ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΧΡΗΣΗ ΤΩΝ ΤΕΧΝΟΛΟΓΙΩΝ ΕΠΑΥΞΗΜΕΝΗΣ ΠΡΑΓΜΑΤΙΚΟΤΗΤΑΣ ΚΑΙ ΑΝΑΓΝΩΡΙΣΗΣ ΧΕΙΡΟΝΟΜΙΩΝ ΣΕ ΠΕΙΡΑΜΑΤΑ ΧΗΜΕΙΑΣ ΤΟΥ ΓΥΜΝΑΣΙΟΥ

Του φοιτητή
Βαλσαμής Αντώνιος
Αρ. Μητρώου: 174869

Επιβλέπων
Ονοματεπώνυμο Ευκλείδης Κεραμόπουλος
Βαθμίδα Καθηγητής

Ημερομηνία 04/09/2025

Τίτλος Δ.Ε.
Χρήση των τεχνολογιών επαυξημένης πραγματικότητας και αναγνώρισης χειρονομιών σε πειράματα Χημείας του Γυμνασίου
Κωδικός Δ.Ε. 25230
Ονοματεπώνυμο φοιτητή/τών Αντώνιος Βαλσαμής
Ονοματεπώνυμο εισηγητή Ευκλείδης Κεραμόπουλος
Ημερομηνία ανάληψης Δ.Ε. 30-03-2025
Ημερομηνία περάτωσης Δ.Ε. 04-09-2025

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Στην οικογένεια και τους φίλους μου

Πρόλογος

Η τεχνολογία έχει φέρει επανάσταση στην εκπαιδευτική διαδικασία, προσφέροντας νέα εργαλεία που μπορούν να ενισχύσουν τη μάθηση και να την κάνουν πιο διαδραστική. Μία από τις καινοτόμες τεχνολογίες που αξιοποιούνται τα τελευταία χρόνια στην εκπαίδευση είναι η Επαυξημένη Πραγματικότητα (Augmented Reality - AR), η οποία επιτρέπει στους μαθητές να αλληλεπιδρούν με ψηφιακά αντικείμενα στον πραγματικό κόσμο. Ιδιαίτερα στη διδασκαλία των φυσικών επιστημών, η AR μπορεί να προσφέρει μία βιωματική προσέγγιση, καθιστώντας την εκμάθηση πιο κατανοητή και ελκυστική.

Περίληψη

Στην παρούσα διπλωματική εργασία στόχος ήταν η σχεδίαση και ανάπτυξη μιας εκπαιδευτικής εφαρμογής επαυξημένης πραγματικότητας (Augmented Reality - AR) για τη διδασκαλία της Χημείας στη Β' Γυμνασίου. Η εφαρμογή προσομοιώνει ένα πείραμα ανάμιξης διαλυμάτων, επιτρέποντας στους μαθητές να αλληλεπιδρούν με το περιβάλλον του πειράματος μέσω αναγνώρισης χειρονομιών (hand tracking). Το πείραμα που χρησιμοποιείται βασίζεται στη διδακτέα ύλη του σχολικού βιβλίου Χημείας της Β' Γυμνασίου.

Για την ανάπτυξη της εφαρμογής χρησιμοποιήθηκε η πλατφόρμα **Unity**, σε συνδυασμό με το **ARCore** και το **AR Foundation** για την επαυξημένη πραγματικότητα. Η αναγνώριση χειρονομιών υλοποιήθηκε με τη χρήση του **MediaPipe Hands**, ενώ τα τρισδιάστατα μοντέλα υλοποιήθηκαν με το εμπορικό εργαλείο **Liquid Volume Pro**.

Η εργασία αυτή στοχεύει στην ενσωμάτωση καινοτόμων τεχνολογιών στη διδασκαλία της Χημείας, προσφέροντας έναν πιο διαδραστικό και βιωματικό τρόπο εκμάθησης για τους μαθητές.

Demonstrating middle school Chemistry lab experiments using Augmented Reality and Hand Tracking technologies

Antonios Valsamis

Abstract

The objective of this thesis was the design and development of an educational augmented reality (AR) application for teaching Chemistry to middle school students. The application simulates a liquid mixing experiment, allowing students to interact with the experiment environment using hand tracking technology. The experiment used in the application is based on the curriculum of the Chemistry textbook for the second grade of Greek middle school.

The development of the application was carried out using the **Unity** platform, in combination with **ARCore** and **AR Foundation** for augmented reality implementation. Hand tracking was implemented using a **MediaPipe Hands Unity Plugin**, while 3D models were implemented with **LiquidVolume Pro Unity Plugin**.

This work aims to integrate innovative technologies into Chemistry education, providing a more interactive and immersive learning experience for students.

Περιεχόμενα

Κατάλογος Σχημάτων

Συντομογραφίες

AR	Augmented Reality
VR	Virtual Reality
MR	Mixed Reality
XR	Extended Reality
HGR	Hand Gesture Recognition

Κεφάλαιο 1ο: Εισαγωγή

1.1 Γενικό υπόβαθρο και κίνητρα

Η τεχνολογία έχει μετασχηματίσει την εκπαιδευτική διαδικασία, προσφέροντας νέες δυνατότητες για πιο διαδραστική και αποτελεσματική μάθηση. Μία από τις καινοτόμες τεχνολογίες που αξιοποιούνται τα τελευταία χρόνια στην εκπαίδευση είναι η Επαυξημένη Πραγματικότητα (Augmented Reality - AR), η οποία επιτρέπει την ενσωμάτωση εικονικών αντικειμένων στον πραγματικό κόσμο, ενισχύοντας την κατανόηση σύνθετων εννοιών. Ιδιαίτερα στις φυσικές επιστήμες, όπου τα πειράματα παίζουν κεντρικό ρόλο στη μάθηση, η AR μπορεί να προσφέρει μία βιωματική προσέγγιση, καθιστώντας τη διδασκαλία πιο ελκυστική και κατανοητή.

Στα μαθήματα των φυσικών επιστημών, όπου συχνά οι έννοιες είναι αφηρημένες ή οι πειραματικές διαδικασίες απαιτούν εργαστηριακό εξοπλισμό που δεν είναι πάντα προσβάσιμος, η χρήση της AR μπορεί να γεφυρώσει το χάσμα μεταξύ θεωρίας και πράξης. Το συγκεκριμένο έργο βασίζεται σε ένα πείραμα από το σχολικό βιβλίο Χημείας της Β' Γυμνασίου, το οποίο εξετάζει τη συμπεριφορά διαλυμάτων με διαφορετική πυκνότητα (νερό, λάδι, μελάνι) και τον τρόπο που αλληλεπιδρούν μεταξύ τους. Η επιλογή αυτού του πειράματος έγινε επειδή είναι εύκολα κατανοητό, οπτικά εντυπωσιακό και αποτελεί μια ιδανική περίπτωση για επαυξημένη προσομοίωση.

Το επιπλέον στοιχείο που διαφοροποιεί αυτή την εργασία είναι η ενσωμάτωση χειρονομιών (hand tracking) ως μέσο αλληλεπίδρασης με την εφαρμογή, επιτρέποντας στους μαθητές να "πραγματοποιούν" το πείραμα με φυσικές κινήσεις, χωρίς την ανάγκη αγγίγματος της οθόνης. Αυτή η μορφή αλληλεπίδρασης καθιστά την εμπειρία πιο φυσική και καθηλωτική.

Το προσωπικό ενδιαφέρον για τη συνδυαστική χρήση AR, φυσικών επιστημών και εκπαιδευτικής τεχνολογίας, σε συνδυασμό με την επιθυμία να δημιουργηθεί ένα εργαλείο που θα μπορούσε να αξιοποιηθεί στην τάξη, αποτέλεσε την κινητήρια δύναμη για την εκπόνηση αυτής της πτυχιακής εργασίας.

1.2 Δομή της Εργασίας

Η εργασία αυτή οργανώνεται σε διάφορα κεφάλαια που καλύπτουν τις βασικές πτυχές της ανάπτυξης της εκπαιδευτικής εφαρμογής επαυξημένης πραγματικότητας (AR). Κάθε κεφάλαιο έχει συγκεκριμένο στόχο και περιεχόμενο που συμβάλλει στην κατανόηση και υλοποίηση του έργου.

Το **πρώτο κεφάλαιο** παρουσιάζει το γενικό υπόβαθρο του έργου, διατυπώνει το πρόβλημα που επιλύεται και περιγράφεται η σημασία της τεχνολογίας AR στην εκπαίδευση.

Στο **δεύτερο κεφάλαιο** αναλύεται η τεχνολογία της επαυξημένης πραγματικότητας (AR), οι διαφορετικοί τύποι της, και πώς αυτή χρησιμοποιείται στην εκπαίδευση. Επίσης, παρουσιάζονται οι τεχνολογίες χειρονομιών και το πώς η αναγνώριση κινήσεων επηρεάζει την αλληλεπίδραση του χρήστη με την εφαρμογή. Τέλος, γίνεται σύγκριση διαφόρων βιβλιοθηκών και εργαλείων που χρησιμοποιούνται στην ανάπτυξη εφαρμογών AR, με αναφορά στις τεχνολογίες που επιλέχθηκαν για αυτό το έργο.

Στο **τρίτο κεφάλαιο**, περιγράφεται η αρχιτεκτονική του συστήματος και ο τρόπος αλληλεπίδρασης του χρήστη με την εφαρμογή. Εξετάζονται οι απαιτήσεις του συστήματος, οι λειτουργίες που πρέπει να

υποστηρίζει και η διαδικασία σχεδιασμού του περιβάλλοντος εργασίας. Επίσης, αναλύονται τα εργαλεία που χρησιμοποιήθηκαν.

Το **τέταρτο κεφάλαιο** επικεντρώνεται στη διαδικασία ανάπτυξης της εφαρμογής. Εδώ περιγράφεται ο τρόπος που χρησιμοποιήθηκαν οι τεχνολογίες **Unity**, **ARCore**, **AR Foundation**, **MediaPipe Hands**, και **Liquid Volume Pro**.

Στο **τελευταίο κεφάλαιο**, αναλύονται τα συμπεράσματα της εργασίας και παρουσιάζονται προτάσεις για μελλοντικές βελτιώσεις, τόσο σε τεχνικό όσο και σε εκπαιδευτικό επίπεδο, με σκοπό τη βελτίωση της εμπειρίας του χρήστη και της εκπαιδευτικής αξίας της εφαρμογής.

Κεφάλαιο 2ο: Θεωρητικό Υπόβαθρο

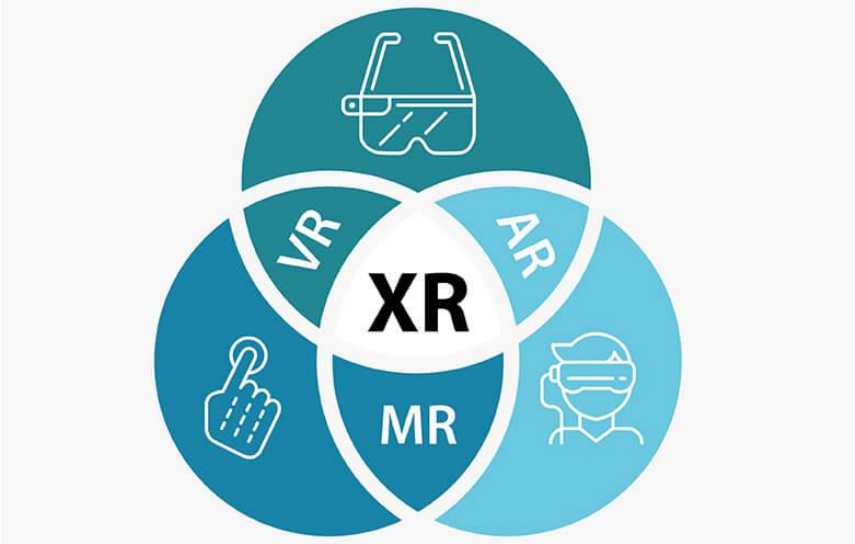
2.1 Εισαγωγή

Το παρόν κεφάλαιο παρουσιάζει το θεωρητικό και τεχνολογικό υπόβαθρο που στηρίζει την ανάπτυξη της εκπαιδευτικής εφαρμογής. Γίνεται αναφορά στις βασικές έννοιες της Επαυξημένης Πραγματικότητας, στις τεχνολογίες αναγνώρισης χειρονομιών (hand tracking), στις πλατφόρμες ανάπτυξης, καθώς και στη δημιουργία τρισδιάστατων μοντέλων. Παράλληλα, εξετάζονται και συγκρίνονται σχετικές τεχνολογίες και εργαλεία, με στόχο τη δικαιολόγηση των επιλογών που υιοθετήθηκαν κατά την ανάπτυξη της εφαρμογής.

2.2 Εκτεταμένη πραγματικότητα

2.2.1 Εκτεταμένη πραγματικότητα (Extended Reality - XR)

Η Εκτεταμένη Πραγματικότητα (XR) αποτελεί γενικό όρο που καλύπτει όλες τις τεχνολογίες οι οποίες συνδυάζουν τον φυσικό με τον ψηφιακό κόσμο. Οι τεχνολογίες αυτές περιλαμβάνουν την Εικονική Πραγματικότητα (Virtual Reality - VR), τη Μικτή Πραγματικότητα (Mixed Reality - MR) και την Επαυξημένη Πραγματικότητα (Augmented Reality - AR). Όλες οι παραπάνω καταστάσεις τοποθετούνται κατά μήκος του “Reality-Virtuality Continuum”, το οποίο εκτείνεται από το πλήρως φυσικό έως το πλήρως ψηφιακό περιβάλλον [1].



2.1 Εκτεταμένη πραγματικότητα

2.2.2 Εικονική πραγματικότητα (Virtual Reality - VR)

Η Εικονική Πραγματικότητα αναφέρεται σε ένα πλήρως υπολογιστικά παραγόμενο περιβάλλον, όπου ο χρήστης είναι πλήρως αποκομμένος από τον πραγματικό κόσμο και αλληλεπιδρά αποκλειστικά με το εικονικό περιεχόμενο. Τα περιβάλλοντα αυτά παρέχουν τη δυνατότητα απεικόνισης καταστάσεων που προσομοιώνουν ή υπερβαίνουν τη φυσική πραγματικότητα. Η VR τοποθετείται στο άκρο του συνεχούς, όπου δεν υφίσταται καμία πρόσληψη φυσικού ερεθίσματος [1].

2.2 Εικονική πραγματικότητα

2.2.3 Μικτή πραγματικότητα (Mixed Reality - MR)

Η Μικτή Πραγματικότητα αποτελεί ενδιάμεση κατηγορία μεταξύ του φυσικού και του εικονικού κόσμου, στην οποία συνυπάρχουν και αλληλεπιδρούν αντικείμενα από αμφότερα τα περιβάλλοντα. Υβριδικά συστήματα MR διαφοροποιούνται μεταξύ τους με βάση χαρακτηριστικά όπως το αν η απεικόνιση του φυσικού κόσμου γίνεται άμεσα ή έμμεσα, ο τύπος της προβολής (monitor ή HMD), και το αν υπάρχει ορθοσκοπικός συσχετισμός μεγέθους μεταξύ πραγματικών και εικονικών στοιχείων [1].

Η ταξινόμηση των συστημάτων γίνεται με βάση τρεις βασικές διαστάσεις:

- **Extent of World Knowledge (EWK)**: βαθμός πληροφόρησης για τον πραγματικό κόσμο,
- **Reproduction Fidelity (RF)**: ποιότητα αναπαραγωγής του περιβάλλοντος,
- **Extent of Presence Metaphor (EPM)**: επίπεδο αίσθησης παρουσίας στο απεικονιζόμενο περιβάλλον [2].

Η θέση ενός συστήματος στο φάσμα MR εξαρτάται από τον τρόπο απεικόνισης, τον βαθμό αλληλεπίδρασης, καθώς και το ποσοστό εικονικών και πραγματικών στοιχείων που περιλαμβάνει [2].

2.3 Μικτή πραγματικότητα

2.2.4 Επαυξημένη πραγματικότητα (Augmented Reality - AR)

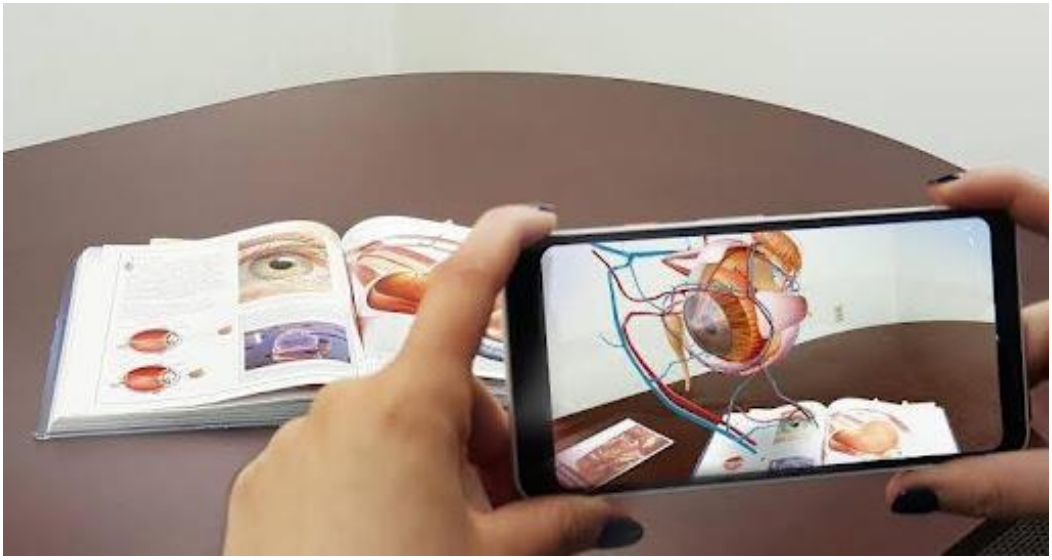
Η Επαυξημένη Πραγματικότητα (AR) ενσωματώνει εικονικά στοιχεία στο φυσικό περιβάλλον, σε πραγματικό χρόνο και με γεωμετρική συνέπεια. Σε αντίθεση με την Εικονική Πραγματικότητα, η οποία αποκλείει τον πραγματικό κόσμο, η AR βασίζεται στη συγχώνευση του φυσικού και του ψηφιακού περιεχομένου μέσα σε ένα κοινό πεδίο προβολής. Ένα σύστημα AR ορίζεται ως τέτοιο όταν πληροί τρία βασικά κριτήρια: συνδυασμό πραγματικού και εικονικού, διαδραστικότητα σε πραγματικό χρόνο και οπτικοποίηση σε τρεις διαστάσεις [3].

Η τεχνολογία AR μπορεί να εφαρμοστεί μέσω διαφόρων τύπων συσκευών, όπως head-mounted displays (HMDs), φορητές συσκευές (smartphones, tablets) και συστήματα προβολής με χρήση κάμερας. Η καταγραφή της πραγματικότητας και η ενσωμάτωση εικονικών αντικειμένων πραγματοποιούνται μέσω αισθητήρων, καμερών και αλγορίθμων εντοπισμού θέσης και προσανατολισμού [3].

Η AR βρίσκει εφαρμογή σε πλήθος τομέων όπως η βιομηχανία, η αρχιτεκτονική, η ιατρική απεικόνιση, ο στρατός και η εκπαίδευση. Στο πεδίο της εκπαίδευσης, η ενσωμάτωση AR έχει αποδειχθεί ιδιαίτερα αποτελεσματική σε γνωστικά αντικείμενα που απαιτούν χωρική κατανόηση ή αλληλεπίδραση με αφηρημένες έννοιες, όπως οι φυσικές επιστήμες, τα μαθηματικά και η γλωσσομάθεια [4].

2.4 Επαυξημένη πραγματικότητα

Η χρήση της AR στην εκπαιδευτική διαδικασία σχετίζεται με βελτίωση της κατανόησης, ενίσχυση της συγκέντρωσης και αύξηση των κινήτρων μάθησης. Τα AR περιβάλλοντα επιτρέπουν στους μαθητές να εξερευνούν πληροφορίες με ενεργό τρόπο, ενισχύοντας τη βιωματική και συνεργατική μάθηση, ενώ η οπτικοποίηση αφηρημένων εννοιών διευκολύνει την αφομοίωση [4]. Επιπλέον, η χρήση AR έχει συνδεθεί με την ενίσχυση της μακροπρόθεσμης μνήμης και της γνωστικής εμπλοκής, προσφέροντας ελκυστικά σενάρια αλληλεπίδρασης και άμεσης ανατροφοδότησης.



2.5 AR στην εκπαιδευτική διαδικασία

2.3 Τεχνολογίες αναγνώρισης χειρονομιών (Hand Gesture Recognition - HGR)

Η αναγνώριση χειρονομιών (Hand Gesture Recognition – HGR) είναι ένα υποπεδίο της αλληλεπίδρασης ανθρώπου–υπολογιστή (Human-Computer Interaction – HCI) το οποίο επιτρέπει στους χρήστες να ελέγχουν και να επικοινωνούν με ψηφιακά συστήματα μέσω κινήσεων των χεριών,

χωρίς τη χρήση φυσικών συσκευών εισόδου. Οι χειρονομίες χωρίζονται σε δύο βασικές κατηγορίες: στατικές, οι οποίες βασίζονται σε συγκεκριμένες θέσεις του χεριού, και δυναμικές, που περιλαμβάνουν διαδοχικές κινήσεις στον χρόνο [5].

Ανάλογα με τη μέθοδο καταγραφής των χειρονομιών, τα συστήματα αναγνώρισης διακρίνονται σε τρεις βασικούς τύπους:

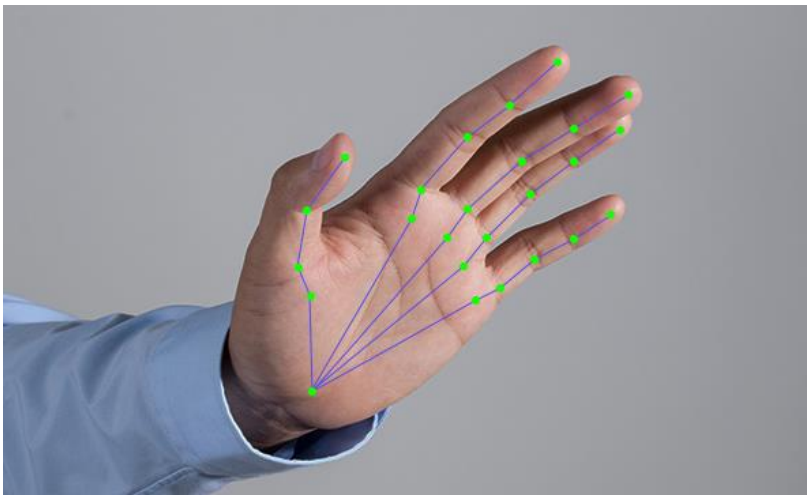
1. Συστήματα βάσει γαντιών (Glove-based), τα οποία χρησιμοποιούν αισθητήρες για την καταγραφή των κινήσεων και της κάμψης των δακτύλων.

2.6 Glove-based HGR

2. Συστήματα βάσει χρώματος ή μαρκαρίσματος (Color-marker-based), τα οποία βασίζονται στην τοποθέτηση χρωματικών δεικτών στα χέρια για την ανίχνευση θέσης και κίνησης.

2.7 Color-marker based HGR

3. Οπτικά συστήματα (Vision-based), τα οποία χρησιμοποιούν κάμερες για την ανάλυση της εικόνας του χεριού, με ή χωρίς πρόσθετα βοηθήματα [5].



2.8 Vision-based HGR

Η επεξεργασία των δεδομένων σε συστήματα HGR γίνεται με τη βοήθεια αλγορίθμων τεχνητής νοημοσύνης και μηχανικής μάθησης. Συχνά χρησιμοποιούνται τεχνικές όπως τα τεχνητά νευρωνικά δίκτυα (Artificial Neural Networks - ANN), τα κρυμμένα μοντέλα Μαρκόφ (Hidden Markov Models - HMM), τα Finite-State Machines (FSM), καθώς και στατιστικές μέθοδοι ταξινόμησης. Οι τεχνικές αυτές επιτρέπουν την εξαγωγή χαρακτηριστικών και την ταξινόμηση των χειρονομιών με βάση την ακρίβεια και τη χρονική απόκριση [5].

Η αναγνώριση χειρονομιών βρίσκει ευρεία εφαρμογή σε τομείς όπως η εικονική και επαυξημένη πραγματικότητα, η υποβοηθούμενη επικοινωνία (π.χ. νοηματική γλώσσα), τα παιχνίδια και τα διαδραστικά περιβάλλοντα. Στην εκπαίδευση, προσφέρει τη δυνατότητα φυσικής και αβίαστης αλληλεπίδρασης με το περιεχόμενο, ενισχύοντας την ενεργητική εμπλοκή των μαθητών και την εμπειρική μάθηση.

Η επιλογή της κατάλληλης τεχνολογίας εξαρτάται από τις απαιτήσεις κάθε εφαρμογής. Η ακρίβεια, ο ρυθμός δειγματοληψίας, η καθυστέρηση απόκρισης και η ευκολία ενσωμάτωσης με άλλα συστήματα αποτελούν βασικούς παράγοντες κατά τον σχεδιασμό τέτοιων λύσεων. Τα οπτικά συστήματα παρουσιάζουν σημαντικά πλεονεκτήματα στην ανάπτυξη φορητών και χαμηλού κόστους εφαρμογών AR, λόγω της δυνατότητάς τους να λειτουργούν με απλές κάμερες χωρίς εξωτερικούς αισθητήρες [5].

2.4 Πλατφόρμες ανάπτυξης AR και Hand Tracking

2.4.1 Εισαγωγή στις πλατφόρμες ανάπτυξης

Η ανάπτυξη εφαρμογών επαυξημένης πραγματικότητας (AR) που ενσωματώνουν αναγνώριση χειρονομιών απαιτεί τη χρήση κατάλληλων εργαλείων και βιβλιοθηκών. Το **Unity** αποτελεί μία από τις πιο διαδεδομένες μηχανές ανάπτυξης εφαρμογών 2D και 3D, προσφέροντας ευρεία υποστήριξη για AR μέσω του πακέτου **AR Foundation**. Το AR Foundation επιτρέπει την ανάπτυξη εφαρμογών με κοινό API τόσο για το **ARCore** (Android) όσο και για το **ARKit** (iOS), επιτρέποντας τη δημιουργία φορητών και επεκτάσιμων AR εφαρμογών [6], [7].

2.4.2 Βιβλιοθήκες αναγνώρισης χειρονομιών

Οι βασικές επιλογές για αναγνώριση χειρονομιών περιλαμβάνουν:

- **MediaPipe Hands**: Μοντέλο ML που εντοπίζει **21 landmarks** ανά χέρι σε RGB εικόνα, σε **real time**, με **on-device inference** και υψηλή ακρίβεια. Στο Unity αξιοποιείται μέσω του **homuler plugin**, το οποίο γεφυρώνει τα native γραφήματα του MediaPipe με GameObjects/Transforms. Ιδανικό για mobile AR [8] [9].
- **HandPose (TensorFlow.js)**: Βιβλιοθήκη βασισμένη στο TensorFlow.js, σχεδιασμένη για χρήση μέσω browser, με υποστήριξη ανίχνευσης πολλών χεριών και βασικών σημείων σε 2D/3D [10].
- **OpenCV**: Βιβλιοθήκη υπολογιστικής όρασης που μπορεί να χρησιμοποιηθεί για αναγνώριση χειρονομιών, αν και απαιτεί εκτενέστερη προγραμματιστική υλοποίηση και παραμετροποίηση [11].
- **Leap Motion Controller**: Εξειδικευμένη συσκευή με υπέρυθρες κάμερες, που προσφέρει εξαιρετική ακρίβεια ανίχνευσης χεριών και SDK για ανάπτυξη σε desktop ή XR εφαρμογές. Δεν ενδείκνυται για mobile πλατφόρμες [12].
- **OpenXR**: Προδιαγραφή ανοιχτού προτύπου από το Khronos Group, σχεδιασμένη για τη διαλειτουργικότητα μεταξύ διαφορετικών XR συσκευών και πλατφορμών. Υποστηρίζεται από πλήθος κατασκευαστών και ενσωματώνεται στο Unity μέσω κατάλληλων πακέτων. Ωστόσο, δεν προσφέρει αυτόνομα μοντέλα ή μεθόδους για αναγνώριση χειρονομιών· λειτουργεί ως ενδιάμεσος abstraction layer για XR εφαρμογές, και πρέπει να συνδυάζεται με άλλες βιβλιοθήκες ή plugins για χειρονομίες [13].
- **ManoMotion SDK**: Εμπορική βιβλιοθήκη με εύκολη ενσωμάτωση και έτοιμες χειρονομίες, αλλά με **κλειστό κώδικα** και μικρότερη παραμετροποίηση [14].

2.4.3 Επιλογή Τενχολογίας Hand Tracking

Παρότι το **ManoMotion SDK** προσφέρει μια «έτοιμη» λύση με προδιαμορφωμένες χειρονομίες και απλή ενσωμάτωση στο Unity, παραμένει **κλειστό** και **εμπορικό** SDK, με μικρότερη διαφάνεια ως προς τον αλγόριθμο και μεγαλύτερη εξάρτηση από τον προμηθευτή. Αυτό περιορίζει την παραμετροποίηση όταν απαιτούνται **προσαρμοσμένοι κανόνες/κατώφλια** για εκπαιδευτικά σενάρια ή ακαδημαϊκή τεκμηρίωση.

Αντίθετα, το **MediaPipe Hands** σε συνδυασμό με το **homuler Unity plugin** παρέχει ανοιχτού κώδικα υποδομή με πλήρη πρόσβαση στα **21 landmarks** ανά καρέ. Έτσι μπόρεσαν να υλοποιηθούν προσαρμοσμένοι αλγόριθμοι με ελεγχόμενα **thresholds, smoothing και debounce**, διασφαλίζοντας χαμηλή καθυστέρηση και on-device λειτουργία σε Android. Η αρχική ρύθμιση που απαιτείται από το MediaPipe (native bindings/graphs) απλοποιείται ουσιαστικά από το homuler plugin (έτοιμα prefabs/παραδείγματα), με αποτέλεσμα μια σταθερή και επεκτάσιμη ολοκλήρωση στο Unity.

Συμπέρασμα επιλογής. Με γνώμονα την ανάγκη για ευελιξία**,** παραμετροποίηση σε επίπεδο landmarks**,** ακαδημαϊκή διαφάνεια και μηδενικό κόστος αδειών, επιλέχθηκε το **MediaPipe Hands (homuler)** έναντι του ManoMotion.

2.5 Δημιουργία Τρισδιάστατων Μοντέλων για την Εφαρμογή

2.5.1 Σημασία των 3D μοντέλων στις εφαρμογές επαυξημένης πραγματικότητας

Στις εφαρμογές επαυξημένης πραγματικότητας (AR), τα τρισδιάστατα (3D) μοντέλα αποτελούν τον πυρήνα της εμπειρίας του χρήστη, επιτρέποντας την προβολή και αλληλεπίδραση με ψηφιακά

αντικείμενα στον πραγματικό κόσμο. Η ποιότητα, η απόδοση και η ρεαλιστικότητα αυτών των μοντέλων επηρεάζουν άμεσα την αποτελεσματικότητα και την εμβύθιση της εφαρμογής. Επομένως, η δημιουργία βέλτιστων 3D μοντέλων είναι κρίσιμη για την επιτυχία μιας AR εφαρμογής.

2.5.2 Επιλογές εργαλείων για 3D περιεχόμενο και υγρά (Blender & Liquid Volume Pro)

Blender

Το **Blender** είναι ένα δωρεάν και ανοιχτού κώδικα λογισμικό δημιουργίας 3D περιεχομένου, το οποίο προσφέρει ένα πλήρες σύνολο εργαλείων για μοντελοποίηση, υφή, animation και rendering. Η ευελιξία του Blender, σε συνδυασμό με την ενεργή κοινότητά του και την εκτενή τεκμηρίωση, το καθιστούν ιδανική επιλογή για την ανάπτυξη 3D μοντέλων που προορίζονται για εφαρμογές AR. Επιπλέον, υποστηρίζει την εξαγωγή μοντέλων σε διάφορες μορφές αρχείων, όπως FBX και glTF, που είναι συμβατές με τις περισσότερες πλατφόρμες AR [15].

Liquid Volume Pro (LVP)

Το LVP είναι ένα εμπορικό shader-based Unity Plugin που προσομοιώνει ρεαλιστικά υγρά μέσα σε δοχεία και λειτουργεί σε 3D σκηνές αλλά και σε 2D/UI (π.χ. δείκτες/μετρητές). Υποστηρίζει πολλαπλά επίπεδα λεπτομέρειας, διαφορετικές τοπολογίες (Sphere/Cylinder/Cube/Irregular) και διαθέτει "Multiple" detail για πολλαπλές στρώσεις υγρών με ιδιότητες όπως amount, density, viscosity, murkiness και mixing (υγρά της ίδιας πυκνότητας αναμιγνύονται). Υπάρχει επίσης C# API για προγραμματιστικό έλεγχο,ενώ παρέχονται βέλτιστες πρακτικές για mobile.

Τέλος, το LVP περιλαμβάνει **demo σκηνές, έτοιμα prefabs** όπως ποτήρια εργαστηρίου και σταγονόμετρα τα οποία είναι διαθέσιμα για άμεση εισαγωγή στη σκηνή, καθώς και UI prefabs. [16]

2.5.3 Ροές Παραγωγής: Μοντέλα με Blender vs LVP

Α. Παραγωγή 3D αντικειμένων με Blender (props/σκεύη/περιβάλλον).

Βήματα:

(i) Πολυγωνική μοντελοποίηση με έλεγχο polycount,

(ii) UV mapping

(iii) Υλικά/υφές σε ανάλυση κατάλληλα για κινητά

(iv) Custom animation/εφέ για τη ροή υγρών

(v) (iv) Εξαγωγή σε σωστή κλίμακα/μονάδες, ώστε τα μοντέλα να ενσωματωθούν ομαλά στο Unity. [15]

Β. Ροή οπτικοποίησης/συμπεριφοράς υγρών με LVP.

Βήματα:

(i) Εισαγωγή έτοιμων prefab στη σκηνή (ποτήρια, σταγονόμετρα, κτλ.)

(ii) Προσθήκη component **LiquidVolume** σε δοχεία (σφαίρα/κύλινδρο/κύβο ή Irregular)

(iii) Επιλογή **Topology/Detail**

(iv) Ρύθμιση **Liquid/Layer** settings: χρώματα, επίπεδο πλήρωσης, multi-layer με density/miscible για ανάμιξη (νερό-λάδι-μελάνι)

(v) Χρήση **API** για δυναμικές αλλαγές επιπέδου/χρώματος/μείξης από Unity script. [16]

2.5.4 Επιλογή Τεχνολογίας 3D Μοντελοποίησης και Προσομοίωσης Υγρών

Η επιλογή του **Liquid Volume Pro (LVP)** κρίθηκε καταλληλότερη διότι επιτάχυνε ουσιαστικά όλη τη ροή ανάπτυξης, προσφέροντας **έτοιμα προς χρήση prefabs** (beakers, flasks, pipettes, σταγόνες/spills) και **ενσωματωμένη προσομοίωση υγρών** με στρώματα, μίξη και παραμέτρους πυκνότητας, καθώς και **C# API** για άμεσο έλεγχο (π.χ. στάθμη, στρώσεις, σημείο εκροής) μέσα από τα scripts.

Σε αντίθεση, το **Blender**—παρά τα σημαντικά πλεονεκτήματα ως ανοικτού κώδικα εργαλείο για ακριβή μοντελοποίηση και πλήρη καλλιτεχνικό έλεγχο—θα απαιτούσε **mesh-based** υλοποίηση των υγρών (animation/εφέ), περισσότερους κύκλους βελτιστοποίησης για Android και αυξημένο χρόνο συντήρησης, χωρίς «ζωντανή» αμφίδρομη σύνδεση με το gameplay.

Το LVP, μολονότι **εμπορικό προϊόν** (κόστος άδειας και εξάρτηση από τρίτο πάροχο), πλεονεκτεί σε **ταχύτητα ενσωμάτωσης**, **συνέπεια οπτικής συμπεριφοράς** σε φορητές συσκευές, **μειωμένο τεχνικό ρίσκο** και **ευκολία συντήρησης** (π.χ. αλλαγές σε επίπεδα/μείξη γίνονται προγραμματιστικά). Συνολικά, ο **χρόνος που εξοικονομήθηκε** μαζί με τη σαφήνεια της διδακτικής εμπειρίας που απαιτεί άμεση, ρεαλιστική απεικόνιση της ροής/μείξης—καθιστά το **LVP** την **ξεκάθαρη επιλογή** για την παρούσα εφαρμογή, ενώ το Blender διατηρεί τη χρησιμότητά του κυρίως ως συμπληρωματικό εργαλείο για γενικά 3D assets.

2.6 Επίλογος

Στο παρόν κεφάλαιο παρουσιάστηκε το θεωρητικό και τεχνολογικό υπόβαθρο που υποστηρίζει την ανάπτυξη της εκπαιδευτικής εφαρμογής επαυξημένης πραγματικότητας. Αρχικά αναλύθηκαν οι βασικές έννοιες της εκτεταμένης, εικονικής, μικτής και επαυξημένης πραγματικότητας, με στόχο την κατανόηση του πλαισίου στο οποίο εντάσσεται η εφαρμογή. Στη συνέχεια, εξετάστηκαν οι τεχνολογίες αναγνώρισης χειρονομιών και οι βιβλιοθήκες που τις υποστηρίζουν, με έμφαση στη λειτουργικότητα, την ακρίβεια και την καταλληλότητά τους για φορητές συσκευές.

Παρουσιάστηκαν οι κύριες πλατφόρμες ανάπτυξης AR, όπως το Unity, το AR Foundation και το ARCore, καθώς και τα εργαλεία αναγνώρισης χειρονομιών, όπως το MediaPipe, το ManoMotion SDK, το Leap Motion, το OpenXR και το TensorFlow HandPose.

Παρουσιάστηκε επίσης το LiquidVolumePro, ως εξειδικευμένο εργαλείο οπτικοποίησης και προσομοίωσης υγρών με έτοιμα prefabs δοχείων.

Επιπλέον, έγινε αναφορά στο λογισμικό Blender και στη σημασία της δημιουργίας τρισδιάστατων μοντέλων για την οπτικοποίηση και αλληλεπίδραση εντός του εκπαιδευτικού περιβάλλοντος.

Η συγκριτική ανάλυση που πραγματοποιήθηκε ανέδειξε την καταλληλότητα του τεχνολογικού συνδυασμού που επελέγη για την παρούσα εργασία, συμπεριλαμβανομένης της χρήσης του **MediaPipe Hands (homuler)** και του **LVP**, βάσει λειτουργικότητας, απόδοσης και ευκολίας ενσωμάτωσης. Τα εργαλεία που επιλέχθηκαν υποστηρίζουν τη δημιουργία μιας προσβάσιμης και διαδραστικής εμπειρίας μάθησης, αξιοποιώντας τις δυνατότητες των σύγχρονων κινητών συσκευών.

Στο επόμενο κεφάλαιο, η εστίαση μετατοπίζεται από το θεωρητικό επίπεδο στη **σχεδίαση και αρχιτεκτονική του συστήματος**, όπου θα παρουσιαστεί αναλυτικά η λειτουργική και τεχνική ανάλυση της εφαρμογής.

Κεφάλαιο 3ο: Σχεδίαση και Αρχιτεκτονική Συστήματος

3.1 Εισαγωγή

Πριν την έναρξη της υλοποίησης της εφαρμογής, είναι απαραίτητη η σχεδίαση της συνολικής αρχιτεκτονικής και της λειτουργικής δομής του συστήματος. Η σχεδίαση επιτρέπει την αποσαφήνιση των βασικών συνιστωσών της εφαρμογής, τον προσδιορισμό των απαιτήσεων και τη μελέτη της αλληλεπίδρασης μεταξύ χρηστών, λογισμικού και υλικού. Μέσω αυτής της διαδικασίας διασφαλίζεται η οργάνωση της ανάπτυξης, η αποφυγή σφαλμάτων και η ευκολότερη συντήρηση του έργου στο μέλλον.

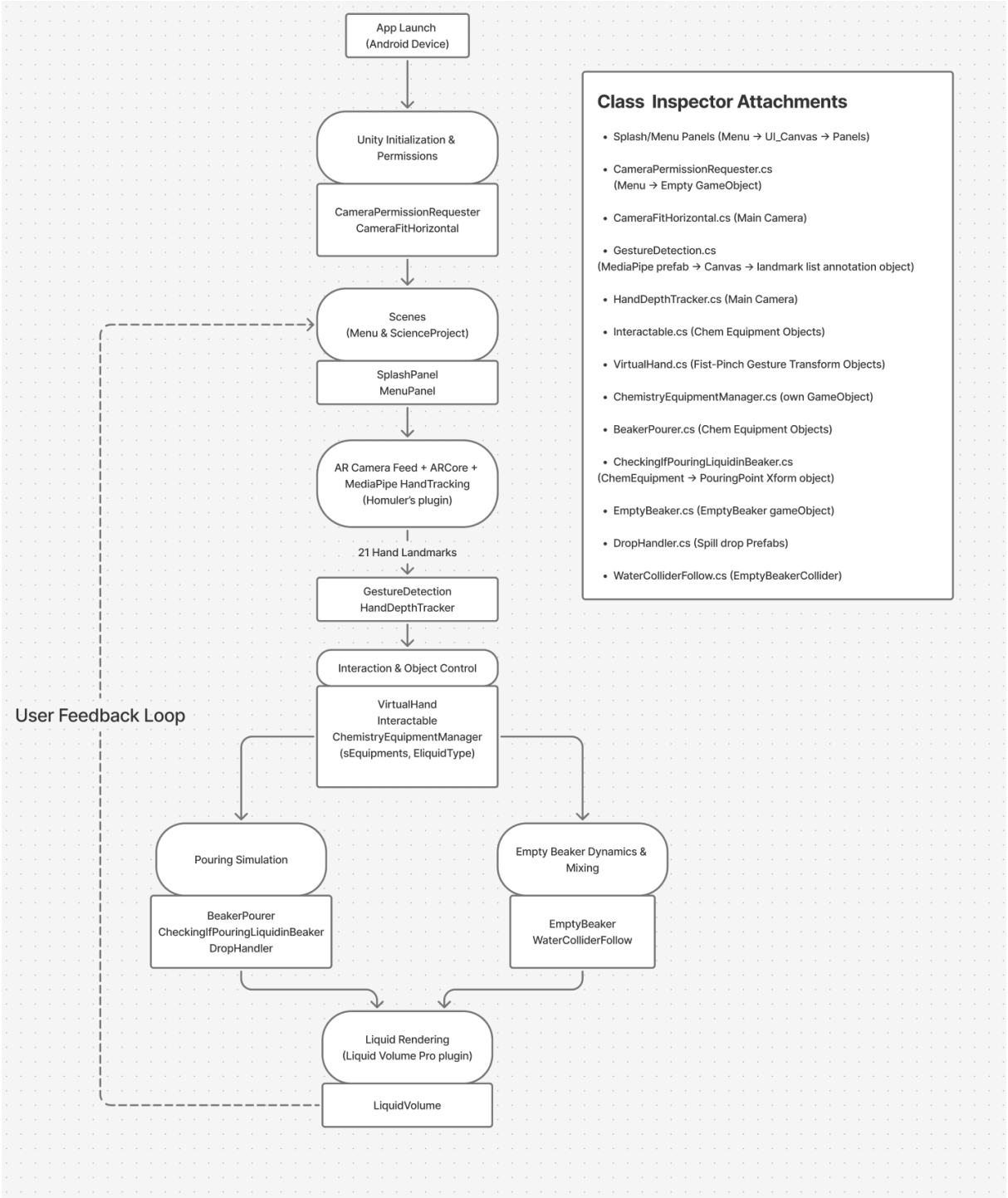
Στο παρόν κεφάλαιο παρουσιάζεται το γενικό πλαίσιο της αρχιτεκτονικής της εφαρμογής, το μοντέλο αλληλεπίδρασης χρήστη–συστήματος, καθώς και οι τεχνικές και λειτουργικές απαιτήσεις. Επιπλέον, καταγράφονται οι περιορισμοί και οι σχεδιαστικές αποφάσεις που λήφθηκαν κατά την προετοιμασία της ανάπτυξης, λαμβάνοντας υπόψη τόσο τις δυνατότητες των τεχνολογιών που επιλέχθηκαν, όσο και τις εκπαιδευτικές ανάγκες που καλείται να καλύψει η εφαρμογή.

3.2 Περιγραφή συνολικού συστήματος

Η εφαρμογή σχεδιάστηκε ως ένα εκπαιδευτικό εργαλείο επαυξημένης πραγματικότητας (AR) για φορητές συσκευές Android, με στόχο την αναπαράσταση ενός χημικού πειράματος μέσω διαδραστικών στοιχείων. Το σύστημα αξιοποιεί την κάμερα της συσκευής για την αναγνώριση του περιβάλλοντος και την προβολή τρισδιάστατων αντικειμένων, ενώ επιτρέπει στον χρήστη να αλληλεπιδρά με το εικονικό περιεχόμενο μέσω χειρονομιών, χωρίς τη χρήση πρόσθετου εξοπλισμού.

Το σύστημα αποτελείται από τα εξής βασικά υποσυστήματα:

- **Ενότητα Επαυξημένης Πραγματικότητας (AR Engine):** Χρησιμοποιεί το AR Foundation και το ARCore για την αναγνώριση επιφανειών, τον εντοπισμό της κάμερας στο χώρο και την τοποθέτηση των ψηφιακών αντικειμένων.
- **Ενότητα Αναγνώρισης Χειρονομιών (Hand Tracking):** Βασίζεται στο MediaPipe Hands για την ανίχνευση και παρακολούθηση κινήσεων του χεριού σε πραγματικό χρόνο.
- **Ενότητα Εκπαιδευτικού Περιεχομένου:** Περιλαμβάνει τα τρισδιάστατα μοντέλα (π.χ. ποτήρι, νερό, λάδι, μελάνι) που αντιπροσωπεύουν τα υλικά του πειράματος, καθώς και τα σενάρια διάδρασης που καθορίζουν τη συμπεριφορά τους κατά την αλληλεπίδραση.
- **Ενότητα Χρήστη (User Interaction Layer):** Συντονίζει τη σχέση μεταξύ των χειρονομιών και της αντίδρασης της εφαρμογής, επιτρέποντας λειτουργίες όπως «ρίχνω υγρό», «αναμιγνύω», ή «επαναφορά».



3.1 Διάγραμμα Ροής Εφαρμογής

3.3 Σχεδιασμός Αλληλεπίδρασης Χρήστη – Συστήματος

Η διεπαφή χρήστη και η εμπειρία αλληλεπίδρασης αποτελούν κρίσιμο στοιχείο για την επιτυχία μιας εκπαιδευτικής εφαρμογής επαυξημένης πραγματικότητας. Στην παρούσα εφαρμογή, ο σχεδιασμός της διάδρασης ακολουθεί ένα φυσικό και ενστικτώδη μοντέλο, στο οποίο ο χρήστης χειρίζεται το περιεχόμενο μέσω χειρονομιών, χωρίς χρήση φυσικών πλήκτρων ή βοηθητικών συσκευών.

Ο χρήστης κρατά τη φορητή συσκευή (κινητό ή tablet) με το ένα χέρι, ενώ με το άλλο πραγματοποιεί κινήσεις στον χώρο, μπροστά από την κάμερα. Η εφαρμογή χρησιμοποιεί το MediaPipe Hands για την

αναγνώριση της χειρονομίας και την αντιστοίχισή της σε μία εντολή ή ενέργεια εντός του εικονικού περιβάλλοντος.

3.3.1 Τυπική σειρά αλληλεπίδρασης

Η συνολική ροή λειτουργίας περιλαμβάνει την εξής σειρά:

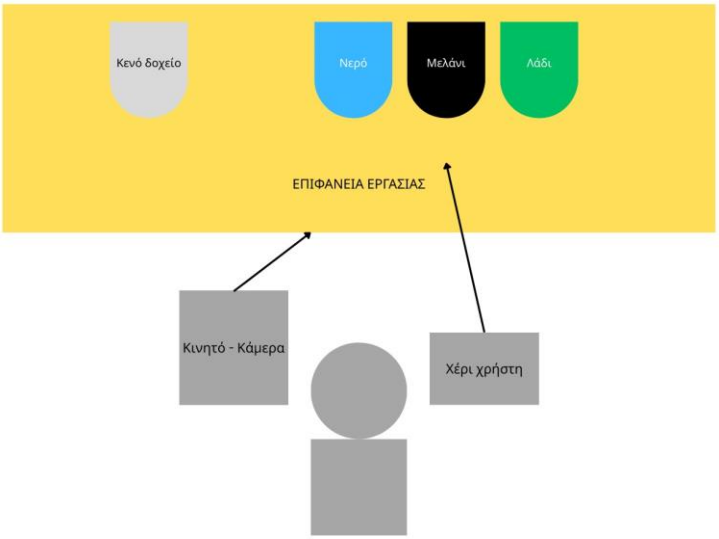
1. Ο χρήστης κρατάει τη συσκευή με το ένα χέρι, εκκινεί την εφαρμογή και εμφανίζεται το μενού.
2. Στη συνέχεια εμφανίζεται η AR σκηνή με τον εξοπλισμό και τις οδηγίες για τον χρήστη.
3. Ενεργοποιείται η κάμερα.
4. Ο χρήστης πραγματοποιεί χειρονομίες με το ελεύθερο χέρι του.
5. Το MediaPipe Hands αναγνωρίζει το χέρι και μεταβιβάζει πληροφορίες στο σύστημα.
6. Η εφαρμογή προσομοιώνει το αντίστοιχο χημικό αποτέλεσμα.

Η αρχιτεκτονική σχεδιάστηκε ώστε να είναι επεκτάσιμη, επιτρέποντας στο μέλλον την ενσωμάτωση περισσότερων πειραμάτων ή τρόπων αλληλεπίδρασης.

3.3.2 Εμπειρία χρήστη (User Perspective)

Η διάδραση βασίζεται σε ένα σενάριο «παρατήρησης και ενέργειας». Ο χρήστης παρακολουθεί τα αντικείμενα να εμφανίζονται στο τραπέζι του μέσω της οθόνης, και έχει την αίσθηση ότι τα «αγγίζει» ή τα χειρίζεται φυσικά. Το κάθε βήμα σχεδιάστηκε ώστε να προκαλεί οπτική και λογική ανατροφοδότηση, ενισχύοντας την κατανόηση και τη μνήμη.

Η χρήση χειρονομιών επιλέχθηκε έναντι κουμπιών ή γραφικών στοιχείων, καθώς ενισχύει τη βιωματική μάθηση, επιτρέποντας στον μαθητή να αναλάβει ενεργό ρόλο κατά την εξερεύνηση του φαινομένου.



3.2 Σχεδιάγραμμα σκηνής αλληλεπίδρασης

3.4 Λειτουργικές και Μη Λειτουργικές Απαιτήσεις

Ο καθορισμός απαιτήσεων αποτελεί βασικό στάδιο στον σχεδιασμό ενός συστήματος, καθώς θέτει τα όρια λειτουργίας και τις επιδιωκόμενες ιδιότητες του τελικού προϊόντος. Στην περίπτωση της παρούσας εφαρμογής, οι απαιτήσεις διακρίνονται σε **λειτουργικές**, που αφορούν το «τι κάνει» το σύστημα, και **μη λειτουργικές**, που αφορούν το «πώς το κάνει».

3.4.1 Λειτουργικές απαιτήσεις

Οι βασικές λειτουργικές απαιτήσεις της εφαρμογής περιλαμβάνουν:

- **Εκκίνηση AR περιβάλλοντος** κατά την ενεργοποίηση της κάμερας, μέσω ARCore.
- **Απεικόνιση εικονικού πειράματος** (νερό, λάδι, μελάνι) σε περιβάλλον επαυξημένης πραγματικότητας.
- **Αναγνώριση χειρονομιών χρήστη** μέσω της κάμερας και μετατροπή τους σε εντολές συστήματος (π.χ. ρίψη, ανάμιξη).
- **Διαχείριση επαναφοράς πειράματος**, με χειρονομία ή πλήκτρο.
- **Παροχή ανατροφοδότησης** στον χρήστη για κάθε ενέργεια (οπτική ή ηχητική).
- **Καθοδήγηση** του χρήστη μέσω γραπτών οδηγιών.

3.4.2 Μη λειτουργικές απαιτήσεις

Οι μη λειτουργικές απαιτήσεις διαμορφώνονται βάσει τεχνικών, παιδαγωγικών και εμπειρικών παραμέτρων:

- **Συμβατότητα με Android συσκευές** που υποστηρίζουν ARCore και κάμερα RGB.
- **Ομαλή απόδοση σε πραγματικό χρόνο**, με ελάχιστη καθυστέρηση στην ανίχνευση χειρονομιών και απόδοση σκηνής.
- **Φιλικότητα προς τον χρήστη (Usability):** Το περιβάλλον πρέπει να είναι απλό, κατανοητό και προσβάσιμο από μαθητές γυμνασίου.
- **Ευκολία εγκατάστασης και χρήσης:** Η εφαρμογή δεν απαιτεί εξωτερικό εξοπλισμό ή ρυθμίσεις από τον τελικό χρήστη.
- **Παιδαγωγική συνάφεια:** Το περιεχόμενο πρέπει να είναι σύμφωνο με τη σχολική ύλη της Χημείας Β΄ Γυμνασίου.
- **Επεκτασιμότητα:** Το σύστημα να μπορεί να δεχτεί μελλοντικά νέα πειράματα ή μορφές αλληλεπίδρασης χωρίς ριζική ανακατασκευή.

3.5 Τεχνικές προδιαγραφές

Η λειτουργία της εφαρμογής βασίζεται σε ένα σύνολο τεχνολογιών που απαιτούν συγκεκριμένες προδιαγραφές συσκευών και περιβάλλοντος για την απρόσκοπτη εμπειρία του χρήστη. Παράλληλα, αναγνωρίζονται περιορισμοί που απορρέουν από τις τεχνολογικές επιλογές, τις συνθήκες χρήσης και τη στοχευόμενη ηλικιακή ομάδα.

3.5.1 Τεχνικές προδιαγραφές

Για τη σωστή εκτέλεση της εφαρμογής απαιτείται η τήρηση των παρακάτω τεχνικών προδιαγραφών:

- **Λειτουργικό σύστημα:** Android 10 (API level 29) ή νεότερο.
- **Υποστήριξη ARCore:** Η συσκευή πρέπει να είναι συμβατή με την πλατφόρμα ARCore της Google.
- **Κάμερα:** Ενσωματωμένη πίσω κάμερα με ανάλυση τουλάχιστον 720p για αξιόπιστη ανίχνευση χειρονομιών.
- **Επεξεργαστής:** Τουλάχιστον 4 πυρήνες (quad-core) και αρχιτεκτονική ARM64.
- **RAM:** Τουλάχιστον 4 GB για αποφυγή καθυστερήσεων κατά την επεξεργασία των μοντέλων.
- **Διαθέσιμος χώρος αποθήκευσης:** Τουλάχιστον 300 MB.

3.5.2 Λογισμικά και βιβλιοθήκες

Η ανάπτυξη και λειτουργία της εφαρμογής βασίζεται σε:

- **Unity 2022.3.5f1** με εγκατεστημένα τα πακέτα:
    - AR Foundation
    - ARCore XR Plugin
- **Liquid Volume Pro (**Unity Asset Store**).**

3.5.3 Περιορισμοί

Παρότι η εφαρμογή σχεδιάστηκε για φορητότητα και ευχρηστία, υπάρχουν περιορισμοί που πρέπει να ληφθούν υπόψη:

- **Φωτισμός:** Η αναγνώριση χειρονομιών επηρεάζεται σημαντικά από τον φωτισμό του περιβάλλοντος. Ιδανικά απαιτείται ομοιόμορφος φυσικός ή τεχνητός φωτισμός.
- **Εμβέλεια κάμερας:** Η κάμερα πρέπει να βρίσκεται σε επαρκή απόσταση για την πλήρη απεικόνιση της επιφάνειας του χεριού.
- **Ορατότητα χειρονομιών:** Οι χειρονομίες πρέπει να εκτελούνται εντός του οπτικού πεδίου της κάμερας και με σαφήνεια, ώστε να εντοπίζονται αξιόπιστα από το MediaPipe Hands.
- **Περιβαλλοντική σταθερότητα:** Αστάθεια στην τοποθέτηση του κινητού ή μετακινήσεις του χρήστη ενδέχεται να επηρεάσουν την αναγνώριση των χειρονομιών.
- **Σχολικά δίκτυα ή περιορισμοί εγκατάστασης:** Εάν χρησιμοποιηθεί σε σχολικό περιβάλλον, ενδέχεται να απαιτείται έγκριση εγκατάστασης από τον διαχειριστή των συσκευών.

3.6 Επίλογος

Το παρόν κεφάλαιο παρουσίασε τη σχεδίαση και την αρχιτεκτονική της εφαρμογής, θέτοντας τις βάσεις για την υλοποίησή της. Αναλύθηκε η γενική δομή του συστήματος και ο τρόπος με τον οποίο αλληλεπιδρούν μεταξύ τους τα επιμέρους υποσυστήματα, όπως το περιβάλλον επαυξημένης πραγματικότητας, η αναγνώριση χειρονομιών και τα εικονικά αντικείμενα.

Ιδιαίτερη έμφαση δόθηκε στον σχεδιασμό της εμπειρίας του χρήστη, περιγράφοντας τη ροή της αλληλεπίδρασης από τη σκοπιά του μαθητή. Καταγράφηκαν επίσης οι λειτουργικές και μη λειτουργικές απαιτήσεις του συστήματος, καθώς και οι τεχνικές προδιαγραφές και περιορισμοί που σχετίζονται με την τεχνολογία και τις συνθήκες χρήσης.

Η ολοκληρωμένη ανάλυση του συστήματος στο στάδιο του σχεδιασμού επιτρέπει την οργάνωση της ανάπτυξης με σαφές πλάνο, διασφαλίζοντας τη σταθερότητα και τη λειτουργικότητα του τελικού προϊόντος. Στο επόμενο κεφάλαιο, παρουσιάζεται αναλυτικά η διαδικασία υλοποίησης της εφαρμογής, με έμφαση στα επιμέρους στάδια ανάπτυξης και ενσωμάτωσης των επιλεγμένων τεχνολογιών.

Κεφάλαιο 4ο: Ανάπτυξη και Υλοποίηση

4.1 Εισαγωγή

Το παρόν κεφάλαιο περιγράφει αναλυτικά τη διαδικασία ανάπτυξης της εφαρμογής επαυξημένης πραγματικότητας, η οποία υλοποιήθηκε στο πλαίσιο της παρούσας πτυχιακής εργασίας. Καταγράφονται τα εργαλεία και οι τεχνολογίες που χρησιμοποιήθηκαν, οι ρυθμίσεις που πραγματοποιήθηκαν στο περιβάλλον ανάπτυξης, καθώς και τα επιμέρους στάδια κατασκευής της εφαρμογής.

Η ανάπτυξη βασίστηκε στη μηχανή **Unity**, με στόχο τη δημιουργία μιας λειτουργικής εμπειρίας **AR** σε φορητή **Android** συσκευή. Για την επίτευξη του στόχου αξιοποιήθηκαν βιβλιοθήκες όπως το **AR Foundation**, για την υποστήριξη λειτουργιών επαυξημένης πραγματικότητας, και το **MediaPipe**, μέσω του **Unity Plugin** του **homuler** [9], για την αναγνώριση χειρονομιών (hand tracking) σε πραγματικό χρόνο.

Η υλοποίηση ακολούθησε μια βήμα προς βήμα προσέγγιση: ξεκινώντας από τη δημιουργία του έργου και τη ρύθμιση των απαραίτητων εξαρτήσεων, προχώρησε στη διαμόρφωση της σκηνής AR, στην ενσωμάτωση του συστήματος αναγνώρισης χεριών, και τελικά στην κατασκευή του διαδραστικού σεναρίου, το οποίο βασίζεται στην αναπαράσταση ενός χημικού πειράματος με νερό, λάδι και μελάνι.

Για την οπτική αναπαράσταση των υγρών και του χημικού εξοπλισμού, χρησιμοποιήθηκε το εμπορικό εργαλείο **Liquid Volume Pro** [16], το οποίο προσφέρει δυνατότητες ρεαλιστικής προσομοίωσης υγρών, με χρήση διαφανούς απόδοσης (transparency), επιπέδων (layers) και κινούμενων επιφανειών (turbulence).

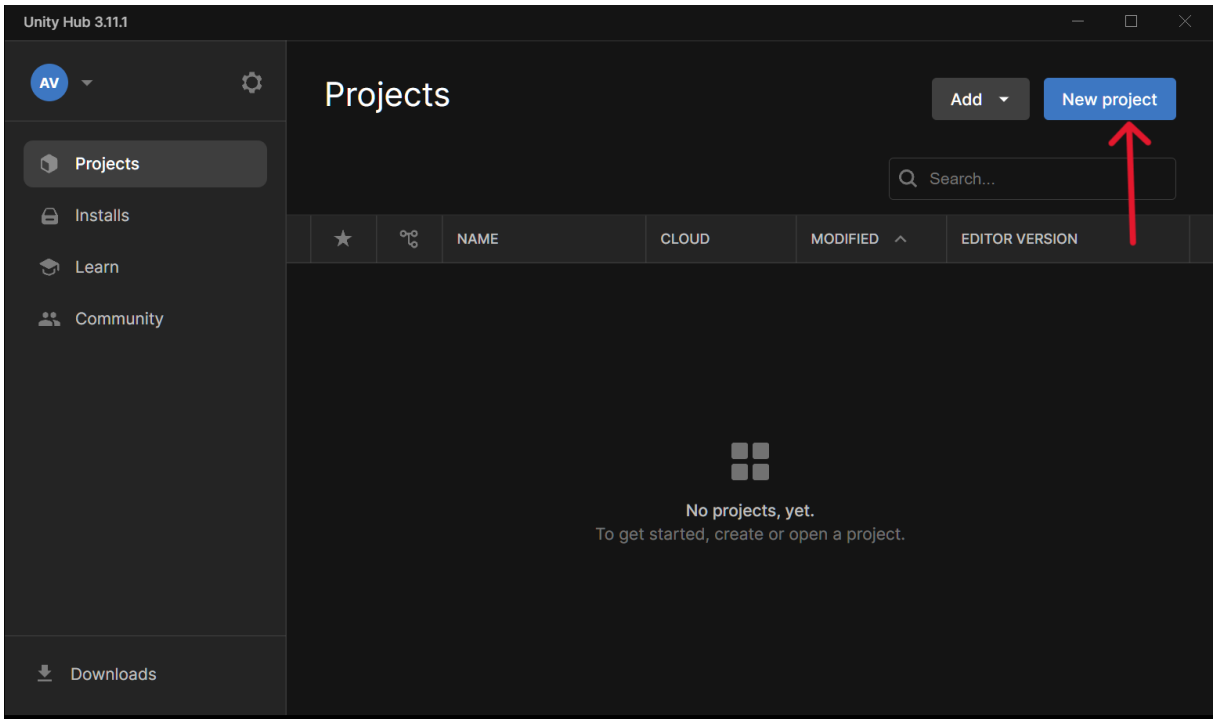
4.2 Περιβάλλον ανάπτυξης και αρχικοποίηση έργου

Η ανάπτυξη της εφαρμογής πραγματοποιήθηκε στη μηχανή παιχνιδιών **Unity 2022.3.5f1**, η οποία προσφέρει ευρεία υποστήριξη για εφαρμογές επαυξημένης πραγματικότητας μέσω του πακέτου **AR Foundation**.

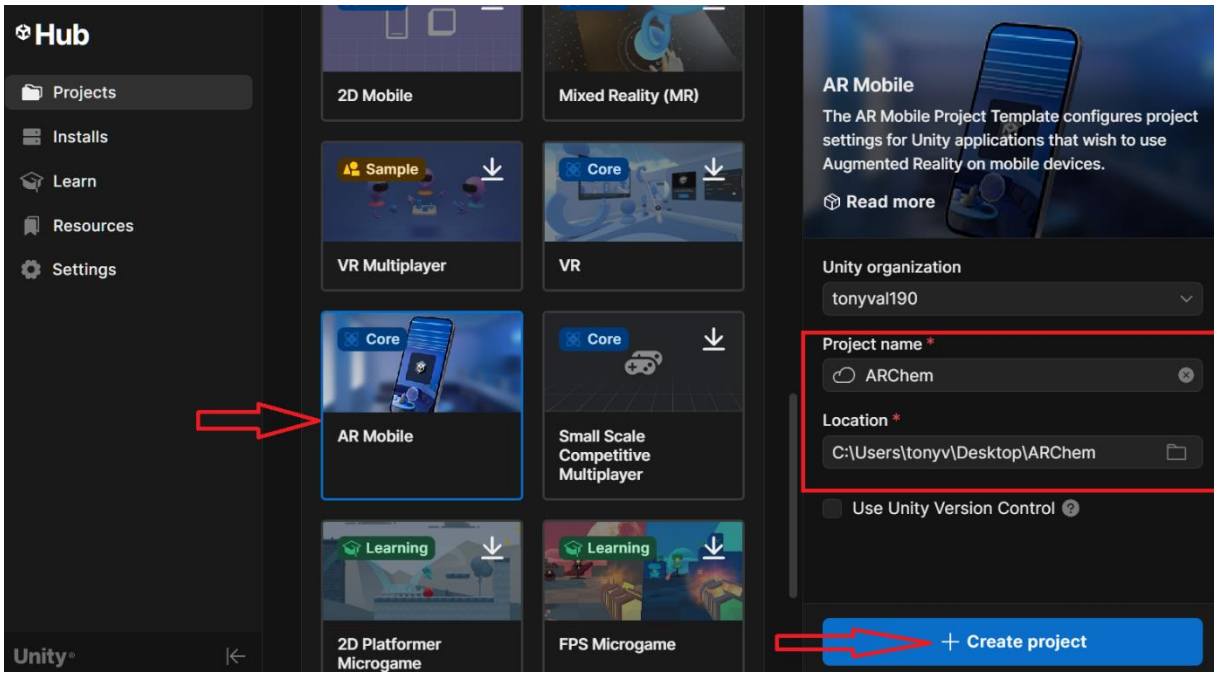
Η επιλογή της συγκεκριμένης πλατφόρμας βασίστηκε στην ευκολία ενσωμάτωσης εξωτερικών βιβλιοθηκών μέσω του **Package Manager**, στη μεγάλη υποστήριξη για **Android** συσκευές και στην ευελιξία που προσφέρει στον σχεδιασμό διαδραστικών περιβαλλόντων.

4.2.1 Δημιουργία Έργου με Unity Hub

Η διαδικασία ξεκίνησε με τη δημιουργία νέου έργου από το **Unity Hub**, όπου επιλέχθηκε το πρότυπο **AR Mobile**, το οποίο προσφέρει βελτιωμένη απόδοση σε φορητές συσκευές και είναι πλήρως συμβατό με το AR Foundation. Το έργο ονομάστηκε κατάλληλα, και αποθηκεύτηκε σε τοπικό φάκελο του υπολογιστή.



4.1 Δημιουργία νέου έργου Unity μέσω του Unity Hub



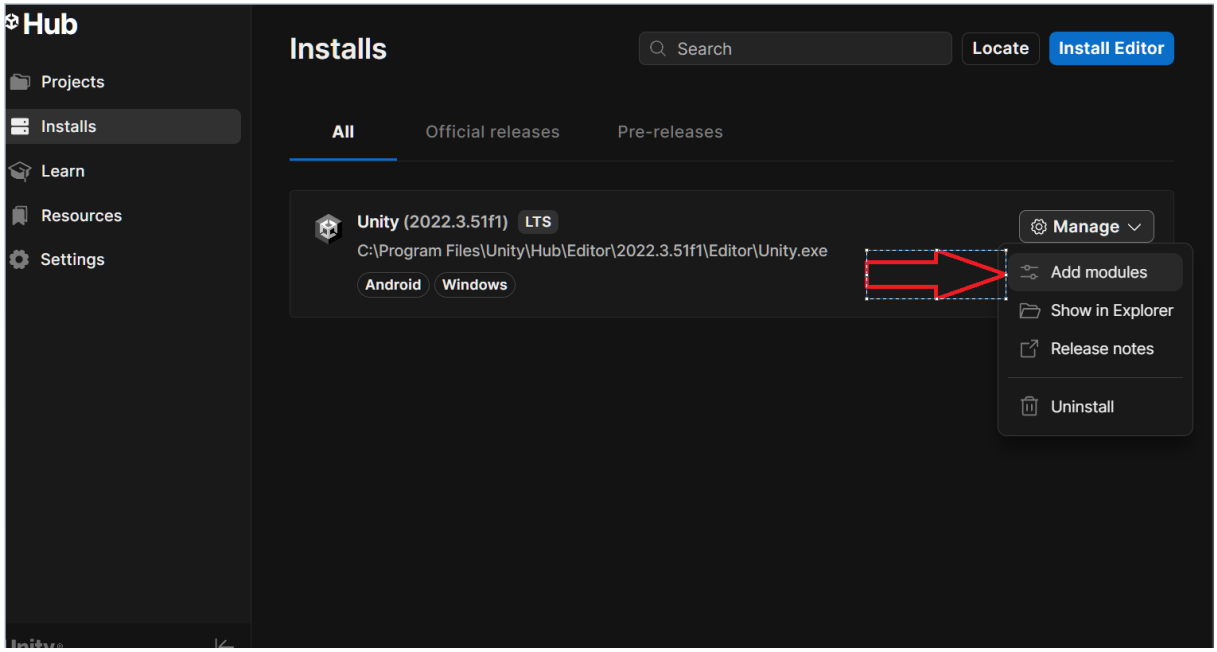
4.2 Επιλογή AR Mobile και καθορισμός τοποθεσίας - ονόματος του έργου

Με την ολοκλήρωση της αρχικοποίησης του έργου, πραγματοποιήθηκε η πρώτη σύνδεση με τον **Package Manager** της Unity, ώστε να επιβεβαιωθεί η εγκατάσταση των βασικών εξαρτήσεων και να προστεθούν τα πακέτα που απαιτούνται για την ανάπτυξη AR εφαρμογών σε Android.

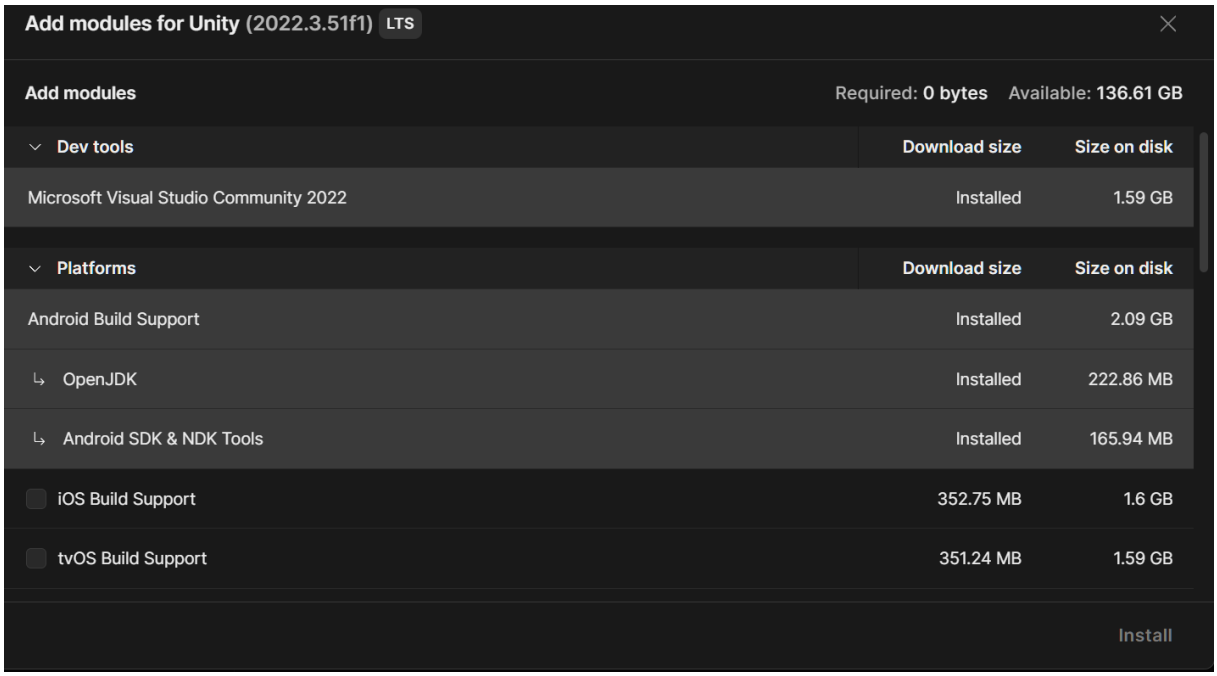
4.2.2 Υποστήριξη Android Συσκευών

Για την υποστήριξη Android συσκευών, ήταν απαραίτητη η εγκατάσταση των κατάλληλων εργαλείων και SDKs μέσω του **Unity Hub**. Πιο συγκεκριμένα, κατά την εγκατάσταση του Unity Editor,

επιλέχθηκαν επιπλέον modules όπως το **Android Build Support**, το **OpenJDK**, το **Android SDK & NDK Tools**, τα οποία είναι απαραίτητα για την εξαγωγή και εγκατάσταση της εφαρμογής σε Android συσκευές.



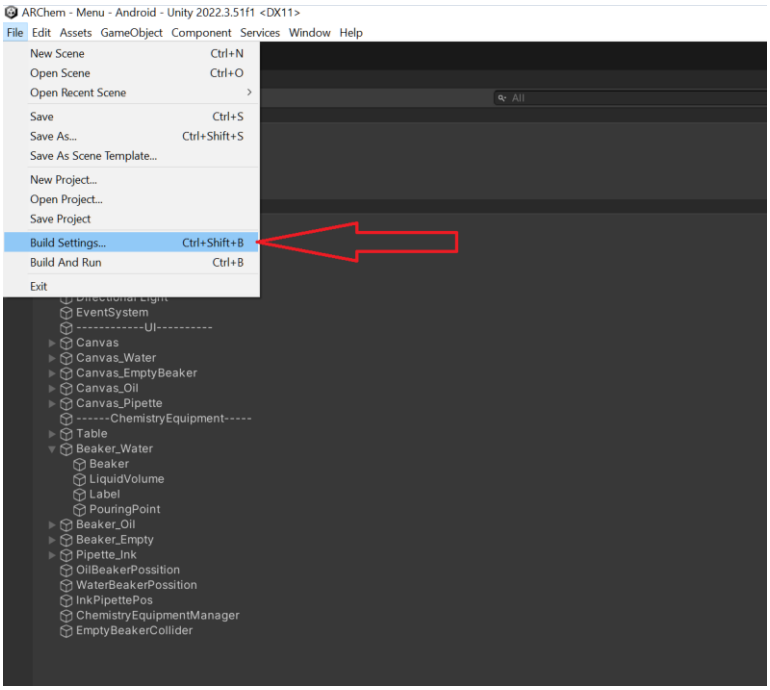
4.3 Εγκατάσταση επιπλέον modules στο unity project



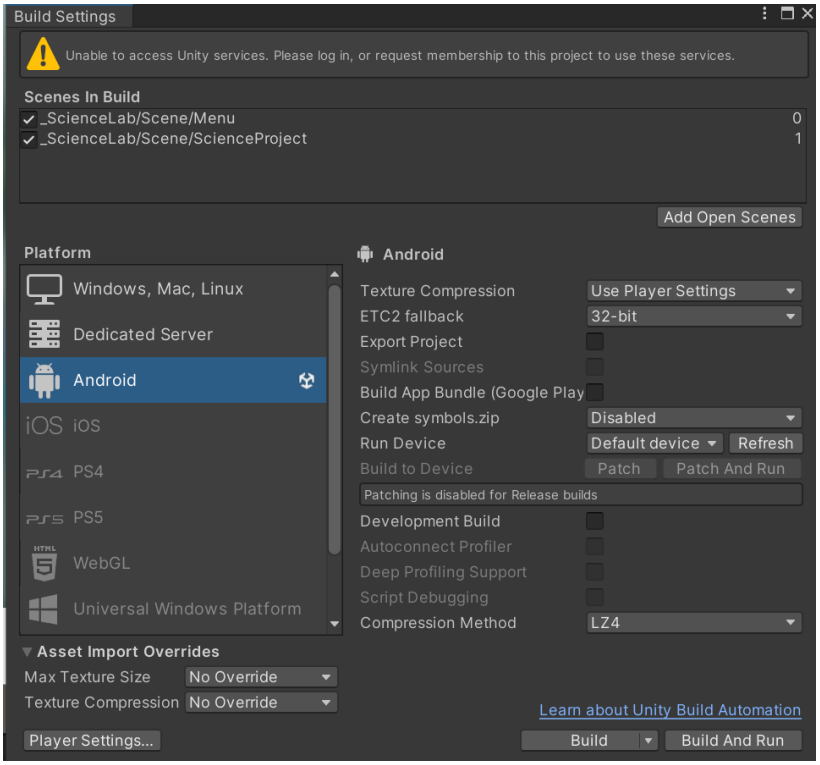
4.4 Επιβεβαίωση εγκατάστασης Android SDK & NDK και JDK μέσω του unity hub

4.2.3 Build Settings και Player Settings

Αμέσως μετά την έναρξη του έργου, το **Build Target** άλλαξε σε Android μέσω του μενού **File > Build Settings**. Εκεί προστέθηκαν οι σκηνές **Menu** και **ScienceProject** και επιλέχθηκε η πλατφόρμα Android.



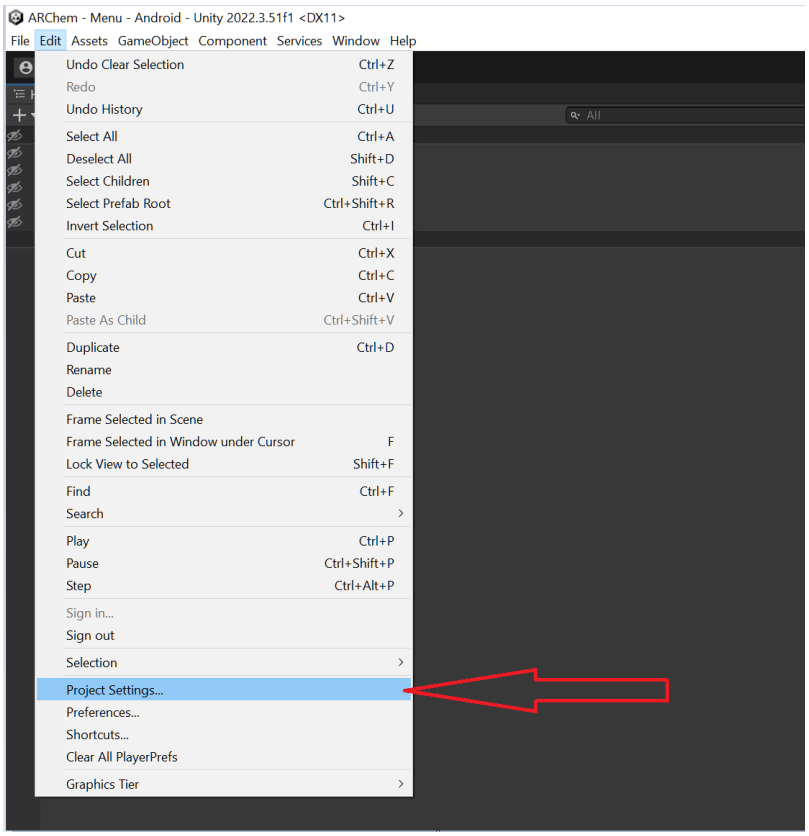
4.5 File - build profiles unity settings



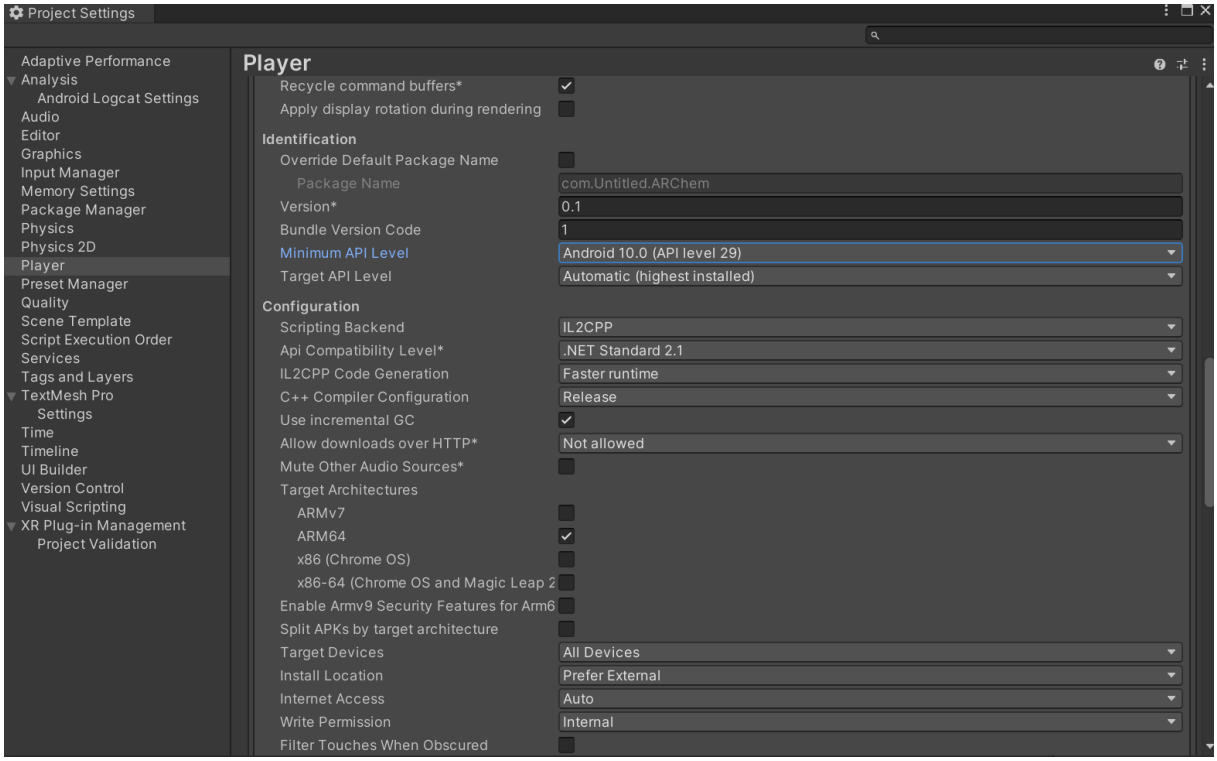
4.6 Αλλαγή build target σε Android

Έπειτα, μέσω του **Player Settings**, πραγματοποιήθηκαν οι εξής ρυθμίσεις:

- Ορισμός μοναδικού Package Name
- Ελάχιστη έκδοση Android 10
- Επιλογή Scripting Backend: IL2CPP
- Ενεργοποίηση της αρχιτεκτονικής ARM64, η οποία απαιτείται από το Google Play.



4.7 Edit - Project settings unity settings

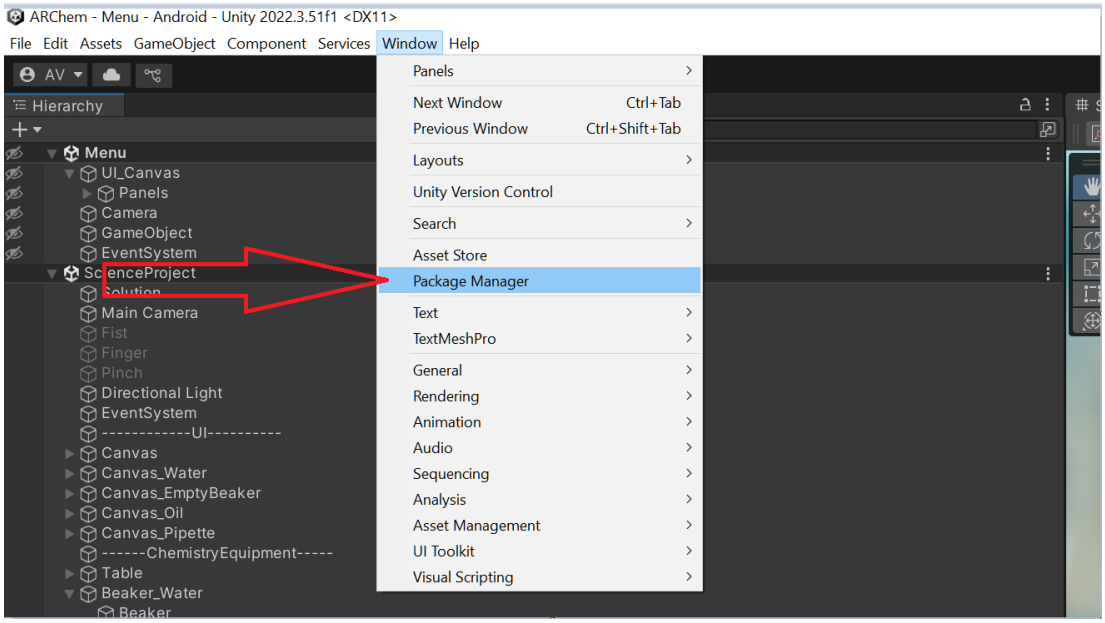


4.8 Ρυθμίσεις Player για Android

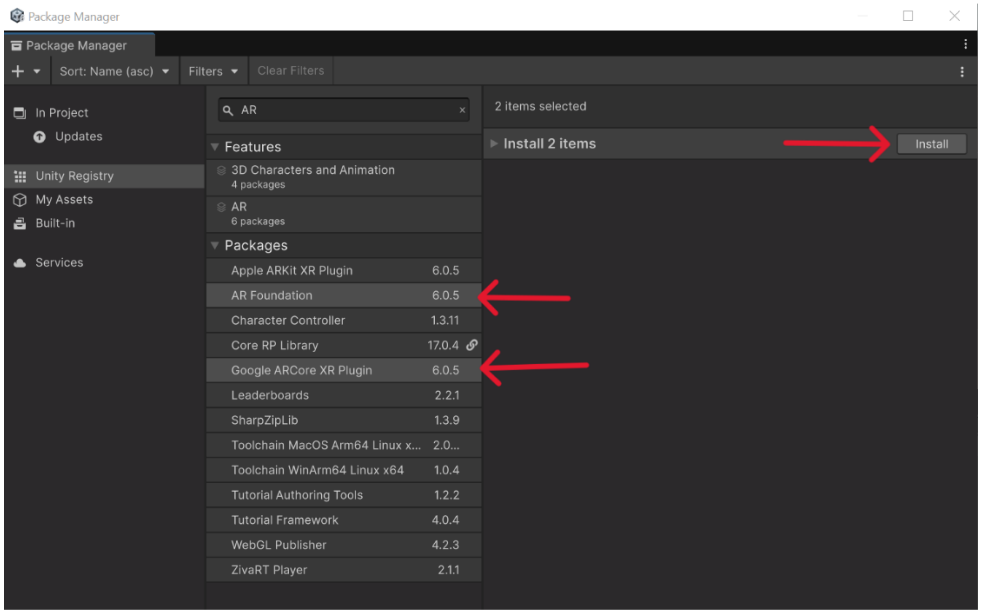
4.2.4 Εγκατάσταση Πακέτων μέσω Unity Package Manager

Ακολούθησε η εγκατάσταση των απαραίτητων πακέτων μέσω του **Package Manager**, μέσα από το μενού Window > Package Manager. Τα βασικά πακέτα που προστέθηκαν ήταν:

- AR Foundation (com.unity.xr.arfoundation)
- Google ARCore XR Plugin (com.unity.xr.arcore)
- TextMeshPro (προεγκατεστημένο)
- XR Plugin Management
- UGUI (Unity UI) (προεγκατεστημένο)
- MediaPipe Hand Tracking (com.github.homuler.mediapipe) (χειροκίνητα προστιθέμενο ως embedded package μέσω packages-lock.json)
- LiquidVolume Pro (εισαγόμενο asset μέσω Assets/)



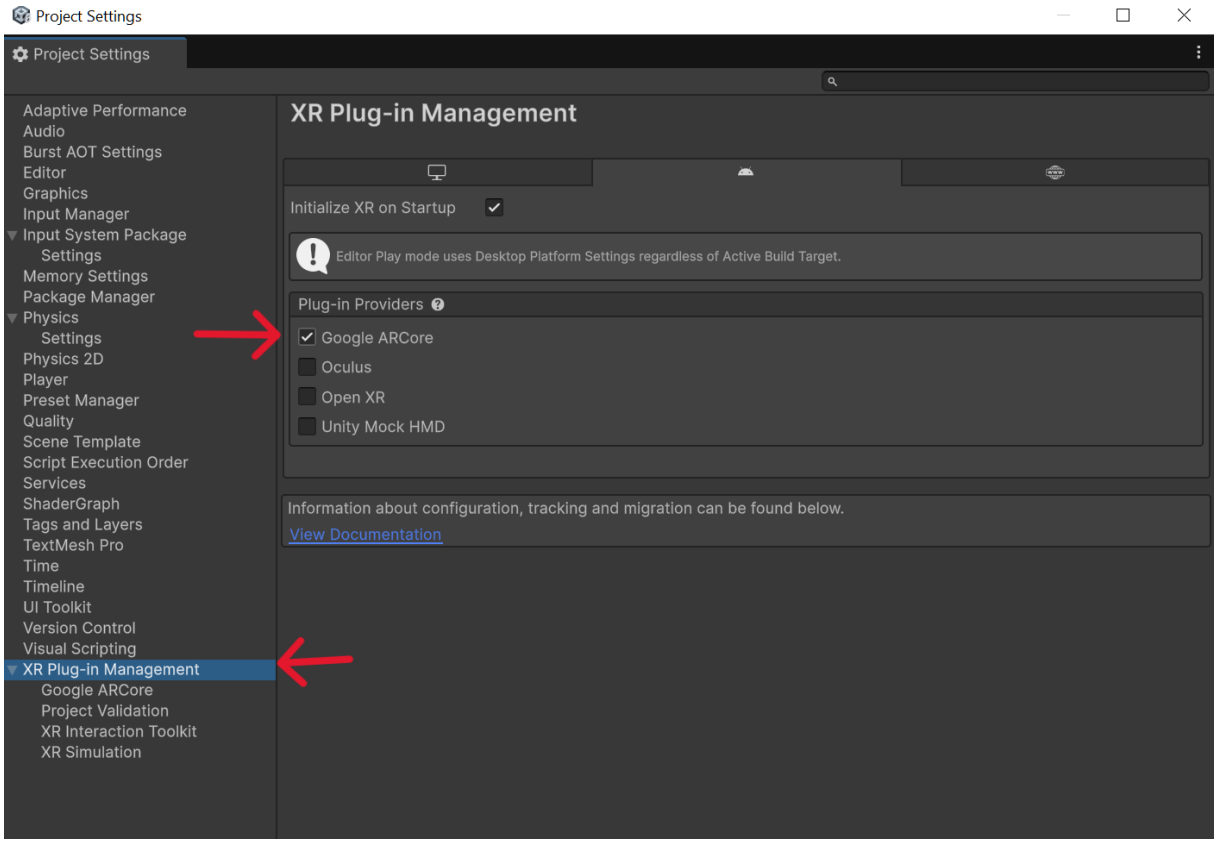
4.9 Window - Package manager unity settings



4.10 Εγκατάσταση AR Foundation - Google ARCore πακέτων

4.2.5 Ενεργοποίηση XR Plug-in Management

Τέλος, μέσω του μενού **XR Plug-in Management**, ενεργοποιήθηκε το σύστημα XR για την πλατφόρμα Android και επιλέχθηκε ως **AR Provider** το **Google ARCore**.



4.11 Ενεργοποίηση Google ARCore στο XR Plug-in

Με την ολοκλήρωση των παραπάνω ρυθμίσεων, το περιβάλλον ανάπτυξης ήταν πλέον έτοιμο για την κατασκευή της AR εφαρμογής και τη σύνδεση με το σύστημα ανίχνευσης χειρονομιών.

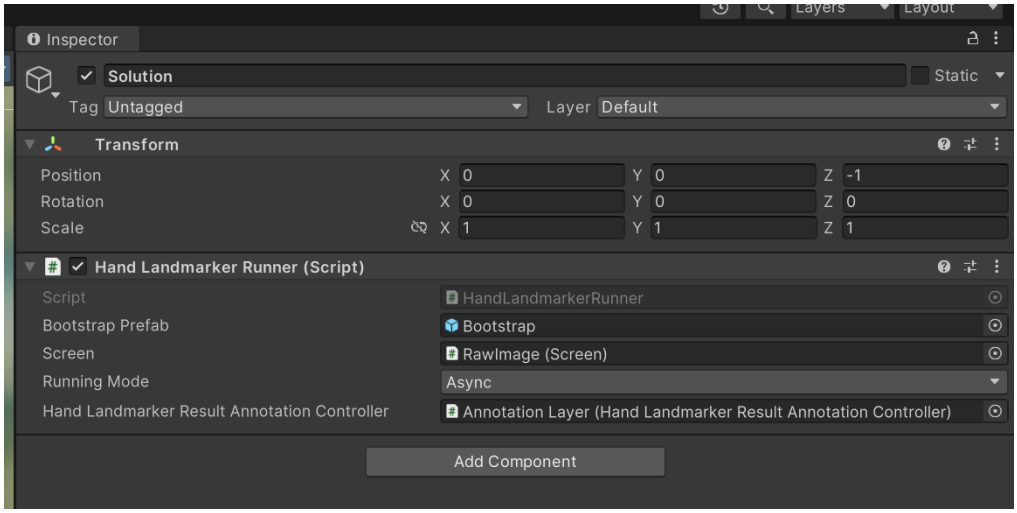
4.3 Υλοποίηση Hand Tracking

Η αλληλεπίδραση του χρήστη με την εφαρμογή υλοποιείται μέσω **ανίχνευσης χειρονομιών** σε πραγματικό χρόνο, αξιοποιώντας τη βιβλιοθήκη **MediaPipe** της Google. Η ενσωμάτωσή της στην πλατφόρμα Unity πραγματοποιείται με τη χρήση του **MediaPipe Unity Plugin** του **homuler**, το οποίο παρέχει πρόσβαση στα **21 σημεία αναφοράς (landmarks)** του χεριού, όπως αυτά παράγονται από το μοντέλο HandTracking της MediaPipe.

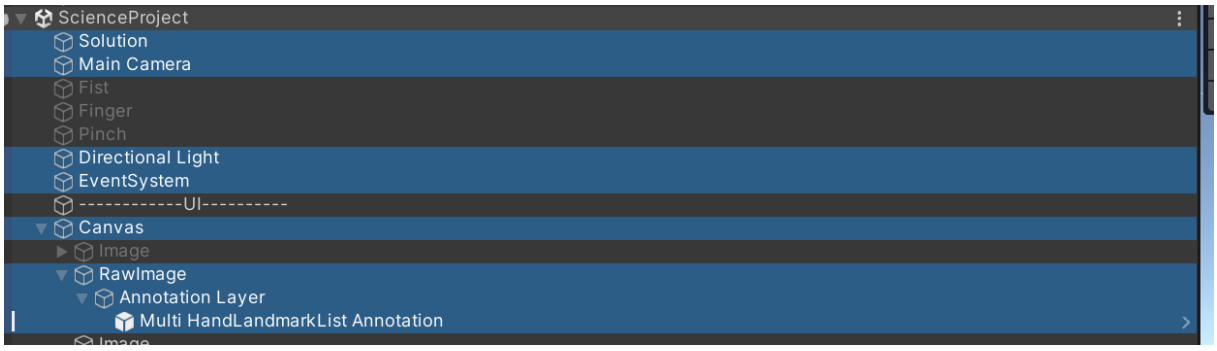
4.3.1 Τεχνική ενσωμάτωσης

Το πακέτο **com.github.homuler.mediapipe** ενσωματώθηκε στο έργο ως **embedded package** και η ρύθμισή του πραγματοποιήθηκε μέσω prefab που παρέχει έτοιμα Graphs για ανίχνευση χεριών. Με την έναρξη της σκηνής ScienceProject.unity, ενεργοποιείται το prefab εντοπισμού, το οποίο επεξεργάζεται σε πραγματικό χρόνο την **κάμερα της συσκευής Android**, εντοπίζει το χέρι του χρήστη, και παράγει τις θέσεις των 21 landmark σημείων.

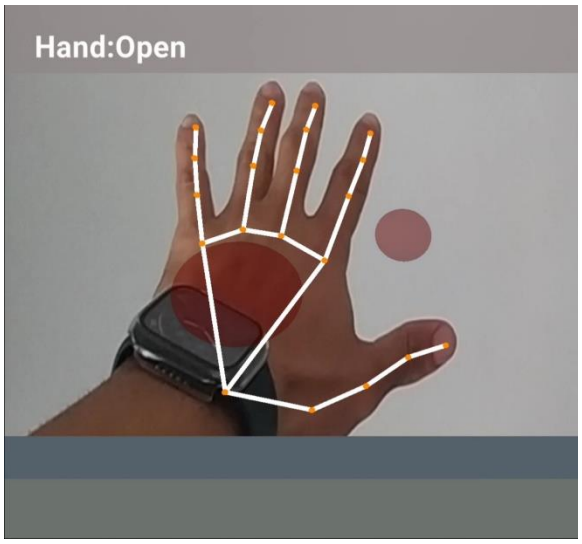
Αυτά τα landmarks μετατρέπονται σε **GameObjects/Transforms** στο Unity, δίνοντας τη δυνατότητα χρήσης τους για ανίχνευση και απόδοση κινήσεων, μέσω των custom scripts (κλάσεις) του project.



4.12 MediaPipe HandLandmarker Runner prefab 1



4.13 Mediapipe HandLandmarker Runner prefab 2



4.14 Θέσεις των 21 Landmarks της Mediapipe

4.3.2 Εντοπισμός Χειρονομιών

4.3.2.1 GestureDetection.cs

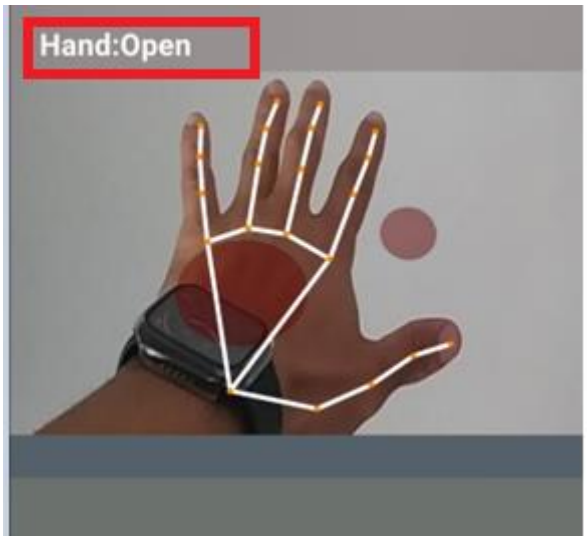
Η κλάση **GestureDetection** αποτελεί τον πυρήνα του συστήματος αναγνώρισης χειρονομιών. Χρησιμοποιεί τα δεδομένα που παρέχει το **MediaPipe Hand Tracking** (μέσω του homuler plugin) για να ανιχνεύσει τα 21 landmarks του χεριού, να εξομαλύνει τις θέσεις τους, να υπολογίσει γωνίες κάμψης των δακτύλων, να υπολογίσει αποστάσεις μεταξύ κρίσιμων σημείων, και τελικά να ταξινομήσει την κίνηση του χρήστη σε μια από τις προκαθορισμένες χειρονομίες: **Open**, **Fist**, **Pinch**, **Point** ή **None**. Η πληροφορία της τρέχουσας χειρονομίας (**currentGesture**) είναι διαθέσιμη σε ολόκληρη την εφαρμογή μέσω μιας στατικής αναφοράς (**instance**), ώστε οι υπόλοιπές κλάσεις να τη χρησιμοποιούν άμεσα.

Στην αρχή της κλάσης δηλώνονται οι βασικές δομές και οι μεταβλητές:

```
1 reference
public class GestureDetection : MonoBehaviour
{
    2 references
    public static GestureDetection instance;


    1 reference
    public TextMeshProUGUI handDetails;

    16 references | 3 references | 2 references | 2 references | 2 references | 2 references
    public enum HandGesture { None, Open, Fist, Pinch, Point }

    [Header("Output")]
    3 references
    public HandGesture currentGesture = HandGesture.None;
```

4.15 GestureDetection.cs 1

Η μεταβλητή **instance** χρησιμοποιείται για να εφαρμόσει το μοτίβο **singleton**, κάνοντας την κλάση εύκολα προσβάσιμη από τις υπόλοιπες κλάσεις. Το enumeration **HandGesture** ορίζει όλες τις χειρονομίες που υποστηρίζει η εφαρμογή, ενώ η **currentGesture** κρατάει την τρέχουσα κατάσταση. Επιπλέον, η handDetails χρησιμοποιείται για να εμφανίζει πληροφορίες στο UI κατά τη φάση debugging μέσω του **TextMeshPro**.



4.16 GestureDetection - handDetails

Ακολουθεί η ενότητα που χειρίζεται τα landmarks:

```
[SerializeField] private Transform handLandmarkHolder;
5 references
public List<Transform> landmarks = new();
9 references
private Vector3[] smoothed = new Vector3[21];
2 references
private Vector3[] previous = new Vector3[21];
1 reference
[Range(0f, 1f)] public float smoothFactor = 0.4f;
```

4.17 GestureDetection.cs 2

Η μεταβλητή **handLandmarkHolder** είναι το γονικό αντικείμενο όπου το MediaPipe αποθηκεύει τα landmarks του χεριού. Η λίστα **landmarks** περιέχει τα 21 σημεία, ενώ οι πίνακες **smoothed** και **previous** χρησιμοποιούνται για την εξομάλυνση της κίνησης, ώστε να μειωθεί το τρεμόπαιγμα του χεριού. Το **smoothFactor** καθορίζει πόσο έντονα εφαρμόζεται το smoothing.

Στη συνέχεια, ορίζονται τα όρια που χρησιμοποιούνται για να ανιχνευτούν οι χειρονομίες:

```
[Header("Thresholds")]
0 references
public float curlThresholdCurl = 60f;
0 references
public float curlThresholdStraight = 160f;
1 reference
public float pinchDistanceThreshold = 0.22f;
1 reference
public float activationDelay = 0.1f;
```

4.18 GestureDetection.cs 3

Η **curlThresholdCurl** και η **curlThresholdStraight** καθορίζουν τα όρια γωνιών για να θεωρηθεί ότι ένα δάχτυλο είναι κλειστό ή τεντωμένο. Το **pinchDistanceThreshold** είναι η απόσταση ανάμεσα στον αντίχειρα και τον δείκτη για να αναγνωριστεί το Pinch. Η **activationDelay** χρησιμοποιείται για **debouncing**, ώστε να αποφευχθεί η εναλλαγή χειρονομιών λόγω μικρών τυχαίων κινήσεων.

Η επόμενη ενότητα αφορά τα Unity events που ενεργοποιούνται όταν ανιχνεύεται κάποια χειρονομία:

```
[Header("Events")]
1 reference | 1 reference | 1 reference | 1 reference
public UnityEvent OnFist, OnOpen, OnPinch, OnPoint;
```

4.19 GestureDetection.cs 4

Αυτά μπορούν να συνδεθούν στο Unity Inspector, ώστε να ενεργοποιούν άλλες λειτουργίες όταν ο χρήστης κάνει μια συγκεκριμένη χειρονομία.

Στη μέθοδο **Awake()** αρχικοποιείται το **singleton**, ενώ στο **OnEnable()** γίνεται έλεγχος αν υπάρχουν διαθέσιμα landmarks και, αν όχι, ξεκινά ένα **coroutine**:

```
0 references
private void Awake()
{
    if (instance == null) instance = this;
}

0 references
private void OnEnable()
{
    if (landmarks.Count == 0)
        StartCoroutine(CheckForLandmarks());
}
```

4.20 GestureDetection.cs 5

Το **CheckForLandmarks()** περιμένει μέχρι να φορτωθούν τα landmarks από το MediaPipe και στη συνέχεια τα αποθηκεύει:

```
1 reference
IEnumerator CheckForLandmarks()
{
    while (handLandmarkHolder.childCount == 0)
        yield return null;

    Transform root = handLandmarkHolder.GetChild(0).GetChild(0);
    for (int i = 0; i < root.childCount; i++)
    {
        landmarks.Add(root.GetChild(i));
    }

    handReady = true;
```

4.21 GestureDetection.cs 6

Η κύρια λογική βρίσκεται μέσα στη μέθοδο **Update()**. Πρώτα, γίνεται smoothing των θέσεων, με τη μέθοδο **Lerp**:

```
private void Update()
{
    if (!handReady || landmarks.Count < 21) return;

    // Smooth landmark positions
    for (int i = 0; i < 21; i++)
    {
        Vector3 current = landmarks[i].position;
        smoothed[i] = Vector3.Lerp(previous[i], current, smoothFactor);
        previous[i] = smoothed[i];
    }
```

4.22 GestureDetection.cs 7

Τι είναι το Lerp;

Η λέξη **Lerp** προέρχεται από το *Linear Interpolation* (γραμμική παρεμβολή).

Ο μαθηματικός τύπος είναι:

$$Lerp(a, b, t) = a + (b - a) * t$$

- a → αρχική τιμή (π.χ. παλιά θέση)
- b → τελική τιμή (π.χ. καινούργια θέση)
- t → ποσοστό (0 έως 1) που καθορίζει πόσο κοντά στο b θα πάμε
- Αν $\boldsymbol{t = 0}$ → μένουμε στο a
- Αν $\boldsymbol{t = 1}$ → πηγαίνουμε κατευθείαν στο b
- Αν $\boldsymbol{t = 0.5}$ → παίρνουμε το ενδιάμεσο σημείο (50% a + 50% b)

Στη Unity, όταν μιλάμε για **Vector3.Lerp(a, b, t)**, αυτό σημαίνει ότι βρίσκουμε μια **νέα θέση στο διάστημα μεταξύ των δύο θέσεων** a και b.

Πώς εφαρμόζεται εδώ (GestureDetection.cs):

1. **Current**

   Παίρνουμε την τρέχουσα θέση του landmark από το tracking (π.χ. άκρη δακτύλου).

2. $\boldsymbol{Lerp(previous[i], current, smoothFactor)}$

Δεν χρησιμοποιούμε την «απόλυτη» νέα θέση, αλλά μια **μεικτή θέση** ανάμεσα:

- previous[i] = η θέση του landmark στο προηγούμενο frame,
- current = η νέα θέση που δίνει το tracking,
- smoothFactor = πόσο να «ακολουθήσουμε» τη νέα θέση.

Αν **smoothFactor** = 0.4:

→ το τελικό smoothed[i] θα είναι 40% πιο κοντά στο current, 60% στο previous.

Έτσι αποφεύγουμε το «τρέμουλο» που προκαλεί το hand tracking.

3. previous[i] = smoothed[i]

Αποθηκεύουμε την εξομαλυμένη τιμή για το επόμενο frame. Έτσι συνεχίζεται το smoothing από frame σε frame.

Στη συνέχεια, υπολογίζονται οι γωνίες κάμψης κάθε δαχτύλου με τη βοήθεια της **GetFingerCurl**():

```
// Get curl angles
index = GetFingerCurl(5, 6, 8);
middle = GetFingerCurl(9, 10, 12);
ring = GetFingerCurl(13, 14, 16);
pinky = GetFingerCurl(17, 18, 20);
thumb = GetFingerCurl(1, 2, 4);
```

4.23 GestureDetection.cs 8

Η **GetFingerCurl**() παίρνει τρία landmarks για κάθε δάχτυλο, υπολογίζει τα διανύσματα μεταξύ τους και επιστρέφει τη γωνία:

```
5 references
float GetFingerCurl(int baseIndex, int midIndex, int tipIndex)
{
    Vector3 basePos = smoothed[baseIndex];
    Vector3 midPos = smoothed[midIndex];
    Vector3 tipPos = smoothed[tipIndex];

    Vector3 dir1 = (basePos - midPos).normalized;
    Vector3 dir2 = (tipPos - midPos).normalized;

    return Vector3.Angle(dir1, dir2);
}
```

4.24 GestureDetection.cs 9

Με βάση αυτές τις γωνίες και την απόσταση μεταξύ αντίχειρα και δείκτη, καλείται η **DetectGesture**() για να αποφασιστεί η τρέχουσα χειρονομία:

```
float pinchDist = Vector3.Distance(smoothed[4], smoothed[8]); // thumb tip to index tip
float handScale = Vector3.Distance(smoothed[0], smoothed[9]);
normalizedPinchDist = pinchDist / handScale;
// Determine gesture
HandGesture detected = DetectGesture(index, middle, ring, pinky, thumb, normalizedPinchDist);
```

4.25 GestureDetection.cs 10

Η **DetectGesture**() ταξινομεί την κίνηση συγκρίνοντας τις τιμές με προκαθορισμένα κατώφλια (thresholds):

```
HandGesture DetectGesture(float index, float middle, float ring, float pinky, float thumb, float pinchDistance)
{
    // Fist: all fingers curled
    if (index < 140 && middle < 140 && ring < 140 && pinky < 140 /*&& thumb < 140*/)
        return HandGesture.Fist;

    // Pinch: thumb and index curled but tips close together
    if (pinchDistance < pinchDistanceThreshold && index < 160 && thumb > 160)
        return HandGesture.Pinch;

    // Pointing: index straight, others curled
    if (index > 160 && middle < 160 && ring < 160 && pinky < 160)
        return HandGesture.Point;

    // Open: all fingers straight
    if (index > 160 && middle > 165 && ring > 165 && pinky > 165 && thumb > 165)
        return HandGesture.Open;

    return HandGesture.None;
}
```

4.26 GestureDetection 11

Συγκρίνει τις γωνίες κάθε δαχτύλου με **κατώφλια (140°, 160°, 165°)** και την απόσταση αντίχειρα–δείκτη (**0.22**). Αν οι συνθήκες ταιριάζουν, αναγνωρίζεται η αντίστοιχη χειρονομία.

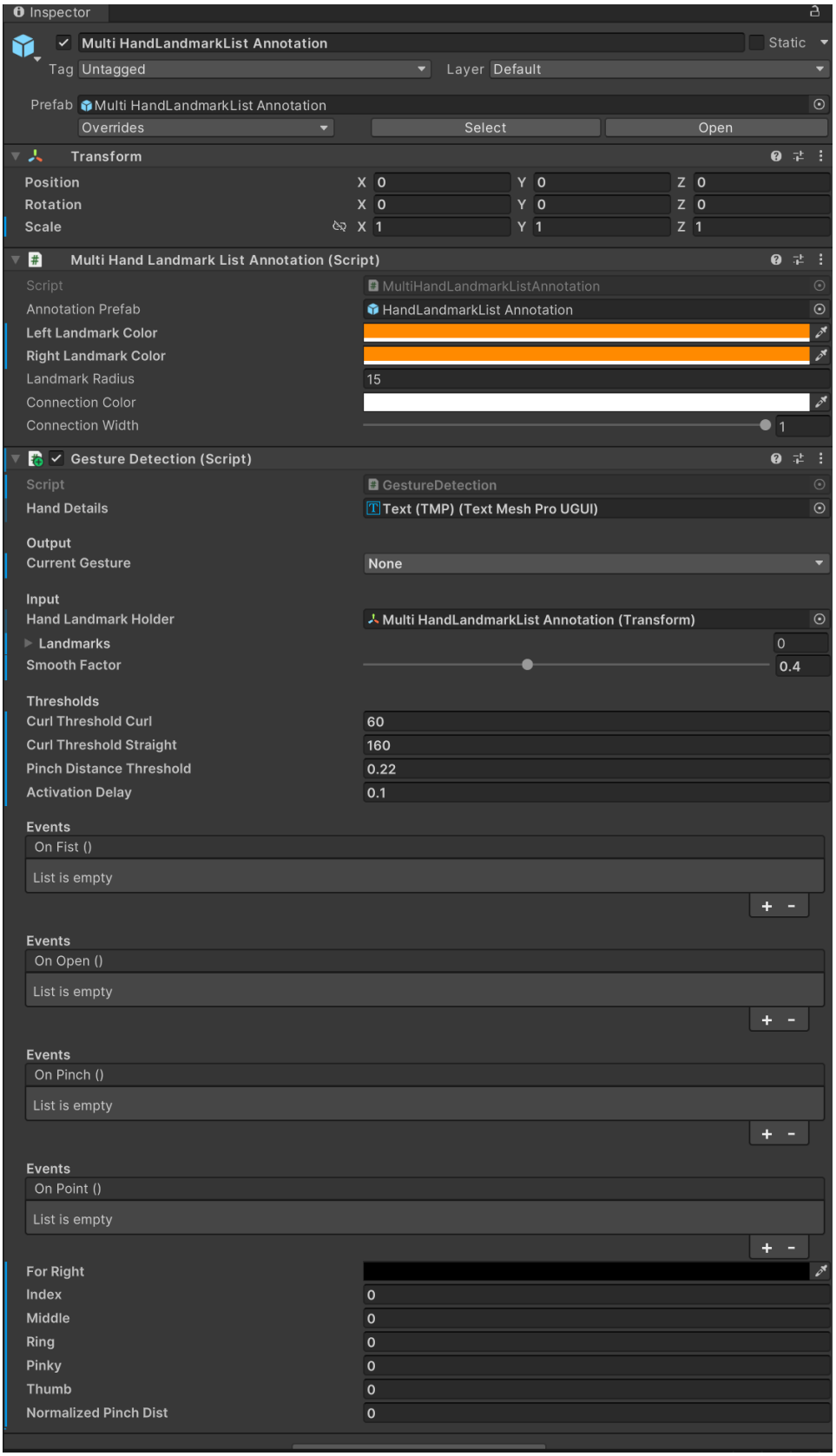
Η τελική μέθοδος **TriggerGestureEvent**() ενεργοποιεί το κατάλληλο Unity event όταν ανιχνεύεται αλλαγή χειρονομίας:

```
1 reference
void TriggerGestureEvent(HandGesture gesture)
{
    switch (gesture)
    {
        case HandGesture.Fist:
            OnFist?.Invoke();
            break;
        case HandGesture.Open:
            OnOpen?.Invoke();
            break;
        case HandGesture.Pinch:
            OnPinch?.Invoke();
            break;
        case HandGesture.Point:
            OnPoint?.Invoke();
            break;
    }
```

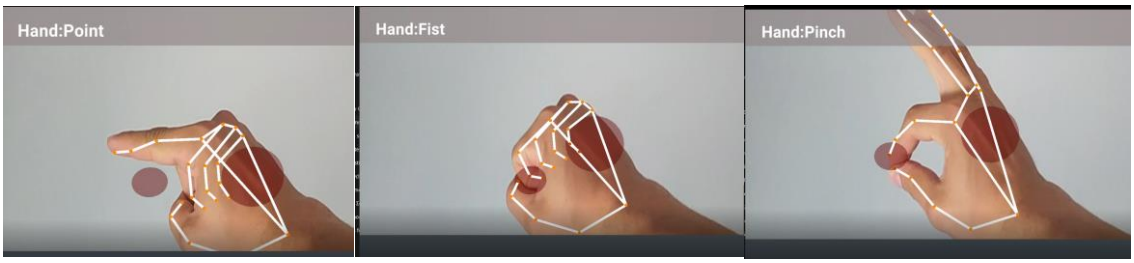
4.27 GestureDetection.cs 12

Συνολικά, η **GestureDetection** αποτελεί τη βάση όλου του συστήματος χειρονομιών.

1. Συλλέγει τα 21 σημεία (landmarks) του MediaPipe Hands.
2. Εξομαλύνει τις τοποθεσίες τους.
3. Υπολογίζει τις γωνίες των δακτύλων και την απόσταση δείκτη-αντίχειρα.
4. Ορίζει τις χειρονομίες: None/Open/Fist/Pinch/Point
5. Ενεργοποιεί τα αντίστοιχα Unity Event όταν αλλάζει η χειρονομία.



4.28 GestureDetection Inspector View



4.29 GestureDetection - Gestures

4.3.2.2 HandDepthTracker.cs

Η κλάση **HandDepthTracker.cs** είναι υπεύθυνη για τον υπολογισμό και την τοποθέτηση τριών βασικών σημείων αλληλεπίδρασης του χεριού στον χώρο:

- Το **fistObject** για χειρονομία **Fist**
- Το **pichObject** για χειρονομία **Pinch**
- Το **pointObject** για χειρονομία **Point**

Η τοποθέτηση αυτών των σημείων δεν γίνεται τυχαία, αλλά βασίζεται σε πραγματικά δεδομένα των landmarks που ανιχνεύονται από τη **GestureDetection**. Στόχος της κλάσης είναι να τοποθετήσει αυτά τα εικονικά αντικείμενα στον σωστό χώρο, ώστε να συγχρονίζονται με την προβολή του χεριού στην οθόνη, ανεξάρτητα από το βάθος.

Στην αρχή δηλώνονται οι βασικές μεταβλητές:

```
public class HandDepthTracker : MonoBehaviour
{
    6 references
    public Camera mainCamera;
    //public Transform farObject;    // Object that's far away
    2 references
    public Transform fistObject;   // Object to move closer but match screen position
    2 references
    public Transform pichObject;   // Object to move closer but match screen position
    2 references
    public Transform pointObject;   // Object to move closer but match screen position
    3 references
    public float nearZ = 1.5f;     // Distance from camera in world units for the near object
```

4.30 HandDepthTracker.cs 1

Η **mainCamera** χρησιμοποιείται για τις μετατροπές συντεταγμένων μεταξύ κόσμου και οθόνης. Τα **fistObject**, **pichObject** και **pointObject** είναι τα εικονικά σημεία που αντιστοιχούν στις χειρονομίες. Η μεταβλητή **nearZ** καθορίζει την απόσταση από την κάμερα σε μονάδες κόσμου στην οποία θα τοποθετούνται αυτά τα σημεία, διασφαλίζοντας ότι είναι πάντα "μπροστά" από το χέρι στην κάμερα.

Η βοηθητική μέθοδος **GetPositionBetween**() απλοποιεί τον υπολογισμό ενδιάμεσων σημείων μεταξύ δύο landmarks:

```
3 references
public Vector3 GetPositionBetween(Vector3 a, Vector3 b, float t)
{
    return Vector3.Lerp(a, b, t);
}
```

4.31 HandDepthTracker.cs 2

Η κύρια λειτουργία της κλάσης βρίσκεται στη μέθοδο **Update**():

```
0 references
void Update()
{
    if (GestureDetection.instance.landmarks.Count != 21) return;
```

4.32 HandDepthTracker 3

Το πρώτο πράγμα που γίνεται είναι να ελεγχθεί αν τα 21 landmarks του χεριού έχουν ανιχνευθεί σωστά από τη **GestureDetection**. Αν όχι, η μέθοδος τερματίζει αμέσως ώστε να αποφευχθούν σφάλματα.

Στη συνέχεια, υπολογίζονται τα ενδιάμεσα σημεία του χεριού που χρησιμοποιούνται για τοποθέτηση των τριών σημείων αλληλεπίδρασης:

```
possition_1 = GetPositionBetween(GestureDetection.instance.landmarks[9].position,
GestureDetection.instance.landmarks[13].position,.5f);
possition_2 = GetPositionBetween(possition_1,
GestureDetection.instance.landmarks[0].position, .4f);
possition_3 = GetPositionBetween(GestureDetection.instance.landmarks[4].position,
GestureDetection.instance.landmarks[8].position, .5f);
possition_4 = GestureDetection.instance.landmarks[8].position;
```

4.33 HandDepthTracker.cs 4

- Η **possition_1** είναι το σημείο ανάμεσα στις αρθρώσεις των δαχτύλων **μέσο** και **παράμεσο** (landmarks 9 και 13).
- Η **possition_2** είναι πιο κοντά στο κέντρο της παλάμης, καθώς παίρνει το σημείο μεταξύ της possition_1 και του landmark 0, που είναι η βάση της παλάμης. Αυτή η θέση χρησιμοποιείται για την κεντρική αναφορά του **fistObject**.
- Η **possition_3** είναι το σημείο ανάμεσα στο άκρο του αντίχειρα (landmark 4) και του δείκτη (landmark 8), κατάλληλο για το **pinchObject**.
- Η **possition_4** παίρνει την πραγματική θέση του άκρου του δείκτη, που χρησιμοποιείται για το **pointObject**.

Ακολουθεί η μετατροπή των θέσεων αυτών σε συντεταγμένες οθόνης:

```
// Step 1: Get screen position of the far object
Vector3 farScreenPos = mainCamera.WorldToScreenPoint(possition_2);
Vector3 farScreenPos_pich = mainCamera.WorldToScreenPoint(possition_3);
Vector3 farScreenPos_point = mainCamera.WorldToScreenPoint(possition_4);
// Step 2: Use same screen position, but with a closer Z
Vector3 nearScreenPos = new Vector3(farScreenPos.x, farScreenPos.y, nearZ);
Vector3 nearScreenPos_pinch = new Vector3(farScreenPos_pich.x, farScreenPos_pich.y, nearZ);
Vector3 nearScreenPos_point = new Vector3(farScreenPos_point.x, farScreenPos_point.y, nearZ);


// Step 3: Convert back to world position and apply
Vector3 newNearWorldPos = mainCamera.ScreenToWorldPoint(nearScreenPos);
Vector3 newNearWorldPos_pinch = mainCamera.ScreenToWorldPoint(nearScreenPos_pinch);
Vector3 newNearWorldPos_pooint = mainCamera.ScreenToWorldPoint(nearScreenPos_point);
```

4.34 HandDepthTracker.cs 5

Εδώ, οι θέσεις των landmarks μετατρέπονται σε pixel coordinates της οθόνης, λαμβάνοντας υπόψη τη θέση της κάμερας. Έπειτα, ορίζονται νέες συντεταγμένες με το ίδιο **x** και **y** αλλά διαφορετικό **z**, χρησιμοποιώντας το **nearZ**.

Αυτό αναγκάζει τα αντικείμενα να "προβάλλονται" πάντα σε συγκεκριμένη απόσταση από την κάμερα, ώστε να είναι πιο σταθερά και προβλέψιμα. Στη συνέχεια γίνεται η αντίστροφη μετατροπή πίσω σε world coordinates.

Τέλος, εφαρμόζεται κίνηση με **Lerp** στα αντικείμενα ώστε να κινούνται ομαλά, χωρίς απότομες αλλαγές θέσης:

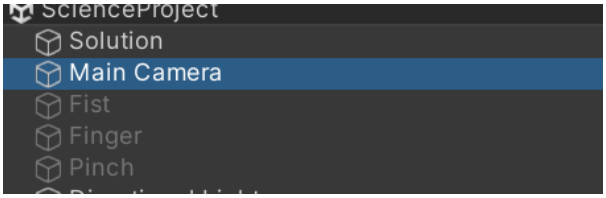
```
fistObject.position = Vector3.Lerp(fistObject.position, newNearWorldPos, Time.deltaTime * 10);
pichObject.position = Vector3.Lerp(pichObject.position, newNearWorldPos_pinch, Time.deltaTime * 10);
pointObject.position = Vector3.Lerp(pointObject.position, newNearWorldPos_pooint, Time.deltaTime * 10);
```

4.35 HandDepthTracker.cs 6

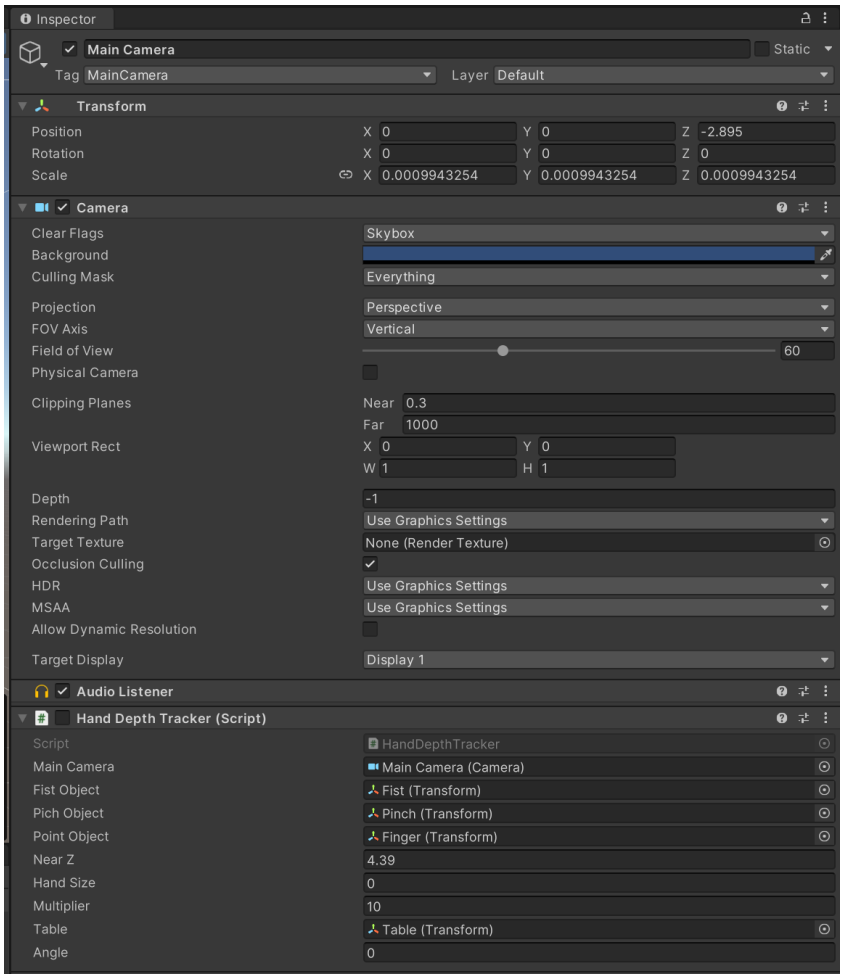
Συνολικά, η **HandDepthTracker** λειτουργεί ως σύνδεσμος μεταξύ των δεδομένων της **GestureDetection** και της τοποθέτησης αντικειμένων στον χώρο. Υπολογίζει σταθερά, αξιόπιστα σημεία που ακολουθούν το χέρι, αλλά τα τοποθετεί σε σταθερή απόσταση από την κάμερα ώστε να είναι πιο εύκολος ο έλεγχος και η αλληλεπίδραση με αντικείμενα. Στην πράξη, επιτρέπει στον χρήστη να κάνει Pinch, Point ή Fist με φυσική κίνηση που αποδίδεται ομαλά στο περιβάλλον του παιχνιδιού.

Πιο απλά:

1. Παίρνει τα landmark του Mediapipe από τη GestureDetection.
2. Υπολογίζει 3 σημεία αλληλεπίδρασης που ακολουθούν το χέρι, και τα μετατρέπει σε συντεταγμένες οθόνης.
3. Μετατρέπει αυτά τα σημεία σε συντεταγμένες κόσμου με σταθερή απόσταση από την κάμερα.
4. Μετακινεί ομαλά τα σημεία (Transforms) στις νέες συντεταγμένες κόσμου.



4.36 HandDepthTracker Inspector 1



4.37 HandDepthTracker Inspector 2

4.4 Εικονικό Χέρι και Αλληλεπίδραση

4.4.1 Interactable.cs

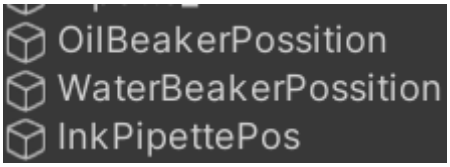
Η κλάση **Interactable.cs** είναι υπεύθυνη για τη συμπεριφορά των αντικειμένων που μπορούν να αλληλεπιδράσουν με το εικονικό χέρι του χρήστη. Όταν ο χρήστης «πιάνει» ένα αντικείμενο μέσω χειρονομιών, αυτή η κλάση το κάνει να ακολουθεί ομαλά το χέρι. Όταν το αφήνει, το αντικείμενο είτε μένει στη θέση που το άφησε είτε επιστρέφει στην αρχική του θέση με ελεγχόμενη κίνηση.

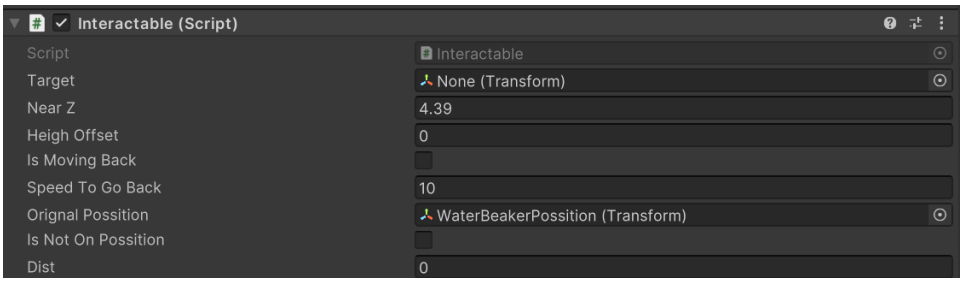
Στην αρχή δηλώνονται οι μεταβλητές που καθορίζουν αυτή τη συμπεριφορά:

```
public class Interactable : MonoBehaviour
{
    2 references
    public Transform Target;
    1 reference
    public float nearZ = 1.5f;
    1 reference
    Rigidbody myRigidbody;
    3 references
    BoxCollider myCollider;
    1 reference
    public float heighOffset;
    2 references
    public bool isMovingBack;
    1 reference
    public float speedToGoBack=10;
    3 references
    public Transform orignalPossition;
    0 references
    public bool isNotOnPossition;
```

4.38 Interactable.cs 1

Η μεταβλητή **Target** ορίζει το σημείο που πρέπει να ακολουθεί το αντικείμενο όταν ο χρήστης το πιάνει. Αυτό δεν ορίζεται χειροκίνητα στον Inspector, αλλά ενημερώνεται από την **VirtualHand (**θα αναλυθεί στη συνέχεια**)** κατά την εκτέλεση του παιχνιδιού, ανάλογα με τη χειρονομία που εντοπίστηκε. Αντίθετα, η **orignalPossition** είναι ένα Transform που ορίζεται στον Inspector. Για κάθε αντικείμενο που χρησιμοποιεί η κλάση, τοποθετούμε στη σκηνή ένα **empty GameObject** στη θέση όπου πρέπει να επιστρέφει το αντικείμενο και το συνδέουμε στο συγκεκριμένο πεδίο. Οι **nearZ** και **heighOffset** χρησιμοποιούνται για τη σωστή τοποθέτηση του αντικειμένου στον χώρο και ρυθμίζονται από τον Inspector, ενώ η **speedToGoBack** καθορίζει πόσο γρήγορα θα επιστρέψει το αντικείμενο στην αρχική του θέση όταν αφήνεται.





4.39 Interactable.cs Inspector View

Η μέθοδος Start() προετοιμάζει την κλάση, παίρνοντας αναφορές στο Rigidbody και στο BoxCollider του αντικειμένου:

```
0 references
private void Start()
{
    myRigidbody = GetComponent<Rigidbody>();
    myCollider = GetComponent<BoxCollider>();
}
```

4.40 Interactable.cs 2

Η κύρια λειτουργία βρίσκεται στη μέθοδο Update():

```
private void Update()
{
    if (Target != null)
    {
        myCollider.enabled = false;
        Vector3 farScreenPos = Camera.main.WorldToScreenPoint(Target.position);
        Vector3 nearScreenPos = new Vector3(farScreenPos.x, farScreenPos.y, nearZ);
        Vector3 newNearWorldPos = Camera.main.ScreenToWorldPoint(nearScreenPos);
        transform.position = Vector3.Lerp(transform.position,
        newNearWorldPos + new Vector3(0, heighOffset, 0), Time.deltaTime * 10);
    }
    else
    {
        if(isMovingBack)
        StartCoroutine(MoveBack());
    }
```

4.41 Interactable.cs 3

Αν υπάρχει **Target**, δηλαδή το αντικείμενο έχει "πιαστεί" από το εικονικό χέρι, το script απενεργοποιεί το BoxCollider για να αποφύγει συγκρούσεις και υπολογίζει τη σωστή θέση του αντικειμένου στην οθόνη χρησιμοποιώντας τις μεθόδους **WorldToScreenPoint** και **ScreenToWorldPoint**. Έτσι το αντικείμενο ακολουθεί το Target στο σωστό βάθος nearZ. Η τελική τοποθέτηση γίνεται ομαλά μέσω της Vector3.Lerp, ώστε η κίνηση να είναι φυσική και όχι απότομη. Όταν το αντικείμενο δεν έχει πλέον Target, ελέγχεται αν το **isMovingBack** είναι ενεργό, οπότε ξεκινά η διαδικασία επαναφοράς.

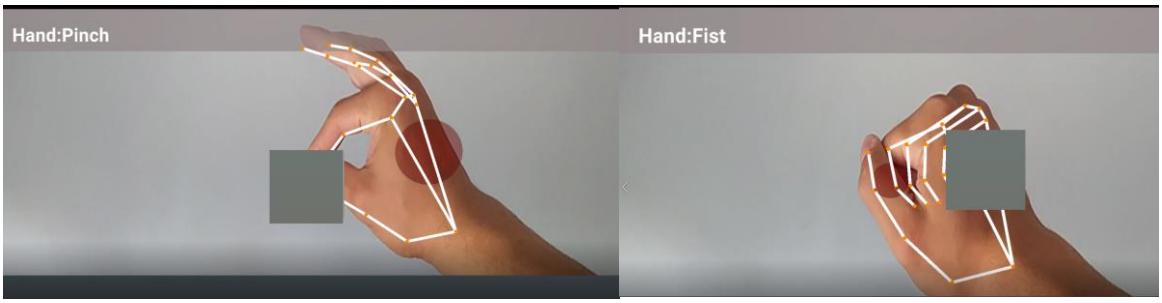
Η επαναφορά στην αρχική θέση υλοποιείται από το coroutine **MoveBack**():

```
IEnumerator MoveBack()
{
    myCollider.enabled = true;
    dist = Vector3.Distance(transform.position, orignalPossition.position);
    while (dist > 0.02f)
    {
        dist = Vector3.Distance(transform.position, orignalPossition.position);
        transform.position = Vector3.Lerp(
            transform.position, orignalPossition.position, Time.deltaTime * speedToGoBack);

        yield return null;
    }
    isMovingBack = false;
    ChemistryEquipmentManager.instance.holdingALiquid = false;
    yield return null;
}
```

4.42 Interactable.cs 4

Αρχικά επανενεργοποιείται ο BoxCollider ώστε να αποκατασταθεί η σωστή συμπεριφορά φυσικής. Υπολογίζεται η απόσταση από την τρέχουσα θέση του αντικειμένου ως την καθορισμένη **orignalPossition** και, όσο αυτή είναι μεγαλύτερη από **0.02**, το αντικείμενο κινείται ομαλά προς το σημείο αυτό. Μόλις φτάσει στην αρχική του θέση, η σημαία **isMovingBack** μηδενίζεται και ενημερώνεται η **ChemistryEquipmentManager (**θα αναλυθεί στη συνέχεια**)** ότι δεν κρατάμε πλέον υγρό.

Η **Interactable** επιτρέπει στον χρήστη να πιάνει, να μετακινεί και να αφήνει αντικείμενα με φυσικό τρόπο.



4.43 Interactable.cs Object Interaction

4.4.2 VirtualHand.cs

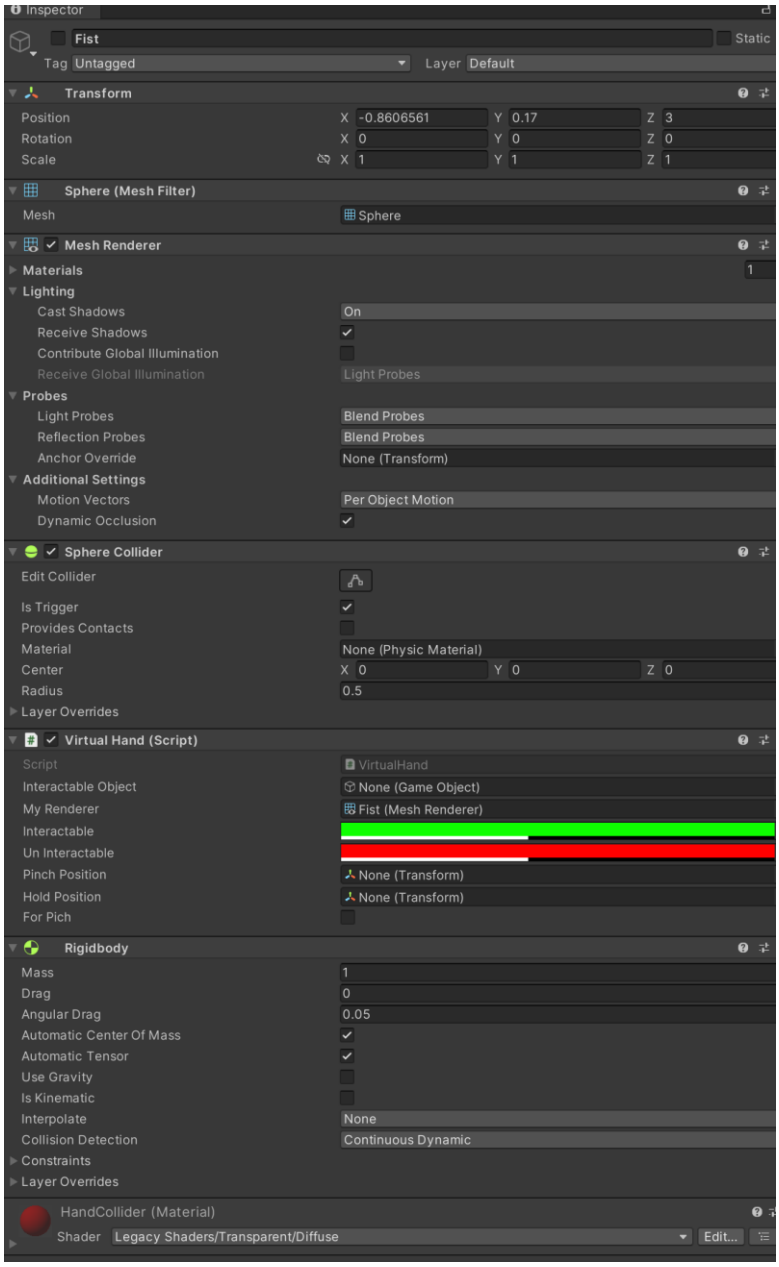
Η κλάση **VirtualHand.cs** είναι το κεντρικό σημείο σύνδεσης ανάμεσα στη **GestureDetection**, που ανιχνεύει χειρονομίες μέσω MediaPipe, και την **Interactable**, που χειρίζεται την κίνηση των αντικειμένων. Ο ρόλος της είναι να ελέγχει πότε το εικονικό χέρι μπορεί να αλληλεπιδράσει με αντικείμενα, να τα «κολλάει» στο κατάλληλο σημείο του χεριού, και να τα αφήνει όταν η χειρονομία αλλάζει.

Στην αρχή έχουμε τις δηλώσεις μεταβλητών:

```
public class VirtualHand : MonoBehaviour
{
    17 references
    public GameObject interactableObject;
    3 references
    public Renderer myRenderer;
    1 reference | 2 references
    public Color interactable, UnInteractable;
    2 references
    public Transform pinchPosition;
    2 references
    public Transform holdPosition;

    2 references
    public bool forPich;
```

4.44 VirtualHand.cs 1

Η μεταβλητή **interactableObject** αναφέρεται στο αντικείμενο που μπορεί να πιαστεί από το χέρι, και αρχικά είναι null. Το **myRenderer** συνδέεται από τον Inspector και είναι ο Renderer του εικονικού χεριού. Οι **interactable** και **UnInteractable** είναι δύο χρώματα που ορίζονται επίσης από τον Inspector και χρησιμοποιούνται για να αλλάζει το χρώμα του χεριού, δίνοντας οπτικό feedback στο χρήστη όταν βρίσκεται πάνω από ένα πιάσιμο-διαθέσιμο αντικείμενο. Οι **pinchPosition** και **holdPosition** είναι Transform θέσεις πάνω στο χέρι — η πρώτη αντιστοιχεί στο σημείο επαφής του δείκτη με τον αντίχειρα (pinch), ενώ η δεύτερη είναι το κέντρο της παλάμης. Το **forPich** είναι μια boolean παράμετρος, που ρυθμίζεται στον Inspector, για να καθορίσει αν το συγκεκριμένο VirtualHand χρησιμοποιείται για pinch gestures ή όχι (π.χ. όταν υπάρχουν δύο χέρια στη σκηνή).



4.45 Interactable Objects - Inspector

Η μέθοδος Start() υλοποιείται ως coroutine γιατί πρέπει να περιμένει να φορτωθούν τα landmarks από τη **GestureDetection** πριν ρυθμιστούν οι θέσεις:

```
private IEnumerator Start()
{
    while (GestureDetection.instance.landmarks.Count != 21)
    {
        yield return null;
    }
    pinchPosition = GestureDetection.instance.landmarks[8];
    holdPosition = transform;
}
```

4.46 VirtualHand.cs 2

Η συνθήκη $\boldsymbol{landmarks.Count} != \mathbf{21}$ περιμένει μέχρι το MediaPipe να έχει εντοπίσει και φορτώσει τα 21 σημεία του χεριού. Όταν είναι διαθέσιμα, η **pinchPosition** ορίζεται στο landmark **8** (άκρο του δείκτη) και η **holdPosition** γίνεται η θέση του ίδιου του VirtualHand αντικειμένου.

Η μέθοδος **Update**() περιέχει την κύρια λογική αλληλεπίδρασης:

```
private void Update()
{
    if (interactableObject != null)
    {
        if (GestureDetection.instance.currentGesture == GestureDetection.HandGesture.Fist
        || GestureDetection.instance.currentGesture == GestureDetection.HandGesture.Pinch)
        {
            if (GestureDetection.instance.currentGesture == GestureDetection.HandGesture.Fist && ! forPich)
            {
                if (!ChemistryEquipmentManager.instance.holdingALiquid)
                {
                    ChemistryEquipmentManager.instance.holdingALiquid= true;
                    interactableObject.GetComponent<Interactable>().Target = holdPosition;
                }


            }
            else if (GestureDetection.instance.currentGesture == GestureDetection.HandGesture.Pinch
            && forPich)
            {
                if (!ChemistryEquipmentManager.instance.holdingALiquid)
                {
                    ChemistryEquipmentManager.instance.holdingALiquid = true;
                    interactableObject.GetComponent<Interactable>().Target = pinchPosition;
                }

            }
            else
            {
                if (interactableObject != null)
                {
                    interactableObject.GetComponent<Interactable>().isMovingBack = true;
                   interactableObject.GetComponent<Interactable>().Target = null;
                    interactableObject = null;
                }
            }
        }
        else
        {
            if (interactableObject != null)
            {
                interactableObject.GetComponent<Interactable>().isMovingBack = true;
                interactableObject.GetComponent<Interactable>().Target = null;
                interactableObject = null;
            }
        }
    }
}
```

4.47 VirtualHand.cs 3

Αν το **interactableObject** δεν είναι null, σημαίνει ότι το χέρι βρίσκεται μέσα σε κάποιο αντικείμενο που μπορεί να αλληλεπιδράσει. Ελέγχουμε τότε ποια χειρονομία έχει εντοπιστεί.

Αν η χειρονομία είναι **Fist** και το συγκεκριμένο **VirtualHand** έχει ρυθμιστεί ώστε να μην είναι για pinch (forPich == false), τότε το αντικείμενο πιάνεται και τοποθετείται στη θέση της **holdPosition**, δηλαδή στο κέντρο της παλάμης.

Αντίστοιχα, αν η χειρονομία είναι **Pinch** και το **VirtualHand** έχει ρυθμιστεί ως pinch-hand (forPich == true), το αντικείμενο κολλάει στη **pinchPosition**, δηλαδή στη θέση επαφής δείκτη και αντίχειρα.

Σε κάθε περίπτωση, πριν οριστεί το Target, ελέγχουμε αν το

ChemistryEquipmentManager.instance.holdingALiquid είναι false, ώστε να διασφαλίσουμε ότι δεν θα κρατάμε ταυτόχρονα δύο αντικείμενα που περιέχουν υγρά.

Αν η χειρονομία αλλάξει σε λάθος τύπο ή αν το χέρι απλά δεν κάνει κάποια αναγνωρισμένη χειρονομία, το **interactableObject** αφήνεται ελεύθερο: το **Target** του μηδενίζεται, το flag **isMovingBack** γίνεται true ώστε να επιστρέψει στην αρχική θέση, και η αναφορά διαγράφεται.

Η μέθοδος **OnTriggerStay()** είναι υπεύθυνη για τον εντοπισμό αντικειμένων που μπορούν να αλληλεπιδράσουν με το χέρι:

```
public void OnTriggerStay(Collider other)
{
    if (other.transform.CompareTag("Interactable") && interactableObject==null)
    {
        myRenderer.material.color = interactable;

        if (GestureDetection.instance.currentGesture == GestureDetection.HandGesture.Fist
        || GestureDetection.instance.currentGesture == GestureDetection.HandGesture.Pinch)
        {
            interactableObject = other.gameObject;
        }
        else
        {
            if (interactableObject != null)
            {
                interactableObject.GetComponent<Interactable>().isMovingBack = true;
                interactableObject.GetComponent<Interactable>().Target = null;
                interactableObject = null;
            }

        }
    }
    else
    {
        myRenderer.material.color = UnInteractable;
    }
}
```

4.48 VirtualHand.cs 4

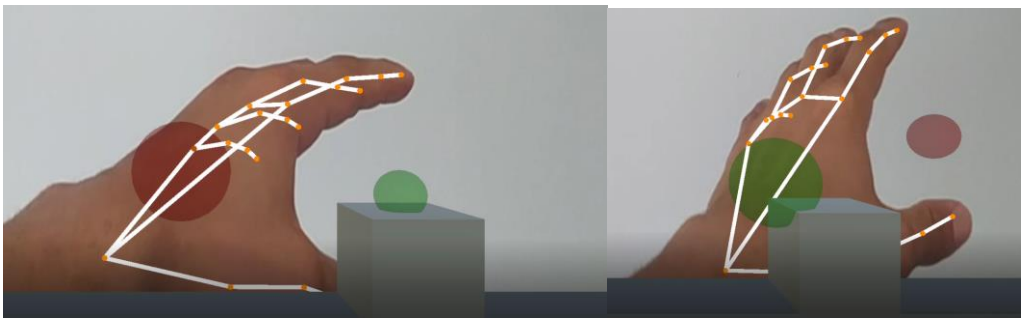
Όταν το **Collider** του χεριού βρίσκεται μέσα σε κάποιο αντικείμενο με tag **Interactable** και δεν κρατάμε ήδη κάποιο άλλο αντικείμενο, αλλάζουμε το χρώμα του χεριού στο **interactable** μέσω του Inspector. Αν η **GestureDetection** δείξει ότι το χέρι κάνει Fist ή Pinch, αποθηκεύουμε το αντικείμενο ως **interactableObject** και το αφήνουμε διαθέσιμο για πιάσιμο. Αν όχι, το αντικείμενο ελευθερώνεται.

Τέλος, η **OnTriggerExit**() φροντίζει να επαναφέρει το χρώμα του χεριού όταν το **Collider** βγει από το αντικείμενο:

```
private void OnTriggerExit(Collider other)
{
    if (other.transform.CompareTag("Interactable"))
    {
        myRenderer.material.color = UnInteractable;
    }

}
```

4.49 VirtualHand.cs 5

Έτσι, ο χρήστης βλέπει ξεκάθαρα πότε το χέρι είναι πάνω σε αντικείμενο και πότε όχι.



4.50 Interactable Color

4.5 Ρύθμιση Κάμερας και Διαχείριση Αδειών

Για τη σωστή λειτουργία της εφαρμογής, είναι απαραίτητο να διασφαλιστεί ότι η κάμερα προσαρμόζεται σωστά σε διαφορετικές αναλύσεις οθόνης και ότι η συσκευή έχει δώσει άδεια πρόσβασης στην κάμερα. Αυτό υλοποιείται με δύο κλάσεις: **CameraFitHorizontal.cs**, **CameraPemissionRequester.cs.**

4.5.1 CameraFitHorizontal.cs

Αυτή η κλάση εξασφαλίζει ότι το **οπτικό πεδίο της κάμερας (Field of View – FOV)** παραμένει σωστό και σταθερό, ανεξάρτητα από την ανάλυση και το aspect ratio της οθόνης. Αυτό είναι ιδιαίτερα κρίσιμο στο πλαίσιο της εφαρμογής μας, επειδή το παιχνίδι προβάλλει αντικείμενα επαυξημένης πραγματικότητας (AR), και οι μαθητές πρέπει να βλέπουν τις σκηνές από σωστή οπτική γωνία.

```
public class CameraFitHorizontal : MonoBehaviour
{
    1 reference
    public float referenceAspect = 16f / 9f;
    1 reference
    public float referenceVerticalFOV = 60f;
    [SerializeField]
    2 references
    Camera cam;
    [SerializeField]
    3 references | 3 references
    int w, h;

    0 references
    void Awake()
    {
        Screen.sleepTimeout = SleepTimeout.NeverSleep;
        cam = GetComponent<Camera>();
        w = Screen.width; h = Screen.height;
        Apply();
        InvokeRepeating(nameof(CheckSize), 0.25f, 0.25f); // updates on rotate/res change in build
    }
    1 reference
    void CheckSize()
    {
        if (w != Screen.width || h != Screen.height) { w = Screen.width; h = Screen.height; Apply(); }
    }


    2 references
    void Apply()
    {
        float aspect = (float)Screen.width / Mathf.Max(1, Screen.height);
        // Keep horizontal FOV constant → scale vertical FOV with (referenceAspect / aspect)
        float refV = referenceVerticalFOV * Mathf.Deg2Rad;
        float newV = 2f * Mathf.Atan(Mathf.Tan(refV * 0.5f) * (referenceAspect / aspect));
        cam.fieldOfView = Mathf.Clamp(newV * Mathf.Rad2Deg, 10f, 100f);
    }
}
```

4.51 CameraFitHorizontal.cs

Η μεταβλητή **referenceAspect** καθορίζει το aspect ratio αναφοράς (16:9) και η **referenceVerticalFOV** ορίζει το βασικό κατακόρυφο πεδίο όρασης. Στη μέθοδο **Awake**(), η συσκευή ορίζεται ώστε να μην «κοιμάται» η οθόνη (Screen.sleepTimeout), λαμβάνεται η κάμερα, αποθηκεύονται οι τρέχουσες διαστάσεις οθόνης, εφαρμόζονται οι αρχικές ρυθμίσεις μέσω της **Apply**(), και ενεργοποιείται η **InvokeRepeating**() ώστε να γίνεται συνεχής έλεγχος.

Η μέθοδος **CheckSize**() ελέγχει αν άλλαξε το μέγεθος ή η περιστροφή της οθόνης. Αν ναι, καλείται ξανά η **Apply**().

Η **Apply**() είναι η βασική μέθοδος: υπολογίζει το νέο κάθετο FOV ώστε να διατηρείται **σταθερό οριζόντιο FOV**, ακόμα και όταν αλλάζει το aspect ratio της οθόνης. Το αποτέλεσμα εφαρμόζεται στο **cam**.**fieldOfView** και περιορίζεται μεταξύ **10°** και **100°** για λόγους σταθερότητας.

4.5.2 CameraPermissionRequester.cs

Η δεύτερη κλάση, **CameraPermissionRequester.cs**, χειρίζεται την αίτηση άδειας πρόσβασης στην κάμερα, κάτι που είναι απαραίτητο για το **MediaPipe Hand Tracking**.

```
public class CameraPermissionRequester : MonoBehaviour
{
    0 references
    IEnumerator Start()
    {
#if UNITY_ANDROID && !UNITY_EDITOR
        if (!Application.HasUserAuthorization(UserAuthorization.WebCam))
        {
            yield return Application.RequestUserAuthorization(UserAuthorization.WebCam);
        }

#else
        yield return null;
#endif
    }

}
```

4.52 CameraPermissionRequester.cs

Η κλάση εκτελείται στην εκκίνηση **Start().** Αρχικά ελέγχει αν η εφαρμογή έχει ήδη άδεια χρήσης της κάμερας μέσω του

Application.HasUserAuthorization. Αν όχι, ζητάει άδεια με την **RequestUserAuthorization()**.

Με αυτές τις δύο κλάσεις, διασφαλίζεται ότι:

1. Η **κάμερα** προσαρμόζεται σωστά στις διαφορετικές Android συσκευές.
2. Η **άδεια** για χρήση της κάμερας ζητείται με τον σωστό τρόπο ώστε το MediaPipe να έχει πρόσβαση στο live βίντεο.

4.6 Χημικός Εξοπλισμός και Υγρά

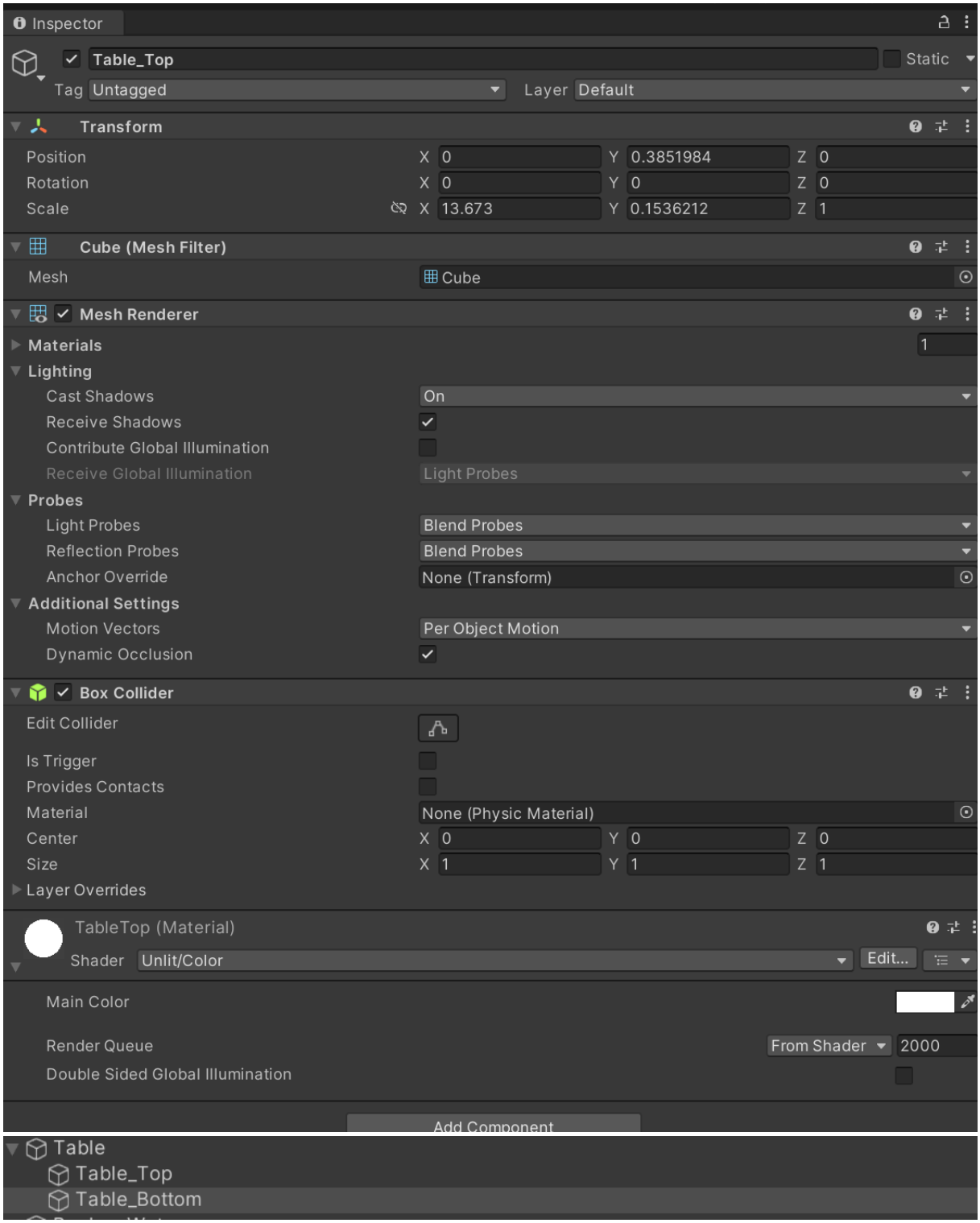
4.6.1 Το Τραπέζι Εργαστηρίου (Laboratory Table)

Το τραπέζι είναι το βασικό στατικό αντικείμενο πάνω στο οποίο τοποθετούνται όλα τα εργαστηριακά σκεύη. Στη σκηνή αποτελείται από δύο ξεχωριστά τμήματα:

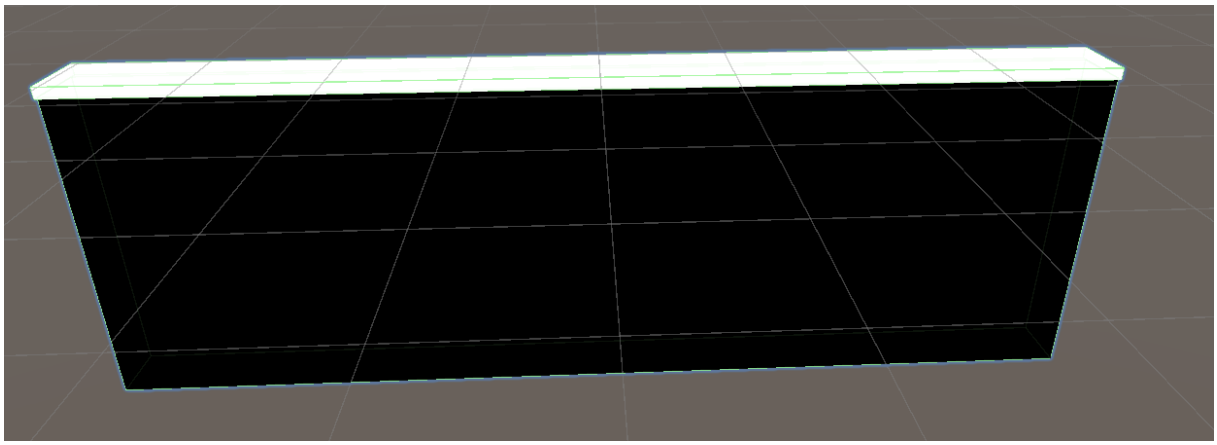
- **Table_Top** → Η επάνω επιφάνεια, όπου τοποθετούνται τα ποτήρια ζέσεως και τα υπόλοιπα αντικείμενα. Έχει λευκό **Unlit/Color** υλικό, ώστε να υπάρχει οπτική αντίθεση με τα υπόλοιπα στοιχεία.

- **Table_Bottom** → Η βάση του τραπεζιού, η οποία παρέχει τη στήριξη στην επάνω επιφάνεια. Έχει μαύρο **Unlit/Color** υλικό ώστε να ξεχωρίζει από το υπόλοιπο περιβάλλον.

Και τα δύο μέρη χρησιμοποιούν **Cube Meshes** και **Box Colliders** ώστε να υποστηρίζονται οι φυσικές συγκρούσεις με τα αντικείμενα.



4.53 Τραπέζι Εργαστηρίου - Unity Inspector

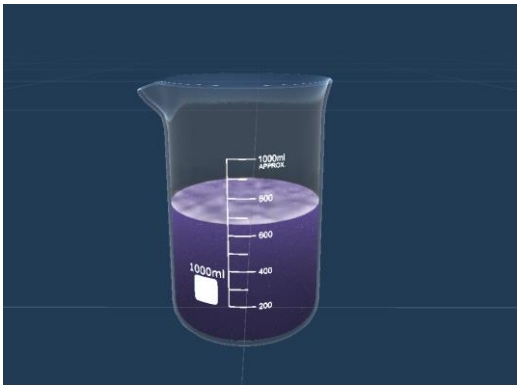


4.54 Table Unity Scene View

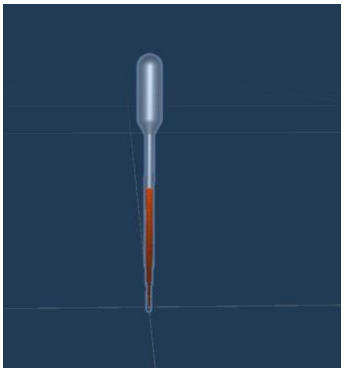
4.6.2 Liquid Volume Pro Prefabs

Για την προσομοίωση των υγρών που χρησιμοποιούνται στο πείραμα, επιλέχθηκε το εργαλείο **Liquid Volume Pro.** Η επιλογή του βασίστηκε στην υψηλή απόδοση, τα ρεαλιστικά οπτικά εφέ και τη δυνατότητα προσαρμογής ιδιοτήτων όπως το **χρώμα**, η **στάθμη του υγρού**, η **διαφάνεια**, η **ανάδευση** και η **μίξη**.

Στη σκηνή, ολόκληρη η χημική διάταξη έχει υλοποιηθεί με τη χρήση των έτοιμων **prefabs Beaker_1000ml** και **Pippete** από το **Liquid Volume Pro**.



4.55 Beaker_1000ml.prefab - Liquid Volume Pro



4.56 Pipette.prefab - Liquid Volume Pro

Τα κύρια αντικείμενα που χρησιμοποιούνται είναι:

- **Beaker_Water**
- **Beaker_Oil**
- **Beaker_Empty**
- **Pipette_Ink**

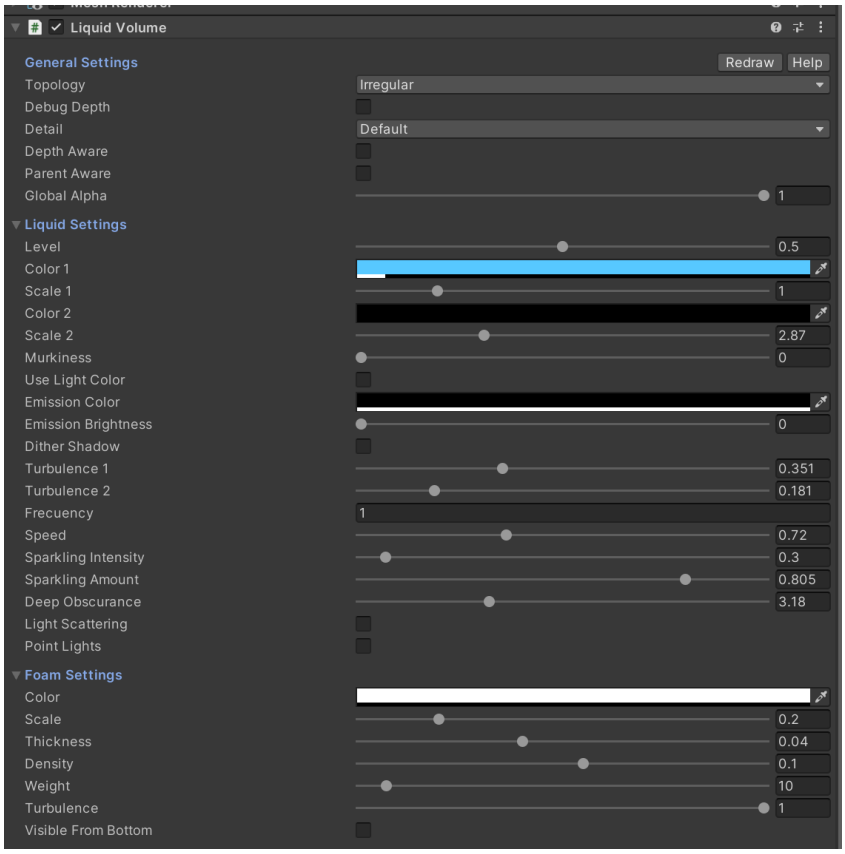
Κάθε prefab περιέχει συγκεκριμένα components:

- Ένα **Beaker Mesh** για την οπτική αναπαράσταση.
- Ένα component **LiquidVolume** του Liquid Volume Pro, που διαχειρίζεται το υγρό.
- Ένα **Label Mesh** για τις ετικέτες.
- Ένα **PouringPoint** τοποθετημένο στο στόμιο, για να καθορίζει το σημείο από όπου "ρέει" το υγρό.
- Κλάσεις, όπως η **BeakerPourer** και **CheckingIfPouringLiquidinBeaker**, οι οποίες ελέγχουν την προσομοίωση της ροής του υγρού (θα αναλυθούν στη συνέχεια).

4.57 Χημικός Εξοπλισμός - Unity Hierarchy View

4.6.3 Liquid Volume Inspector Settings

Το **Liquid Volume Pro** χρησιμοποιείται για να αποδοθεί ρεαλιστική εμφάνιση στα υγρά, με εφέ όπως ανατάραξη, θολότητα, φυσαλίδες και αντανακλάσεις. Παρακάτω εξηγούνται τα πιο σημαντικά πεδία που ρυθμίστηκαν:



4.58 Liquid Volume Inspector Settings

Γενικές Ρυθμίσεις

- **Topology**: Ορίζεται σε **Irregular**, ώστε η επιφάνεια του υγρού να είναι φυσική και όχι τέλεια επίπεδη.
- **Detail**: **Default**, για ισορροπία μεταξύ ποιότητας και επιδόσεων.
- **Global Alpha**: Ρυθμίζει τη διαφάνεια του υγρού.

Ρυθμίσεις Υγρού (Liquid Settings)

- **Level**: Ορίζει τη στάθμη του υγρού μέσα στο ποτήρι.
- **Color1 & Color2**: Τα χρώματα του υγρού. Στην περίπτωσή μας, ρυθμίστηκαν ώστε να αναπαριστούν το νερό, το λάδι ή το μελάνι.
- **Murkiness**: Ρυθμίζει τη θολότητα. Χρησιμοποιείται π.χ. για το μελάνι, ώστε να φαίνεται πιο αδιαφανές.
- **Emission Color & Brightness**: Προσθέτει φωτεινότητα ή λάμψη στο υγρό αν χρειάζεται.
- **Turbulence1 & Turbulence2**: Ορίζουν πόσο "ταράζεται" η επιφάνεια του υγρού όταν κινείται το ποτήρι.
- **Sparkling Intensity & Amount**: Προσθέτει αντανακλάσεις για πιο ρεαλιστικό αποτέλεσμα.

Ρυθμίσεις Αφρού (Foam Settings)

- **Color**: Το χρώμα του αφρού.

- **Scale, Thickness, Density**: Ελέγχουν το μέγεθος, τη ρευστότητα και την πυκνότητα του αφρού.
- **Turbulence**: Επηρεάζει την κατανομή του αφρού ανάλογα με την κίνηση.

Οι συγκεκριμένες τιμές έχουν οριστεί ξεχωριστά για το **νερό**, το **λάδι** και το **μελάνι**, ώστε το οπτικό αποτέλεσμα να είναι όσο το δυνατόν πιο κοντά στα πραγματικά υγρά του πειράματος.

4.6.4 ChemistryEquipmentManager.cs

Η κλάση **ChemistryEquipmentManager** είναι η **κεντρική κλάση** που ελέγχει όλα τα στοιχεία του εξοπλισμού και συνεργάζεται με το Liquid Volume Pro για να διαχειριστεί τα επίπεδα των υγρών, τις μίξεις, το άδειασμα και τις αρχικές θέσεις των αντικειμένων.-

Ορισμός της Κλάσης και Singleton

```
1 reference
public class ChemistryEquipmentManager : MonoBehaviour
{
    2 references
    public static ChemistryEquipmentManager instance;
```

4.59 ChemistryEquipmentManager.cs 1

Η κλάση ακολουθεί το **Singleton pattern**, ώστε να υπάρχει μόνο ένα ενεργό instance σε όλη τη σκηνή. Αυτό επιτρέπει σε άλλες κλάσεις να έχουν εύκολη πρόσβαση στις λειτουργίες του manager.

Δήλωση Δεδομένων Εξοπλισμού

```
6 references
public sEquipments[] equipments;

0 references
public bool holdingALiquid;

0 references
public LiquidVolume emptyBeakerLiquidVolume;

1 reference
public EmptyBeaker emptyBeaker;
```

4.60 ChemistryEquipmentManager.cs 2

- **equipments**: Ένα array με όλα τα διαθέσιμα σκεύη. Κάθε στοιχείο είναι τύπου sEquipments (δομή ορισμένη παρακάτω).
- **holdingALiquid**: Flag που χρησιμοποιείται για να ελέγξει αν ο χρήστης «κρατά» υγρό.
- **emptyBeakerLiquidVolume**: Reference στο LiquidVolume του άδειου ποτηριού.
- **emptyBeaker**: Σύνδεση με την κλάση EmptyBeaker (θα αναλυθεί παρακάτω), για να διαχειρίζεται τις μίξεις και τα χρώματα.

Δομή sEquipments

```
public struct sEquipments
{
    0 references
    public eLiquidType type;
    0 references
    public Transform equipment;
    0 references
    public Transform orignalPosOfEquipment;
    6 references
    public LiquidVolume liquidVolume;

}
```

4.61 ChemistryEquipmentManager.cs 3

Κάθε αντικείμενο του εργαστηρίου έχει:

- **type**: Τύπος υγρού (WATER, OIL, INK, EMPTY).
- **equipment**: Αναφορά στο ίδιο το prefab.
- **orignalPosOfEquipment**: Χρησιμοποιείται από το **Interactable.cs** για να επιστρέφει το ποτήρι στην αρχική του θέση.
- **liquidVolume**: Το component του Liquid Volume Pro, ώστε να μπορεί να ενημερωθεί δυναμικά.

Αρχικοποίηση Singleton

```
private void Awake()
{
    if (instance == null) instance = this;
    else Destroy(gameObject);
}
```

4.62 ChemistryEquipmentManager 4

Σιγουρεύει ότι υπάρχει **μόνο ένα** ChemistryEquipmentManager στη σκηνή.

Γεμίσματα και Αδειάσματα

```
public void FillWater()
{
    equipments[0].liquidVolume.level = .5f;
}
```

4.63 ChemistryEquipmentManager.cs 5

Παρόμοια μέθοδος υπάρχει για το **FillOil()** και το **FillInk()**. Η τιμή .5 σημαίνει ότι το ποτήρι γεμίζει στο 50% της χωρητικότητάς του.

Για το άδειασμα:

```
public void EmptyBeaker()
{
    for (int i = 0; i < equipments[2].liquidVolume.liquidLayers.Length; i++)
    {
        equipments[2].liquidVolume.liquidLayers[i].amount = 0;
        equipments[2].liquidVolume.UpdateLayers();
    }
    emptyBeaker.ResetWaterColor();
}
```

4.64 ChemistryEquipmentManager.cs 6

- Όλα τα layers του υγρού μηδενίζονται.
- Το UpdateLayers() της Liquid Volume Pro εφαρμόζει την αλλαγή.
- Η μέθοδος ResetWaterColor() από την EmptyBeaker επαναφέρει το χρώμα του νερού.

Όλα τα references ορίζονται μέσα από το Unity Inspector. Έτσι, ο manager έχει πρόσβαση σε όλα τα αντικείμενα της σκηνής χωρίς να τα αναζητά μέσω κώδικα.

Συνοπτικά, η **ChemistryEquipmentManager** :

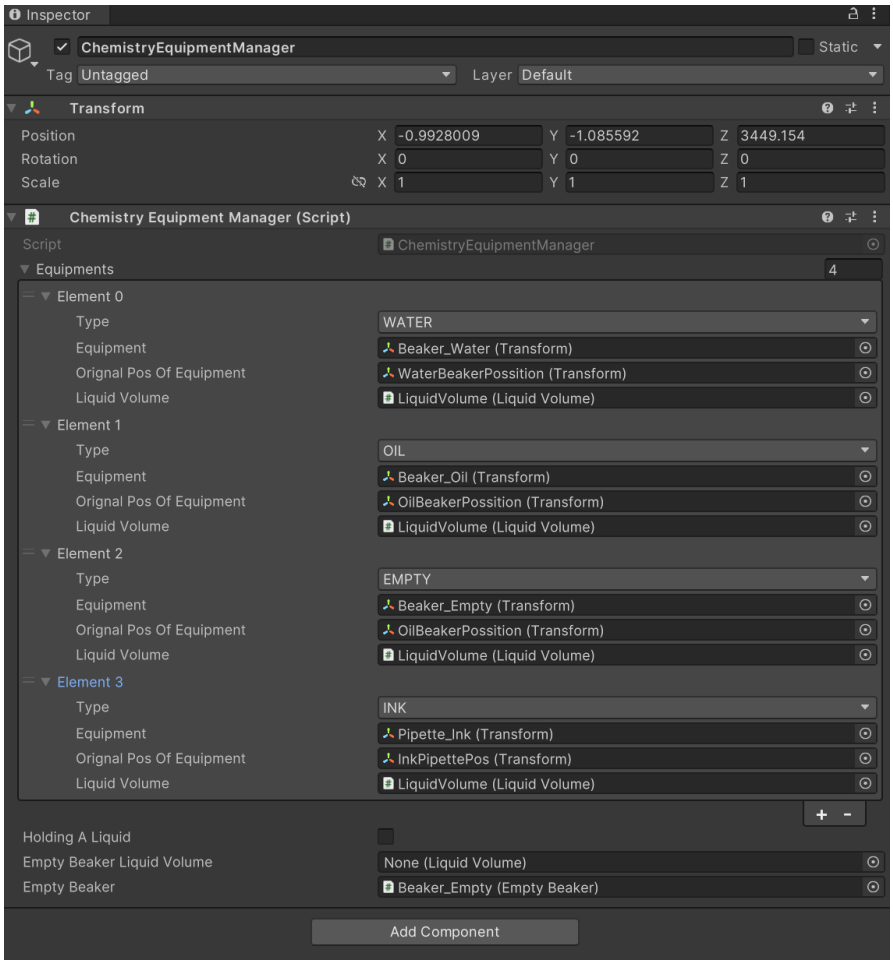
1. Παρέχει singleton πρόσβαση (instance) σε άλλες κλάσεις για τη διαχείρηση των αντικείμενων αλληλεπίδρασης.

2. Ορίζει ένα πίνακα sEquipments με αναφορές Transform / LiquidVolume για κάθε αντικείμενο του χημικού εξοπλισμού.

3. Προσφέρει μεθόδους για να γεμίσει ο εξοπλισμός με τα υγρά.

4. «Αδειάζει» το «άδειο» δοχείο μηδενίζοντας όλα τα στρώματα υγρών (liquidLayers[i].amount) και καλώντας την UpdateLayers() και την emptyBeaker.ResetWaterColor().

5. Ορίζει το global flag holdingALiquid που χρησιμοποιείται από άλλες κλάσεις για την αποτροπή διπλού κρατήματος.



4.65 ChemistryEquipmentManager Inspector View

4.6.5 BeakerPourer.cs

Η κλάση **BeakerPourer** είναι υπεύθυνη για τον έλεγχο της διαδικασίας χυσίματος υγρών από ένα ποτήρι ζέσεως (beaker) σε ένα άλλο.

Διαχειρίζεται:

- Την κλίση του ποτηριού ώστε να ξεκινήσει η ροή.
- Την προσομοίωση της ροής μέσω Liquid Volume Pro.
- Τη δημιουργία σταγόνων μελανιού ή υπερχείλισης.
- Την ενημέρωση του Empty Beaker μέσω του ChemistryEquipmentManager.

4.6.5.1 Μεταβλητές

```
[Header("Target Beaker Detection")]
3 references
public Transform targetBeaker;          // The receiving beaker
1 reference
public float horizontalRange = 0.3f;    // Max XZ distance to trigger pour
1 reference
public float verticalOffset = 0.2f;     // Must be this much higher to pour
```

4.66 BeakerPourer.cs 1

- **targetBeaker**: Το ποτήρι-στόχος όπου καταλήγει το υγρό (ορίζεται στο Inspector).
- **horizontalRange**: Μέγιστη οριζόντια απόσταση ώστε να επιτραπεί το χύσιμο.
- **verticalOffset**: Το ποτήρι πρέπει να είναι τουλάχιστον αυτό το ύψος πάνω από τον στόχο.

```
[Header("Dynamic Rotation Settings")]
1 reference
public float maxPourAngle = 70f;          // Maximum rotation angle when very high
1 reference
public float minPourAngle = 20f;          // Minimum tilt angle when just above
1 reference
public float angleMultiplier = 150f;      // How quickly tilt increases with height
2 references
public float rotationSpeed = 30f;         // Degrees per second
1 reference
public bool autoReturn = true;            // Return to upright when not pouring
```

4.67 BeakerPourer.cs 2

- **minPourAngle:** Ελάχιστη γωνία κλίσης για ροή.
- **maxPourAngle**: Μέγιστη γωνία κλίσης για ροή.
- **angleMultiplier**: Επηρεάζει πόσο αυξάνεται η γωνία της κλίσης ανάλογα με το ύψος.
- **rotationSpeed**: Ταχύτητα περιστροφής.
- **autoReturn**: Όταν είναι ενεργό, το ποτήρι επιστρέφει αυτόματα στη θέση του αν σταματήσει η ροή.

```
[Header("Liquid Drain Settings")]
9 references
public LiquidVolume liquidVolume;
0 references
public int layerIndex = 0;
0 references
public float drainStartAngle = 30f;
0 references
public float drainSpeed = 0.1f;
```

4.68 BeakerPourer.cs 3

- **liquidVolume**: Το Liquid Volume Pro component του ποτηριού.
- **layerIndex**: Η «στρώση» του υγρού που θα επηρεαστεί.
- **drainStartAngle**: Από πόσες μοίρες κλίσης και πάνω ξεκινάει η ροή.
- **drainSpeed**: Πόσο γρήγορα αδειάζει το υγρό.

```
2 references | 3 references
[SerializeField] private float horizontalDistance, heightDifference;
3 references
public bool isPouring = false;
```

4.69 BeakerPourer.cs 4

- **horizontalDistance**: Οριζόντια απόσταση από το ποτήρι-δέκτη.
- **heightDifference**: Διαφορά ύψους.
- **isPouring**: Σημαία που δείχνει αν αυτήν τη στιγμή γίνεται ροή.

```
[Header("Drip Visuals")]
2 references
public GameObject spillDropPrefab;
2 references
public int maxActiveSpills = 10;

2 references
public Transform spillPoint;

1 reference
public float spillSensitivity = 0.002f;

3 references
public eLiquidType type;
```

4.70 BeakerPourer.cs 5

- **spillDropPrefab**: Το prefab της σταγόνας, ορισμένο από το Inspector (Liquid Volume Pro έτοιμο prefab).
- **maxActiveSpills**: Μέγιστος αριθμός σταγόνων ανά κύκλο.
- **spillPoint**: Το Transform που δείχνει στο στόμιο του ποτηριού, από όπου δημιουργούνται οι σταγόνες.
- **spillSensitivity**: Ελέγχει πόσο κοντά πρέπει να είναι η στάθμη του υγρού στο χείλος ώστε να ξεκινήσει το χύσιμο.

```
3 references
public eLiquidType type;

3 references
bool isInkDropDrain;
```

4.71 BeakerPourer.cs 6

- **type**: Ορίζει τον τύπο του υγρού (WATER, OIL, ή INK).
- **isInkDropDrain**: Χρησιμοποιείται ώστε οι σταγόνες μελανιού να παράγονται μόνο μία φορά ανά κλίση.

```
float pouringAmount = 0.002f;
[SerializeField]
1 reference
int drops = 30;
1 reference
```

4.72 BeakerPourer.cs 7

- **pouringAmount**: Ελάχιστη ποσότητα που αφαιρείται ανά ροή.
- **Drops**: Αριθμός σταγόνων που δημιουργούνται ανά κύκλο αποστράγγισης.

4.6.5.2 Μέθοδοι RotateTowardsPour() & RotateBackToOriginal()

```
1 reference
void RotateTowardsPour(float targetAngle)
{
    float currentZ = NormalizeAngle(transform.localEulerAngles.z);
    float targetZ = Mathf.MoveTowards(currentZ, targetAngle, rotationSpeed * Time.deltaTime);
    transform.localEulerAngles = new Vector3(
        transform.localEulerAngles.x,
        transform.localEulerAngles.y,
        targetZ
    );
}
```

4.73 BeakerPourer.cs 8

```
void RotateBackToOriginal()
{
    float currentZ = NormalizeAngle(transform.localEulerAngles.z);
    float targetZ = Mathf.MoveTowards(currentZ, 0f, rotationSpeed * Time.deltaTime);
    transform.localEulerAngles = new Vector3(
        transform.localEulerAngles.x,
        transform.localEulerAngles.y,
        targetZ
    );

    if (Mathf.Approximately(targetZ, 0f))
        isPouring = false;
}
```

4.74 BeakerPourer.cs 9

Η μέθοδος **RotateTowardsPour()** γέρνει ομαλά το ποτήρι μέχρι να φτάσει τη γωνία που απαιτείται για τη ροή. Υπολογίζει την τρέχουσα γωνία στον τοπικό άξονα Z και τη μεταβάλλει σταδιακά χρησιμοποιώντας **Mathf.MoveTowards**.

Η **RotateBackToOriginal()** λειτουργεί αντίστροφα, επαναφέροντας το ποτήρι στην αρχική του θέση. Όταν η γωνία επανέλθει πλήρως, το flag **isPouring** μηδενίζεται.

4.6.5.3 Μέθοδος Reset()

```
1 reference
public void Reset()
{
    RotateBackToOriginal();
    isInkDropDrain = false;
}
```

4.75 BeakerPourer.cs 10

- Επαναφέρει το ποτήρι στην όρθια θέση.
- Επαναφέρει τη σημαία **isInkDropDrain**, ώστε να μπορούν να δημιουργηθούν ξανά σταγόνες μελανιού στην επόμενη προσπάθεια.

4.6.5.4 Μέδοθος DrainLiquid()

```
void DrainLiquid()
{
    if (liquidVolume == null) return;

    Vector3 spillPos;
    float spillAmount;
    liquidVolume.spillEdge = spillPoint;

    if (liquidVolume.GetSpillPoint(out spillPos, out spillAmount, liquidVolume.level, spillSensitivity))
    {

        for (int k = 0; k < drops; k++)
        {
            int template = Random.Range(0, maxActiveSpills);
            GameObject oneSpill = Instantiate(spillDropPrefab) as GameObject;
            oneSpill.SetActive(true);
            Rigidbody rb = oneSpill.GetComponent<Rigidbody>();
            rb.position = spillPos + Random.insideUnitSphere * 0.01f;
            rb.AddForce(new Vector3(Random.value - 0.5f, Random.value * 0.1f - 0.2f, Random.value - 0.5f));
        }
        liquidVolume.level -= spillAmount / 10f + pouringAmount;
        if (ChemistryEquipmentManager.instance.emptyBeakerLiquidVolume != null)
        {
            if(type==eLiquidType.WATER)
            ChemistryEquipmentManager.instance.emptyBeakerLiquidVolume.liquidLayers[0].amount += spillAmount / 10f + pouringAmount;
            else if (type == eLiquidType.OIL)
                ChemistryEquipmentManager.instance.emptyBeakerLiquidVolume.liquidLayers[1].amount += spillAmount / 10f + pouringAmount;

            ChemistryEquipmentManager.instance.emptyBeakerLiquidVolume.UpdateLayers();
        }
    }
}
```

4.76 BeakerPourer.cs 11

Η **DrainLiquid**() χειρίζεται τη δημιουργία σταγόνων, την προσαρμογή της στάθμης του υγρού και την ενημέρωση του δοχείου-στόχου.

Αρχικά, ορίζεται το σημείο υπερχείλισης (**spillEdge**) που αντιστοιχεί στο χείλος του δοχείου. Στη συνέχεια, η μέθοδος **GetSpillPoint()** του Liquid Volume Pro υπολογίζει αν το υγρό έχει φτάσει στο χείλος και επιστρέφει:

- το **ακριβές σημείο ροής** (spillPos),
- και την **ποσότητα υγρού** που μπορεί να χυθεί (spillAmount).

Εφόσον εντοπιστεί υπερχείλιση, δημιουργούνται σταγόνες με το **Instantiate()** και εφαρμόζεται **τυχαία δύναμη** μέσω της **Rigidbody.AddForce()** για να φαίνεται φυσική η κίνηση τους. Στη συνέχεια, η στάθμη του υγρού στο τρέχον δοχείο (**liquidVolume.level**) μειώνεται, ενώ ταυτόχρονα ενημερώνεται

και το δοχείο-στόχος. Η κατάλληλη ποσότητα προστίθεται στο σωστό **layer** μέσα στον πίνακα **liquidLayers[]** του δοχείου-στόχου, και το **UpdateLayers()** εξασφαλίζει ότι η νέα στάθμη εμφανίζεται σωστά στην οθόνη.

Λειτουργίες Liquid Volume Pro

- **level**: Παρακολουθεί το επίπεδο υγρών του δοχείου (0 έως 1).
- **spillEdge**: Ορίζει το σημείο απ' όπου υπολογίζεται η υπερχείλιση.
- **GetSpillPoint**(): Επιστρέφει το ακριβές σημείο και την ποσότητα του υγρού που πρόκειται να χυθεί.
- **liquidLayers**[]: Διαχειρίζεται τα διαφορετικά στρώματα υγρού στο δοχείο (νερό, λάδι, μελάνι).
- **UpdateLayers**(): Ενημερώνει την απεικόνιση ώστε να φαίνεται σωστά η αλλαγή της στάθμης.

4.6.5.5 Μέθοδος DrainInkDrop()

```
void DrainInkDrop()
{
    if (liquidVolume == null) return;

    Vector3 spillPos = spillPoint.position;

    if (liquidVolume.level > 0)
    {
        for (int k = 0; k < 1; k++)
        {
            int template = Random.Range(0, maxActiveSpills);
            GameObject oneSpill = Instantiate(spillDropPrefab) as GameObject;
            oneSpill.SetActive(true);
            Rigidbody rb = oneSpill.GetComponent<Rigidbody>();
            rb.position = spillPos + Random.insideUnitSphere * 0.01f;
            rb.AddForce(new Vector3(Random.value - 0.5f, Random.value * 0.1f - 0.2f, Random.value - 0.5f));

        }
        liquidVolume.level -= 1 / 10f + pouringAmount;
        if (ChemistryEquipmentManager.instance.emptyBeakerLiquidVolume != null)
        {
            if (ChemistryEquipmentManager.instance.emptyBeakerLiquidVolume.liquidLayers[1].amount == 0)
            {
                ChemistryEquipmentManager.instance.emptyBeakerLiquidVolume.liquidLayers[2].amount += .1f / 10f + pouringAmount;
                ChemistryEquipmentManager.instance.emptyBeakerLiquidVolume.UpdateLayers();
            }

        }
    }

}
```

4.77 BeakerPourer.cs 12

Σε αντίθεση με τα υπόλοιπα υγρά, η μέθοδος αυτή, παρόμοια με την **DrainLiquid()**, δημιουργεί **μία μόνο σταγόνα κάθε φορά** και μειώνει τη στάθμη πολύ πιο αργά. Αυτό μιμείται την πραγματική συμπεριφορά του μελανιού, το οποίο στάζει πιο σταδιακά.

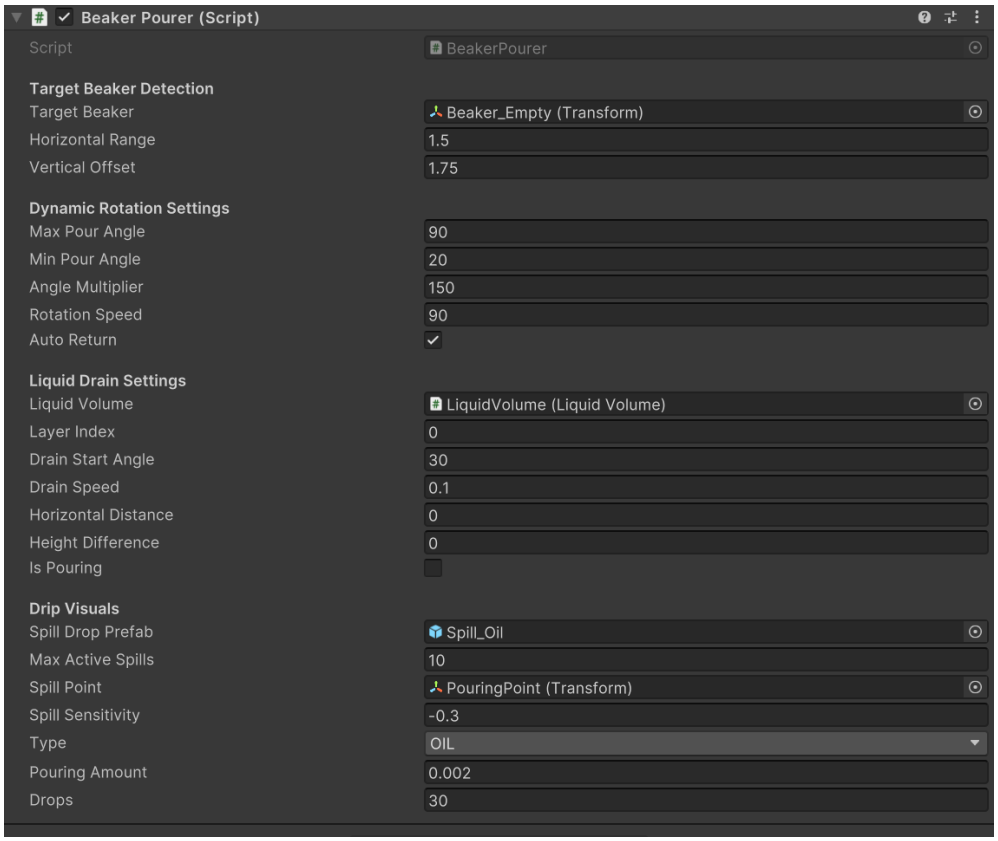
4.6.5.6 Μέθοδος Update()

Η βασική λειτουργικότητα της κλάσης υλοποιείται στη μέθοδο **Update()**. Σε κάθε frame ελέγχονται η απόσταση και η διαφορά ύψους ανάμεσα στο τρέχον δοχείο και το δοχείο-στόχο. Αν πληρούνται οι προϋποθέσεις, η μέθοδος ξεκινά τη διαδικασία ροής.

```
void Update()
{
    if (targetBeaker == null || liquidVolume == null) return;

    // Measure position difference
    Vector3 positionOffset = targetBeaker.position - transform.position;
    horizontalDistance = new Vector2(positionOffset.x, positionOffset.z).magnitude;
    heightDifference = transform.position.y - targetBeaker.position.y;

    // Check if within pour rangeW
    if (horizontalDistance < horizontalRange && heightDifference > verticalOffset)
    {
        isPouring = true;

        if (type != eLiquidType.INK)
        {
            float dynamicPourAngle = Mathf.Clamp(
                heightDifference * angleMultiplier, minPourAngle, maxPourAngle);
            RotateTowardsPour(dynamicPourAngle);

            // Drain the liquid
            DrainLiquid();
        }
        else
        {
            if (!isInkDropDrain)
            {
                DrainInkDrop();
                isInkDropDrain = true;
            }

        }


    }
    else if (autoReturn && isPouring)
    {
        Reset();
    }
}
```

4.78 BeakerPourer.cs 13

Αρχικά, υπολογίζεται η **οριζόντια απόσταση** μεταξύ των δύο δοχείων και η **διαφορά ύψους**. Αν το τρέχον δοχείο είναι αρκετά κοντά και βρίσκεται πιο ψηλά από το άλλο, ενεργοποιείται η ροή. Στην περίπτωση που το υγρό είναι νερό ή λάδι, το ποτήρι περιστρέφεται δυναμικά προς τη σωστή γωνία με τη μέθοδο **RotateTowardsPour()** και καλείται η **DrainLiquid()** για να ξεκινήσει η ροή. Στην περίπτωση του μελανιού, χρησιμοποιείται η **DrainInkDrop()**. Αν σταματήσει να ισχύει η συνθήκη ροής, το ποτήρι επανέρχεται σταδιακά στην αρχική του θέση μέσω της **Reset()**.



4.79 BeakerPourer Inspector View

Η **BeakerPourer** ελέγχει τη ροή υγρών μεταξύ των δοχείων στο πείραμα. Ανιχνεύει πότε το ποτήρι γέρνει αρκετά, υπολογίζει το σημείο και την ποσότητα υπερχείλισης και συνεργάζεται με το **Liquid Volume Pro** για να προσομοιώσει ρεαλιστικά την κίνηση του υγρού. Περιστρέφει δυναμικά το ποτήρι, δημιουργεί σταγόνες μέσω **prefabs** και εφαρμόζει φυσικές δυνάμεις (**Rigidbody.AddForce**) για να φαίνονται φυσικές. Παράλληλα, ενημερώνει τη στάθμη του υγρού στο τρέχον δοχείο και προσθέτει την αντίστοιχη ποσότητα στο δοχείο-στόχο. Για το μελάνι, η ροή γίνεται πιο σταδιακά ώστε να μιμείται το αργό στάξιμο.

4.6.6 ChekingIfPouringLiquidInBeaker.cs

Η κλάση **CheckingIfPouringLiquidinBeaker** λειτουργεί ως σύνδεσμος ανάμεσα στο **BeakerPourer.cs** και το δοχείο που πρόκειται να δεχτεί το υγρό. Ο ρόλος του είναι να ελέγχει αν το υγρό κατευθύνεται πράγματι σε ένα κατάλληλο δοχείο και, αν ναι, να συνδέει αυτό το δοχείο με το **ChemistryEquipmentManager** ώστε να ενημερώνεται σωστά η στάθμη του υγρού.

4.6.6.1 Μεταβλητές

```
1 reference
public BeakerPourer myBeaker;
1 reference
public LayerMask layerToCheck;
```

4.80 CheckingIfPouringLiquidInBeaker.cs 1

Η μεταβλητή **myBeaker** συνδέεται με την **BeakerPourer** και την ενημερώνει αν το ποτήρι βρίσκεται σε κατάσταση ροής. Η **layerToCheck** καθορίζει ποια επίπεδα θα λαμβάνονται υπόψιν από το **raycast**,

ώστε να γίνεται έλεγχος μόνο για αντικείμενα που σχετίζονται με τη ροή (τα δοχεία) και να αγνοούνται άσχετα αντικείμενα.

4.6.6.2 Μέθοδος Update()

```
private void Update()
{
    if (!myBeaker.isPouring) return;
    if (Physics.Raycast(transform.position,transform.forward,out RaycastHit hit, layerToCheck))
    {
        if (hit.transform.CompareTag("EmptyBeaker"))
        {
            Debug.DrawLine(transform.position, hit.point, Color.red);
            ChemistryEquipmentManager.instance.emptyBeakerLiquidVolume = hit.transform.GetComponentInParent<EmptyBeaker>().myLiquidVolume;
        }
        else
        {
            ChemistryEquipmentManager.instance.emptyBeakerLiquidVolume = null;
        }
    }
    else
    {
        ChemistryEquipmentManager.instance.emptyBeakerLiquidVolume = null;
    }
```

4.81 CheckingIfPouringLiquidInBeaker.cs 2

Η μέθοδος εκτελείται σε κάθε frame και περιλαμβάνει τρία βασικά βήματα:

1. Έλεγχος κατάστασης ροής:

   Αν η **BeakerPourer** δεν έχει ενεργοποιήσει τη ροή ($isPouring == false$), τερματίζεται άμεσα η εκτέλεση, ώστε να μην κάνει περιττούς ελέγχους.

2. Χρήση **Raycast** για ανίχνευση στόχου:

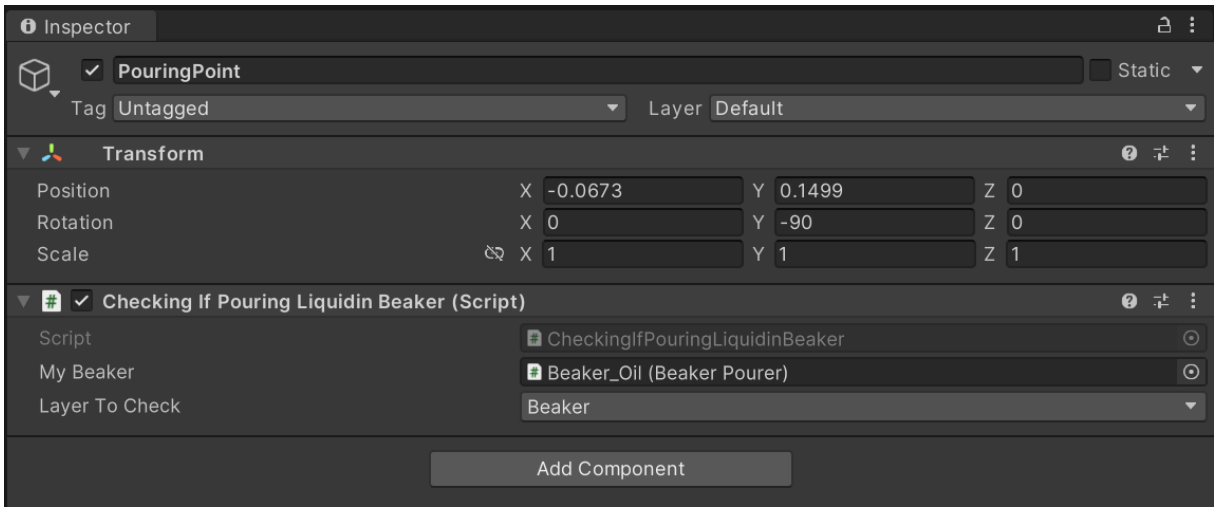
   Εκτελείται ένα **raycast** από τη θέση του τρέχοντος αντικειμένου προς τα εμπρός. Αν το ray εντοπίσει σύγκρουση με αντικείμενο, γίνεται περαιτέρω έλεγχος για να διαπιστωθεί αν πρόκειται για έγκυρο δοχείο.

3. Έλεγχος αν ο στόχος είναι **EmptyBeaker**:

4. Αν το αντικείμενο έχει το tag “**EmptyBeaker**” ενημερώνεται το

   ChemistryEquipmentManager με το **LiquidVolume** του συγκεκριμένου δοχείου.

5. Αν ο στόχος δεν είναι κατάλληλο δοχείο, ή αν δεν υπάρχει χτύπημα από το **raycast**, η τιμή ορίζεται σε **null**. Με αυτόν τον τρόπο το παιχνίδι γνωρίζει ότι δεν υπάρχει έγκυρος στόχος για ροή υγρού.



4.82 CheckingIfPouringLiquidInBeaker Inspector View

Τι είναι το Raycasting

Το **raycasting** στη Unity είναι μια τεχνική που «εκτοξεύει» μια αόρατη ακτίνα από ένα σημείο προς μια κατεύθυνση και ελέγχει αν αυτή συναντά κάποιο αντικείμενο με **Collider**. Η μέθοδος

Physics.Raycast() επιστρέφει μια **τιμή τύπου bool**:

- **true**: όταν η ακτίνα χτυπήσει κάποιο αντικείμενο.
- **false**: όταν δεν βρεθεί καμία σύγκρουση.

Επιπλέον, όταν χρησιμοποιούμε την παράμετρο **out RaycastHit hit**, η Unity αποθηκεύει σε αυτήν τη δομή πληροφορίες για το αντικείμενο που χτυπήθηκε, **μόνο αν η μέθοδος επιστρέψει true**. Μέσα στο hit μπορούμε να βρούμε:

- **hit.point**: το ακριβές σημείο πρόσκρουσης,
- **hit.transform**: το αντικείμενο που χτυπήθηκε,
- **hit.distance**: την απόσταση του σημείου επαφής από το σημείο εκκίνησης. [17]

Στο πλαίσιο του παιχνιδιού, το raycasting χρησιμοποιείται για να ελέγχει αν το υγρό κατευθύνεται σε **EmptyBeaker**. Έτσι το παιχνίδι γνωρίζει πού πρέπει να καταλήξουν οι ενημερώσεις της στάθμης υγρού.

Σημείωση:

Η μέθοδος **Physics.Raycast** ορίζεται ως εξής:

Physics.Raycast(Vector3 origin, Vector3 direction, out RaycastHit hitInfo, float maxDistance).

Αυτό σημαίνει, ότι στην τρέχουσα περίπτωση το **layerToCheck** (**LayerMask**) περνά ως 4ο όρισμα και αντιμετωπίζεται ως **maxDistance (πχ 1, 2, 4, 8…)**. Δηλαδή, δεν γίνεται φιλτράρισμα των layers, αλλά ορίζεται ένα μήκος ακτίνας ίσο με τον ακέραιο της μάσκας.

Άρα, με την τρέχουσα κλήση, το ray θα χτυπήσει οποιοδήποτε layer, σε απόσταση έως «τιμή μάσκας».

4.6.7 EmptyBeaker.cs

Η κλάση **EmptyBeaker** είναι υπεύθυνη για τη συμπεριφορά του δοχείου που δέχεται υγρά στο πείραμα. Συνεργάζεται με το **Liquid Volume Pro** ώστε να διαχειρίζεται την ποσότητα, το χρώμα και τα στρώματα των υγρών, καθώς και τη διαδικασία ανάμειξης μελανιού και νερού. Παράλληλα, ελέγχει αν το δοχείο έχει γεμίσει, δημιουργεί εφέ υπερχείλισης και καταστρέφει σταγόνες που δεν χρειάζονται πλέον.

4.6.7.1 Μεταβλητές

```
22 references
public LiquidVolume myLiquidVolume;
1 reference
public GameObject spillDropPrefab;
1 reference
public Transform spillSpawnPoint;

0 references
public Transform beakerBase;
1 reference
public float beakerHeight = 0.2f;

3 references
public Color forInkWater;
1 reference
public Color normalWaterColor;

1 reference
private bool isbeakerFull;

1 reference
public bool hasInkDrop;
```

4.83 EmptyBeaker.cs 1

myLiquidVolume: το component του **Liquid Volume Pro** που χειρίζεται τη στάθμη και το χρώμα του υγρού.

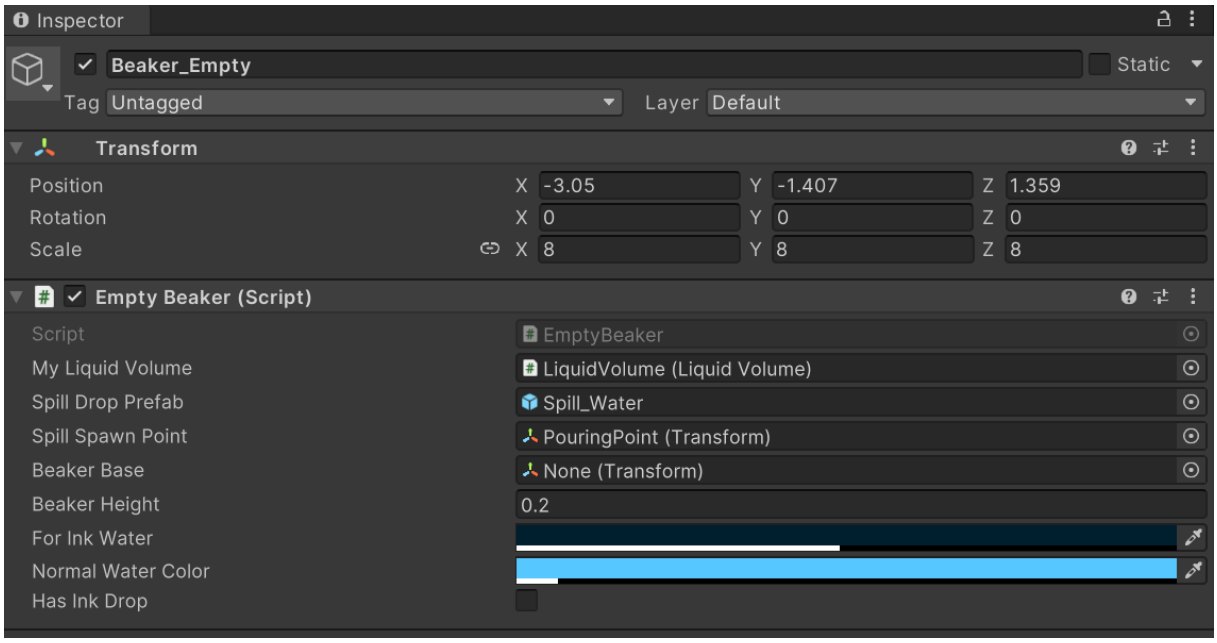
spillDropPrefab & **spillSpawnPoint**: χρησιμοποιούνται για τη δημιουργία εφέ σταγόνων όταν το δοχείο ξεχειλίζει.

beakerBase & **beakerHeight**: καθορίζουν το ύψος του δοχείου και χρησιμοποιούνται για τον υπολογισμό της θέσης των σταγόνων.

forInkWater & **normalWaterColor**: τα χρώματα που εφαρμόζονται στο υγρό όταν αναμειγνύεται με μελάνι ή όταν επαναφέρεται στο αρχικό χρώμα.

isbeakerFull: δείχνει αν το δοχείο έχει γεμίσει.

hasInkDrop: υποδεικνύει αν υπάρχουν ενεργές σταγόνες μελανιού στη σκηνή.



4.84 EmptyBeaker Inspector View

4.6.7.2 Μέθοδος ResetWaterColor()

```
public void ResetWaterColor()
{
    myLiquidVolume.liquidLayers[0].color = normalWaterColor;
    myLiquidVolume.UpdateLayers();
    GameObject[] allInkDrops = GameObject.FindGameObjectsWithTag("InkDrop");
    for (int i = 0; i < allInkDrops.Length; i++)
    {
        Destroy(allInkDrops[i]);
    }
}
```

4.85 EmptyBeaker.cs 2

Επαναφέρει το χρώμα του νερού στην αρχική του κατάσταση και καθαρίζει όλες τις σταγόνες μελανιού. Καλείται όταν το πείραμα «επαναφέρεται».

4.6.7.3 Μέθοδος SpawnSpillParticle()

```
void SpawnSpillParticle(Color color)
{
    GameObject drop = Instantiate(spillDropPrefab);
    drop.GetComponent<Renderer>().material.color = color;
    drop.transform.position = spillSpawnPoint.position; // edge of beaker
    drop.SetActive(true);

    Rigidbody rb = drop.GetComponent<Rigidbody>();
    rb.AddForce(Vector3.down * 0.5f + Random.insideUnitSphere * 0.2f);

    Destroy(drop, 2f);
}
```

4.86 EmptyBeaker.cs 3

Δημιουργεί σταγόνες στο σημείο υπερχείλισης και προσθέτει τυχαία φυσική κίνηση μέσω της **Rigidbody.AddForce()**. Οι σταγόνες αυτοκαταστρέφονται μετά από 2 δευτερόλεπτα.

4.6.7.4 Μέθοδος IsBeakerFull()

```
bool IsBeakerFull(LiquidVolume lv)
{
    float totalAmount = 0f;
    foreach (var layer in lv.liquidLayers)
        totalAmount += layer.amount;

    return totalAmount >= 1.0f;
}
```

4.87 EmptyBeaker.cs 4

Ελέγχει αν το συνολικό ύψος όλων των στρωμάτων υγρού έχει φτάσει το μέγιστο (**1.0**), ώστε να ενεργοποιηθεί η υπερχείλιση.

4.6.7.5 Μέθοδοι GetLowestDensityLayerIndex() & GetSecondLowestDensityLayerIndex()

```
int GetLowestDensityLayerIndex(LiquidVolume lv)
{
    int minIndex = 0;
    float minDensity = lv.liquidLayers[0].density;

    for (int i = 1; i < lv.liquidLayers.Length; i++)
    {
        if (lv.liquidLayers[i].density < minDensity)
        {
            minDensity = lv.liquidLayers[i].density;
            minIndex = i;
        }
    }
    return minIndex;
}

1 reference
int GetSecondLowestDensityLayerIndex(LiquidVolume lv, int excludeIndex)
{
    float minDensity = float.MaxValue;
    int secondIndex = -1;

    for (int i = 0; i < lv.liquidLayers.Length; i++)
    {
        if (i == excludeIndex) continue;

        if (lv.liquidLayers[i].density < minDensity)
        {
            minDensity = lv.liquidLayers[i].density;
            secondIndex = i;
        }
    }
    return secondIndex;
}
```

4.88 EmptyBeaker.cs 5

GetLowestDensityLayerIndex()

Χρησιμοποιώντας το LiquidVolume Component, επιστρέφει το **index** του στρώματος με τη μικρότερη πυκνότητα, ώστε να αποφασιστεί ποιο υγρό θα ξεχειλίσει πρώτο.

GetSecondLowestDensityLayerIndex()

Χρησιμοποιείται για να υπολογίσει το δεύτερο ελαφρύτερο στρώμα, αν το πρώτο έχει ήδη αδειάσει.

4.6.7.6 Μέθοδος GetCurrentLayerIndex()

```
int GetCurrentLayerIndex(Vector3 dropPosition)
{
    // Get local Y distance from beaker base
    float localDropY = transform.InverseTransformPoint(dropPosition).y;

    float cumulativeHeight = 0f;
    for (int i = 0; i < myLiquidVolume.liquidLayers.Length; i++)
    {
        float layerHeight = myLiquidVolume.liquidLayers[i].amount * beakerHeight;
        float layerTop = cumulativeHeight + layerHeight;

        if (localDropY <= layerTop)
            return i; // Drop has touched this layer

        cumulativeHeight += layerHeight;
    }

    return -1; // Drop is above all layers
}
```

4.89 EmptyBeaker.cs 6

Υπολογίζει σε ποιο στρώμα υγρού θα πέσει μια σταγόνα, συγκρίνοντας το ύψος της με το άθροισμα του ύψους όλων των στρωμάτων.

- Αν βρεθεί έγκυρο στρώμα, επιστρέφεται ο δείκτης του.
- Αν η σταγόνα βρίσκεται πάνω από όλα τα στρώματα, επιστρέφεται **-1**.

4.6.7.7 Μέθοδος MixInkWithWater()

```
public void MixInkWithWater()
{
    GameObject[] allInkDrops = GameObject.FindGameObjectsWithTag("InkDrop");
    myLiquidVolume.liquidLayers[0].color = forInkWater;
    myLiquidVolume.UpdateLayers();
    for (int i = 0; i < allInkDrops.Length; i++)
    {
        Destroy(allInkDrops[i]);
    }
}
```

4.90 EmptyBeaker.cs 7

Αυτή η μέθοδος αναμιγνύει το μελάνι με το νερό όταν ο χρήστης κάνει συγκεκριμένη κίνηση χειρονομίας (**gesture**). Αλλάζει το χρώμα του νερού, ενημερώνει την οπτική απεικόνιση και καθαρίζει όλες τις σταγόνες μελανιού.

4.6.7.8 Μέθοδος DetectDropLayerContact()

```
void DetectDropLayerContact()
{
    bool inkTouchWater = false;
    GameObject[] allInkDrops = GameObject.FindGameObjectsWithTag("InkDrop");

    foreach (var drop in allInkDrops)
    {
        int layerIndex = GetCurrentLayerIndex(drop.transform.position);
        if (layerIndex != -1)
        {
            if (layerIndex == 0)
            {
                inkTouchWater = true;
                break;
            }
        }
    }

    if (inkTouchWater)
    {
        myLiquidVolume.liquidLayers[0].color = forInkWater;
        myLiquidVolume.UpdateLayers();
        for (int i = 0; i < allInkDrops.Length; i++)
        {
            Destroy(allInkDrops[i]);
        }
    }
}
```

4.91 EmptyBeaker.cs 8

Η μέθοδος ελέγχει αν κάποια σταγόνα μελανιού πέφτει στο πρώτο στρώμα (**νερό**).

- Αν υπάρχει επαφή, αλλάζει το χρώμα του νερού σε **forInkWater**.
- Ενημερώνει την απεικόνιση με **UpdateLayers()**.
- Καταστρέφει όλες τις σταγόνες μελανιού, καθώς θεωρούνται απορροφημένες.

4.6.7.9 Μέθοδος HandleMixing()

```
void HandleMixing()
{
    var layers = myLiquidVolume.liquidLayers;

    // Water = index 0, Oil = index 1, Ink = index 2
    if (layers.Length >= 3 && layers[0].amount > 0 && layers[2].amount > 0 && layers[1].amount == 0)
    {
        float inkAmount = layers[2].amount;

        // Mix ink into water
        layers[0].amount += inkAmount;
        layers[2].amount = 0f;

        layers[0].color = forInkWater;
        myLiquidVolume.UpdateLayers();
    }
}
```

4.92 EmptyBeaker.cs 9

Εκτελεί την ανάμειξη όταν:

- υπάρχει νερό (**layer 0**),
- υπάρχει μελάνι (**layer 2**),
- δεν υπάρχει λάδι (**layer 1**).

Η ποσότητα του μελανιού ενσωματώνεται στο νερό και ενημερώνεται το χρώμα του.

4.6.7.10 Μέθοδος HandleSpilling()

```
void HandleSpilling()
{
    float totalAmount = 0f;
    foreach (var layer in myLiquidVolume.liquidLayers)
        totalAmount += layer.amount;

    if (totalAmount > 1f)
    {
        int topLayerIndex = GetLowestDensityLayerIndex(myLiquidVolume);
        float spillStep = 0.01f;

        // Spill the lowest density liquid first
        if (myLiquidVolume.liquidLayers[topLayerIndex].amount > 0f)
        {
            myLiquidVolume.liquidLayers[topLayerIndex].amount -= spillStep;
            SpawnSpillParticle(myLiquidVolume.liquidLayers[topLayerIndex].color);
            myLiquidVolume.UpdateLayers();
        }
        else
        {
            // If top layer is empty, spill the next-lowest density
            int secondTopLayerIndex = GetSecondLowestDensityLayerIndex(myLiquidVolume, topLayerIndex);
            if (secondTopLayerIndex != -1 && myLiquidVolume.liquidLayers[secondTopLayerIndex].amount > 0f)
            {
                myLiquidVolume.liquidLayers[secondTopLayerIndex].amount -= spillStep;
                SpawnSpillParticle(myLiquidVolume.liquidLayers[secondTopLayerIndex].color);
                myLiquidVolume.UpdateLayers();
            }
        }
    }
}
```

4.93 EmptyBeaker.cs 10

Η μέθοδος ελέγχει αν το δοχείο έχει γεμίσει:

- Αν ναι, βρίσκει το στρώμα με τη μικρότερη πυκνότητα (συνήθως λάδι).
- Αφαιρεί σταδιακά μικρές ποσότητες υγρού (spillStep).
- Δημιουργεί οπτικά εφέ υπερχείλισης καλώντας το **SpawnSpillParticle()**.
- Αν το πρώτο στρώμα είναι άδειο, ελέγχει το επόμενο ελαφρύτερο.

4.6.7.11 Μέθοδος Update()

```
0 references
void Update()
{
    hasInkDrop = GameObject.FindGameObjectsWithTag("InkDrop").Length > 0;

    isbeakerFull = IsBeakerFull(myLiquidVolume);

    HandleSpilling();
    HandleMixing();

    // Example: check all falling drops in scene
    DetectDropLayerContact();
}
```

4.94 EmptyBeaker.cs 11

Η μέθοδος **Update()** τρέχει σε κάθε frame και εκτελεί τέσσερις βασικές λειτουργίες:

- Ελέγχει αν υπάρχουν σταγόνες μελανιού στη σκηνή (**hasInkDrop**).
- Υπολογίζει αν το δοχείο έχει γεμίσει καλώντας τη **IsBeakerFull()**.
- Ελέγχει αν πρέπει να ξεκινήσει υπερχείλιση μέσω της **HandleSpilling()**.
- Διαχειρίζεται την ανάμειξη υγρών και την επαφή σταγόνων με νερό.

Συνοπτικά, η **EmptyBeaker**:

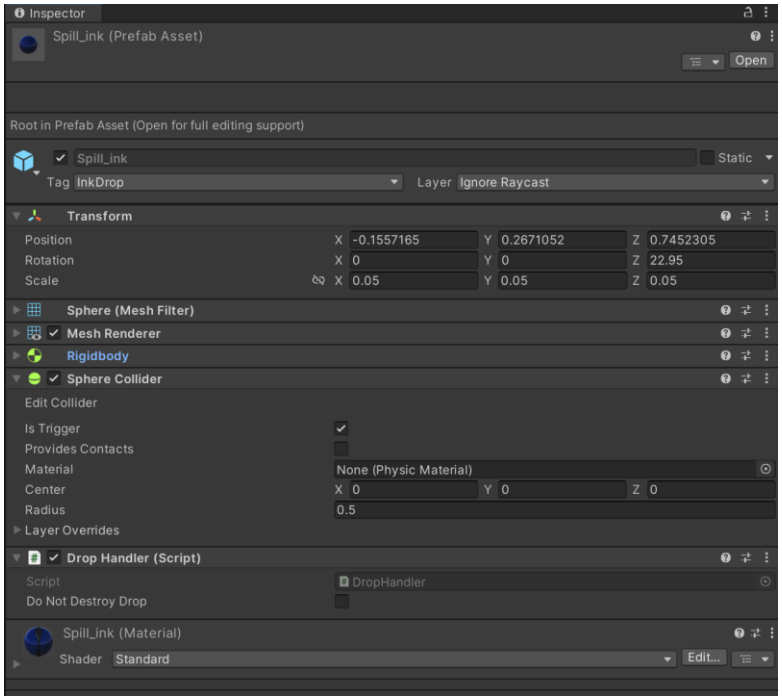
1. Παρακολουθεί το **LiquidVolume** του άδειου δοχείου.
2. Αν βρεθεί μελάνι που αγγίζει το layer του νερού, βάφει το νερό και καθαρίζει τις σταγόνες από μελάνι.
3. Αναμιγνύει νερό και μελάνι αν δεν υπάρχει λάδι, μεταφέροντας την ποσότητα του μελανιού και αλλάζοντας το χρώμα του νερού.
4. Αν υπάρχει υπερχείλιση, ($.level > 1.0$), «χύνεται» το υγρό με τη μικρότερη πυκνότητα, με οπτικά spill drops.
5. Επίσης εδώ βρίσκεται η μέθοδος **ResetWaterColor()** η οποία επαναφέρει το χρώμα του νερού όταν αδειάζει το δοχείο.

4.6.8 DropHandler.cs

Η κλάση **DropHandler.cs** είναι υπεύθυνη για τη **διάρκεια ζωής** και τη **συμπεριφορά** κάθε μεμονωμένης σταγόνας υγρού που δημιουργείται στο παιχνίδι, είτε πρόκειται για νερό, μελάνι ή σταγόνες υπερχείλισης. Αν είναι μικρή, αλληλεπιδρά άμεσα με την **WaterColliderFollow** (αναλύεται

στη συνέχεια), η οποία μπορεί να αλλάξει τη συμπεριφορά της σταγόνας ώστε να την «κρατήσει» στην επιφάνεια του υγρού αντί να την καταστρέψει.

Στο Unity Inspector, είναι προσαρτημένη στα **spillDrop prefabs** και ενεργοποιείται αυτόματα κάθε φορά που η **BeakerPourer** δημιουργεί νέες σταγόνες



4.95 DropHandler Inspector View

```
public class DropHandler : MonoBehaviour
{
    1 reference
    public bool doNotDestroyDrop;

    // Start is called before the first frame update
    0 references
    void Start()
    {
        StartCoroutine(DestroySpill());
    }

    1 reference
    IEnumerator DestroySpill()
    {
        yield return new WaitForSeconds(4f);

        if (!doNotDestroyDrop)
        {
            Destroy(gameObject);
        }
    }
}
```

4.96 DropHandler.cs

doNotDestroyDrop: ελέγχει αν η σταγόνα πρέπει να παραμείνει στη σκηνή ή να καταστραφεί αυτόματα.

4.6.8.1 Μέθοδος Start()

Η μέθοδος **Start()** καλείται αυτόματα όταν η σταγόνα δημιουργείται. Αμέσως εκκινεί την Coroutine **DestroySpill()**, η οποία ελέγχει αν η σταγόνα πρέπει να καταστραφεί ή να παραμείνει ενεργή στη σκηνή.

4.6.8.2 Coroutine DestroySpill()

- Η μέθοδος περιμένει **4 δευτερόλεπτα** πριν εκτελέσει οποιαδήποτε ενέργεια.
- Αν η μεταβλητή **doNotDestroyDrop** είναι **false**, η σταγόνα **καταστρέφεται αυτόματα** μέσω Destroy(gameObject).
- Αν όμως είναι **true**, η σταγόνα **παραμένει ενεργή** και μπορεί να «επιπλέει» ή να χρησιμοποιηθεί για άλλες διεργασίες, όπως η ανάμειξη μελανιού και νερού.

4.6.9 WaterColliderFollow.cs

Η κλάση **WaterColliderFollow** ελέγχει τη θέση ενός **collider** που τοποθετείται ακριβώς πάνω από την επιφάνεια του υγρού, ώστε να ανιχνεύει σωστά επαφές σταγόνων, μελανιού και χειρονομιών. Συνεργάζεται με το **Liquid Volume Pro** για να υπολογίζει το πραγματικό ύψος της επιφάνειας του υγρού και με την **EmptyBeaker** για να χειρίζεται την ανάμειξη μελανιού όταν αυτό απαιτείται.

4.6.9.1 Μεταβλητές

```
public LiquidVolume liquidVolume;
2 references
public Transform colliderTransform; // The collider to move with water
1 reference
public Vector3 offset;
2 references
public EmptyBeaker myBeaker;
```

4.97 WaterColliderFollow.cs 1

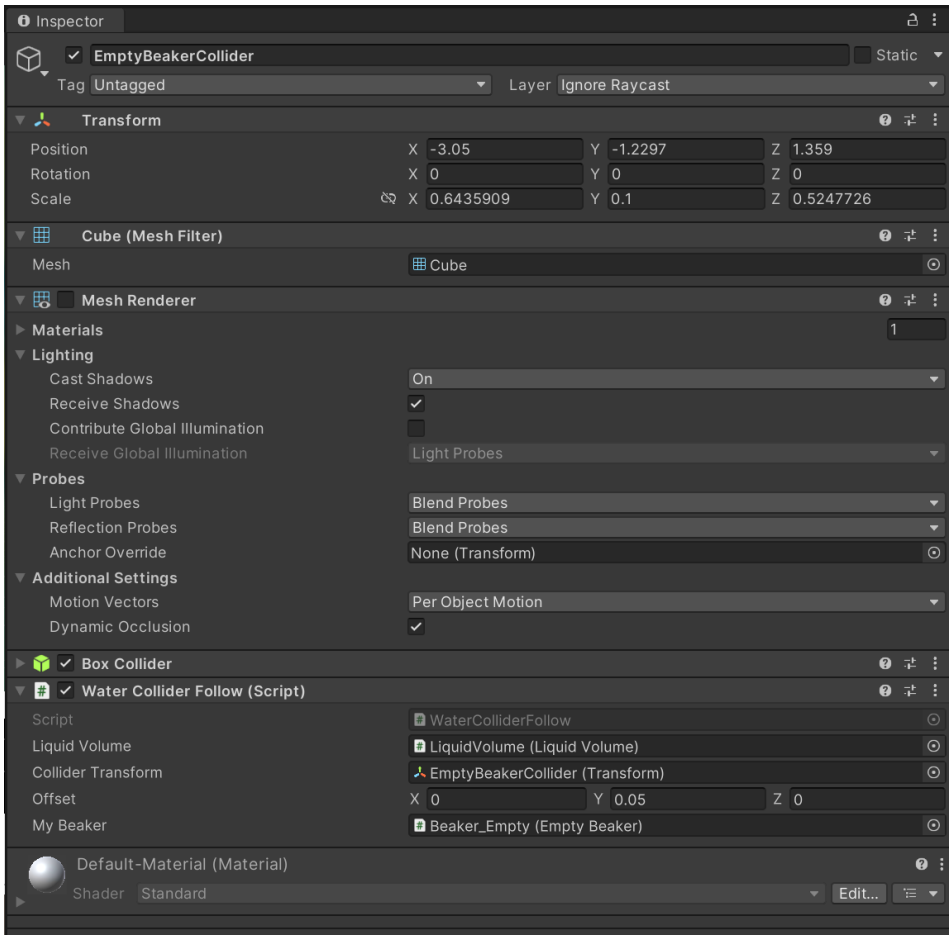
liquidVolume: αναφορά στο **Liquid Volume Pro** για να γνωρίζουμε τη στάθμη του υγρού.

colliderTransform: ο collider που «ακολουθεί» την επιφάνεια του υγρού.

Offset: επιτρέπει μικρή κατακόρυφη διόρθωση, ώστε να τοποθετούμε τον collider ακριβώς πάνω από το υγρό.

myBeaker: αναφορά στην **EmptyBeaker**, ώστε να καλούνται μέθοδοι όπως η

MixInkWithWater().



4.98 WaterColliderFollow Inspector View

4.6.9.2 Μέθοδος Update()

```
void Update()
{
    if (liquidVolume == null || colliderTransform == null) return;

    // 0 to 1 normalized liquid level (based on fill)
    float fillLevel = liquidVolume.level;

    // Compute world Y position of liquid surface
    float containerHeight = liquidVolume.transform.localScale.y;
    float localY = (fillLevel - 0.5f) * containerHeight;
    Vector3 worldPos = liquidVolume.transform.position;
    worldPos.y += localY;

    colliderTransform.position = worldPos+ offset;
}
```

4.99 WaterColliderFollow.cs 1

Η μέθοδος **Update()** εκτελείται σε κάθε frame και συγχρονίζει τη θέση του **collider** με την πραγματική στάθμη του υγρού:

- **Υπολογισμός στάθμης**: παίρνει την τρέχουσα αναλογία πλήρωσης από το **liquidVolume.level** (0 έως 1).

- **Υπολογισμός θέσης**: χρησιμοποιεί το ύψος του δοχείου (**containerHeight**) και τη στάθμη για να βρει την ακριβή τοποθεσία της επιφάνειας.
- **Μετακίνηση collider**: ενημερώνει τη θέση του collider ώστε να ευθυγραμμίζεται με το πάνω μέρος του υγρού.

Αυτό είναι κρίσιμο για να εξασφαλιστεί ότι οι σταγόνες ανιχνεύονται τη σωστή στιγμή όταν πέφτουν στο υγρό.

4.6.9.3 Μέθοδος OnTriggerEnter()

```
private void OnTriggerEnter(Collider other)
{
    if (other.transform.CompareTag("Drops"))
    {
        Destroy(other.gameObject);
    }
    else if(other.transform.CompareTag("InkDrop") && liquidVolume.liquidLayers[1].amount>0)
    {
        //Debug.LogError("here");
        other.transform.GetComponent<Rigidbody>().isKinematic = true;
        other.transform.GetComponent<DropHandler>().doNotDestroyDrop = true;
    }
    else if(other.transform.CompareTag("Finger")
    && GestureDetection.instance.currentGesture == GestureDetection.HandGesture.Point
    && myBeaker.hasInkDrop) // add finger gesture as well
    {
        myBeaker.MixInkWithWater();
    }
}
```

4.100 WaterColliderFollow.cs 2

Η μέθοδος **OnTriggerEnter()** χειρίζεται τις επαφές μεταξύ του collider της επιφάνειας και άλλων αντικειμένων:

1. Σταγόνες νερού ("Drops")

Αν το αντικείμενο έχει tag **"Drops"**, η σταγόνα καταστρέφεται άμεσα, καθώς θεωρείται ότι απορροφήθηκε.

2. Σταγόνες μελανιού ("InkDrop")

Αν η σταγόνα μελανιού πέσει πάνω από το στρώμα λαδιού (**liquidLayers[1] > 0**), το **Rigidbody** γίνεται **kinematic**, σταματώντας την κίνηση.

Στη συνέχεια, η ιδιότητα **doNotDestroyDrop** από την **DropHandler** ενεργοποιείται, ώστε η σταγόνα να «επιπλέει» αντί να βυθιστεί ή να εξαφανιστεί.

3. Επαφή με δάχτυλο ("Finger")

Όταν το χέρι του χρήστη ανιχνευτεί στην επιφάνεια σε **InkDrop**, και η τρέχουσα χειρονομία είναι **Point** μέσω του **GestureDetection**, η μέθοδος **MixInkWithWater()** από την **EmptyBeaker** καλείται, αναμειγνύοντας το μελάνι με το νερό.

Πότε καλείται η OnTriggerEnter()

Η μέθοδος **OnTriggerEnter(Collider other)** είναι ένα **Unity event** που καλείται **αυτόματα** όταν ένα **Collider** με την ιδιότητα **“Is Trigger”** ενεργοποιημένη, συναντήσει ένα άλλο **Collider** μέσα στη σκηνή.

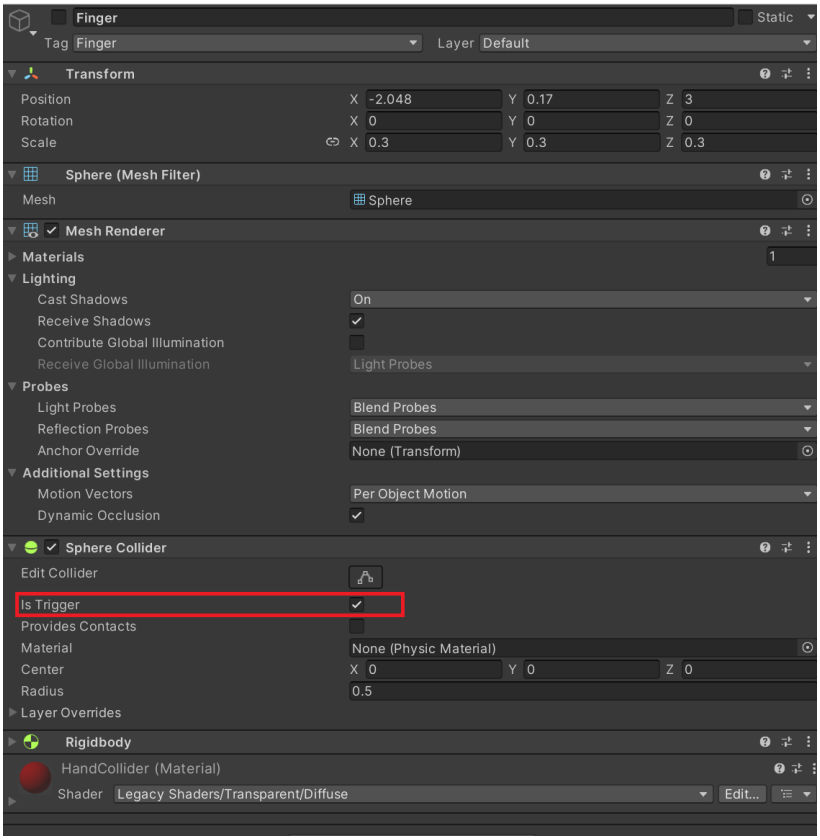
Για να λειτουργήσει σωστά:

- Ο collider του **WaterColliderFollow** πρέπει να έχει ενεργοποιημένη την επιλογή **Is Trigger**.
- Το αντικείμενο που εισέρχεται πρέπει να έχει **Collider** και, συνήθως, ένα **Rigidbody** για να ανιχνευθεί η φυσική επαφή.

Στο δικό μας παιχνίδι:

- Ο **collider** που ακολουθεί τη στάθμη του υγρού λειτουργεί ως «αισθητήρας».
- Όταν μια **σταγόνα νερού** ή **μελανιού** πέσει πάνω του, ή όταν το **δάχτυλο του χρήστη** το αγγίξει (μέσω hand tracking), η Unity καλεί **αυτόματα** την **OnTriggerEnter()**.
- Από εκεί, το script αποφασίζει τι να κάνει, π.χ. να καταστρέψει τη σταγόνα, να την κάνει να επιπλέει ή να ξεκινήσει την ανάμειξη.

Συνοπτικά, **OnTriggerEnter()** ενεργοποιείται τη στιγμή που δύο colliders «τέμνονται», αλλά, επειδή το **Is Trigger** είναι ενεργό, η Unity δεν εμποδίζει την κίνηση των αντικειμένων — απλά αναγνωρίζει την επαφή και μας δίνει τη δυνατότητα να αντιδράσουμε μέσω κώδικα. [18]



4.101 IsTrigger Gesture Collider Setting

4.7 Υλοποίηση Περιβάλλοντος Χρήστη (UI)

Η διεπαφή χρήστη (User Interface – **UI**) της εφαρμογής έχει σχεδιαστεί ώστε να υποστηρίζει τη ροή του πειράματος με απλό και κατανοητό τρόπο, παρέχοντας στους μαθητές **οπτική καθοδήγηση** και **αλληλεπιδραστικά εργαλεία**.

Οι μαθητές καθοδηγούνται βήμα προς βήμα μέσα από το πείραμα, χρησιμοποιώντας διαδραστικά **κουμπιά (Buttons)**, επεξηγηματικές **ετικέτες (Labels)** και ενημερωτικά **παράθυρα (Info Panels)**.Το UI αναπτύχθηκε χρησιμοποιώντας το **Unity UI System** σε συνδυασμό με το **TextMeshPro**. Οι περισσότερες λειτουργίες των κουμπιών συνδέονται άμεσα με την **ChemistryEquipmentManager**, η οποία ελέγχει τη συμπεριφορά του πειράματος.

Για όλα τα κείμενα της διεπαφής χρησιμοποιείται το **TextMeshPro (TMP)**, καθώς προσφέρει υψηλή ευκρίνεια, σωστή υποστήριξη ελληνικών χαρακτήρων και δυνατότητες αυτόματης προσαρμογής (**Auto Size**).

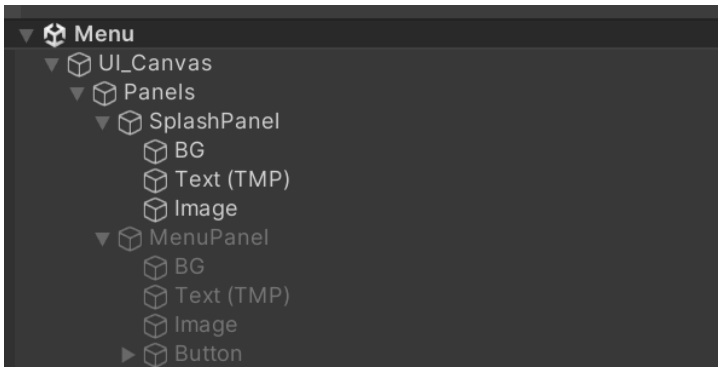
4.7.1 Οθόνη Splash και Μενού

Η εφαρμογή ξεκινά με μία **Splash Screen**, η οποία εμφανίζει το λογότυπο και τον τίτλο του έργου. Αυτή η οθόνη υλοποιείται μέσω της κλάσης **SplashPanel**, η οποία ελέγχει τη διάρκεια προβολής της και, όταν ολοκληρωθεί, μεταβαίνει αυτόματα στην οθόνη μενού.

4.102 Splash Screen

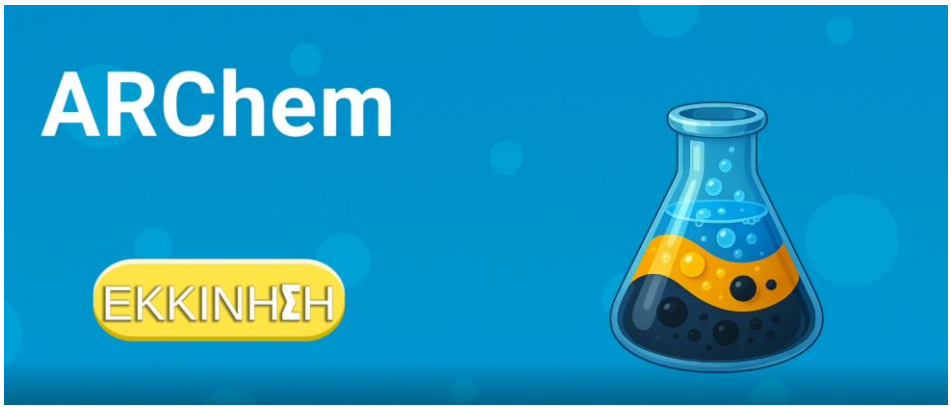
```
public class SplashPanel : MonoBehaviour
{
    1 reference
    public UnityEvent eventToCall;
    1 reference
    public float Delay;
    // Start is called before the first frame update
    0 references
    void Start()
    {
        Invoke("DelayCallEvent", Delay);
    }

    0 references
    void DelayCallEvent()
    {
        eventToCall?.Invoke();
    }

}
```

4.103 SplashPanel.cs



4.104 Menu Scene Hierarchy

Στη συνέχεια, ο χρήστης οδηγείται στο **κύριο μενού** (**MenuPanel.cs**), το οποίο προσφέρει την επιλογή εκκίνησης του πειράματος:



4.105 Start Menu

```
public class MenuPanel : MonoBehaviour
{
0 references
public void GoToScienceProject()
    {
        SceneManager.LoadScene(1);
    }
}
```

4.106 MenuPanel.cs

- Όταν ο χρήστης πατήσει το κουμπί, καλείται η **GoToScienceProject()**.
- Η μέθοδος χρησιμοποιεί τη **SceneManager.LoadScene(1)**, φορτώνοντας τη σκηνή με index **1**, δηλαδή τη σκηνή **ScienceProject** όπου διεξάγεται το παιχνίδι.

4.107 Main Scene View

4.7.2 Δομή του UI

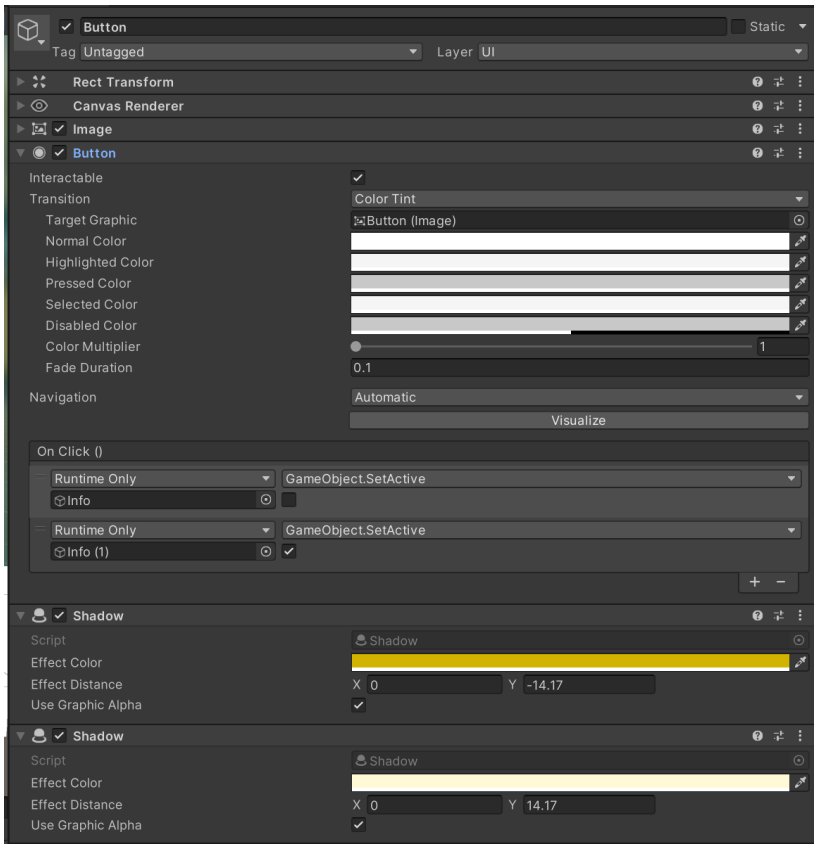
Στην κύρια σκηνή **ScienceProject**, η διεπαφή αποτελείται από επιμέρους **Canvas** που αντιστοιχούν σε διαφορετικές ενέργειες του πειράματος. Κάθε Canvas περιέχει κουμπιά και κείμενα που καθοδηγούν τον μαθητή. Τα βασικά Canvases είναι:

- **Canvas_Water**: Κουμπί για γέμισμα του δοχείου νερού.
- **Canvas_EmptyBeaker**: Κουμπί για άδειασμα του «κενού» δοχείου.
- **Canvas_Oil**: Κουμπί για γέμισμα του δοχείου λαδιού.
- **Canvas_Pipette**: Κουμπί για γέμισμα του σταγονόμετρου με μελάνι.

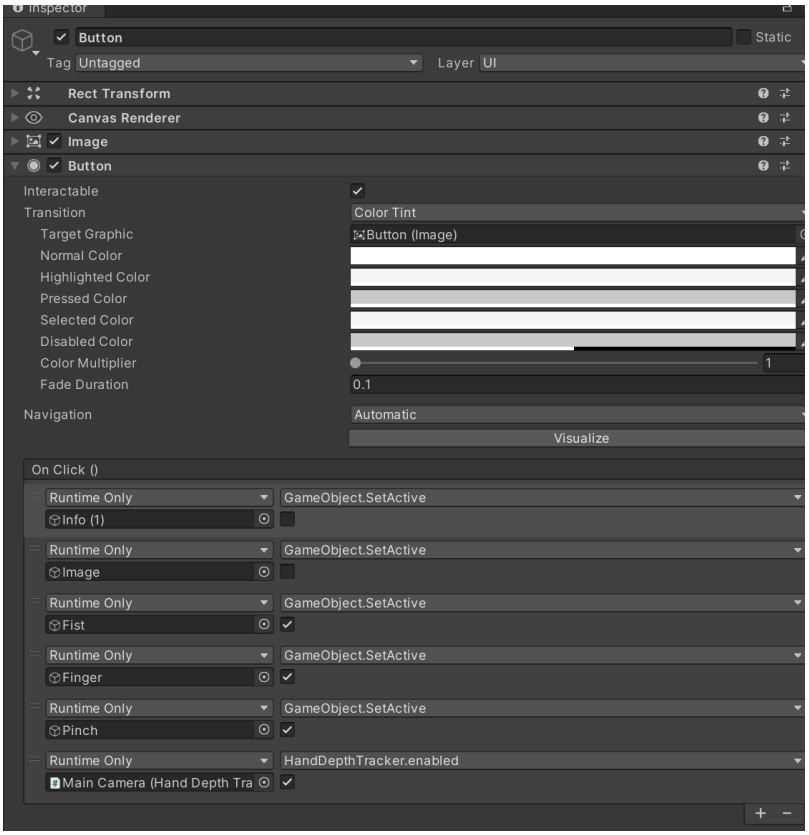
Κάθε **Canvas** περιλαμβάνει:

Δύο κουμπιά (Buttons):

1. **Info Button**: Προχωράει στο επόμενο **Info Panel,** με εξαίρεση το τελευταίο που ενεργοποιεί τα κύρια component της σκηνής (Fist, Finger, Pinch, HandDepthTracker).



4.108 Info Button Inspector View

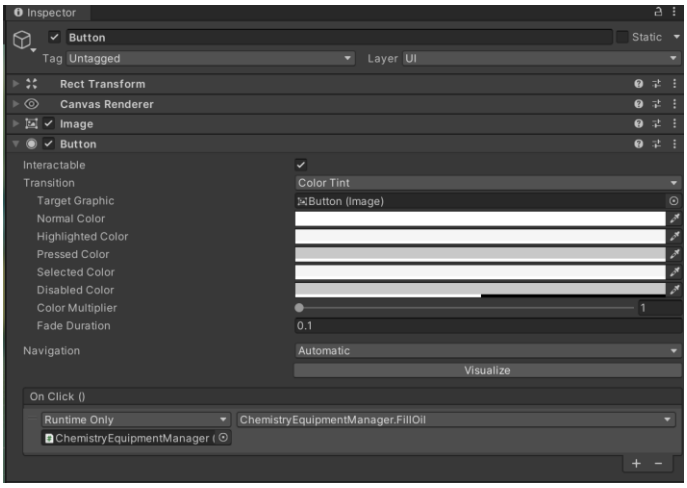


4.109 Last Info Button Inspector View

2. **Action Button**: Εκτελεί την κύρια λειτουργία (π.χ. γέμισμα δοχείου, άδειασμα, γέμισμα λαδιού).

Κάθε **Action Button** στο UI είναι συνδεδεμένο με τον **ChemistryEquipmentManager** μέσω του πεδίου **OnClick()** στο **Inspector**. Αυτό επιτρέπει την άμεση ενεργοποίηση της λογικής του πειράματος:

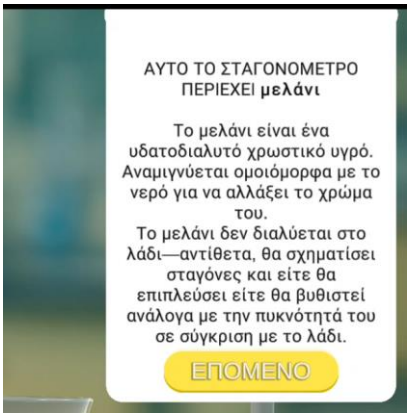
- Το κουμπί στο **Canvas_Water** καλεί τη μέθοδο **FillWater()**, η οποία γεμίζει το δοχείο νερού.
- Το κουμπί στο **Canvas_EmptyBeaker** καλεί τη μέθοδο **EmptyBeaker()**, που αδειάζει το κενό δοχείο.
- Το κουμπί στο **Canvas_Oil** καλεί τη μέθοδο **FillOil()**, η οποία γεμίζει το δοχείο λαδιού.
- Το κουμπί στο **Canvas_Pipette** καλεί τη μέθοδο **FillInk()**, η οποία γεμίζει το σταγονόμετρο με το μελάνι.



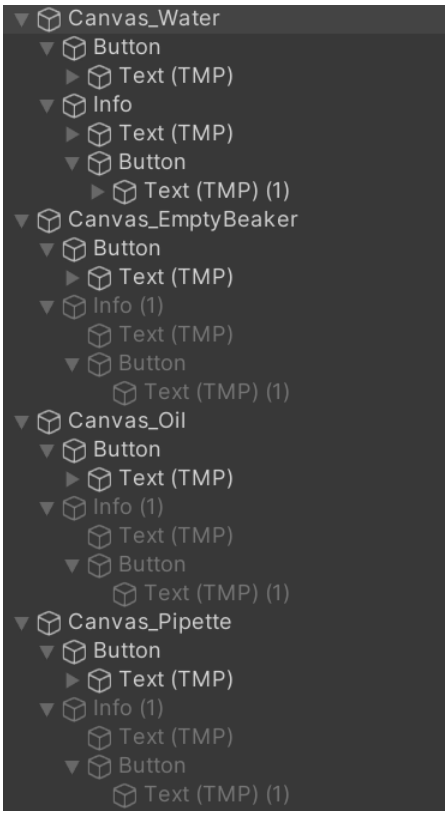
4.110 Action Button Inspector View

Δύο TextMeshPro Labels:

- Μία ετικέτα πάνω στο **Action Button** για την ένδειξη της ενέργειας (πχ. «ΓΕΜΙΣΕ ΛΑΔΙ»).
- Μία ετικέτα μέσα στο **Info Panel** που παρέχει οδηγίες στο χρήστη.



4.111 Ingame Labels



4.112 Canvases Hierarchy

4.7.3 Ροή Χρήστη

Η λογική της διεπαφής ακολουθεί μια **σειριακή ροή**:

1. Ο μαθητής ξεκινάει από το πρώτο **Info Panel** (νερό).
2. Διαβάζει τις οδηγίες και πατάει το κουμπί «ΕΠΟΜΕΝΟ»
3. Το τρέχον panel απενεργοποιείται και ενεργοποιείται το επόμενο.
4. Όταν φτάσει στο τελευταίο panel, η διεπαφή ενεργοποιεί τις **κύριες χειρονομίες** και τα **συστατικά αλληλεπίδρασης**.
5. Στη συνέχεια, ο μαθητής μπορεί να χρησιμοποιήσει τα **Action Buttons** για να εκτελέσει ενέργειες, όπως γέμισμα ή άδειασμα δοχείων, καθώς και να δοκιμάσει διάφορες μίξεις χρησιμοποιώντας χειρονομίες.

4.8 Επίλογος

Στο παρόν κεφάλαιο περιγράφηκε αναλυτικά η **διαδικασία ανάπτυξης** της εφαρμογής. Ξεκινώντας από τη δημιουργία του έργου στο **Unity 2022.3.5f1**, έγινε η ρύθμιση του περιβάλλοντος για **Android** (Build/Player Settings, XR Plug-in Management) και η εγκατάσταση των απαιτούμενων πακέτων (**AR Foundation/ARCore**, UI, LiquidVolume Pro). Διαμορφώθηκαν οι σκηνές **Menu** και **ScienceProject**, ενώ υλοποιήθηκαν οι βοηθητικές λειτουργίες για **κάμερα** και **δικαιώματα** (Camera Fit/Permissions), ώστε η εφαρμογή να εκκινεί με σταθερές εργονομικές παραμέτρους.

Ακολούθησε η **ενσωμάτωση του MediaPipe Hands (homuler)** για on-device ανίχνευση των **21 landmarks** και η σύνδεσή τους με τη λογική **GestureDetection** και **HandDepthTracker**, που χαρτογραφούν τις χειρονομίες σε ενέργειες. Το μοντέλο αλληλεπίδρασης υλοποιήθηκε με τις κλάσεις **VirtualHand** και **Interactable**, επιτρέποντας φυσικούς χειρισμούς αντικειμένων στη σκηνή. Για την απεικόνιση και συμπεριφορά των υγρών αξιοποιήθηκε το **Liquid Volume Pro**, με **έτοιμα prefabs** εργαστηριακού εξοπλισμού και παραμετροποίηση μέσω **C# API** (στάθμη, layering/μίξη), ενώ η ενορχήστρωση του πειράματος έγινε από τις σχετικές κλάσεις χημικού εξοπλισμού.

Τέλος, υλοποιήθηκε η **διεπαφή χρήστη (UI)** με οθόνες **Splash/Μενού**, οδηγίες και ροή χρήστη που υποστηρίζει το σενάριο του πειράματος (νερό–λάδι–μελάνι). Το αποτέλεσμα είναι μια **λειτουργική, επεκτάσιμη και συνεκτική** εφαρμογή AR. Στο επόμενο κεφάλαιο παρουσιάζονται τα συμπεράσματα, μαζί με προτάσεις για μελλοντικές βελτιώσεις.

Κεφάλαιο 5ο: Συμπεράσματα και Μελλοντικές Βελτιώσεις

5.1 Συμπεράσματα

Η εργασία έδειξε ότι μια εφαρμογή **AR** μπορεί να μετατρέψει ένα σχολικό πείραμα Χημείας σε **βιωματική εμπειρία**: ο μαθητής αλληλεπιδρά με χειρονομίες και βλέπει άμεσα την επίδρασή τους σε ρεαλιστικά υγρά (γέμισμα, άδειασμα, ανάμιξη). Ο τεχνολογικός συνδυασμός **Unity / AR Foundation / ARCore**, **MediaPipe Hands (homuler)** και **Liquid Volume Pro (LVP)** παρείχε αξιόπιστη λειτουργία σε Android, φυσική αίσθηση χειρισμού χωρίς πρόσθετο εξοπλισμό και ταχύτερη ανάπτυξη χάρη στα έτοιμα prefabs και το απλό C# API του LVP. Η αρχιτεκτονική που υλοποιήθηκε είναι καθαρή και επεκτάσιμη, ώστε να υποστηρίζει νέα πειράματα με την ίδια λογική.

Παράλληλα αναδείχθηκαν πρακτικοί περιορισμοί (μεταβλητός φωτισμός, απόσταση/γωνία κάμερας, απόδοση σε πιο αδύναμες συσκευές). Αυτοί δεν αναιρούν το αποτέλεσμα, αλλά κατευθύνουν συγκεκριμένες βελτιώσεις για ακόμη πιο ομαλή εμπειρία.

5.2 Μελλοντικές Βελτιώσεις

Για να βελτιωθεί ακόμη περισσότερο η εμπειρία, προτείνονται οι παρακάτω στοχευμένες παρεμβάσεις που ενισχύουν την ευχρηστία, τη σταθερότητα σε διαφορετικές συσκευές και την εύκολη επέκταση σε νέα πειράματα, χωρίς να αλλάξει ο βασικός κορμός της εφαρμογής.

- **Ρύθμιση “pinch” (fine tuning ευαισθησίας):** Fine-tuning της ευαισθησίας με προσαρμογή στο μέγεθος/βάθος του χεριού και μικρή καθυστέρηση επιβεβαίωσης για αποφυγή «τρεμοπαίγματος».
- **Γωνία περιστροφής beaker (fine tuning):** πιο ομαλή και «έξυπνη» κλίση όταν το γεμάτο δοχείο πλησιάζει το άδειο σε μικρή απόσταση ύψους, με διορθώσεις και καλύτερη ευθυγράμμιση του στομίου προς τον στόχο.
- **Νέα πειράματα με την ίδια λογική:** π.χ. δείκτες οξέων–βάσεων, διάχυση, στήλες πυκνότητας, με απλή αλλαγή ιδιοτήτων υγρών και κανόνων.
- **Σύντομο onboarding:** 2–3 οθόνες με βασικές χειρονομίες/βήματα, ώστε ο χρήστης να ξεκινά γρήγορα και με λιγότερα λάθη.
- **Προσαρμογή απόδοσης:** αυτόματη μείωση εσωτερικής ανάλυσης/βαριών εφέ όταν πέφτουν τα καρέ, για σταθερή εμπειρία σε πιο αδύναμες συσκευές.

Βιβλιογραφία

[1] «Milgram, Paul & Kishino, Fumio. (1994). A Taxonomy of Mixed Reality Visual Displays. IEICE Trans. Information Systems. vol. E77-D, no. 12. 1321-1329.».

[2] «Milgram, Paul & Takemura, Haruo & Utsumi, Akira & Kishino, Fumio. (1994). Augmented reality: A class of displays on the reality-virtuality continuum. Telemanipulator and Telepresence Technologies. 2351. 10.1117/12.197321.,» τόμ. 2351.

[3] «Ronald T. Azuma; A Survey of Augmented Reality. Presence: Teleoperators and Virtual Environments 1997; 6 (4): 355–385. doi: https://doi.org/10.1162/pres.1997.6.4.355».

[4] «Bacca-Acosta, Jorge & Baldiris, Silvia & Fabregat, Ramón & Graf, Sabine & Kinshuk, Dr. (2014). Augmented Reality Trends in Education: A Systematic Review of Research and Applications. Educational Technology and Society. 17. 133-149.».

[5] «Al-Saedi, Ahmed Kadem Hamed & Al-Asadi, Abbas. (2019). Survey of Hand Gesture Recognition Systems. Journal of Physics: Conference Series. 1294. 042003. 10.1088/1742-6596/1294/4/042003.».

[6] «Unity Technologies, "AR Foundation documentation," Unity Manual, 2023.,» [Ηλεκτρονικό]. Available: https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@6.1/manual/index.html.

[7] «Google Developers, "Get started with ARCore in Unity," ARCore Developer Guide, 2023.,» [Ηλεκτρονικό]. Available: https://developers.google.com/ar/develop.

[8] «V. Bazarevsky et al., "MediaPipe Hands: On-device Real-time Hand Tracking," arXiv preprint, arXiv:2006.10214, 2020.,» [Ηλεκτρονικό]. Available: https://arxiv.org/abs/2006.10214.

[9] [Ηλεκτρονικό]. Available: https://github.com/homuler/MediaPipeUnityPlugin.

[10] «TensorFlow Blog, "3D Hand Pose Estimation in the Browser using TensorFlow.js," 2021.,» [Ηλεκτρονικό]. Available: https://blog.tensorflow.org/2021/11/3D-handpose.html.

[11] «OpenCV Documentation.,» [Ηλεκτρονικό]. Available: https://opencv.org/.

[12] «Ultraleap, "Hand Tracking SDK," 2023.,» [Ηλεκτρονικό]. Available: https://docs.ultraleap.com/hand-tracking/.

[13] [Ηλεκτρονικό]. Available: https://www.khronos.org/openxr.

[14] «ManoMotion AB, "ManoMotion Unity SDK Documentation," 2024.,» [Ηλεκτρονικό]. Available: https://www.manomotion.com/products/unity/.

[15] «Overly App, "How to create 3D content in Blender for mobile augmented reality projects," Overly Blog, 2022.,» [Ηλεκτρονικό]. Available: https://overlyapp.com/blog/how-to-create-3d-content-in-blender-for-mobile-augmented-reality-projects/.

[16] [Ηλεκτρονικό]. Available: https://assetstore.unity.com/packages/vfx/shaders/liquid-volume-pro-2-129967.

[17] [Ηλεκτρονικό]. Available: https://docs.unity3d.com/ScriptReference/Physics.Raycast.html.

[18] [Ηλεκτρονικό]. Available: https://docs.unity3d.com/ScriptReference/Collider.OnTriggerEnter.html.