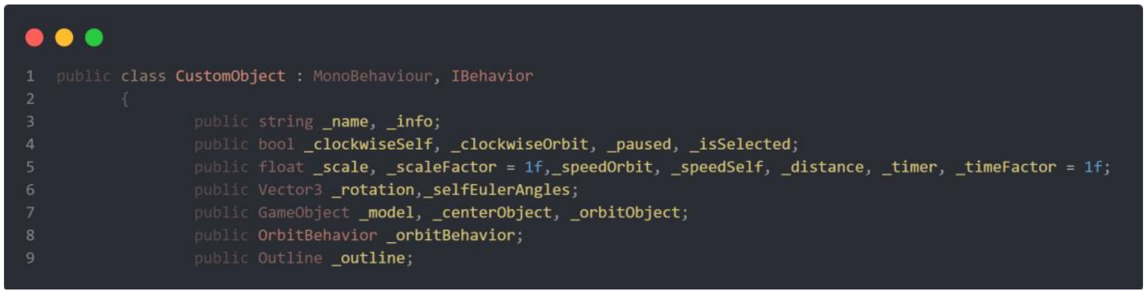


```
public class OrbitEditorHandler : EditorWindow
    {
        public bool ValidateInput(string[] allCustomObjectsStringsArray, string customName, string info, GameObject model, bool validateUpdate)
        {
            switch (validateUpdate)
            {
                case false when !allCustomObjectsStringsArray.Contains(customName) && customName != null && info != null && model != null:
                case true when customName != null && info != null && model != null:
                    return true;
                default:
                    Debug.LogError(allCustomObjectsStringsArray.Contains(customName)
                        ? "There's another custom object with that name! Please use a different name.."
                        : "Please fill in all the fields with data!");
                    return false;
            }
        }

        public bool CreateCustomObject(string[] allCustomObjectsStringsArray, List<CustomObject> allCustomObjects, string customName, string info,
            GameObject parentGameObject, GameObject model, GameObject centerObject, GameObject orbitObject, int selectedCenterIndex, int selectedOrbitIndex,
            Vector3 rotation, float distance, float scale, float speedOrbit, float speedSelf, bool clockwiseSelf, bool clockwiseOrbit)
        {
            if (!ValidateInput(allCustomObjectsStringsArray, customName, info, model, false)) return false;
            var customObject =  Instantiate(model, parentGameObject.transform.position, Quaternion.identity, parentGameObject.transform).AddComponent<CustomObject>();
            centerObject = selectedCenterIndex == 0 ? parentGameObject : GetCustomObjectByName(allCustomObjectsStringsArray[selectedCenterIndex], allCustomObjects);
            orbitObject = selectedOrbitIndex == 0 ? parentGameObject : GetCustomObjectByName(allCustomObjectsStringsArray[selectedOrbitIndex], allCustomObjects);
            customObject.SetData(rotation, distance, scale, speedOrbit, speedSelf,
                clockwiseSelf,clockwiseOrbit, centerObject,orbitObject, customName, info, model);
            customObject.SetupObject();
            customObject.name = customName;
            return true;
        }

        public bool UpdateCustomObject(CustomObject customObject, string[] allCustomObjectsStringsArray, List<CustomObject> allCustomObjects, string customName, string info,
            GameObject parentGameObject, GameObject model, GameObject centerObject, GameObject orbitObject, int selectedCenterIndex, int selectedOrbitIndex,
            Vector3 rotation, float distance, float scale, float speedOrbit, float speedSelf, bool clockwiseSelf, bool clockwiseOrbit)
        {
            if (!ValidateInput(allCustomObjectsStringsArray, customName, info, model, true) || customObject == null)
            {
                Debug.LogError("You must first select an object from the \"Update object\" dropdown field to proceed with the update method!");
                return false;
            }
            centerObject = selectedCenterIndex == 0
                ? parentGameObject
                : GetCustomObjectByName(allCustomObjectsStringsArray[selectedCenterIndex], allCustomObjects);
            orbitObject = selectedOrbitIndex == 0
                ? parentGameObject
                : GetCustomObjectByName(allCustomObjectsStringsArray[selectedOrbitIndex], allCustomObjects);

            customObject.SetData(rotation, distance, scale, speedOrbit, speedSelf,
                clockwiseSelf,clockwiseOrbit, centerObject,orbitObject, customName, info, model);
            customObject.PrintDetails();
            customObject.SetupObject();
            customObject.name = customName;
            return true;
        }
```

Figure 20. Orbit Editor Handler code snippet.

5.2.2.3 Custom Object

Every custom object has all data from the user's input, saved in variables. In Figure 21, you can see all of them that are necessary for the correct and smooth use of them in AR experiences.

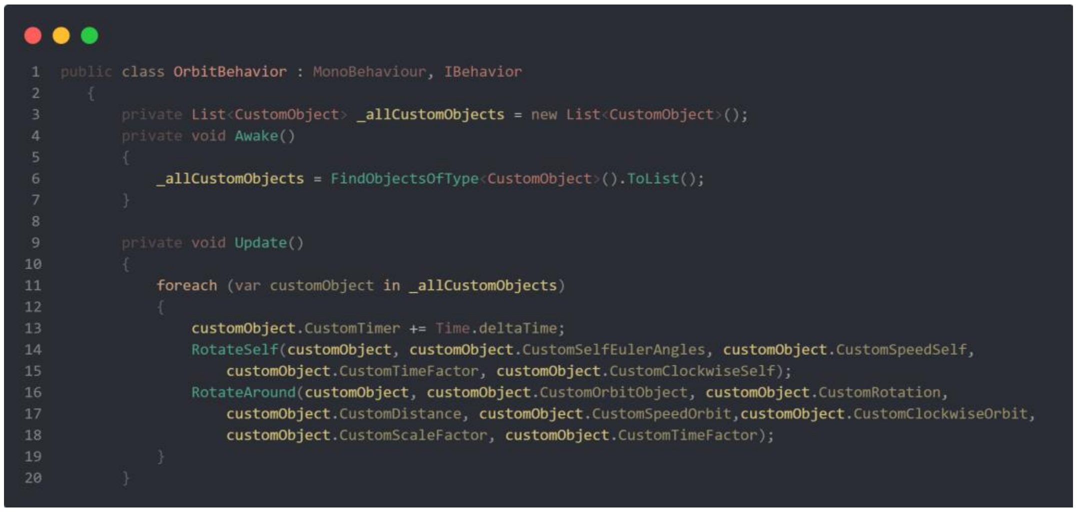


```
public class CustomObject : MonoBehaviour, IBehavior
        {
                public string _name, _info;
                public bool _clockwiseSelf, _clockwiseOrbit, _paused, _isSelected;
                public float _scale, _scaleFactor = 1f,_speedOrbit, _speedSelf, _distance, _timer, _timeFactor = 1f;
                public Vector3 _rotation,_selfEulerAngles;
                public GameObject _model, _centerObject, _orbitObject;
                public OrbitBehavior _orbitBehavior;
                public Outline _outline;
```

Figure 21. Custom Object code snippet.

5.2.2.4 Orbit Behavior Implementation

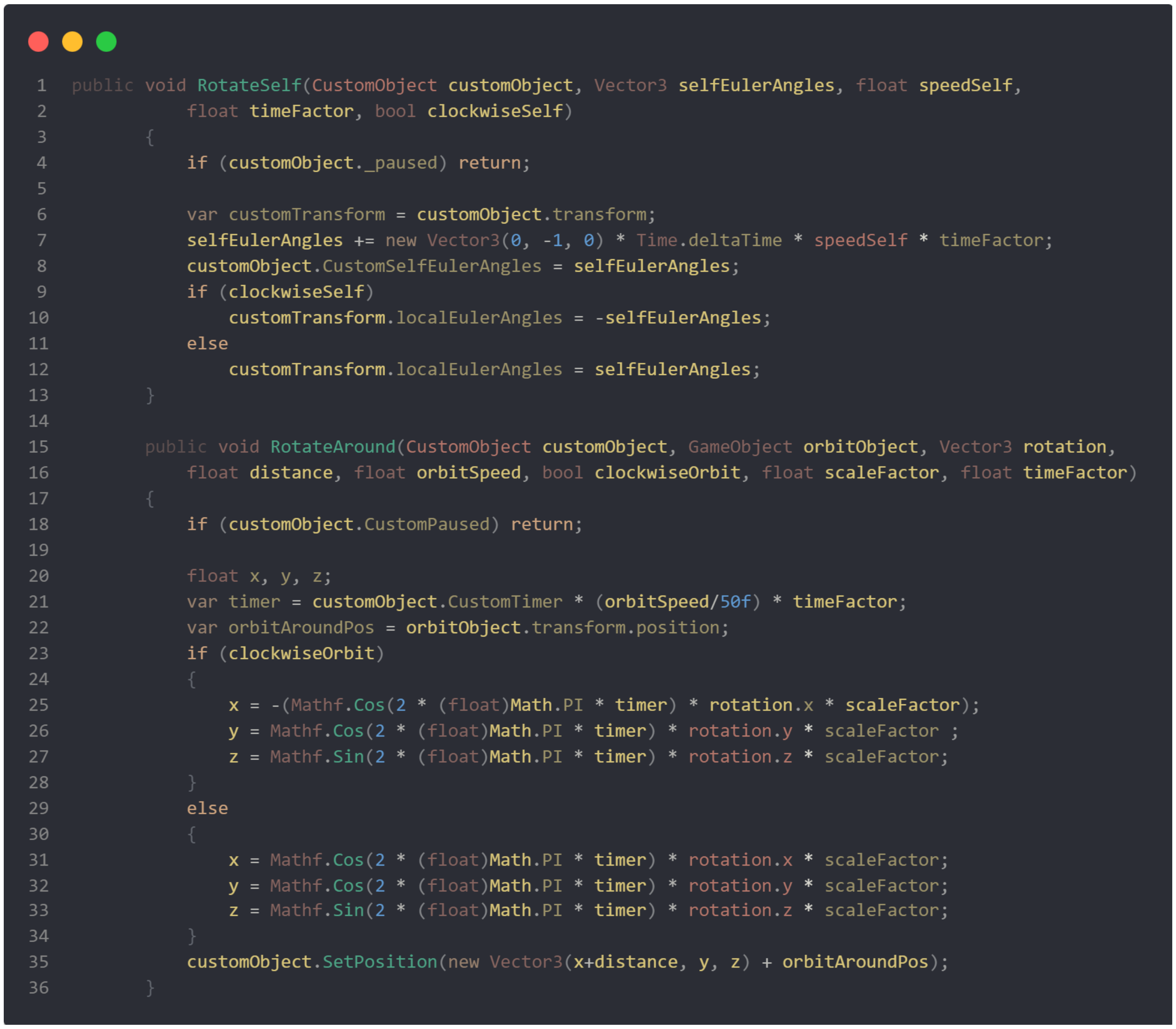
The actual implementation of the custom object's orbit around any object, and the custom object's orbit or spin around itself, is implemented in the Orbit Behavior script. This script is used in the AR experience's actual scene and it's responsible for the visualization of the objects with the orbit behavior.



```
public class OrbitBehavior : MonoBehaviour, IBehavior
   {
       private List<CustomObject> _allCustomObjects = new List<CustomObject>();
       private void Awake()
       {
           _allCustomObjects = FindObjectsOfType<CustomObject>().ToList();
       }

       private void Update()
       {
           foreach (var customObject in _allCustomObjects)
           {
               customObject.CustomTimer += Time.deltaTime;
               RotateSelf(customObject, customObject.CustomSelfEulerAngles, customObject.CustomSpeedSelf,
                   customObject.CustomTimeFactor, customObject.CustomClockwiseSelf);
               RotateAround(customObject, customObject.CustomOrbitObject, customObject.CustomRotation,
                   customObject.CustomDistance, customObject.CustomSpeedOrbit,customObject.CustomClockwiseOrbit,
                   customObject.CustomScaleFactor, customObject.CustomTimeFactor);
           }
       }
```

Figure 22. Orbit Behavior code snippet – Awake & Update.

Its main mechanics are the Rotate Self, and Rotate Around function, which you can see in depth at figure 23. The first one, spins all the custom objects around themselves, and the second one, orbits all the custom objects that are active in the hierarchy, around each one's selected orbit around object.



```
public void RotateSelf(CustomObject customObject, Vector3 selfEulerAngles, float speedSelf,
            float timeFactor, bool clockwiseSelf)
        {
            if (customObject._paused) return;

            var customTransform = customObject.transform;
            selfEulerAngles += new Vector3(0, -1, 0) * Time.deltaTime * speedSelf * timeFactor;
            customObject.CustomSelfEulerAngles = selfEulerAngles;
            if (clockwiseSelf)
                customTransform.localEulerAngles = -selfEulerAngles;
            else
                customTransform.localEulerAngles = selfEulerAngles;
        }

        public void RotateAround(CustomObject customObject, GameObject orbitObject, Vector3 rotation,
            float distance, float orbitSpeed, bool clockwiseOrbit, float scaleFactor, float timeFactor)
        {
            if (customObject.CustomPaused) return;

            float x, y, z;
            var timer = customObject.CustomTimer * (orbitSpeed/50f) * timeFactor;
            var orbitAroundPos = orbitObject.transform.position;
            if (clockwiseOrbit)
            {
                x = -(Mathf.Cos(2 * (float)Math.PI * timer) * rotation.x * scaleFactor);
                y = Mathf.Cos(2 * (float)Math.PI * timer) * rotation.y * scaleFactor ;
                z = Mathf.Sin(2 * (float)Math.PI * timer) * rotation.z * scaleFactor;
            }
            else
            {
                x = Mathf.Cos(2 * (float)Math.PI * timer) * rotation.x * scaleFactor;
                y = Mathf.Cos(2 * (float)Math.PI * timer) * rotation.y * scaleFactor;
                z = Mathf.Sin(2 * (float)Math.PI * timer) * rotation.z * scaleFactor;
            }
            customObject.SetPosition(new Vector3(x+distance, y, z) + orbitAroundPos);
        }
```

Figure 23. Orbit Behavior code snippet – RotateSelf & RotateAround.

5.3 Tool Results – AR Experience Use Case

The use case AR experience that got implemented is the visualization of the solar system. A small part of our solar system that involves the Sun, earth and the moon were implemented. The models of the planets to be used, were designed, and created in Blender and converted accordingly to fit Unity's needs. The result is a mobile application, where the user can interact with the planets via touching the objects themselves with the combination of the UI. The user can either affect the whole composition of custom objects, in this use case the planets, or each custom object on its own. The main actions that can be taken are scaling up or down, slowing down or speeding up the time, changing the position by dragging the objects in the AR space, and more.

5.3.1 Creation of the Models

The normal textures and the elevated data textures were needed to create each planet with their own details. The software needed for some retouching and detailing, as mentioned before, is Blender. You can see the main project, specifically about the development of the Sun's model, in Figure 24.

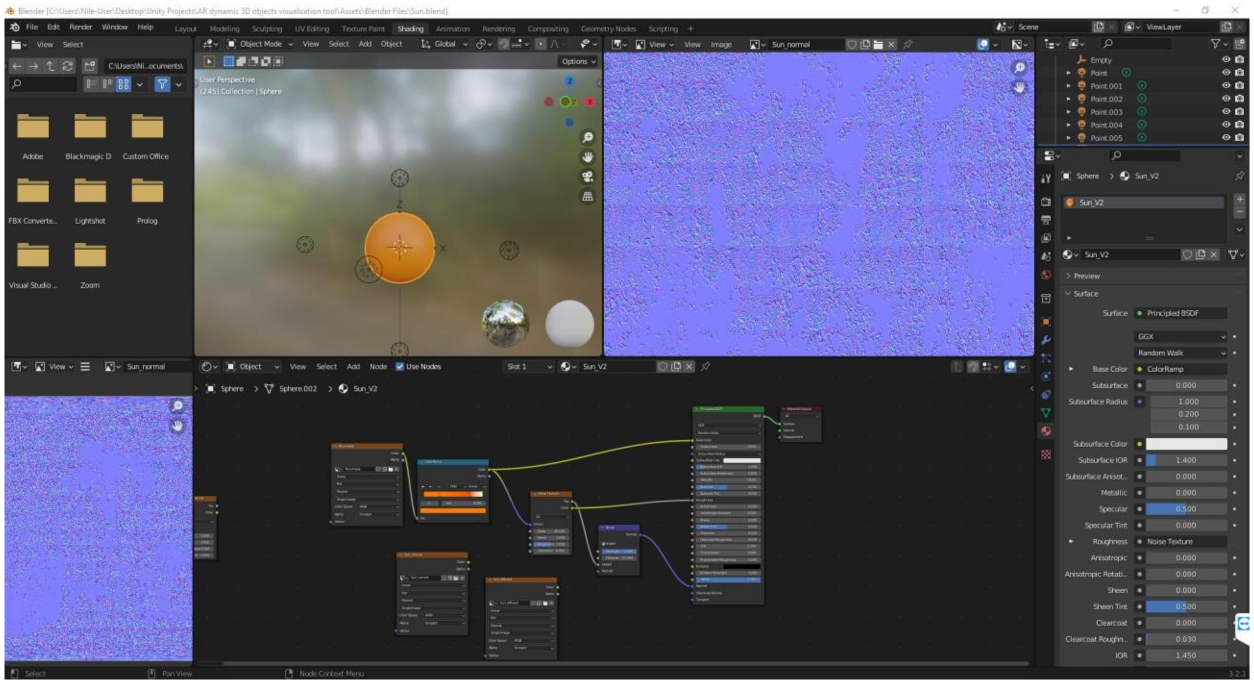
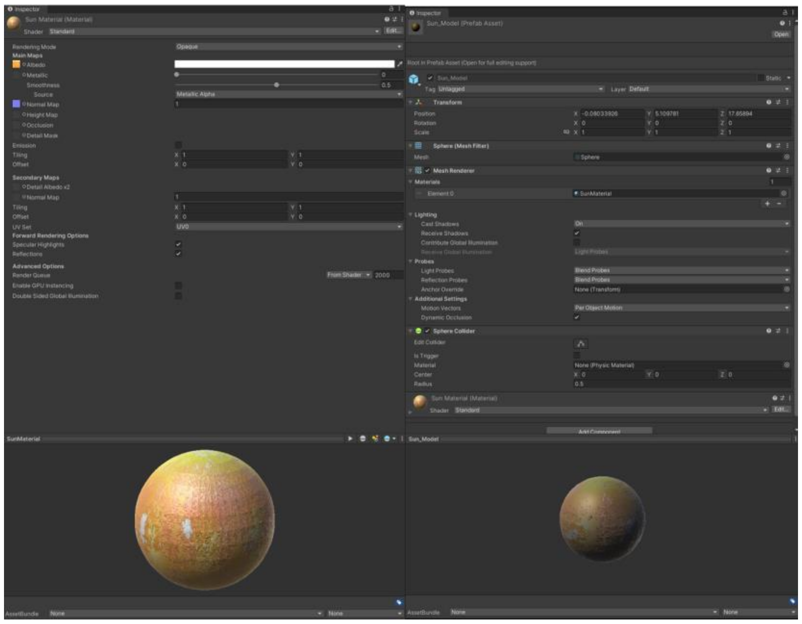


Figure 24. Blender's shading tab – Sun's model.

After all the work that needed to be done at Blender's environment, the conversion into Unity was required. The sun model got exported as a .fbx object, were the normal and the elevation data textures were included and got imported in Unity. After that, a basic material needed to be created, that uses the previously mentioned textures. Then, a default Sphere 3D object was created, and the custom planet's material got applied, to create the final game object that represents the Sun and was saved in the resources folder of the project as a prefab. This process was done for the rest of the planets that were included in the use case application. Both custom material and prefab that were created, can be seen more in-depth in Figure 25.



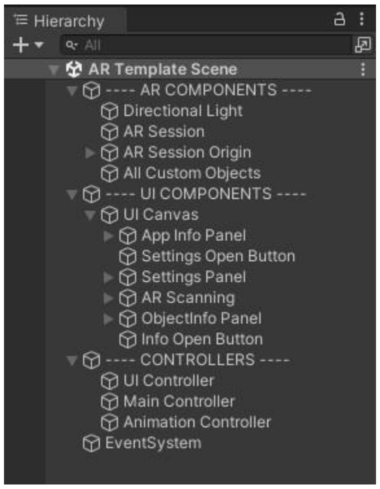
```
Inspector
Sun Material (Material)
Shader  Standard                                   Edit...
Rendering Mode                    Opaque
Main Maps
Albedo
Metallic                                           0
Smoothness                                         0.5
Source                            Metallic Alpha
Normal Map                        1
Height Map
Occlusion
Detail Mask
Emission
Tiling                            X 1     Y 1
Offset                            X 0     Y 0
Secondary Maps
Detail Albedo x2
Normal Map                        1
Tiling                            X 1     Y 1
Offset                            X 0     Y 0
UV Set                            UV0
Forward Rendering Options
Specular Highlights               ✓
Reflections                       ✓
Advanced Options
Render Queue                      From Shader  2000
Enable GPU Instancing
Double Sided Global Illumination
SunMaterial
AssetBundle  None                 None
```

```
Inspector
Sun_Model (Prefab Asset)                           Open
Root in Prefab Asset (Open for full editing support)
✓ Sun_Model                                        Static
Tag  Untagged                     Layer  Default
Transform
Position          X -0.08033926   Y 5.109781   Z 17.65894
Rotation          X 0             Y 0          Z 0
Scale             X 1             Y 1          Z 1
Sphere (Mesh Filter)
Mesh              Sphere
✓ Mesh Renderer
Materials                                          1
Element 0         SunMaterial
Lighting
Cast Shadows                      On
Receive Shadows                   ✓
Contribute Global Illumination
Receive Global Illumination       Light Probes
Probes
Light Probes                      Blend Probes
Reflection Probes                 Blend Probes
Anchor Override                   None (Transform)
Additional Settings
Motion Vectors                    Per Object Motion
Dynamic Occlusion                 ✓
✓ Sphere Collider
Edit Collider
Is Trigger
Material                          None (Physic Material)
Center            X 0             Y 0          Z 0
Radius            0.5
Sun Material (Material)
Shader  Standard                                   Edit...
Add Component
Sun_Model
AssetBundle  None                 None
```

Figure 25. Custom Sun material and prefab created in Unity.

5.3.2 Adding the planets in the scene.

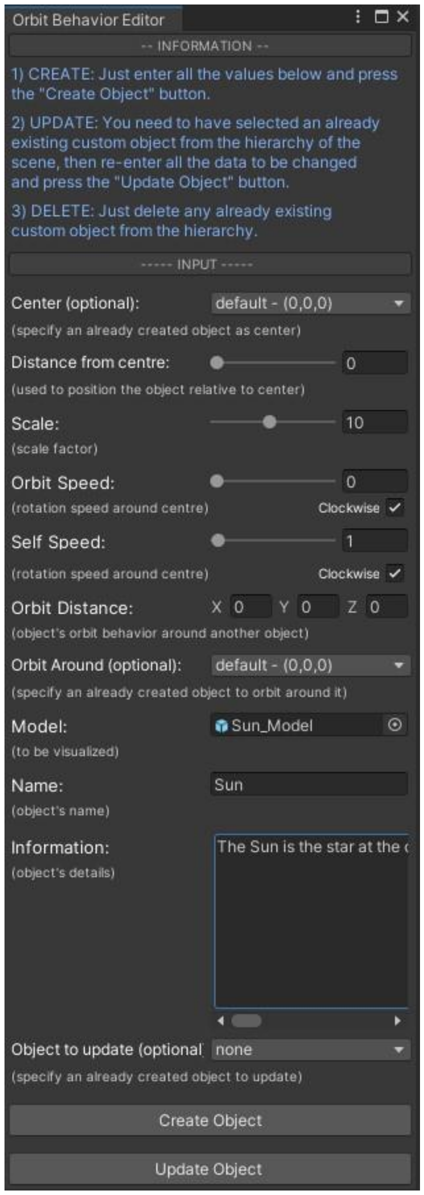
Then, the process that follows have to do with the actual use of the toolset. In this part, each custom object of the planet was added in the hierarchy of the scene through the toolset, after all the essential data settings were applied for each planet. In Figure 26 you can see the hierarchy of the use case scene in Unity before any planet was added.



```
Hierarchy
All
AR Template Scene
  ---- AR COMPONENTS ----
    Directional Light
    AR Session
    AR Session Origin
    All Custom Objects
  ---- UI COMPONENTS ----
    UI Canvas
      App Info Panel
      Settings Open Button
      Settings Panel
      AR Scanning
      ObjectInfo Panel
      Info Open Button
  ---- CONTROLLERS ----
    UI Controller
    Main Controller
    Animation Controller
  EventSystem
```

Figure 26. Unity's hierarchy before adding the planets.

The first custom object to be created was the Sun. All the essential settings were added through the friendly UI of the toolset. In Figure 27 you can see all the settings that were applied.



Orbit Behavior Editor

-- INFORMATION --

1) CREATE: Just enter all the values below and press the "Create Object" button.

2) UPDATE: You need to have selected an already existing custom object from the hierarchy of the scene, then re-enter all the data to be changed and press the "Update Object" button.

3) DELETE: Just delete any already existing custom object from the hierarchy.

----- INPUT -----

Center (optional): default - (0,0,0)
(specify an already created object as center)

Distance from centre: 0
(used to position the object relative to center)

Scale: 10
(scale factor)

Orbit Speed: 0
(rotation speed around centre) Clockwise ✓

Self Speed: 1
(rotation speed around centre) Clockwise ✓

Orbit Distance: X 0 Y 0 Z 0
(object's orbit behavior around another object)

Orbit Around (optional): default - (0,0,0)
(specify an already created object to orbit around it)

Model: Sun_Model
(to be visualized)

Name: Sun
(object's name)

Information: The Sun is the star at the
(object's details)

Object to update (optional: none
(specify an already created object to update)

Create Object

Update Object

Figure 27. Creating the Sun as a custom object through the toolset.

After clicking the create object button, the Sun custom object was immediately created in the hierarchy and ready to be used for the AR experience. This process was also done for the rest of the planets. In the end, the final state of the hierarchy was ready, and the actual application can be used from any user’s mobile device. In Figure 28, you can see the final version of the hierarchy after all the planets were created.

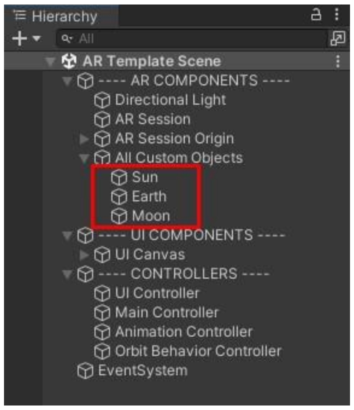


Figure 28. Unity’s hierarchy after adding all the planets.

5.3.3 The final application

After building the project, the final AR experience was ready to be used from any user. After giving the required permissions for the application to use the camera of the phone, the experience starts by scanning the area that the camera sees and starts the scan of vertical planes. A light visual starts to cover the detected area in the AR experience, and the user starts the visualization of the planets by touching a position from the planes that were detected. In Figure 29, you can see the process of plane detecting and the actual visualization of the planets, orbiting around as defined during the toolset’s settings application step.

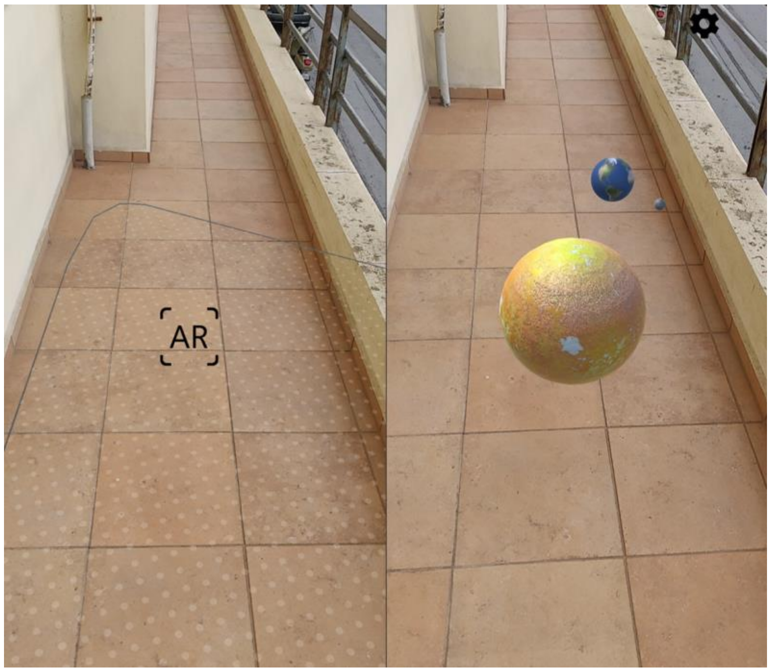
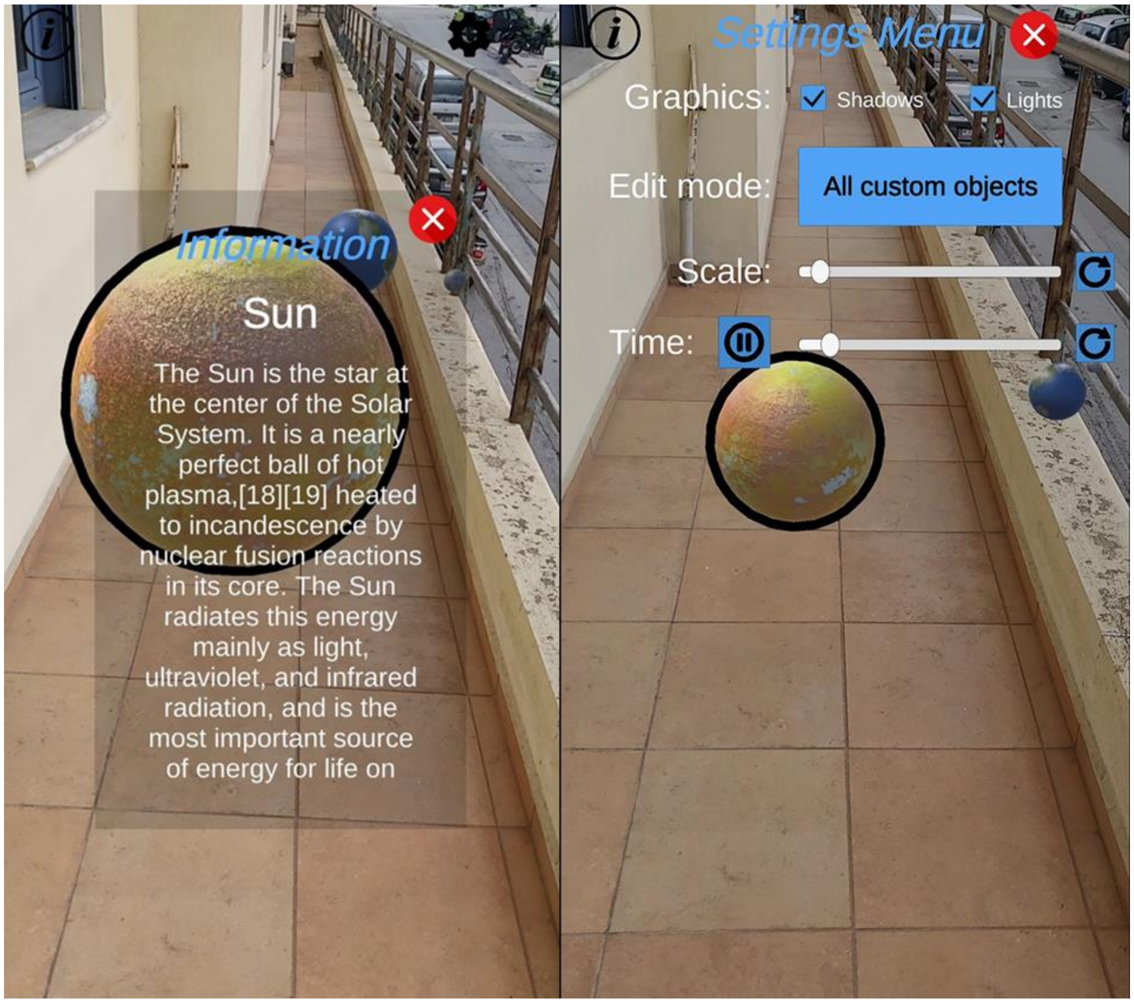


Figure 29. Plane detection and visualization of the planets.

The user can touch any planet to mark it as selected and has an option to see all the details of that custom object, just by tapping on the top left icon on the UI. In the left part of Figure 30, you can see all the information about the Sun that were added through the toolset.


Information

Sun

The Sun is the star at the center of the Solar System. It is a nearly perfect ball of hot plasma,[18][19] heated to incandescence by nuclear fusion reactions in its core. The Sun radiates this energy mainly as light, ultraviolet, and infrared radiation, and is the most important source of energy for life on

Settings Menu

Graphics: ✓ Shadows ✓ Lights

Edit mode: All custom objects

Scale:

Time:

Figure 30. Sun’s information panel and settings menu.

Finally, the user can also access some additional settings by tapping the top right icon on the UI. He can toggle on or off some graphics settings about the shadows and the lights. Also, he can affect some visualization settings that have to do with the scale of the planets and about how fast or slow they are moving. Last but not least, by tapping the edit mode button, he toggles between affecting all custom objects as a composition, or affecting only one custom object, which needs to be marked as selected by just touching it. In the right part of Figure 30, you can see all the settings UI that the user can apply his preferences.

Chapter 6 - Conclusion & Future Work

6.1 Conclusion

This thesis project had as a beginning as goal, the creation of a powerful, and at the same time, friendly for the user, toolset that will be able to create custom objects ready to be used in any AR experience, with any movement behavior he prefers. This gives the chance to any developer who is a beginner, as concerns the AR programming, to build his own immerse experiences, in a fast and interactive way. Hopefully, it can be a fantastic addition to the AR developing community, especially since the toolset can be extended way more.

The idea was unique and interesting, especially since there weren't many toolsets available to the public, which can help the developer during the development phase of their applications. In the beginning, the project faced some challenges such as the limited resources available for AR development and the lack of knowledge of the development team about AR programming. However, after several trials and errors, the creation of the toolset was successful, which met the initial goals and exceeded expectations. As mentioned in the previous chapter, the toolset was tested by several developers and received positive feedback, proving its effectiveness and user-friendliness.

The conclusion of this thesis project is that the creation of a powerful and user-friendly AR toolset is not only possible but also necessary in today's fast-growing AR industry. The toolset provides a solution to the common challenges faced by AR developers, especially beginners, and helps them create their own immersive experiences in a fast and interactive way. The team's work is a testament to the idea that innovation and creativity can lead to the development of valuable tools that can benefit the AR community and contribute to the growth of the industry.

6.2 Future Work

The toolset has the potential to be extended even further, offering even more capabilities and features to AR developers. Currently, the toolset supports only the orbit behavior, since there were many tricky parts that concerned both the GUI of the toolset and the actual implementation and mechanics of the orbit behavior. One of the major things that need to be implemented in the future is the support of more than one behavior for any custom object to be created from the user.

This requires the creation of a specific GUI that will be included in the toolset and the main support of its settings and options, to be used in each custom object that the user prefers to create.

Additionally, the current user interface for the AR toolset could be even simpler, clearer and more appealing, as concerns the visual design. To address this, the work on improving the user interface by making it more intuitive and visually appealing is essential. This could include adding clear and concise instructions, streamlined controls, and a more modern and appealing design. Also, another addition could be by incorporating drag-and-drop functionality to make it easier for users to create custom objects and add behaviors to them. The interface could also include helpful tips and suggestions to guide users through the process of creating their AR experiences.

Another great addition would be the support of physics-based behaviors to the toolset, such as gravity and collision detection, allowing developers to create more realistic and interactive AR experiences or/and the support for more complex animations and motion patterns, giving developers even more flexibility and control over their AR creations. Of course, the integration with popular AR platforms such as ARKit, ARCore, and Vuforia to increase the accessibility of the toolset to a wider range of developers and platforms would be essential, once the toolset is at a very steady and final version.

In closing, the future of the AR toolset is full of potential and opportunities. By continuously evolving and improving the AR toolset, the team hopes to provide AR developers with the tools they need to bring their innovative ideas to life and help the AR industry continue to grow and evolve. It is an exciting time for AR technology, and the toolset has the potential to be a driving force in the growth and evolution of AR technology, inspiring future generations of AR developers to take their ideas to the next level and create extraordinary immersive AR experiences.

Chapter 8 - References

[1] Azuma, R., Baillot, Y., Behringer, R., Feiner, S., Julier, S., & MacIntyre, B. (2001). Recent advances in augmented reality. IEEE computer graphics and applications, 21(6), 34-47., https://doi.org/10.1109/38.963459

[2] Milgram, P., & Kishino, F. (1994). A taxonomy of mixed reality visual displays. IEICE TRANSACTIONS on Information and Systems, 77(12), 1321-1329.,

[3] Coomans, M. K., & Timmermans, H. J. (1997, August). Towards a taxonomy of virtual reality user interfaces. In Proceedings. 1997 IEEE Conference on Information Visualization (Cat. No. 97TB100165) (pp. 279-284). IEEE., https://doi.org/10.1109/IV.1997.626531

[4] Ratcliffe, J., Soave, F., Bryan-Kinns, N., Tokarchuk, L., & Farkhatdinov, I. (2021, May). Extended Reality (XR) remote research: a survey of drawbacks and opportunities. In Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems (pp. 1-13)., https://doi.org/10.1145/3411764.3445170

[5] Wen, Y. (2021). Augmented reality enhanced cognitive engagement: Designing classroom-based collaborative learning activities for young language learners. *Educational Technology Research and Development*, *69*(2), 843-860., http://dx.doi.org/10.1007/s11423-020-09893-z

[6] Marín-Vega, H., Alor-Hernández, G., Colombo-Mendoza, L. O., Bustos-López, M., & Zataraín-Cabada, R. (2022). ZeusAR: a process and an architecture to automate the development of augmented reality serious games. Multimedia Tools and Applications, 81(2), 2901-2935., https://doi.org/10.1007/s11042-021-11695-1

[7] Logothetis, I., Sfyrakis, M., & Vidakis, N. (2023). EduARdo - Unity Components for Augmented Reality Environments

[8] Unity Technologies, "Unity3D." Unity Technologies, Jun. 2005, https://unity.com/ (accessed Feb. 10, 2023)

[9] Kumar, A., Mantri, A., & Dutta, R. (2021). Development of an augmented reality-based scaffold to improve the learning experience of engineering students in embedded system course. Computer Applications in Engineering Education, 29(1), 244-257., https://doi.org/10.1002/cae.22245

[10] Lin, H. Y., & Tsai, S. C. (2021). Student perceptions towards the usage of AR-supported STEMUP application in mobile courses development and its implementation into English learning. Australasian Journal of Educational Technology, 37(3), 88-103., https://doi.org/10.14742/ajet.6125

[11] Buchner, J., Krüger, J. M., Bodemer, D., & Kerres, M. Teachers' use of augmented reality in the classroom., http://dx.doi.org/10.13140/RG.2.2.28026.16326

[12] Logothetis, I., Katsaris, I., Sfyrakis, M., & Vidakis, N. (2023). 3D Geography Course using AR: The Case of the Map of Greece

[13] Koh, R. K. C., Duh, H. B. L., & Gu, J. (2010, October). An integrated design flow in user interface and interaction for enhancing mobile AR gaming experiences. In 2010 IEEE International Symposium on Mixed and Augmented Reality-Arts, Media, and Humanities (pp. 47-52). IEEE., https://doi.org/10.1109/ISMAR-AMH.2010.5643296

[14] McCallum, S., & Boletsis, C. (2013). Augmented reality & gesture-based architecture in games for the elderly. Studies in health technology and informatics, 189, 139-144., http://dx.doi.org/10.3233/978-1-61499-268-4-139

[15] Hamilton, R. (2019, March). Collaborative and competitive futures for virtual reality music and sound. In *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)* (pp. 1510-1512). IEEE., https://doi.org/10.1109/VR.2019.8798166

[16] Virtual Reality at the Library, https://dit-ringsted.dk/virtual-reality-paa-biblioteket/ (accessed Feb. 10, 2023)

[17] Doolani, S., Wessels, C., Kanal, V., Sevastopoulos, C., Jaiswal, A., Nambiappan, H., & Makedon, F. (2020). A review of extended reality (xr) technologies for manufacturing training. Technologies, 8(4), 77., https://doi.org/10.3390/technologies8040077

[18] Extended Reality (XR): A Sneak Peek into an Alternate Reality https://martechlive.com/extended-reality/ (accessed Feb. 10, 2023)

[19] Liarokapis, F., Macan, L., Malone, G., Rebolledo-Mendez, G., & De Freitas, S. (2009, March). A pervasive augmented reality serious game. In 2009 Conference in Games and Virtual Worlds for Serious Applications (pp. 148-155). IEEE., https://doi.org/10.1109/VS-GAMES.2009.40

[20] Jin, Q., Wang, D., Deng, X., Zheng, N., & Chiu, S. (2018, June). AR-Maze: a tangible programming tool for children based on AR technology. In Proceedings of the 17th ACM Conference on Interaction Design and Children (pp. 611-616)., https://doi.org/10.1145/3202185.3210784

[21] Seichter, H., Looser, J., & Billinghurst, M. (2008, September). ComposAR: An intuitive tool for authoring AR applications. In 2008 7th IEEE/ACM International Symposium on Mixed and Augmented Reality (pp. 177-178). IEEE., https://doi.org/10.1109/ISMAR.2008.4637354

[22] Sannikov, S., Zhdanov, F., Chebotarev, P., & Rabinovich, P. (2015). Interactive educational content based on augmented reality and 3D visualization. Procedia Computer Science, 66, 720-729., https://doi.org/10.1016/j.procs.2015.11.082

[23] Bell, D. (2004). Uml basics: The component diagram. IBM Global Services.

[24] Ohst, D., Welle, M., & Kelter, U. (2003, September). Differences between versions of UML diagrams. In Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering (pp. 227-236)., https://doi.org/10.1145/940071.940102

[25] Li, X., Liu, Z., & Jifeng, H. (2004, April). A formal semantics of UML sequence diagram. In 2004 Australian Software Engineering Conference. Proceedings. (pp. 168-177). IEEE.

[26] Joshi, B., & Joshi, B. (2016). Overview of SOLID Principles and Design Patterns. Beginning SOLID Principles and Design Patterns for ASP. NET Developers, 1-44., https://doi.org/10.1007/978-1-4842-1848-8_1

[27] Blender, https://www.blender.org/, (accessed Feb. 10, 2023)