ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

"Η εξέλιξη των τεχνολογιών εικονικής πραγματικότητας. Ανάπτυξη σχετικής εφαρμογής"

ΔΗΜΗΤΡΗΣ ΠΑΠΑΖΑΦΕΙΡΗΣ
ΑΜ 2738

Επιβλέπων καθηγητής: Τζήμας Ιωάννης

Πάτρα – Αύγουστος 2021

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή

Πάτρα, Ημερομηνία

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1. Ονοματεπώνυμο, Υπογραφή
2. Ονοματεπώνυμο, Υπογραφή
3. Ονοματεπώνυμο, Υπογραφή

Ευχαριστίες

Ευχαριστώ πολύ τον Andrew David Jahchan για την συνεχή του υποστήριξη και τις σημαντικές του συμβουλές και τον Tronz για την παροχή του στα μουσικά κομμάτια.

Περίληψη

Ο σκοπός αυτής της πτυχιακής εργασίας ήταν η ανάλυση και επεξήγηση των τρόπων υλοποίησης των εφαρμογών εικονικής πραγματικότητας καθώς και η ανάπτυξη μιας εφαρμογής η οποία χρησιμοποιεί κάποιο τρόπο εικονικής απεικόνισης. Η ιδέα προήλθε από το γεγονός ότι η εικονική πραγματικότητα είναι στην άνθιση του και μολονοτι είναι ένας κλάδος που έχει ερευνηθεί απο παλία, η ραγδαια ανάπτυξη στην ποικιλία και την προσβασιμότητα σε τετοιες τεχνολογίες έχει γινει την τελευταια δεκαετία. Παρ'οτι οι εφαρμογες αυτες χρησιμοποιούνται κατα κυριο λογο για ψυχαγωγία πια, ειχαν και εχουν βοηθησει και στην εκπαίδευση. Το βασικό τους χαρακτηριστικό βρίσκεται στο γεγονος οτι ενα άτομο μπορει να βρεθει σε οποιοδηποτε διαθεσιμο εικονικό περιβαλλον και να διαδρασει με αυτο εως ένα βαθμο, ο οποιος βελτιωνεται συνεχώς, για να φτασει στο σημειο που η εικονικη δεν θα διαφερει απο την δική μας πραγματικότητα. Για την επίτευξη της δημιουγιας της εφαρμογης, χρησιμοποιήθηκε η μηχανή ανάπτυξης Unity με πολλά απο τα περιεχόμενα εργαλεία που παρέχει, καθως είναι πιο προσιτη και εύκολη για βασική υλοποιηση. Ακόμη, γίνεται μια συγκριτική μεταξύ των δυο επικρατέστερων μηχανών ανάπτυξης, Unity και Unreal για να σημειωθουν οι μικρές διαφορές που εχουν και μπορούν να οδηγήσουν στην καταλληλότερη επιλογή για την υλοποιηση, ανάλογα την περιπτωση.

1. Εικονική Πραγματικότητα - Ιστορική Αναδρομή

Εικονική Πραγματικότητα (Virtual Reality): είναι μια διαδραστική εμπειρία δημιουργημένη από τον υπολογιστή που λαμβάνει χώρα μέσα σε ένα προσομοιωμένο περιβάλλον. Ενσωματώνει κυρίως ακουστική και οπτική ανατροφοδότηση, αλλά μπορεί επίσης να επιτρέψει και άλλους τύπους αισθητηριακών ανατροφοδοτήσεων όπως η απτική. Αυτό το εντυπωσιακό περιβάλλον μπορεί να είναι παρόμοιο με τον πραγματικό κόσμο ή μπορεί να είναι φανταστικό. Η έννοια της εικονικής πραγματικότητας υπάρχει πολλά χρόνια πίσω στην ιστορία της τεχνολογίας αλλά τώρα έχουν καταφέρει να γίνουν τα πιο δραματικά βήματα προς την εξέλιξή της.

1.1. 19ος αιώνας

Μια από τις πρώτες προσπάθειες προσομοίωσης ενός γεγονότος γινόταν από τους διάφορους ζωγράφους με την τεχνική της πανοραμικής ζωγραφικής. Έτσι απεικόνιζαν διάφορες σκηνές σε 360 μοίρες με σκοπό να κάνουν το άτομο που βλέπει τον πίνακα να νιώθει πιο πολύ ότι είναι μέσα σε αυτόν.

1.1.1. 1838

Ο Charles Wheatstone εφεύρει το γνωστό στο κόσμο ViewMaster. Η φιλοσοφία του βασιζόταν στην προβολή δύο στερεοσκοπικών 2D εικόνων μέσα από το στερεοσκόπιο και ο εγκέφαλός μας το μετέφραζε αυτό σε μια 3D εικόνα. Κατά κύριο λόγο χρησιμοποιήθηκε για ψηφιακό τουρισμό και η τεχνική αυτή χρησιμοποιείται ακόμη και σήμερα από το Google Cardboard.

1.2. 20ος αιώνας

1.2.1. 1929 - Ο πρώτος προσομιωτής πτήσης “Link Trainer”

Ο Edward Link εφηύρε τον Link Trainer ,πατενταρήστηκε το 1931, και ήταν ο πρώτος εμπορικός προσομοιωτής όπου ήταν ηλεκτρομηχανικός. Χρησιμοποιούταν για την εκπαίδευση πιλότων σε συνθήκες πτήσης κάνοντας με μια μικρή συσκευή τις αναταράξεις και τις διαταραχές του αεροπλάνου και με ένα μοχλό χειρίζονταν το πηδάλιο. Ο Αμερικανικός Στρατός είχε αγοράσει έξι στην τιμή των 3500 δολαρίων -αναλογικά με σήμερα 50000 δολάρια- για την εκπαίδευση στρατιωτών στον Β' Παγκόσμιο Πόλεμο. Πιο μετά, η έννοια γυαλιών που επιτρέπουν στο χρήστη να χρησιμοποιεί τις αισθήσεις του σε ένα άλλο κόσμο , άρχισε να εμφανίζεται και στα μυθιστορήματα επιστημονικής φαντασίας που πρακτικά είχαν προβλέψει την τωρινή τεχνολογία VR glasses.

1.2.2. 1957 - Sensorama

Ο σκηνοθέτης Morton Heilig ανέπτυξε το Sensorama (πατενταρισμένο 1962), το οποίο ήταν ένα γραφείο θεάτρου σε στυλ arcade που θα διέγειρε όλες τις αισθήσεις και όχι μόνο την όραση και τον ακοή. Περιείχε στερεοφωνικά ηχεία, στερεοσκοπική οθόνη 3D, ανεμιστήρες, γεννήτριες μυρωδιών και δονούμενη καρέκλα. Έτσι ο χρήστης ένιωθε κάθε τι που γινόταν μέσα στην ταινία όπως ο άνεμος, οι διάφορες μυρωδιές και η κίνηση.

Το Sensorama προοριζόταν να ενσωματώσει πλήρως το άτομο στην ταινία. Δημιούργησε επίσης έξι ταινίες μικρού μήκους για την εφεύρεσή του, όλες τις οποίες πυροβόλησε, παράγει και επεξεργάζεται.

1.2.3. 1960 - Telesphere

Έπειτα ο Heilig κατασκεύασε το Telesphere και ήταν το πρώτο παράδειγμα μιας οθόνης τοποθετημένης στο κεφάλι, αν και δεν είχε κάποιο αλληλεπιδραστικό μέσο με αισθητήρα εντοπισμού κίνησης. Το σετ μικροφώνου-ακουστικού προσέφερε στερεοσκοπική 3D και ευρεία όραση με στερεοφωνικό ήχο.

1.2.4. 1961 - Headsight

Το 1961, δύο μηχανικοί της Philco Corporation (Comeau & Bryan) ανέπτυξαν τον πρώτο πρόδρομο της οθόνης τοποθετημένης στο κεφάλι (HMD) όπως τη γνωρίζουμε σήμερα - το Headsight. Έχει ενσωματωθεί μια οθόνη βίντεο για κάθε μάτι και ένα μαγνητικό σύστημα εντοπισμού κίνησης, το οποίο συνδεόταν με μια κάμερα κλειστού κυκλώματος. Το Headsight δεν αναπτύχθηκε για εφαρμογές εικονικής πραγματικότητας (ο όρος δεν υπήρχε τότε), αλλά για να επιτρέψει την εμβάθυνση στην απομακρυσμένη προβολή επικίνδυνων καταστάσεων για τον στρατό. Οι κινήσεις του κεφαλιού θα μετακινούσαν μια απομακρυσμένη κάμερα, επιτρέποντας στον χρήστη να κοιτάζει φυσικά γύρω του στο περιβάλλον. Η Headsight ήταν το πρώτο βήμα στην εξέλιξη της οθόνης VR τοποθετημένη στο κεφάλι, αλλά δεν είχε την ενσωμάτωση στην παραγωγή υπολογιστών και εικόνων.

1.2.5. 1965 - “Η απόλυτη απεικόνιση”

Ο Ivan Sutherland περιέγραψε την ιδέα της "απόλυτης απεικόνισης" που θα μπορούσε να προσομοιώσει την πραγματικότητα στο σημείο που δεν θα μπορούσε κανείς να πει τη διαφορά από την αληθινή. Η ιδέα του περιλάμβανε:

Έναν εικονικό κόσμο που θα προβάλλεται μέσω ενός HMD και θα φαίνεται ρεαλιστικός μέσω του ενισχυμένου 3D ήχου και της απτικής ανάδρασης.

Το υλικό του υπολογιστή για να δημιουργήσει την εικονικό κόσμο και να τον διατηρεί σε πραγματικό χρόνο.

Η ικανότητα των χρηστών να αλληλοεπιδρούν με αντικείμενα στον εικονικό κόσμο με ρεαλιστικό τρόπο.

Αυτό η ερευνά του θα αποτελέσει βασικό υπόδειγμα για τις έννοιες που περιλαμβάνουν την εικονική πραγματικότητα σήμερα. Η άποψή του ήταν: <<Η απόλυτη απεικόνιση θα ήταν φυσικά ένα δωμάτιο μέσα στο οποίο ο υπολογιστής μπορεί να ελέγξει την ύπαρξη της ύλης.

Μια καρέκλα που εμφανίζεται σε ένα τέτοιο δωμάτιο θα ήταν αρκετά καλή για να καθίσει. Οι χειροπέδες που εμφανίζονται σε ένα τέτοιο δωμάτιο θα ήταν δεσμευτικές και μια σφαίρα που εμφανίζεται σε ένα τέτοιο δωμάτιο θα ήταν θανατηφόρα. Με κατάλληλο προγραμματισμό μια τέτοια προβολή θα μπορούσε κυριολεκτικά να είναι η χώρα των θαυμάτων στην οποία περπάτησε η Αλίκη.>>

1.2.6. 1968 - Το σπαθί του Δαμοκλή

Μετά από τρία χρόνια, ο Ivan Sutherland μαζί με τον μαθητή του Bob Sproull κατασκευάζουνε το πρώτη οθόνη VR/AR κεφαλιού η οποία ήταν συνδεδεμένη με έναν υπολογιστή και όχι με κάμερα. Λόγω της μεγάλη και επιβλητικής, κατακόρυφης κατασκευής της οθόνη και της μη βολικότητας στο απλά να το φορέσει κάποιος, αναγκάστηκαν να το τοποθετήσουν κρεμάμενο στο ταβάνι απ' όπου πήρε και το όνομά του αφού έμοιαζε με σπαθί. Ο χρήστης έπρεπε να δεθεί και αυτός με την κατασκευή και τα αντικείμενα που παρήγαγε ο υπολογιστής ήταν πολύ βασικά δωμάτια και αντικείμενα.

1.2.7. 1969 - "Τεχνητή Πραγματικότητα"

Ο Myron Kruegere ένας καλλιτέχνης ηλεκτρονικής εικονικής πραγματικότητας ανέπτυξε μια σειρά εμπειριών που ονομάστηκε «τεχνητή πραγματικότητα», στην οποία ανέπτυξε περιβάλλοντα που δημιουργημένα από υπολογιστές που ανταποκρίνονταν στους ανθρώπους μέσα σε αυτό.

Τα έργα που ονομάζονταν GLOWFLOW, METAPLAY και PSYCHIC SPACE αποτέλεσαν εξέλιξη στην έρευνά του που τελικά επέτρεψε την ανάπτυξη της τεχνολογίας VIDEOPLACE. Αυτή η τεχνολογία επέτρεψε στους ανθρώπους να επικοινωνούν μεταξύ τους σε ένα ανταποκρινόμενο περιβάλλον που κατασκεύασε ένας υπολογιστής, παρά το γεγονός ότι έμεναν σε απόσταση χιλιομέτρων.

1.2.8. 1970 έως 1980 - Super Cockpit

Σχεδόν την ίδια στιγμή που ο Ivan Sutherland δούλευε στο έργο του, ένας στρατιωτικός μηχανικός που ονομάζεται Thomas Furness ήταν απασχολημένος με την ανάπτυξη ενός φιλόδοξου έργου εξομοιωτή πτήσης το οποίο τελικά θα καταλήξει σε κάτι που ονομάζεται "Super Cockpit".

Η Furness συνέχισε να εργάζεται για το έργο μέσα από τη δεκαετία του '80, με αποτέλεσμα ένα cockpit εκπαίδευσης ικανό να σχεδιάσει υπολογιστικούς 3D χάρτες, υπέρυθρες και ραντάρ εικόνες, καθώς και δεδομένα αεροηλεκτρονικής σε πραγματικό χρόνο στο 3D χώρο.

Σύμφωνα με τον Furness, το κόστος του έργου ανέρχεται σε "εκατοντάδες εκατομμύρια". Όντας μπροστά από την εποχή του, το Super Cockpit επιτρέπει σε έναν ασκούμενο πιλότο να ελέγχει αεροσκάφος χρησιμοποιώντας χειρονομίες, ομιλία και ακόμη και κινήσεις των ματιών.

1.2.9. 1978 - Aspen Movie Map

Αναπτύχθηκε από το MIT το 1978, με ένα χέρι βοήθειας από τη DARPA, το Aspen Movie Map βασικά ήταν μια εικονική πραγματικότητα που μοιάζει στο Google Street View. Αντί για τα βασικά τρισδιάστατα γραφικά που θα μπορούσαν να δημιουργηθούν εκείνη την εποχή, χρησιμοποίησε φωτογραφίες που τραβήχτηκαν από ένα αυτοκίνητο που οδηγούσε στο Άσπεν του Κολοράντο, δίνοντας στον χρήστη ένα διαδραστικό ταξίδι πρώτου προσώπου σε όλη την πόλη.

Το τρέξιμο απαιτεί αρκετές συσκευές αναπαραγωγής Laserdisc, έναν υπολογιστή και μια οθόνη αφής. Ενώ δεν υπήρχε εξάρτημα HMD στο έργο, η καινοτόμος χρήση της δια-δραστικότητας του πρώτου προσώπου του

Aspen Movie Map αντιπροσώπευε μια έξυπνη εξέταση του τρόπου με τον οποίο το VR θα μπορούσε να χρησιμοποιηθεί για τη μεταφορά των ανθρώπων σε άλλα μέρη.

1.2.10. 1987 - Η γέννηση της έννοιας “Εικονική Πραγματικότητα”

Ακόμη και μετά από όλη αυτή την εξέλιξη στην εικονική πραγματικότητα, δεν υπήρχε ακόμα ένας περιεκτικός όρος για την περιγραφή του πεδίου.Όλα αυτά άλλαξαν το 1987, όταν ο Jaron Lanier, ιδρυτής του εργαστηρίου οπτικού προγραμματισμού (VPL), δημιούργησε τον όρο "εικονική πραγματικότητα".

Μέσω της έρευνας της εταιρείας VPL, ο Jaron ανέπτυξε μια σειρά εργαλείων εικονικής πραγματικότητας, όπως το Dataglove (μαζί με τον Tom Zimmerman) και την οθόνη EyePhone. Ήταν η πρώτη εταιρεία που πουλούσε γυαλιά εικονικής πραγματικότητας (eyePhone 1 9.400 δολάρια , EyePhone HRX 49.000 δολάρια) και γάντια (9000 δολάρια) όπου ήταν ένα σημαντικό βήμα για την εξέλιξη των απτικών μέσων που αλληλοεπιδρούν σε ένα εικονικό περιβάλλον.

1.2.11. 1991 - Μηχανές Arcade του Virtuality Group

Αρχίσαμε να βλέπουμε τις συσκευές εικονικής πραγματικότητας στις οποίες είχε πρόσβαση το κοινό, παρόλο που η οικιακή ιδιοκτησία τεχνολογιών εικονικής πραγματικότητας επείχε πάρα πολύ. Το Virtuality Group ξεκίνησε να παράγει μια σειρά μηχανών και παιχνιδιών arcade. Οι παίκτες φορούσαν ένα ζευγάρι γυαλιών VR και θα έπαιζαν σε μηχανές παιχνιδιών με στερεοσκοπικά 3D γραφικά σε πραγματικό χρόνο (λιγότερο από 50ms καθυστέρηση).

Κάποιες μηχανές μάλιστα ήταν και συνδεδεμένες μεταξύ τους μέσω ενός δικτύου για να επιτρέπει το παιχνίδι πολλαπλών παικτών μαζί. Πλέον η έννοια της εικονικής πραγματικότητας προσπαθεί να γίνει και ευρύτερα γνωστή στο απλό κοινό με την ενσωμάτωσή της σε ταινίες επιστημονικής φαντασίας.

1.2.12. 1993 - SEGA VR

Η Sega ανακοίνωσε το Sega VR για την κονσόλα Sega Genesis στο Consumer Electronics Show το 1993. Τα πρωτότυπα γυαλιά είχαν παρακολούθηση κίνησης κεφαλιού, στερεοφωνικό ήχο και οθόνες LCD στην προσωπίδα. Η Sega είχε πλήρη πρόθεση να κυκλοφορήσει το προϊόν σε τιμή περίπου 200 δολάρια τότε, ή περίπου 322 δολάρια τώρα. Ωστόσο, οι τεχνικές δυσκολίες εξέλιξης σήμαιναν ότι η συσκευή θα παρέμενε για πάντα στη φάση του πρωτότυπου παρότι έχει αναπτύξει 4 παιχνίδια για αυτό το προϊόν. Αυτό ήταν μια τεράστια αποτυχία για τη Sega.

1.2.13. 1996 - Nintendo Virtual Boy

Το Nintendo Virtual Boy (αρχικά γνωστό ως VR-32) ήταν μια κονσόλα παιχνιδιών τρισδιάστατης τεχνολογίας που υποσχέθηκε να είναι η πρώτη ποτέ φορητή κονσόλα που θα μπορούσε να εμφανίσει αληθινά τρισδιάστατα γραφικά. Πρώτα κυκλοφόρησε στην Ιαπωνία και τη Βόρεια Αμερική σε τιμή 180 δολάρια, αλλά ήταν μια εμπορική αποτυχία παρά τις πτώσεις τιμών. Οι αναφερθείσες αιτίες αυτής της αποτυχίας ήταν η έλλειψη χρώματος στα γραφικά (παιχνίδια ήταν κόκκινα και μαύρα), υπήρχε έλλειψη υποστήριξης λογισμικού

και ήταν δύσκολο να χρησιμοποιηθεί η κονσόλα σε μια άνετη θέση. Τον επόμενο χρόνο διέκοψαν την παραγωγή και την πώληση.

1.3. 21ος αιώνας

1.3.1. 2000 έως 2010

Τα πρώτα 10 χρόνια του 21ου αιώνα η τεχνολογική ανάπτυξη του VR ήταν σχετικά σε περίοδο ανάπαυσης, χωρίς κάποια σημαντική πρόοδος.

1.3.2. 2012 - Oculus Rift

Η Oculus, με ιδρυτή και δημιουργό τον Palmer Luckey, ξεκίνησε μια εκστρατεία Kickstarter το 2012 για να χρηματοδοτήσει την ανάπτυξη του Rift, αφού ιδρύθηκε ως ανεξάρτητη εταιρεία δύο μήνες πριν. Το έργο αποδείχθηκε επιτυχημένο, αυξάνοντας τα 2,5 εκατομμύρια δολάρια. Τον Μάρτιο του 2014, το Facebook αγόρασε το Oculus για 2 δισεκατομμύρια δολάρια.

Το Rift διαθέτει δύο οθόνες Pentile OLED, ανάλυση 1080 × 1200 ανά μάτι, ρυθμό ανανέωσης 90 Hz και οπτικό πεδίο 110 μοιρών.Η συσκευή διαθέτει επίσης δυνατότητα παρακολούθησης περιστροφής και θέσης και εσωτερικά ακουστικά που παρέχουν 3D εφέ.

1.3.3. 2014 - Google Cardboard

Το Google Cardboard είναι μια πλατφόρμα εικονικής πραγματικότητας (VR) που αναπτύχθηκε από την Google για χρήση μιας βάσης κεφαλιού για ένα smartphone και δημιουργήθηκε από τους David Coz και Damien Henry. Η πλατφόρμα, ονομαζόμενη για τον πτυσσόμενο προβολέα από χαρτόνι, προορίζεται ως σύστημα χαμηλού κόστους για την ενθάρρυνση του ενδιαφέροντος και της ανάπτυξης εφαρμογών VR.

Οι χρήστες μπορούν να δημιουργήσουν το δικό τους προβολέα από απλά, χαμηλού κόστους υλικά, αγοράζοντας έναν από τους πολλούς προκατασκευασμένους προβολείς που χρησιμοποιούν προδιαγραφές από την Google . Τα μέρη που αποτελούν τον προβολέα είναι ένα κομμάτι χαρτόνι κομμένο σε ακριβές σχήμα, φακοί εστιακού μήκους 45 mm, μαγνήτες ή χωρητική ταινία, συνδετήρας αγκίστρου και βρόχου (όπως Velcro), λαστιχένια ζώνη και προαιρετικό κοντινό πεδίο επικοινωνίας (NFC).

Για να χρησιμοποιήσουμε την πλατφόρμα, οι χρήστες εκτελούν εφαρμογές συμβατές με το Cardboard στο τηλέφωνό τους, τοποθετούν το τηλέφωνο στο πίσω μέρος του προβολέα και προβάλλουν περιεχόμενο μέσω των φακών.

1.3.4. 2015 - HTC Vive

Το HTC Vive είναι ένα κράνος εικονικής πραγματικότητας που αναπτύχθηκε από την HTC και τη Valve Corporation. Το κράνος χρησιμοποιεί την τεχνολογία παρακολούθησης "κλίμακας δωματίου", επιτρέποντας στον χρήστη να μετακινείται σε τρισδιάστατο χώρο και να χρησιμοποιεί χειριστήρια χειρός που παρακολουθούνται με κίνηση για να αλληλοεπιδράσουν με το περιβάλλον. Το κράνος Vive έχει ρυθμό

ανανέωσης 90 Hz και οπτικό πεδίο 110 μοιρών. Η συσκευή χρησιμοποιεί δύο πάνελ OLED, ένα ανά μάτι, το καθένα από τα οποία έχει ανάλυση οθόνης 1080 × 1200.

Τα χαρακτηριστικά ασφαλείας περιλαμβάνουν μια κάμερα που βλέπει προς τα εμπρός και επιτρέπει στο χρήστη να παρατηρεί το περιβάλλον του χωρίς να αφαιρεί το κράνος. Το λογισμικό μπορεί επίσης να χρησιμοποιήσει την κάμερα για να εντοπίσει οποιαδήποτε κινούμενα ή στατικά αντικείμενα σε ένα δωμάτιο, αυτή η λειτουργικότητα μπορεί να χρησιμοποιηθεί ως μέρος του συστήματος ασφαλείας "Chaperone", το οποίο θα εμφανίζει αυτόματα έναν εικονικό τοίχο ή μια τροφοδοσία από την κάμερα για την ασφαλή καθοδήγηση των χρηστών από εμπόδια ή πραγματικούς τοίχους.

Μέσα από τα εξωτερικά κελύφη του κράνους είναι δεκάδες αισθητήρες υπέρυθρων που ανιχνεύουν τους παλμούς IR των σταθμών βάσης για να καθορίσουν την τρέχουσα θέση του κράνους σε ένα χώρο. Άλλοι αισθητήρες περιλαμβάνουν αισθητήρα G, γυροσκόπιο και αισθητήρα εγγύτητας. Το 2018 κυκλοφόρησε μια αναβαθμισμένη έκδοση του κράνους , το HTC Vive Pro και το 2019 ανακοινώθηκε η επόμενη έκδοση του, το HTC Vive Pro Eye.

1.3.5. 2016 - Playstation VR

Το PlayStation VR, γνωστό από την κωδική ονομασία Project Morpheus κατά τη διάρκεια της ανάπτυξης, είναι ένα κράνος εικονικής πραγματικότητας που αναπτύχθηκε από την Sony Computer Entertainment, το οποίο κυκλοφόρησε τον Οκτώβριο του 2016. Σχεδιάστηκε για να είναι πλήρως λειτουργικό με την κονσόλα παιχνιδιών PlayStation 4.

Σε ορισμένα παιχνίδια και demo για το VR, ο φορέας που φοράει το κράνος δρα ξεχωριστά από άλλους παίκτες χωρίς το κράνος. Το σύστημα PlayStation VR μπορεί να προβάλει μια εικόνα τόσο στο κράνος PlayStation VR όσο και στην τηλεόραση ταυτόχρονα, με την τηλεόραση να αντικατοπτρίζει την εικόνα που εμφανίζεται στο κράνος ή να εμφανίζει μια ξεχωριστή εικόνα για ανταγωνιστικό ή συνεργατικό παιχνίδι. Το PlayStation VR λειτουργεί είτε με τον κανονικό χειριστήριο DualShock 4 είτε με τα χειριστήρια PlayStation Move.

Το PlayStation VR διαθέτει οθόνη OLED 5,7 ιντσών, με ανάλυση οθόνης 1080p.

Το κράνος έχει επίσης ένα κουτί επεξεργαστή που επιτρέπει την προβολή βίντεο Social Screen στην τηλεόραση, καθώς και την επεξεργασία του εφέ τρισδιάστατου ήχου και χρησιμοποιεί υποδοχή ακουστικών 3,5 χιλιοστών. Το κράνος έχει επίσης LED 9 θέσεων στην επιφάνεια του για την κάμερα PlayStation για να παρακολουθεί την κίνηση του κεφαλιού 360 μοιρών.

1.3.6. Microsoft Hololens

Το Microsoft HoloLens, γνωστό ως Project Baraboo, είναι ένα ζευγάρι smartglasses μικτής πραγματικότητας που αναπτύχθηκε και κατασκευάστηκε από τη Microsoft. Τα HoloLens ήταν ένας από τους πρώτους υπολογιστές που χρησιμοποιούσαν την πλατφόρμα Windows Mixed Reality στο πλαίσιο του λειτουργικού

συστήματος Windows 10. Για το HoloLens μπορεί να εντοπιστεί η έμπνευσή του στο Kinect, ένα πρόσθετο για την κονσόλα παιχνιδιών Xbox της Microsoft που εμφανίστηκε το 2010.

Αποτελείται από μια οθόνη προβολής τοποθετημένη στο κεφάλι που συνδέεται με ένα ρυθμιζόμενο εσωτερικό προστατευτικό κεφαλιού που μπορεί να στραφεί προς τα πάνω και προς τα κάτω, καθώς και προς τα εμπρός και προς τα πίσω. Για να φορέσει τη συσκευή, ο χρήστης προσαρμόζει τα HoloLens στο κεφάλι του, χρησιμοποιώντας ένα τροχό ρύθμισης στο πίσω μέρος του προστατευτικού κεφαλιού για να το ασφαλίσει γύρω από το στέμμα, στηρίζοντας και διανέμοντας το βάρος της μονάδας εξίσου για άνεση. Το 2019 η Microsoft παρουσίασε το δεύτερο μοντέλο της συσκευής.

1.3.7. Google Daydream

Η Daydream είναι μια πλατφόρμα εικονικής πραγματικότητας (VR) που αναπτύχθηκε από την Google και ενσωματώνεται στο λειτουργικό σύστημα Android για κινητά.

Τα συμβατά τηλέφωνα που ακολουθούν τις προδιαγραφές του λογισμικού και του υλικού της πλατφόρμας (και επομένως ορίζονται ως "Daydream-ready") χρησιμοποιούνται στο VR της Google Daydream View. Το Daydream είναι η δεύτερη πλατφόρμα VR της εταιρείας που ακολουθεί το Google Cardboard, το οποίο ήταν ένα σύστημα χαμηλού κόστους που προοριζόταν να ενθαρρύνει το ενδιαφέρον για το VR και ενσωματώθηκε σε συμβατές κινητές εφαρμογές και όχι στο ίδιο το λειτουργικό σύστημα.

1.3.8. 2019 - Nintendo LABO VR για το Switch

Προγραμματισμένη για κυκλοφορία στις 12 Απριλίου 2019, το κιτ LABO VR επικεντρώνεται στα γυαλιά από χαρτόνι που επιτρέπουν στους παίκτες να βλέπουν στερεοσκοπικές εικόνες 3D χρησιμοποιώντας την κονσόλα Switch, παρόμοια με αυτή του Google Cardboard. Το κύριο κιτ έρχεται με πέντε στοιχεία που συνδέονται με τα γυαλιά: ένα blaster, μια φωτογραφική μηχανή, ένα πουλί, ένας ελέφαντας και ένας διαστημικός θεατής.

1.4. Microsoft vs Google

Η Microsoft σε θέμα υλικού στο τομέα της εικονικής πραγματικότητας, προσφέρει το Microsoft Hololens όπου παρέχει κάμερες και αισθητήρες καταγραφής κίνησης και χειρονομιών χωρίς κάποιο επιπλέον εξάρτημα και μόνο με τη σύνδεσή του σε κάποιο WiFi μπορεί να προβάλει- πέρα από εικονική- και επαυξημένη πραγματικότητα.

Μπορείς να αλληλεπιδράσεις με ολογράμματα σε καλή ανάλυση με ένα αρκετά μεγάλο εύρος προβολής και να χρησιμοποιήσεις φωνητικές εντολές ακόμη και σε θορυβώδες περιβάλλον.

Με τον καλό σχεδιασμό τους, τα γυαλιά μένουν στη θέση τους συνεχώς και δεν κουράζουν αυτόν που τα φορά. Η τιμή, αν και υψηλή, είναι κατάλληλη για της δυνατότητες που προσφέρει. Η πλατφόρμα της Microsoft για ανάπτυξη τέτοιων εφαρμογών χρησιμοποιεί την Unity Engine και μπορούν να τρέξουν σε απευθείας σε Windows. Επίσης, υπάρχουν ειδικές κατηγορίες για ανάπτυξη εφαρμογών μικτής πραγματικότητας όπως το Simplygon και το Holosketch και μπορεί να ενσωματωθούν υπηρεσίες θέσης του Azure.

Με το Dynamics 365 Remote Assist ενσωματώνει άμεσο βοηθό προβλημάτων και πληροφορίες για την εργασία που γίνεται τώρα καθώς και διαγράμματα για την εφαρμογή που χρησιμοποιείται.

Αντιθέτως, η Google προσφέρει δύο επιλογές, το Cardboard και το Daydream, όπου το πρώτο είναι κατασκευασμένο από χαρτόνι και μπορεί να το διπλώσει ο χρήστης και να βάλει την συσκευή του να την χρησιμοποιήσει και το δεύτερο είναι μια πιο ολοκληρωμένη κατασκευή που και αυτή προσφέρει ένα σύστημα αισθητήρων εντοπισμού κίνησης που μπορεί να αντιλαμβάνεται τον χώρο γύρω του και συνοδεύεται από ένα χειριστήριο για αλληλεπίδραση με τα ψηφιακά στοιχεία και μπορεί να λειτουργήσει είτε σαν κανονικό Headset είτε σαν προβολέας VR εμπειριών όπως και το Cardboard.

Η τιμή του πρώτου είναι εξαιρετικά χαμηλή λόγω της απλής χρήσης και αναλωσιμότητάς του ενώ το δεύτερο είναι σχετικά πιο ακριβό αλλά και πάλι φθηνότερο από το Hololens. Η πλατφόρμα της Google κατά κύριο λόγο στοχεύει στην ανάπτυξη εφαρμογών περισσότερο στα Smartphones απ'ότι στο PC και χωρίζεται σε πολλές υποκατηγορίες, όπως VR180 Cameras, Tilt Brush, Earth VR, Cardboard, Expeditions και JUMP, ανάλογα το είδος της εφαρμογής που θέλουμε να φτιάξουμε.

Ακόμη, παρέχει πρόσβαση σε πάρα πολλές συμβατές πλατφόρμες παρακολούθησης ταινιών, σειρών και διαφόρων άλλων τηλεοπτικών γεγονότων(NETFLIX.Youtube,Next κ.α.) χρησιμοποιώντας το VR κράνος ως ένα μικρό σινεμά. Χρησιμοποιεί Android και IOS για την ανάπτυξη εφαρμογών και εμπειριών εικονικής πραγματικότητας στα και μπορεί να αξιοποιήσει και τα Unity και Unreal Engine σε αντίθεση με της Microsoft.

1.5. ΚΟΣΤΟΛΟΓΗΣΗ ΑΝΑ ΤΑ ΧΡΟΝΙΑ

Το κόστος των συσκευών εικονικής πραγματικότητας ξεκίνησε πολύ υψηλό λόγω της δυσκολίας κατασκευής τους και τη λιγοστή ζήτηση και διάθεση τους(χρησιμοποιούνταν μόνο για προσομοιώσεις).

1900-τιμή 8330 δολάρια

1970-τιμή ανέρχεται σε εκατομμύρια

Αργότερα ξεκίνησε να μπαίνει το VR στο χώρο της διασκέδασης και να πλασάρεται στο απλό κόσμο ως πολυτέλεια.

1987-τιμή 9000 ως 50000 δολάρια

1993-τιμή 322 δολάρια(Sega)

1995-τιμή 180 δολάρια(Nintendo)

Από το 2010 και μετά που άρχισε να ανθίζει η ανάπτυξη των συσκευών VR κάθε εταιρεία ανάπτυσσε το δικό της μοντέλο με διαφορετικές τιμές για τα διάφορα στατιστικά που προσέφεραν.

2014-τιμή 5 δολάρια(Google Cardboard)

-απαιτείται smartphone για να λειτουργήσει: +150 με 250 δολάρια(ελάχιστο)

2016-τιμή 300 δολάρια(Oculus Rift)

-απαιτείται XBOX Controller ή Oculus Touch για χειρονομίες: + 50 με 60 δολάρια

2016-τιμή 400 δολάρια(Playstation VR)

-απαιτείται Playstation Camera για να λειτουργήσει: +50 με 60 δολάρια

-απαιτείται Playstation Controller ή Move Controller για χειρονομίες: +60 δολάρια

2016-τιμή 500 δολάρια(HTC Vive)

2016-τιμή 80 με 100 δολάρια(Google Daydream)

-απαιτείται smartphone* για να λειτουργήσει: +400 με 800 δολάρια(ελάχιστο)

*Στην περίπτωση που είναι η έκδοση για smartphones.

2016-τιμή 3000 με 5000 δολάρια(Hololens)(Διαφημιστική Έκδοση)

2019-τιμή 3500 δολάρια(Hololens 2)

2019-τιμή 40 δολάρια(LABO VR)

Απαιτείται Nintendo Switch για να λειτουργήσει: + 350 με 400 δολάρια

Οι τιμές αυτές αφορούν μόνο τα κράνη VR χωρίς να λαμβάνουμε υπόψη το γεγονός ότι μερικά απ' αυτά χρειάζονται και διάφορα άλλα παρελκόμενα όπως χειριστήρια για να χρησιμοποιηθούν σωστά και έτσι το συνολικό κόστος αυξάνεται. Η τιμή του PC στο οποίο χρησιμοποιούνται το περισσότερα από τα παραπάνω κράνη VR δεν χρειάζεται να αναφερθεί γιατί είναι αυτονόητη και πολυποίκιλη.

2. Πλατφόρμες ανάπτυξης λογισμικού εικονικής πραγματικότητας

2.1. OpenVR SDK

Ένα από τα καλύτερα VR API σήμερα είναι το OpenVR, μια προγραμματιστική διεπαφή ανοικτού κώδικα που δημιουργήθηκε από τη Valve για να επιτρέψει την επικοινωνία με ένα σύστημα VR. Χάρη στην ανοικτή φύση του και την υποστήριξή του από τους δημιουργούς του, το OpenVR υποστηρίζεται στην πραγματικότητα από σχεδόν όλα τα κύρια ,τελευταίας τεχνολογίας ,κράνη της αγοράς, όπως το HTC Vive, το Oculus Rift και άλλα πολλά. Αυτό σημαίνει ότι εάν χρησιμοποιήσουμε αυτό το API για να επικοινωνήσουμε με το σύστημα εικονικής πραγματικότητας στην εφαρμογή VR, θα μπορέσουμε να το εκτελέσουμε σε κάθε μια από τις πολλές πλατφόρμες που υπάρχουν με σχεδόν μηδενικές τροποποιήσεις.

Οι περισσότερες από τις μηχανές με υποστήριξη VR πολλαπλών πλατφορμών το μόνο που κάνουν είναι απλά να προσθέτουν το OpenVR στο pipeline τους, έτσι ώστε η αναπτυχθείσα εφαρμογή να μπορεί να τρέξει σε όλες τις πλατφόρμες VR.

Αξίζει να αναφέρουμε ότι το OpenVR δεν είναι το μοναδικό VR API εκεί έξω, έτσι οι μηχανές υποστηρίζουν συνήθως άλλα VR API όπως το Oculus SDK (επιτρέποντάς του να δημιουργεί εφαρμογές που εκτελούνται απευθείας στο runtime Oculus), το GoogleVR SDK (η οποία διατρέχει τις ώρες εκτέλεσης του Google Cardboard και του Google Daydream), κτλ. Ως προγραμματιστής, δεν έχουμε τον έλεγχο του χρόνου εκτέλεσης στον οποίο θα εκτελεστεί η εφαρμογή μας (δηλαδή: δεν μπορούμε να χρησιμοποιήσουμε το OpenVR και επιλέξουμε να εκτελέσουμε άμεσα το runtime του Oculus).Θα υπαγορευτεί από το API / SDK που χρησιμοποιούμε αντ' αυτού.

2.2. WebVR

Εικονα 2.1 Λογοτυπο WebXR

Το WebVR είναι ένα πειραματικό περιβάλλον προγραμματισμού εφαρμογών JavaScript (API) που παρέχει υποστήριξη σε συσκευές εικονικής πραγματικότητας και βασίζεται και αυτό στην ιδέα του "γράψτε μία φορά, αναπτύξτε παντού", χρησιμοποιώντας πρότυπα ιστού για να επιτρέψει τη δημιουργία εμπειριών VR που μπορούν να διανεμηθούν μέσω διευθύνσεων URL και εκτελούνται ανεξάρτητα από το αν χρησιμοποιούμε VR ή και μη VR μέσα, εφόσον περιλαμβάνουν πρόγραμμα περιήγησης με δυνατότητα WebVR όπως smartphones και tablet.

Αυτό το API έχει σχεδιαστεί με γνώμονα τους ακόλουθους στόχους: Να μπορεί να εντοπίσει τις διαθέσιμες συσκευές εικονικής πραγματικότητας, να μας ενημερώσει για τις δυνατότητες των συσκευών και αφού

διερευνήσει τη θέση και τον προσανατολισμό της συσκευής, να προβάλει εικόνες στη συσκευή με τον κατάλληλο ρυθμό καρέ.

Το WebVR API εκθέτει μερικές νέες διεπαφές (όπως VR Display, VR pose) που επιτρέπουν στις εφαρμογές web να παρουσιάζουν περιεχόμενο στην εικονική πραγματικότητα, χρησιμοποιώντας το WebGL με τις απαραίτητες ρυθμίσεις της κάμερας και τις αλληλεπιδράσεις των συσκευών (όπως ελεγκτές ή point of view). Το API έχει σχεδιαστεί για να ακολουθεί μια συγκεκριμένη μεθοδολογία, η οποία είναι πολύ παρόμοια με άλλα παρεμβατικά API Web, όπως το API Geolocation. Μπορεί ακόμη και να φιλοξενήσει εφαρμογές AR ως μέρος του ίδιου API, χάρη σε μια επέκταση που ονομάζεται WebXR.

2.3. Unity Engine

Εικόνα **2.2** Λογότυπο Unity

Η Unity είναι ένα engine πραγματικού χρόνου, ο οποίος αναπτύχθηκε από την Unity Technologies και ανακοινώθηκε για πρώτη φορά τον Ιούνιο του 2005 στο Παγκόσμιο Συνέδριο Προγραμματιστών της Apple Inc. ως αποκλειστική μηχανή παιχνιδιών OS X. Από το 2018, το engine έχει επεκταθεί για να υποστηρίξει 27 πλατφόρμες. Το engine μπορεί να χρησιμοποιηθεί και για προσομοιώσεις για τις πολλές πλατφόρμες του. Αρκετές σημαντικές εκδόσεις της Unity έχουν κυκλοφορήσει από την κυκλοφορία της, με την τελευταία σταθερή έκδοση να είναι η Unity 2018.3.9.

2.3.1. Unity Editor

Παρέχει έναν all-in-one editor που μπορεί να επεκτείνεται ώστε να ταιριάζει με τη ροή της εργασίας παραγωγής. Είναι η πιο διαδεδομένη πλατφόρμα ανάπτυξης VR και πάνω από το 91% των εμπειριών HoloLens είναι από το Unity. Είτε πρόκειται για VR, AR ή MR, μπορούμε να βασιστούμε στο άκρως βελτιστοποιημένο rendering pipeline του Unity και στις δυνατότητες των εργαλείων του Editor για να καταστήσουμε την XR έργα που θέλουμε να υλοποιήσουμε δυνατά.

Ο Unity Editor με τα πολλά εργαλεία που διαθέτει, επιτρέπει την ταχεία επεξεργασία και επανάληψη των κύκλων ανάπτυξης, συμπεριλαμβανομένης της λειτουργίας αναπαραγωγής για γρήγορες προεπισκοπήσεις της εργασίας μας σε πραγματικό χρόνο.

Η λειτουργία αναπαραγωγής Unity είναι ένα απίστευτα ισχυρό εργαλείο ανάπτυξης για γρήγορη επαναληπτική επεξεργασία. Πατώντας το κουμπί Play θα είμαστε μέσα στο παιχνίδι μας, παίζοντας και προεπισκοπώντας πώς θα φανεί στην πλατφόρμα της συγκεκριμένης τελικής κατασκευής. Θέτοντας τη σε παύση και αλλάζοντας τις τιμές, τα στοιχεία ενεργητικού, τα σενάρια και άλλες ιδιότητες θα μπορούμε να δούμε αμέσως τα αποτελέσματα. Αν προχωρήσουμε μέσα στο παιχνίδι μας πλαίσιο ανά πλαίσιο, θα κάνουμε εύκολη αποσφαλμάτωση με την εγγενή ενσωμάτωση του Visual Studio.

Ο Editor, ο οποίος είναι διαθέσιμος σε Windows και Mac, περιλαμβάνει μια ποικιλία φιλικών προς τον καλλιτέχνη εργαλείων για το σχεδιασμό επιβλητικών εμπειριών και κόσμων παιχνιδιών, καθώς και μια ισχυρή συλλογή εργαλείων προγραμματιστών για την εφαρμογή λογικής παιχνιδιών και gameplay υψηλής απόδοσης.

Το Unity υποστηρίζει την ανάπτυξη 2D και 3D εμπειριών και παιχνιδιών με δυνατότητες και λειτουργικότητα για τις συγκεκριμένες ανάγκες μας σε όλους τους τύπους και είδη. Μέσα στις 2D εμπειρίες, η Unity επιτρέπει την εισαγωγή των sprites και ενός προηγμένου 2D renderer κόσμου. Για 3D εμπειρίες, η Unity επιτρέπει προδιαγραφές συμπίεσης textures, mipmaps και ρυθμίσεων ανάλυσης για κάθε πλατφόρμα που υποστηρίζει η μηχανή παιχνιδιών και παρέχει υποστήριξη για bump mapping, reflection mapping, parallax mapping,screen space ambient occlusion (SSAO),δυναμικές σκιές χρησιμοποιώντας χάρτες σκιάς,render-to-texture και post-processing effects πλήρης οθόνης.

Επίσης, περιλαμβάνει εργαλεία AI pathfinding όπου είναι ένα σύστημα πλοήγησης που μας επιτρέπει να δημιουργούμε NPC που μπορούν να κινηθούν έξυπνα γύρω από τον κόσμο παιχνιδιών. Το σύστημα χρησιμοποιεί πλέγματα πλοήγησης που δημιουργούνται αυτόματα από τη γεωμετρία της σκηνής μας, ή ακόμα και δυναμικά εμπόδια, για να αλλάξουν την πλοήγηση των χαρακτήρων κατά το χρόνο εκτέλεσης.

Υπάρχουν και τα Unity Prefabs, τα οποία είναι προ-ρυθμισμένα αντικείμενα παιχνιδιών, μας παρέχουν αποτελεσματικές και ευέλικτες ροές εργασίας.

Οι νέες ροές εργασίας Prefab, επιτρέπουν να χωρίσουμε τα Prefabs σε ένα κοκκώδες επίπεδο. Αυτό μας δίνει μεγαλύτερη ευελιξία, αυξάνει την παραγωγικότητά μας και μας δίνει τη δυνατότητα να εργάζεστε με αυτοπεποίθηση χωρίς να ανησυχούμε για την παραγωγή χρονοβόρων σφαλμάτων.

Ο μακροπρόθεσμος στόχος ήταν όχι μόνο η υλοποίηση της υποστήριξης για την εμφωλίαση, αλλά και η επανεξέταση των ροών εργασίας του Prefab, ώστε διαφορετικά μέλη της ομάδας να μπορούν να επεξεργάζονται ταυτόχρονα με συνέπεια και αποτελεσματικότητα τα Prefabs.

Το ενσωματωμένο σύστημα UI μας επιτρέπει να δημιουργούμε διεπαφές χρήστη γρήγορα και διαισθητικά και καταφέρνουμε να επωφεληθούμε από την υποστήριξη Box2D και NVIDIA PhysX για εξαιρετικά ρεαλιστικό και υψηλής απόδοσης gameplay.

Μπορούμε να επεκτείνουμε τον editor με τα εργαλεία που χρειαζόμαστε για να ταιριάζει με τη ροή εργασίας της ομάδας μας. Δημιουργώντας και προσθέτοντας προσαρμοσμένες επεκτάσεις ή παίρνοντας οτιδήποτε χρειαζόμαστε από το Asset Store, το οποίο διαθέτει χιλιάδες πόρους, εργαλεία και επεκτάσεις οδηγεί στην επιτάχυνση των έργων μας.

Ένα από τα καλύτερα εργαλεία που υπάρχει στο Asset Store για να κάνουμε αυτοματοποιημένες λειτουργίες χωρίς ούτε μια γραμμή κώδικα, είναι το Playmaker όπου μας προσφέρει έναν ισχυρό editor οπτικών καταστάσεων στην εργαλειοθήκη μας που μπορεί να διασυνδεθεί με δέσμες ενεργειών ή επεκταθεί με προσαρμοσμένες ενέργειες.

Το Engine ενώ είναι κατάλληλο και πιο βολικό για έργα ενός ατόμου καθώς παρέχει μια πιο βελτιωμένη διαδικασία ανάπτυξης ειδικά στη φάση των πρωτοτύπων, δεν σημαίνει ότι δεν μπορούν να το χρησιμοποιήσουν και ομάδες περισσότερων ατόμων.

Μπορούμε να δούμε σε τι εργάζονται τα άλλα μέλη της ομάδας, ακριβώς το Unity Editor, όπου θα περνά το μεγαλύτερο μέρος του χρόνου ο προγραμματιστής.

2.3.2. Unity Particle System (Σύστημα Σωματιδίων)

Το Σύστημα Σωματιδίων της Unity μας επιτρέπει να δημιουργούμε εύκολα υγρά, σύννεφα και φλόγες, δημιουργώντας και ζωντανεύοντας μεγάλο αριθμό μικρών 2D εικόνων στη σκηνή. Κάθε σωματίδιο έχει μια προκαθορισμένη διάρκεια ζωής, τυπικά λίγα δευτερόλεπτα, κατά την οποία μπορεί να υποστεί διάφορες αλλαγές. Αρχίζει τη ζωή του όταν παράγεται ή εκπέμπεται από το σύστημα σωματιδίων του. Το σύστημα εκπέμπει σωματίδια σε τυχαίες θέσεις μέσα σε μια περιοχή διαστήματος που έχει σχήμα σφαίρα, ημισφαίριο, κώνο, κιβώτιο ή οποιοδήποτε αυθαίρετο πλέγμα. Το σωματίδιο εμφανίζεται μέχρι να τελειώσει ο χρόνος του, οπότε αφαιρείται από το σύστημα. Ο ρυθμός εκπομπής του συστήματος δείχνει κατά προσέγγιση τον αριθμό των σωματιδίων που εκπέμπονται ανά δευτερόλεπτο, αν και οι ακριβείς χρόνοι εκπομπής τυγχάνουν ελαφρώς τυχαίας επιλογής.

Η επιλογή του ρυθμού εκπομπής και της μέσης διάρκειας ζωής των σωματιδίων καθορίζει τον αριθμό των σωματιδίων στην «σταθερή» κατάσταση (δηλαδή, όπου συμβαίνει ο θάνατος των σωματιδίων και των εκπομπών σωματιδίων με τον ίδιο ρυθμό) και πόσο χρόνο το σύστημα χρειάζεται για να φτάσει σε αυτήν την κατάσταση.

Οι ρυθμίσεις εκπομπής και διάρκειας ζωής επηρεάζουν τη συνολική συμπεριφορά του συστήματος, αλλά τα μεμονωμένα σωματίδια μπορούν επίσης να αλλάξουν με την πάροδο του χρόνου. Το καθένα έχει ένα διάνυσμα ταχύτητας που καθορίζει την κατεύθυνση και την απόσταση που κινείται το σωματίδιο με κάθε ενημέρωση πλαισίου. Η ταχύτητα μπορεί να αλλάξει με τις δυνάμεις και τη βαρύτητα που ασκούνται από το ίδιο το σύστημα ή όταν τα σωματίδια φουσκώνουν γύρω από μια ζώνη αέρα σε ένα Terrain.

Το χρώμα, το μέγεθος και η περιστροφή κάθε σωματιδίου μπορούν επίσης να αλλάξουν κατά τη διάρκεια της ζωής του ή ανάλογα με την τρέχουσα ταχύτητα κίνησης του. Το χρώμα περιλαμβάνει ένα στοιχείο άλφα (διαφάνεια), έτσι ώστε ένα σωματίδιο να μπορεί να εξασθενεί βαθμιαία μέσα και έξω από την ύπαρξη και όχι απλά να εμφανίζεται και να εξαφανίζεται απότομα. Χρησιμοποιούμενη σε συνδυασμό, η δυναμική των σωματιδίων μπορεί να χρησιμοποιηθεί για την προσομοίωση πολλών ειδών ρευστών αποτελεσμάτων αρκετά πειστικά. Για παράδειγμα, ένας καταρράκτης μπορεί να προσομοιωθεί χρησιμοποιώντας ένα λεπτό σχήμα εκπομπής και αφήνοντας τα σωματίδια νερού απλά να πέσουν κάτω από τη βαρύτητα, επιταχύνοντας καθώς πηγαίνουν.

2.3.3. Unity Timeline

Με το Timeline, έχουμε τη δυνατότητα να εργαστούμε in-context και να επαναλαμβάνουμε μέχρι το περιεχόμενο να είναι σωστό. Είναι ένα ισχυρό, αλλά φιλικό προς το χρήστη εργαλείο, το οποίο μας επιτρέπει να δημιουργήσουμε πλούσιο περιεχόμενο χρησιμοποιώντας τα αντικείμενα του παιχνιδιού, τα κινούμενα σχέδια, τους ήχους, τα σωματίδια και άλλα στοιχεία σκηνής. Ακόμη, καταφέρνουμε να συνθέσουμε, χορογράψουμε και ενορχηστρώσουμε τις κινούμενες εικόνες, τις κάμερες, τον ήχο και οποιαδήποτε άλλα αντικείμενα του παιχνιδιού, να δημιουργήσουμε ταχύτερα το περιεχόμενο χρησιμοποιώντας μια διεπαφή πολλαπλών σταδίων για να "σύρουμε και αποθέσουμε" αντικείμενα και να ελέγξουμε και να εγγράψουμε κινούμενα σχέδια για να δημιουργήσουμε γρήγορα πλούσιες σκηνές.

Παραδοσιακά, κατά τη δημιουργία αλληλεπιδράσεων αντικειμένου-παιχνιδιού με μια ποικιλία πιθανών αποτελεσμάτων, κάθε αποτέλεσμα έπρεπε να προγραμματιστεί. Τώρα μπορούμε να χρησιμοποιήσουμε το Timeline για να αντικαταστήσουμε τον κώδικα για πολλά σενάρια, όπως χρονικές αλληλεπιδράσεις αντικειμένου-παιχνιδιού, δημιουργία μεταβάσεων μεταξύ κινούμενων εικόνων, σύνθεση cutscenes, cinematics και ακολουθιών gameplay, δημιουργία προτύπων περιβάλλοντος όπως ο καιρός, ο χρόνος και οι εποχές.

Μας παρέχει τη δυνατότητα κλιπ ακολουθίας και ήχου όπως στην επεξεργασία βίντεο με τη χρήση των έξυπνων κάμερών Cinemachine και μπορούμε να προσθέσουμε εφέ μετά επεξεργασίας και ταξινόμηση χρωμάτων.

Επίσης, δημιουργούμε γρηγορότερα σκηνές κόσμου, επιπέδου και περιβάλλοντος βάζοντας σε ακολουθία και αναπαράγοντας στοιχεία σκηνής με συμβάντα εκκίνησης και μπορούμε να προσθέσουμε εφέ όπως φωτοβολίδες και σωματίδια φακών. Οι έλεγχοι κινούμενων εικόνων που κάνουμε χωρίς κώδικα μας επιτρέπει να καταγράψουμε νέα ή χρησιμοποιήσουμε τα υπάρχοντα κινούμενα σχέδια και τα τοποθετήστε εγκαίρως, να ρυθμίσουμε, να βάλουμε σε βρόχο και να αναμείξουμε διαφορετικές ακολουθίες κινούμενων εικόνων και να εργαστούμε με πολλαπλούς χαρακτήρες/αντικείμενα ταυτόχρονα.

Ακόμη, δημιουργούμε και διαχειριζόμαστε σκηνές με γραφικό περιβάλλον χρήστη με την ικανότητα να κάνουμε μετάβαση από το gameplay σε cutscene και πίσω, να ελέγξουμε χαρακτήρες που δεν είναι παίκτες και να αναπαράξουμε βίντεο και ήχο και σωματίδια. Μπορούμε και να αποκτήσουμε ένα νέο επίπεδο ελέγχου με το Timeline όπου μας δίνει πλήρη δημιουργική ελευθερία και την ικανότητα επίλυσης προβλημάτων κατά τη σύνθεση της σκηνής μας και την ενορχήστρωση των στοιχείων μας.

Με αυτό, μπορούμε να χρησιμοποιήσουμε διαφορετικά κομμάτια για να επηρεάσουμε διαφορετικές συμπεριφορές και ρυθμίσεις, προκειμένου να αποκτήσουμε σαφή έλεγχο της ενεργοποίησης και της κίνησης του αντικειμένου-παιχνιδιού που ρυθμίζοντας κάθε αντικείμενο παιχνιδιού μεμονωμένα από έναν ή περισσότερους τύπους κομματιών και ζωντανεύοντας τις μοναδικές παραμέτρους αντικειμένου για εικονογραφημένα εφέ εφαρμόζοντας πολλά κομμάτια.

Μπορούμε να αποκτήσουμε απλώς τη συμπεριφορά που αναζητάμε χρησιμοποιώντας συνδυασμούς τροποποιητών και να διατηρήσουμε την διαχείριση έργων τακτοποιημένη με ομαδοποίηση, επέκταση και κατάρρευση κομματιών και έλεγχο άλλων χρονοδιαγραμμάτων και να επεκτείνουμε την δημιουργική ελευθερία με προσαρμόσιμα Playables αφού μπορούν εύκολα μπορούμε να επεκτείνουμε τη λειτουργικότητα του Timeline για καλλιτέχνες και σχεδιαστές. Τα playables επιτρέπουν την ανάπτυξη προηγμένων συμπεριφορών, αυτοματοποίησης και προσαρμοσμένων στοιχείων ελέγχου.

Μπορούμε να χρησιμοποιήσουμε τα Default Playables για τα συνηθισμένα σενάρια (συμπεριλαμβανομένων των LightControl, NavMeshControl, ScreenFader, TextSwitcher, TimeDilation, TransformTween, Video) και έναν Οδηγό Αναπαραγωγής για να επιταχύνουμε τη δημιουργία ιδίων ιδεών.

2.3.4. Unity scripting

Προχωρημένα εργαλεία δημιουργίας προφίλ καταφέρνουμε να βελτιστοποιήσουμε συνεχώς το περιεχόμενό μας καθ'όλη τη διάρκεια της ανάπτυξης με τα χαρακτηριστικά προφίλ της Unity. Ελέγχοντας αν το περιεχόμενό μας είναι CPU ή GPU, για παράδειγμα, και επισημάνουμε εκείνες τις περιοχές που χρειάζονται βελτίωση θα μπορούμε έτσι να παρέχουμε στο κοινό μας μια ομαλή εμπειρία.

Επωφελούμαστε και από τις C ++ επιδόσεις μας με την πλατφόρμα IL2CPP (Intermediate Language To C ++) που αναπτύχθηκε από την Unity με scripting runtime Mono /.NET 4.6 / C # 7.3.

Το υψηλής απόδοσης πολυνηματικό σύστημα μας επιτρέπει να χρησιμοποιήσουμε πλήρως τους επεξεργαστές πολλαπλών πυρήνων χωρίς βαρύ προγραμματισμό με τρία υποσυστήματα, το C # Job System, το οποίο δίνει ένα ασφαλές και εύκολο sandbox για τη γραφή παράλληλου κώδικα, το Σύστημα Component Entity (ECS), ένα πρότυπο για τη σύνταξη κώδικα υψηλής απόδοσης από προεπιλογή και το Burst Compiler, το οποίο παράγει έναν εξαιρετικά βελτιστοποιημένο εγγενή κώδικα.

Το εξαιρετικά αρθρωτό runtime της Unity δίνει τη δυνατότητα στους προγραμματιστές να δημιουργούν ελαφρές και γρήγορες εμπειρίες χρησιμοποιώντας το γραφικό περιβάλλον ενός εκτάσιμου Editor που χαρακτηρίζονται ως άμεσες εμπειρίες και άμεσα παιχνίδια.

Μας δίνει τον έλεγχο των χαρακτηριστικών που θέλουμε να συμπεριλάβουμε, τα οποία μας δίνουν πλήρη έλεγχο του μεγέθους του χρόνου εκτέλεσης. Και, μπορούμε να συσκευάσουμε το περιεχόμενο που χρειάζεστε στα μικρότερα μεγέθη αρχείων χρησιμοποιώντας την τεχνολογία συμπίεσης της Unity. Το Σύστημα Component Entity (ECS) προσφέρει απαράμιλλη απόδοση και επεκτασιμότητα. Σε ένα ευρύ φάσμα συσκευών, μπορούμε να προσφέρουμε υψηλής ποιότητας εμπειρίες που πηγαίνουν οπουδήποτε τους χρειαζόμαστε. Και, τα άμεσα παιχνίδια που δημιουργούμε σήμερα είναι κατασκευασμένα χρησιμοποιώντας την ίδια τεχνολογία αιχμής που θα τροφοδοτήσει ακόμα και σε μελλοντικά έργα. Οι προγραμματιστές άμεσων παιχνιδιών μπορούν να προσφέρουν αυτές τις εμπειρίες χρησιμοποιώντας εργαλεία που έχουν σχεδιαστεί για να επιταχύνουν την παραγωγή.

2.3.5. Unity XR

Σε ότι αφορά τα XR, το Unity υποστηρίζει όλες τις τελευταίες και μεγαλύτερες πλατφόρμες με εγγενής υποστήριξη να είναι πλέον διαθέσιμη για τους Oculus Rift, Steam VR / Vive, Playstation VR, Gear VR, Microsoft HoloLens και Daydream της Google προσεγγίζοντας έτσι το ευρύτερο δυνατό κοινό.Η επέκτασιμότητα του Unity επιτρέπει επίσης σε οποιονδήποτε third-party κατασκευαστή υλικού να αναπτύξει τα δικά του plugins και SDK. Δεν θα έχει όλα τα πλεονεκτήματα του άριστα βελτιστοποιημένου pipeline, αλλά μπορούμε ακόμα να χρησιμοποιήσουμε το Unity για να δημιουργήσουμε μια εμπειρία VR / AR. Τα ήδη διαθέσιμα πρόσθετα τρίτων περιλαμβάνουν το SDK Cardboard για Unity, OSVR, MergeVR και Vuforia.

Η προτιμώμενη πλατφόρμα για τη δημιουργία κινητών εμπειριών AR, ο AR Editor Remote του Unity επιτρέπει να χρησιμοποιήσουμε τη φωτογραφική μηχανή μιας συσκευής για να τραβήξουμε δεδομένα στον πραγματικό κόσμο στο Editor, γεγονός που επιτρέπει να δοκιμάζουμε γρήγορα, να επαναλαμβάνουμε και να πειραματιζόμαστε απευθείας στην κινητή συσκευή μας χωρίς να χρειάζεται να εγκαταλείψουμε τον επεξεργαστή Unity Editor. Υπάρχει και το Έργο: MARS που είναι μια επέκταση του Unity που παρέχει δυνατότητες AR, δίνοντας μας τη δυνατότητα να δημιουργούμε εφαρμογές που αλληλοεπιδρούν έξυπνα με οποιοδήποτε πραγματικό περιβάλλον, με ελάχιστη ή μη προσαρμοσμένη κωδικοποίηση. Στην νοοτροπία των άμεσων παιχνιδιών, δημιουργούμε πλούσιο, άμεσο περιεχόμενο AR και εμπειρίες που είναι μικρές, ελαφρές και γρήγορες.

2.3.6. Unity rendering

Το rendering engine της Unity σε πραγματικό χρόνο επιτρέπει εκπληκτική οπτική πιστότητα. Υποστηρίζει πολλαπλές πλατφόρμες, αλλά εξακολουθεί να παραμένει κοντά στο API γραφικών χαμηλού επιπέδου κάθε πλατφόρμα που μας επιτρέπει να επωφεληθούμε από τις τελευταίες βελτιώσεις GPU και hardware όπως Vulkan, iOS Metal, DirectX12, nVidia VRWorks ή AMD LiquidVR.

Το Unity παρέχει αρκετές ενσωματωμένες λειτουργίες rendering, οι οποίες είναι αρκετές για τα περισσότερα παιχνίδια. Το SRP θα μας επιτρέψει να εκμεταλλευτούμε πλήρως τη δύναμη του σύγχρονου υλικού και των GPU χωρίς να χρειαστεί να γράψουμε εκατομμύρια γραμμές κώδικα.

Υπάρχει περισσότερος έλεγχος για τους προγραμματιστές αφού η δυνατότητα ρύθμισης της παράδοσης σε Unity από C# scripts θα μας επιτρέψει να βελτιστοποιήσουμε την απόδοση για συγκεκριμένο υλικό, προσαρμόσουμε τις διαδικασίες απόδοσης σε πάρα πολύ μικρό επίπεδο ανάλογα με τις ανάγκες μας και να ελέγξουμε πώς χρησιμοποιούνται οι πόροι απόδοσης. Μπορούμε να χρησιμοποιήσουμε τους ενσωματωμένους pipelines που μας προσφέρονται, να αναπτύξουμε τον δικό μας pipeline από την αρχή ή να αρχίσουμε να τροποποιούμε έναν από τους παρεχόμενους pipelines για να καλύψουν τις συγκεκριμένες απαιτήσεις μας.

Ο στόχος του Lightweight Render Pipeline είναι να παρέχει βελτιστοποιημένη απόδοση σε πραγματικό χρόνο σε πλατφόρμες με περιορισμένη απόδοση, κάνοντας κάποιες συμφωνίες όσον αφορά τον φωτισμό και τη σκίαση.

Το κοινό-στόχος για το LW RP είναι όταν θέλουμε να αναπτύξουμε σε ένα ευρύ φάσμα πλατφορμών κινητών τηλεφώνων, VR και όταν θέλουμε να αναπτύξουμε παιχνίδια με περιορισμένες ανάγκες φωτισμού σε πραγματικό χρόνο. Το LW RP εκτελεί μονόπλευρη απόδοση εμπρός με ένα φως σκιάς σε πραγματικό χρόνο και σάρωση φωτός ανά αντικείμενο με το πλεονέκτημα ότι όλα τα φώτα σκιάζονται σε ένα μόνο πέρασμα. Σε σύγκριση με την προηγούμενη απόδοση του pipeline παλαιού τύπου, η οποία εκτελεί ένα επιπλέον πέρασμα ανά εικονοστοιχείο φωτός εντός εύρους, η χρήση του pipeline LW θα έχει ως αποτέλεσμα λιγότερες κλήσεις κλήρωσης. Αυτό είναι εις βάρος κάποιας πρόσθετης πολυπλοκότητας των shader λόγω περισσότερων φώτων ανά δίοδο. Το LW RP υποστηρίζεται επίσης από το εργαλείο Shader Graph, το οποίο θα προσφέρει κάποια πρόσθετα οφέλη όσον αφορά τη ροή εργασιών συγγραφής shader.

Το Shader Graph μας επιτρέπει να δημιουργούμε εύκολα shaders, κατασκευάζοντάς τα οπτικά και να βλέπουμε τα αποτελέσματα σε πραγματικό χρόνο και να δημιουργούμε και συνδέουμε κόμβους σε ένα γράφημα δικτύου αντί να γράφουμε κώδικα. Το Shader Graph ανοίγει το πεδίο για καλλιτέχνες και άλλα μέλη της ομάδας, διευκολύνοντας τη δημιουργία shaders. Απλά συνδέοντας κόμβους σε ένα δίκτυο γραφικών και μπορούμε να δούμε τις αλλαγές μας αμέσως. Μπορούμε να κάνουμε πράγματα όπως να τροποποιήσουμε τη διαδικασία μας επιφανειακά, να τυλίξουμε και να ζωντανέψουμε UV, να τροποποιήσουμε την εμφάνιση των αντικειμένων μας χρησιμοποιώντας γνωστές λειτουργίες ρύθμισης εικόνας, να αλλάξουμε την επιφάνεια του αντικειμένου μας με βάση πληροφορίες σχετικά με αυτό, όπως η θέση του στον κόσμο, οι κανονικοί, η απόσταση από τη φωτογραφική μηχανή κ.λπ. Ακόμη, να ρυθμίσουμε γρήγορα τα γραφικά του shader στο πλαίσιο μιας σκηνής χρησιμοποιώντας τον Επιθεωρητή Υλικών και να μοιραστούμε δίκτυα κόμβων μεταξύ πολλαπλών γραφημάτων και χρηστών δημιουργώντας υπογραφήματα.

Το πλαίσιο γραφημάτων μας δείχνει τα αποτελέσματα των ενεργειών μας καθώς εργαζόμαστε. Δεν υπάρχει περίοδος κατασκευής όπου πρέπει να περιμένουμε να ετοιμάζονται οι αλλαγές. Ακόμα και οι νέοι χρήστες μπορούν απλά να αρχίσουν να πειραματίζονται.

Τα εργαλεία προσαρμογής και οπτικής μας επιτρέπουν να δημιουργούμε καλλιτεχνικά ή άλλα ειδικά εφέ, όπως συσκευές θερμότητας, χιόνι και εξαφάνισης. Το σύστημα Shader Graph έχει σχεδιαστεί για να λειτουργεί με τη λειτουργία Scriptable Render Pipeline. Οι κύριοι κόμβοι που λειτουργούν με ελαφρύ σωλήνα διαχωρισμού (LWRP) και σωλήνα υψηλής ευκρίνειας (HDRP) περιλαμβάνονται εκτός πλαισίου, μπορεί να επεκταθεί για να συνεργαστεί με οποιοδήποτε προσαρμοσμένο αγωγό κατασκευής και μας δίνει τη δυνατότητα να ορίσουμε τη συμπεριφορά του προσαρμοσμένου κόμβου απευθείας στο αρχείο Shader Graph ή μέσω αρχείων HLSL.

Το HDRP στοχεύει σε υπολογιστές και κονσόλες υψηλής τεχνολογίας και δίνει προτεραιότητα σε εκπληκτικά οπτικά εφέ υψηλής ευκρίνειας. Είναι ιδανικό για παιχνίδια με ποιότητα AAA, επιδείξεις και εφαρμογές που εκτελούνται σε συσκευές συμβατές με το Compute Shader / GPU, όπως υπολογιστές και κονσόλες με πολλαπλά κανάλια. Υποστηρίζει τόσο την εμπρόσθια όσο και την ανασταλτική απόδοση και χρησιμοποιεί φυσικό φωτισμό και υλικά.

Ο σύγχρονος φωτισμός παιχνιδιών κάνει εκτεταμένη χρήση του «παγκόσμιου φωτισμού» όπου ο παγκόσμιος φωτισμός είναι ένας όρος που χρησιμοποιείται για να περιγράψει μια σειρά από τεχνικές και μαθηματικά μοντέλα που επιχειρούν να προσομοιώσουν τη σύνθετη συμπεριφορά του φωτός καθώς αναπηδά και αλληλοεπιδρά με τον κόσμο. Η προσομοίωση του παγκόσμιου φωτισμού με ακρίβεια είναι προκλητική και μπορεί να είναι υπολογιστικά δαπανηρή.

Εξαιτίας αυτού, τα παιχνίδια χρησιμοποιούν μια σειρά προσεγγίσεων για να χειριστούν αυτούς τους υπολογισμούς εκ των προτέρων, παρά κατά τη διάρκεια του παιχνιδιού.

Στην Unity, υπάρχουν δύο διακεκριμένες τεχνικές διαθέσιμες για προ-υπολογισμού παγκόσμιου φωτισμού (Global Illumination), ή αναπήδησης φωτισμού. Αυτά είναι τα Baked GI και το Precomputed Realtime GI.

Όταν χρησιμοποιούμε το Precomputed Realtime GI, ένας προκαταρκτικός υπολογισμός φωτισμού είναι η διαδικασία υπολογισμού της αναπήδησης του φωτός γύρω από τη στατική γεωμετρία μέσα σε μια Σκηνή στο Unity Editor και την αποθήκευση αυτών των δεδομένων για χρήση κατά το χρόνο εκτέλεσης. Αυτή η διαδικασία μειώνει τον αριθμό των υπολογισμών φωτισμού που πρέπει να εκτελούνται κατά το χρόνο εκτέλεσης, επιτρέποντας φωτισμό σε πραγματικό χρόνο, διατηρώντας παράλληλα τα interactive framerates.

Όταν χρησιμοποιούμε το Baked GI, οι παραδοσιακές υφές του φωτός δημιουργούνται εκτός σύνδεσης κατά τη διάρκεια της διαδικασίας προεπεξεργασίας. Αυτές οι υφές, στη συνέχεια, υπάρχουν ως assets στο πλαίσιο του έργου και δεν μπορούν να τροποποιηθούν κατά το χρόνο εκτέλεσης. Το Precomputed Realtime GI δεν δημιουργεί στοιχεία ενεργητικού lightmap με τον ίδιο τρόπο.

Αντ 'αυτού, τα δεδομένα φωτισμού αποθηκεύονται ως στοιχείο φωτισμού δεδομένων που περιέχει τις πληροφορίες που απαιτούνται για τη δημιουργία και την ενημέρωση ενός συνόλου φωτεινών σημείων χαμηλής ανάλυσης διαδραστικά, κατά το χρόνο εκτέλεσης. Εκτός εάν η Σκηνή μας έχει προετοιμαστεί και βελτιστοποιηθεί σωστά, ο χρόνος που απαιτείται για την ολοκλήρωση αυτών των υπολογισμών μπορεί να γίνει υπερβολικός.

Μόλις μάθουμε και εφαρμόσουμε μερικές από τις τεχνικές, μπορούμε να επωφεληθούμε από τα πλεονεκτήματα της χρήσης του Precomputed Realtime GI: γρήγοροι χρόνοι επανάληψης φωτισμού, η δυνατότητα πειραματισμού πιο γρήγορα κατά τη διάρκεια της διαδικασίας φωτισμού και ο αναβρασμός σε πραγματικό χρόνο κατά τη διάρκεια του παιχνιδιού.

Η νέα στοίβα μετά επεξεργασίας Unity συνδυάζει ένα πλήρες σύνολο εφέ εικόνας σε έναν ενιαίο αγωγό μετά την επεξεργασία. Μπορούμε να συνδυάσουμε πολλά εφέ σε ένα post-process pipeline και ένα προκαθορισμένο asset-based σύστημα διαμόρφωσης, καθιστά εύκολη τη διαχείριση.

Παρέχει τα ακόλουθα εφέ: Antialiasing (FXAA, Temporal AA), Ambient Occlusion, Προβολές χώρου οθόνης, Ομίχλη, Βάθος πεδίου, Θάλασσα κινήσεων, Προσαρμογή ματιών, Bloom, Χρωματική ταξινόμηση, Χρήστης Lut, Χρωματική εκτροπή, Grain, Vignette, Dithering.

2.3.7. Unity Cinemachine

Το Σύστημα έξυπνης κάμερας Cinemachine μας προσφέρει τη δυνατότητα να χειριστούμε σειρές ελέγχου όπως ένας σκηνοθέτης με τα προηγμένα εργαλεία κάμερας, συμπεριλαμβανομένων των ρυθμίσεων πραγματικής κάμερας και να καταφέρουμε να συγκεντρώσουμε φωτογραφίες με έμφαση στην κατεύθυνση της τέχνης, όχι τις λεπτομέρειες εφαρμογής.

Ακόμη, δίνοντας στις έξυπνες κάμερες Cinemachine απλές οδηγίες να ακολουθήσουν όπως "ακολουθήστε το κεφάλι του χαρακτήρα", έτσι ώστε εάν αλλάξει η κινούμενη εικόνα, η καταγραφή μας θα συνεχίσει να λειτουργεί σωστά. Μερικά από τα δυνατά χαρακτηριστικά είναι η Διαδικαστική πλαισίωση όπου η κάμερα συνθέτει αυτόματα τη δράση με έλεγχο της σύνθεσης και το αποτέλεσμα να είναι το γεγονός ότι ρυθμίζει ένα μεγάλο αριθμό διατάξεων, και ακόμη και όταν αλλάζουμε τις κινούμενες εικόνες, να έχουμε ακόμη τις λήψεις και ας έχουν αλλάξει οι κινούμενες εικόνες, τους Εικονικούς χειριστές κάμερας όπου φωτογραφική μηχανή ακολουθεί αυτόματα τη δράση και μας δίνει μυριάδες ελέγχους για τον τρόπο με τον οποίο οι κάμερες πρέπει να ακολουθήσουν τη δράση.

Ακόμη, έχουμε Post-Processing για καλύτερη έγχρωμη ποιότητα κάθε λήψης όπου η ταξινόμηση χρωμάτων, η εξομοίωση φακών, το βάθος πεδίου - και πολλά άλλα - ζουν σε κάθε λήψη. Εάν αλλάξουμε την επεξεργασία, ο βαθμός χρώματος αλλάζει μαζί και με τον Μίκτη λήψεων μπορούμε να αναμείξουμε λήψεις Cinemachine στο Timeline δημιουργώντας animation και όλες οι υπόλοιπες λειτουργίες θα ακολουθήσουν.

Ρυθμίζοντας τις ακολουθίες λήψεων μας και συνδυάζοντάς τα για την επιθυμητή διάρκεια θα έχουμε την ομαλή κίνηση της κάμερας να λειτουργεί σε δευτερόλεπτα. Τέλος, η λειτουργία ζουμ αυτόματα θα λαμβάνει το αντικείμενο με το σωστό μέγεθος στην οθόνη σε σενάρια όπου οι θέσεις των χαρακτήρων θα μπορούσαν να είναι μεταβλητές. Η κάμερα θα ρυθμίσει δυναμικά το ζουμ για να εξασφαλίσει ότι τα θέματα είναι το επιθυμητό μέγεθος οθόνης, ό,τι και να συμβεί και τα πακέτα φακού Cinemachine μας επιτρέπουν να ορίσουμε μια λίστα αγαπημένων φακών ως επιλογές dropdown προσθέτοντας συνέπεια στο έργο μας, αφού περιορίζουμε τα εστιακά μήκη σε ένα προκαθορισμένο κιτ.

2.3.8. Unity CAD

Μέσω της συνεργασίας με το λογισμικό PiXYZ, την καλύτερη λύση για τη βελτιστοποίηση των τρισδιάστατων δεδομένων, μας προσφέρει όλα όσα χρειαζόμαστε για να εισάγουμε, να διαχειριστούμε και να βελτιστοποιήσουμε γρήγορα τις μεγάλες συναρμολογήσεις μας CAD σε Unity για έργα απεικόνισης σε πραγματικό χρόνο. Το PiXYZ αναλαμβάνει τα δύσκολα της εξίσωσης δεδομένων CAD-prep, εξασφαλίζοντας ότι όλα τα αρχεία CAD μας είναι βελτιστοποιημένα για την Unity, ανεξάρτητα από την πηγή.

Το PiXYZ PLUGIN ξεκλειδώνει τα δεδομένα μας CAD παραδίδοντας ομοιογενής ενοποίηση στην Unity, συμπεριλαμβανομένης της εισαγωγής μεταδεδομένων, έτοιμα προς χρήση assets με τέλεια μονωμένα meshes και μειωμένο αριθμό πολυγώνων μέσω αυτόματων LODs και UV generation και άμεση εισαγωγή χρόνου εκτέλεσης, ώστε να μπορούμε να ανανεώνουμε τις βιομηχανικές εφαρμογές αμέσως καθώς αλλάζουν τα δεδομένα σχεδίασης.

Το PiXYZ STUDIO είναι μια διαδραστική εφαρμογή προετοιμασίας δεδομένων που μας βοηθά στην εισαγωγή και μετατροπή βαρέων και σύνθετων δεδομένων 3D CAD σε έτοιμο προς χρήση περιεχόμενο μέσα στην Unity, να απλοποιούμε και βελτιστοποιούμε τα δεδομένα CAD γρήγορα και αποτελεσματικά, με περισσότερους από 120 ισχυρούς αλγόριθμους, συμπεριλαμβανομένων της εξομάλυνσης, της διόρθωσης τοπολογίας CAD, της αποδεκατισμό και της παραγωγής UV και αυτοματοποιήσουμε τις σύνθετες εργασίες προετοιμασίας δεδομένων με δέσμες ενεργειών Python.

Το PiXYZ REVIEW είναι ένας υψηλής ποιότητας πρόγραμμα προβολής CAD με προηγμένες δυνατότητες συνεργασίας για την ανταλλαγή και αλληλεπίδραση με εσωτερικές ομάδες, προμηθευτές και πελάτες με γρήγορη εισαγωγή σύνθετων συγκροτημάτων με πρόσβαση σε ολόκληρη την ιεραρχία προϊόντων, υποστήριξη για όλα τις κοινές μορφές CAD, συμπεριλαμβανομένων των CATIA, NX, SOLIDWORKS και JT και υποστηρίζει πλέον τις πιο διαδεδομένες συσκευές AR και VR για επιβλητική συνεργασία.

2.3.9. Unity developing tools

Το Unity κατασκευάζει μια σειρά από εργαλεία, μηχανικές agnostic υπηρεσίες και υποδομή που απαιτείται για τη δημιουργία παιχνιδιών πολλών χρηστών και για το οποιοδήποτε επίπεδο κλίμακας για Cloud-enabled εμπειρίες.

Επίσης, μπορούμε να βασιστούμε σε ελαστικές και κλιμακούμενες λύσεις υποδομής για υποστήριξη οποιουδήποτε παιχνιδιού σε κλίμακα, που φιλοξενείται σε ένα παγκόσμιο δίκτυο bare-metal και cloud servers, να παρέχουμε την καλύτερη εμπειρία παίκτη με τη διαχείριση της φιλοξενούμενης λύσης της Vivox για συνομιλία με φωνή και κείμενο που μπορεί εύκολα να ενσωματωθεί μέσω SDK ή plug-ins και λειτουργεί με οποιαδήποτε μηχανή ή πλατφόρμα, να διαχειριστούμε και ενορχηστρώσουμε την υποδομή μας με ενσωματωμένο matchmaking, συνδέοντας τους παίκτες μας με τους πόρους του cloud και να επωφεληθούμε από τον άκαμπτο, αποδοτικό κώδικα δικτύωσης που γίνεται ρητά για παιχνίδια με γρήγορο ρυθμό. Είναι διαφανές και αρθρωτό για εύκολη βελτιστοποίηση. Ένας ασύγχρονος χρόνος εκτέλεσης διακομιστών αυξάνει την απόδοση και την αποδοτικότητα των διακομιστών των παιχνιδιών μας.

Τα Unity Teams μας επιτρέπει να δημιουργήσουμε μαζί με το να διατηρούμε ένα πλήρες ιστορικό του έργου μας και μπορούμε να κάνουμε αναίρεση αλλαγών και επαναφορά στις προηγούμενες εκδόσεις ανάλογα με τις ανάγκες και να εξοικονομήσουμε χρόνο εξορθολογίζοντας τον τρόπο με τον οποίο δημιουργούμε και διανέμουμε κατασκευές σε ολόκληρη την ομάδα μας. Οι Collaborate λειτουργίες διευκολύνουν τις ομάδες Unity να αποθηκεύουν, να μοιράζονται και να συγχρονίζουν το έργο τους με άλλους. και να ανεβάσουμε ολόκληρο το έργο μας στο σύννεφο, οπότε δημιουργείται αντίγραφο ασφαλείας και είναι προσβάσιμο από οπουδήποτε το cloud είναι ενεργοποιημένο, ενσωματωμένο απευθείας στην Unity και έχει μια απλοποιημένη ροή εργασιών που είναι εύκολη στη χρήση, ανεξάρτητα από την τοποθεσία.

Το Cloud Build καθιστά απλό και εύκολο να δημιουργήσουμε και να μοιραστούμε τις κατασκευές του παιχνιδιού μας. Συγκεντρώνει αυτόματα, αναπτύσσει και δοκιμάζει το παιχνίδι μας, ώστε να μπορούμε να επαναλάβουμε γρήγορα με την ομάδα μας. Η εγκατάσταση διαρκεί δευτερόλεπτα και λειτουργεί με το υπάρχον αποθετήριο ελέγχου πηγής.

Με αυτοματοποιημένες ροές εργασίας μπορεί και δημιουργεί αυτόματα διαδικασίες(builds) στο σύννεφο χρησιμοποιώντας λιγότερο χρόνο για να διατηρήσουμε,

Ταυτοχρόνως κατασκευάζει πολλά έργα σε πολλές πλατφόρμες για πολλούς πελάτες με τη δυνατότητα μεγέθυνσης καθώς δημιουργούμε περισσότερες διαδικασίες για περισσότερες περιπτώσεις. Ακόμη, μπορούμε να διανέμουμε παντού καθώς οι διαδικασίες αποθηκεύονται και φιλοξενούνται στο σύννεφο έχοντας εύκολη κοινή χρήση και ανάπτυξη διαδικασιών για τα μέλη της ομάδας.

Το Unity Cloud Diagnostics είναι μια σουίτα εργαλείων που υποστηρίζουν το cloud που μας βοηθούν αντιδρώντας σε πραγματικό χρόνο καταφέρνουμε να κρατάμε τους χρήστες μας ικανοποιημένους και εμπλεκόμενους. Μας βοηθά να παρακολουθούμε το τι αντιμετωπίζουν οι χρήστες μας, να απαντάμε σε ανατροφοδότηση και να διορθώνουμε γρήγορα προβλήματα.

Κάνει εύκολο για εμάς και τους δοκιμαστές μας να συλλέγουν και να μοιράζονται τις αναφορές crash και exception και τα σχόλια που δημιουργούν οι χρήστες, κατά την ανάπτυξη και μετά την εκκίνηση, και να κρατούν τους χρήστες ικανοποιημένους.

Εκτός από τις αναφορές σφαλμάτων, λαμβάνουμε και τις προτάσεις που παρέχονται από τον χρήστη, οι οποίες περιλαμβάνουν στιγμιότυπα οθόνης, αποθηκευμένα παιχνίδια, βίντεο ή οτιδήποτε άλλο, για να βοηθήσουμε στη διόρθωση και βελτίωση της εμπειρίας των χρηστών.

Ελέγχει αυτόματα τα crash και τα exceptions με τις αναφορές να συγκεντρώνονται και να παρουσιάζονται σε πίνακα εργαλείων σε πραγματικό χρόνο, ώστε να δίνουμε προτεραιότητα στα ζητήματα που επηρεάζουν περισσότερο τους πελάτες μας.

Μπορούμε να αποκτήσουμε πρόσβαση σε χρήσιμες πληροφορίες που θα μας βοηθήσουν να κάνουμε ταχύτερη αποσφαλμάτωση, συμπεριλαμβανομένων των ιχνών στοίβας, των ημερολογίων εντοπισμού σφαλμάτων, των πληροφοριών συσκευών, των στιγμιότυπων οθόνης και ακόμη και των προσαρμοσμένων συμβάντων που ορίζουμε και ενημερωνόμαστε μέσω των εργαλείων που χρησιμοποιούμε ήδη (Email, Slack, Discord, JIRA και πολλά άλλα).

Το Unity Analytics μας δίνει γρήγορη και εύκολη πρόσβαση σε σημαντικές πληροφορίες κάνοντας αναλύσεις σε πραγματικό χρόνο με live-ops analytics που μας βοηθούν να βελτιώσουμε την οικονομία μας στο παιχνίδι και την εμπειρία του παίκτη.

Πλήρες σύνολο λειτουργιών ζωντανού χειρισμού (αναλύσεις παιχνιδιών & παικτών, χάρτες θεμάτων, παρακολούθηση επιδόσεων) για την παρακολούθηση της δραστηριότητας του παίκτη και όντας ενσωματωμένο το δεν υπάρχει SDK. Με το Data Live-Ops της Unity, μπορούμε να παρέχουμε την καλύτερη δυνατή εμπειρία σε κάθε συσκευή προσαρμόζοντας τις ρυθμίσεις απόδοσης σε πραγματικό χρόνο. Μπορούμε να προσαρμόσουμε τις προσφορές μας για διαφορετικά τμήματα παικτών χωρίς αναδιάταξη. Και μπορούμε να τα κάνουμε όλα από έναν κεντρικό πίνακα ελέγχου Operate που μας δίνει πρόσβαση σε όλα τα χαρακτηριστικά βελτιστοποίησης.

Με τις Απομακρυσμένες Ρυθμίσεις, δεν χρειάζεται να ανακατανεμηθούμε σε ένα κατάστημα εφαρμογών ή να περιμένουμε εβδομάδες για τους παίκτες να ενημερώσουν την εφαρμογή μας. Μπορούμε να τροποποιήσουμε άμεσα το παιχνίδι μας για διαφορετικά τμήματα παικτών. Επιπλέον, οι Απομακρυσμένες Ρυθμίσεις περιλαμβάνουν μια πρόσθετη εφαρμογή Editor, ώστε να ξεκινήσουμε γρήγορα και εύκολα. Μπορούμε να παρακολουθήσουμε συγκεκριμένες συμπεριφορές παικτών στο παιχνίδι μας με τα Standard Events, που μας δίνουν τη δυνατότητα να λάβουμε γνώση σχετικά με σύνολα επιμέρους συμβάντων παικτών που σχετίζονται με την επιβίβαση, την εξέλιξη, την εμπλοκή, τη δημιουργία εσόδων και την πλοήγηση στην εφαρμογή μας. Με το εργαλείο ανίχνευσης συμβάντων στο Analytics, μπορούμε να εφαρμόσουμε τυποποιημένες εκδηλώσεις χωρίς κωδικό.

Ενσωματωμένη με το Unity, η δοκιμή A / B μας επιτρέπει να κάνουμε τις αποφάσεις ενημέρωσης παιχνιδιών με εμπιστοσύνη με βάση τα στατιστικά αποτελέσματα των πειραμάτων μας. Η δοκιμή A / B έχει σχεδιαστεί για να λειτουργεί άψογα με την Unity.

Το LiveStream σάς δίνει τη δυνατότητα να παρακολουθούμε προσαρμοσμένα συμβάντα που είναι σημαντικά για την επιτυχία του παιχνιδιού μας, όπως ο αντίκτυπος των προσφορών μας αμέσως μετά την κυκλοφορία. Παρέχει εύκολη κατανόηση των εκδηλώσεων μέσω γεωγραφικής χαρτογράφησης σε πραγματικό χρόνο.

Το API εξαγωγής ακατέργαστων δεδομένων μας δίνει πλήρη έλεγχο των δικών μας δεδομένων και του τρόπου με τον οποίο τα χρησιμοποιούμε. Έχουμε τη δυνατότητα να δημιουργήσουμε το δικό μας δίαυλο δεδομένων, εκτελέσουμε προσαρμοσμένα ερωτήματα ή ακόμα και συνδέσουμε τα ακατέργαστα δεδομένα στα αγαπημένα μας εργαλεία επιχειρηματικής ευφυΐας. Όλα χωρίς ενσωμάτωση SDK. Η Εξαγωγή ακατέργαστων δεδομένων μας παρέχει ταχύτερη και ευκολότερη πρόσβαση στα δεδομένα μας.

Το Unity δεν είναι απλά ένα μέρος για τη δημιουργία παιχνιδιών. Είναι επίσης μια πλατφόρμα στην οποία μπορούμε να λειτουργήσουμε και να δημιουργήσουμε έσοδα από το παιχνίδι μας για να εξασφαλίσουμε τη συνεχή επιτυχία του προσφέροντας ενσωματωμένες λύσεις για τη μεγιστοποίηση της επιτυχίας μας όπως διαφημίσεις & IAP.

Με κλίμακα και ακρίβεια σε όλες τις πηγές διαφημίσεων η Ενοποιημένη Δημοπρασία προσφέρει τα υψηλότερα έσοδα στους προγραμματιστές, ανεξάρτητα από το αν ο διαφημιζόμενος προέρχεται από την Unity ή από έναν από τους 40 άλλους συνεργάτες ζήτησης. Μπορούμε να ενσωματώσουμε φυσικά τυχόν μορφές - όπως βραβευμένο βίντεο, banners, παρενθετικά, εμπλουτισμένα μέσα, playables ή ακόμα και AR - σε εμπειρίες παιχνιδιών παικτών για την αύξηση των αποτελεσμάτων CPM. Με απλές ρυθμίσεις η ενσωμάτωση διαφημίσεων απευθείας στην πλατφόρμα ανάπτυξης της Unity είναι εύκολη.

Ακόμη και τα παιχνίδια που είναι χτισμένα σε άλλες πλατφόρμες μπορούν απλά να ενσωματωθούν με το Unity Monetization 3.0 για τη δημιουργία διατηρητικών εσόδων μέσω της βελτιστοποίησης της διάρκειας ζωής και της προηγμένης ανάλυσης. Το Unity διευκολύνει την υλοποίηση αγορών εντός της εφαρμογής και επιφανειακών προσφορών στο παιχνίδι μας, εξοικονομώντας μας σημαντικές ώρες εργασίας ενώ παράλληλα δημιουργούμε περισσότερα έσοδα.

Κάθε παίκτης είναι διαφορετικός, όμως οι περισσότερες λύσεις είτε αντιμετωπίζουν όλους τους παίκτες το ίδιο είτε στοχεύουν με βάση τις ομάδες. Καθορίζοντας τους στόχους μας στην αρχή της καμπάνιας και χρησιμοποιώντας τη δυναμική τιμολόγηση, μπορούμε να μειώσουμε την ανάγκη για χειροκίνητες βελτιστοποιήσεις και να διασφαλίσουμε ότι δαπανούμε αποτελεσματικά την επένδυσή μας. Με βάση την προβλεπόμενη συμπεριφορά εντός εφαρμογής σε επίπεδο μεμονωμένου παίκτη, το Audience Pinpointer διασφαλίζει ότι πληρώνουμε τη σωστή τιμή για κάθε παίκτη μέσω του δυναμικού CPI (κόστος ανά εγκατάσταση) βάσει των στόχων βελτιστοποίησης.

Το μοντέλο μηχανικής μάθησης της Unity χρησιμοποιεί αποτίμηση σε πραγματικό χρόνο του χρήστη κατά τη στιγμή της υποβολής αιτήματος διαφήμισης. Ο αλγόριθμος αποφασίζει για μια προσφορά για κάθε χρήστη βάσει ατομικής αποτίμησης. Το σύστημα ενημερώνεται συνεχώς με βάση την πραγματική συμπεριφορά των χρηστών και τις προτιμήσεις εκατομμυρίων χρηστών σε όλη την πλατφόρμα της Unity.

Μπορούμε να προωθήσουμε παιχνίδια στο χαρτοφυλάκιο παιχνιδιών μας και να αποκτήσουμε εύκολα νέους παίκτες με τη λύση Cross Promotion. Εισάγοντας τους παίκτες στα άλλα παιχνίδια μας, επωφελούμαστε από την αναγνώριση της μάρκας παίκτη και την αφοσίωση, ειδικά στο ίδιο είδος αφού επεκτείνουμε την αξία της ζωής τους (LTV) στο χαρτοφυλάκιό μας. Εξοικονομούμε χρόνο και πόρους εκμεταλλευόμενοι την εύχρηστη λύση του Unity αντί για χρονοβόρα εσωτερική εργασία και υποστήριξη. Το Cross Promotion είναι εύκολο να εγκατασταθεί στο ταμπλό αυτο-εξυπηρέτησής μας και δεν απαιτείται επιπλέον SDK και μπορούμε να δούμε όλα τα αποτελέσματά μας και ενισχύστε την επιτυχία της καμπάνιας με εξαιρετικά λεπτομερείς αναφορές αυτο-εξυπηρέτησης στον πίνακα ελέγχου Advertizer.

2.4. Unreal Engine

Εικόνα **2.3** Λογότυπο Unreal

Το Unreal Engine είναι μια μηχανή παιχνιδιού που αναπτύχθηκε από τους Epic Games, που παρουσιάστηκε για πρώτη φορά στο παιχνίδι Unreal του πρώτου προσώπου του 1998. Αν και αρχικά αναπτύχθηκε για shooters πρώτου προσώπου, έχει χρησιμοποιηθεί με επιτυχία σε διάφορα άλλα είδη, όπως stealth, παιχνίδια μάχης, MMORPGs και άλλα RPG. Με τον κώδικα γραμμένο σε C++, ο Unreal Engine διαθέτει υψηλό βαθμό φορητότητας και είναι ένα εργαλείο που χρησιμοποιείται σήμερα από πολλούς προγραμματιστές παιχνιδιών, καθώς είναι διαθέσιμο από την πηγή. Η πιο πρόσφατη έκδοση είναι η Unreal Engine 4, η οποία κυκλοφόρησε το 2014.

2.4.1. Unreal rendering

Το σύστημα απόδοσης στο Unreal Engine 4 είναι ένα ολοκαίνουργιο pipeline DirectX 11 που περιλαμβάνει αναβολική σκίαση, παγκόσμιο φωτισμό, φωτισμένη διαφάνεια και μετά την επεξεργασία, καθώς και προσομοίωση σωματιδίων GPU χρησιμοποιώντας πεδία διανυσμάτων.

Χρησιμοποιώντας την αναβαλλόμενη σκίαση όλα τα φώτα εφαρμόζονται ανασταλτικά στο Unreal Engine 4, σε αντίθεση με τη διαδρομή φωτισμού προς τα εμπρός που χρησιμοποιείται στο Unreal Engine 3. Τα υλικά γράφουν τα χαρακτηριστικά τους στα GBuffers και οι φωτισμοί διαβάζονται στις ιδιότητες του υλικού ανά εικονοστοιχείο και εκτελούν φωτισμό μαζί τους.

Υπάρχουν τρεις διαδρομές φωτισμού στο UE4, όπως η πλήρως δυναμική με κινητά φώτα, μερικώς στατική με στατικά φώτα και η πλήρως στατική με στατικά φώτα.

Αυτά είναι εργαλεία με διαφορετικές επιλογές μεταξύ ποιότητας, απόδοσης και μεταβλητότητας κατά τη διάρκεια του παιχνιδιού. Κάθε παιχνίδι μπορεί να επιλέξει ποια διαδρομή είναι η πιο κατάλληλη για τις ανάγκες τους.

Με την ακτινοβολία να φωτίζεται και να σκιάζεται σε ένα μόνο πέρασμα προς τα εμπρός, διασφαλίζει τη σωστή ανάμιξη με άλλη διαφάνεια, κάτι που δεν είναι εφικτό με τις τεχνικές φωτισμού πολλαπλών καναλιών και τα υλικά έχουν ένα νέο μοντέλο σκίασης MLM_Subsurface το οποίο προορίζεται για υλικά όπως κερί ή νεφρίτη που φαίνονται αδιαφανή, αλλά ο φωτισμός διασκορπίζεται μέσα τους.

Το UE4 υποστηρίζει την προσομοίωση σωματιδίων στη GPU. Τα παραδοσιακά συστήματα CPU επιτρέπουν χιλιάδες σωματίδια σε ένα πλαίσιο. Η προσομοίωση GPU επιτρέπει την προσομοίωση εκατοντάδων χιλιάδων σωματιδίων και την αποδοτική λειτουργία τους.

Το πιο ενδιαφέρον χαρακτηριστικό των σωματιδίων GPU, εκτός από την αποτελεσματικότητά τους, είναι τα πεδία διάνυσμα. Ένα πεδίο διάνυσμα είναι ένα ομοιόμορφο πλέγμα των φορέων που επηρεάζει την κίνηση των σωματιδίων. Τα πεδία διανυσμάτων τοποθετούνται στον κόσμο ως ηθοποιούς και μπορούν να μεταφραστούν, να περιστραφούν και να κλιμακωθούν όπως κάθε άλλος ηθοποιός. Είναι δυναμικές και μπορούν να μετακινηθούν ανά πάσα στιγμή.

Ένα πεδίο μπορεί επίσης να τοποθετηθεί μέσα στο Cascade, περιορίζοντας την επιρροή του στον εκπομπό με τον οποίο συσχετίζεται. Όταν ένα σωματίδιο εισέρχεται στα όρια του πεδίου διανύσματος, η κίνηση του θα επηρεαστεί από αυτό και όταν ένα σωματίδιο φεύγει από τα όρια η επιρροή του πεδίου θα εξασθενίσει.

Επίσης, το Unreal Engine 4 παρέχει διάφορα εφέ μετά την επεξεργασία για να επιτρέψει σε καλλιτέχνες και σχεδιαστές να τροποποιήσουν τη συνολική εμφάνιση και αίσθηση της σκηνής. Παραδείγματα στοιχείων και αποτελεσμάτων περιλαμβάνουν άνθηση (επίδραση άνθησης HDR σε φωτεινά αντικείμενα), απόφραξη περιβάλλοντος και χαρτογράφηση τόνων.

Περιέχει το φαινόμενο Ambient Occlusion ,που είναι μια εφαρμογή SSAO (Screen Space Ambient Occlusion) και βασίζεται επί του παρόντος μόνο στο buffer βάθους. Αυτό σημαίνει ότι οι κανονικές λεπτομέρειες του χάρτη ή οι ομάδες εξομάλυνσης δεν επηρεάζουν τα αποτελέσματα. Τα πολύ χαμηλά πολυ - πλέγματα ενδέχεται

να εμφανίζονται πιο γωνιακά με το εφέ ενεργοποιημένο. Στο UE4, η επίδραση εφαρμόζεται μόνο στο περιβάλλον, το οποίο σημαίνει μόνο για το Ambient Cubemap.

Το φαινόμενο Ambient Cubemap εφαρμόζει μια υφή κορμού στο φωτισμό ολόκληρης της σκηνής. Αυτή η επίδραση είναι ανεξάρτητη από τη θέση από την οποία ανάβει ένα υλικό. Η θέση του θεατή, η τραχύτητα του υλικού (για τις κατοπτρικές επιδράσεις) και η κανονική επιφάνεια του υλικού λαμβάνονται όλα υπόψη. Αυτό επιτρέπει αποτελεσματικά και υψηλής ποιότητας περιβάλλοντα φωτισμού.

Έχοντας το Bloom, που είναι ένα φαινόμενο φωτισμού πραγματικού κόσμου, μπορεί να προσθέσει σε μεγάλο βαθμό στον αντιληπτό ρεαλισμό μιας παραγόμενης εικόνας σε ένα μέτριο κόστος απόδοσης. Το Bloom μπορεί να το δούμε με γυμνό μάτι όταν βλέπουμε πολύ φωτεινά αντικείμενα που βρίσκονται σε πολύ πιο σκούρο φόντο. Ακόμα φωτεινότερα αντικείμενα προκαλούν και άλλα εφέ (ραβδώσεις, φωτοβολίδες φακών), αλλά αυτές δεν καλύπτονται από το κλασικό Bloom εφέ. Επειδή οι οθόνες (π.χ. τηλεόραση, TFT, ...) συνήθως δεν υποστηρίζουν HDR (υψηλή δυναμική εμβέλεια), δεν μπορούμε πραγματικά να αποδίδουμε πολύ φωτεινά αντικείμενα. Αντ' αυτού, προσομοιώνουμε τα αποτελέσματα που συμβαίνουν στο μάτι (θωράκιση υπογείων επιφανειών του αμφιβληστροειδούς), όταν το φως αγγίζει το φιλμ (σκέδαση κάτω από την επιφάνεια του φιλμ) ή μπροστά από την κάμερα (γαλακτώδες γυάλινο φίλτρο). Το αποτέλεσμα μπορεί να μην είναι πάντα φυσικά σωστό, αλλά μπορεί να συμβάλει στο να υπονοήσει τη σχετική φωτεινότητα των αντικειμένων ή να προσθέσει ρεαλισμό στην εικόνα LDR (χαμηλή δυναμική περιοχή) και το εφέ Bloom Dirt Mask χρησιμοποιεί μια υφή για να φωτίζει την άνθηση σε μερικές καθορισμένες περιοχές οθόνης. Αυτό μπορεί να χρησιμοποιηθεί για να δημιουργήσουμε μια κάμερα πολέμου, πιο εντυπωσιακό εφέ HDR ή ατέλειες κάμερας.

Με τη προσαρμογή των ματιών ή την αυτόματη έκθεση, προκαλεί την αυτόματη προσαρμογή της έκθεσης της σκηνής για να αναδημιουργήσει το εφέ που παρατηρείται καθώς προσαρμόζονται τα ανθρώπινα μάτια κατά τη μετάβαση από ένα φωτεινό περιβάλλον σε ένα σκοτεινό περιβάλλον ή το αντίστροφο και το φαινόμενο Φωτεινός φακός είναι μια τεχνική που βασίζεται στην εικόνα και δημιουργεί αυτόματα φλόγα φακών κατά την προβολή φωτεινών αντικειμένων.

Ακόμη, η χαρτογράφηση τόνων επιτρέπει τη μετατόπιση ή την τροποποίηση των χρωμάτων της επεξεργασμένης σκηνής για να αλλάξει το τελικό αποτέλεσμα. Αυτό μπορεί να χρησιμοποιηθεί για τη δημιουργία εφέ όπως ένα φίλτρο σέπια, τα εφέ χτυπήματος (δηλ. ένα κόκκινο φλας) και πολλά άλλα και το εφέ Vignette προκαλεί τη μείωση της φωτεινότητας της rendered σκηνής όσο αυξάνεται η απόσταση από το κέντρο του Viewport.

2.4.2. Unreal Editor

Το Sequencer Editor είναι το κινηματογραφικό εργαλείο επεξεργασίας μέσα στο Unreal Engine 4. Ο Sequencer χρησιμοποιεί διάφορα εξειδικευμένα Tracks τα οποία χρησιμοποιούνται για τον ορισμό των σκηνών μας.

Προσθέτοντας δύο κομμάτια σκελετικών mesh στον Sequencer και έπειτα προσθέτοντας Trace Animation για να ζωντανέψουμε αυτά τα σκελετικά mesh και τοποθετώντας μια διαδρομή κάμερας και κοψίματα κάμερας για να δώσουμε μια προοπτική και κύκλο ανάμεσα στις κάμερες και έχουμε τα θεμέλια για τη δημιουργία μιας κινηματογραφικής ακολουθίας.

Το Unreal Motion Graphic UI Designer (UMG) είναι ένα οπτικό εργαλείο συγγραφής UI το οποίο μπορεί να χρησιμοποιηθεί για τη δημιουργία στοιχείων UI όπως HUDs, μενού ή άλλα γραφικά που σχετίζονται με τη διεπαφή που θέλουμε να παρουσιάσουμε στους χρήστες μας. Στον πυρήνα του UMG είναι τα Widgets, τα οποία είναι μια σειρά προκατασκευασμένων λειτουργιών που μπορούν να χρησιμοποιηθούν για την κατασκευή της διεπαφής μας (πράγματα όπως κουμπιά, πλαίσια ελέγχου, ρυθμιστικά, γραμμές προόδου κ.λπ.). Αυτά τα Widgets εκδίδονται σε ένα εξειδικευμένο Widget Blueprint, το οποίο χρησιμοποιεί δύο καρτέλες για την κατασκευή, η καρτέλα Designer επιτρέπει την οπτική διάταξη της διεπαφής και των βασικών λειτουργιών ενώ η καρτέλα Graph παρέχει τη λειτουργικότητα πίσω από τα Widgets που χρησιμοποιούνται.

Με το σύστημα κίνησης στο Unreal Engine 4 (UE4), που αποτελείται από διάφορα Εργαλεία και Επεξεργαστές Ζωγραφικής, αναμιγνύουν τη σκελετική παραμόρφωση των ματιών με παραμορφώσεις κορυφών που βασίζονται σε morph για να επιτρέψουν την πολύπλοκη κινούμενη εικόνα. Αυτό το σύστημα μπορεί να χρησιμοποιηθεί για να γίνει πιο ρεαλιστική η κίνηση των βασικών παικτών παίζοντας και συνδυάζοντας τις κονσέρβες Animation Sequences, δημιουργώντας εξατομικευμένες ειδικές κινήσεις όπως κλιμακοστάσια και τοίχους χρησιμοποιώντας Anim Montages, εφαρμόζοντας φθορές ή εκφράσεις προσώπου μέσω Morph Targets, μετασχηματισμούς των οστών χρησιμοποιώντας το Σκελετικό Έλεγχο ή δημιουργώντας Λογικές Κρατικές Μηχανές που καθορίζουν ποια κινούμενη εικόνα θα πρέπει να χρησιμοποιήσει ένας χαρακτήρας σε μια δεδομένη κατάσταση.

Το Plugin Composure που υπάρχει μαζί με τον Unreal Engine 4 (UE4) μπορεί να χρησιμοποιηθεί για να μετατρέψει τον UE4 Editor μας σε πρόγραμμα σύνθεσης σε πραγματικό χρόνο, το οποίο μας επιτρέπει να αναμειγνύουμε υλικό που δημιουργείται από τον υπολογιστή με ταινίες πραγματικού κόσμου, όπως ταινίες και τηλεοπτικές εκπομπές. Από τη δυνατότητα αντιστοίχισης της παραμόρφωσης φακού κάμερας ώστε να είναι σε θέση να προσδιορίσει τη διαφορά μεταξύ των αντικειμένων στο προσκήνιο και το φόντο, η λειτουργία Composure λειτουργεί όμοια με οποιοδήποτε άλλο λογισμικό σύνθεσης, με τη μεγαλύτερη διαφορά ότι η Composure κάνει τα πάντα σε πραγματικό χρόνο ενώ ο επεξεργαστής Niagara είναι ένα από τα δύο εργαλεία που μπορούμε να χρησιμοποιήσουμε για να δημιουργήσουμε και να προσαρμόσουμε οπτικά εφέ μέσα στο Unreal Engine 4 (UE4). Πριν από το Niagara, ο πρωταρχικός τρόπος για να δημιουργήσουμε και να επεξεργαστούμε οπτικά εφέ στο UE4 ήταν να χρησιμοποιήσουμε το Cascade. Ενώ το Niagara έχει πολλές από τις ίδιες μεθόδους χειρισμού σωματιδίων που προσφέρει το Cascade, ο τρόπος με τον οποίο αλληλεπιδράμε και δημιουργούμε οπτικά εφέ με το Niagara είναι πολύ διαφορετικός.

2.4.3. Unreal Sound Cue Editor

Το σύστημα ήχου αποτελείται από διάφορα στοιχεία, το καθένα από τα οποία συνεργάζεται για να παράγει την εμπειρία ήχου για τους παίκτες. Όταν εισάγουμε ένα αρχείο ήχου στον κινητήρα και το αφήσουμε σε ένα επίπεδο, θα έχουμε αρκετές επιλογές, όπως τα βασικά επίπεδα έντασης ή κλίσης για ρύθμιση, καθώς και περισσότερες ρυθμίσεις λεπτού συντονισμού, όπως η εξασθένηση ήχου, η οποία καθορίζει τον τρόπο με τον οποίο ένας ήχος ακούγεται με βάση την απόσταση μας από την προέλευσή της.

Επιτρέπει επίσης την κατασκευή σύνθετων ήχων με τη μορφή Sound Cues και του Sound Cue Editor, οι οποίοι μας επιτρέπουν να συνδυάσουμε τους ήχους καθώς και να εφαρμόσουμε τροποποιητές που ονομάζονται Κόμβοι Ήχων για να αλλάξουμε την τελική έξοδο. Η συμπεριφορά της αναπαραγωγής ήχου ορίζεται στο Sound Cues. Ο επεξεργαστής Sound Cue είναι ένας επεξεργαστής που βασίζεται σε κόμβους και χρησιμοποιείται για την εργασία με ήχο. Η έξοδος ήχου του συνδυασμού κόμβων που δημιουργήθηκε στον επεξεργαστή Sound Cue αποθηκεύεται ως Sound Cue.

2.4.4. Unreal AI

Τα δέντρα συμπεριφοράς είναι ένα ισχυρό εργαλείο για τη δημιουργία τεχνητής νοημοσύνης στο Unreal Engine 4 και είναι ένας συνδυασμός δύο τύπων asset, ο Blackboard και το Tree Behavior.

Είναι οδηγούμενα από τα γεγονότα και αποφεύγουν να δουλεύουν πολύ κάθε καρέ. Αντί να ελέγχουν συνεχώς αν έχει υπάρξει κάποια σχετική αλλαγή, τα δέντρα συμπεριφοράς απλώς παθητικά ακούν για "συμβάντα" που μπορούν να προκαλέσουν αλλαγές στο δέντρο. Η αρχιτεκτονική που βασίζεται σε συμβάντα παρέχει βελτιώσεις τόσο στην απόδοση όσο και στην εκσφαλμάτωση. Ωστόσο, για να επωφεληθούμε από αυτές τις βελτιώσεις, θα πρέπει να κατανοήσουμε τις άλλες διαφορές της UE4 και να διαμορφώσουμε κατάλληλα τα δέντρα συμπεριφοράς μας.

Δεδομένου ότι ο κώδικας δεν χρειάζεται να επαναλάβει ολόκληρο το δέντρο κάθε δευτερόλεπτο, η απόδοση είναι πολύ καλύτερη. Όταν σπάζουμε προς τα εμπρός και προς τα πίσω το ιστορικό εκτέλεσης του δέντρου συμπεριφοράς για να διορθώσουμε οπτικά τη συμπεριφορά, είναι ιδανικό να κάνουμε το ιστορικό να δείχνει σχετικές αλλαγές και να μην εμφανίζονται άσχημες. Στην υλοποίηση που βασίζεται σε γεγονότα, δεν είναι απαραίτητο να φιλτραριστούν άσχετα βήματα που επαναλαμβάνονται στο δέντρο και επιλέγουν την ίδια συμπεριφορά όπως πριν, διότι αυτή η επιπρόσθετη επανάληψη δεν έπρεπε ποτέ να συμβεί στην πρώτη θέση. Αντ' αυτού, μόνο αλλαγές στη θέση εκτέλεσης στο δέντρο ή στις τιμές μαυροπίνακα ουσία, και είναι εύκολο να δείξει ακριβώς αυτές τις διαφορές.

Το σύστημα Environment Query είναι ένα χαρακτηριστικό του συστήματος Τεχνητής Νοημοσύνης στο Unreal Engine 4 για τη συλλογή δεδομένων σχετικά με το περιβάλλον, την αποστολή ερωτημάτων μέσω των Δοκιμών και στη συνέχεια επιστροφή ενός στοιχείου που ταιριάζει καλύτερα στις ερωτήσεις που τέθηκαν.Το Περιβαλλοντικό Query Testing System (EQSTestingPawn) είναι μια εξειδικευμένη κατηγορία Pawn που μας δίνει τη δυνατότητα να ελέγξουμε τι κάνουν πραγματικά τα Environment Queries μας. Το ακριβές σχέδιο του Environment Query μας θα καθορίσει το μέγεθος και το σχήμα του τι δημιουργείται, αλλά θα

αντιπροσωπεύεται πάντοτε ως έγχρωμες σφαίρες. Οι σφαίρες που βρίσκονται στη χρωματική κλίμακα από πράσινο έως κόκκινο, υποδεικνύουν κάποιο επίπεδο αντιστοίχισης για τις διάφορες δοκιμές που διεξήχθησαν στο περιβάλλον ερώτημά μας. Οι μπλε σφαίρες υποδεικνύουν μια αποτυχία ή μια δοκιμή τύπου boolean που επέστρεψε ψευδής.

2.4.5. Unreal physics

Με τη μηχανή physics PhysX 3.3, το Unreal κατευθύνει τους υπολογισμούς φυσικής προσομοίωσης και εκτελεί όλους τους υπολογισμούς συγκρούσεων. Το PhysX παρέχει τη δυνατότητα να πραγματοποιεί ακριβή ανίχνευση σύγκρουσης καθώς και να προσομοιώνει τις φυσικές αλληλεπιδράσεις μεταξύ αντικειμένων στον κόσμο. Η ύπαρξη φυσικής στο παιχνίδι μας θα βοηθήσει στη βελτίωση της αξίας εμβάπτισης κάθε σκηνής, καθώς βοηθά τους παίκτες να πιστεύουν ότι αλληλοεπιδρούν με τη σκηνή και ότι η σκηνή αποκρίνεται με κάποιο τρόπο.

Οι Responses σύγκρουσης και οι Trace Responses αποτελούν τη βάση για το πως η μηχανή χειρίζεται τη σύγκρουση και τη χύτευση ακτινών κατά τη διάρκεια του χρόνου εκτέλεσης. Κάθε αντικείμενο που μπορεί να συγκρουστεί παίρνει έναν τύπο αντικειμένου και μια σειρά απαντήσεων που καθορίζουν πώς αλληλοεπιδρά με όλους τους άλλους τύπους αντικειμένων. Όταν συμβαίνει ένα συμβάν σύγκρουσης ή επικάλυψης, τα δύο (ή όλα) αντικείμενα που εμπλέκονται μπορούν να ρυθμιστούν ώστε να επηρεάζουν ή να επηρεάζονται από το κλείδωμα, την επικάλυψη ή την αδιαφορία μεταξύ τους.

Χρησιμοποιώντας τα Traces (ή Raycasts) για να ρίξουμε μια αόρατη ακτίνα που θα ανιχνεύσει τη γεωμετρία μεταξύ δύο σημείων και εάν χτυπήσει τη γεωμετρία, να επιστρέψει αυτό που χτυπήθηκε, ώστε να κάνουμε κάτι με αυτό.

Υπάρχουν διάφορες διαθέσιμες επιλογές όταν εκτελούμε ένα Trace. Μπορούμε να εκτελέσουμε ένα Trace για να ελέγξουμε για σύγκρουση με οποιαδήποτε Αντικείμενα όπου έχουν επιστρέψει αντικείμενα που χτυπήθηκαν ή μπορούμε να εκτελέσουμε ένα Trace by Trace Channel όπου οποιοσδήποτε Αντικείμενο χτύπησε θα επιστρέψει τις πληροφορίες χτυπήματος εάν το Object έχει ρυθμιστεί να ανταποκρίνεται συγκεκριμένα σε ένα συγκεκριμένο Trace Channel όπου μπορεί να οριστεί μέσω Ρυθμίσεων σύγκρουσης.

Εκτός από τα Trace που τρέχουν από τα Αντικείμενα ή από το Trace Channel, μπορούμε να εκτελέσουμε το Trace για να ανιχνεύσουμε μονά ή πολλαπλά χτυπήματα, όπου ένα Single Trace επιστρέφει ένα μοναδικό αποτέλεσμα χτυπήματος και ένα Multi Trace επιστρέφει πολλαπλά αποτελέσματα που προκύπτουν από το Trace.

Με τα Traces, μπορούμε επίσης να καθορίσουμε τον τύπο της ακτίνας που χρησιμοποιείται π.χ. μια ευθεία γραμμή, ένα κουτί, μια κάψουλα ή μια σφαίρα. Οι Trace Responses λειτουργούν βασικά με τον ίδιο τρόπο που λειτουργούν οι Responses σύγκρουσης, εκτός από το γεγονός ότι το ίχνος (ray cast) μπορεί να οριστεί ως ένας από τους τύπους Trace Response, επιτρέποντας έτσι στους Actors να το μπλοκάρουν ή να το αγνοούν με βάση τις απαντήσεις Trace τους.

Επίσης, η NVIDIA παράγει ένα εργαλείο που ονομάζεται APEX PhysX Lab το οποίο μπορεί να χρησιμοποιηθεί για τη δημιουργία καταστρεπτικών meshes. Ωστόσο, θα πρέπει να σημειωθεί ότι σήμερα ο Unreal Engine 4 υποστηρίζει μόνο APEX Destructibles και APEX Cloth. Για να δημιουργήσουμε στοιχεία APEX Cloth απαιτείται η χρήση του plug-in PhysX: Maxs / Maya DCC ή το αυτόνομο εργαλείο περιλαμβάνεται στο APEX SDK.

Καταστρέψιμοι Actors που μπορούν να δημιουργηθούν μέσα στο Unreal Editor 4 χρησιμοποιώντας Voronoi με ένα ενιαίο επίπεδο βάθους. Τα καταστρεφόμενα assets APEX με περισσότερα από ένα επίπεδα βάθους ή με τη χρήση μιας λειτουργίας διαφορετικής από Voronoi για τη θραύση του πλέγματος, πρέπει να δημιουργούνται επί του παρόντος στο εργαστήριο APEX PhysX ενώ τα εργαλεία Open World μας επιτρέπουν να τοποθετήσουμε στατικά πλέγματα μέσα στα επίπεδα μας χρησιμοποιώντας έναν αλγόριθμο υπολογιστή για να καθορίσουμε την τοποθέτηση αντί να κάνουμε αυτό το έργο χειροκίνητα. Αυτό μας δίνει τη δυνατότητα να δημιουργήσουμε γρήγορα ένα φυσικό τοπίο που αποτελείται από πολλά διαφορετικά είδη δένδρων, φυτών και θάμνων.

Όταν μιλάμε για τα εργαλεία Open World, μιλάμε για τα δύο διαφορετικά σύνολα εργαλείων μέσα στο UE4. Το σετ εργαλείων που χρησιμοποιούμε εξαρτάται από το τι θέλουμε να ολοκληρώσουμε. Για παράδειγμα, εάν θέλουμε να δημιουργήσουμε ένα εκτεταμένο δάσος, τότε θα θέλουμε να χρησιμοποιήσουμε το Εργαλείο διαδικασιών φύλλων. Αν θέλουμε να γεμίσουμε πυκνά το τοπίο εδάφους που χρησιμοποιείται σε αυτό το δάσος τότε θα χρησιμοποιούσαμε το εργαλείο χόρτου.

Το Εργαλειο Procedural Foliage χρησιμοποιείται για την προσομοίωση τεράστιων δασών που είναι γεμάτα με πολλά διαφορετικά είδη δέντρων και θάμνων. Αυτό το εργαλείο λειτουργεί με προσομοίωση του τρόπου με τον οποίο αυξάνεται το δάσος κατά τη διάρκεια των ετών χρησιμοποιώντας τα βήματα για να καθορίσουμε τα χρόνια ανάπτυξης. Σε κάθε βήμα της προσομοίωσης, οι νέοι εικονικοί σπόροι φυτεύονται στο επίπεδο για να λειτουργούν ως τοποθεσίες γεννήσεων για άλλους Foliage Actors. Αλλάζοντας τις διάφορες επιλογές Τύπου φύλλων, όπως το Spread, το Μέγεθος και τις Ρυθμίσεις Προτεραιότητας, μπορούμε να επηρεάσουμε τον τρόπο με τον οποίο οι Τύποι Φυλλώματος εξαπλώνονται και ανταγωνίζονται για τις θέσεις ωοτοκίας μέσω του επιπέδου στο οποίο έχει τοποθετηθεί ο Procedural Foliage Spawner.

Το εργαλείο χόρτου χρησιμοποιείται για την πυκνή κάλυψη του εδάφους τοπίου με πράγματα όπως γρασίδι, λουλούδια και μικρούς βράχους. Μπορούμε να χρησιμοποιήσουμε το εργαλείο χόρτου για να ορίσουμε ένα στατικό πλέγμα που θα δημιουργηθεί όταν ένα τοπικό στρώμα τοπίου είναι ζωγραφισμένο στο έδαφος του τοπίου. Αυτό το εργαλείο δεν χρησιμοποιεί κανένα είδος προσομοίωσης όπως το Procedural Foliage Spawner. Αντ' αυτού, απλώς γεμίζει την καθορισμένη περιοχή στην καθορισμένη πυκνότητα με το καθορισμένο στατικό πλέγμα.

Το σύστημα Τοπίου μας δίνει τη δυνατότητα να δημιουργήσουμε το έδαφος για τον κόσμο μας - βουνά, κοιλάδες, άνισα ή κεκλιμένα εδάφη, ακόμη και ανοίγματα για σπηλιές - και εύκολα να τροποποιήσουμε τόσο το σχήμα όσο και την εμφάνισή του χρησιμοποιώντας μια σειρά εργαλείων. Με το σύστημα Instanced Mesh

Foliage, μας επιτρέπει να ζωγραφίζουμε γρήγορα ή να διαγράφουμε σύνολα Στατικών Πλεγμάτων για τους Τοπικούς Actors, άλλους Static Mesh Actors και γεωμετρία BSP. Αυτά τα μάτια συσσωματώνονται αυτόματα σε παρτίδες που αποδίδονται με τη χρήση του instances υλικού, πράγμα που σημαίνει ότι πολλές περιπτώσεις μπορούν να αποτυπωθούν με μία μόνο κλήση κλήρωσης.

2.4.6. Unreal Level Streaming

Ακόμη, με τη λειτουργία streaming επιπέδου, μας καθιστά δυνατή τη φόρτωση και εκφόρτωση των αρχείων χαρτών στη μνήμη καθώς και την εναλλαγή της προβολής τους κατά τη διάρκεια της αναπαραγωγής. Αυτό καθιστά δυνατό τον διαχωρισμό των κόσμων σε μικρότερα κομμάτια, έτσι ώστε μόνο τα σχετικά μέρη του κόσμου να αναλαμβάνουν πόρους και να αποδίδονται σε οποιοδήποτε σημείο. Εάν γίνει σωστά, αυτό επιτρέπει τη δημιουργία πολύ μεγάλων, απρόσκοπτων παιχνιδιών που μπορούν να κάνουν τον παίκτη να αισθάνεται σαν να παίζει μέσα σε έναν κόσμο που νιώθει μικρός σε μέγεθος.

Το streaming επιπέδων με τον τύπο streaming Blueprint μπορούν να ελεγχθούν με ενότητες streaming επιπέδου, Blueprints ή κώδικα C ++. Αυτά τα επίπεδα μπορούν να φορτωθούν ή να εκφορτωθούν δυναμικά. Οι ενότητες streaming επιπέδου κάνουν πολύ εύκολη τον έλεγχο του streaming του επιπέδου. Η ιδέα είναι ότι οι αιτήσεις φόρτωσης / εκφόρτωσης για ένα streaming επίπεδο εκπέμπονται με βάση το αν η οπτική γωνία βρίσκεται μέσα σε οποιονδήποτε από τις ενότητες streaming επιπέδου που σχετίζονται με ένα επίπεδο. Συγκεκριμένα, οι ενότητες streaming επιπέδου μπορούν να χρησιμοποιηθούν με δύο τρόπους, στο παιχνίδι, οι ενότητες streaming επιπέδου προκαλούν φόρτωση των επιπέδων όταν η οπτική γωνία του παίκτη είναι μέσα στην ένταση και ξεφορτώνεται όταν η οπτική γωνία του παίκτη είναι εκτός της έντασης.

Στον editor, οι ενότητες streaming επιπέδου μπορούν να χρησιμοποιηθούν για την προεπισκόπηση της ροής του επιπέδου με την αυτόματη απόκρυψη / εξαφάνιση των επιπέδων με βάση τη θέση της φωτογραφικής μηχανής του προοπτικού προβολής. Το streaming επιπέδου με βάση ενότητες είναι απλό στη χρήση και δεν απαιτεί δέσμες ενεργειών, καθιστώντας τον ιδανικό τρόπο για να ελέγχουμε τη ροή στο επίπεδο.

Επιπλέον, είναι πολύ πιο εύκολο στη διατήρηση από το scripted streaming αφού όταν οι απαιτήσεις για το σύστημα φόρτωσης αλλάζουν, η συμπεριφορά φόρτωσης / εκφόρτωσης επιπέδου μπορεί να τροποποιηθεί απλά με αλλαγή μεγέθους των ενοτήτων streaming.

Υπάρχουν και τα Hierarchical Levels of Detail όπου στην απλούστερη μορφή τους, συνδυάζουν προϋπάρχοντες Static Mesh Actors σε έναν Static Mesh Actor και υλικό, με ατρακτοποιημένες υφές, για να μειώσουν τις κλήσεις κλήρωσης που με τη σειρά τους βοηθούν στην απόδοση του παιχνιδιού μας. Υπάρχει ένα ισχυρό σύστημα που μας δίνει τη δυνατότητα να ορίσουμε τι πρέπει να δημιουργηθεί όταν δημιουργούμε αυτούς τους HLOD Actors.

Με τα στοιχεία Blueprint Spline τα οποία είναι απλά διαδρομές για να ορίσουμε και να χρησιμοποιήσουμε δεδομένα θέσης. Μπορούμε να το χρησιμοποιήσουμε για να μετακινήσουμε τους Actors (ή άλλα Components)

σε όλο τον κόσμο ή να τοποθετήσουμε μια σειρά Actors (ή άλλων Components) κατά μήκος του spline. Είναι πλήρως επεξεργάσιμα στο Blueprint Viewport και στον Level Editor, με τη δυνατότητα προσθήκης / αφαίρεσης / αντιγραφής σημείων Spline, αλλαγής των εφαπτόμενων τύπων τους και ακόμη και της εμψύχωσής τους. Επιπλέον, μπορούν επίσης να επεξεργαστούν με τη χρήση του Blueprint Construction Script, λαμβάνοντας τις τροποποιήσεις που έγιναν στο Blueprint Viewport ή στο Level Editor, και βελτιώνοντάς τους περαιτέρω.Τα Blueprint Spline Mesh Components έχει μια εντελώς διαφορετική περίπτωση χρήσης. Αυτά παραμορφώνουν ένα ενιαίο στατικό πλέγμα κατά μήκος μιας σφήνας δύο σημείων. Δεν μπορούμε να προσθέσουμε περισσότερους πόντους Spline σε ένα στοιχείο Blueprint Spline Mesh, αλλά τα δύο σημεία είναι πλήρως ελεγχόμενα μέσω Blueprints.

2.4.7. Unreal Matinee

Το εργαλείο κινουμένων σχεδίων Matinee παρέχει τη δυνατότητα να ζωντανεύει τις ιδιότητες των Ηθοποιών με την πάροδο του χρόνου, για να δημιουργήσει δυναμικές διαδρομές παιχνιδιού ή κινηματογραφικές εν κινήσει. Το σύστημα βασίζεται στη χρήση εξειδικευμένων κομματιών κινούμενων εικόνων, στις οποίες μπορούμε να τοποθετήσουμε τα βασικά καρέ για να ορίσουμε τις τιμές ορισμένων ιδιοτήτων των Ηθοποιών στο επίπεδο. Ο επεξεργαστής Matinee είναι παρόμοιος με τους μη γραμμικούς επεξεργαστές που χρησιμοποιούνται για την επεξεργασία βίντεο, καθιστώντας τον εξοικειωμένο με τους επαγγελματίες του βίντεο.

Το Media Framework χρησιμοποιεί διάφορα στοιχεία για να επιτρέψει την αναπαραγωγή βίντεο στο Unreal Engine 4 (UE4). Τα βίντεο μπορούν να καθαριστούν, να τεθούν σε παύση ή να επανέλθουν μέσα σε ένα περιουσιακό στοιχείο Media Player, καθώς επίσης και να ελέγχονται μέσω C ++ ή Blueprints Visual Scripting. Είτε θέλουμε μια επιφάνεια στο επίπεδο μας να παίζει ένα βίντεο είτε θέλουμε να δημιουργήσουμε ένα στοιχείο UI με το UMG για να δώσουμε στους παίκτες τον έλεγχο της αναπαραγωγής του βίντεο, θα πρέπει πρώτα να καθορίσουμε την Πηγή Πολυμέσων έτσι ώστε ο Μηχανισμός να εντοπίσει το περιουσιακό μας υλικό.

2.4.8. Unreal XR

Οι εμπειρίες επαυξημένης πραγματικότητας καθίστανται όλο και πιο απαιτητικές για εφαρμογές κινηματογραφικών και ραδιοτηλεοπτικών μέσων αλλά το Unreal Engine μπορεί να λειτουργεί ως ένα απρόσκοπτο κομμάτι του pipeline παραγωγής βίντεο αφού καταφέρνει να μας κάνει να αναπαράγουμε βίντεο και ήχο επαγγελματικού επιπέδου ζωντανά, συνθέτοντας τα στον εικονικό 3D κόσμο απευθείας, να εφαρμόσουμε εφέ στο εισαγόμενο βίντεο απευθείας, όπως χρωματογραφία, αντικατάσταση φακού, διόρθωση χρώματος και πολλά άλλα, να συγχρονίσουμε το Engine με τον χρονομετρητή και τον ρυθμό καρέ του βίντεο εισόδου μας, για να εξαλείψουμε τα προβλήματα χρονισμού και να ρυθμίσουμε τις τροφοδοσίες βίντεο από το engine ή από το τρέχον έργο παιχνιδιών μας πίσω στο video pipeline του στούντιο μας.

Παντού υλοποίηση είναι ένα θέμα για την πραγματοποίηση παιχνιδιών σε πραγματικό χρόνο. Για να δημιουργήσουμε την ψευδαίσθηση των κινούμενων εικόνων, χρειαζόμαστε ρυθμό καρέ τουλάχιστον 15 καρέ

ανά δευτερόλεπτο. Ανάλογα με την πλατφόρμα και το παιχνίδι, 30, 60 ή ακόμα περισσότερα καρέ ανά δευτερόλεπτο μπορεί να είναι ο στόχος. Γι' αυτό το Unreal Engine παρέχει πολλές λειτουργίες και έχει διαφορετικά χαρακτηριστικά απόδοσης προκειμένου να βελτιστοποιηθεί το περιεχόμενο ή ο κώδικας για να επιτευχθεί η απαιτούμενη απόδοση, βλέποντας πού δαπανάται η απόδοση. Για το λόγο αυτό, μπορούμε να χρησιμοποιήσουμε τα εργαλεία δημιουργίας προφίλ κινητήρα. Κάθε περίπτωση είναι διαφορετική και απαιτούνται κάποιες γνώσεις σχετικά με τα εσωτερικά του υλικού και του λογισμικού.

2.4.9. Unreal developing tools

Στο κομμάτι του patching, όπου η διαδικασία περιλαμβάνει συνήθως νέο περιεχόμενο που αντιμετωπίζει γνωστά ζητήματα ή που επιλύει τρωτά σημεία στην αρχική έκδοση υπάρχουν διάφορες μέθοδοι για τη δημιουργία επιδιορθώσεων, αλλά το Engine χρησιμοποιεί μία από τις δύο προσεγγίσεις. Μια προσέγγιση διατηρεί τα αρχεία από την αρχική έκδοση ή τα προηγούμενα patches, αλλά προσθέτει ένα δείκτη στο νέο περιεχόμενο. Η άλλη προσέγγιση είναι να μετατρέψει το περιεχόμενο στην αρχική κατασκευή χρησιμοποιώντας μια δυαδική εμπλοκή.

Διαθέτει και ένα σύστημα Replay, το οποίο μπορεί να καταγράψει το gameplay για μεταγενέστερη προβολή. Αυτή η λειτουργία είναι διαθέσιμη σε όλα τα παιχνίδια, από ζωντανά, παιχνίδια για πολλούς παίκτες που παίζονται σε αποκλειστικούς διακομιστές, σε παιχνίδια με έναν παίκτη και ακόμη και σε περιόδους Play-In-Editor. Σε υψηλό επίπεδο, το σύστημα Replay λειτουργεί με τη χρήση ενός DemoNetDriver για την ανάγνωση δεδομένων που προέρχονται από το ενσωματωμένο σύστημα αναπαραγωγής, παρόμοιο με το πώς λειτουργεί το NetDriver σε ένα ζωντανό, δικτυακό περιβάλλον παιχνιδιού. Ακόμα κι αν ένα έργο δεν έχει λειτουργία πολλαπλών παικτών, οποιοδήποτε έργο που έχει ρυθμιστεί για την αναπαραγωγή δεδομένων είναι συμβατό με το Replay System χωρίς την ανάγκη περαιτέρω τροποποίησης.

Ο τρόπος με τον οποίο αυτό συνεπάγεται το DemoNetDriver που διαβιβάζει τα δεδομένα δικτύου σε ένα Replay Streamer, το οποίο χειρίζεται τη διαδικασία φιλτραρίσματος και αποθήκευσης των δεδομένων. Κατά την προβολή μιας επανάληψης, το DemoNetDriver θα έχει πρόσβαση σε όλες τις πληροφορίες αναπαραγωγής που ήταν διαθέσιμες κατά τη διάρκεια της ζωντανής αναπαραγωγής (καθώς και ειδικά πεδία δεδομένων που έχουν οριστεί ως συναφή μόνο για επαναλήψεις) ώστε να μπορούν να αναδημιουργήσουν τα συμβάντα του παιχνιδιού από αυτά τα δεδομένα.

Το framework UE4 κατασκευάζεται με βάση το παιχνίδι για πολλούς παίκτες. Εφόσον ακολουθούμε τις βασικές συμβάσεις-πλαίσιο, γενικά δεν χρειάζεται να κάνουμε πολλά για να επεκτείνουμε την εμπειρία ενός παίκτη στο multiplayer. Η δικτύωση UE4 βασίζεται στο μοντέλο διακομιστή / πελάτη. Αυτό σημαίνει ότι θα υπάρχει ένας διακομιστής που είναι έγκυρος (κάνει όλες τις σημαντικές αποφάσεις) και αυτός ο διακομιστής θα διασφαλίσει ότι όλοι οι συνδεδεμένοι πελάτες ενημερώνονται συνεχώς, ώστε να διατηρούν την πιο ενημερωμένη προσέγγιση του κόσμου του διακομιστή. Ακόμη και τα παιχνίδια που δεν συνδέονται σε δίκτυο,

για μεμονωμένους παίκτες έχουν διακομιστή. το τοπικό μηχάνημα ενεργεί ως διακομιστής σε αυτές τις περιπτώσεις.

Το χαρτί 2D στο Unreal Engine 4 (UE4) είναι ένα σύστημα βασισμένο σε sprite για τη δημιουργία υβριδικών παιχνιδιών 2D και 2D / 3D αποκλειστικά μέσα στον επεξεργαστή. Στον πυρήνα του 2D είναι τα Sprites (τα οποία είναι ένα Mapped Mesh Planar Mesh και το σχετικό Υλικό). Μπορούμε να επεξεργαστούμε τα Sprites μέσα στο UE4 με τον επεξεργαστή Sprite και να δημιουργήσουμε κινούμενα σχέδια βασισμένα σε sprite με Flipbooks (τα οποία animate μια σειρά Sprites διαδοχικά χρησιμοποιώντας πλαίσια κλειδιών και καθορίζοντας μια διάρκεια σε πλαίσια για να τα εμφανίσουμε).

Οι Distributions είναι μια ομάδα τύπων δεδομένων που παρέχουν ευελιξία επιτρέποντας σταθερές τιμές, τυχαίες τιμές εντός εύρους, τιμές που παρεμβάλλονται κατά μήκος μιας καμπύλης και τιμές που καθοδηγούνται από παραμέτρους. Αυτά χρησιμοποιούνται γενικά στις ιδιότητες των σωματιδιακών συστημάτων και των κόμβων στο SoundCues.

Η μετακίνηση ή η μετονομασία ενός asset στο UE4 αφήνει έναν Redirector στην παλιά του θέση. Αυτό είναι έτσι ώστε τα πακέτα που δεν φορτώνονται αυτή τη στιγμή, αλλά αναφέρονται σε αυτό το στοιχείο, θα ξέρουν πού να τα βρουν στη νέα τους θέση. Επιλέγοντας ένα σύστημα ονοματολογίας νωρίς και αν μείνουμε με αυτό, θα αποφευχθούν πολλά από τα προβλήματα που αντιμετωπίζονται με τους Redirectors.

Ο Unreal Engine 4 έρχεται με δύο ολοκληρωμένες μεθόδους που επιτρέπουν στους ανθρώπους να συνεργαστούν στα έργα τους χρησιμοποιώντας λογισμικό ελέγχου έκδοσης όπως το Perforce και το SVN. Ο έλεγχος έκδοσης επιτρέπει στους χρήστες της ομάδας μας να μοιράζονται στοιχεία και να κωδικοποιούν μεταξύ τους, καθώς και να παρέχουν αντίγραφα ασφαλείας και ιστορικό αλλαγών, ώστε οποιαδήποτε αρχεία να μπορούν να μεταφερθούν σε μια παλαιότερη έκδοση, εάν κάτι δεν πήγε σωστά στο αρχείο.

Το Derivative Data Cache (DDC) αποθηκεύει τις εκδόσεις των στοιχείων ενεργητικού στις μορφές που χρησιμοποιεί η Unreal Engine στις πλατφόρμες προορισμού της, σε αντίθεση με τις μορφές πηγών που δημιουργούν οι καλλιτέχνες που εισάγονται στο Editor και αποθηκεύονται σε αρχεία .uset.

Το περιεχόμενο που είναι αποθηκευμένο στο DDC είναι διαθέσιμο στο ότι μπορεί πάντα να αναγεννηθεί ανά πάσα στιγμή χρησιμοποιώντας τα δεδομένα που είναι αποθηκευμένα στο αρχείο .uset. Η αποθήκευση αυτών των προερχόμενων μορφών εξωτερικά καθιστά δυνατή την εύκολη προσθήκη ή αλλαγή των μορφών που χρησιμοποιεί ο κινητήρας χωρίς να απαιτείται τροποποίηση του αρχείου ενεργητικού προέλευσης. Τα Studios θα πρέπει να χρησιμοποιούν κοινόχρηστο DDC που έχουν πρόσβαση σε όλους τους χρήστες σε μια συγκεκριμένη τοποθεσία. Με αυτόν τον τρόπο, μόνο ένα άτομο πρέπει να δημιουργήσει τις παραγόμενες μορφές περιουσιακών στοιχείων και θα είναι αυτόματα διαθέσιμο σε όλους τους άλλους χρήστες. Περιστασιακά θα υπάρχουν πάγκοι όταν χρειάζεται η επεξεργασία των στοιχείων ενεργητικού, αλλά τα

αποτελέσματα αποθηκεύονται και μοιράζονται. Ακόμη και σε μια αρκετά μικρή ομάδα, η ανταλλαγή εργασιών επεξεργασίας περιουσιακών στοιχείων με αυτόν τον τρόπο θα μειώσει το χρόνο επεξεργασίας.

2.4.10. Unreal Geometry Proxy

Το σετ εργαλείων Geometry Proxy αναπτύχθηκε ως ένας τρόπος για να αυξήσουμε την απόδοση του έργου Unreal Engine 4 (UE4) διατηρώντας ταυτόχρονα την αληθινή ποιότητα του έργου μας. Ο στόχος του εργαλείου Proxy Geometry είναι να συμβάλει στη μείωση του κόστους εκτέλεσης του Static Meshes και των αντίστοιχων Υλικών και Υφασμάτων. Το εργαλείο μεσολάβησης γεωμετρίας επιτυγχάνει αυτό συνδυάζοντας πολλαπλά στατικά μάτια και τα αντίστοιχα υλικά τους σε ένα ενιαίο στατικό πλέγμα με ένα ενιαίο σύνολο υφασμάτων και υλικών που εξακολουθεί να ταιριάζει με το αρχικό σχήμα στατικού πλέγματος και να φαίνεται αλλά με μειωμένο αριθμό τριγώνων. Αυτό το μειωμένο αποτέλεσμα μπορεί να χρησιμοποιηθεί ως υποκατάστατο για την αρχική γεωμετρία σε περιπτώσεις όπου η διαφορά στην ποιότητα είναι είτε αποδεκτή είτε όχι, για παράδειγμα δομές που απέχουν από την κάμερα.

Το Εργαλείο Geometry Proxy θα δημιουργήσει επίσης ένα νέο σύνολο υφών που αντιστοιχούν στη νέα γεωμετρία στατικού πλέγματος που δημιουργήθηκε. Την πρώτη φορά που το σύστημα διακομιστή μεσολάβησης επεξεργάζεται ένα στοιχείο γεωμετρίας, μια κρυφή μνήμη shader στο νήμα του παιχνιδιού γεμίζει, προκαλώντας ένα κόστος της μιας φοράς. Αυτό σημαίνει ότι οι επακόλουθες επαναλήψεις (για παράδειγμα, αλλάζοντας κάποια παράμετρο και ανοικοδόμηση αυτού του διακομιστή μεσολάβησης) μπορεί να είναι πολύ πιο γρήγορα. Όταν συγκρίνεται με τις υπάρχουσες επιλογές τρίτου μέρους, παρατηρείται βελτίωση ταχύτητας 2 έως 3 φορές με αυτό το νέο σύστημα σε μέτρια γεωμετρικά συγκροτήματα χωρικών διαστάσεων, αλλά σε πολύ μεγάλη γεωμετρία ο χρόνος ολοκλήρωσης είναι παρόμοιος.

2.4.11. Unreal Blueprints

Το Σύστημα Visual Scripting Blueprints στο Unreal Engine είναι ένα πλήρες σενάριο παιχνιδιού που βασίζεται στην ιδέα της χρήσης μιας διασύνδεσης που βασίζεται σε κόμβο για να δημιουργήσει στοιχεία παιχνιδιού μέσα από τον Unreal Editor. Όπως συμβαίνει με πολλές κοινές γλώσσες δέσμης ενεργειών, χρησιμοποιείται για τον ορισμό αντικειμενοστραφών (ΟΟ) κλάσεων ή αντικειμένων στον κινητήρα.

Καθώς χρησιμοποιούμε το UE4, συχνά θα διαπιστώσουμε ότι τα αντικείμενα που έχουν οριστεί με το Blueprint αναφέρονται χωριστά ως απλά "Blueprints". Το σύστημα αυτό είναι εξαιρετικά ευέλικτο και ισχυρό, καθώς παρέχει τη δυνατότητα στους σχεδιαστές να χρησιμοποιούν ουσιαστικά το πλήρες φάσμα των εννοιών και των εργαλείων που είναι γενικά διαθέσιμα μόνο στους προγραμματιστές. Επιπλέον, η σχεδίαση Blueprint που είναι διαθέσιμη στην εφαρμογή C ++ της Unreal Engine επιτρέπει στους προγραμματιστές να δημιουργούν συστήματα βάσης που μπορούν να επεκταθούν από τους σχεδιαστές.

Ο προγραμματισμός του παιχνιδιού και ό, τι χρησιμοποιήθηκε στο παρελθόν το UnrealScript μπορεί ακόμα να αντιμετωπιστεί μέσω κώδικα με τη χρήση της C ++. Ταυτόχρονα, ενώ τα Blueprints δεν προορίζονται για αντικατάσταση του UnrealScript, εξυπηρετούν πολλούς από τους ίδιους σκοπούς που χειρίστηκαν το

UnrealScript, όπως, η επέκταση τάξεων, αποθήκευση και τροποποίηση των προεπιλεγμένων ιδιοτήτων, διαχείριση απεικόνισης (π.χ., συστατικά στοιχεία) για τις κλάσεις. Η προσδοκία είναι ότι οι προγραμματιστές του παιχνιδιού θα δημιουργήσουν κατηγορίες βάσεων που θα εκθέτουν ένα χρήσιμο σύνολο λειτουργιών και ιδιοτήτων που μπορούν να χρησιμοποιήσουν και να επεκτείνουν τα Blueprints από αυτές τις βασικές κατηγορίες.

Τα Blueprints μπορούν να είναι ένας από τους διάφορους τύπους που έχουν τη δική τους συγκεκριμένη χρήση από τη δημιουργία νέων τύπων σε συμβάντα σε επίπεδο δέσμης ενεργειών, έως τον ορισμό διεπαφών ή μακροεντολών που θα χρησιμοποιηθούν από άλλα Blueprints. Μια κλάση Blueprint, συχνά συντομευμένη ως Blueprint, είναι ένα asset που επιτρέπει στους δημιουργούς περιεχομένου να προσθέτουν εύκολα λειτουργικότητα πέρα από τις υπάρχουσες κατηγορίες παιχνιδιού. Τα Blueprints δημιουργούνται μέσα από τον επεξεργαστή Unreal οπτικά, αντί με την πληκτρολόγηση κώδικα και αποθηκεύονται ως στοιχεία ενεργητικού σε ένα πακέτο περιεχομένου. Αυτά ουσιαστικά ορίζουν μια νέα κλάση ή τύπο Actor που μπορεί στη συνέχεια να τοποθετηθεί σε χάρτες ως περιπτώσεις που συμπεριφέρονται όπως οποιοσδήποτε άλλος τύπος Actor.

Ένα Blueprint μόνο για δεδομένα είναι μια κατηγορία Blueprint που περιέχει μόνο τον κώδικα (με τη μορφή γραφικών κόμβων), τις μεταβλητές και τα στοιχεία που κληρονόμησε από τον γονέα του. Αυτά επιτρέπουν να τροποποιηθούν και να τροποποιηθούν αυτές οι κληρονομούμενες ιδιότητες, αλλά δεν μπορούν να προστεθούν νέα στοιχεία. Αυτά είναι ουσιαστικά αντικατάσταση των αρχέτυπων και μπορούν να χρησιμοποιηθούν για να επιτρέψουν στους σχεδιαστές να τροποποιήσουν τις ιδιότητες ή να ορίσουν αντικείμενα με παραλλαγές.

Το Blueprint μόνο για δεδομένα επεξεργάζεται σε ένα σύνθετο πρόγραμμα επεξεργασίας ιδιοτήτων, αλλά μπορεί επίσης να μετατραπεί σε πλήρη Blueprints με την απλή προσθήκη κώδικα, μεταβλητών ή στοιχείων χρησιμοποιώντας τον πλήρη επεξεργαστή Blueprint. Το Level Blueprint είναι ένας εξειδικευμένος τύπος Blueprint που λειτουργεί ως γενικό επίπεδο πλατφόρμα γεγονότων.

Κάθε επίπεδο του έργου μας έχει το δικό του Level Blueprint που δημιουργήθηκε από προεπιλογή και μπορεί να επεξεργαστεί μέσα στον επεξεργαστή Unreal, ωστόσο δεν είναι δυνατή η δημιουργία νέων επιπέδων Blueprints μέσω της διεπαφής editor. Τα γεγονότα που αφορούν το επίπεδο ως σύνολο ή συγκεκριμένες περιπτώσεις των Ηθοποιών μέσα στο επίπεδο, χρησιμοποιούνται για να πυροδοτήσουν ακολουθίες ενεργειών με τη μορφή λειτουργιών κλήσεων ή λειτουργιών ελέγχου ροής. Επίπεδο Blueprints παρέχουν επίσης ένα μηχανισμό ελέγχου για το επίπεδο ροής και Matinee, καθώς και για την δέσμευση των γεγονότων σε Actors που τοποθετούνται στο επίπεδο.

3. Δομή της εφαρμογής Point Blanc - Εγκατάσταση Unity

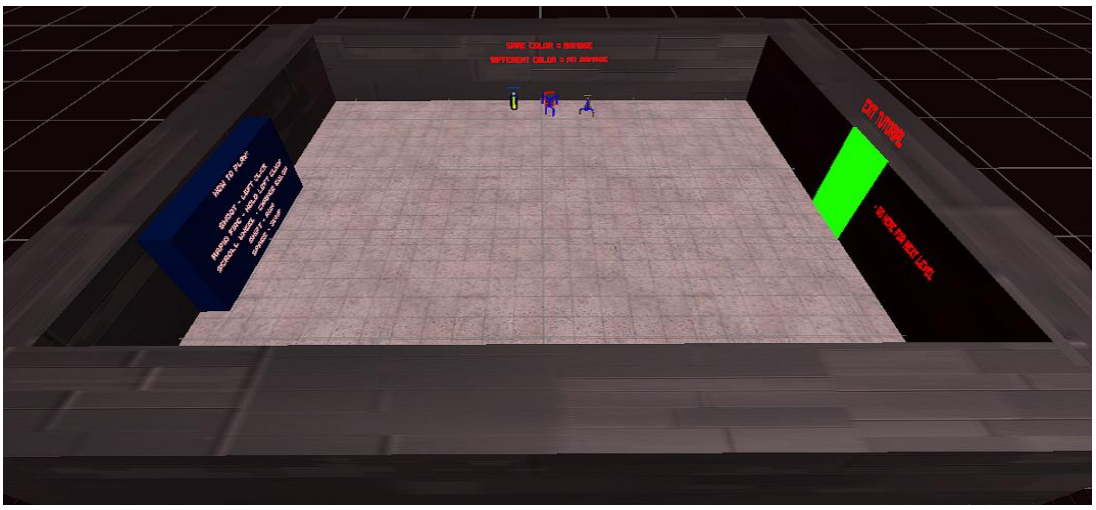
3.1. Δομή εφαρμογής

Η βασική δομή της εφαρμογής αποτελείται από 4 σκηνές :

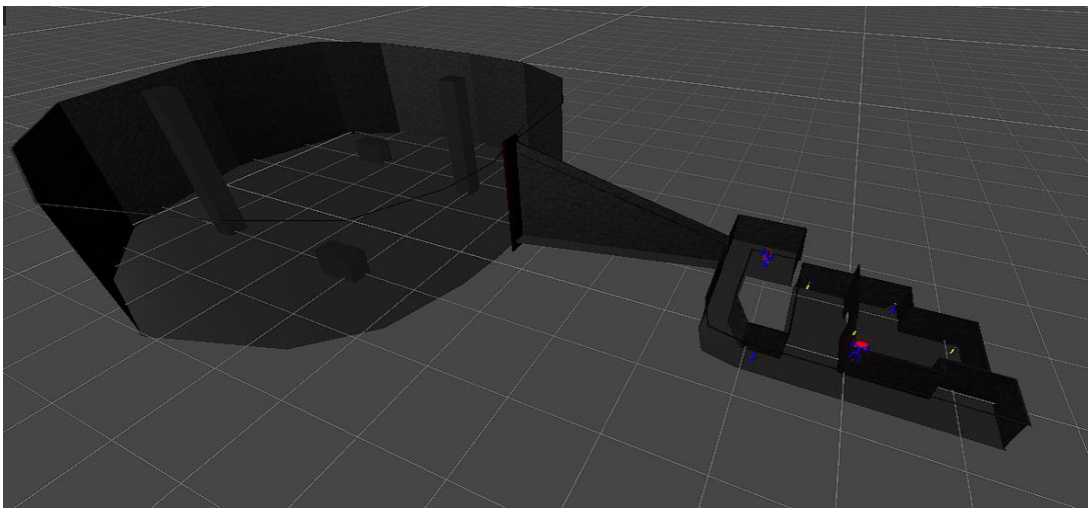
3.1.1. Σκηνή αρχικού μενού

Εικόνα **3.1** Μενού Pointy Blanc

3.1.2. Σκηνές επιπέδου



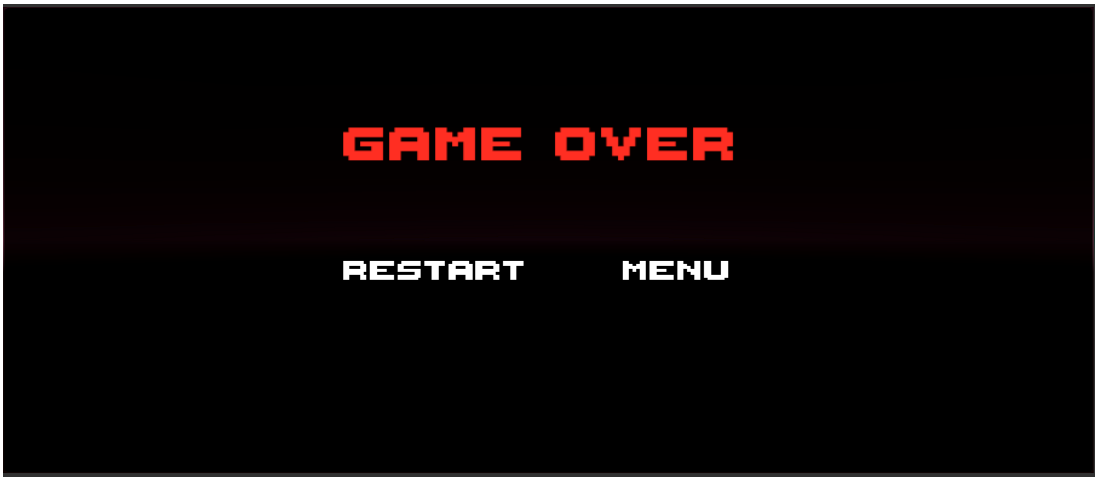
Εικόνα **3.2** Επίπεδο εκπαίδευσης



Εικόνα **3.3** Πρώτο Επίπεδο

Το πρώτο επίπεδο στο οποίο θα βρισκόμαστε πάντα, θα είναι το επίπεδο TUTORIAL, για να μας εξηγηθούν οι τρόποι με τους οποίους παιζεται το παιχνίδι. Έπειτα, οδηγούμαστε σε ένα επόμενο επίπεδο. Η σκηνή επιπέδου είναι επεκτάσιμη και έτσι μπορεί να περιέχει οποιονδήποτε αριθμό επιπέδων.

3.1.3. Σκηνή ήττας



Εικόνα **3.4** Μενού Ήττας

Η σκηνή ήττας μας παρουσιάζεται όταν πληρεί ο παίκτης τις προϋποθέσεις για να χάσει. Μας επιτρέπει να ξαναπροσπαθήσουμε το επίπεδο στο οποίο χάσαμε, με το button RESTART, είτε να οδηγηθούμε στο αρχικό μενού αν δεν θέλουμε να παίξουμε άλλο, με το button MENU.

3.1.4. Σκηνή νίκης

Εικόνα **3.5** Μενού Νίκης

Παρομοίως, η σκηνή νίκης θα μας παρουσιαστεί όταν νικήσουμε το παιχνίδι. Και εδώ, το button MENU, μας οδηγεί στο αρχικό μενού.

3.1.5. Λειτουργικά συστήματα και απαιτήσεις συστήματος Unity

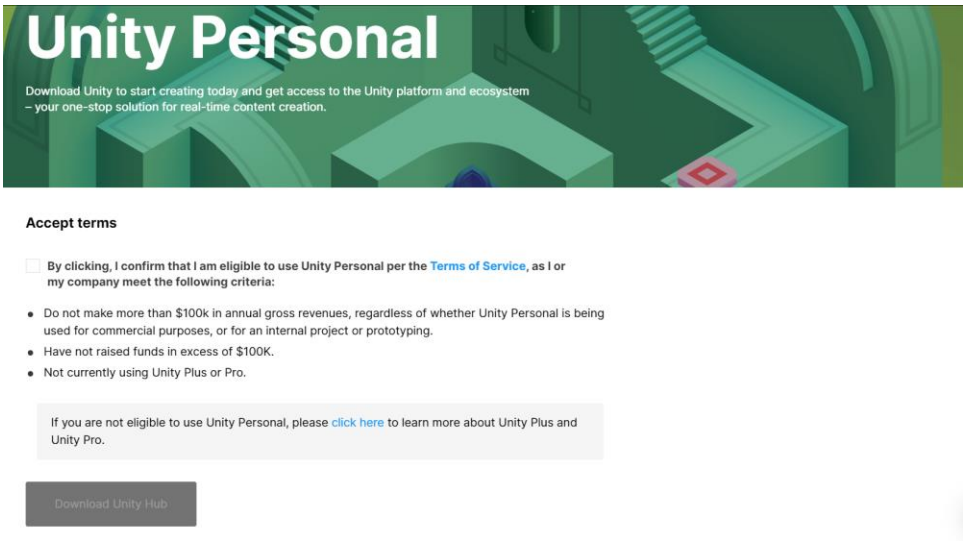
Windows - Windows 7 (SP1+) και Windows 10, 64-bit versions μόνο.

macOS - High Sierra 10.13+.

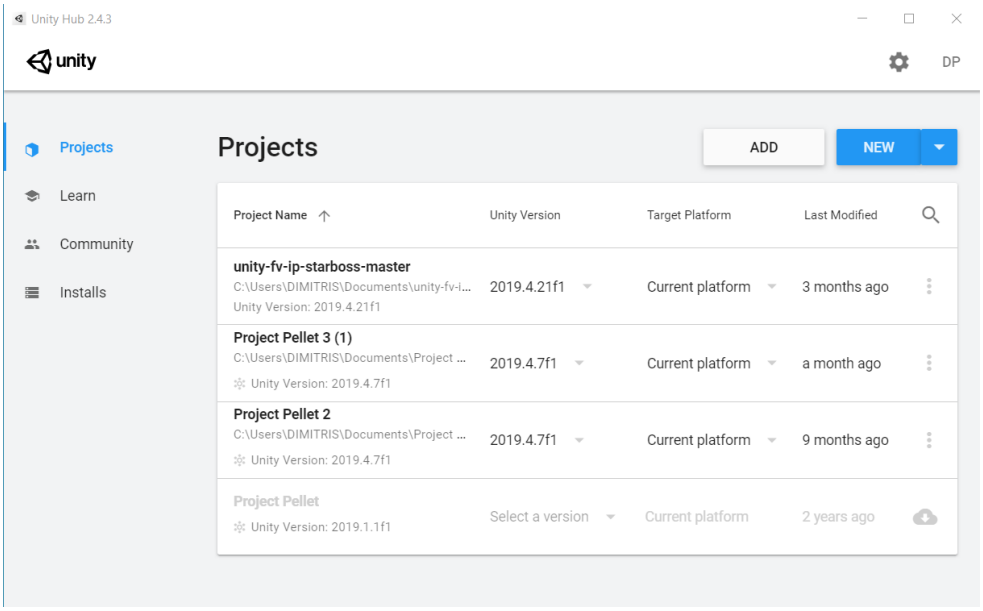
Linux - Ubuntu 20.4, Ubuntu 18.04, και CentOS 7.

3.1.6. Εγκατάσταση Unity Hub

Παρακάτω επεξηγούμε πως να εγκαταστήσουμε το περιβάλλον Unity Hub, το οποίο μας επιτρέπει να δημιουργούμε και να διαχειριζόμαστε τα Unity projects μας, να εγκαθιστούμε τις τελευταίες εκδόσεις του Editor καθώς και επιπλέον components καθώς και να αντλούμε έτοιμα tutorial projects, δημιουργημένα από το community του Unity, για εκμάθηση και εξάσκηση. Για να κατεβάσουμε το Unity Hub, πηγαίνουμε στην ιστοσελίδα του Unity (https://store.unity.com/download?ref=personal), αποδεχόμαστε τους όρους και πατάμε στο κουμπί κατέβασμα Unity Hub.



Εικόνα **3.6** Ιστοσελίδα εγκατάστασης Unity

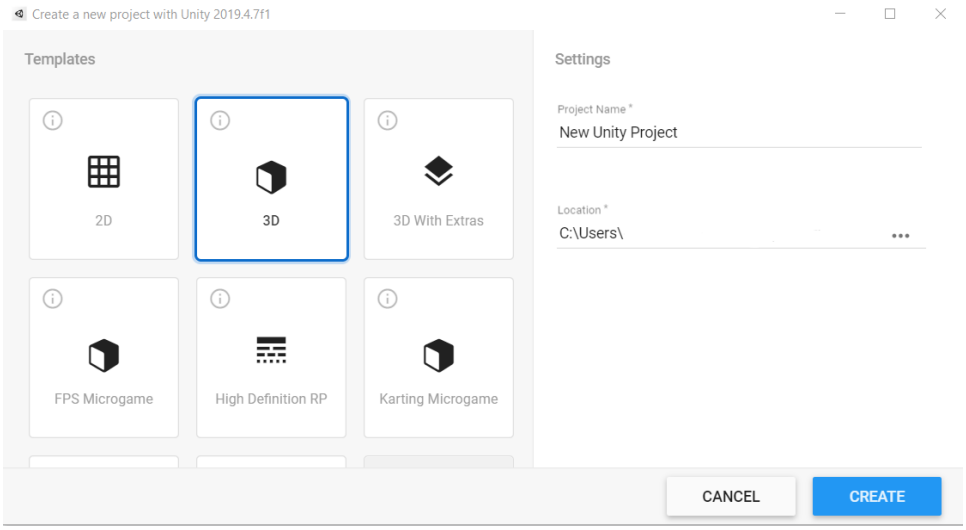


Εικόνα **3.7** Αρχική Οθόνη Unity Hub

4. Ανάλυση και υλοποίηση εφαρμογής

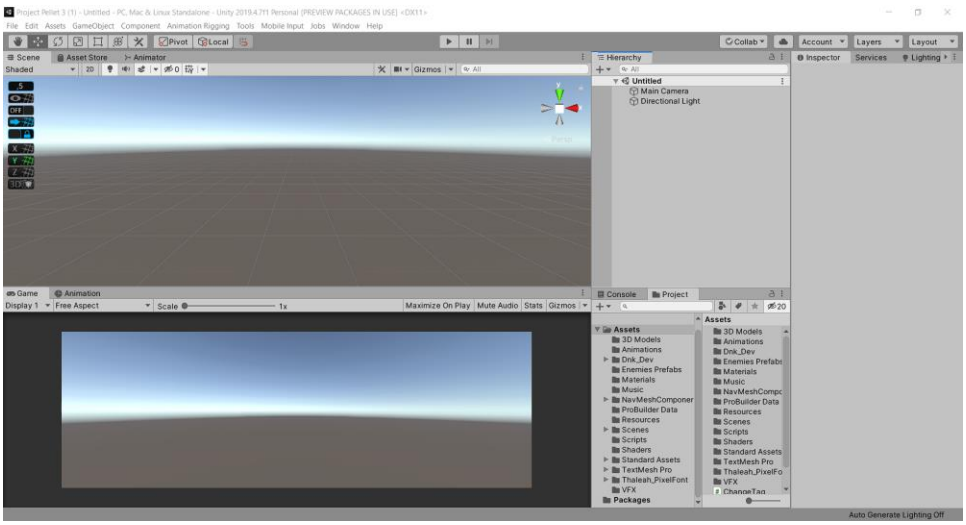
4.1. Δημιουργία Unity project

Για να ξεκινήσουμε ένα νέο Unity project πατάμε αρχικά στο NEW και μας παρουσιάζεται η οθόνη επιλογής προκαθορισμένων template. Επιλέγουμε το project μας να είναι 3D και επιλέγουμε το όνομα που θέλουμε και τον χώρο αποθήκευσης που θα βρίσκεται.



Εικόνα 4.1 Οθόνη δημιουργίας νέου project

Έπειτα, μας εμφανίζεται ο χώρος εργασίας του Unity. (Η διάταξη των διαφόρων παραθύρων είναι ενδεικτική.)



Εικόνα **4.2** Επιφάνεια εργασίας Unity

4.2. Στόχος του Pointy Blanc

Η ιδέα πίσω από το Pointy Blanc είναι ένα διαδραστικό τρισδιάστατο παιχνίδι πρώτου προσώπου όπου ο χαρακτήρας που χειριζόμαστε θα πρέπει να περιηγηθεί σε πολλαπλά επίπεδα και να νικήσει εχθρούς πυροβολώντας τους, με σκοπό να φτάσει στο τέλος.

Η εφαρμογή αυτή είναι φτιαγμένη έτσι ώστε ο χειριστής να κινεί το κεφάλι του χαρακτήρα με το κράνος εικονικής πραγματικότητας και να τον μετακινεί στο χώρο με ένα χειριστήριο. Το γεγονός ότι δημιουργήθηκε με σκοπό να μπορεί να παιχθεί σε ηλεκτρονικό υπολογιστή, δεν μας περιορίζει σε έναν καθώς το Unity παρέχει στην τελική κατασκευή του παιχνιδιού, την δυνατότητα να μπορεί να παιχθεί και σε κινητές συσκευές μέσω εσωτερικών αλλαγών στον Editor.

Αυτό μας διευκολύνει στην συγκεκριμένη περίπτωση, αφού μπορεί να αποφευχθεί η αγορά ακριβού εξοπλισμού, να ελέγχουμε στην περίοδο κατασκευής του, να δοκιμάσουμε και να παίξουμε το παιχνίδι σε μια κινητή συσκευή, που θα πάρει τον ρόλο του κράνους.

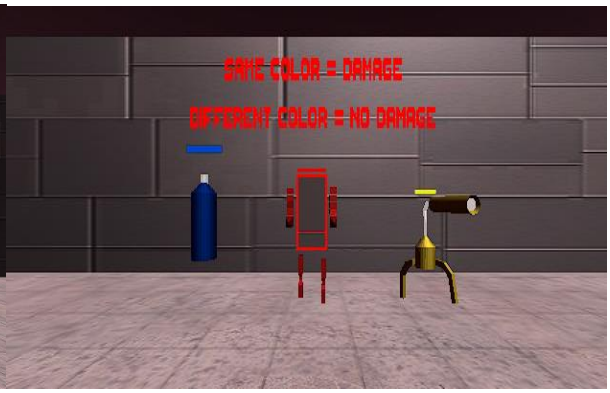
4.3. Ανάλυση της εφαρμογής

Το παιχνίδι ξεκινά με το αρχικό μενού. Σε αυτό περιέχεται ένα άδειο περιβάλλον με ένα τυπικό animation και soundtrack και στοιχεία UI που παρέχουν τις επιλογές:

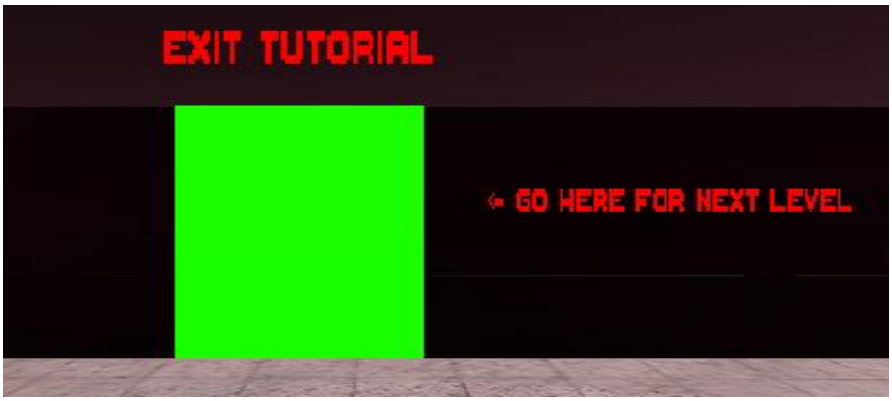
PLAY για να ξεκινήσουμε το παχνίδι, **OPTIONS** για να ρυθμίσουμε την ένταση του ήχου, **QUIT** για να κλείσει η εφαρμογή και **CREDITS** για να δούμε τις ευχαριστίες. Αφού πατήσουμε το **PLAY**, οδηγούμαστε στο επίπεδο εκμάθησης, όπου είναι ένα επίπεδο πέρα από τα βασικά, ώστε ο

χειριστής να μάθει ποιά κουμπιά χρησιμοποιούνται και για ποιό σκοπό, ποιοί είναι οι εχθροί και πώς οδηγούμαστε στο επόμενο επίπεδο.

Εικόνα **4.3** Κουμπιά χειρισμού χαρακτήρα

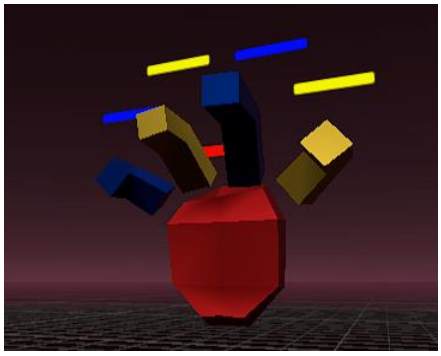


Εικόνα **4.4** Εχθροί



Εικόνα **4.5** Έξοδος

Αφού περάσουμε την έξοδο, βρισκόμαστε στο πρώτο κανονικό επίπεδο. Κάθε επίπεδο έχει έναν αριθμό των διάφορων εχθρών και πρέπει να εξοντωθούν όλα, καθώς είναι προαπαιτούμενα για να ανοίξει η πόρτα στο τέλος του επιπέδου που οδηγεί στον τελικό εχθρό. Ο τελικός εχθρός διαφέρει σε συμπεριφορά και δυσκολία από τους απλούς εχθρούς και μόνο όταν νικηθεί και αυτός θα προχωρήσει ο παίχτης στο επόμενο επίπεδο ή στο τέλος.



Εικόνα **4.6** Τελικός εχθρός

4.4. Κώδικες εφαρμογής

4.4.1. AudioManager.cs

```
using System.Collections;

using System.Collections.Generic;

using UnityEngine;
```

```
public class AudioManager : MonoBehaviour
{
    public AudioSource BGM;
    public void ChangeBGM(AudioClip music)
    {
        if (BGM.clip.name == music.name)
        {
            return;
        }
        BGM.Stop();
        BGM.clip = music;
        BGM.Play();
    }
}
```

4.4.2. RedDamage.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class RedDamage : MonoBehaviour
{   public int damage;
    void Start()
   { Destroy(gameObject, 2.5f); }
  void OnCollisionEnter(Collision collision)  {
 if (collision.collider.tag == "RedEnemy")
      {
         collision.gameObject.GetComponent<Enemy>().TakeDamage(damage);
         Destroy(gameObject);
         PopUp.Create(transform.position, damage);
      }
      else if (collision.collider.tag == "YellowEnemy")
      {
         collision.gameObject.GetComponent<Enemy>().TakeDamage(0);
         Destroy(gameObject);
         PopUp.Create(transform.position, 0);
      }
      else if (collision.collider.tag == "BlueEnemy")
      {  collision.gameObject.GetComponent<Enemy>().TakeDamage(0);
         Destroy(gameObject);
         PopUp.Create(transform.position, 0);
      }
      else if (collision.collider.tag == "RedBoss")
      {
         collision.gameObject.GetComponent<Boss>().TakeDamage(damage);
         Destroy(gameObject);
         PopUp.Create(transform.position, damage);
      }
      else if (collision.collider.tag == "YellowBoss")
      {
         collision.gameObject.GetComponent<Boss>().TakeDamage(0);
         Destroy(gameObject);
         PopUp.Create(transform.position, 0);
```

```
    }
    else if (collision.collider.tag == "BlueBoss")
    {
       collision.gameObject.GetComponent<Boss>().TakeDamage(0);
       Destroy(gameObject);
       PopUp.Create(transform.position, 0);
    }else {  Destroy(gameObject); }}}
```

4.4.3. Boss.cs

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

public class Boss : MonoBehaviour
{
   public int maxHealth = 50;
   public int currentHealth;

   public HealthBar healthbar;

   public bool death = false;

   void Start()
   {
      currentHealth = maxHealth;
      healthbar.SetMaxHealth(maxHealth);
   }

  public void TakeDamage(int damage)
   {
      currentHealth -= damage;

      healthbar.SetHealth(currentHealth);
```

```
        if (currentHealth <= 0)
        {
            Die();
            death = true;
        }
    }

    void Die()
    {
        gameObject.SetActive(false);
        foreach (Transform child in transform)
            child.gameObject.SetActive(false);
    }
}
```

4.4.4. BossMelee.cs

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

public class BossMelee : MonoBehaviour
{
    public GameObject palm;
    public ChangeMaterial change5;

    public void bossActiveAttack()
    {
        palm.GetComponent<Collider>().enabled = true;
    }

    public void bossDeactiveAttack()
    {
        palm.GetComponent<Collider>().enabled = false;
    }
```

```
    public void activeAttack5()
    {
        change5.ChangeMat1();
    }

    public void deactiveAttack5()
    {
        change5.ChangeMat2();
    }
}
```

4.4.5. BossShooting.cs

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

public class BossShooting : MonoBehaviour
{
    public GameObject projectile, projectile2 ;
    public Transform player, firePoint, firePoint2, firePoint3, firePoint4;
    public ChangeMaterial change, change2, change3, change4;

    public void Attack1()
    {
        Rigidbody rb = Instantiate(projectile, firePoint.position, Quaternion.identity).GetComponent<Rigidbody>();
        rb.velocity = (player.position - firePoint.position).normalized * 100;
        Destroy(rb, 2f);
    }

    public void Attack2()
    {
        Rigidbody rb = Instantiate(projectile2, firePoint2.position, Quaternion.identity).GetComponent<Rigidbody>();
        rb.velocity = (player.position - firePoint2.position).normalized * 100;
```

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

public void Attack3()
{
    Rigidbody rb = Instantiate(projectile, firePoint3.position, Quaternion.identity).GetComponent<Rigidbody>();
    rb.velocity = (player.position - firePoint3.position).normalized * 100;
    Destroy(rb, 2f);
 }

public void Attack4()
{
    Rigidbody rb = Instantiate(projectile2, firePoint4.position, Quaternion.identity).GetComponent<Rigidbody>();
    rb.velocity = (player.position - firePoint4.position).normalized * 100;
    Destroy(rb, 2f);
}

public void activeAttack()
{
   change.ChangeMat1();

}

public void deactiveAttack()
{
   change.ChangeMat2();

}

public void activeAttack2()
{
   change2.ChangeMat1();

}
```

```
    public void deactiveAttack2()
    {
        change2.ChangeMat2();

    }
    public void activeAttack3()
    {
        change3.ChangeMat1();

    }

    public void deactiveAttack3()
    {
        change3.ChangeMat2();

    }
    public void activeAttack4()
    {
        change4.ChangeMat1();

    }

    public void deactiveAttack4()
    {
        change4.ChangeMat2();

    }
}
```

4.4.6. BossMovement.cs

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

```
using UnityEngine.AI;

public class BossMovement : MonoBehaviour
{
    public Transform player;
    [SerializeField] private Transform[] destinationPoints = null;
    public BossAiRanged ai, ai2, ai3, ai4;

    private int index = 0;
    private NavMeshAgent agent;
    private bool arrivedAtDestination;

    private Animator animator = null;
    [SerializeField] private GameObject inner = null;

    void Start()
    {
        agent = GetComponent<NavMeshAgent>();
        animator = inner.GetComponent<Animator>();
    }

    void FixedUpdate()
    {
        inner.transform.LookAt(player.position);
        SetNextDestination();
        CheckIfArrivedAtDestination();
    }

    void CheckIfArrivedAtDestination()
    {
        if (Vector3.Distance(agent.destination, transform.position) > (agent.stoppingDistance + 0.1f))
        {
            return;
        }
```

```
        ai.numberOfAttacks = 0;
        ai2.numberOfAttacks = 0;
        ai3.numberOfAttacks = 0;
        ai4.numberOfAttacks = 0;
        arrivedAtDestination = true;
    }

    void SetNextDestination()
    {
        if (!arrivedAtDestination)
        {
            return;
        }

        var destination = destinationPoints[index].position;
        NavMeshHit navHit;
        if (NavMesh.SamplePosition(destination, out navHit, 100, -1))
        {
            destination = navHit.position;
        }

        agent.SetDestination(destination);
        arrivedAtDestination = false;
        index++;
        if (index >= destinationPoints.Length)
            index = 0;
    }
}
```

4.4.7. BossDeath.cs

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

```
public class BossDeath : MonoBehaviour
{
    public Boss palm;

    void Update()
    {
        if (palm.death)
        {
            SceneManager.LoadScene(4);
        }
    }
}
```

4.4.8. BossAiMelee.cs

```
using UnityEngine;
using UnityEngine.AI;

public class BossAiMelee : MonoBehaviour
{
    public NavMeshAgent agent;

    public Transform player;

    public LayerMask Ground, Player;

    private Animator animator = null;

    [SerializeField] private GameObject inner = null;
    [SerializeField] private GameObject outer = null;
    [SerializeField] private GameObject thumb = null;

    public float timeBetweenAttacks;
    bool alreadyAttacked;
```

```
public float sightRange, attackRange;
public bool playerInSightRange, playerInAttackRange;

private void Awake()
{
    player = GameObject.FindWithTag("Player").transform;
    agent = outer.GetComponent<NavMeshAgent>();
    animator = inner.GetComponent<Animator>();
}

private void Start()
{
    agent.stoppingDistance = 9f;
    thumb.GetComponent<BossAiRanged>().enabled = false;
    var pos = inner.transform.position;
    inner.transform.position = new Vector3(pos.x, pos.y - 7, pos.z);
    animator.SetBool("Attack1", false);
    animator.SetBool("Attack2", false);
    animator.SetBool("Attack3", false);
    animator.SetBool("Attack4", false);
}

private void Update()
{
    playerInSightRange = Physics.CheckSphere(transform.position, sightRange, Player);
    playerInAttackRange = Physics.CheckSphere(transform.position, attackRange, Player);

    if (playerInSightRange && !playerInAttackRange) ChasePlayer();
    if (playerInAttackRange && playerInSightRange) AttackPlayer();
}

private void ChasePlayer()
{
```

```
    agent.SetDestination(player.position);
    inner.transform.LookAt(player.position);

}

private void AttackPlayer()
{
    agent.SetDestination(transform.position);

    transform.LookAt(player.position);
    inner.transform.LookAt(player.position);

    animator.SetTrigger("Attack5");
    animator.SetTrigger("ChangeMat5");

    if (!alreadyAttacked)
    {
        alreadyAttacked = true;
        Invoke(nameof(ResetAttack), timeBetweenAttacks);
    }
}

private void ResetAttack()
{
    alreadyAttacked = false;
}


private void OnDrawGizmosSelected()
{
    Gizmos.color = Color.red;
    Gizmos.DrawWireSphere(transform.position, attackRange);
    Gizmos.color = Color.yellow;
    Gizmos.DrawWireSphere(transform.position, sightRange);
```

```
        Gizmos.color = Color.blue;
    }
}
```

4.4.9. BossAiRanged.cs

```
using UnityEngine;
using UnityEngine.AI;

public class BossAiRanged : MonoBehaviour
{
    public Transform player;

    public LayerMask Ground, Player;

    private Animator animator = null;

    [SerializeField] private GameObject inner = null;
    [SerializeField] private GameObject outer = null;

    [SerializeField] private GameObject pinkie = null;
    [SerializeField] private GameObject ring = null;
    [SerializeField] private GameObject pointer = null;
    [SerializeField] private GameObject thumb = null;
    [SerializeField] private GameObject palm = null;

    [SerializeField] private GameObject canvas = null;
    [SerializeField] private GameObject canvas2 = null;
    [SerializeField] private GameObject canvas3 = null;
    [SerializeField] private GameObject canvas4 = null;
    [SerializeField] private GameObject canvas5 = null;

    public int numberOfAttacks;
    public float timeBetweenAttacks;
```

```
bool alreadyAttacked;

public float attackRange;
public bool playerInAttackRange;

private void Awake()
{
   player = GameObject.FindWithTag("Player").transform;
   animator = inner.GetComponent<Animator>();
   animator.SetBool("Attack1", false);

   pinkie.GetComponent<Collider>().enabled = true;
   ring.GetComponent<Collider>().enabled = false;
   pointer.GetComponent<Collider>().enabled = false;
   thumb.GetComponent<Collider>().enabled = false;
   palm.GetComponent<Collider>().enabled = false;

   canvas.SetActive(true);
   canvas2.SetActive(false);
   canvas3.SetActive(false);
   canvas4.SetActive(false);
   canvas5.SetActive(false);

}

private void Update()
{
   playerInAttackRange = Physics.CheckSphere(transform.position, attackRange, Player);

      if (numberOfAttacks < 3)
      {
         AttackPlayer();
         outer.GetComponent<BossMovement>().enabled = false;
```

```
        if (numberOfAttacks == 3)
        {
            animator.SetBool("Attack1", false);
            animator.SetBool("Attack2", false);
            animator.SetBool("Attack3", false);
            animator.SetBool("Attack4", false);
            outer.GetComponent<BossMovement>().enabled = true;
        }
    }

    if (pinkie.activeInHierarchy == false && ring.activeInHierarchy == true && pointer.activeInHierarchy == true
&& thumb.activeInHierarchy == true && palm.activeInHierarchy == true)
    {
        ring.GetComponent<Collider>().enabled = true;
        canvas2.SetActive(true);
    }

    if (pinkie.activeInHierarchy == false && ring.activeInHierarchy == false && pointer.activeInHierarchy == true
&& thumb.activeInHierarchy == true && palm.activeInHierarchy == true)
    {
        pointer.GetComponent<Collider>().enabled = true;
        canvas3.SetActive(true);
    }

    if (pinkie.activeInHierarchy == false && ring.activeInHierarchy == false && pointer.activeInHierarchy == false
&& thumb.activeInHierarchy == true && palm.activeInHierarchy == true)
    {
        thumb.GetComponent<Collider>().enabled = true;
        canvas4.SetActive(true);
    }
    if (pinkie.activeInHierarchy == false && ring.activeInHierarchy == false && pointer.activeInHierarchy == false
&& thumb.activeInHierarchy == false && palm.activeInHierarchy == true)
    {
        palm.GetComponent<Collider>().enabled = true;
        canvas5.SetActive(true);
        outer.GetComponent<BossMovement>().enabled = false;
```

```
        palm.GetComponent<BossAiMelee>().enabled = true;
        palm.GetComponent<BossAiRanged>().enabled = false;
    }
}

private void AttackPlayer()
{
    transform.LookAt(player.position);
    inner.transform.LookAt(player.position);

    if (!alreadyAttacked)
    {
        if (pinkie.activeInHierarchy == true && ring.activeInHierarchy == true && pointer.activeInHierarchy == true
&& thumb.activeInHierarchy == true && palm.activeInHierarchy == true)
        {
            animator.SetBool("Attack1", true);
            animator.SetTrigger("ChangeMat");

        }

        if (pinkie.activeInHierarchy == false && ring.activeInHierarchy == true && pointer.activeInHierarchy == true
&& thumb.activeInHierarchy == true && palm.activeInHierarchy == true)
        {
            animator.SetBool("Attack1", false);
            animator.SetBool("Attack2", true);
            animator.SetTrigger("ChangeMat2");

        }

        if (pinkie.activeInHierarchy == false && ring.activeInHierarchy == false && pointer.activeInHierarchy == true
&& thumb.activeInHierarchy == true && palm.activeInHierarchy == true)
        {
            animator.SetBool("Attack1", false);
            animator.SetBool("Attack2", false);
            animator.SetBool("Attack3", true);
```

```
            animator.SetTrigger("ChangeMat3");


        }


        if (pinkie.activeInHierarchy == false && ring.activeInHierarchy == false && pointer.activeInHierarchy == false && thumb.activeInHierarchy == true && palm.activeInHierarchy == true)
        {
            animator.SetBool("Attack1", false);
            animator.SetBool("Attack2", false);
            animator.SetBool("Attack3", false);
            animator.SetBool("Attack4", true);
            animator.SetTrigger("ChangeMat4");


        }
        if (pinkie.activeInHierarchy == false && ring.activeInHierarchy == false && pointer.activeInHierarchy == false && thumb.activeInHierarchy == false && palm.activeInHierarchy == true)
        {
            animator.SetBool("Attack1", false);
            animator.SetBool("Attack2", false);
            animator.SetBool("Attack3", false);
            animator.SetBool("Attack4", false);
        }


        alreadyAttacked = true;
        numberOfAttacks++;
        Invoke(nameof(ResetAttack), timeBetweenAttacks);
    }
  }


  private void ResetAttack()
  {
    alreadyAttacked = false;
  }
```

```
    private void OnDrawGizmosSelected()
    {
        Gizmos.color = Color.red;
        Gizmos.DrawWireSphere(transform.position, attackRange);
        Gizmos.color = Color.blue;
    }
}
```

4.4.10. BossMeleeDamage.cs

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

public class BossMeleeDamage : MonoBehaviour
{
    public int damage;

    public CharacterController controller;

    void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.tag == "Player")
        {
            if (other == controller)
                other.gameObject.GetComponent<Player>().TakeDamage(damage);
        }
    }
}
```

4.4.11. ChangeMaterial.cs

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

```
public class ChangeMaterial : MonoBehaviour
{
   public Material[] materials;
   public Renderer rend;


  public void ChangeMat1()
   {
      rend = GetComponent<Renderer>();
      rend.enabled = true;
      rend.sharedMaterial = materials[0];
   }

   public void ChangeMat2()
   {
      rend = GetComponent<Renderer>();
      rend.enabled = true;
      rend.sharedMaterial = materials[1];
   }
}
```

4.4.12. GameAssets.cs

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

public class GameAssets : MonoBehaviour
{
   private static GameAssets _i;

   public static GameAssets i
   {
```

```
        get
        {
            if (_i == null) _i = Instantiate(Resources.Load<GameAssets>("GameAssets"));
            return _i;
        }
    }

    public Transform DamagePopup;
}
```

4.4.13. HealthBar.cs

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

public class HealthBar : MonoBehaviour
{
    public Slider slider;

    public void SetMaxHealth(int health)
    {
        slider.maxValue = health;
        slider.value = health;
    }

    public void SetHealth(int health)
    {
        slider.value = health;
    }
}
```

4.4.14. LevelLoader.cs

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

public class LevelLoader : MonoBehaviour
{
    public Animator transition;
    public float transitionTime = 3.5f;

    void Update()
    {
        if (Input.GetMouseButtonDown(1))
        {
            LoadNextLevel();
        }
    }

    public void LoadNextLevel()
    {
        StartCoroutine(LoadLevel(SceneManager.GetActiveScene().buildIndex + 1));
    }

    IEnumerator LoadLevel(int levelIndex)
    {
        transition.SetTrigger("Start");

        yield return new WaitForSeconds(transitionTime);

        SceneManager.LoadScene(levelIndex);
    }
}
```

4.4.15. MainMenu.cs

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

public class MainMenu : MonoBehaviour
{
    public Animator transition, fade;
    public float transitionTime = 3f;

    public void PlayGame ()
    {
        StartCoroutine(LoadLevel(SceneManager.GetActiveScene().buildIndex + 1));
    }

    public void QuitGame()
    {
        Debug.Log("QUIT");
        Application.Quit();
    }

    IEnumerator LoadLevel(int levelIndex)
    {
        transition.SetTrigger("Start");
        fade.SetTrigger("Start");

        yield return new WaitForSeconds(transitionTime);

        SceneManager.LoadScene(levelIndex);
    }
}
```

4.4.16. MusicSwap.cs

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

public class MusicSwap : MonoBehaviour
{
    public AudioClip newTrack;
    private AudioManager audioManager;
    [SerializeField] private GameObject boss = null;
    [SerializeField] private GameObject door = null;

    void Start()
    {
        audioManager = FindObjectOfType<AudioManager>();
    }

    void OnTriggerEnter(Collider other)
    {
        if(other.tag == "Player")
        {
            if(newTrack != null)
            {
                audioManager.ChangeBGM(newTrack);
            }
            door.SetActive(true);
            boss.SetActive(true);
        }
    }
}
```

4.4.17. NextStage.cs

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

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class NextStage : MonoBehaviour
{
    public Animator fade;
    public float transitionTime = 2f;

    public GameObject Fps;

    public void Start()
    {
        Fps = GameObject.FindWithTag("Player");
    }

    public void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.tag == "Player")
        {
            StartCoroutine(LoadLevel(SceneManager.GetActiveScene().buildIndex + 1));
            Fps.GetComponent<UnityStandardAssets.Characters.FirstPerson.FirstPersonController>().enabled = false;
        }
    }

    IEnumerator LoadLevel(int levelIndex)
    {
        fade.SetTrigger("Start");

        yield return new WaitForSeconds(transitionTime);

        SceneManager.LoadScene(levelIndex);
    }
}
```

4.4.18. Player.cs

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

public class Player : MonoBehaviour
{
    public int maxHealth = 100;
    public int currentHealth;
    public int index;

    public HealthBar healthbar;

    void Start()
    {
        currentHealth = maxHealth;
        healthbar.SetMaxHealth(maxHealth);

        index = SceneManager.GetActiveScene().buildIndex;
        PlayerPrefs.SetInt("SavedScene", index);
        Debug.Log(index);
    }

    public void TakeDamage(int damage)
    {
        currentHealth -= damage;

        healthbar.SetHealth(currentHealth);

        if (currentHealth <= 0)
        {
            SceneManager.LoadScene(3);
            Die();
```

```
    }
  }

  void Die()
  {
    gameObject.SetActive(false);
    foreach (Transform child in transform)
      child.gameObject.SetActive(false);
  }
}
```

4.4.19. PlayerShooting.cs

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

public class PlayerShooting : MonoBehaviour
{
  public Camera cam;
  public GameObject projectile;
  public Transform firePoint;

  [SerializeField] private float projectileSpeed = 30f;
  [SerializeField] private float fireRate = 0.03f;
  [SerializeField] public float minimumHeldDuration = 1f;
  [SerializeField] public float maximumFireTime = 2f;
  [SerializeField] public float abilityCooldownTime = 6f;

  private float timeSinceLastAbility = 0;
  private float timeSinceLastFire = 0;
  public float timeHeldDuration = 0;
  public float timeFiring = 0;
  public bool isFiring = false;
```

```
private void Update()
{
  if (Input.GetButtonDown("Fire1"))
  {
                timeSinceLastFire = 0;
                timeHeldDuration = 0;
                isFiring = true;
  }
  else if (Input.GetButtonUp("Fire1"))
  {
                if (timeHeldDuration < minimumHeldDuration) {
                        ShootProjectile();
                } else {
                        timeFiring = 0;
                        timeSinceLastAbility = abilityCooldownTime;
                }

                timeHeldDuration = 0;
                isFiring = false;
  }

        if (timeHeldDuration < minimumHeldDuration && timeSinceLastAbility > 0) {
                timeSinceLastAbility = Mathf.Max(0, timeSinceLastAbility - Time.deltaTime);
        }

  Ability();
}

public void Ability()
{
        if (!isFiring) {
                return;
        }
```

```
        if (timeSinceLastAbility > 0) {
                return;
        }

        if (timeHeldDuration < minimumHeldDuration) {
                timeHeldDuration += Time.deltaTime;
                return;
        }

        if (timeFiring > maximumFireTime) {
                return;
        }
        timeFiring += Time.deltaTime;

        if (timeSinceLastFire < fireRate) {
                timeSinceLastFire += Time.deltaTime;
                return;
        }
        timeSinceLastFire = 0;

        ShootProjectile();
}


private void ShootProjectile()
{
   var ray = cam.ViewportPointToRay(new Vector3(0.5f, 0.5f, 0.5f));
   RaycastHit hit;
   var destination = ray.GetPoint(1000);
                if (Physics.Raycast(ray, out hit, Mathf.Infinity, ~(1 << LayerMask.NameToLayer("Bullet")))) {
      destination = hit.point;
   }

   InstantiateProjectile(firePoint, destination);
```

```
    }

    private void InstantiateProjectile(Transform firePoint, Vector3 destination)
    {
                    var direction = (destination - firePoint.position).normalized;
                    var directionRotation = Quaternion.LookRotation(direction);
                    var projectileObj = Instantiate(projectile, firePoint.position, directionRotation) as GameObject;
                    projectileObj.GetComponent<Rigidbody>().AddForce(projectileObj.transform.forward *
projectileSpeed);
            }
}
```

4.4.20. PopUp.cs

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

public class PopUp : MonoBehaviour
{

    public static PopUp Create(Vector3 position, int damageAmount)
    {
        Transform damagePopupTransform = Instantiate(GameAssets.i.DamagePopup, position, Quaternion.identity);

        PopUp damagePopup = damagePopupTransform.GetComponent<PopUp>();
        damagePopup.Setup(damageAmount);

        return damagePopup;
    }

    private TextMeshPro textMesh;
    private float disappearTimer;
    private Color textColor;
```

```
  private void Awake()
  {
    textMesh = transform.GetComponent<TextMeshPro>();
  }

  public void Setup(int damageAmount)
  {
    textMesh.SetText(damageAmount.ToString());
    textColor = Color.blue;
    disappearTimer = 1f;
  }

  private void Update()
  {
    float moveYSpeed = 2f;
    transform.position += new Vector3(0, moveYSpeed) * Time.deltaTime;

    disappearTimer -= Time.deltaTime;
    if (disappearTimer < 0)
    {
      float disappearSpeed = 3f;
      textColor.a -= disappearSpeed * Time.deltaTime;
      textMesh.color = textColor;
      if(textColor.a < 0)
      {Destroy(gameObject); }}}
```

4.4.21. WeaponSwap.cs

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

public class WeaponSwap : MonoBehaviour
```

```
{
    public int selected = 0;

    [SerializeField] private Material mat = null;

    void Start()
    {
        SelectColor();
    }

    void Update()
    {
        int previousSelected = selected;

        if (Input.GetAxis("Mouse ScrollWheel") > 0f)
        {
            if (selected >= transform.childCount - 1)
                selected = 0;
            else
                selected++;
        }

        if (Input.GetAxis("Mouse ScrollWheel") < 0f)
        {
            if (selected <= 0)
                selected = transform.childCount - 1;
            else
                selected--;
        }

        if (previousSelected != selected)
        {
            SelectColor();
        }
```

```
    switch (selected)
    {
        case 0:
            mat.color = Color.red;
            break;
        case 1:
            mat.color = Color.yellow;
            break;
        case 2:
            mat.color = Color.blue;
            break;
        default:
            mat.color = Color.gray;
            break; }}

void SelectColor ()
{
    int i = 0;
    foreach (Transform color in transform)
    {
        if (i == selected)
            color.gameObject.SetActive(true);
        else
            color.gameObject.SetActive(false);
        i++;  }}}
```

Βιβλιογραφία

https://www.vrs.org.uk/virtual-reality/history.html

https://www.digitaltrends.com/cool-tech/history-of-virtual-reality/

https://www.fi.edu/virtual-reality/history-of-virtual-reality

https://www.colocationamerica.com/blog/history-of-virtual-reality

https://en.wikipedia.org/wiki/Virtual_reality

https://en.wikipedia.org/wiki/HTC_Vive

https://en.wikipedia.org/wiki/Google_Cardboard

https://www.microsoft.com/en-us/hololens?SilentAuth=1&wa=wsignin1.0

https://teslasuit.io/blog/history-virtual-reality-ultimate-guide-part-2/

https://thinkmobiles.com/blog/best-vr-sdk/

https://thinkmobiles.com/services/virtual-reality/

https://arvrjourney.com/what-vr-platforms-are-best-for-game-development-b5b65084a2c2?gi=2bfbb6a45873

https://skarredghost.com/2018/03/15/introduction-to-openvr-101-series-what-is-openvr-and-how-to-get-started-with-its-apis/

https://www.unrealengine.com/en-US/features

https://en.wikipedia.org/wiki/Unity_(game_engine)

https://docs.unrealengine.com/4.26/en-US/

https://unity.com/