ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΟΙΚΟΝΟΜΙΚΩΝ ΕΠΙΣΤΗΜΩΝ ΚΑΙ ΔΙΟΙΚΗΣΗΣ ΕΠΙΧΕΙΡΗΣΕΩΝ

ΔΙΟΙΚΗΤΙΚΗΣ ΕΠΙΣΤΗΜΗΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΤΡΙΣΔΙΑΣΤΑΤΕΣ ΠΛΑΤΦΟΡΜΕΣ ΑΝΑΠΤΥΞΗΣ ΠΑΙΧΝΙΔΙΩΝ

ΜΕΛΕΤΗ ΠΕΡΙΠΤΩΣΗΣ ΔΗΜΙΟΥΡΓΊΑΣ ΕΚΠΑΙΔΕΥΤΙΚΗΣ ΔΙΑΔΡΑΣΤΙΚΗΣ ΕΦΑΡΜΟΓΗΣ ΜΕ ΤΗ ΧΡΗΣΗ ΠΛΑΤΦΟΡΜΑΣ ΠΑΙΧΝΙΔΙΩΝ “UNREAL ENGINE”

ΕΠΙΜΕΛΕΙΑ: ΖΩΓΡΑΦΟΥ ΜΑΡΙΑ ΑΡΕΤΗ

ΚΟΥΤΣΑΚΗ ΙΩΑΝΝΑ ΕΙΡΗΝΗ

ΕΠΙΒΛΕΠΩΝ: ΧΑΛΚΙΟΠΟΥΛΟΣ ΚΩΝΣΤΑΝΤΙΝΟΣ

ΠΑΤΡΑ 2020

ΠΡΟΛΟΓΟΣ

Η πτυχιακή εργασία αυτή ολοκληρώθηκε μετά από αρκετά μεγάλο χρονικό διάστημα και έρευνα σε ένα πεδίο που συνεχώς αναπτύσσεται χάρη στις ραγδαίες εξελίξεις στο υλικό και λογισμικό που χρησιμοποιούν οι ηλεκτρονικής υπολογιστές. Εμείς με τη σειρά μας θα θέλαμε να ευχαριστήσουμε τον εισηγητή του θέματός μας και επιβλέποντα καθηγητή μας, ο οποίος μας στήριξε στην προσπάθειά μας και μας παρείχε απαραίτητο υλικό για να ξεκινήσουμε. Θα θέλαμε επίσης να ευχαριστήσουμε όλους εκείνους τους προγραμματιστές που έχουν δημοσιεύσει υλικό και tutorials πάνω στο θέμα που μας βοήθησαν να εξοικειωθούμε με την πλατφόρμα και να ρυθμίσουμε τη σωστή λειτουργία του παιχνιδιού μας.

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΕΡΙΛΗΨΗ

Η παρούσα πτυχιακή εργασία εξετάζει την ανάπτυξη παιχνιδιού σε πλατφόρμα δημιουργίας Unreal Engine 4, η οποία έχει πλέον εδραιωθεί ως κορυφαία για της δυνατότητές της. Η δημιουργία διαδραστικών παιχνιδιών ως μέσο ψυχαγωγίας ξεκίνησε από επιστήμονες υπολογιστών σαν κομμάτι για τις έρευνές τους ή για πλάκα[1] και πλέον έχει εξελιχθεί σε μία από τις πιο κερδοφόρες βιομηχανίες. Αυτό που κατέστησε τη δημιουργία παιχνιδιών πιο προσιτή είναι οι μηχανές δημιουργίας παιχνιδιών. Στην πτυχιακή αυτή θα αναφερθούμε στις επικρατέστερες τέτοιες μηχανές και κυρίως στην Unreal Engine 4, στην οποία και δημιουργήσαμε το παιχνίδι μας.

[1] https://en.wikipedia.org/wiki/History_of_video_games

ABSTRACT

The present thesis examines the game development based in the Unreal Engine 4 platform, which has now established itself as a leading platform on the industry for its capabilities. The development of interactive games as an entertainment has begun from computer scientists as part of their research or simply for fun[1] and has since evolved in one of the most profitable industries. What made game development more accessible are these game development engines. In this thesis we will take a look in the most mainstream game engines and principally on Unreal Engine 4, in which we developed our game.

ΚΕΦΑΛΑΙΟ 1ο : ΤΙ ΕΙΝΑΙ ΤΟ GAMING ΚΑΙ ΤΑ ΒΑΣΙΚΑ ΕΙΔΗ ΤΟΥ

1.1 Τι είναι το Gaming

Ένα παιχνίδι βίντεο είναι ένα ηλεκτρονικό παιχνίδι που περιλαμβάνει αλληλεπίδραση με ένα περιβάλλον εργασίας χρήστη ή μια συσκευή εισόδου, όπως ένα joystick, ένα χειριστήριο, ένα πληκτρολόγιο ή συσκευές ανίχνευσης κίνησης, για τη δημιουργία οπτικής ανάδρασης σε μια συσκευή προβολής βίντεο δύο ή τριών, όπως μια τηλεόραση, μια οθόνη, μια οθόνη αφής ή ακουστικά εικονικής πραγματικότητας. Τα βιντεοπαιχνίδια αυξάνονται με ηχητικά σχόλια από ηχεία ή ακουστικά και προαιρετικά με άλλους τύπους συστημάτων ανάδρασης, συμπεριλαμβανομένης της απτικής τεχνολογίας.

Τα βιντεοπαιχνίδια ορίζονται με βάση την πλατφόρμα τους, η οποία περιλαμβάνει παιχνίδια arcade, παιχνίδια κονσόλας και παιχνίδια pc. Πιο πρόσφατα, ο κλάδος έχει επεκταθεί σε κινητά παιχνίδια μέσω smartphones και tablet υπολογιστές, εικονικά και επαυξημένης πραγματικότητας συστήματα, και απομακρυσμένα παιχνίδια σύννεφο. Τα βιντεοπαιχνίδια ταξινομούνται σε ένα ευρύ φάσμα ειδών με βάση τον τύπο του παιχνιδιού και του σκοπού τους.

Τα πρώτα βιντεοπαιχνίδια ήταν απλές επεκτάσεις των ηλεκτρονικών παιχνιδιών που χρησιμοποιούν βίντεο-όπως εξόδου από μεγάλους υπολογιστές δωμάτιο-μέγεθος στη δεκαετία του 1950 και του 1960, ενώ τα πρώτα βιντεοπαιχνίδια στη διάθεση των καταναλωτών εμφανίστηκε το 1971 μέσω της απελευθέρωσης του arcade παιχνίδι Computer Space, ακολούθησε το επόμενο έτος από Pong, και με την πρώτη κονσόλα σπίτι το Magnavo Odysseyx το 1972. Σήμερα, η ανάπτυξη βιντεοπαιχνιδιών απαιτεί πολλές δεξιότητες για να φέρει ένα παιχνίδι στην αγορά, συμπεριλαμβανομένων των προγραμματιστών, των εκδοτών, των διανομέων, των εμπόρων λιανικής πώλησης, της κονσόλας και άλλων κατασκευαστών τρίτων κατασκευαστών, καθώς και άλλων ρόλων.

Από τη δεκαετία του 2010, η εμπορική σημασία της βιομηχανίας βιντεοπαιχνιδιών αυξάνεται. Οι αναδυόμενες ασιατικές αγορές και τα κινητά παιχνίδια στα smartphones ειδικότερα οδηγούν την ανάπτυξη του κλάδου. Από το 2018, τα βιντεοπαιχνίδια απέφεραν πωλήσεις 134,9 δισεκατομμυρίων δολαρίων ΗΠΑ ετησίως παγκοσμίως, και ήταν το τρίτο μεγαλύτερο τμήμα στην αγορά ψυχαγωγίας των ΗΠΑ, πίσω από τη μετάδοση και την καλωδιακή τηλεόραση.

1.2 Είδη Game

Ένα είδος βιντεοπαιχνιδιών είναι μια συγκεκριμένη κατηγορία παιχνιδιών που σχετίζονται με παρόμοια χαρακτηριστικά παιχνιδιού. Τα είδη βιντεοπαιχνιδιών δεν καθορίζονται συνήθως από τη ρύθμιση ή την ιστορία του παιχνιδιού ή του μέσου παιχνιδιού , αλλά από τον τρόπο που ο παίκτης αλληλεπιδρά με το παιχνίδι. Για παράδειγμα ένας σκοπευτής πρώτου προσώπου εξακολουθεί να είναι ένας σκοπευτής πρώτου προσώπου, ανεξάρτητα από το αν λαμβάνει χώρα σε μια επιστημονική φαντασία, φαντασία, ή στρατιωτικό περιβάλλον εφ 'όσον διαθέτει μια φωτογραφική μηχανή μιμείται την προοπτική του πρωταγωνιστή (πρώτου προσώπου) και το gameplay επικεντρώνεται γύρω από τη χρήση των όπλων .

Τα είδη μπορεί να περιλαμβάνουν μια μεγάλη ποικιλία παιχνιδιών, οδηγώντας σε ακόμα πιο συγκεκριμένες ταξινομήσεις που ονομάζονται υποείδος. Για παράδειγμα, ένα παιχνίδι δράσης μπορεί να ταξινομηθεί σε πολλά υποείδη, όπως παιχνίδια πλατφόρμας και παιχνίδια μάχης. Ορισμένα παιχνίδια, κυρίως browser και κινητών συνήθως ταξινομούνται σε πολλαπλά είδη.

Ακολουθεί μια λίστα με τα πιο γνωστά είδη βιντεοπαιχνιδιών

Action

MMO

Adventure

Role-playing

Simulation

Strategy

Sports

Multiplayer

Arcade

1.3 Action Games

Τα παιχνίδια δράσης δίνουν έμφαση στις φυσικές προκλήσεις που απαιτούν συντονισμό χεριών-ματιών και κινητικές ικανότητες για να ξεπεραστούν. Επικεντρώνονται γύρω από τον παίκτη, ο οποίος έχει τον έλεγχο του μεγαλύτερου μέρους της δράσης. Τα περισσότερα από τα πρώτα βιντεοπαιχνίδια θεωρήθηκαν παιχνίδια δράσης. σήμερα, εξακολουθεί να είναι ένα τεράστιο είδος που καλύπτει όλα τα παιχνίδια που περιλαμβάνουν φυσικές προκλήσεις.

Τα παιχνίδια δράσης ταξινομούνται από πολλά υποείδη Τα παιχνίδια πλατφόρμας και τα παιχνίδια πάλης είναι από τα πιο γνωστά υποείδη, ενώ τα παιχνίδια shooter έγιναν και συνεχίζουν να είναι ένα από τα κυρίαρχα είδη στα βιντεοπαιχνίδια από τη δεκαετία του 1990.

1.4 MMO

Ένα μαζικά multiplayer online παιχνίδι (που ονομάζεται επίσης MMO και MMOG) είναι ένα multiplayer online video game το οποίο είναι σε θέση να υποστηρίξει μεγάλο αριθμό παικτών ταυτόχρονα. Κατ ' ανάγκη, παίζονται στο Διαδίκτυο. Πολλά παιχνίδια έχουν τουλάχιστον έναν επίμονο κόσμο, ωστόσο άλλοι έχουν μόνο μεγάλο αριθμό παικτών που ανταγωνίζονται ταυτόχρονα με τη μία μορφή ή την άλλη χωρίς καμία διαρκή επίδραση με τον κόσμο . Αυτά τα παιχνίδια μπορούν να βρεθούν για τις περισσότερες πλατφόρμες με δυνατότητα δικτύου, συμπεριλαμβανομένου του προσωπικού υπολογιστή, της κονσόλας βιντεοπαιχνιδιών ή των smartphones και άλλων κινητών συσκευών. Ένα παράδειγμα είναι το παιχνίδι Minecraft που παίζεται ευρέως και το οποίο μπορεί να παιχτεί τόσο ως MMO όσο και ως παιχνίδι για έναν παίκτη.

MMO παιχνίδια μπορούν να επιτρέψουν στους παίκτες να συνεργαστούν και να ανταγωνίζονται μεταξύ τους σε μεγάλη κλίμακα, και μερικές φορές να αλληλεπιδρούν με νόημα με τους ανθρώπους σε όλο τον κόσμο. Περιλαμβάνουν μια ποικιλία τύπων παιχνιδιού, που αντιπροσωπεύουν πολλά είδη βιντεοπαιχνιδιών, όπως MMORPG, MMOFPS, MMORTS, MMOTBS και MMO παιχνίδια προσομοίωσης.

1.5 Adventure

Τα παιχνίδια περιπέτειας ήταν μερικά από τα πρώτα παιχνίδια που δημιουργήθηκαν, ξεκινώντας με την περιπέτεια κειμένου Κολοσσιαία Περιπέτεια Σπηλαίων στη δεκαετία του 1970. Αυτό το παιχνίδι είχε αρχικά τίτλο απλά "Περιπέτεια", και είναι ο συνονόματος του είδους. Με την πάροδο του χρόνου, τα γραφικά έχουν εισαχθεί στο είδος και η διεπαφή έχει εξελιχθεί.Σε αντίθεση με τις ταινίες περιπέτειας, τα παιχνίδια περιπέτειας δεν ορίζονται από την ιστορία ή το περιεχόμενο. Αντίθετα, η περιπέτεια περιγράφει έναν τρόπο παιχνιδιού χωρίς αντανακλαστικές προκλήσεις ή δράση. Απαιτούν κανονικά ο παίκτης να λύσει διάφορους γρίφους με την αλληλεπίδραση με τους ανθρώπους ή το περιβάλλον, τις περισσότερες φορές σε ένα μη συγκρουσιακό τρόπο. Θεωρείται ένα "καθαρτικό" είδος και τείνει να αποκλείσει οτιδήποτε περιλαμβάνει στοιχεία δράσης πέρα από ένα μίνι παιχνίδι.

1.6 Role-Playing

Τα βιντεοπαιχνίδια που παίζουν ρόλους αντλούν το παιχνίδι τους από παραδοσιακά επιτραπέζια παιχνίδια ρόλων όπως το Dungeons & Dragons. Τα περισσότερα από αυτά τα παιχνίδια δίνουν στον παίκτη τον ρόλο ενός αδύναμου χαρακτήρα που μεγαλώνει σε δύναμη και εμπειρία κατά τη διάρκεια του παιχνιδιού. Ξεπερνώντας δύσκολες προκλήσεις και/ή νικώντας τέρατα, ο παίκτης κερδίζει πόντους εμπειρίας που αντιπροσωπεύουν την πρόοδο του χαρακτήρα σε ένα επιλεγμένο επάγγελμα ή κατηγορία (όπως μάχη σώμα με σώμα ή μαγικά ξόρκια) και επιτρέπει στον παίκτη να αποκτήσει νέες ικανότητες μόλις ληφθεί ένα καθορισμένο ποσό. Πολλά RPGs περιέχουν έναν ανοικτό κόσμο γνωστό ως over world, ο οποίος συνήθως κατοικείται με τέρατα, που επιτρέπει την πρόσβαση σε πιο σημαντικές τοποθεσίες παιχνιδιών, όπως πόλεις, μπουντρούμια και κάστρα. Δεδομένου ότι η εμφάνιση των υπολογιστών στο σπίτι συνέπεσε με τη δημοτικότητα του χαρτιού και μολυβιού στα παιχνίδια ρόλων, αυτό το είδος ήταν ένα από τα πρώτα σε βιντεοπαιχνίδια και συνεχίζει να είναι δημοφιλής σήμερα. Στοιχεία παιχνιδιού που συνδέονται έντονα με RPG, όπως η ανάπτυξη στατιστικών χαρακτήρων μέσω της απόκτησης των σημείων

εμπειρίας, έχουν προσαρμοστεί ευρέως σε άλλα είδη, όπως τα παιχνίδια δράσης-περιπέτειας. Αν και σχεδόν όλες οι πρώτες καταχωρήσεις στο είδος ήταν turn-based παιχνίδια, πολλά σύγχρονα παιχνίδια ρόλων έχουν πρόοδο σε πραγματικό χρόνο. Έτσι, το είδος έχει ακολουθήσει την τάση του παιχνιδιού στρατηγικής της μετάβασης από τη σειρά με βάση την καταπολέμηση σε πραγματικό χρόνο. Η μετάβαση στην μάχη σε πραγματικό χρόνο ξεκίνησε με την κυκλοφορία του Final Fantasy IV της Square (νυν Square Enix), το πρώτο παιχνίδι που χρησιμοποίησε το σύστημα Active Time Battle. Αυτό ακολουθήθηκε γρήγορα από άλλα παιχνίδια ρόλων όπως η σειρά Mana, Soul Blazer και Ultima VII.

1.7 Simulation

Προσομοίωση βιντεοπαιχνιδιών είναι μια ποικίλη σούπερ-κατηγορία παιχνιδιών, γενικά έχει σχεδιαστεί για να προσομοιώσει στενά πτυχές μιας πραγματικής ή φανταστικής πραγματικότητας.

1.8 Strategy

Τα βιντεοπαιχνίδια στρατηγικής επικεντρώνονται στο gameplay που απαιτεί προσεκτική και επιδέξια σκέψη και σχεδιασμό για να επιτευχθεί η νίκη και οι κλίμακες δράσης από την παγκόσμια κυριαρχία στην ομάδα με βάση την τακτική. "Στα περισσότερα βιντεοπαιχνίδια στρατηγικής", λέει ο Andrew Rollings, "ο παίκτης έχει μια θεϊκή άποψη του κόσμου του παιχνιδιού, έμμεσα τον έλεγχο των μονάδων υπό τις διαταγές του." Το Rollings σημειώνει επίσης ότι «η προέλευση των παιχνιδιών στρατηγικής είναι ριζωμένη στα στενά ξαδέρφια τους, τα επιτραπέζια παιχνίδια.» Τα βιντεοπαιχνίδια στρατηγικής λαμβάνουν γενικά μία από τις τέσσερις αρχετυπικές μορφές, ανάλογα με το αν το παιχνίδι βασίζεται σε σειρά ή σε πραγματικό χρόνο και αν η εστίαση του παιχνιδιού είναι στη στρατηγική ή την τακτική. Τα παιχνίδια στρατηγικής σε πραγματικό χρόνο είναι συχνά ένα παιχνίδι επιλογής πολλαπλών μονάδων (πολλοί χαρακτήρες παιχνιδιών μπορούν να επιλεγούν ταυτόχρονα για την εκτέλεση διαφορετικών εργασιών, σε αντίθεση με την επιλογή μόνο ενός χαρακτήρα κάθε φορά) με μια προβολή ουρανού (προβολή που κοιτάζει προς τα κάτω από πάνω), αλλά μερικά πρόσφατα παιχνίδια όπως το EndWar του Tom Clancy, είναι επιλογή μίας μονάδας και προβολή τρίτου προσώπου. Όπως πολλά RPG παιχνίδια, πολλά παιχνίδια στρατηγικής σταδιακά απομακρύνονται από συστήματα που βασίζονται στη σειρά σε περισσότερα συστήματα σε πραγματικό χρόνο.

1.9 Sports

Τα βιντεοπαιχνίδια αθλημάτων που προσομοιώνουν τον αθλητισμό. Αυτή η αντίπαλη ομάδα μπορεί να ελεγχθεί από άλλους ανθρώπους της πραγματικής ζωής ή τεχνητή νοημοσύνη.

1.10 Multiplayer

Ένα παιχνίδι βίντεο για πολλούς παίκτες είναι ένα βιντεοπαιχνίδι στο οποίο περισσότερα από ένα άτομα μπορούν να παίξουν στο ίδιο περιβάλλον παιχνιδιού ταυτόχρονα, είτε τοπικά (π.χ. New Super Mario Bros. Wii) είτε online μέσω του διαδικτύου (π.χ. World of Warcraft, Call Of Duty). Τα παιχνίδια για πολλούς παίκτες συνήθως απαιτούν από τους παίκτες να μοιράζονται τους πόρους ενός συστήματος παιχνιδιού ή να χρησιμοποιούν την τεχνολογία δικτύωσης για να παίζουν μαζί σε μεγαλύτερη απόσταση. οι παίκτες μπορούν να ανταγωνιστούν έναν ή περισσότερους ανθρώπινους διαγωνιζόμενους, να συνεργαστούν με έναν ανθρώπινο σύντροφο για την επίτευξη ενός κοινού στόχου ή να επιβλέπουν τη δραστηριότητα άλλων παικτών. Τα παιχνίδια για πολλούς παίκτες επιτρέπουν στους παίκτες αλληλεπίδραση με άλλα άτομα σε συνεργασία, ανταγωνισμό ή αντιπαλότητα, παρέχοντάς τους κοινωνική επικοινωνία που απουσιάζει από τα παιχνίδια ενός παίκτη.

ΚΕΦΑΛΑΙΟ 2ο: Ιστορική αναδρομή του gaming

Η ιστορία των βιντεοπαιχνιδιών πηγαίνει τόσο πίσω όσο και στις αρχές της δεκαετίας του 1950, όταν Ακαδημαικοί επιστήμονες υπολογιστών άρχισαν να σχεδιάζουν απλά παιχνίδια και προσομοιώσεις ως μέρος της έρευνάς τους ή μόνο για διασκέδαση. Στο MIT τη δεκαετία του 1960, καθηγητές και φοιτητές έπαιξαν παιχνίδια όπως 3D tic-tac-toe και Σελήνη προσγείωση. Αυτά τα παιχνίδια έπαιξαν σε υπολογιστή όπως το IBM 1560, και οι κινήσεις έγιναν μέσω των punched card*(A punched card ή punch card είναι ένα κομμάτι άκαμπτου χαρτιού που μπορεί να χρησιμοποιηθεί για να περιέχει ψηφιακά δεδομένα που αντιπροσωπεύονται από την παρουσία ή την απουσία οπών σε προκαθορισμένες θέσεις. Τα ψηφιακά δεδομένα μπορούν να χρησιμοποιηθούν για εφαρμογές επεξεργασίας δεδομένων ή, σε προηγούμενα παραδείγματα, που χρησιμοποιούνται για τον άμεσο έλεγχο αυτοματοποιημένων μηχανημάτων.). Το βίντεο Gaming δεν είχε φτάσει στην επικρατούσα δημοτικότητα μέχρι τη δεκαετία του 1970 και του 1980, όταν τα παιχνίδια βιντεοπαιχνιδιών και οι κονσόλες παιχνιδιών με χρήση joystick, κουμπιών και άλλων ελεγκτών, μαζί με γραφικά σε οθόνες υπολογιστών και παιχνίδια οικιακού υπολογιστή, εισήχθησαν στο γενικό δημόσιο. Από τη δεκαετία του 1980, το βίντεο gaming έχει γίνει μια δημοφιλής μορφή ψυχαγωγίας και ένα μέρος της σύγχρονης λαϊκής κουλτούρας στα περισσότερα μέρη του κόσμου. Ένα από τα πρώτα παιχνίδια ήταν το Spacewar,το οποίο αναπτύχθηκε από επιστήμονες υπολογιστών. Τα arcade βιντεοπαιχνίδια αναπτύχθηκαν από 1972 εως το 1978. Κατά τη διάρκεια της δεκαετίας του 1970, η πρώτη γενιά των οικιακών κονσόλων αναδύθηκε, συμπεριλαμβανομένου του δημοφιλούς Game Pong και διαφόρων "κλωνοποιημένων". Η δεκαετία του 1970 ήταν επίσης η εποχή των mainframe computer games. Η χρυσή εποχή των βιντεοπαιχνιδιών arcade ήταν από 1978 έως 1982.Τα Arcades με μεγάλα,διακοσμημένα γραφικά μηχανήματα με κερματοδέκτες ήταν συνηθισμένα σε εμπορικά κέντρα και δημοφιλείς, προσιτές οικιακές κονσόλες, όπως το Atari 2600 και το Intellivision, επέτρεψαν στους χρήστες να παίξουν παιχνίδια στις οικιακές τηλεοράσεις τους. Κατά τη διάρκεια της δεκαετίας του 1980, εμφανίστηκαν υπολογιστές τυχερών παιχνιδιών, νωρίς online gaming και φορητά παιχνίδια LCD. Αυτή η εποχή επηρεάστηκε από την συντριβή των video games των 1983. Από το 1976 έως 1992, εμφανίστηκε η δεύτερη γενιά κονσόλων video games. Η τρίτη γενιά κονσόλων, που ήταν μονάδες 8-bit, αναδύθηκαν από 1983 σε 1995. Η τέταρτη γενιά κονσολών, που ήταν μοντέλα 16-bit, αναδύθηκαν από 1987 σε 1999. Η δεκαετία του 1990 είδε την αναβίωση και την παρακμή των στοών, τη μετάβαση σε βιντεοπαιχνίδια 3D, βελτιωμένα παιχνίδια χειρός και gaming PC. Η πέμπτη γενιά κονσόλων, που ήταν 32 και 64-bit μονάδες, ήταν από 1993 έως 2006. Κατά τη διάρκεια αυτής της εποχής, εμφανίστηκαν παιχνίδια κινητού τηλεφώνου. Κατά τη διάρκεια της δεκαετίας του 2000, προέκυψε η έκτη γενιά κονσόλων (1998 – 2013). Κατά τη διάρκεια αυτής της περιόδου, το online gaming και τα κινητά παιχνίδια έγιναν σημαντικές πτυχές της κουλτούρας των τυχερών παιχνιδιών. Η έβδομη γενιά κονσόλων ήταν από 2005 έως 2012. Αυτή η εποχή σημαδεύτηκε από τεράστιους προϋπολογισμούς ανάπτυξης για ορισμένα παιχνίδια, με ορισμένα κινηματογραφικά γραφικά. την εκκίνηση της κονσόλας Wii με την κορυφαία πώληση, στην οποία ο χρήστης μπορούσε να ελέγχει τις ενέργειες του παιχνιδιού με κίνηση πραγματικού βίου του υπευθύνου επεξεργασίας. η

αύξηση των περιστασιακών gaming PC που διατίθωνταν στο εμπόριο σε μη παίκτες; [απαιτείται μνεία] και την εμφάνιση του cloud computing σε βιντεοπαιχνίδια. Το 2013 εμφανίστηκε η όγδοη γενιά κονσόλας, όπως το Wii της Nintendo, Wii U και το Nintendo 3DS, το Xbox One της Microsoft και το PlayStation 4 και το PlayStation Vita της Sony. Το gaming PC κατέχει μεγάλο μερίδιο αγοράς στην Ασία και την Ευρώπη για δεκαετίες και συνεχίζει να αυξάνεται λόγω της ψηφιακής διανομής. Δεδομένου ότι η ανάπτυξη και η εκτεταμένη χρήση των κινητών τηλεφώνων από τους καταναλωτές, τα παιχνίδια για κινητές συσκευές είναι ένας παράγοντας οδήγησης των παιχνιδιών, καθώς μπορούν να προσεγγίσουν άτομα που άλλοτε δεν ενδιαφέρονται για τα τυχερά παιχνίδια και εκείνοι που δεν μπορούν να αντέξουν οικονομικά ή να υποστηρίξουν ειδικό υλικό, όπως κονσόλες βιντεοπαιχνιδιών.

ΚΕΦΑΛΑΙΟ 3^{ο}: Στατιστικές μελέτες για το gaming

Η αξία της αγοράς βιντεοπαιχνιδιών στις Ηνωμένες Πολιτείες εκτιμήθηκε σε 17.690.000.000 δολάρια ΗΠΑ το 2016, ενώ ολόκληρη η παγκόσμια αγορά βιντεοπαιχνιδιών εκτιμήθηκε στα 75.000.000.000 δολάρια των ΗΠΑ το ίδιο έτος. Στις Η.Π.Α., το λογισμικό δημιουργεί περίπου το ήμισυ των εσόδων της βιομηχανίας βιντεοπαιχνιδιών, ενώ τα αξεσουάρ έχουν λογαριασμό για τα υπόλοιπα. Τα τελευταία οκτώ χρόνια, η μείωση των φυσικών πωλήσεων βιντεοπαιχνιδιών συνοδεύεται από αύξηση του μεριδίου της ψηφιακής μορφής. στο 2009, το 80 τοις εκατό των βιντεοπαιχνιδιών που πουλήθηκαν ήταν φυσικά αντίγραφα και το 20% ήταν ψηφιακά. Κατά 2016, ο διαχωρισμός ευνοούσε σε μεγάλο βαθμό την ψηφιακή διανομή. Οι μεγαλύτεροι παίκτες στην αγορά περιλαμβάνουν εταιρείες όπως η Sony, Activision, Blizzard, Electronic Arts,και η Ubisoft, για να ονομάσουμε λίγες μόνο. Το 2016, η Sony gaming ψυχαγωγίας βρήκε το δρόμο της για την αίθουσα φήμης των gaming εταιρειών, με έσοδα από παιχνίδια 7.800.000.000 δολαρίων ΗΠΑ, δεύτερη-μόνο στα 10.200.000.000 έσοδα από τα παιχνίδια του Tencent's, καθώς και τη Microsoft από τις 1.300.000.000 και τη Nintendo από 6.000.000.000 Αμερικάνικα δολάρια εκείνη τη χρονιά.

Το 2015, το 49% των Αμερικανών καταναλωτών είπε ότι είχε παίξει ένα βιντεοπαιχνίδι τουλάχιστον μία φορά στη ζωή του. Οι άνδρες εξακολουθούν να είναι το κυρίαρχο φύλο μεταξύ των παικτών, και ενώ υπήρξε μια εποχή κατά την οποία το ποσοστό των γυναικών παικτών είχε δει μια αύξηση, φαίνεται ότι οι άνδρες αναδιεκδικούν την πλειοψηφία τους ως παίκτες. Ο μέσος

Αμερικανός παίκτης ξόδεψε σχεδόν 29 ώρες με μια κονσόλα παιχνιδιού κάθε μήνα. Σε εβδομαδιαία βάση, το 27 τοις εκατό των χρηστών της κονσόλας ξόδεψαν κατά μέσο όρο τρεις έως πέντε ώρες παίζοντας παιχνίδια σε αυτήν τη συσκευή. Παρά το γεγονός ότι το 2012 είδε ταχεία μείωση των καταναλωτικών δαπανών σε 20.770.000.000 δολάρια ΗΠΑ, από 24.820.000.000 σε 2011 ή 25.130.000.000 σε 2010, τα πρόσφατα στοιχεία για τις καταναλωτικές δαπάνες στα παιχνίδια δείχνουν ότι η βιομηχανία μπορεί να ξαναμαζεύει, καθώς 2016 ήταν να δει την εικόνα να αυξάνετε σε 30.400.000.000 δολάρια ΗΠΑ. Η αποκλειστική παγκόσμια έρευνα καταναλωτών των στατιστικών διατηρεί περισσότερες πληροφορίες σχετικά με τη συμπεριφορά βιντεοπαιχνιδιών των χρηστών Διαδικτύου ενηλίκων στις Ηνωμένες Πολιτείες. Ο πλήρης κατάλογος της έρευνας για τους καταναλωτές παρέχει λεπτομερή στοιχεία σχετικά με τις προτιμήσεις των συσκευών, τις συνήθειες δαπανών παικτών, τις συχνά χρησιμοποιούμενες πλατφόρμες ροής, καθώς και την ένταση του παιχνιδιού και τις εβδομαδιαίες ώρες που αφιερώνονται στη δραστηριότητα. Για περισσότερες εμπεριστατωμένες πληροφορίες, κάθε σημείο δεδομένων αναλύονται από διάφορες ομάδες-στόχους, ώστε να απεικονίζουν καλύτερα συγκεκριμένες συνήθειες τυχερών παιχνιδιών. Όλα αυτά και πολλά περισσότερα μπορούν να διερευνηθούν στην παγκόσμια έρευνα για τους καταναλωτές.

ΚΕΦΑΛΑΙΟ 4^{ο}: Gamers and gaming behavior

Τα παιδιά που παίζουν βιντεοπαιχνίδια για έως και μια ώρα την ημέρα είναι πιο ευτυχισμένα, πιο κοινωνικά και λιγότερο υπερκινητικά από εκείνους που δεν παίζουν καθόλου,(η έρευνα αυτή βρήκε ότι) παρά τους γενικευμένες φόβους ότι η χρήση βιντεοπαιχνιδιών είναι επιβλαβής, μια μελέτη του Πανεπιστημίου της Οξφόρδης για αγόρια και κορίτσια ηλικίας μεταξύ 10 και 15 βρέθηκαν ότι παίζοντας για έως και 60 λεπτά την ημέρα θα μπορούσε πραγματικά να είναι ευεργετική. Η μελέτη δοκιμάστηκε σε σχεδόν 5.000 παιδιά, συγκρίνοντας εκείνους που δεν έπαιζαν καθόλου με εκείνους που ξόδεψαν διάφορα χρονικά διαστήματα παίζοντας παιχνίδια κονσόλας, όπως το Nintendo Wii και το Sony PlayStation, ή παιχνίδια με βάση τον υπολογιστή. "Οι νέοι που είχαν επιδοθεί σε ένα μικρό παιχνίδι βιντεοπαιχνιδιών συσχετίστηκαν με την καλύτερη προσαρμογή από εκείνους που δεν έπαιξαν ποτέ ή με εκείνους που βρίσκονταν σε βιντεοπαιχνίδια για τρεις ώρες ή περισσότερο", κατέληξε στο συμπέρασμα. "Όσοι έπαιζαν βιντεοπαιχνίδια για λιγότερο από μια ώρα... συσχετίστηκαν με τα υψηλότερα επίπεδα κοινωνιμότητας και ήταν πιθανότερο να πουν ότι ήταν ικανοποιημένοι με τη ζωή τους. Εμφανίστηκαν επίσης να έχουν λιγότερη φιλία και συναισθηματικά προβλήματα, και ανέφεραν λιγότερο υπερκινητικότητα από τις άλλες ομάδες. "

Εκείνοι που έπαιζαν τέτοια παιχνίδια για μεγαλύτερο χρονικό διάστημα, αλλά ακόμα "μέτριες", περίοδους μεταξύ μίας και τρεις ώρες την ημέρα φάνηκε να μην αντιμετωπίζαν καμία επίδραση, είτε θετική είτε αρνητική.

Ωστόσο, εκείνοι που έπαιζαν «υπερβολικά» για περισσότερο από τρεις ώρες την ημέρα είδαν κάποια επιβλαβή αποτελέσματα.

Η μελέτη ανέφερε ότι υπάρχουν καλοί λόγοι για να σκεφτεί κανείς ότι η αναπαραγωγή παιχνιδιού θα μπορούσε να έχει ευεργετικά αποτελέσματα σε σύγκριση με τη μη διαδραστική ψυχαγωγία, όπως η παρακολούθηση τηλεόρασης.

"Τα παιχνίδια παρέχουν ένα ευρύ φάσμα νέων γνωστικών προκλήσεων, ευκαιρίες για εξερεύνηση, χαλάρωση και κοινωνικοποίηση με ομότιμους", ανέφερε. "Όπως οι μη ψηφιακά μεσολαβούμενες μορφές παιδικού παιχνιδιού, τα παιχνίδια μπορεί να ενθαρρύνουν την ευημερία των παιδιών και την υγιή κοινωνική προσαρμογή."

Η μελέτη, η οποία δημοσιεύθηκε στο περιοδικό Παιδιατρική, εκτιμά ότι τρία από τα τέσσερα βρετανικά παιδιά και εφήβους παίζουν βιντεοπαιχνίδια σε καθημερινή βάση.

Αναφέρει ότι η προηγούμενη έρευνα δείχνει ότι περίπου οι μισοί νέοι άνθρωποι στο Ηνομένο Βασίλειο είναι "μικροί παίκτες" του είδους που προτείνει η μελέτη είναι επωφελής.

Σχεδόν το ένα τρίτο των παιδιών περνούν από μία έως τρεις ώρες, ενώ μεταξύ 10 και 15 τοις εκατό περνούν πάνω από τρεις ώρες την ημέρα σε τέτοια ηλεκτρονικά παιχνίδια.

(Μόνο εκείνοι στην τελευταία κατηγορία φάνηκαν να υποφέρουν από επιβλαβείς επιπτώσεις και να είναι λιγότερο καλά προσαρμόστηκαν, διαπίστωσε η μελέτη του Πανεπιστημίου της Οξφόρδης.)

Πρότεινε ότι "αυτό μπορεί να οφείλεται στο γεγονός ότι αστοχούν σε άλλες δραστηριότητες εμπλουτισμού και ενδέχεται να εκτεθούν σε ακατάλληλο περιεχόμενο σχεδιασμένο για ενήλικες".

Η μελέτη διαπίστωσε ότι τα αποτελέσματα είτε ήταν μικρά, "υποδεικνύοντας ότι τόσο οι γενικοί φόβοι και οι ελπίδες για τα παιχνίδια μπορεί να είναι υπερβολικές".

Άλλοι παράγοντες, όπως "το αν το παιδί προέρχεται από μια λειτουργική οικογένεια, οι σχολικές σχέσεις τους, και το αν είναι ουσιωδώς στερημένοι" είχαν πολύ πιο σημαντικό αντίκτυπο στη συμπεριφορά τους.

Ο Δρ Άντριου Πρπρμπιλάνσκι, συντάκτης της έκθεσης, ανέφερε ότι "τα υψηλά επίπεδα παιχνιδιού βιντεοπαιχνιδιών φαίνεται να συνδέονται μόνο ασθενώς με τα προβλήματα συμπεριφοράς των παιδιών στον πραγματικό κόσμο".

"Ομοίως, τα μικρά, θετικά αποτελέσματα που παρατηρήσαμε για τα χαμηλά επίπεδα παιχνιδιού σε ηλεκτρονικά παιχνίδια δεν υποστηρίζουν την ιδέα ότι τα βιντεοπαιχνίδια από μόνα τους μπορούν να βοηθήσουν τα παιδιά να αναπτυχθούν σε έναν ολοένα και πιο ψηφιακό κόσμο", ανέφερε.

Ζήτησε περαιτέρω έρευνα για να προσδιορίσει ποια είδη παιχνιδιών ήταν ευεργετικά ή επιβλαβή και είπε ότι τα συνιστώμενα χρονικά όρια στο παιχνίδι βιντεοπαιχνιδιών είχαν "μικρή επιστημονική βάση".

ΚΕΦΑΛΑΙΟ 5^{ο}: Τεχνιτή Νοημοσύνη

5.1 Τεχνιτή Νοημοσύνη Στα Video Games

Στα βιντεοπαιχνίδια, η τεχνητή νοημοσύνη (AI) χρησιμοποιείται για τη δημιουργία αποκριτικών, προσαρμοστικών ή έξυπνων συμπεριφορών κυρίως σε χαρακτήρες που δεν είναι παίκτες (NPCs) όμοιοι με την ανθρώπινη νοημοσύνη. Η τεχνητή νοημοσύνη αποτελεί αναπόσπαστο μέρος των βιντεοπαιχνιδιών από την ίδρυσή της τη δεκαετία του 1950. Ο ρόλος της τεχνητής νοημοσύνης στα βιντεοπαιχνίδια έχει επεκταθεί σε μεγάλο βαθμό από την εισαγωγή της. Τα σύγχρονα παιχνίδια συχνά εφαρμόζουν υπάρχουσες τεχνικές από τον τομέα της τεχνητής νοημοσύνης, όπως η εύρεση και τα δέντρα αποφάσεων για να καθοδηγήσουν τις ενέργειες των NPCs. Επιπλέον, η τεχνητή νοημοσύνη χρησιμοποιείται συχνά σε μηχανισμούς που δεν είναι άμεσα ορατοί στο χρήστη, όπως τα δεδομένα και τη διαδικασία παραγωγής περιεχομένου.

Ο όρος "Game AI" χρησιμοποιείται για να αναφέρεται σε ένα ευρύ σύνολο αλγορίθμων που περιλαμβάνουν επίσης τεχνικές από τη θεωρία ελέγχου, τη ρομποτική, τα γραφικά υπολογιστών και την επιστήμη των υπολογιστών γενικά, και έτσι το Gaming AI μπορεί συχνά να μην συνιστά "True AI" στο ότι τέτοιες τεχνικές δεν διευκολύνει αναγκαστικά την εκμάθηση υπολογιστών ή άλλα τυποποιημένα κριτήρια, αποτελώντας μόνο "αυτοματοποιημένο υπολογισμό" ή ένα προκαθορισμένο και περιορισμένο σύνολο απαντήσεων σε ένα προκαθορισμένο και περιορισμένο σύνολο εισροών.

Πολλές βιομηχανίες και εταιρίες ισχυρίζονται ότι το λεγόμενο Gaming AI έχει κάνει πολύ δρόμο με την έννοια ότι έχει επανασταώσει τον τρόπο με τον οποίο οι άνθρωποι αλληλεπιδρούν με όλες τις μορφές της τεχνολογίας, αν και πολλοί ειδικοί ερευνητές είναι σκεπτικοί με τέτοιους ισχυρισμούς, και ιδιαίτερα της ιδέας ότι αυτές οι τεχνολογίες ταιριάζουν στον ορισμό της «ευφυΐας» κατά κανόνα που χρησιμοποιείται στις γνωστικές επιστήμες. οι φωνές της βιομηχανίας κάνουν το επιχείρημα ότι η τεχνητή νοημοσύνη έχει γίνει πιο ευέλικτη στον τρόπο με τον οποίο χρησιμοποιούμε όλες τις τεχνολογικές συσκευές για περισσότερο από τον επιδιωκόμενο σκοπό τους, επειδή η τεχνητή νοημοσύνη επιτρέπει στην τεχνολογία να λειτουργεί με πολλαπλούς τρόπους, φέρεται να αναπτύσσει τη δικιά της προσωπικότητα και την πραγματοποίηση σύνθετων οδηγιών του χρήστη.

Ωστόσο, στον τομέα της τεχνητής νοημοσύνης έχουν ισχυριστεί ότι το βιντεοπαιχνίδι AI δεν είναι αληθινή νοημοσύνη, αλλά μια λέξη κλιπ που χρησιμοποιείται για να περιγράψει προγράμματα υπολογιστών που χρησιμοποιούν απλή διαλογή και ταιριαστούς αλγόριθμους για να δημιουργήσει την ψευδαίσθηση της ευφυούς συμπεριφοράς, δηλαδή ένα λογισμικό με παραπλανητικό σκοπό

5.2 Η ιστορία της Τεχνιτής Νοημοσύνης

Τα παιχνίδια ήταν μια περιοχή έρευνας στην τεχνητή νοημοσύνη από την ίδρυσή της. Ένα από τα πρώτα παραδείγματα της τεχνητής νοημοσύνης είναι το μηχανογραφημένο παιχνίδι του Νιμ που έγινε το 1951 και δημοσιεύθηκε το 1952. Παρά το γεγονός ότι είναι προηγμένη τεχνολογία κατά τη διάρκεια του έτους που έγινε, 20 χρόνια πριν από το Pong, το παιχνίδι πήρε τη μορφή ενός σχετικά μικρού κουτιού και ήταν σε θέση να κερδίσει τακτικά παιχνίδια ακόμη και κατά των εξαιρετικά ειδικευμένων παικτών του παιχνιδιού. Το 1951, με τη χρήση του μηχανήματος Ferranti Mark 1 του Πανεπιστημίου του Μάντσεστερ, ο Κρίστοφερ Στράτσεϊ έγραψε ένα πρόγραμμα για τη ντάμα και ο Ντίτριχ Πρενς έγραψε ένα για το σκάκι, αυτά ήταν μεταξύ των πρώτων προγραμμάτων υπολογιστών που γράφτηκαν ποτέ. Το πρόγραμμα του Άρθουρ Σάμιουελ, που αναπτύχθηκε στα μέσα της δεκαετίας του ' 50 και στις αρχές του ' 60, τελικά πέτυχε επαρκή επιδεξιότητα για να αμφισβητήσει έναν αξιοσέβαστο ερασιτέχνη. Οι εργασίες για τη ντάμα και το σκάκι θα καταλήξουν στην ήττα του Γκάρι Κασπαρόφ από τον υπολογιστή Deep Blue της IBM το 1997. Τα πρώτα βιντεοπαιχνίδια που αναπτύχθηκαν στη δεκαετία του 1960 και στις αρχές της δεκαετίας του 1970, όπως το διαστημικό Μουσείο!, Pong, και σας έπιασα (1973), ήταν παιχνίδια που εφαρμόστηκαν σε διακριτή λογική και αυστηρά βασισμένη στον ανταγωνισμό δύο παικτών, χωρίς τεχνητή νοημοσύνη.

Παιχνίδια που χαρακτήριζαν μια λειτουργία ενός παίκτη με εχθρούς άρχισαν να εμφανίζονται στη δεκαετία του 1970. Τα πρώτα αξιοσημείωτα για το arcade εμφανίστηκε στο 1974: ο αγώνας Ταίτο παιχνίδι ταχύτητας (αγωνιστικά παιχνίδι βίντεο) και το Ατάρι παιχνίδια Qwak (κυνήγι πάπια ελαφρύ πιστόλι σκοπευτή) και την καταδίωξη (μαχητικό αεροσκάφος κυνομαχίες προσομοιωτή). Δύο παιχνίδια με βάση το κείμενο από το 1972, το κυνήγι του Wumpus και το Star Trek, είχαν επίσης εχθρούς. Το εχθρικό κίνημα βασίστηκε σε αποθηκευμένα μοτίβα. Η ενσωμάτωση των μικρομεταποιητών θα επέτρεπε περισσότερο υπολογισμό και τυχαία στοιχεία που έχουν υπερκαθοριστεί σε μοτίβα κίνησης.

Ήταν κατά τη διάρκεια της χρυσής εποχής των βιντεοπαιχνιδιών (βιντεοπαιχνίδια) ότι η ιδέα των αντιπάλων της τεχνητής νοημοσύνης ήταν σε μεγάλο βαθμό διαδόθηκε, λόγω της επιτυχίας των διαστημικών Εισδρομείς (1978), η οποία είχε ένα αυξανόμενο επίπεδο δυσκολίας, διακριτά μοτίβα κίνησης, και συμβάντα εντός του παιχνιδιού που εξαρτώνται από τον κατακερματισμό λειτουργίες με βάση την είσοδο του παίκτη. Το γαλαξιάν (1979) πρόσθεσε πιο περίπλοκα και ποικίλα εχθρικά κινήματα, συμπεριλαμβανομένων των ελιγμών από μεμονωμένους εχθρούς που βγαίνουν από το σχηματισμό.

Το Pac-Man (1980) εισήγαγε μοτίβα τεχνητής νοημοσύνης σε παιχνίδια Λαβύρινθου, με την προστιθέμενη ιδιικότητα διαφορετικών προσωπικοτήτων για κάθε εχθρό. Ο πρωταθλητής Καράτε (1984) εισήγαγε αργότερα μοτίβα AI για την καταπολέμηση των παιχνιδιών, αν και η φτωχή AI προκάλεσε την απελευθέρωση μιας δεύτερης έκδοσης. Η πρώτη βασίλισσα (1988) ήταν μια τακτική δράση που χαρακτήρισε τους χαρακτήρες που μπορούν να ελεγχθούν από την τεχνητή νοημοσύνη του υπολογιστή ακολουθώντας τον ηγέτη. Ο ρόλος παίζοντας βίντεο παιχνίδι Dragon αναζήτηση IV (1990) εισήγαγε ένα σύστημα "τακτική", όπου ο χρήστης μπορεί να προσαρμόσει τις AI ρουτίνες των χαρακτήρων μη-παίκτη κατά τη διάρκεια της μάχης, μια ιδέα αργότερα εισήχθη στο είδος δράσης-παίζοντας παιχνίδι ρόλων από Secret της Mana (1993).

Παιχνίδια όπως το ποδόσφαιρο Μάντεν, ο Ερλ Γουίβερ μπέιζμπολ και ο Tony La Russa μπέιζμπολ όλα βασισμένα στην τεχνητή νοημοσύνη τους σε μια προσπάθεια να αναπαράγουν στον υπολογιστή το προπονητικό ή διευθυντικό στυλ της επιλεγμένης διασημότητας. Ο Μάντεν, ο Γουίβερ και η La Russa έκαναν εκτεταμένη εργασία με αυτές τις ομάδες ανάπτυξης παιχνιδιών για να μεγιστοποιήσουν την ακρίβεια των αγώνων. Οι νεότεροι αθλητικοί τίτλοι επέτρεψαν στους χρήστες να "συντονίζουν" μεταβλητές στην τεχνητή νοημοσύνη για την παραγωγή.

Η εμφάνιση νέων ειδών παιχνιδιών στη δεκαετία του 1990 ώθησε στη χρήση επίσημων εργαλείων τεχνητής νοημοσύνης όπως μηχανήματα πεπερασμένων καταστάσεων. Τα παιχνίδια στρατηγικής σε πραγματικό χρόνο φορολογούν την τεχνητή νοημοσύνη με πολλά αντικείμενα, ελλιπείς πληροφορίες, προβλήματα με την εύρεση προβλημάτων, αποφάσεις σε πραγματικό χρόνο και οικονομικό προγραμματισμό, μεταξύ άλλων οι πρώτοι αγώνες του είδους είχαν διαβόητα προβλήματα. Ο Χέρτζογκ Ζγουέι (1989), για παράδειγμα, είχε σχεδόν σπάσει το εύρημα και πολύ βασικά τρία-κρατικά μηχανήματα για τον έλεγχο της μονάδας, και οι αμμόλοφους II (1992) επιτέθηκαν στη βάση των παικτών σε μια γραμμή και χρησιμοποίησαν αμέτρητους απατεώνες. αργότερα παιχνίδια στο είδος παρουσίασε πιο σοφιστικέ AI.

Αργότερα τα παιχνίδια χρησιμοποιούν μεθόδους τεχνητής νοημοσύνης από κάτω προς τα πάνω, όπως η αναδυόμενη συμπεριφορά και η αξιολόγηση των ενεργειών των παικτών σε παιχνίδια όπως πλάσματα ή μαύρο & λευκό. Πρόσοψη (διαδραστική ιστορία) κυκλοφόρησε στο 2005 και χρησιμοποιείται αλληλεπιδραστικό αμφίδρομο διάλογο και AI ως την κύρια πτυχή του παιχνιδιού.

Τα παιχνίδια έχουν παράσχει ένα περιβάλλον για την ανάπτυξη τεχνητής νοημοσύνης με πιθανές εφαρμογές πέρα από το παιχνίδι. Παραδείγματα είναι ο Watson, ένας κίνδυνος!-αναπαραγωγή υπολογιστή και το τουρνουά Ρομποκυπέλλου, όπου τα ρομπότ εκπαιδεύονται να ανταγωνίζονται στο ποδόσφαιρο.

5.3 Η ιστορία της Τεχνιτής Νοημοσύνης Στα σύγχρονα βιντεοπαιχνίδια

Τα παιχνίδια AI/ευριστικοί αλγόριθμοι χρησιμοποιούνται σε μια μεγάλη ποικιλία αρκετά ανόμοιων πεδίων μέσα σε ένα παιχνίδι. Το πιο προφανές είναι ο έλεγχος οποιωνδήποτε NPCs στο παιχνίδι, αν και η "δέσμη ενεργειών" (δέντρο αποφάσεων) είναι επί του παρόντος το πιο κοινό μέσο ελέγχου. Αυτές οι χειρόγραφες αποφάσεις δέντρα συχνά οδηγούν σε "τεχνητή ηλιθιότητα" όπως επαναλαμβανόμενη συμπεριφορά, απώλεια εμβάπτισης, ή μη φυσιολογική συμπεριφορά σε καταστάσεις που οι προγραμματιστές δεν σχεδιάζουν.

Μια άλλη κοινή χρήση για την τεχνητή νοημοσύνη, παρατηρείται ευρέως σε παιχνίδια στρατηγικής σε πραγματικό χρόνο. Η αναζήτηση είναι η μέθοδος για τον προσδιορισμό του πώς να πάει ένα NPC από ένα σημείο σε ένα χάρτη σε ένα άλλο, λαμβάνοντας υπόψη το έδαφος, τα εμπόδια και ενδεχομένως "ομίχλη του πολέμου". Τα εμπορικά βιντεοπαιχνίδια συχνά χρησιμοποιούν γρήγορη και απλή «αναζήτηση με βάση το πλέγμα», όπου το έδαφος αντιστοιχίζεται σε ένα άκαμπτο πλέγμα ενιαίων τετραγώνων και ένας αλγόριθμος εύρεσης

διαλογίας όπως A * ή IDA * εφαρμόζεται στο πλέγμα αντί μόνο για ένα άκαμπτο πλέγμα, ορισμένα παιχνίδια χρησιμοποιούν ακανόνιστα πολυγωνικά και συναρμολογούν ένα πλέγμα πλοήγησης έξω από τις περιοχές του χάρτη με τα οποία μπορούν να περπατήσουν οι NPCs, ως τρίτη μέθοδος, είναι μερικές φορές βολικό για τους προγραμματιστές να επιλέξουν χειροκίνητα "σημεία" που θα πρέπει να χρησιμοποιούν τα NPCs για την πλοήγηση. το κόστος είναι ότι τέτοια σημεία μπορεί να δημιουργήσει αφύσικη κίνηση. Επιπλέον, τα σημεία έχουν την τάση να αποδίδουν χειρότερα από τα δικτυώματα πλοήγησης σε σύνθετα περιβάλλοντα πέρα από τη στατική εύρεση, η πλοήγηση είναι ένα υπο-πεδίο του Game AI

5.4 Τεχνιτή Νοημοσύνη σε Βιντεοπαιχνίδι μάχης AI

Πολλά σύγχρονα βιντεοπαιχνίδια εμπίπτουν στην κατηγορία δράσης, τον πρώτο εκτελεστή ή την περιπέτεια. Στους περισσότερους από αυτούς τους τύπους παιχνιδιών υπάρχει κάποιο επίπεδο μάχης που λαμβάνει χώρα. Η ικανότητα του AI να είναι αποτελεσματική στη μάχη είναι σημαντική σε αυτά τα είδη. Ένας κοινός στόχος σήμερα είναι να γίνει η τεχνητή νοημοσύνη πιο ανθρώπινη, ή τουλάχιστον να εμφανιστεί έτσι.

Ένα από τα πιο θετικά και αποτελεσματικά χαρακτηριστικά που βρέθηκαν στο σύγχρονο βιντεοπαιχνίδι AI είναι η ικανότητα να κυνηγούν. Ο AI αρχικά αντέδρασε με πολύ μαύρο και άσπρο τρόπο. Εάν ο παίκτης βρισκόταν σε συγκεκριμένο τομέα, τότε η τεχνητή νοημοσύνη θα αντιδρούσε είτε με πλήρη επιθετικό τρόπο είτε με απόλυτη άμυνα. Τα τελευταία χρόνια έχει εισαχθεί η ιδέα του "κυνηγιού", σε αυτή την "κυνηγετική" κατάσταση, η τεχνητή νοημοσύνη θα ψάχνει για ρεαλιστικούς δείκτες, όπως ήχους που γίνονται από τον χαρακτήρα ή τα ίχνη που μπορεί να έχουν αφήσει πίσω τους. Οι εξελίξεις αυτές επιτρέπουν τελικά μια πιο περίπλοκη μορφή παιχνιδιού. Με αυτό το χαρακτηριστικό, ο παίκτης μπορεί πραγματικά να εξετάσει πώς να προσεγγίσει ή να αποφύγει έναν εχθρό. Αυτό είναι ένα χαρακτηριστικό που είναι ιδιαίτερα διαδεδομένο στο είδος της μυστικότητας.

Μια άλλη εξέλιξη σε πρόσφατο παιχνίδι AI ήταν η ανάπτυξη του "ένστικτου επιβίωσης". Οι υπολογιστές εντός του παιχνιδιού μπορούν να αναγνωρίσουν διαφορετικά αντικείμενα σε ένα περιβάλλον και να καθορίσουν αν είναι ευεργετικό ή επιζήμιο για την επιβίωσή του. Όπως ένας χρήστης, το AI μπορεί να ψάχνει για κάλυψη σε μια μάχη πριν από τη λήψη ενεργειών που θα αφήσει διαφορετικά ευάλωτη, όπως η επαναφόρτωση ενός όπλου ή ρίχνοντας μια χειροβομβίδα. Μπορεί να υπάρχουν δείκτες που να λένε πότε να αντιδράσει με συγκεκριμένο τρόπο. Για παράδειγμα, εάν το AI έχει μια εντολή για να ελένχει την υγεία του σε όλο το παιχνίδι, τότε μπορούν να οριστούν περαιτέρω εντολές έτσι ώστε να αντιδρά με συγκεκριμένο τρόπο σε ένα συγκεκριμένο ποσοστό της υγείας. Εάν η υγεία είναι κάτω από ένα ορισμένο όριο, τότε η AI

μπορεί να ρυθμιστεί για να τρέξει μακριά από τον παίκτη και να το αποφύγει μέχρι να ενεργοποιηθεί μια άλλη λειτουργία. Ένα άλλο παράδειγμα ,θα μπορούσε να είναι εάν το AI παρατηρεί ότι είναι από τις σφαίρες, θα βρεί ένα αντικείμενο κάλυψης και θα κρυφτεί πίσω από το έως ότου έχει ξαναφορτώσει. Ενέργειες σαν αυτές κάνουν την AI να φαίνεται πιο ανθρώπινη. Ωστόσο, εξακολουθεί να υπάρχει ανάγκη βελτίωσης σε αυτόν τον τομέα.

5.5 Τεχνιτή Νοημοσύνη Χρήσεις σε παιχνίδια πέραν των NPCs

Ο Γεώργιος ν. Γιανακάκης προτείνει ότι οι ακαδημαϊκές εξελίξεις της τεχνητής νοημοσύνης μπορούν να παίξουν ρόλους στο παιχνίδι AI πέρα από το παραδοσιακό παράδειγμα της Τεχνιτής συμπεριφοράς του NPC. επισημαίνει τέσσερις άλλους πιθανούς τομείς εφαρμογής:

1.Μοντέλο παίκτη-εμπειρία: να διακρίνει την ικανότητα και τη συναισθηματική κατάσταση του παίκτη, ώστε να προσαρμόσει κατάλληλα το παιχνίδι. Αυτό μπορεί να περιλαμβάνει δυναμικό παιχνίδι εξισορρόπησης δυσκολία, η οποία συνίσταται στην προσαρμογή της δυσκολίας σε ένα βιντεοπαιχνίδι σε πραγματικό χρόνο με βάση την ικανότητα του παίκτη. Το Game AI μπορεί επίσης να βοηθήσει να συμπεράνει την πρόθεση παίκτη (όπως η αναγνώριση χειρονομίας).

2.Διαδικασία-δημιουργία περιεχομένου: δημιουργία στοιχείων του περιβάλλοντος παιχνιδιού, όπως περιβαλλοντικές συνθήκες, επίπεδα, ακόμα και μουσική με αυτοματοποιημένο τρόπο. Οι μέθοδοι AI μπορούν να δημιουργήσουν νέο περιεχόμενο ή διαδραστικές ιστορίες.

3.Εξόρυξη δεδομένων σχετικά με τη συμπεριφορά των χρηστών: Αυτό επιτρέπει στους σχεδιαστές παιχνιδιών να εξερευνήσουν τον τρόπο με τον οποίο οι χρήστες χρησιμοποιούν το παιχνίδι, τα μέρη που παίζουν περισσότερο, και τι τους προκαλεί να σταματήσουν να παίζουν, επιτρέποντας στους προγραμματιστές να συντονίσουν το παιχνίδι ή να βελτιώσουν τη δημιουργία εσόδων.

4.Εναλλακτικές προσεγγίσεις σε NPCs: αυτές περιλαμβάνουν την αλλαγή του παιχνιδιού set-up για να ενισχύσει NPCS πιστευιμότητα και την Εξερεύνηση κοινωνική και όχι ατομική συμπεριφορά NPCS.

Αντί της διαδικασίας παραγωγής, ορισμένοι ερευνητές χρησιμοποίησαν γενετική δίκτυα (GANS) για να δημιουργήσουν νέο περιεχόμενο. Το 2018 ερευνητές στο Πανεπιστήμιο Κορνουάλη εκπαίδευσαν ένα GAN σε χίλια ανθρώπινα επίπεδα για την ΚΑΤΑΣΤΡΟΦΉ (1993); το πρωτότυπο του νευρικού δικτύου ήταν σε θέση να σχεδιάσει νέα επίπεδα αναπαραγωγής

5.6 Εξαπάτηση AI

Στο πλαίσιο της τεχνητής νοημοσύνης στα βιντεοπαιχνίδια, η εξαπάτηση αναφέρεται στον προγραμματιστή που δίνει στους πράκτορες ενέργειες και πρόσβαση σε πληροφορίες που δεν θα ήταν διαθέσιμες στον παίκτη στην ίδια κατάσταση. Πιστεύοντας ότι το Ατάρι 8-bit δεν θα μπορούσε να ανταγωνιστεί εναντίον ενός ανθρώπινου παίκτη, ο Chris Crawford δεν διόρθωσε ένα bug στο ανατολικό μέτωπο (1941) που ωφέλησε την ελεγχόμενη από τον υπολογιστή ρωσική πλευρά. Ο κόσμος τυχερών παιχνιδιών στο 1994 ανέφερε ότι "είναι γνωστό ότι πολλοί AIs" εξαπατούν " για να μπορούν να τηρούν τους ανθρώπινους παίκτες".

Για παράδειγμα, αν οι πράκτορες θέλουν να γνωρίζουν αν ο παίκτης είναι κοντά, μπορούν είτε να τους δοθούν σύνθετοι, ανθρώπινοι αισθητήρες (βλέποντας, ακούγοντας κ. λπ.), είτε να κάνουν ζαβολιες απλά ζητώντας από τη μηχανή του παιχνιδιού τη θέση του παίκτη. Κοινές παραλλαγές περιλαμβάνουν δίνοντας AIs υψηλότερες ταχύτητες σε αγωνιστικά παιχνίδια για να προλάβει τον παίκτη ή την αναπαραγωγή τους σε πλεονεκτικές θέσεις στο πρώτο άτομο σκοπευτή. Η χρήση της εξαπάτησης εφαρμόζεται συχνά για λόγους απόδοσης, όπου σε πολλές περιπτώσεις μπορεί να θεωρηθεί αποδεκτή, εφόσον το αποτέλεσμα δεν είναι προφανές για τον παίκτη. Ενώ η εξαπάτηση αναφέρεται μόνο σε προνόμια που δίνονται ειδικά στο AI-δεν περιλαμβάνει την απάνθρωπη ταχύτητα και ακρίβεια σε έναν υπολογιστή-ένας παίκτης θα μπορούσε να καλέσει εγγενή πλεονεκτήματα του υπολογιστή "εξαπάτηση" αν έχουν ως αποτέλεσμα ο πράκτορας να ενεργεί, σε αντίθεση με έναν ανθρώπινο παίκτη. Ο Sid Meier δήλωσε ότι παρέλειψε τις συμμαχίες multiplayer στον πολιτισμό επειδή διαπίστωσε ότι ο υπολογιστής ήταν σχεδόν τόσο καλός όσο οι άνθρωποι στη χρησιμοποίηση τους, το οποίο ανάγκασε τους φορείς να σκεφτούν ότι ο υπολογιστής εξαπατούσε. Οι υπεύθυνοι για την ανάπτυξη λένε ότι οι περισσότεροι είναι τίμιοι αλλά αντιπαθούν τους φορείς που παραπονιούνται εσφαλμένα για «την εξαπάτηση» AI. Επιπλέον, οι άνθρωποι χρησιμοποιούν τακτικές ενάντια στους υπολογιστές που δεν θα ήταν ενάντια σε άλλους ανθρώπους.

ΚΕΦΑΛΑΙΟ 6^{ο}:Παραδείγματα Παιχνιδιών

6.1 Creatures (1996)

Τα πλάσματα είναι ένα τεχνητό πρόγραμμα ζωής όπου ο χρήστης "καταπακτές" μικρά μαλλιαρά ζώα και τους διδάσκει πώς να συμπεριφέρονται. Αυτά τα "Νόρνς" μπορούν να μιλήσουν, να τραφούν και να προστατευτούν από μοχθηρά πλάσματα. Ήταν η πρώτη δημοφιλής εφαρμογή της μηχανικής μάθησης σε μια διαδραστική προσομοίωση. Τα νευρικά δίκτυα χρησιμοποιούνται από τα πλάσματα για να μάθουν τι να κάνουν. Το παιχνίδι θεωρείται ως μια ανακάλυψη στην τεχνητή έρευνα ζωής, η οποία στοχεύει να διαμορφώσει τη συμπεριφορά των πλασμάτων που αλληλεπιδρούν με το περιβάλλον τους.

Σιντ Μάγιερ Άλφα Κενταύρου (1999)

6.2 Halo: εξέλιξη μάχης (2001)

Ένας εκτελεστής του πρώτου προσώπου, όπου ο παίκτης αναλαμβάνει το ρόλο του πλοιάρχου, πολεμώντας διάφορους εξωγήινους πεζοί ή σε οχήματα. Οι εχθροί χρησιμοποιούν την κάλυψη πολύ σοφά και απασχολούν την καταστολή της φωτιάς και των βομβίδων. Η κατάσταση της ομάδας επηρεάζει τα άτομα, έτσι ορισμένοι εχθροί φεύγουν όταν ο ηγέτης τους πεθαίνει. Δίνεται μεγάλη προσοχή στις μικρές λεπτομέρειες, με εχθρούς που κυρίως ρίχνουν χειροβομβίδες ή μέλη της ομάδας που ανταποκρίνονται σε εσάς. Η υποκείμενη τεχνολογία "δέντρου συμπεριφοράς" έχει γίνει πολύ δημοφιλής στη βιομηχανία παιχνιδιών (ειδικά από το Halo 2).

6.3 F.E.A.R. (2005)

Ένας δράστης πρώτου προσώπου, όπου ο παίκτης βοηθά να περιέχει υπερφυσικό φαινόμενο και στρατούς κλωνοποιημένων στρατιωτών. Η τεχνητή νοημοσύνη χρησιμοποιεί ένα σχεδιαστή για να δημιουργήσει συμπεριφορές ευαίσθητες στο περιβάλλον, την πρώτη φορά σε ένα βασικό παιχνίδι. Αυτή η τεχνολογία εξακολουθεί να χρησιμοποιείται ως αναφορά για πολλά στούντιο. Οι εχθροί είναι ικανοί να χρησιμοποιούν το περιβάλλον πολύ έξυπνα, βρίσκοντας κάλυψη πίσω από τραπέζια, αναποδογυρίζει ράφια, ανοίγοντας πόρτες, συντρίβοντας τα παράθυρα, και ούτω καθεξής. Οι τακτικές της διμοιρίας χρησιμοποιούνται σε μεγάλο αποτέλεσμα. Οι εχθροί εκτελούν πλευρικές ασκήσεις, χρησιμοποιούν καταστολή φωτιάς, κλπ

6.4 Σειρά S.T.A.L.K.E.R. (2007-)

Ένα παιχνίδι τρόμου για την επιβίωση του πρώτου προσώπου, όπου ο παίκτης πρέπει να αντιμετωπίσει πειράματα, στρατιωτικούς στρατιώτες και μισθοφόρους που είναι γνωστοί ως διώκτες. Οι διάφοροι εχθροί που συναντούν (εάν το επίπεδο δυσκολίας έχει οριστεί στο υψηλότερο) χρησιμοποιούν τακτικές μάχης και συμπεριφορές όπως η επούλωση πληγών Συμμάχων, η παροχή παραγγελιών, η παράπλευρα του παίκτη ή η χρήση όπλων με ακρίβεια ακριβείας. [απαιτείται μνεία]

6.5 Far Cry 2 (2008)

Ένας εκτελεστής, όπου ο παίκτης πολεμάει πολλούς μισθοφόρους και δολοφονεί αρχηγούς. Η τεχνητή νοημοσύνη βασίζεται στη συμπεριφορά και χρησιμοποιεί την επιλογή δράσης, απαραίτητη εάν η τεχνητή νοημοσύνη είναι να κάνει πολυεργασίες ή να αντιδρά σε μια κατάσταση. Η τεχνητή νοημοσύνη μπορεί να αντιδράσει με απρόβλεπτο τρόπο σε πολλές περιπτώσεις. Οι εχθροί απαντούν σε ήχους και οπτικές περισπασμούς όπως φωτιά ή κοντινές εκρήξεις και μπορούν να γίνουν για να ερευνήσουν, ο παίκτης χρησιμοποιεί αυτούς τους περισπασμούς προς όφελός τους. Ορισμένοι χαρακτήρες AI που δεν μπορούν να αναπαραχθούν θα αντιδράσουν αρνητικά στην αντιμετώπιση, για παράδειγμα, με το σπρώξιμο, την ορκωμοσία ή την στόχευση του παίκτη με το όπλο του. Οι τραυματισμένοι εχθροί μπορούν να καλούν για βοήθεια ενώ βρίσκονται στο έδαφος, ή να εμφανίζουν δυσφορία, κλπ. [απαιτείται μνεία]

6.6 StarCraft II (2010)

Ένα παιχνίδι στρατηγικής σε πραγματικό χρόνο, όπου ένας παίκτης παίρνει τον έλεγχο μίας από τις τρεις φατρίες σε ένα 1v1, 2v2, ή 3v3 αρένα μάχης. Ο παίκτης πρέπει να νικήσει τους αντιπάλους του καταστρέφοντας όλες τις μονάδες και τις βάσεις τους. Αυτό επιτυγχάνεται με τη δημιουργία μονάδων που είναι αποτελεσματικές στην αντιμετώπιση των μονάδων των αντιπάλων σας. Οι παίκτες μπορούν να παίξουν ενάντια σε πολλαπλά διαφορετικά επίπεδα ΤΕΧΝΗΤΗς δυσκολίας που κυμαίνονται από πολύ εύκολο να Κερατούν 3 (τρελό). Το AI είναι σε θέση να εξαπατήσει με τη δυσκολία απατεώνας 1 (όραμα), όπου μπορεί να δει μονάδες και βάσεις όταν ένας παίκτης στην ίδια κατάσταση δεν θα μπορούσε. Απατεώνας 2 δίνει το AI επιπλέον πόρους, ενώ απατεώνας 3 δώσει ένα εκτεταμένο πλεονέκτημα πάνω από τον αντίπαλό του.

ΚΕΦΑΛΑΙΟ 7^{ο}:Η Ιστορία τηςEpic Games

Η Epic Games Inc. (παλαιότερα συστήματα υπολογιστών Potomac και αργότερα Epic MegaGames, Inc.) είναι μια αμερικανική εταιρεία βιντεοπαιχνιδιών και ανάπτυξης λογισμικού με έδρα τον Κάρι της Βόρειας Καρολίνα. Η εταιρεία ιδρύθηκε από τον Τιμ Σουίνι ως κομπιούτερ Potomac στο 1991, που αρχικά βρισκόταν στο σπίτι των γονιών του στο Ποτόμακ, στο Μέριλαντ. Μετά την πρώτη του εμπορική κυκλοφορία βιντεοπαιχνιδιών, η ZZT (1991), η εταιρεία έγινε επική πρωταμάτης στις αρχές της 1992 και έφερε τον Μαρκ Ράριν, ο οποίος είναι ο Αντιπρόεδρος της εταιρείας μέχρι σήμερα. Μετακινώντας την έδρα τους στον Κάρι το 1999, το όνομα του στούντιο απλοποιήθηκε σε Epic Games.

Η Epic Games αναπτύσσουν την Unreal Engine , ένα εμπορικά διαθέσιμο engine παιχνιδιού που επίσης τροφοδοτεί τα εσωτερικά ανεπτυγμένα βιντεοπαιχνίδια τους, όπως το Fortnite ,το Unreal, Gears of War και Infinity Blade series. Το 2014,η Unreal Engine ονομάστηκε "η πιο επιτυχημένη μηχανή βιντεοπαιχνιδιών" από τα Guinness World Records

Η Epic Games κατέχουν προγραμματιστές παιχνιδιών Chair Entertainment και Psyonix,καθώς και προγραμματιστές λογισμικού που βασίζεται στο cloud Cloudgine, και λειτουργεί επώνυμα sub-Studios στο Σιάτλ, Αγγλία, Βερολίνο, Γιοκοχάμα και Σεούλ. Ενώ ο Σουίνι παραμένει ο πλειοψηφικός μέτοχος, ο Τάνσεντ απέκτησε ένα 48,4% εκκρεμές ποντάρισμα, εξισώνοντας το 40% του συνολικού επικού, στην εταιρεία το 2012, μετά από τα επικά παιχνίδια συνειδητοποίησε ότι η βιομηχανία βιντεοπαιχνιδιών εξελισσόταν σε μεγάλο βαθμό προς τα παιχνίδια ως μοντέλο σέρβις. Μετά την απελευθέρωση του δημοφιλούς Fortnite Battle Royale, το 2017, η εταιρεία απέκτησε πρόσθετες επενδύσεις που επέτρεψαν να επεκτείνει τις προσφορές του Unreal Engine , να δημιουργήσει εκδηλώσεις στο Fortnite και να ξεκινήσει το Epic Games Store. Από το 2018, η εταιρεία έχει μια εκτιμώμενη αποτίμηση 15 δισεκατομμυρίων δολαρίων ΗΠΑ.

ΚΕΦΑΛΑΙΟ 8^{ο} :Γνωστές Μηχανές δημιουργίας Video Games και κρητίρια επιλογής

8.1 Unreal Engine 4 vs. Unity:

Unreal Engine 4 (UE4) και η Unity είναι αναμφισβήτητα δύο από τις πιο δημοφιλείς μηχανές παιχνιδιών στη διάθεση του κοινού σήμερα. Ενώ πολλά στούντιο ανάπτυξης παιχνιδιών χρησιμοποιούν τη δική τους ιδιόκτητη μηχανή παιχνιδιού υπάρχει ακόμα μια τεράστια αγορά για indie προγραμματιστές και ακόμη μεγαλύτερα στούντιο που χρειάζονται μια μεγάλη μηχανή παιχνιδιού για να τους βοηθήσει να δημιουργήσουν το παιχνίδι τους. Παιχνίδια όπως dead island 2 και Hitman Sniper αναπτύσσονται σε αυτές τις μηχανές παιχνιδιών. Έτσι, αν θέλετε να μπείτε στο σχεδιασμό παιχνιδιών το επόμενο βήμα είναι να επιλέξετε ποια μηχανή παιχνιδιού χρειάζετε να μάθετε, και ποια μηχανή παιχνιδιού χρειάζετε για να ξεκινήσει το παιχνίδι σας. Ενώ και οι δύο Unreal Engine 4 και η Unity είναι άριστες μηχανές παιχνιδιών, ανάλογα με το τι θέλετε να κάνετε, μπορεί κάποια να είναι μια καλύτερη επιλογή για σας, πάνω από την άλλη. Είτε είναι απλά το interface που προτιμάτε, ή ίσως η γλώσσα προγραμματισμού. Όποια και αν είναι η περίπτωση, Ας ερευνίσουμε κάθε μηχανή παιχνιδιών και να εξετάσουμε ισχυρές περιοχές τους, ώστε να μπορείτε να αποφασίσετε ποια θα λειτουργήσει καλύτερα για σας.

8.2 Τι είδους παιχνίδια πρόκειται να δημιουργήσετε;

Η πρώτη ερώτηση που πρέπει να ρωτήσετε τον εαυτό σας γιατί πρόκειται να χρησιμοποιήσετε τη μηχανή του παιχνιδιού , και τι είδους παιχνίδια θέλετε να δημιουργήσετε; Φυσικά, αν θέλετε να κάνετε ένα παιχνίδι που είναι, επειδή οι μηχανές παιχνιδιών μπορούν να χρησιμοποιηθούν για όχι μόνο την ανάπτυξη του παιχνιδιού. Θέλετε να δημιουργήσετε μια πλατφόρμα 2D ή ένα παιχνίδι περιπέτειας 3D δράσης; Ίσως ένα παιχνίδι με ένα υβρίδιο 2D και 3D. Μπορείτε να αρχίσετε ένα μικρό παιχνίδι και να δημιουργήσετε έναν απλό στοιβαχτή γρίφων με κάποια βασική φυσική συμμετοχή. Θέλετε να ξεκινήσετε το παιχνίδι σε ένα πρόγραμμα περιήγησης στο Web ή κινητές πλατφόρμες φιλοδοξούν να; Πώς σκοπεύετε να κερδίσετε χρήματα από το παιχνίδι; Μέσω αγορών εντός εφαρμογής, διαφημίσεων εντός του παιχνιδιού ή μόνο από την τιμή αγοράς μόνο; Απαντώντας σε αυτές τις ερωτήσεις μπορούν να σας βοηθήσουν να προσδιορίσετε αν θα πρέπει να χρησιμοποιήσετε Unreal Engine 4 ή Unity . Και οι δύο μηχανές μπορούν βεβαίως να πάρουν την εργασία γίνοντη, αλλά ανάλογα με αυτό που δημιουργείτε, μια μηχανή να βασιλεύσει ανώτατη στην εργασία σας. Αν θέλετε να κάνετε ένα 2D παιχνίδι, η Unity μπορεί να είναι μια μεγάλη επιλογή, διότι έχει κάποια μεγάλη 2D χαρακτηριστικά και μπορεί να είναι πολύ εύκολο να ξεκινήσετε. Τούτου λεχθέντος, Unreal Engine 4 διαθέτει και αυτή μερικά ισχυρά 2D χαρακτηριστικά. Αν θέλετε να δημιουργήσετε ένα παιχνίδι 3D, η Unity είναι επίσης ένα πολύ ισχυρό 3D μηχανή παιχνιδιού. Αν και τα γραφικά δεν είναι στο ίδιο επίπεδο με την unreal engine 4, αν δεν χρειάζεται να δημιουργήσετε γραφικά επόμενης γενιάς επίπεδο τότε Unreal Engine 4 μπορεί να μην απαιτείται. Πολλοί καλλιτέχνες χρησιμοποιούν τώρα τόσο την Unity όσο και την Unreal Engine 4 ως ένας τρόπος για να δημιουργήσουν προ-οπτικοποίηση και αρχιτεκτονική απεικόνιση για να δημιουργήσουν μια πιο διαδραστική εμπειρία για τους πελάτες

τους. Η Unreal Engine 4 πρόσφατα προσπάθησε να ωθήσει τη μηχανή τους μεταξύ των κινητών προγραμματιστών με ισχυρά 2D χαρακτηριστικά. Ίσως θέλετε να δημιουργήσετε ένα 3D παιχνίδι, καλά? Ενότητα είναι επίσης ένα πολύ ισχυρό 3D μηχανή παιχνιδιού, καθώς και. Ενώ είναι γραφικά δεν είναι στο ίδιο επίπεδο με Unreal Engine 4, αν δεν χρειάζεται να δημιουργήσετε Next-gen γραφικά επίπεδο, στη συνέχεια, έχοντας κάτι σαν Unreal Engine 4 μπορεί να μην απαιτείται. Έχετε επίσης την επιλογή να κάνετε την προ-οπτικοποίηση ή ακόμα και την αρχιτεκτονική απεικόνιση. Αυτό είναι κάτι περισσότερα στούντιο επεκτείνονται και την εξεύρεση τρόπων για να χρησιμοποιήσετε μια μηχανή παιχνιδιού για να δημιουργήσετε μια διαδραστική εμπειρία για τους πελάτες τους. Πολλοί καλλιτέχνες χρησιμοποιούν τόσο την ενότητα και Unreal Engine 4 ως ένας τρόπος για τη δημιουργία αυτών των διαδραστικών εμπειριών.

8.3 Τιμές

Ενώ η επιλογή ενός engine παιχνιδιού μπορεί να βασιστεί σε μεγάλο βαθμό για το τι είδους παιχνίδι θέλετε να δημιουργήσετε, και ποια πλατφόρμα θέλετε να ξεκινήσει σε υπάρχει επίσης ένα άλλο πολύ σημαντικό παράγοντα, ότι, ανάλογα με τη χρηματοδότησή σας μπορεί να είναι ο αποφασιστικός παράγοντας για το αν θα επιλέξει Unreal Engine 4 ή Unity. Αν είστε ένας επίδοξος προγραμματιστής παιχνιδιών που απλά θέλει να μάθει πώς να κάνει τα παιχνίδια, ίσως δεν θέλετε να δαπανήσουν πολλά χρήματα σε μια μηχανή παιχνιδιού. Η τιμολόγηση μιας μηχανής παιχνιδιών μπορεί να είναι ένα τεράστιο σημείο πώλησης για κάθε προγραμματιστή. Αν τελικά θέλετε να ξεκινήσετε το παιχνίδι σας υπάρχει επίσης δικαιώματα που πρέπει να ληφθούν υπόψη. Η Unity προσφέρει μια εντελώς δωρεάν έκδοση έτοιμη για download. Αναπλά θέλετε να ξεκινήσετε αυτό μπορεί να είναι μια μεγάλη επιλογή για σας. Αν και σίγουρα δεν έχει όλα τα χαρακτηριστικά που έχει το Unity Pro είναι ακόμα σε θέση να κάνει εξαιρετικά παιχνίδια χωρίς να χρειάζεται να δαπανήσει ποτέ μια δεκάρα. Αν θέλετε να αναβαθμίσετε σε Unity Pro, αυτό μπορεί να είναι λίγο ακριβό. Μπορείτε να αγοράσετε το πρόγραμμα για $1.500, ή να πληρώσετε $75/month, ενώ οι πληρωμή μπορεί να φαίνεται απότομη, ειδικά για κάποιον που απλά θέλουν να ξεκινήσουν. Αν θέλετε ποτέ να δημοσιεύσετε το παιχνίδι σας δεν χρειάζεται να πληρώσετε τα δικαιώματα για την Unity. Εάν είστε ένας μεγαλύτερος προγραμματιστής αυτό μπορεί να είναι ένα μεγάλο δέλεαρ για σας. Εντούτοις, πρέπει να πληρώσετε άλλα $1.500 ανά κινητή πλατφόρμα που δημοσιεύετε επάνω.Με την Unreal Engine 4 δεν υπάρχουν Pro ή δωρεάν εκδόσεις. Στην πρόσφατη ανακοίνωση Unreal Engine 4 είναι εντελώς δωρεάν. Τα πάντα είναι κάτω από μία στέγη, και θα εξακολουθούν να έχουν πρόσβαση σε ό, τι η μηχανή του παιχνιδιού έχει να προσφέρει, χωρίς κανένα κόστος για εσάς. Τώρα, η Unreal Engine 4 έχει αμοιβές δικαιωμάτων του 5%. Έτσι, κάθε χρηματικό ποσό που κάνετε από το παιχνίδι σας, η Epic Games παίρνει το 5% του ό, τι κερδίζετε, συμπεριλαμβανομένων των in-app αγορές, in-game διαφημίσεις και την τιμή του παιχνιδιού. Unreal Engine 4 είναι επίσης εντελώς δωρεάν για κολέγια και πανεπιστήμια. Αν είστε δάσκαλος τότε Unreal Engine 4 μπορεί να είναι η πιο λογική επιλογή. Όχι μόνο παίρνετε Unreal Engine 4 δωρεάν στην τάξη σας,αλλά θα σας επιτρέψει επίσης να δώσετε την Unreal Engine 4 στους μαθητές σας για τη δική τους προσωπική

χρήση στο σπίτι τους κατά τη διάρκεια του χρόνου τους στο σχολείο. Αν είστε φοιτητής, μπορεί να έχετε ήδη ελεύθερη πρόσβαση σε Unreal Engine 4, έτσι Ρωτήστε το δάσκαλό σας! Αν κάνετε ένα παιχνίδι που θέλετε να δημοσιεύσετε, ενώ στο σχολείο, μπορείτε ακόμα, φυσικά το 5% των δικαιωμάτων είναι ακόμα επισυνάπτεται. Και οι δύο μηχανές παιχνιδιών είναι εξαιρετικά προσιτές, η Unity σας δίνει πρόσβαση στην ελεύθερη έκδοση της Unity, η οποία εξακολουθεί να είναι μια ισχυρή μηχανή. Ωστόσο, αν είστε δάσκαλος η Unreal Engine 4 μπορεί απλά να βγει στην κορυφή από την άποψη της τιμολόγησης.

8.4 Γλώσσες προγραμματισμού

Ενώ η τιμολόγηση ενός game engine μπορεί να ταλαντεύει σίγουρα την απόφασή σας, τα χαρακτηριστικά του game engine είναι προφανώς ένας τεράστιος παράγοντας για να κάνετε μια επιλογή, και να αποφασίσετε ποια είναι καλύτερη για σας. Δεν είναι μυστικό ότι για να δημιουργήσετε ένα παιχνίδι θα πρέπει να κάνετε αρκετά ένα κομμάτι του προγραμματισμού. Ανάλογα με το ποια γλώσσα προγραμματισμού είστε εξοικειωμένοι μπορεί να σας βοηθήσει να αποφασίσετε αν θα επιλέξετε με Unreal Engine 4 ή την Unity. Η Unreal Engine 4 χρησιμοποιεί C++ και η Unity χρησιμοποιεί κυρίως C# ή JavaScript. Η απόφαση για το ποιο πρόγραμμα είναι καλύτερο από την άποψη των γλωσσών προγραμματισμού έρχεται πραγματικά κάτω στην προσωπική προτίμηση. Μερικοί άνθρωποι σκέφτονται C++ είναι αρχαϊκή, ενώ άλλοι ορκίζονται από αυτό. Εάν προτιμάτε οποιαδήποτε από αυτές τις γλώσσες πέρα από άλλη έπειτα η απόφαση μπορεί να είναι αρκετά εύκολη για σας.

8.5 Blueprint

Unreal Engine 4 έχει σχεδιάγραμμα οπτικού scripting. Μια μέθοδος δέσμης ενεργειών βασισμένη σε κόμβο, ακριβώς μέσα στην Unreal Engine 4. Τεχνικά δεν χρειάζεται ποτέ να γράψετε μια ενιαία γραμμή κώδικα. Αυτό είναι μεγάλο για γρήγορα πρωτότυπα επίπεδα, και μπορείτε να δημιουργήσετε ακόμη και ολόκληρα τα παιχνίδια χρησιμοποιώντας το σχεδιάγραμμα. Αν δεν είστε ένας οδηγός προγραμματισμού, αλλά απλά θέλετε να κάνουν ένα παιχνίδι, αυτό μπορεί να κάνει Unreal Engine 4 μια μεγάλη επιλογή για εσάς. Φυσικά, υπάρχουν περιορισμοί στις ικανότητες του σχεδιαγράμματος, αλλά εάν είστε νέοι στο σχέδιο παιχνιδιών μπορεί να είναι ένας μεγάλος τρόπος να απολεφθούν πολλά προβλήματα. Αν θέλετε να χρησιμοποιήσετε Unreal Engine 4 για τα πράγματα όπως απεικονίσεις και αρχιτεκτονικά Walk-throughs σχεδιάγραμμα θα ήταν εξαιρετική για αυτό, επειδή δεν θα έχετε να ανησυχείτε για τη συγγραφή κώδικα και μπορεί να γίνει πολύ γρήγορα.

8.6 Asset Store

Και οι δύο Unreal Engine 4 και η Utnity έχουν ένα κατάστημα περιουσιακών στοιχείων. Αφήνοντας να κατεβάσετε διάφορα περιουσιακά στοιχεία του παιχνιδιού, όπως χαρακτήρες, στηρίγματα και ακόμη και τα πράγματα όπως ήχους και εφέ σωματιδίων. Ωστόσο, η Unity βγαίνει πραγματικά στην κορυφή από την άποψη του μεγέθους του καταστήματος περιουσιακών στοιχείων ths. Προσφέροντας τα πάντα, από διαισθητική animation και ξάρτια εργαλεία για GUI γεννήτριες και λογισμικό καταγραφής κίνησης. Υπάρχουν όλα όσα χρειάζεστε για να δημιουργήσετε το παιχνίδι σας.

8.7 No Profiler in Unity Free

Ένα πράγμα που πρέπει να θυμάστε με την ελεύθερη έκδοση της Unity είναι ότι δεν υπάρχει Profiler. Αυτό περιλαμβάνεται μόνο στην Unity Pro. Αυτό από μόνο του μπορεί να είναι ένας τεράστιος αποτρεπτικός παράγοντας για ορισμένους προγραμματιστές και μπορεί να είναι ένας μεγάλος αποφασιστικό παράγοντα. Το Profiler βασικά σας επιτρέπει να βελτιστοποιήσετε το παιχνίδι σας, μπορείτε να παίξετε το παιχνίδι σας με το Profiler και θα καταγράφει την απόδοση του παιχνιδιού σας και να σας δείξει το ποσοστό του χρόνου που δαπανάται στο να κάνει εργασίες όπως η απόδοση, και animation παίζοντας το παιχνίδι σας.Αυτό δεν το έχουν διαθέσιμο στην ελεύθερη έκδοση καθιστά εξαιρετικά δύσκολη την κατανόηση στι περιοχές όπου στο παιχνίδι προκαλούντε επιβραδύνσεις, αλλά και τι πρέπει να καθοριστεί, προκειμένου να έχουν τη βέλτιστη απόδοση.

8.8 Graphical Capabilities

Όταν πρόκειται για γραφικά ,η Unreal Engine 4 είναι πραγματικά ένα Next-Gen game engine. Ικανή να δημιουργεί γραφικά στο ίδιο επίπεδο με τα παιχνίδια που βλέπετε να απελευθερώνονται στις κονσόλες παιχνιδιών Next-Gen. Από σύνθετα συστήματα προσομοίωσης σωματιδίων έως προηγμένο δυναμικό φωτισμό. Φυσικά, με την επερχόμενη Unity 5, έχει υπάρξει επίσης μια μεγάλη γραφική αύξηση. Αν και, η ημερομηνία κυκλοφορίας δεν έχει ακόμη ανακοινωθεί. Τα γραφικά φυσικά δεν είναι τα πάντα, και πολύ απλά ίσως δεν χρειάζεται να δημιουργήσετε ένα παιχνίδι που ανταγωνίζεται με σημερινούς τίτλους παιχνιδιών. Τούτου λεχθέντος, η Unreal Engine 4 έχει τις δυνατότητες να δημιουργήσει πραγματικά οποιοδήποτε

τύπο οπτικού στυλ που θέλετε 2D ή 3D. Είναι πάντα καλό να είναι σε θέση να έχουν τη δυνατότητα να ωθήσει τα γραφικά περαιτέρω, αν επιλέξετε να το πράξουν.

8.9 Ease of Use

Όσον αφορά το ποιο πρόγραμμα είναι πιο εύκολο στη χρήση, έρχεται πραγματικά κάτω στην προσωπική προτίμηση.η Unity γενικά θεωρείται ως το πιο διαισθητικό και πιο εύκολο να κατανοήσουν τη μηχανή του παιχνιδιού, ωστόσο,η Unreal Engine 4 με την πλήρης αναθεώρηση UI έχει φέρει μαζί της ένα πολύ εύκολο στην κατανόηση UI που δεν θα πρέπει να πάρει πολύ για να σηκωθεί και να λειτουργήσει. Εάν βασίζετε την απόφασή σας σχετικά με το ποια μηχανή παιχνιδιού είναι πιο εύκολο να κατανοήσετε τότε η καλύτερη επιλογή είναι να δοκιμάσετε και τα δύο. Βάλτε τους μέσω του κουδουνίσματος και να προσπαθήσουμε να καταλάβετε τι λειτουργεί καλύτερα για σας. Ο καθένας είναι διαφορετικός, και ενώ μπορεί να έχετε συστήσει να χρησιμοποιήσετε την Unreal Engine 4, για σας, η Unity μπορεί να είναι η σωστή επιλογή ή και το αντίστροφο. Και οι δύο αυτές μηχανές παιχνιδιών έχουν ισχυρά κομμάτια του λογισμικού ικανό να βοηθήσει στην ώθηση των ιδεών σας στο επόμενο επίπεδο.

ΚΕΦΑΛΑΙΟ 9^{ο} :Unity vs Unreal Engine: ποια μηχανή παιχνιδιών είναι για εσάς;

Η Unity με την Unreal Engine έχουν μια επανειλημμένη συζητούμενη διαμάχη μεταξύ των ψηφιακών καλλιτεχνών και προγραμματιστών παιχνιδιών για χρόνια τώρα. Αναμφισβήτητα δύο από τις πιο δημοφιλείς μηχανές παιχνιδιών που διατίθενται σήμερα, η Unity και η Unreal Engine χρησιμοποιούνται από μεγάλα στούντιο και indie προγραμματιστές . Αλλά ποιο από τα δύο ταιριάζει καλύτερα στις ανάγκες σας;

Εδώ, James Burrows, τεχνικός διευθυντής σε immersive στούντιο, αποκαλύπτει τα τέσσερα βασικά πράγματα που πρέπει να εξετάσετε κατά την επιλογή μεταξύ αυτών των δύο κορυφαίων κινητήρων παιχνίδι.

9.1 Ποιο επίπεδο οπτικών επιθυμείτε

Μία από τις κύριες διαφοροποιήσεις κατά την εξέταση της Unity εναντίοω της Unreal Engine είναι η ποιότητα των οπτικών. Η Unreal Engine προσφέρει υψηλής πιστότητας visuals κατ ' ευθείαν από το κουτί, ενώ η Unity εξακολουθεί να είναι σε θέση να παράγει υψηλής ποιότητας

γραφικά αλλά παίρνει πολύ περισσότερη δουλειά για να πάρει τα περιουσιακά στοιχεία σας κοιτάζοντας κοντά στο ίδιο επίπεδο με εξωπραγματικό. Και ακόμα και τότε, δεν θα παράγει αρκετά την ίδια ποιότητα.

Είναι για το λόγο αυτό θα βρείτε Unreal χρησιμοποιείται περισσότερο για μεγάλα παιχνίδια και παραγωγές από τα μεγάλα στούντιο. Έτσι, αν θέλετε όσο πιο κοντά στην φωτορεαλιστική περιουσιακά στοιχεία του δυνατού, είναι πιο γρήγορα και πιο εύκολο να επιτευχθεί αυτό με εξωπραγματικό.

9.2 Σε ποια συσκευή προορίζεται το έργο σας;

Εάν κοιτάζετε για να δημιουργήσετε ένα πρόγραμμα για να τρέξετε στις χαμηλότερες-τροφοδοτημένες συσκευές, όπως τα κινητά τηλέφωνα, κατόπιν η υψηλή δύναμη επεξεργασίας απαιτήτε από την Unreal Engine δεν είναι η κατάλληλη για σας. Αλλά εκεί είναι όπου η Unity έρχεται πραγματικά στο δικό της τομέα. Αρχικά σχεδιασμένο για να τρέξει στις συσκευές όπως τις κονσόλες και τα τηλέφωνα, η Unity επιτρέπει σε σας να δημιουργήσετε τα σύνθετα προγράμματα για τις low-end συσκευές χωρίς απαίτηση μιας τέτοιας ισχυρής οργάνωσης PC . Εάν, από την άλλη πλευρά, θέλετε να δημιουργείσετε μια εμπειρία για high-end συσκευές, τότε είτε η Unity είτε Unreal Engine είναι καλές επιλογές. Αλλά αυτό εξαρτάται επίσης από άλλα κριτήρια εκτός από τις αποδόσεις της κάθε μηχανής παιχνιδιών .

9.3 το μέγεθος της ομάδας

Η συναίνεση μεταξύ της συλλογικής εμπειρίας της καθηλωτικής ομάδας Dev είναι ότι για να πάρει το πολύ καλύτερο από εξωπραγματικό, χρειάζεστε μια μεγάλη και εξειδικευμένη ομάδα που είναι αφιερωμένη σε διάφορα μέρη της διαδικασίας, για παράδειγμα, κάποιος είναι αφιερωμένος μόνο σε σωματίδια ή κάποιος μόνο για τους shader.

Η Unity, αφ ' ενός, είναι πολύ ευκολότερη για τους υπεύθυνους για την ανάπτυξη για να πάρει στα πιασίματα με αμέσως που κάνει την μια καλή επιλογή για τις ζώνες ένας-ατόμων και τις μικρότερες ομάδες για να δημιουργήσει μια αποτελεσματική εμπειρία. Το κατάστημα περιουσιακών στοιχείων είναι επίσης σημαντικά μεγαλύτερο, καθιστώντας απλούστερο να συμπληρώσετε το παιχνίδι ή την εμπειρία σας, αν δεν έχετε μια τεράστια ομάδα.

9.4 Είστε προγραμματιστής ή εικαστικός καλλιτέχνης;

Δεν υπάρχει καμία αμφιβολία γι ' αυτό, αυτό φαίνεται να επηρεάζει την προτίμηση. Οι προγραμματιστές μας προτιμούν την Unity, αλλά εικαστικοί καλλιτέχνες μας επιλέγουν την Unreal Engine και αυτό είναι καθαρά κάτω από τη διαφορά στα οπτικά εφέ. Και οι δύο μηχανές παιχνιδιών προσφέρουν το ίδιο είδος της λειτουργικότητας και της ικανότητας, όμως συσκευάζονται με διαφορετικούς τρόπους.

Εδώ και λίγο καιρό, οι γραμμές που διαφοροποιούν τις δύο έχουν αρχίσει να θολώνουν καθώς η Unreal Engine ξεκίνησε ως μηχανή παιχνιδιών AAA και έχει ως στόχο να γίνει πιο προσιτή για μικρότερες ομάδες και εμπειρίες, ενώ η Unity αρχικά προτιμάται από indie studios για απλά παιχνίδια και εμπειρίες αλλά συνεχίζει να εργάζεται το δρόμο της κορυφής με την προσθήκη pro-level χαρακτηριστικά.

Η κύρια διαφορά είναι η οπτική ποιότητα και η πλατφόρμα στόχος σας αλλά πιστεύουμε ότι δεν θα είναι μεγάλη μέχρι και οι δύο μηχανές παιχνιδιών να φτάσουν σε παρόμοιο επίπεδο και από τις δύο απόψεις. Σε αυτή την περίπτωση, σύντομα θα είναι απλά μια περίπτωση προσωπικής προτίμησης

ΚΕΦΑΛΑΙΟ 10 ο CryEngine

Η τελευταία, αλλά σίγουρα όχι η λιγότερο σημαντική που έχουμε στη λίστα μας για τις καλύτερες μηχανές παιχνιδιών στο 2018 είναι η CryEngine. Για πρώτη φορά από τη μεγάλη εταιρεία ανάπτυξης, Crytek, στο πρώτο παιχνίδι Far Cry, η CryEngine είναι σίγουρα μία από τις πιο ισχυρές και κυρίαρχες μηχανές παιχνιδιών που έχουμε σήμερα. Αυτό που κάνει την CryEngine αντάξια της λίστας είναι οι γραφικές ικανότητές της όπου επισκιάζουν τις γραφικές δυνατότητες της Unity και είναι ισοδύναμες με αυτές που η Unreal Engine κατέχει. Αν και είναι μια βαριά και ισχυρή μηχανή παιχνιδιών, η CryEngine παίρνει λίγο χρόνο για να είναι σε θέση να χρησιμοποιήσει αυτή την πλατφόρμα αποτελεσματικά και λίγο πιο δύσκολη για να κατανοηθεί από τους αρχάριους που δεν έχουν χρησιμοποιήσει άλλες μηχανές παιχνιδών εκ των προτέρων.

Η CryEngine υποστηρίζει την εικονική πραγματικότητα και έχει τα εκπληκτικά οπτικά αποτελέσματα συμπεριλαμβανομένης της ογκομετρικής ομίχλης και του συστήματος απόδοσης σύννεφων του, το οποίο δίνει στα σύννεφα μια πλήρη τρισδιάστατη χωρική απόδοση και μια ρεαλιστική απεικόνιση για την ομίχλη και τα καιρικά αποτελέσματα. Επιπλέον, το καλύτερο μέρος της επιλογής της πλατφόρμας CryEngine για το σχεδιασμό παιχνιδιού είναι ότι δεν απαιτεί από τους χρήστες του να καταβάλουν το τέλος δικαιωμάτων κατά το σχεδιασμό του παιχνιδιού. Εντούτοις, πρέπει να πληρώσετε ένα σταθερό ποσό δηλαδή $9,90/-το μήνα προκειμένου να έχετε πρόσβαση στην CryEngine. Εκτός αυτού, η CryEngine έχει μια αφιέρωση Q * ένα φόρουμ

αποκαλούμενο CryEngine απαντήσεις, το οποίο καθαρίζει όλες τις αμφιβολίες και τις απορίες σας, και σας βοηθά να έχετε μία καλύτερη εμπειρία.

Έτσι, τώρα που έχουμε τη σύγκριση μηχανών παιχνιδιών, το καλύτερο 3D παιχνίδι κινητήρα 2018;, ας σας βοηθήσει να συνοψίσουμε τη σύγκριση για την καλήτερη μηχανή 3D παιχνιδιών.

Συγκρίνοντας Unreal Engine ,Unity ,CryEngine 2018, συναντήσαμε τα καλύτερα χαρακτηριστικά που μας παρέχουν αυτές οι μηχανές παιχνιδιών. Ενώ οι επιδόσεις των Unreal Engine και Unity ταιριάζουν , έχουμε κατανοήσει ότι η Unity είναι η καλύτερη πλατφόρμα για την ανάπτυξη κινητών και 2D /3D παιχνιδιών, ενώ Unreal Engine είναι καταλληλότερη για την ανάπτυξη ιδιαίτερων γραφικών και φωτορεαλιστικών παιχνιδιών. Αυτό αποδεικνύει ότι είναι μια μεγάλη διαφορά μεταξύ Unreal Engine και Unity.

Η CryEngine, από την άλλη πλευρά, κατέχει τις δυνατότητες δημιουργίας υψηλών γραφικών παιχνίδια επίσης. Εκτός αυτού, συγκρίνοντας την CryEngine με την Unreal Engine 2018 για την κλίμακα της παροχής της επόμενης γενιάς χαρακτηριστικών πλατφόρμας με πιο ελκυστικό μοντέλο τιμολόγησης, η CryEngine κερδίζει σίγουρα την Unreal Engine με την οικονομική δομή της. Ωστόσο, για έναν αρχάριο, όταν πρόκειται για CryEngine 5 ενάντιας Unreal Engine 4 με βάση την ευκολία και τον μινιμαλισμό, παρά το γεγονός ότι τα χαρακτηριστικά που έχει η CryEngine είναι ετός συναγονισμού,η Unreal Engine 4 και η Unity σίγουρα αξίζουν μια δοκιμή.

ΚΕΦΑΛΑΙΟ 11^{ο} GAME DESIGN

Η σχεδίαση παιχνιδιού είναι ίσως μία από τις πιο περίπλοκες διαδικασίες καθώς απαιτεί σωστή προετοιμασία και εργασία σε πολλά επίπεδα. Από τη σχεδίαση του περιβάλλοντος, στην κωδικοποίηση των χαρακτήρων και τη συμπεριφορά των ενεργών στοιχείων στο χάρτη, μέχρι την σχεδίαση των οπτικών εφέ και την εμφάνιση των χαρακτήρων που ολοκληρώνουν ένα παιχνίδι, κάνοντας τον παίκτη να βυθίζεται σε αυτό. Σε αυτό το κεφάλαιο θα αναλύσουμε τα διάφορα κομμάτια που αποτελούν το παιχνίδι και τη ροή που ακολουθήσαμε για να πετύχουμε το τελικό αποτέλεσμα.

Το παιχνίδι που δημιουργήσαμε αποτελεί ένα απλό action/survival hack-and-slash παιχνίδι, μια από τις πιο «ελαφριές» κατηγορίες, καθώς δεν απαιτεί κάτι ιδιαίτερο για να επιτευχθεί ο σκοπός παρά μόνο αφοσίωση στο στόχο. Σκοπός του παιχνιδιού δεν είναι η επίλυση κάποιου γρίφου ή η παρουσίαση κάποιας ιστορίας αλλά προκαλεί τον παίκτη να καταφέρει να επιζήσει της επίθεσης.

Η δημιουργική διαδικασία ξεκίνησε με τη δημιουργία του κεντρικού χαρακτήρα, αυτόν δηλαδή που ελέγχει ο παίκτης και παράλληλα, μόλις τελείωσε το σχεδιάγραμμα των βασικών κινήσεων και δυνατοτήτων του χαρακτήρα, ξεκίνησε η σχεδίαση και ο προγραμματισμός των συμπεριφορών των κακών χαρακτήρων που θα αντιμετωπίσει ο ήρωάς μας. Αυτό συμβαίνει συχνά, καθώς αυτά τα δύο

στοιχεία είναι απόλυτα συνδεδεμένα, με τους βοηθητικούς χαρακτήρες να περιστρέφονται γύρω από τις πράξεις και τις κινήσεις του χαρακτήρα-παίκτη μας. Έτσι, προχωρήσαμε στην δημιουργία του δέντρου συμπεριφοράς που οδηγεί του «κακούς» μας και έπειτα επικεντρωθήκαμε στην δημιουργία του χάρτη και των στοιχείων που περιβάλλουν την πίστα μας.

Αρκετοί παίκτες βιντεοπαιχνιδιών προτιμούν την εστίαση των δημιουργών στην καλύτερη σχεδίαση χαρακτήρων-παικτών και της λειτουργικότητας τους, π.χ. τι ιδιότητες έχει ο κάθε παίκτης, τι δυνάμεις, η εμφάνισή του. Παρόλα αυτά, η σχεδίαση του χάρτη αποτελεί αναπόσπαστο κομμάτι και συμβάλει σημαντικά στην απόλαυση του παιχνιδιού, τόσο με την οπτική παροχή πληροφοριών στον παίκτη, όσο και στο ότι δίνει στον παίκτη το συναίσθημα ότι εξερευνάει έναν καινούριο κόσμο. Σε αυτό βοηθάει η χρήση διάφορων εφέ και η σύσταση ορίων που δίνουν μια ώθηση στον παίκτη να εξερευνήσει προς μια συγκεκριμένη κατεύθυνση. Παρόλο που δεν υπάρχει κάποια συγκεκριμένη ιστορία στο παιχνίδι μας παρά μόνο ο στόχος να επιβιώσουμε την επίθεση των τεράτων, η διαδρομή και μετακίνηση στο χώρο όσο παίζουμε μας δίνει την αίσθηση του επιτεύγματος όσο προχωράει το παιχνίδι.

Τελευταίο αλλά όχι λιγότερο σημαντικό είναι το μενού και οι τίτλοι αρχής του παιχνιδιού. Τα στοιχεία αυτά, αν και δεν είναι άκρως απαραίτητα για ένα παιχνίδι, προσθέτουν στην αίσθηση του παιχνιδιού και την gaming εμπειρία για τον παίκτη. Οι τίτλοι αρχής είναι μια καλή ευκαιρία να παρουσιαστούν οι παράγοντες του παιχνιδιού και όσοι έχουν συνεισφέρει στη δημιουργία του. Αρχικά, τα πιο μεγάλα στούντιο που παράγουν μεγάλους τίτλους παιχνιδιών(AAA gaming) σχεδόν κάθε χρόνο, μπορούν να χρησιμοποιήσουν τους τίτλους αρχής για να παρουσιάσουν και να αναγνωρίσουν τις παραγωγικές ομάδες που συνέβαλαν στην δημιουργία του παιχνιδιού αλλά και τους σπόνσορες που χρημάτισαν το έργο.

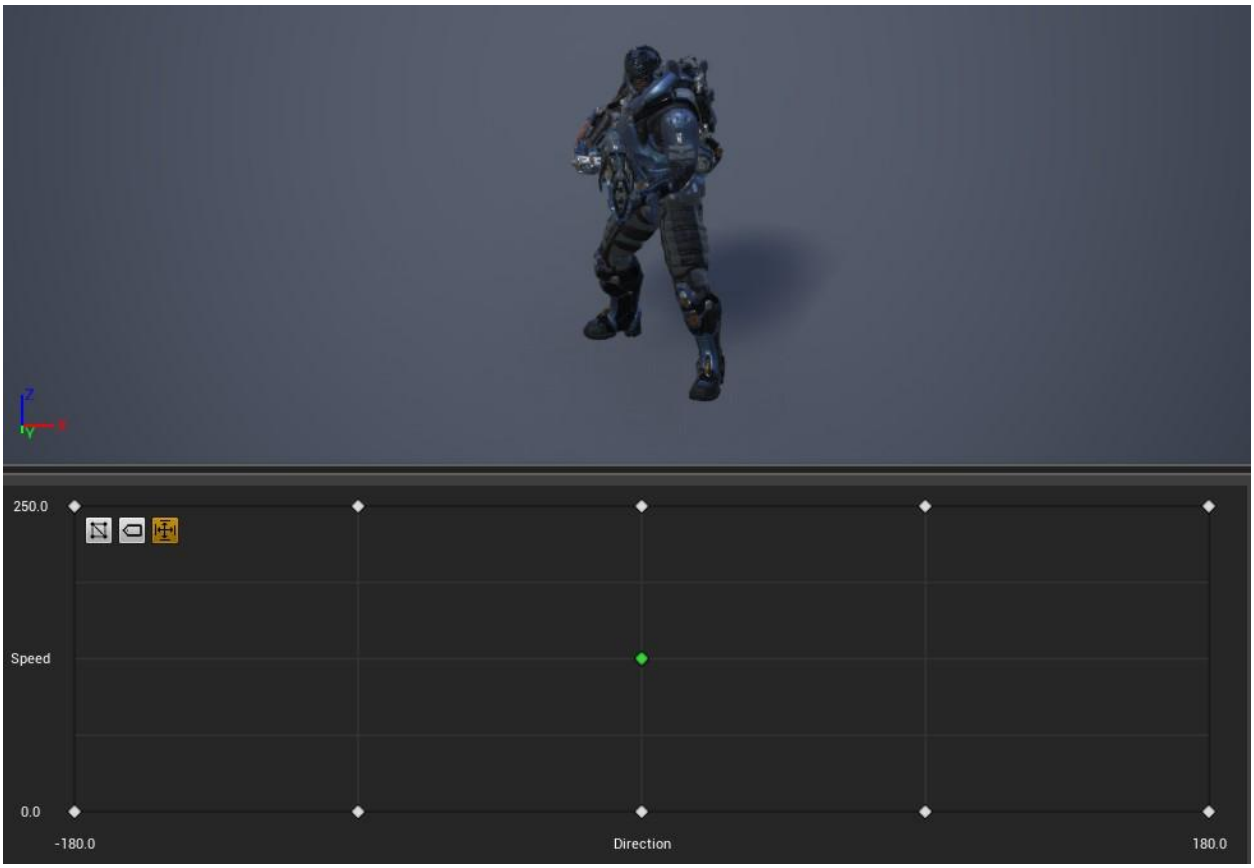
Τέλος, το μενού είναι ένα καθαρά χρηστικό σημείο που δίνει τη δυνατότητα στον παίκτη να καθορίσει συγκεκριμένα πράγματα πριν την έναρξη του παιχνιδιού(ρυθμίσεις γραφικών, χειρισμού, ήχου κ.λπ.) και το βασικότερο, να ξεκινήσει/τερματίσει το παιχνίδι. Ανάλογα το είδος του παιχνιδιού, θα δούμε διάφορα μενού, σχεδιασμένα έτσι ώστε να δίνουν μια μικρή ιδέα στον χρήστη για το τι θα ακολουθήσει.

11.1 Χαρακτήρας παιχνιδιού

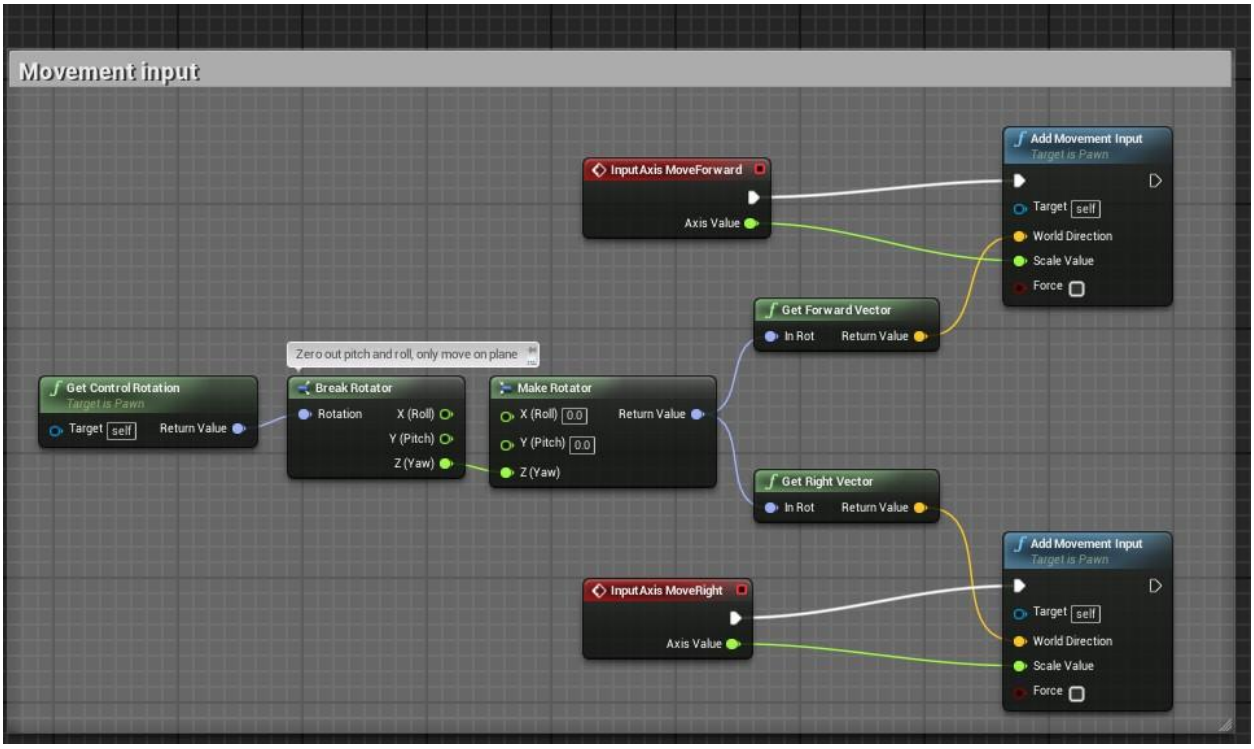
Ο χαρακτήρας μας είναι ίσως το πιο σημαντικό στοιχείο στο παιχνίδι μας, καθώς μας δίνει τη δυνατότητα να κινηθούμε ελεύθερα στον χώρο της πίστας και να εκτελούμε διάφορες ενέργειες, τις οποίες μπορούμε να καθορίσουμε στα σχέδια(blueprints). Στην περίπτωσή μας, μπορούμε να πυροβολήσουμε τους εχθρούς που παρουσιάζονται και να εξερευνήσουμε ελεύθερα το περιβάλλον του παιχνιδιού.

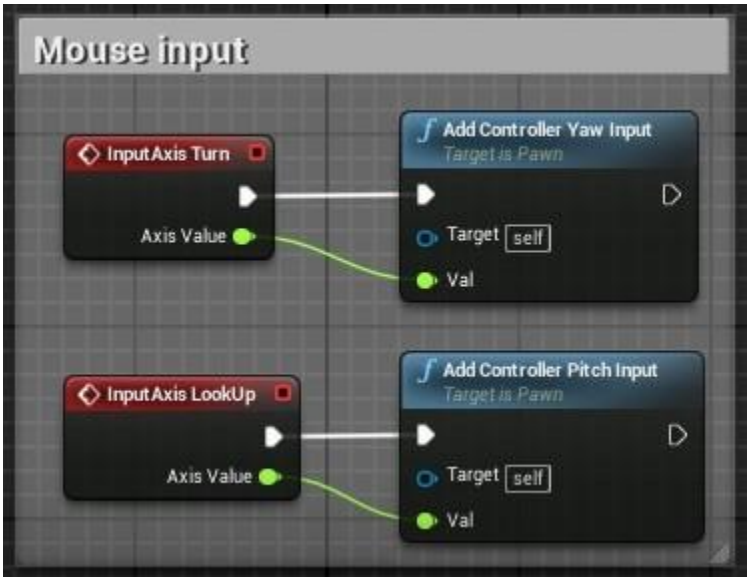
Τα υλικά που χρησιμοποιήθηκαν για τον χαρακτήρα αποκτήθηκαν δωρεάν μέσω του marketplace της Epic Games. Η εξωτερική εμφάνιση του χαρακτήρα έμεινε ίδια με τα αρχικά

assets, έγιναν όμως κάποιες αλλαγές στα blueprints ώστε να προσαρμοστούν οι κινητικές λειτουργίες και ο μηχανισμός για την επίθεση του χαρακτήρα μας, καθώς δεν θα χρησιμοποιήσουμε το σύστημα combo που προϋπήρχε.



Αρχικά, οι κινήσεις του χαρακτήρα μας ορίζονται στο blueprint, σε συγκεκριμένα γεγονότα (events) που χρησιμοποιούν τα πλήκτρα που έχουμε ορίσει στις γενικές ρυθμίσεις του project και συγκεκριμένα στις ρυθμίσεις Input. Με τα events αυτά καθορίζουμε την κίνηση στους άξονες X και Y χρησιμοποιώντας τα πλήκτρα WASD και Space του πληκτρολογίου, ενώ δημιουργήσαμε και events για την κίνηση της κάμερας του χαρακτήρα με την χρήση του ποντικιού (pitch και yaw)

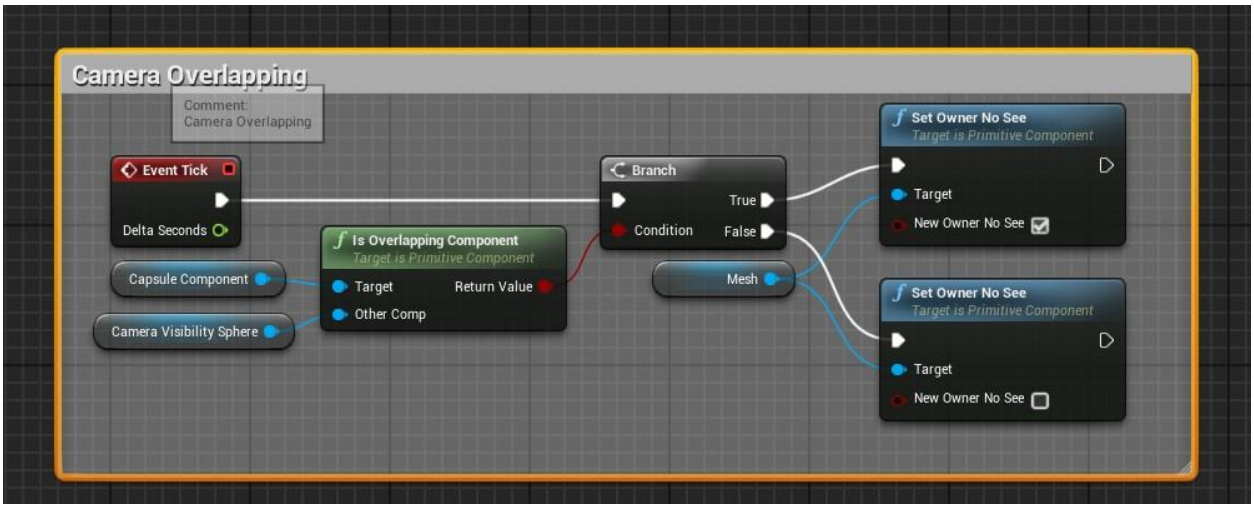




Για να προσαρμόσουμε καλύτερα την οπτική του παίκτη καθώς χρησιμοποιεί τον χαρακτήρα στο παιχνίδι, προσθέσαμε μία σφαίρα σύγκρουσης στην κάμερα ώστε όταν ο χαρακτήρας

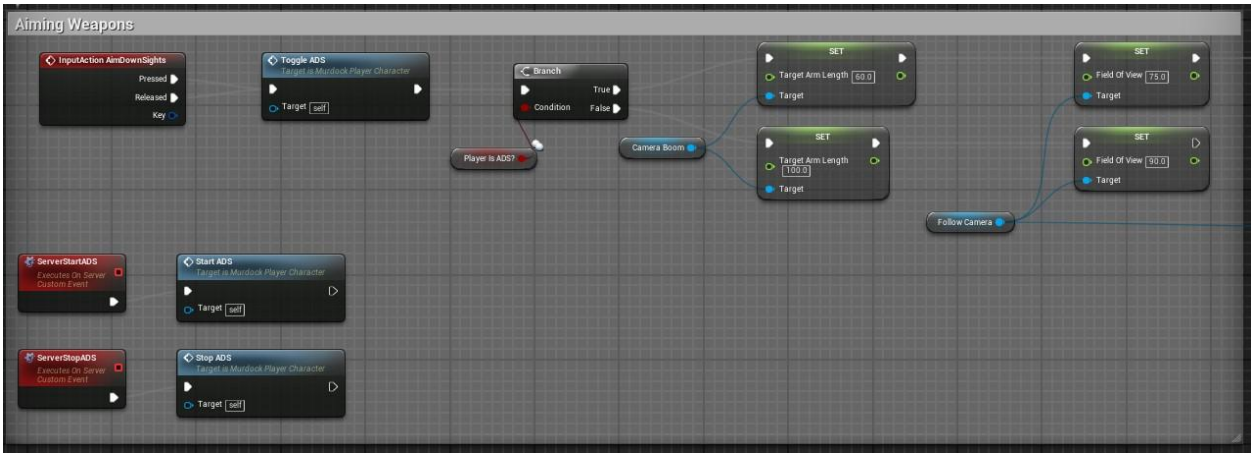
βρίσκεται δίπλα σε έναν τοίχο ή κάποιο εμπόδιο που κόβει την οπτική γωνία, να εξαφανίζει τον χαρακτήρα από την οθόνη, όπως φαίνεται στην παρακάτω εικόνα:

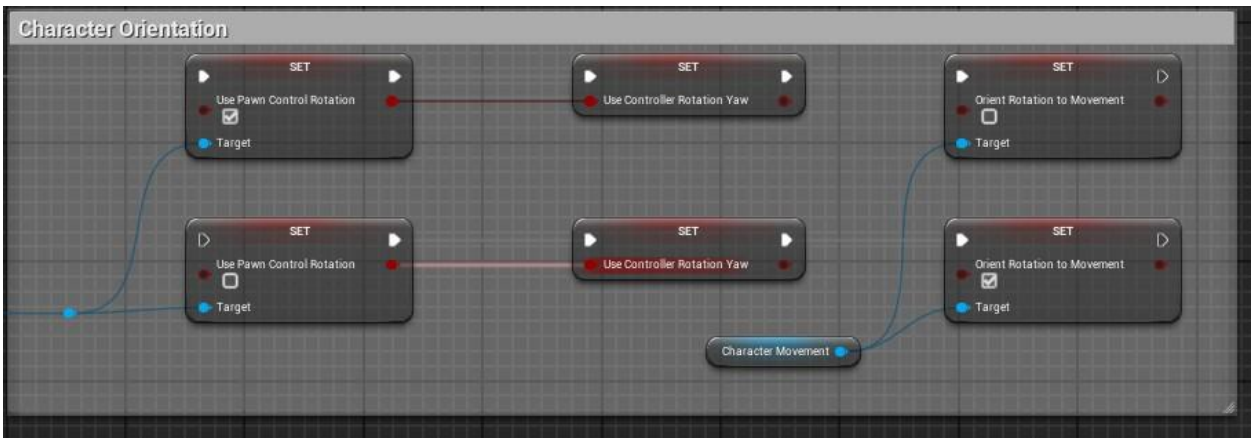
(+εικόνα από gameplay)



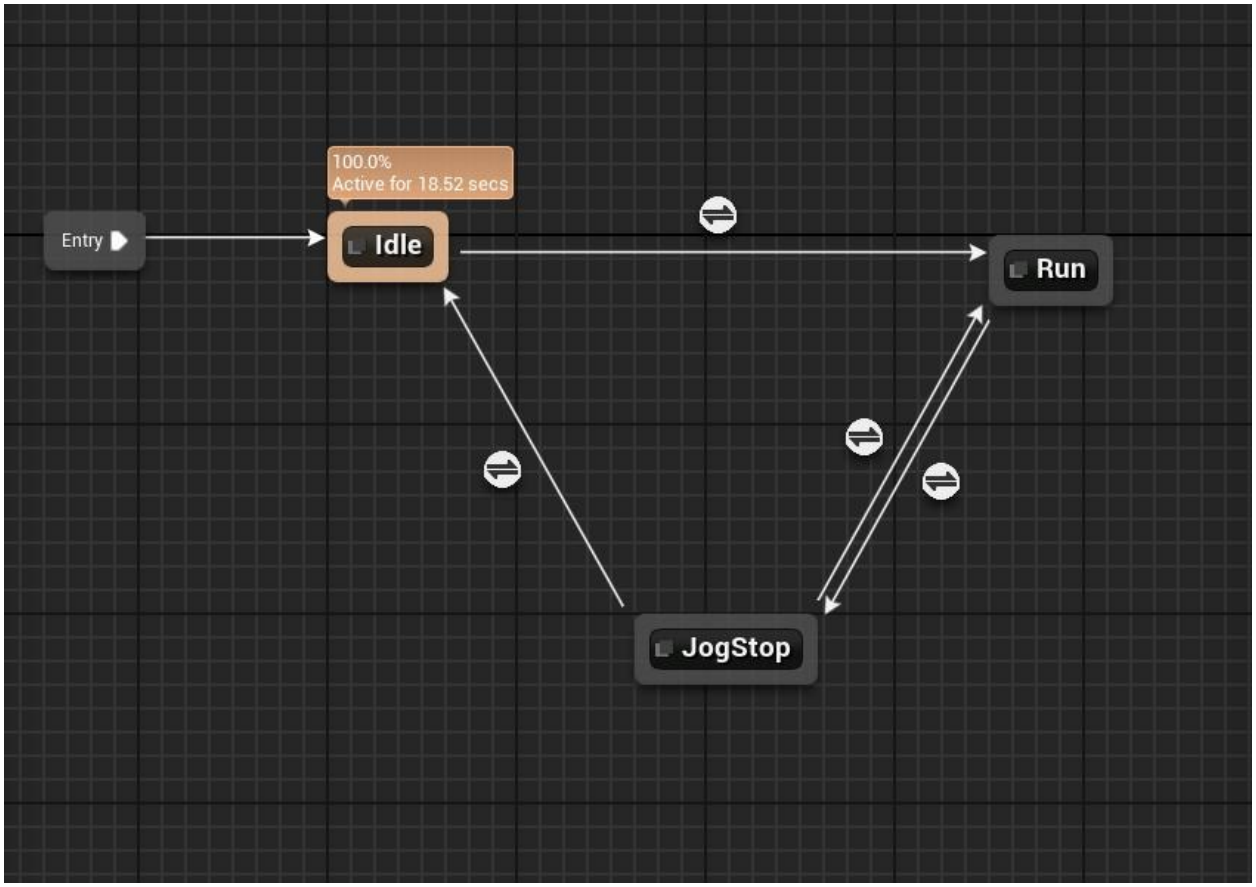
Το Event Tick εκτελείται συνεχόμενα και ελέγχει για εμπόδια μεταξύ κάμερας και χαρακτήρα ώστε να εμποδίσει την κάμερα από το να επικαλύψει το εμπόδιο και να χάσουμε οπτική επαφή με τον χαρακτήρα μας. Όταν εντοπίσει ότι κάποιο εμπόδιο βρίσκεται ανάμεσα σε κάμερα και κάψουλα σύγκρουσης (capsule component) απενεργοποιεί τον σκελετό του χαρακτήρα μας (mesh) έτσι ώστε να έχουμε πλήρη εικόνα του περιβάλλοντος.

Πέρα από τις βασικές κινήσεις που χρησιμοποιούμε για να μετακινηθούμε στο χώρο, ενσωματώσαμε ένα σύστημα στόχευσης με ζουμ, το οποίο βοηθάει τον παίκτη να εστιάσει σε συγκεκριμένο χώρο στο περιβάλλον και να λάβει περισσότερες λεπτομέρειες για τα σκηνικά και τα περιεχόμενά του(π.χ. χώρους για να εξερευνήσει, μη ελεγχόμενους χαρακτήρες-AI και χώρους για να κρυφτεί από αυτούς). Αυτό έγινε προσθέτοντας στις ρυθμίσεις input το δεξί κλικ από το ποντίκι ώστε όσο ο παίκτης κρατάει πατημένο το κουμπί, η κάμερα να παραμένει σε ζουμ. Προσαρμόζοντας το blueprint, προσθέσαμε το event AimDownSights και δημιουργήσαμε ένα σύστημα με λειτουργίες όπου το event tick καταγράφει το πάτημα και ενημερώνει για την κατάσταση του παίκτη. Ακολούθως, προστέθηκε και λειτουργία ώστε όταν ο παίκτης «ζουμάρει» με τον στόχο προς μία κατεύθυνση, να ακολουθεί και ο σκελετός του χαρακτήρα για να μπορεί να επιτεθεί ο παίκτης προς αυτή την κατεύθυνση.



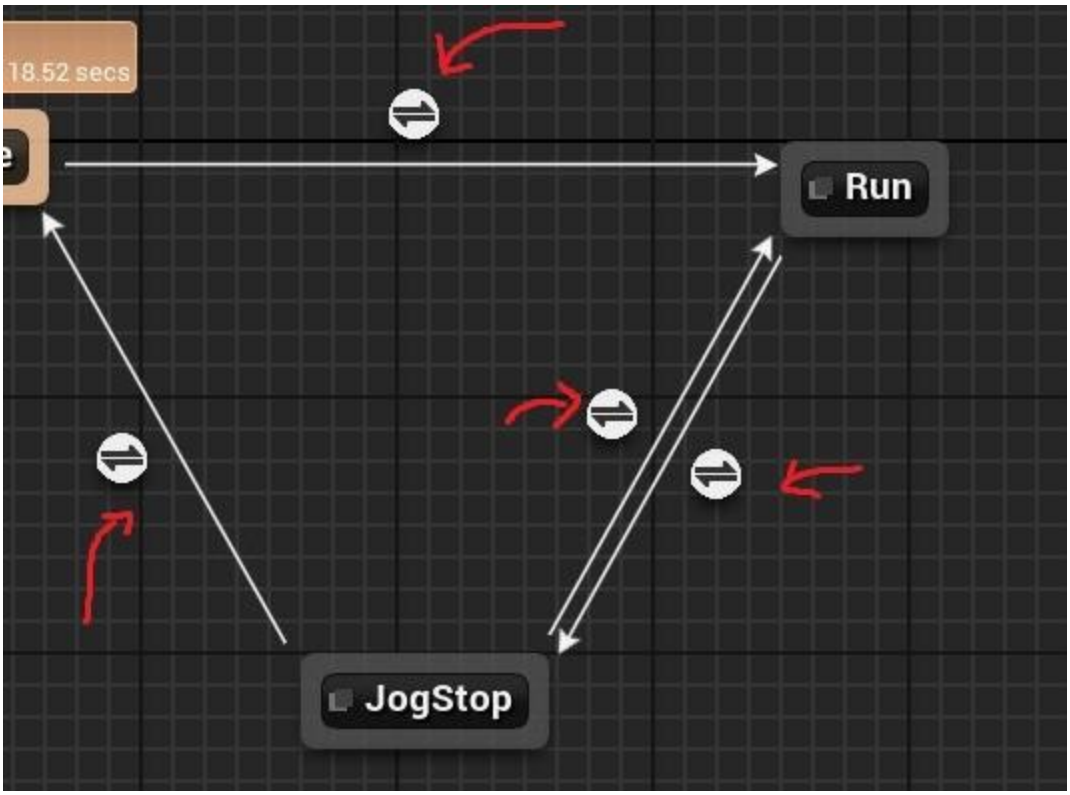


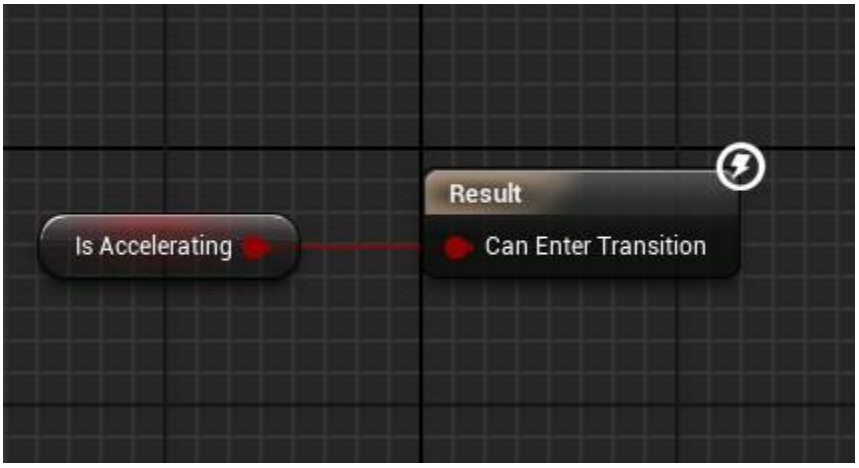
Για το animation του παίκτη χρησιμοποιήσαμε το animation blueprint που παρέχεται με τον χαρακτήρα με μερικές μικρές αλλαγές. Στο γράφημα ground locomotion καθορίζουμε το πώς αλλάζει καταστάσεις ο χαρακτήρας μας, από αδράνεια σε τρέξιμο όταν χρησιμοποιούμε τα πλήκτρα κίνησης και ξανά σε αδράνεια αν σταματήσουμε να



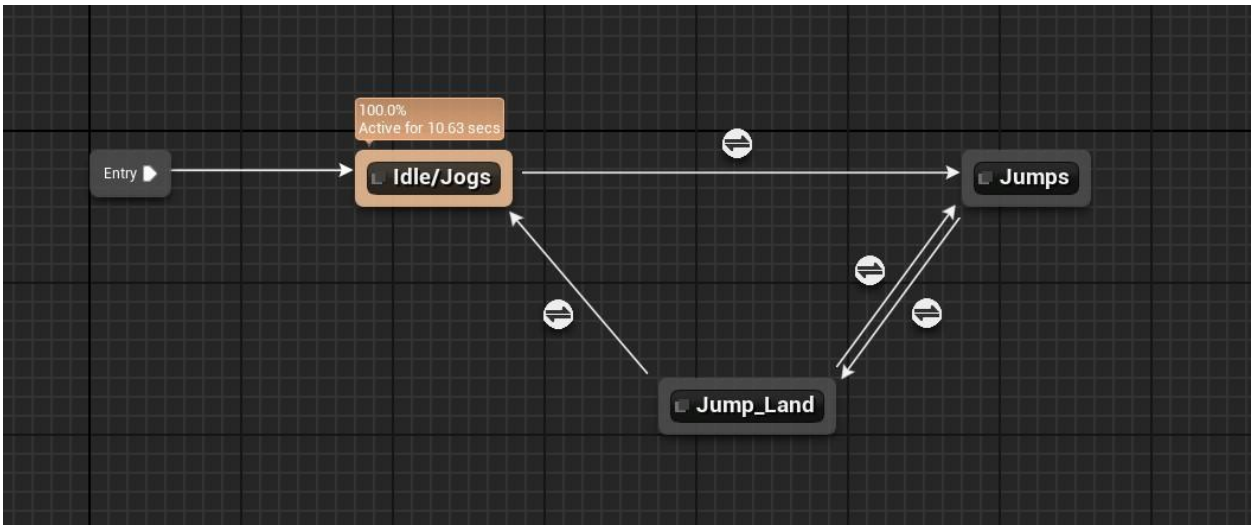
προχωράμε, όπως φαίνεται στο παρακάτω σχήμα:

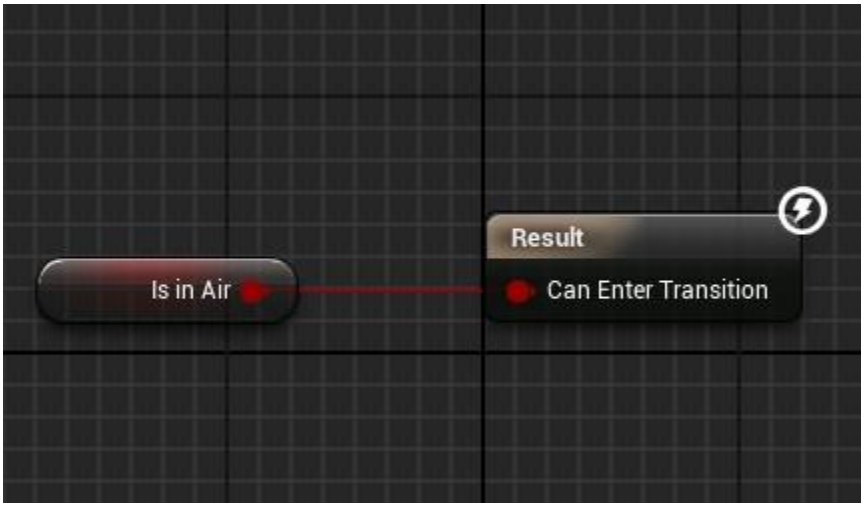
Έτσι όταν χρησιμοποιούμε τα πλήκτρα WASD για να κινήσουμε τον παίκτη μας, ενεργοποιείται το blueprint. Για την αλλαγή καταστάσεων, η μηχανή χρησιμοποιεί ένα σύστημα όπου καθορίσαμε τις εφικτές καταστάσεις που μπορεί να πάρει ο παίκτης και τους όρους που χρειάζεται να ελέγχει κάθε φορά για να μεταβεί από τη μία κατάσταση στην άλλη, π.χ. στην περίπτωση της αλλαγής από idle σε jog είναι εάν ο χαρακτήρας μας επιταχύνει:





Παράλληλα στο γράφημα locomotion, καθορίζεται η κίνηση άλματος του χαρακτήρα, η οποία σε αντίθεση με το γράφημα ground locomotion κινεί το κάτω μέρος του σώματος χωρίς να επηρεάζει τις κινήσεις του κορμού πάνω από τη μέση, πέρα από τη βαρύτητα που χρησιμοποιεί η μηχανή.





Έτσι, πάλι με έλεγχο όρου μετάβασης καθορίζουμε πότε μπορεί να μεταβεί ο παίκτης σε κατάσταση άλματος και πότε μπορεί να επανέλθει στην κανονική του μορφή μόλις προσγειωθεί. Επίσης, καθώς στο animation blueprint έχουμε καθορίσει ξεχωριστή κίνηση για τον άνω και κάτω κορμό του χαρακτήρα μας, μπορεί να εκτελεί και άλλες ενέργειες όσο βρίσκεται στον αέρα.

11.2 Χάρτης

Για την κατασκευή του χάρτη όπου εξελίσσεται το παιχνίδι, χρησιμοποιήσαμε διαφορετικά στοιχεία από πολλές δωρεάν βιβλιοθήκες, κυρίως τα δωρεάν μοντέλα από το παιχνίδι Paragon που διέθεσε η Epic Games και αρκετά άλλα που περιέχει η ίδια η μηχανή στο αρχικό περιεχόμενο.

Η υλοποίηση του βασικού χάρτη έγινε με τη χρήση του εργαλείου σχεδίασης τοπίου(landscape), το οποίο βοηθάει τον προγραμματιστή με μια πληθώρα χαρακτηριστικών για την σχεδίαση και τροποποίηση του τοπίου.

Αρχικά, το βασικό εργαλείο τοπίου μας δίνει τη δυνατότητα σχεδίασης ενός πεδίου όπου μπορούμε να καθορίσουμε το μέγεθος και την λεπτομέρεια που θέλουμε να δώσουμε στην «πίστα». Αυτό μας δίνει μια σημαντική επιλογή, καθώς τα χαρακτηριστικά αυτά παίζουν πολύ σημαντικό ρόλο στην συνολική απόδοση του παιχνιδιού. Αυτό σημαίνει ότι εάν επιλέξουμε να δημιουργήσουμε ένα μεγάλο σε pixel πεδίο με μεγάλη λεπτομέρεια, θα έχει αντίκτυπο στο τι είδους υπολογιστικά συστήματα και με τι χαρακτηριστικά θα μπορούν να εκτελέσουν το παιχνίδι χωρίς να υπάρχει πρόβλημα απόδοσης ή ακόμα και να δημιουργήσει πρόβλημα στον ίδιο τον υπολογιστή.

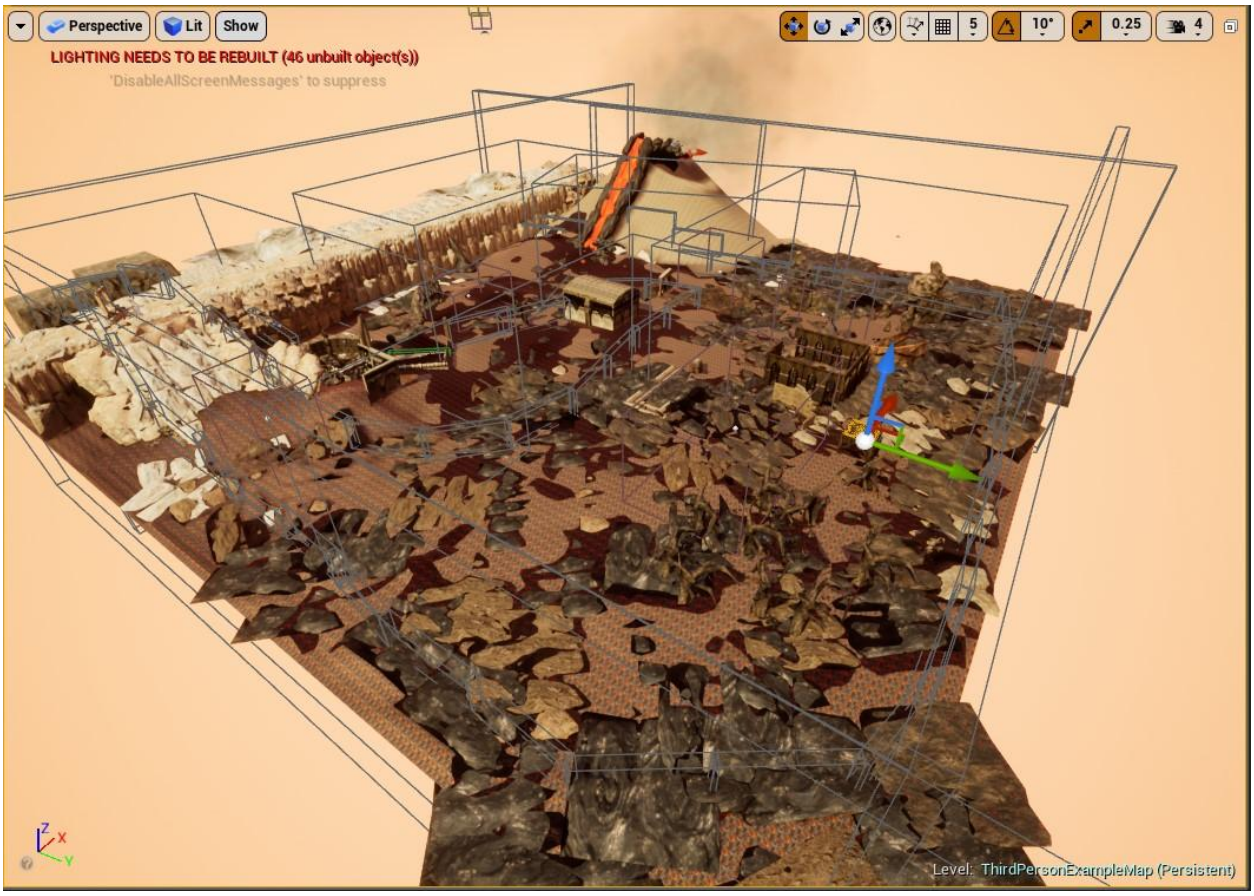
Δημιουργώντας ένα καινούριο πεδίο επιλέγουμε την τοποθεσία του στον «κόσμο» του παιχνιδιού, το μέγεθός του, με τι υλικό θα «γεμιστεί» αν και αυτό μπορεί να γίνει και αργότερα εάν δεν έχουμε αποφασίσει και την συνολική ανάλυση του. Ένα σημαντικό στοιχείο που μπορεί επίσης να επηρεάσει την απόδοση του παιχνιδιού είναι το μέγεθος των ενοτήτων και τα στοιχεία που περιλαμβάνονται σε αυτή.

Για παράδειγμα, εάν φτιάξουμε ένα πεδίο με 8*8 ενότητες όπου η κάθε ενότητα αποτελεί ένα στοιχείο θα έχει το ίδιο μέγεθος με ένα πεδίο που δημιουργήσαμε με 4*4 ενότητας η οποίες όμως είναι 2 ξεχωριστά στοιχεία. Το πεδίο με τις 4*4 ενότητες θα έχει καλύτερη απόδοση και θα είναι πιο «ελαφρύ» για την μηχανή, καθώς έχει λιγότερες ενότητες και επιτρέπει την ταχύτερη ανάλυση και σχεδίασή του από την μηχανή.

Ένα άλλο χρήσιμο κομμάτι της δημιουργίας τοπίου είναι η δυνατότητα εισαγωγής μιας ειδικά τοπογραφημένης περιοχής είτε σχεδιασμένη σε κάποιο άλλο μέσο(Maya3ds, Blender κ.λπ.) ή ακόμα και την εισαγωγή μιας πραγματικής τοποθεσίας. Η δυνατότητα αυτή μας δίνεται μέσω της επιλογής “Import from file” και από εκεί μπορούμε να εισάγουμε τον δικό μας προσχεδιασμένο χάρτη ή περιοχή με αρχεία heightmap που περιέχουν υψομετρικά στοιχεία που επεξεργάζεται η μηχανή του παιχνιδιού.

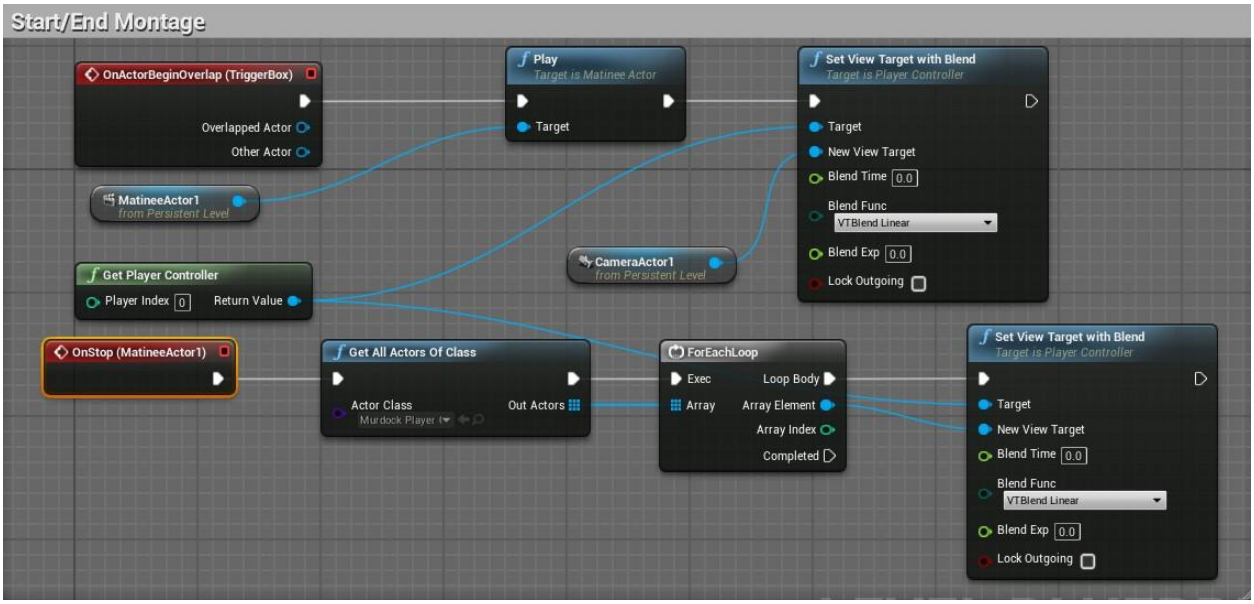
Για το δικό μας παιχνίδι δημιουργήσαμε τέσσερα διαφορετικά landscapes όπου τα δύο χρησιμοποιήθηκαν σαν κύριος χώρος την πίστας, με ένα κεντρικό τοπίο και ένα δευτερεύον για περισσότερες λεπτομέρειες και άλλα δύο landscapes για την υλοποίηση του ηφαιστείου.

Με τη χρήση των εργαλείων μορφοποίησης που μας παρέχει η unreal δημιουργήσαμε ένα πρότυπο ηφαιστιογενές τοπίο. Για την ολοκλήρωση του σκηνικού χρησιμοποιήσαμε τα assets του Paragon: Firelands.



Map Blueprint

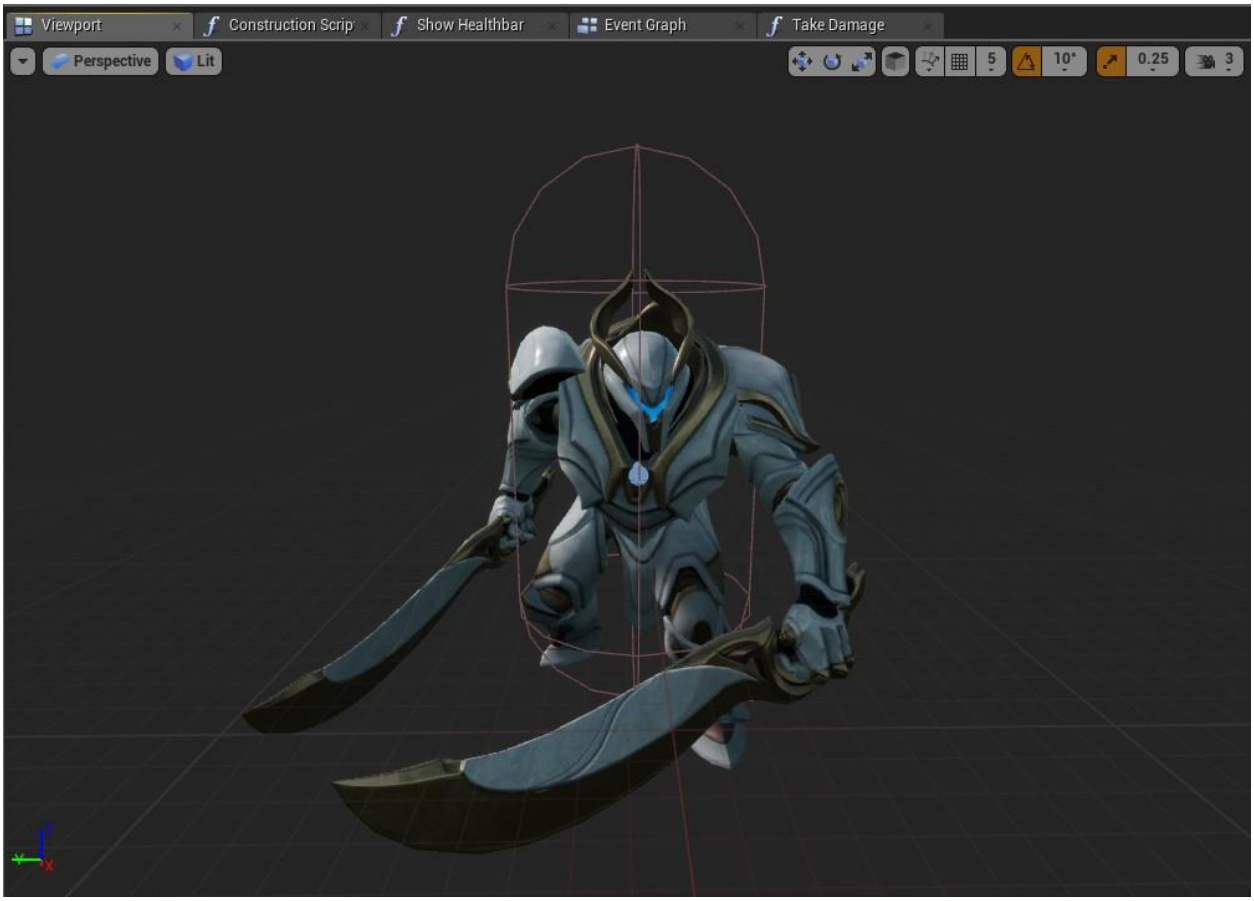
Στο blueprint του χάρτη μας έχουμε προσθέσαμε το μοντάζ που παίζει στην αρχή του παιχνιδιού. Ο παίκτης ξεκινάει το παιχνίδι και μόλις πατήσει στο κουτί που λειτουργεί ως σκανδάλη (triggerbox), το παιχνίδι ξεκινάει να δείχνει το μοντάζ που σχεδιάσαμε και παράλληλα ο χαρακτήρας μας σταματάει να λαμβάνει εντολές εισόδου από τον παίκτη. Μόλις σταματήσει το μοντάζ, ο παίκτης μπορεί να ξαναχρησιμοποιήσει τον χαρακτήρα και να συνεχίσει το παιχνίδι κανονικά.



12. ThirdPersonMap Blueprint

11.3 Minions

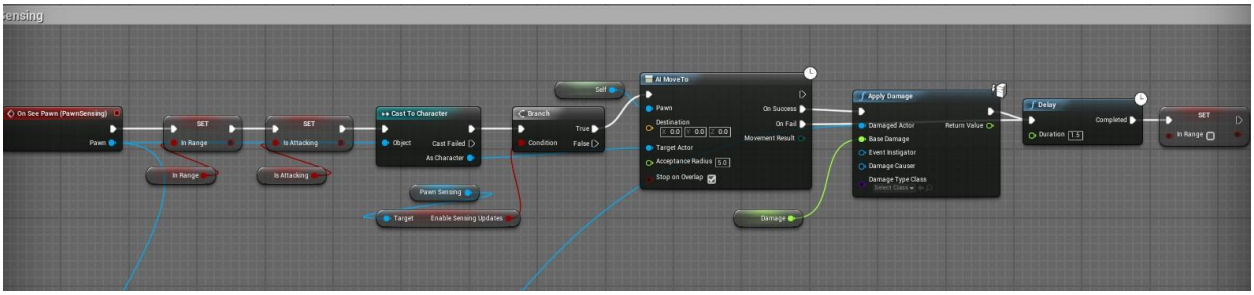
Για τη δημιουργία των «κακών» του παιχνιδιού χρησιμοποιήσαμε έτοιμους σκελετούς του παιχνιδιού Paragon που διατίθενται δωρεάν στο Marketplace από την Epic Games.



1. Ο "κακός" χαρακτήρας του παιχνιδιού

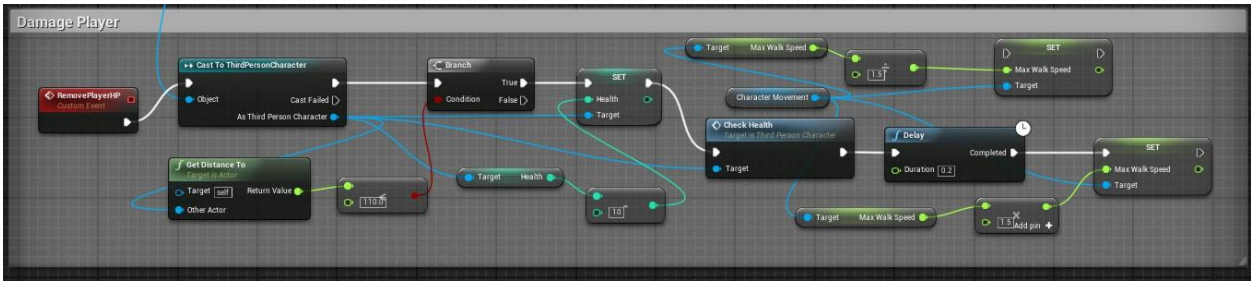
Το mesh δεν άλλαξε σε κάτι ενώ δημιουργήσαμε ένα blendspace για την κίνηση του χαρακτήρα, καθώς και για την προσαρμογή του animation επίθεσης με τη δημιουργία ξεχωριστού locomotion για το πάνω μέρος του σκελετού.

Αρχικά, στο blueprint του χαρακτήρα ξεκινάμε με την χρήση του event OnSeePawn για να καθορίσουμε το τι θα κάνει ο χαρακτήρας μόλις εντοπίσει τον παίκτη, όπου με δύο μεταβλητές (InRange και IsAttacking) το blueprint ελέγχει για την κατάσταση του χαρακτήρα. Εάν τηρεί τις δύο αυτές προϋποθέσεις, ο χαρακτήρας εντοπίζει τον παίκτη και κινείται προς την τοποθεσία του. Αν φτάσει σε σημείο όπου να μπορεί να επιτεθεί, το blueprint καταγράφει το χτύπημα στον παίκτη μας και ο χαρακτήρας κάνει μία παύση, ενώ αν ο παίκτης βρίσκεται ακόμα σε σημείο που να μπορεί να τον εντοπίσει το Pawn Sensing του χαρακτήρα, αυτός συνεχίζει όπως προηγουμένως, διαφορετικά σταματάει να κυνηγάει τον παίκτη μας.



2. AI Sensing

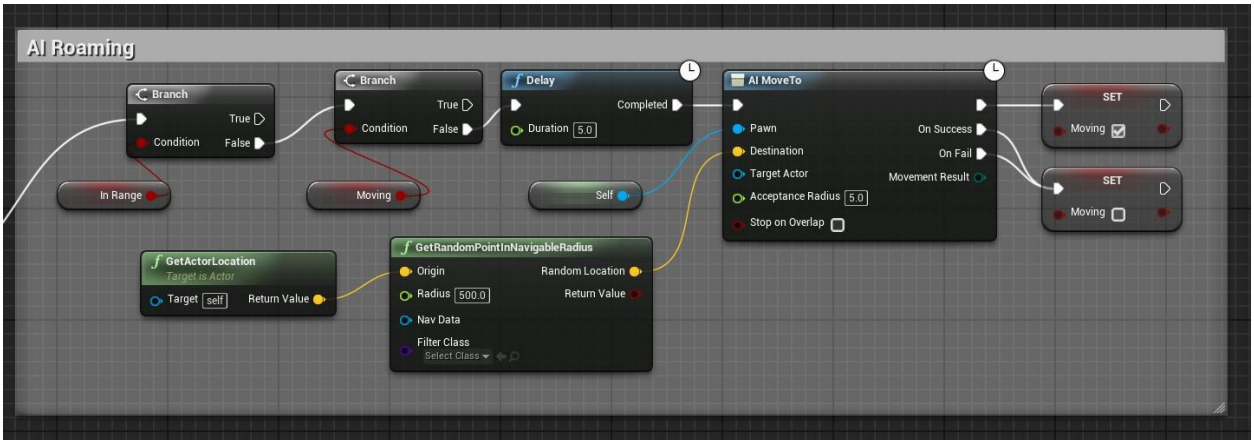
Επιπλέον, δημιουργήσαμε ένα custom event με όνομα RemovePlayerHP, όπου καθορίζουμε ποιος είναι ο στόχος του χαρακτήρα, δηλαδή ο παίκτης μας, και από εκεί ελέγχουμε αν ο χαρακτήρας είναι αρκετά κοντά στον παίκτη μας. Αν αυτό ισχύει, το blueprint δίνει εντολή στο χαρακτήρα να χρησιμοποιήσει ένα attack montage (το animation δηλαδή της επίθεσης του χαρακτήρα) και να κάνει πόντους ζημιάς στον παίκτη. Έπειτα ελέγχει την υγεία του παίκτη, κάνει μια παύση και ανάλογα με την απόστασή του από τον παίκτη επιταχύνει ή επιβραδύνει την ταχύτητα κίνησής του.



Στη συνέχεια, καθώς ο στόχος του χαρακτήρα είναι να μας σκοτώσει, ρυθμίσαμε στο blueprint την περίπτωση του τι πρέπει να κάνει όταν ο παίκτης πεθαίνει.

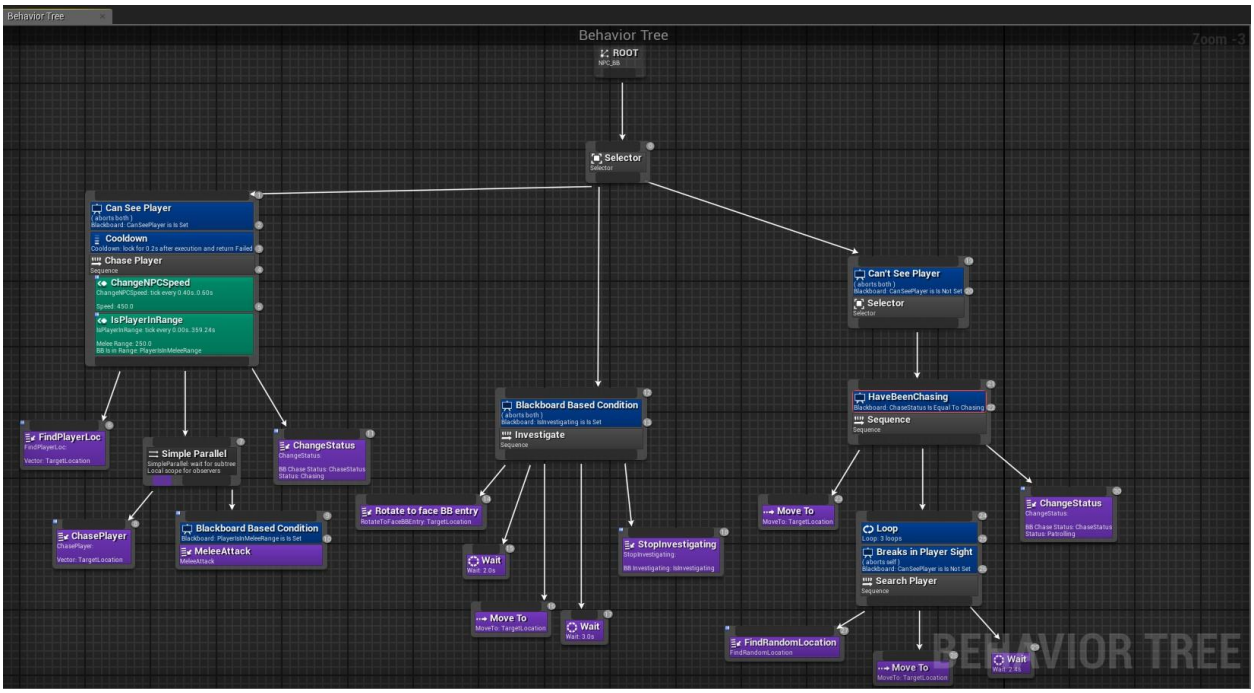
Με τη χρήση του event BeginPlay, το blueprint ελέγχει μόλις ξεκινήσει το παιχνίδι και κατά τη διάρκειά του την κατάσταση του παίκτη μας, έτσι με τη συνάρτηση Set Sensing Updates Enabled του Pawn Sensing που χρησιμοποιεί, ο χαρακτήρας λαμβάνει συνεχόμενα πληροφορίες για το περιβάλλον του και κυρίως του δίνεται η δυνατότητα να καταγράψει τον παίκτη αν τον εντοπίσει.

Με τη χρήση ελέγχου ροής που προσφέρει το branch node, το blueprint ελέγχει αν ο χαρακτήρας κινείται και αν βρίσκεται σε κοντινή απόσταση από τον παίκτη. Ανάλογα με την περίπτωση, ο χαρακτήρας λαμβάνει την τελευταία γνωστή θέση του που παίκτη στο χάρτη και την χρησιμοποιεί για να ορίσει ένα τυχαίο σημείο στο χάρτη σε ακτίνα 500 μονάδων (pixel της Unreal). Μετά, κινείται στο σημείο αυτό, με μια παύση κάθε φορά που επαναλαμβάνει αυτή τη διαδικασία.



3. AI Roaming

Για τη συμπεριφορά του χαρακτήρα δημιουργήσαμε ένα δέντρο συμπεριφοράς (behavior tree) και έναν πίνακα συμπεριφοράς(behavior tree blackboard) που συνοδεύει το δέντρο και παρέχει πληροφορίες και μεταβλητές για τη λειτουργία του.



4. Δέντρο Συμπεριφοράς

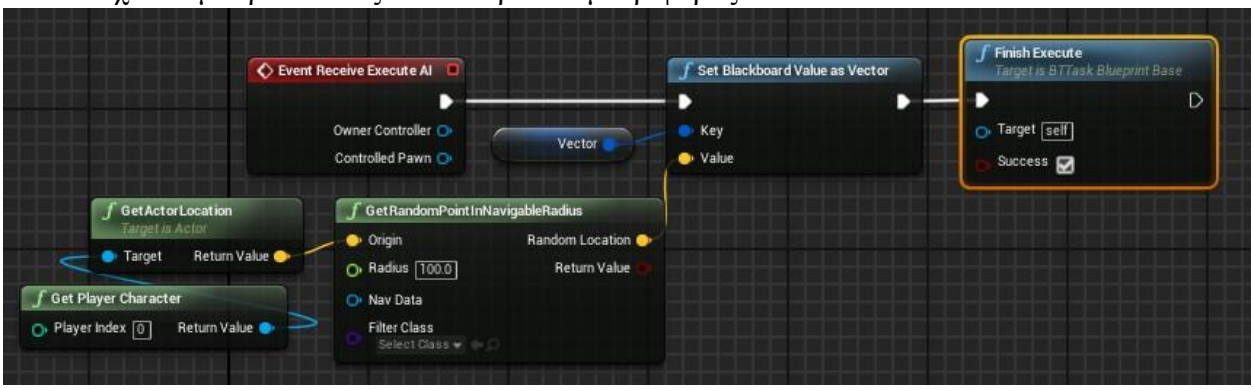
Το δέντρο συμπεριφοράς υλοποιεί καταστάσεις για το πώς αποκρίνεται σε διαφορετικά σενάρια, ανάλογα το περιβάλλον του και τις πληροφορίες που εντοπίζει στο περιβάλλον.

Το χρησιμοποιήσαμε για να προγραμματίσουμε τρεις βασικές καταστάσεις, να κυνηγάει τον χαρακτήρα μας μόλις τον εντοπίζει και να επιτεθεί αν βρίσκεται σε σημείο κοντά στον παίκτη (Chase Player), να ερευνά το περιβάλλον αν αντιληφθεί την παρουσία του παίκτη (Investigate) και να ερευνά την τελευταία γνωστή τοποθεσία του παίκτη (Search Player). Σημαντικό είναι να σημειωθεί ότι η ταξινόμηση των tasks και των επιλογέων παίζουν σημασία καθώς το behavior tree διαβάζει τα δεδομένα με αρχή από πάνω και αριστερά προς τα δεξιά. Επομένως, αν ένα task μετακινηθεί πιο αριστερά από κάποιο άλλο, παίρνει εκείνο προτεραιότητα και αν δεν δοθεί προσοχή υπάρχει περίπτωση να δημιουργηθούν καταστάσεις που πιθανόν να μην θέλουμε στο πρόγραμμά μας.

Στο υποδέντρο Chase Player χρησιμοποιούμε το κλειδιά CanSeePlayer και IsPlayerInRange που έχουμε ορίσει στον μαυροπίνακα που χρησιμοποιεί το δέντρο συμπεριφοράς. Η λειτουργία τους σαν μεταβλητές bool είναι να ελέγχουν αν ο χαρακτήρας βλέπει τον παίκτη και ακολούθως να ελέγξει αν ο παίκτης βρίσκεται σε ακτίνα επίθεσης ώστε να του επιτεθεί.

Επίσης χρησιμοποιεί την υπηρεσία ChangeNPCSpeed, η οποία αυξάνει την ταχύτητα του χαρακτήρα μόλις το CanSeePlayer γίνει θετικό και ο χαρακτήρας έχει εντοπίσει τον παίκτη για να ξεκινήσει να τρέχει προς την κατεύθυνσή του.

Αρχικά δημιουργήσαμε ένα custom task, το FindPlayerLoc που εντοπίζει την τοποθεσία του παίκτη στον χάρτη και παίρνει ένα τυχαίο σημείο στο χάρτη που έχει πρόσβαση σε μια ακτίνα 100 pixel. Έπειτα μετατρέπει αυτό το διάνυσμα σε κλειδί ώστε να μπορεί να το χρησιμοποιήσει σαν στοιχείο ο μαυροπίνακας του δέντρου συμπεριφοράς.

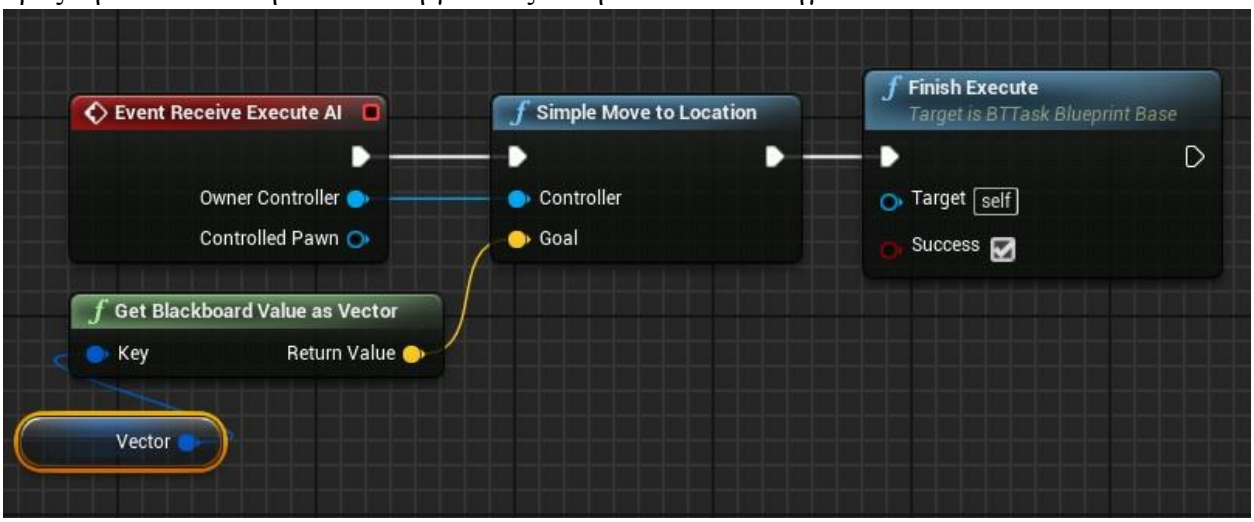


5. FindPlayerLoc

Έπειτα, στο δέντρο συμπεριφοράς επιλέγουμε ένα Simple Parallel (μπορούμε να το βρούμε με δεξί κλικ>Composites>Simple Parallel), με το οποίο μπορούμε να καθορίσουμε δύο εξόδους από μια είσοδο.

Αναφορικά, ο χαρακτήρας μας έχει εντοπίσει τον παίκτη μας και εκεί του δίνονται δύο επιλογές: πρώτα, μπορεί να κυνηγήσει τον παίκτη αν δεν βρίσκεται σε κοντά του και επομένως, μόλις βρεθεί σε κοντινή απόσταση, να μπορέσει να επιτεθεί.

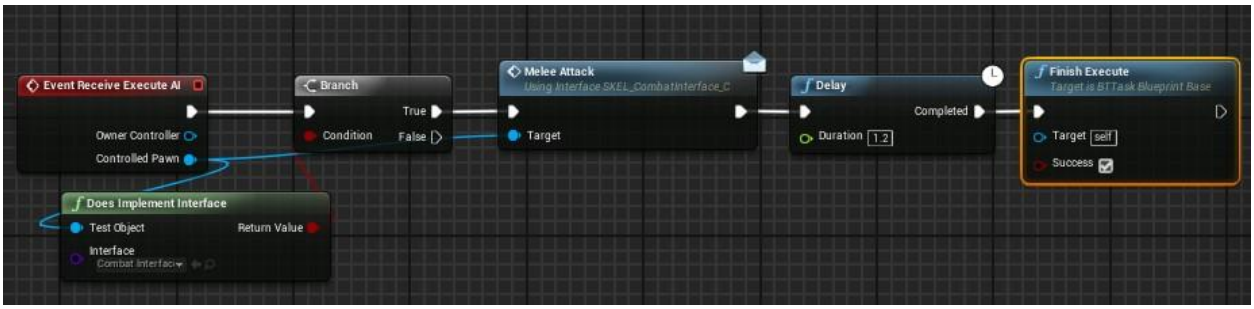
Με το task ChasePlayer, ο χαρακτήρας βρίσκει το διάνυσμα που αποθήκευσε από το προηγούμενο task όπου καταγράφει τη θέση του παίκτη μας και το χρησιμοποιεί για να κινηθεί προς την κατεύθυνση του παίκτη για να ξεκινήσει να τον κυνηγάει.



6. ChasePlayer

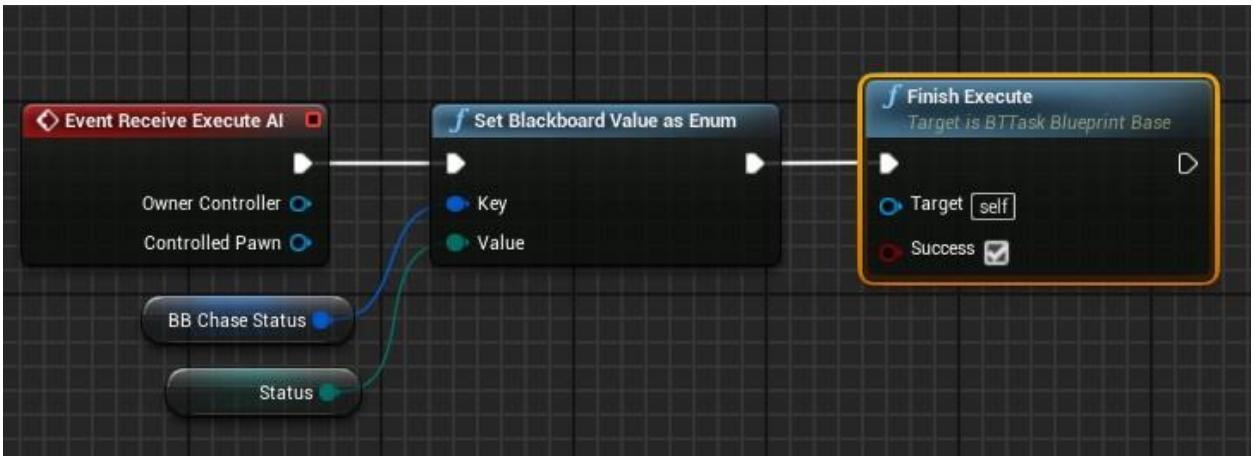
Ακολούθως, με έναν έλεγχο bool, το δέντρο καθορίζει αν ο χαρακτήρας βρίσκεται αρκετά κοντά στον παίκτη (PlayerIsInMeleeRange) και αν ναι, περνάει στο δευτερεύον task του simple parallel, που είναι η επίθεση στον παίκτη μας (MeleeAttack).

Από εκεί, ο χαρακτήρας χρησιμοποιεί το Blueprint Interface που δημιουργήσαμε (Combat Interface) που συνδέει την επίθεση με το αντίστοιχο κλειδί (MeleeAttack).



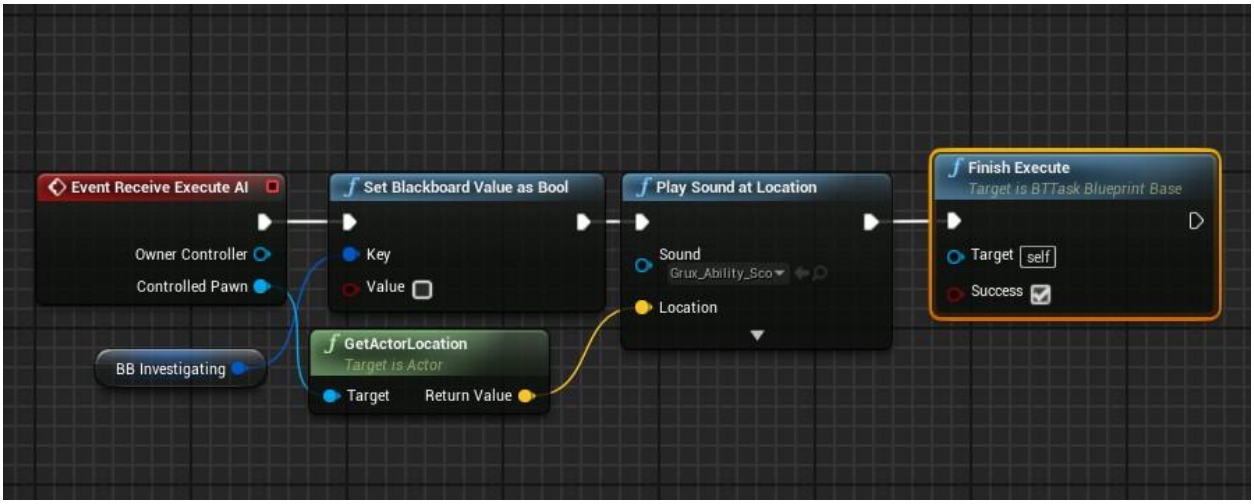
7. Melee Attack

Τέλος, χρησιμοποιούμε το task ChangeStatus σαν διακόπτη για να αλλάξουμε την κατάσταση του χαρακτήρα σε Chasing, όπου του δίνεται η εντολή να κυνηγάει τον παίκτη μας αν τηρεί τις παραπάνω προϋποθέσεις (τοποθεσία παίκτη, βλέπει τον παίκτη).



8. ChangeStatus

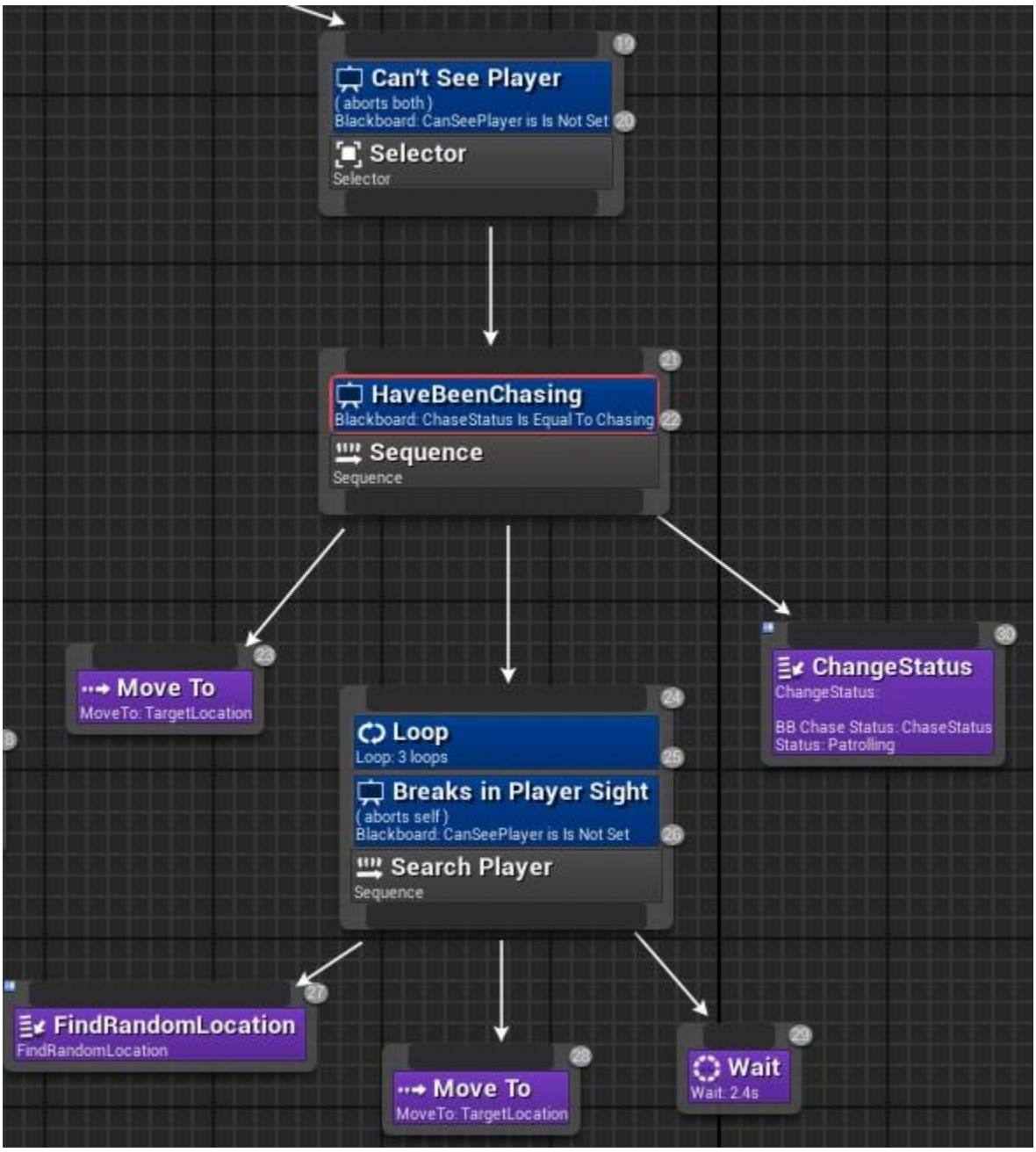
Στο υποδέντρο Investigate χρησιμοποιούμε απλά tasks για να καθορίσουμε πως κινείται ο χαρακτήρας ενώ ερευνά την περιοχή για τον παίκτη. Για να διακόψουμε τη διεργασία αυτή φτιάξαμε ένα ξεχωριστό task όπου ο χαρακτήρας διακόπτει τη διαδικασία:



9. BTTask StopInvestigating

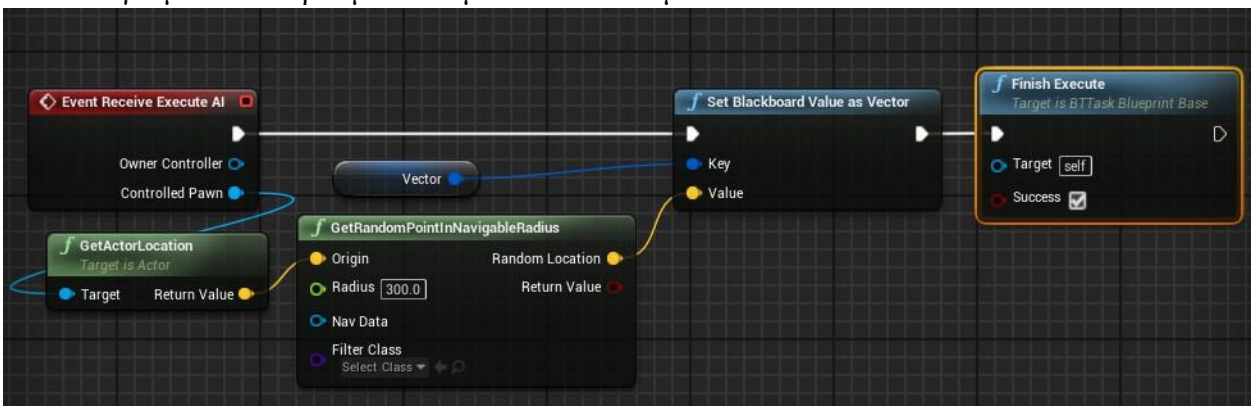
Στην περίπτωση που ο χαρακτήρας χάσει οπτική επαφή με τον παίκτη μας και δεν τον εντοπίσει με την έρευνά του, επιστρέφει σε κατάσταση αδράνειας. Αυτό καθορίζεται με κλειδιά που έχουν

οριστεί σαν επιλογείς κατάστασης στο τελευταίο υποδέντρο SearchPlayer. Αν ο χαρακτήρας δεν βλέπει τον παίκτη, το δέντρο συμπεριφοράς προχωράει στον επόμενο έλεγχο όπου ελέγχει την προηγούμενη κατάσταση του χαρακτήρα. Έτσι μπορεί ή να κινηθεί σε μια τυχαία τοποθεσία στον χάρτη ή αν εντοπίσει τον παίκτη, να γυρίσει ξανά σε κατάσταση ChasePlayer και να τον καταδιώξει.



10. SearchPlayer

Με το task FindRandomLocation, ο χαρακτήρας παίρνει την τωρινή του τοποθεσία στο χάρτη και επιλέγει μια καινούρια μέσα σε μια ακτίνα 300 μονάδων.



11. FindRandomLocation

Έπειτα, κινείται στην τοποθεσία αυτή και περιμένει για ένα διάστημα που ορίζουμε εμείς, στη συγκεκριμένη περίπτωση 2.4 δευτερόλεπτα. Τέλος, αν δεν εντοπίσει πάλι τον παίκτη συνεχίζει σε κατάσταση αδράνειας μέχρι να γίνει κάτι που μπορεί να αλλάξει την κατάστασή του και πάλι.

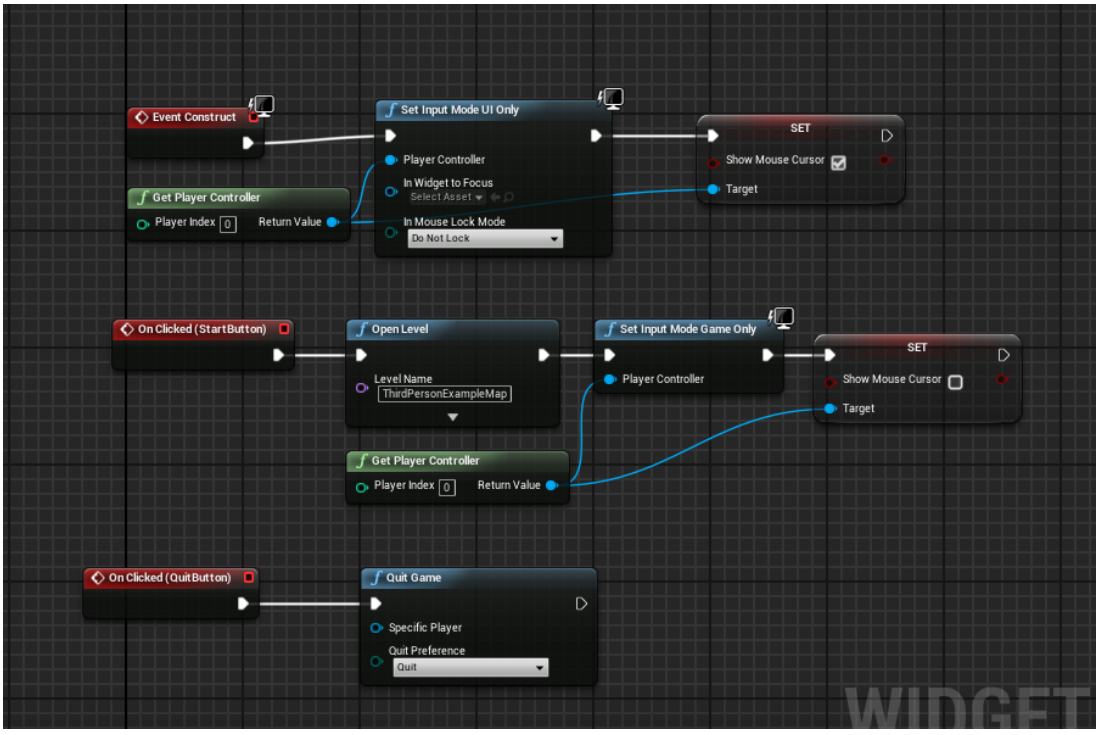
11.4 Τίτλοι, μενού

Όπως αναφέραμε και παραπάνω, οι τίτλοι έναρξης και το μενού είναι δύο βασικά και απαραίτητα γραφικά στοιχεία ενός παιχνιδιού, τόσο γιατί παρουσιάζουν το θέμα του παιχνιδιού σταδιακά αλλά και γιατί κατευθύνουν τον παίκτη. Αρχικά, οι τίτλοι αρχής είναι το εισαγωγικό μέρος ενός παιχνιδιού, όπου δίνεται η δυνατότητα στους δημιουργούς να παρουσιάσουν ποιοι έχουν συνεισφέρει στην δημιουργία του παιχνιδιού, είτε αυτό είναι με εργασία είτε με χορηγίες. Τα «μεγάλα» παιχνίδια που παράγονται με σκοπό το κέρδος δισεκατομμυρίων συχνά έχουν αρκετά στούντιο προγραμματιστών/σχεδιαστών που τους έχει ανατεθεί η σχεδίαση του παιχνιδιού και είναι απαραίτητο να αναφερθούν στα credits.

Επιπλέον, είναι αρκετές οι εταιρίες που χρηματίζουν τέτοια πρότζεκτ, ή ακόμα και εταιρίες που θα ήθελαν να προβάλουν και να διαφημίσουν τη δική τους τεχνολογία(AMD εναντίων Intel/NVidia).

Στο παιχνίδι μας δημιουργήσαμε ένα σύντομο εισαγωγικό σποτ με το λογότυπο του Πανεπιστημίου Πατρών. Αυτό έγινε με τη δημιουργία ενός widget και μιας δεύτερης κάμερας που τοποθετήθηκε στο «ναό» που δημιουργήσαμε εντός του παιχνιδιού. Χρησιμοποιήσαμε μια απλή εικόνα jpeg με τη χρήση του εργαλείου animation και το υπόβαθρο της δεύτερης κάμερας, το οποίο οδηγεί στην εμφάνιση των επιλογών του μενού. Αναφορικά, μπορέσαμε να δημιουργήσουμε ένα εφέ ξεθωριάσματος (fade out) χρησιμοποιώντας τη χρονολόγηση της εικόνας, χρονίζοντάς την να εμφανίζεται στο ένα δευτερόλεπτο της εισαγωγής και να αρχίσει να εξαφανίζεται σταδιακά στα δύο τελευταία δευτερόλεπτα του σποτ.

Για το μενού του παιχνιδιού χρησιμοποιήσαμε ένα απλό μοντέλο όπου ο παίκτης μπορεί να επιλέξει είτε να ξεκινήσει ή να τερματίσει τη λειτουργία του παιχνιδιού. Στο widget που δημιουργήσαμε, χρησιμοποιήσαμε δύο πάνελ για τις επιλογές και ακολούθησε η επιλογή της γραμματοσειράς. Τα πάνελ, πέρα από οπτική πληροφορία, είναι δύο κουμπιά συνδεδεμένα με γεγονότα ώστε να ανταποκρίνονται στις επιλογές του παίκτη. Στο πάτημα του κουμπιού έναρξης, η μηχανή φορτώνει το επίπεδο «ThirdPersonExampleMap» του παιχνιδιού και δίνει σήμα στον μηχανισμό χειρισμού των κινήσεων πληκτρολογίου/ποντικιού ώστε ο παίκτης να μπορεί να χειριστεί και να κινεί τον χαρακτήρα στο παιχνίδι. Αντίθετα, με το πάτημα του κουμπιού τερματισμού, ο παίκτης μπορεί να τερματίσει τη λειτουργία του παιχνιδιού.



Βιβλιογραφία

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

https://www.statista.com/topics/1680/gaming/

https://www.telegraph.co.uk/technology/news/11008989/Playing-video-games-for-up-to-an-hour-a-day-is-good-for-children.html

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

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

https://www.pluralsight.com/blog/film-games/unreal-engine-4-vs-unity-game-engine-best

https://medium.com/@thinkwik/cryengine-vs-unreal-vs-unity-select-the-best-game-engine-eaca64c60e3e

https://www.creativebloq.com/advice/unity-vs-unreal-engine-which-game-engine-is-for-you

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

https://en.wikipedia.org/wiki/List_of_video_game_genres#Action

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

ΚΕΦΑΛΑΙΟ 12^{ο} Κώδικας Project

Κώδικας Project

```
// Copyright 1998-2018 Epic Games, Inc. All Rights Reserved.


#pragma once

/*=============================================================================
	World.h: UWorld definition.
=============================================================================*/

#include "CoreMinimal.h"
#include "HAL/ThreadSafeCounter.h"
#include "UObject/ObjectMacros.h"
#include "UObject/UObjectGlobals.h"
#include "UObject/Object.h"
#include "Misc/Guid.h"
#include "UObject/Class.h"
#include "Engine/EngineTypes.h"
#include "Engine/EngineBaseTypes.h"
#include "CollisionQueryParams.h"
#include "WorldCollision.h"
#include "GameFramework/Pawn.h"
#include "EngineDefines.h"
#include "Engine/Blueprint.h"
#include "Engine/PendingNetGame.h"
```

```
#include "Engine/LatentActionManager.h"
#include "Engine/GameInstance.h"
#include "Engine/DemoNetDriver.h"

#include "World.generated.h"

class ABrush;
class ACameraActor;
class AController;
class AGameModeBase;
class AGameStateBase;
class AMatineeActor;
class APhysicsVolume;
class APlayerController;
class AWorldSettings;
class Error;
class FPhysScene;
class FTimerManager;
class FUniqueNetId;
class FWorldInGamePerformanceTrackers;
class IInterface_PostProcessVolume;
class UAISystemBase;
class UCanvas;
class UGameViewportClient;
class ULevelStreaming;
class ULocalPlayer;
class UMaterialParameterCollection;
class UMaterialParameterCollectionInstance;
class UModel;
```

```
class UNavigationSystem;
class UNetConnection;
class UNetDriver;
class UPrimitiveComponent;
class UTexture2D;
struct FUniqueNetIdRepl;
struct FEncryptionKeyResponse;

template<typename,typename> class TOctree;

/**
 * Misc. Iterator types
 *
 */
typedef TArray<TWeakObjectPtr<AController> >::TConstIterator FConstControllerIterator;
typedef TArray<TWeakObjectPtr<APlayerController> >::TConstIterator FConstPlayerControllerIterator;
typedef TArray<TWeakObjectPtr<APawn> >::TConstIterator FConstPawnIterator;
typedef TArray<TWeakObjectPtr<ACameraActor> >::TConstIterator FConstCameraActorIterator;
typedef TArray<class ULevel*>::TConstIterator FConstLevelIterator;
typedef TArray<TWeakObjectPtr<APhysicsVolume> >::TConstIterator FConstPhysicsVolumeIterator;

DECLARE_LOG_CATEGORY_EXTERN(LogSpawn, Warning, All);

DECLARE_MULTICAST_DELEGATE_OneParam(FOnActorSpawned, AActor*);


/** Proxy class that allows verification on GWorld accesses. */
class UWorldProxy
{
```

```
public:

	UWorldProxy() :
		World(NULL)
	{}

	inline UWorld* operator->()
	{
		// GWorld is changed often on the game thread when in PIE, accessing on any other thread is going to be a race condition
		// In general, the rendering thread should not dereference UObjects, unless there is a mechanism in place to make it safe
		checkSlow(IsInGameThread());
		return World;
	}

	inline const UWorld* operator->() const
	{
		checkSlow(IsInGameThread());
		return World;
	}

	inline UWorld& operator*()
	{
		checkSlow(IsInGameThread());
		return *World;
	}

	inline const UWorld& operator*() const
```

```
	{
		checkSlow(IsInGameThread());
		return *World;
	}

	inline UWorldProxy& operator=(UWorld* InWorld)
	{
		World = InWorld;
		return *this;
	}

	inline UWorldProxy& operator=(const UWorldProxy& InProxy)
	{
		World = InProxy.World;
		return *this;
	}

	inline bool operator==(const UWorldProxy& Other) const
	{
		return World == Other.World;
	}

	inline operator UWorld*() const
	{
		checkSlow(IsInGameThread());
		return World;
	}

	inline UWorld* GetReference()
```

```
	{
		checkSlow(IsInGameThread());
		return World;
	}

private:

	UWorld* World;
};

/** class that encapsulates seamless world traveling */
class FSeamlessTravelHandler
{
private:
	/** set when a transition is in progress */
	bool bTransitionInProgress;
	/** URL we're traveling to */
	FURL PendingTravelURL;
	/** Guid of the destination map (for finding it in the package cache if autodownloaded) */
	FGuid PendingTravelGuid;
	/** whether or not we've transitioned to the entry level and are now moving on to the specified map */
	bool bSwitchedToDefaultMap;
	/** set to the loaded package once loading is complete. Transition to it is performed in the next tick where it's safe to perform the required operations */
	UObject* LoadedPackage;
	/** the world we are travelling from */
	UWorld* CurrentWorld;
	/** set to the loaded world object inside that package. This is added to the root set (so that if a GC gets in between it won't break loading) */
```

```
	UWorld* LoadedWorld;

	/** while set, pause at midpoint (after loading transition level, before loading final destination) */
	bool bPauseAtMidpoint;

	/** set when we started a new travel in the middle of a previous one and still need to clean up that previous attempt */
	bool bNeedCancelCleanUp;

	/** The context we are running in. Can be used to get the FWorldContext from Engine*/
	FName WorldContextHandle;

	/** Real time which we started traveling at  */
	double SeamlessTravelStartTime = 0.f;


	/** copy data between the old world and the new world */
	void CopyWorldData();


	/** callback sent to async loading code to inform us when the level package is complete */
	void SeamlessTravelLoadCallback(const FName& PackageName, UPackage* LevelPackage, EAsyncLoadingResult::Type Result);


	void SetHandlerLoadedData(UObject* InLevelPackage, UWorld* InLoadedWorld);


	/** called to kick off async loading of the destination map and any other packages it requires */
	void StartLoadingDestination();


public:
	FSeamlessTravelHandler()
		: bTransitionInProgress(false)
		, PendingTravelURL(NoInit)
		, PendingTravelGuid(0, 0, 0, 0)
		, bSwitchedToDefaultMap(false)
```

```
        , LoadedPackage(NULL)
        , CurrentWorld(NULL)
        , LoadedWorld(NULL)
        , bPauseAtMidpoint(false)
        , bNeedCancelCleanUp(false)
{}

/** starts traveling to the given URL. The required packages will be loaded async and Tick() will perform the transition once we are ready
 * @param InURL the URL to travel to
 * @param InGuid the GUID of the destination map package
 * @return whether or not we succeeded in starting the travel
 */
bool StartTravel(UWorld* InCurrentWorld, const FURL& InURL, const FGuid& InGuid);

/** @return whether a transition is already in progress */
FORCEINLINE bool IsInTransition() const
{
        return bTransitionInProgress;
}
/** @return if current transition has switched to the default map; returns false if no transition is in progress */
FORCEINLINE bool HasSwitchedToDefaultMap() const
{
        return IsInTransition() && bSwitchedToDefaultMap;
}

/** @return the destination map that is being travelled to via seamless travel */
inline FString GetDestinationMapName() const
```

```
	{
		return (IsInTransition() ? PendingTravelURL.Map : TEXT(""));
	}

	/** @return the destination world that has been loaded asynchronously by the seamless travel handler. */
	inline const UWorld* GetLoadedWorld() const
	{
		return LoadedWorld;
	}

	/** cancels transition in progress */
	void CancelTravel();

	/** turns on/off pausing after loading the transition map
	 * only valid during travel, before we've started loading the final destination
	 * @param bNowPaused - whether the transition should now be paused
	 */
	void SetPauseAtMidpoint(bool bNowPaused);

	/**
	 * Ticks the transition; handles performing the world switch once the required packages have been loaded
	 *
	 * @returns	The new primary world if the world has changed, null if it has not
	 */
	ENGINE_API UWorld* Tick();
};
```

```
/**
 * Helper structure encapsulating functionality used to defer marking actors and their components as pending
 * kill till right before garbage collection by registering a callback.
 */
struct ENGINE_API FLevelStreamingGCHelper
{
	/** Called when streamed out levels are going to be garbage collected */
	DECLARE_MULTICAST_DELEGATE(FOnGCStreamedOutLevelsEvent);
	static FOnGCStreamedOutLevelsEvent OnGCStreamedOutLevels;

	/**
	 * Register with the garbage collector to receive callbacks pre and post garbage collection
	 */
	static void AddGarbageCollectorCallback();

	/**
	 * Request to be unloaded.
	 *
	 * @param InLevel		Level that should be unloaded
	 */
	static void RequestUnload( ULevel* InLevel );

	/**
	 * Cancel any pending unload requests for passed in Level.
	 */
	static void CancelUnloadRequest( ULevel* InLevel );
```

```
	/**
	 * Prepares levels that are marked for unload for the GC call by marking their actors and components as
	 * pending kill.
	 */
	static void PrepareStreamedOutLevelsForGC();

	/**
	 * Verify that the level packages are no longer around.
	 */
	static void VerifyLevelsGotRemovedByGC();

	/**
	 * @return		The number of levels pending a purge by the garbage collector
	 */
	static int32 GetNumLevelsPendingPurge();

private:
	/** Static array of levels that should be unloaded */
	static TArray<TWeakObjectPtr<ULevel> > LevelsPendingUnload;
	/** Static array of level packages that have been marked by PrepareStreamedOutLevelsForGC */
	static TArray<FName> LevelPackageNames;
};

/** Saved editor viewport state information */
USTRUCT()
struct ENGINE_API FLevelViewportInfo
{
	GENERATED_BODY()
```

```
	/** Where the camera is positioned within the viewport. */
	UPROPERTY()
	FVector CamPosition;

	/** The camera's position within the viewport. */
	UPROPERTY()
	FRotator CamRotation;

	/** The zoom value  for orthographic mode. */
	UPROPERTY()
	float CamOrthoZoom;

	/** Whether camera settings have been systematically changed since the last level viewport update. */
	UPROPERTY()
	bool CamUpdated;

	FLevelViewportInfo()
		: CamPosition(FVector::ZeroVector)
		, CamRotation(FRotator::ZeroRotator)
		, CamOrthoZoom(DEFAULT_ORTHOZOOM)
		, CamUpdated(false)
	{
	}

	FLevelViewportInfo(const FVector& InCamPosition, const FRotator& InCamRotation, float InCamOrthoZoom)
		: CamPosition(InCamPosition)
```

```
		, CamRotation(InCamRotation)
		, CamOrthoZoom(InCamOrthoZoom)
		, CamUpdated(false)
	{
	}

	// Needed for backwards compatibility for VER_UE4_ADD_EDITOR_VIEWS, can be removed along with it
	friend FArchive& operator<<( FArchive& Ar, FLevelViewportInfo& I )
	{
		Ar << I.CamPosition;
		Ar << I.CamRotation;
		Ar << I.CamOrthoZoom;

		if ( Ar.IsLoading() )
		{
			I.CamUpdated = true;

			if ( I.CamOrthoZoom == 0.f )
			{
				I.CamOrthoZoom = DEFAULT_ORTHOZOOM;
			}
		}

		return Ar;
	}
};

/**
```

```
* Tick function that starts the physics tick

**/

USTRUCT()

struct FStartPhysicsTickFunction : public FTickFunction

{

	GENERATED_USTRUCT_BODY()


	/** World this tick function belongs to **/

	class UWorld*	Target;


	/**

		* Abstract function actually execute the tick.

		* @param DeltaTime - frame time to advance, in seconds

		* @param TickType - kind of tick for this frame

		* @param CurrentThread - thread we are executing on, useful to pass along as new tasks are created

		* @param MyCompletionGraphEvent - completion event for this task. Useful for holding the completetion of this task until certain child tasks are complete.

	**/

	virtual void ExecuteTick(float DeltaTime, enum ELevelTick TickType, ENamedThreads::Type CurrentThread, const FGraphEventRef& MyCompletionGraphEvent) override;

	/** Abstract function to describe this tick. Used to print messages about illegal cycles in the dependency graph **/

	virtual FString DiagnosticMessage() override;

};


template<>

struct TStructOpsTypeTraits<FStartPhysicsTickFunction> : public TStructOpsTypeTraitsBase2<FStartPhysicsTickFunction>

{
```

```
	enum
	{
		WithCopy = false
	};
};

/**
* Tick function that ends the physics tick
**/
USTRUCT()
struct FEndPhysicsTickFunction : public FTickFunction
{
	GENERATED_USTRUCT_BODY()

	/** World this tick function belongs to **/
	class UWorld*	Target;

	/**
		* Abstract function actually execute the tick.
		* @param DeltaTime - frame time to advance, in seconds
		* @param TickType - kind of tick for this frame
		* @param CurrentThread - thread we are executing on, useful to pass along as new tasks are created
		* @param MyCompletionGraphEvent - completion event for this task. Useful for holding the completetion of this task until certain child tasks are complete.
	**/
	virtual void ExecuteTick(float DeltaTime, enum ELevelTick TickType, ENamedThreads::Type CurrentThread, const FGraphEventRef& MyCompletionGraphEvent) override;
	/** Abstract function to describe this tick. Used to print messages about illegal cycles in the dependency graph **/
```

```
	virtual FString DiagnosticMessage() override;
};

template<>
struct TStructOpsTypeTraits<FEndPhysicsTickFunction> : public
TStructOpsTypeTraitsBase2<FEndPhysicsTickFunction>
{
	enum
	{
		WithCopy = false
	};
};

/**
* Tick function that starts the cloth tick
**/
USTRUCT()
struct FStartAsyncSimulationFunction : public FTickFunction
{
	GENERATED_USTRUCT_BODY()

	/** World this tick function belongs to **/
	class UWorld*	Target;

	/**
	* Abstract function actually execute the tick.
	* @param DeltaTime - frame time to advance, in seconds
	* @param TickType - kind of tick for this frame
```

```
	* @param CurrentThread - thread we are executing on, useful to pass along as new tasks are created

	* @param MyCompletionGraphEvent - completion event for this task. Useful for holding the completetion of this task until certain child tasks are complete.

	**/

	virtual void ExecuteTick(float DeltaTime, enum ELevelTick TickType, ENamedThreads::Type CurrentThread, const FGraphEventRef& MyCompletionGraphEvent) override;

	/** Abstract function to describe this tick. Used to print messages about illegal cycles in the dependency graph **/

	virtual FString DiagnosticMessage() override;

};


template<>

struct TStructOpsTypeTraits<FStartAsyncSimulationFunction> : public TStructOpsTypeTraitsBase2<FStartAsyncSimulationFunction>

{

	enum

	{

		WithCopy = false

	};

};


/* Struct of optional parameters passed to SpawnActor function(s). */

PRAGMA_DISABLE_DEPRECATION_WARNINGS // Required for auto-generated functions referencing bNoCollisionFail

struct ENGINE_API FActorSpawnParameters

{

	FActorSpawnParameters();


	/* A name to assign as the Name of the Actor being spawned. If no value is specified, the name of the spawned Actor will be automatically generated using the form [Class]_[Number]. */
```

```
        FName Name;


        /* An Actor to use as a template when spawning the new Actor. The spawned Actor will be
initialized using the property values of the template Actor. If left NULL the class default object (CDO) will
be used to initialize the spawned Actor. */
        AActor* Template;


        /* The Actor that spawned this Actor. (Can be left as NULL). */
        AActor* Owner;


        /* The APawn that is responsible for damage done by the spawned Actor. (Can be left as NULL).
*/
        APawn*          Instigator;


        /* The ULevel to spawn the Actor in, i.e. the Outer of the Actor. If left as NULL the Outer of the
Owner is used. If the Owner is NULL the persistent level is used. */
        class     ULevel* OverrideLevel;


        /** Method for resolving collisions at the spawn point. Undefined means no override, use the
actor's setting. */
        ESpawnActorCollisionHandlingMethod SpawnCollisionHandlingOverride;


private:


        friend class UPackageMapClient;


        /* Is the actor remotely owned. This should only be set true by the package map when it is
creating an actor on a client that was replicated from the server. */
        uint16   bRemoteOwned:1;


public:
```

```
	bool IsRemoteOwned() const { return bRemoteOwned; }

	/* Determines whether spawning will not fail if certain conditions are not met. If true, spawning will not fail because the class being spawned is `bStatic=true` or because the class of the template Actor is not the same as the class of the Actor being spawned. */
	uint16	bNoFail:1;

	/* Determines whether the construction script will be run. If true, the construction script will not be run on the spawned Actor. Only applicable if the Actor is being spawned from a Blueprint. */
	uint16	bDeferConstruction:1;

	/* Determines whether or not the actor may be spawned when running a construction script. If true spawning will fail if a construction script is being run. */
	uint16	bAllowDuringConstructionScript:1;

#if WITH_EDITOR
	/** Determines whether the begin play cycle will run on the spawned actor when in the editor. */
	uint16 bTemporaryEditorActor:1;
#endif

	/* Flags used to describe the spawned actor/object instance. */
	EObjectFlags ObjectFlags;
};
PRAGMA_ENABLE_DEPRECATION_WARNINGS


/**
 *  This encapsulate World's async trace functionality. This contains two buffers of trace data buffer and alternates it for each tick.
```

```
 *
 *      You can use async trace using following APIs : AsyncLineTrace, AsyncSweep, AsyncOverlap
 *      When you use those APIs, it will be saved to AsyncTraceData
 *      FWorldAsyncTraceState contains two buffers to rotate each frame as you might need the result in the next frame
 *      However, if you do not get the result by next frame, the result will be discarded.
 *      Use Delegate if you would like to get the result right away when available.
 */
struct ENGINE_API FWorldAsyncTraceState
{
        FWorldAsyncTraceState();

        /** Get the Buffer for input Frame */
        FORCEINLINE AsyncTraceData& GetBufferForFrame       (int32 Frame) { return DataBuffer[ Frame          % 2]; }
        /** Get the Buffer for Current Frame */
        FORCEINLINE AsyncTraceData& GetBufferForCurrentFrame ()          { return DataBuffer[ CurrentFrame     % 2]; }
        /** Get the Buffer for Previous Frame */
        FORCEINLINE AsyncTraceData& GetBufferForPreviousFrame()          { return DataBuffer[(CurrentFrame + 1) % 2]; }

        /** Async Trace Data Buffer Array. For now we only saves 2 frames. **/
        AsyncTraceData DataBuffer[2];

        /** Used as counter for Buffer swap for DataBuffer. Right now it's only 2, but it can change. */
        int32 CurrentFrame;
};

#if WITH_EDITOR
```

```
/* FAsyncPreRegisterDDCRequest - info about an async DDC request that we're going to wait on before registering components */
class ENGINE_API FAsyncPreRegisterDDCRequest
{
	/* DDC Key used for the request */
	FString DDCKey;

	/* Handle for Async DDC request. 0 if no longer invalid. */
	uint32 Handle;
public:
	/** constructor */
	FAsyncPreRegisterDDCRequest(const FString& InKey, uint32 InHandle)
	: DDCKey(InKey)
	, Handle(InHandle)
	{}

	/** destructor */
	~FAsyncPreRegisterDDCRequest();

	/** returns true if the request is complete */
	bool PollAsynchronousCompletion();

	/** waits until the request is complete */
	void WaitAsynchronousCompletion();

	/** returns true if the DDC returned the results requested. Must only be called once. */
	bool GetAsynchronousResults(TArray<uint8>& OutData);

	/** get the DDC key associated with this request */
```

```
	const FString& GetKey() const { return DDCKey; }
};
#endif

/**
 * Contains a group of levels of a particular ELevelCollectionType within a UWorld
 * and the context required to properly tick/update those levels. This object is move-only.
 */
USTRUCT()
struct ENGINE_API FLevelCollection
{
	GENERATED_BODY()

	FLevelCollection();

	FLevelCollection(const FLevelCollection&) = delete;
	FLevelCollection& operator=(const FLevelCollection&) = delete;

	FLevelCollection(FLevelCollection&& Other);
	FLevelCollection& operator=(FLevelCollection&& Other);

	/** The destructor will clear the cached collection pointers in this collection's levels. */
	~FLevelCollection();

	/** Gets the type of this collection. */
	ELevelCollectionType GetType() const { return CollectionType; }

	/** Sets the type of this collection. */
	void SetType(const ELevelCollectionType InType) { CollectionType = InType; }
```

```
	/** Gets the game state for this collection. */
	AGameStateBase* GetGameState() const { return GameState; }

	/** Sets the game state for this collection. */
	void SetGameState(AGameStateBase* const InGameState) { GameState = InGameState; }

	/** Gets the net driver for this collection. */
	UNetDriver* GetNetDriver() const { return NetDriver; }

	/** Sets the net driver for this collection. */
	void SetNetDriver(UNetDriver* const InNetDriver) { NetDriver = InNetDriver; }

	/** Gets the demo net driver for this collection. */
	UDemoNetDriver* GetDemoNetDriver() const { return DemoNetDriver; }

	/** Sets the demo net driver for this collection. */
	void SetDemoNetDriver(UDemoNetDriver* const InDemoNetDriver) { DemoNetDriver =
InDemoNetDriver; }

	/** Returns the set of levels in this collection. */
	const TSet<ULevel*>& GetLevels() const { return Levels; }

	/** Adds a level to this collection and caches the collection pointer on the level for fast access.
*/
	void AddLevel(ULevel* const Level);

	/** Removes a level from this collection and clears the cached collection pointer on the level. */
	void RemoveLevel(ULevel* const Level);
```

```
	/** Sets this collection's PersistentLevel and adds it to the Levels set. */
	void SetPersistentLevel(ULevel* const Level);

	/** Returns this collection's PersistentLevel. */
	ULevel* GetPersistentLevel() const { return PersistentLevel; }

	/** Gets whether this collection is currently visible. */
	bool IsVisible() const { return bIsVisible; }

	/** Sets whether this collection is currently visible. */
	void SetIsVisible(const bool bInIsVisible) { bIsVisible = bInIsVisible; }

private:
	/** The type of this collection. */
	ELevelCollectionType CollectionType;

	/**
	 * The GameState associated with this collection. This may be different than the UWorld's GameState
	 * since the source collection and the duplicated collection will have their own instances.
	 */
	UPROPERTY()
	class AGameStateBase* GameState;

	/**
	 * The network driver associated with this collection.
	 * The source collection and the duplicated collection will have their own instances.
	 */
```

```
	UPROPERTY()
	class UNetDriver* NetDriver;

	/**
	 * The demo network driver associated with this collection.
	 * The source collection and the duplicated collection will have their own instances.
	 */
	UPROPERTY()
	class UDemoNetDriver* DemoNetDriver;

	/**
	 * The persistent level associated with this collection.
	 * The source collection and the duplicated collection will have their own instances.
	 */
	UPROPERTY()
	class ULevel* PersistentLevel;

	/** All the levels in this collection. */
	UPROPERTY()
	TSet<ULevel*> Levels;

	/**
	 * Whether or not this collection is currently visible. While invisible, actors in this collection's
	 * levels will not be rendered and sounds originating from levels in this collection will not be played.
	 */
	bool bIsVisible;
};
```

```
template<>
struct TStructOpsTypeTraits<FLevelCollection> : public TStructOpsTypeTraitsBase2<FLevelCollection>
{
	enum
	{
		WithCopy = false
	};
};

/**
 * A helper RAII class to set the relevant context on a UWorld for a particular FLevelCollection within a scope.
 * The constructor will set the PersistentLevel, GameState, NetDriver, and DemoNetDriver on the world and
 * the destructor will restore the original values.
 */
class ENGINE_API FScopedLevelCollectionContextSwitch
{
public:
	/**
	 * Constructor that will save the current relevant values of InWorld
	 * and set the collection's context values for InWorld.
	 * The constructor that takes an index is preferred, but this one
	 * still exists for backwards compatibility.
	 *
	 * @param InLevelCollection The collection's context to use
	 * @param InWorld The world on which to set the context.
	 */
	FScopedLevelCollectionContextSwitch(const FLevelCollection* const InLevelCollection, UWorld* const InWorld);
```

```
		/**
		 * Constructor that will save the current relevant values of InWorld
		 * and set the collection's context values for InWorld.
		 *
		 * @param InLevelCollectionIndex The index of the collection to use
		 * @param InWorld The world on which to set the context.
		 */
		FScopedLevelCollectionContextSwitch(int32 InLevelCollectionIndex, UWorld* const InWorld);

		/** The destructor restores the context on the world that was saved in the constructor. */
		~FScopedLevelCollectionContextSwitch();

private:
		class UWorld* World;
		int32 SavedTickingCollectionIndex;
};

/**
 * The World is the top level object representing a map or a sandbox in which Actors and Components will exist and be rendered.
 *
 * A World can be a single Persistent Level with an optional list of streaming levels that are loaded and unloaded via volumes and blueprint functions
 * or it can be a collection of levels organized with a World Composition.
 *
 * In a standalone game, generally only a single World exists except during seamless area transitions when both a destination and current world exists.
 * In the editor many Worlds exist: The level being edited, each PIE instance, each editor tool which has an interactive rendered viewport, and many more.
```

```
 *
 */

UCLASS(customConstructor, config=Engine)
class ENGINE_API UWorld final : public UObject, public FNetworkNotify
{
	GENERATED_UCLASS_BODY()

	~UWorld();

#if WITH_EDITORONLY_DATA
	/** List of all the layers referenced by the world's actors */
	UPROPERTY()
	TArray< class ULayer* > Layers;

	// Group actors currently "active"
	UPROPERTY(transient)
	TArray<AActor*> ActiveGroupActors;

	/** Information for thumbnail rendering */
	UPROPERTY(VisibleAnywhere, Instanced, Category=Thumbnail)
	class UThumbnailInfo* ThumbnailInfo;
#endif // WITH_EDITORONLY_DATA

	/** Persistent level containing the world info, default brush and actors spawned during gameplay among other things */
	UPROPERTY(Transient)
	class ULevel*                                                    PersistentLevel;
```

```
/** The NAME_GameNetDriver game connection(s) for client/server communication */
UPROPERTY(Transient)
class UNetDriver*                                   NetDriver;

/** Line Batchers. All lines to be drawn in the world. */
UPROPERTY(Transient)
class ULineBatchComponent*                          LineBatcher;

/** Persistent Line Batchers. They don't get flushed every frame.  */
UPROPERTY(Transient)
class ULineBatchComponent*                          PersistentLineBatcher;

/** Foreground Line Batchers. This can't be Persistent.  */
UPROPERTY(Transient)
class ULineBatchComponent*                          ForegroundLineBatcher;

/** Instance of this world's game-specific networking management */
UPROPERTY(Transient)
class AGameNetworkManager*                          NetworkManager;

/** Instance of this world's game-specific physics collision handler */
UPROPERTY(Transient)
class UPhysicsCollisionHandler*                     PhysicsCollisionHandler;

/** Array of any additional objects that need to be referenced by this world, to make sure they
aren't GC'd */
UPROPERTY(Transient)
TArray<UObject*>
ExtraReferencedObjects;
```

```
/**
 * External modules can have additional data associated with this UWorld.
 * This is a list of per module world data objects. These aren't
 * loaded/saved by default.
 */
UPROPERTY(Transient)
TArray<UObject*>                                PerModuleDataObjects;

/** Level collection. ULevels are referenced by FName (Package name) to avoid serialized
references. Also contains offsets in world units */
UPROPERTY(Transient)
TArray<class ULevelStreaming*>                  StreamingLevels;

/** Prefix we used to rename streaming levels, non empty in PIE and standalone preview */
UPROPERTY()
FString
StreamingLevelsPrefix;

/** Pointer to the current level in the queue to be made visible, NULL if none are pending.
                        */
UPROPERTY(Transient)
class ULevel*
CurrentLevelPendingVisibility;

/** Pointer to the current level in the queue to be made invisible, NULL if none are pending.
                        */
UPROPERTY(Transient)
class ULevel*
CurrentLevelPendingInvisibility;
```

```
        /** Fake NetDriver for capturing network traffic to record demos
                                                                        */
        UPROPERTY()
        class UDemoNetDriver*                                   DemoNetDriver;


        /** Particle event manager **/
        UPROPERTY()
        class AParticleEventManager*                        MyParticleEventManager;


private:
        /** DefaultPhysicsVolume used for whole game **/
        UPROPERTY()
        APhysicsVolume*
        DefaultPhysicsVolume;


public:


        /** View locations rendered in the previous frame, if any. */
        TArray<FVector>
        ViewLocationsRenderedLastFrame;


        /** set for one tick after completely loading and initializing a new world
         * (regardless of whether it's LoadMap() or seamless travel)
         */
        uint32 bWorldWasLoadedThisTick:1;


        /**
         * Triggers a call to PostLoadMap() the next Tick, turns off loading movie if LoadMap() has been
called.
         */
```

```
        uint32 bTriggerPostLoadMap:1;


private:
        /** The world's navmesh */
        UPROPERTY(Transient)
        class UNavigationSystem*                                NavigationSystem;

        /** The current GameMode, valid only on the server */
        UPROPERTY(Transient)
        class AGameModeBase*                                    AuthorityGameMode;

        /** The replicated actor which contains game state information that can be accessible to clients.
Direct access is not allowed, use GetGameState<>() */
        UPROPERTY(Transient)
        class AGameStateBase*                                   GameState;

        /** The AI System handles generating pathing information and AI behavior */
        UPROPERTY(Transient)
        class UAISystemBase*                                    AISystem;

        /** RVO avoidance manager used by game */
        UPROPERTY(Transient)
        class UAvoidanceManager*                                AvoidanceManager;

        /** Array of levels currently in this world. Not serialized to disk to avoid hard references.
                                                                */
        UPROPERTY(Transient)
        TArray<class ULevel*>                                   Levels;
```

```
	/** Array of level collections currently in this world. */
	UPROPERTY(Transient, NonTransactional)
	TArray<FLevelCollection>  LevelCollections;

	/** Index of the level collection that's currently ticking. */
	int32
	ActiveLevelCollectionIndex;

	/** Creates the dynamic source and static level collections if they don't already exist. */
	void ConditionallyCreateDefaultLevelCollections();

public:

#if WITH_EDITOR
	/** Hierarchical LOD System. Used when WorldSetting.bEnableHierarchicalLODSystem is true */
	struct FHierarchicalLODBuilder*  HierarchicalLODBuilder;
#endif // WITH_EDITOR

private:

	/** Pointer to the current level being edited. Level has to be in the Levels array and ==
PersistentLevel in the game.  */
	UPROPERTY(Transient)
	class ULevel*  CurrentLevel;

	UPROPERTY(Transient)
	class UGameInstance*  OwningGameInstance;

	/** Parameter collection instances that hold parameter overrides for this world. */
```

```
	UPROPERTY(Transient)
	TArray<class UMaterialParameterCollectionInstance*> ParameterCollectionInstances;

	/**
	 * Canvas object used for drawing to render targets from blueprint functions eg DrawMaterialToRenderTarget.
	 * This is cached as UCanvas creation takes >100ms.
	 */
	UPROPERTY(Transient)
	UCanvas* CanvasForRenderingToTarget;

	UPROPERTY(Transient)
	UCanvas* CanvasForDrawMaterialToRenderTarget;

public:
	/** Set the pointer to the Navgation system. */
	void SetNavigationSystem( UNavigationSystem* InNavigationSystem);

	/** The interface to the scene manager for this world. */
	class FSceneInterface*	Scene;

	/** The current renderer feature level of this world */
	ERHIFeatureLevel::Type	FeatureLevel;

#if WITH_EDITORONLY_DATA
	/** Saved editor viewport states - one for each view type. Indexed using ELevelViewportType from UnrealEdTypes.h.  */
	UPROPERTY(NonTransactional)
	TArray<FLevelViewportInfo>	EditorViews;
```

```
#endif

	/**
	 * Set the CurrentLevel for this world.
	 * @return true if the current level changed.
	 */
	bool SetCurrentLevel( class ULevel* InLevel );

	/** Get the CurrentLevel for this world. **/
	class ULevel* GetCurrentLevel() const;

	/** A static map that is populated before loading a world from a package. This is so UWorld can look up its WorldType in ::PostLoad */
	static TMap<FName, EWorldType::Type> WorldTypePreLoadMap;

	/** Map of blueprints that are being debugged and the object instance they are debugging. */
	typedef TMap<TWeakObjectPtr<class UBlueprint>, TWeakObjectPtr<UObject> > FBlueprintToDebuggedObjectMap;

	/** Return the array of objects currently bieng debugged. */
	const FBlueprintToDebuggedObjectMap& GetBlueprintObjectsBeingDebugged() const{ return BlueprintObjectsBeingDebugged; };

	/** Creates a new FX system for this world */
	void CreateFXSystem();

#if WITH_EDITOR

	/** Change the feature level that this world is current rendering with */
```

```
        void ChangeFeatureLevel(ERHIFeatureLevel::Type InFeatureLevel, bool
bShowSlowProgressDialog = true);

        void RecreateScene(ERHIFeatureLevel::Type InFeatureLevel);

#endif // WITH_EDITOR

        /**
         * Sets whether or not this world is ticked by the engine, but use it at your own risk! This could
         * have unintended consequences if used carelessly. That said, for worlds that are not
interactive
         * and not rendering, it can save the cost of ticking them. This should probably never be used
         * for a primary game world.
         */
        void SetShouldTick(const bool bInShouldTick) { bShouldTick = bInShouldTick; }

        /** Returns whether or not this world is currently ticking. See SetShouldTick. */
        bool ShouldTick() const { return bShouldTick; }

private:

        /** List of all the controllers in the world. */
        TArray<TWeakObjectPtr<class AController> > ControllerList;

        /** List of all the player controllers in the world. */
        TArray<TWeakObjectPtr<class APlayerController> > PlayerControllerList;

        /** List of all the pawns in the world. */
        TArray<TWeakObjectPtr<class APawn> > PawnList;
```

```
	/** List of all the cameras in the world that auto-activate for players. */
	TArray<TWeakObjectPtr<ACameraActor> > AutoCameraActorList;

	/** List of all physics volumes in the world. Does not include the DefaultPhysicsVolume. */
	TArray<TWeakObjectPtr<APhysicsVolume> > NonDefaultPhysicsVolumeList;

	/** Physics scene for this world. */
	FPhysScene*                                                    PhysicsScene;

	/** Set of components that need updates at the end of the frame */
	TSet<TWeakObjectPtr<UActorComponent> > ComponentsThatNeedEndOfFrameUpdate;

	/** Set of components that need recreates at the end of the frame */
	TSet<TWeakObjectPtr<UActorComponent> >
ComponentsThatNeedEndOfFrameUpdate_OnGameThread;

	/** The state of async tracing - abstracted into its own object for easier reference */
	FWorldAsyncTraceState AsyncTraceState;

	/**	Objects currently being debugged in Kismet	*/
	FBlueprintToDebuggedObjectMap BlueprintObjectsBeingDebugged;

	/** Whether the render scene for this World should be created with HitProxies or not */
	bool bRequiresHitProxies;

	/** Whether to do any ticking at all for this world. */
	bool bShouldTick;
```

```
/** a delegate that broadcasts a notification whenever an actor is spawned */
FOnActorSpawned OnActorSpawned;

/** Reset Async Trace Buffer **/
void ResetAsyncTrace();

/** Wait for all Async Trace Buffer to be done **/
void WaitForAllAsyncTraceTasks();

/** Finish Async Trace Buffer **/
void FinishAsyncTrace();

/** Utility function that is used to ensure that a World has the correct WorldSettings */
void RepairWorldSettings();

/** Gameplay timers. */
class FTimerManager* TimerManager;

/** Latent action manager. */
struct FLatentActionManager LatentActionManager;

/** Whether we have a pending call to BuildStreamingData(). */
uint32 bStreamingDataDirty:1;

/** Timestamp (in FPlatformTime::Seconds) when the next call to BuildStreamingData() should be made, if bDirtyStreamingData is true. */
double BuildStreamingDataTimer;

DECLARE_EVENT_OneParam(UWorld, FOnNetTickEvent, float);
```

```
DECLARE_EVENT(UWorld, FOnTickFlushEvent);
/** Event to gather up all net drivers and call TickDispatch at once */
FOnNetTickEvent TickDispatchEvent;

/** Event to gather up all net drivers and call TickFlush at once */
FOnNetTickEvent TickFlushEvent;

/** Event to gather up all net drivers and call PostTickFlush at once */
FOnTickFlushEvent PostTickFlushEvent;

/** All registered net drivers TickDispatch() */
void BroadcastTickDispatch(float DeltaTime)
{
	TickDispatchEvent.Broadcast(DeltaTime);
}
/** All registered net drivers TickFlush() */
void BroadcastTickFlush(float DeltaTime)
{
	TickFlushEvent.Broadcast(DeltaTime);
}
/** All registered net drivers PostTickFlush() */
void BroadcastPostTickFlush(float DeltaTime)
{
	PostTickFlushEvent.Broadcast();
}

/** Called when the number of levels changes. */
DECLARE_EVENT(UWorld, FOnLevelsChangedEvent);
```

```
        /** Broadcasts whenever the number of levels changes */
        FOnLevelsChangedEvent LevelsChangedEvent;

#if WITH_EDITOR

        /** Broadcasts that selected levels have changed. */
        void BroadcastSelectedLevelsChanged();

#endif // WITH_EDITOR

#if WITH_EDITORONLY_DATA
        /** Called when selected level list changes. */
        DECLARE_EVENT( UWorld, FOnSelectedLevelsChangedEvent);

        /** Broadcasts whenever selected level list changes. */
        FOnSelectedLevelsChangedEvent                   SelectedLevelsChangedEvent;

        /** Array of selected levels currently in this world. Not serialized to disk to avoid hard
references.     */
        UPROPERTY(Transient)
        TArray<class ULevel*>                           SelectedLevels;

        /** Disables the broadcasting of level selection change. Internal use only. */
        uint32 bBroadcastSelectionChange:1;
#endif //WITH_EDITORONLY_DATA
public:
        /** The URL that was used when loading this World.
        */
        FURL                                                    URL;
```

```
/** Interface to the FX system managing particles and related effects for this world.
                                                                    */
class FFXSystemInterface*                                   FXSystem;

/** Data structures for holding the tick functions that are associated with the world (line
batcher, etc) **/
class FTickTaskLevel*                                       TickTaskLevel;

/** Whether we are in the middle of ticking actors/components or not
                                                                        */
bool                                                                bInTick;

/** Whether we have already built the collision tree or not                                 */
bool                        bIsBuilt;

/** We are in the middle of actor ticking, so add tasks for newly spawned actors
                                                                    */
bool
bTickNewlySpawned;

/** The current ticking group

        */
ETickingGroup                                               TickGroup;

/** Tick function for starting physics

*/
FStartPhysicsTickFunction StartPhysicsTickFunction;
```

```
	/** Tick function for ending physics

	*/

	FEndPhysicsTickFunction EndPhysicsTickFunction;


	/** Tick function for starting cloth simulation

	*/

	FStartAsyncSimulationFunction StartAsyncTickFunction;


	/**

	 * Indicates that during world ticking we are doing the final component update of dirty components

	 * (after PostAsyncWork and effect physics scene has run.

	 */

	bool
	bPostTickComponentUpdate;


	/** Counter for allocating game- unique controller player numbers
	                                                                */

	int32                                                           PlayerNum;


	/** Whether world object has been initialized via Init()
	                                                                */

	bool
	bIsWorldInitialized;


	/** Number of frames to delay Streaming Volume updating, useful if you preload a bunch of levels but the camera hasn't caught up yet (INDEX_NONE for infinite) */

	int32
	StreamingVolumeUpdateDelay;
```

```
	/** Is level streaming currently frozen?

	*/

	bool
	bIsLevelStreamingFrozen;


	/** Is forcibly unloading streaming levels?

	*/

	bool
	bShouldForceUnloadStreamingLevels;


	/** Is forcibly making streaming levels visible?
                                                                                */

	bool
	bShouldForceVisibleStreamingLevels;


	/** True we want to execute a call to UpdateCulledTriggerVolumes during Tick */

	bool
	bDoDelayedUpdateCullDistanceVolumes;


	/** The type of world this is. Describes the context in which it is being used (Editor, Game,
Preview etc.) */

	EWorldType::Type                                                WorldType;


	/** Force UsesGameHiddenFlags to return true. */

	DEPRECATED(4.14, "bHack_Force_UsesGameHiddenFlags_True is deprecated. Please use
EWorldType::GamePreview (etc.) to enforce correct hidden flag usage for preview scenes.")

	bool
	bHack_Force_UsesGameHiddenFlags_True;


	/** If true this world is in the process of running the construction script for an actor */
```

```
        bool
        bIsRunningConstructionScript;


        /** If true this world will tick physics to simulate. This isn't same as having Physics Scene.
         *  You need Physics Scene if you'd like to trace. This flag changed ticking */
        bool
        bShouldSimulatePhysics;


#if !UE_BUILD_SHIPPING
        /** If TRUE, 'hidden' components will still create render proxy, so can draw info (see
USceneComponent::ShouldRender) */
        bool
        bCreateRenderStateForHiddenComponents;
#endif // !UE_BUILD_SHIPPING


#if WITH_EDITOR
        /** this is special flag to enable collision by default for components that are not Volume
         * currently only used by editor level viewport world, and do not use this for in-game scene
         */
        bool
        bEnableTraceCollision;
#endif


        /** When non-'None', all line traces where the TraceTag match this will be drawn */
        FName    DebugDrawTraceTag;


        /** When set to true, all scene queries will be drawn */
        bool bDebugDrawAllTraceTags;


#if !(UE_BUILD_SHIPPING || UE_BUILD_TEST)
```

```
	bool DebugDrawSceneQueries(const FName& UsedTraceTag) const
	{
		return (bDebugDrawAllTraceTags || ((DebugDrawTraceTag != NAME_None) &&
(DebugDrawTraceTag == UsedTraceTag))) && IsInGameThread();
	}
#endif

	/** An array of post processing volumes, sorted in ascending order of priority.
		*/
	TArray< IInterface_PostProcessVolume * > PostProcessVolumes;

	/** Set of AudioVolumes sorted by  */
	// TODO: Make this be property UPROPERTY(Transient)
	TSet<class AAudioVolume*> AudioVolumes;

	/** Handle to the active audio device for this world. */
	uint32 AudioDeviceHandle;

	/** Time in FPlatformTime::Seconds unbuilt time was last encountered. 0 means not yet.
		*/
	double LastTimeUnbuiltLightingWasEncountered;

	/**  Time in seconds since level began play, but IS paused when the game is paused, and IS
dilated/clamped. */
	float TimeSeconds;

	/**  Time in seconds since level began play, but IS NOT paused when the game is paused, and IS
dilated/clamped. */
	float UnpausedTimeSeconds;
```

```
/** Time in seconds since level began play, but IS NOT paused when the game is paused, and IS NOT dilated/clamped. */
float RealTimeSeconds;

/** Time in seconds since level began play, but IS paused when the game is paused, and IS NOT dilated/clamped. */
float AudioTimeSeconds;

/** Frame delta time in seconds adjusted by e.g. time dilation. */
float DeltaTimeSeconds;

/** time at which to start pause **/
float PauseDelay;

/** Current location of this world origin */
FIntVector OriginLocation;

/** Requested new world origin location */
FIntVector RequestedOriginLocation;

/** World origin offset value. Non-zero only for a single frame when origin is rebased */
FVector OriginOffsetThisFrame;

/** All levels information from which our world is composed */
UPROPERTY()
class UWorldComposition* WorldComposition;

/** Whether we flushing level streaming state */
EFlushLevelStreamingType FlushLevelStreamingType;
```

```
public:
	/** The type of travel to perform next when doing a server travel */
	ETravelType				NextTravelType;

	/** The URL to be used for the upcoming server travel */
	FString NextURL;

	/** Amount of time to wait before traveling to next map, gives clients time to receive final RPCs @see ServerTravelPause */
	float NextSwitchCountdown;

	/** array of levels that were loaded into this map via PrepareMapChange() / CommitMapChange() (to inform newly joining clients) */
	TArray<FName> PreparingLevelNames;

	/** Name of persistent level if we've loaded levels via CommitMapChange() that aren't normally in the StreamingLevels array (to inform newly joining clients) */
	FName CommittedPersistentLevelName;

	/**
	 * This is a int on the level which is set when a light that needs to have lighting rebuilt
	 * is moved.  This is then checked in CheckMap for errors to let you know that this level should
	 * have lighting rebuilt.
	 **/
	uint32 NumLightingUnbuiltObjects;

	uint32 NumUnbuiltReflectionCaptures;

	/** Num of components missing valid texture streaming data. Updated in map check. */
```

```
int32 NumTextureStreamingUnbuiltComponents;

/** Num of resources that have changed since the last texture streaming build. Updated in map check. */
int32 NumTextureStreamingDirtyResources;

/** frame rate is below DesiredFrameRate, so drop high detail actors */
uint32 bDropDetail:1;

/** frame rate is well below DesiredFrameRate, so make LOD more aggressive */
uint32 bAggressiveLOD:1;

/** That map is default map or not **/
uint32 bIsDefaultLevel:1;

/** Whether it was requested that the engine bring up a loading screen and block on async loading. */
uint32 bRequestedBlockOnAsyncLoading:1;

/** Whether actors have been initialized for play */
uint32 bActorsInitialized:1;

/** Whether BeginPlay has been called on actors */
uint32 bBegunPlay:1;

/** Whether the match has been started */
uint32 bMatchStarted:1;

/**  When ticking the world, only update players. */
```

```
uint32 bPlayersOnly:1;

/** Indicates that at the end the frame bPlayersOnly will be set to true. */
uint32 bPlayersOnlyPending:1;

/** Is the world in its actor initialization phase. */
uint32 bStartup:1;

/** Is the world being torn down */
uint32 bIsTearingDown:1;

/**
 * This is a bool that indicates that one or more blueprints in the level (blueprint instances, level script, etc)
 * have compile errors that could not be automatically resolved.
 */
uint32 bKismetScriptError:1;

// Kismet debugging flags - they can be only editor only, but they're uint32, so it doens't make much difference
uint32 bDebugPauseExecution:1;

/** When set, camera is potentially moveable even when paused */
uint32 bIsCameraMoveableWhenPaused:1;

/** Indicates this scene always allows audio playback. */
uint32 bAllowAudioPlayback:1;

/** When set, will tell us to pause simulation after one tick.  If a breakpoint is encountered before tick is complete we will stop there instead. */
```

```
        uint32 bDebugFrameStepExecution:1;

        /** Keeps track whether actors moved via PostEditMove and therefore constraint syncup should be performed. */
        UPROPERTY(transient)
        uint32 bAreConstraintsDirty:1;

#if WITH_EDITORONLY_DATA
        /** List of DDC async requests we need to wait on before we register components. Game thread only. */
        TArray<TSharedPtr<FAsyncPreRegisterDDCRequest>> AsyncPreRegisterDDCRequests;
#endif

        //Experimental: In game performance tracking.
        FWorldInGamePerformanceTrackers* PerfTrackers;

        /**
         * UWorld default constructor
         */
        UWorld(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

        // LINE TRACE

        /**
         *  Trace a ray against the world using a specific channel and return if a blocking hit is found.
         *  @param  Start         Start location of the ray
         *  @param  End           End location of the ray
         *  @param  TraceChannel    The 'channel' that this ray is in, used to determine which components to hit
         *  @param  Params        Additional parameters used for the trace
```

```
	*	@param	ResponseParamResponseContainer to be used for this trace
	*  @return TRUE if a blocking hit is found
	*/
	bool LineTraceTestByChannel(const FVector& Start,const FVector& End,ECollisionChannel TraceChannel, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam, const FCollisionResponseParams& ResponseParam = FCollisionResponseParams::DefaultResponseParam) const;


	/**
	*  Trace a ray against the world using object types and return if a blocking hit is found.
	*  @param  Start		Start location of the ray
	*  @param  End		End location of the ray
	*	@param	ObjectQueryParams	List of object types it's looking for
	*  @param  Params		Additional parameters used for the trace
	*  @return TRUE if any hit is found
	*/
	bool LineTraceTestByObjectType(const FVector& Start,const FVector& End,const FCollisionObjectQueryParams& ObjectQueryParams, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam) const;


	/**
	*  Trace a ray against the world using a specific profile and return if a blocking hit is found.
	*  @param  Start		Start location of the ray
	*  @param  End		End location of the ray
	*  @param  ProfileName	The 'profile' used to determine which components to hit
	*  @param  Params		Additional parameters used for the trace
	*  @return TRUE if a blocking hit is found
	*/
	bool LineTraceTestByProfile(const FVector& Start, const FVector& End, FName ProfileName, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam) const;
```

```
	/**
	 * Trace a ray against the world using a specific channel and return the first blocking hit
	 * @param OutHit		First blocking hit found
	 * @param Start		Start location of the ray
	 * @param End		End location of the ray
	 * @param TraceChannel	The 'channel' that this ray is in, used to determine which components to hit
	 * @param Params		Additional parameters used for the trace
	 *		@param		ResponseParamResponseContainer to be used for this trace
	 * @return TRUE if a blocking hit is found
	 */
	bool LineTraceSingleByChannel(struct FHitResult& OutHit,const FVector& Start,const FVector& End,ECollisionChannel TraceChannel,const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam, const FCollisionResponseParams& ResponseParam = FCollisionResponseParams::DefaultResponseParam) const;

	/**
	 * Trace a ray against the world using object types and return the first blocking hit
	 * @param OutHit		First blocking hit found
	 * @param Start		Start location of the ray
	 * @param End		End location of the ray
	 *		@param		ObjectQueryParams	List of object types it's looking for
	 * @param Params		Additional parameters used for the trace
	 * @return TRUE if any hit is found
	 */
	bool LineTraceSingleByObjectType(struct FHitResult& OutHit,const FVector& Start,const FVector& End,const FCollisionObjectQueryParams& ObjectQueryParams, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam) const;

	/**
```

```
	* Trace a ray against the world using a specific profile and return the first blocking hit
	* @param OutHit		First blocking hit found
	* @param Start		Start location of the ray
	* @param End		End location of the ray
	* @param ProfileName	The 'profile' used to determine which components to hit
	* @param Params		Additional parameters used for the trace
	* @return TRUE if a blocking hit is found
	*/
	bool LineTraceSingleByProfile(struct FHitResult& OutHit, const FVector& Start, const FVector& End, FName ProfileName, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam) const;


	/**
	* Trace a ray against the world using a specific channel and return overlapping hits and then first blocking hit
	* Results are sorted, so a blocking hit (if found) will be the last element of the array
	* Only the single closest blocking result will be generated, no tests will be done after that
	* @param OutHits		Array of hits found between ray and the world
	* @param Start		Start location of the ray
	* @param End		End location of the ray
	* @param TraceChannel	The 'channel' that this ray is in, used to determine which components to hit
	* @param Params		Additional parameters used for the trace
	*		@param		ResponseParamResponseContainer to be used for this trace
	* @return TRUE if OutHits contains any blocking hit entries
	*/
	bool LineTraceMultiByChannel(TArray<struct FHitResult>& OutHits,const FVector& Start,const FVector& End,ECollisionChannel TraceChannel,const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam, const FCollisionResponseParams& ResponseParam = FCollisionResponseParams::DefaultResponseParam) const;
```

```
/**
 * Trace a ray against the world using object types and return overlapping hits and then first blocking hit
 * Results are sorted, so a blocking hit (if found) will be the last element of the array
 * Only the single closest blocking result will be generated, no tests will be done after that
 * @param OutHits Array of hits found between ray and the world
 * @param Start Start location of the ray
 * @param End End location of the ray
 * @param ObjectQueryParams List of object types it's looking for
 * @param Params Additional parameters used for the trace
 * @return TRUE if any hit is found
 */
bool LineTraceMultiByObjectType(TArray<struct FHitResult>& OutHits,const FVector& Start,const FVector& End,const FCollisionObjectQueryParams& ObjectQueryParams, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam) const;

/**
 * Trace a ray against the world using a specific profile and return overlapping hits and then first blocking hit
 * Results are sorted, so a blocking hit (if found) will be the last element of the array
 * Only the single closest blocking result will be generated, no tests will be done after that
 * @param OutHits Array of hits found between ray and the world
 * @param Start Start location of the ray
 * @param End End location of the ray
 * @param ProfileName The 'profile' used to determine which components to hit
 * @param Params Additional parameters used for the trace
 * @return TRUE if OutHits contains any blocking hit entries
 */
```

```
bool LineTraceMultiByProfile(TArray<struct FHitResult>& OutHits, const FVector& Start, const FVector& End, FName ProfileName, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam) const;

/**
 * Sweep a shape against the world using a specific channel and return if a blocking hit is found.
 * @param Start Start location of the shape
 * @param End End location of the shape
 * @param TraceChannel The 'channel' that this trace uses, used to determine which components to hit
 * @param CollisionShape CollisionShape - supports Box, Sphere, Capsule
 * @param Params Additional parameters used for the trace
 * @param ResponseParamResponseContainer to be used for this trace
 * @return TRUE if a blocking hit is found
 */
bool SweepTestByChannel(const FVector& Start, const FVector& End, const FQuat& Rot, ECollisionChannel TraceChannel, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam, const FCollisionResponseParams& ResponseParam = FCollisionResponseParams::DefaultResponseParam) const;

/**
 * Sweep a shape against the world using object types and return if a blocking hit is found.
 * @param Start Start location of the shape
 * @param End End location of the shape
 * @param ObjectQueryParams List of object types it's looking for
 * @param CollisionShape CollisionShape - supports Box, Sphere, Capsule
 * @param Params Additional parameters used for the trace
 * @return TRUE if any hit is found
 */
```

bool SweepTestByObjectType(const FVector& Start, const FVector& End, const FQuat& Rot, const FCollisionObjectQueryParams& ObjectQueryParams, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam) const;

/**

* Sweep a shape against the world using a specific profile and return if a blocking hit is found.

* @param Start Start location of the shape

* @param End End location of the shape

* @param ProfileName The 'profile' used to determine which components to hit

* @param CollisionShape CollisionShape - supports Box, Sphere, Capsule

* @param Params Additional parameters used for the trace

* @return TRUE if a blocking hit is found

*/

bool SweepTestByProfile(const FVector& Start, const FVector& End, const FQuat& Rot, FName ProfileName, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params) const;

/**

* Sweep a shape against the world and return the first blocking hit using a specific channel

* @param OutHit First blocking hit found

* @param Start Start location of the shape

* @param End End location of the shape

* @param TraceChannel The 'channel' that this trace is in, used to determine which components to hit

* @param CollisionShape CollisionShape - supports Box, Sphere, Capsule

* @param Params Additional parameters used for the trace

* @param ResponseParamResponseContainer to be used for this trace

* @return TRUE if OutHits contains any blocking hit entries

*/

```
bool SweepSingleByChannel(struct FHitResult& OutHit, const FVector& Start, const FVector& End, const FQuat& Rot, ECollisionChannel TraceChannel, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam, const FCollisionResponseParams& ResponseParam = FCollisionResponseParams::DefaultResponseParam) const;

/**
 * Sweep a shape against the world and return the first blocking hit using object types
 * @param  OutHit        First blocking hit found
 * @param  Start         Start location of the shape
 * @param  End           End location of the shape
 *         @param        ObjectQueryParams      List of object types it's looking for
 * @param       CollisionShape   CollisionShape - supports Box, Sphere, Capsule
 * @param  Params        Additional parameters used for the trace
 * @return TRUE if any hit is found
 */
bool SweepSingleByObjectType(struct FHitResult& OutHit, const FVector& Start, const FVector& End, const FQuat& Rot, const FCollisionObjectQueryParams& ObjectQueryParams, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam) const;

/**
 * Sweep a shape against the world and return the first blocking hit using a specific profile
 * @param  OutHit        First blocking hit found
 * @param  Start         Start location of the shape
 * @param  End           End location of the shape
 * @param  ProfileName     The 'profile' used to determine which components to hit
 * @param       CollisionShape   CollisionShape - supports Box, Sphere, Capsule
 * @param  Params        Additional parameters used for the trace
 * @return TRUE if OutHits contains any blocking hit entries
 */
```

bool SweepSingleByProfile(struct FHitResult& OutHit, const FVector& Start, const FVector& End, const FQuat& Rot, FName ProfileName, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam) const;

/**

* Sweep a shape against the world and return all initial overlaps using a specific channel (including blocking) if requested, then overlapping hits and then first blocking hit

* Results are sorted, so a blocking hit (if found) will be the last element of the array

* Only the single closest blocking result will be generated, no tests will be done after that

* @param OutHits Array of hits found between ray and the world

* @param Start Start location of the shape

* @param End End location of the shape

* @param TraceChannel The 'channel' that this ray is in, used to determine which components to hit

* @param CollisionShape CollisionShape - supports Box, Sphere, Capsule

* @param Params Additional parameters used for the trace

* @param ResponseParamResponseContainer to be used for this trace

* @return TRUE if OutHits contains any blocking hit entries

*/

bool SweepMultiByChannel(TArray<struct FHitResult>& OutHits, const FVector& Start, const FVector& End, const FQuat& Rot, ECollisionChannel TraceChannel, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam, const FCollisionResponseParams& ResponseParam = FCollisionResponseParams::DefaultResponseParam) const;

/**

* Sweep a shape against the world and return all initial overlaps using object types (including blocking) if requested, then overlapping hits and then first blocking hit

* Results are sorted, so a blocking hit (if found) will be the last element of the array

* Only the single closest blocking result will be generated, no tests will be done after that

* @param OutHits Array of hits found between ray and the world

* @param Start Start location of the shape

```
	 * @param  End		End location of the shape
	 *		@param		ObjectQueryParams	List of object types it's looking for
	 * @param	CollisionShape	CollisionShape - supports Box, Sphere, Capsule
	 * @param  Params		Additional parameters used for the trace
	 * @return TRUE if any hit is found
	 */
	bool SweepMultiByObjectType(TArray<struct FHitResult>& OutHits, const FVector& Start, const FVector& End, const FQuat& Rot, const FCollisionObjectQueryParams& ObjectQueryParams, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam) const;

	/**
	 * Sweep a shape against the world and return all initial overlaps using a specific profile, then overlapping hits and then first blocking hit
	 * Results are sorted, so a blocking hit (if found) will be the last element of the array
	 * Only the single closest blocking result will be generated, no tests will be done after that
	 * @param  OutHits		Array of hits found between ray and the world
	 * @param  Start		Start location of the shape
	 * @param  End		End location of the shape
	 * @param  ProfileName	The 'profile' used to determine which components to hit
	 * @param	CollisionShape	CollisionShape - supports Box, Sphere, Capsule
	 * @param  Params		Additional parameters used for the trace
	 * @return TRUE if OutHits contains any blocking hit entries
	 */
	bool SweepMultiByProfile(TArray<FHitResult>& OutHits, const FVector& Start, const FVector& End, const FQuat& Rot, FName ProfileName, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam) const;

	/**
	 * Test the collision of a shape at the supplied location using a specific channel, and return if any blocking overlap is found
```

```
	* @param  Pos		Location of center of box to test against the world
	* @param  TraceChannel	The 'channel' that this query is in, used to determine which components to hit
	* @param  CollisionShape	CollisionShape - supports Box, Sphere, Capsule, Convex
	* @param  Params		Additional parameters used for the trace
	* @param  ResponseParam	ResponseContainer to be used for this trace
	* @return TRUE if any blocking results are found
	*/
	bool OverlapBlockingTestByChannel(const FVector& Pos, const FQuat& Rot, ECollisionChannel TraceChannel, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam, const FCollisionResponseParams& ResponseParam = FCollisionResponseParams::DefaultResponseParam) const;

	/**
	* Test the collision of a shape at the supplied location using a specific channel, and return if any blocking or overlapping shape is found
	* @param  Pos		Location of center of box to test against the world
	* @param  TraceChannel	The 'channel' that this query is in, used to determine which components to hit
	* @param  CollisionShape	CollisionShape - supports Box, Sphere, Capsule, Convex
	* @param  Params		Additional parameters used for the trace
	* @param  ResponseParam	ResponseContainer to be used for this trace
	* @return TRUE if any blocking or overlapping results are found
	*/
	bool OverlapAnyTestByChannel(const FVector& Pos, const FQuat& Rot, ECollisionChannel TraceChannel, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam, const FCollisionResponseParams& ResponseParam = FCollisionResponseParams::DefaultResponseParam) const;

	/**
	* Test the collision of a shape at the supplied location using object types, and return if any overlap is found
```

```
	* @param  Pos		Location of center of box to test against the world
	* @param  ObjectQueryParams		List of object types it's looking for
	* @param  CollisionShape	CollisionShape - supports Box, Sphere, Capsule, Convex
	* @param  Params	Additional parameters used for the trace
	* @return TRUE if any blocking results are found
	*/
	bool OverlapAnyTestByObjectType(const FVector& Pos, const FQuat& Rot, const FCollisionObjectQueryParams& ObjectQueryParams, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam) const;

	/**
	* Test the collision of a shape at the supplied location using a specific profile, and return if any blocking overlap is found
	* @param  Pos		Location of center of box to test against the world
	* @param  ProfileName	The 'profile' used to determine which components to hit
	* @param	CollisionShape	CollisionShape - supports Box, Sphere, Capsule
	* @param  Params	Additional parameters used for the trace
	* @return TRUE if any blocking results are found
	*/
	bool OverlapBlockingTestByProfile(const FVector& Pos, const FQuat& Rot, FName ProfileName, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam) const;

	/**
	* Test the collision of a shape at the supplied location using a specific profile, and return if any blocking or overlap is found
	* @param  Pos		Location of center of box to test against the world
	* @param  ProfileName	The 'profile' used to determine which components to hit
	* @param	CollisionShape	CollisionShape - supports Box, Sphere, Capsule
	* @param  Params	Additional parameters used for the trace
	* @return TRUE if any blocking or overlapping results are found
```

```
	*/
	bool OverlapAnyTestByProfile(const FVector& Pos, const FQuat& Rot, FName ProfileName, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam) const;



	/**
	 * Test the collision of a shape at the supplied location using a specific channel, and determine the set of components that it overlaps
	 * @param	OutOverlaps		Array of components found to overlap supplied box
	 * @param	Pos				Location of center of shape to test against the world
	 * @param	TraceChannel	The 'channel' that this query is in, used to determine which components to hit
	 * @param		CollisionShape	CollisionShape - supports Box, Sphere, Capsule
	 * @param	Params			Additional parameters used for the trace
	 *		@param			ResponseParamResponseContainer to be used for this trace
	 * @return TRUE if OutOverlaps contains any blocking results
	 */
	bool OverlapMultiByChannel(TArray<struct FOverlapResult>& OutOverlaps, const FVector& Pos, const FQuat& Rot, ECollisionChannel TraceChannel, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam, const FCollisionResponseParams& ResponseParam = FCollisionResponseParams::DefaultResponseParam) const;



	/**
	 * Test the collision of a shape at the supplied location using object types, and determine the set of components that it overlaps
	 * @param	OutOverlaps		Array of components found to overlap supplied box
	 * @param	Pos				Location of center of shape to test against the world
	 *		@param			ObjectQueryParams	List of object types it's looking for
	 * @param		CollisionShape	CollisionShape - supports Box, Sphere, Capsule
```

```
	 * @param  Params		Additional parameters used for the trace
	 * @return TRUE if any overlap is found
	 */
	bool OverlapMultiByObjectType(TArray<struct FOverlapResult>& OutOverlaps, const FVector& Pos, const FQuat& Rot, const FCollisionObjectQueryParams& ObjectQueryParams, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam) const;

	/**
	 * Test the collision of a shape at the supplied location using a specific profile, and determine the set of components that it overlaps
	 * @param  OutOverlaps	Array of components found to overlap supplied box
	 * @param  Pos			Location of center of shape to test against the world
	 * @param  ProfileName	The 'profile' used to determine which components to hit
	 * @param	CollisionShape	CollisionShape - supports Box, Sphere, Capsule
	 * @param  Params		Additional parameters used for the trace
	 * @return TRUE if OutOverlaps contains any blocking results
	 */
	bool OverlapMultiByProfile(TArray<struct FOverlapResult>& OutOverlaps, const FVector& Pos, const FQuat& Rot, FName ProfileName, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam) const;

	// COMPONENT SWEEP

	/**
	 * Sweep the geometry of the supplied component, and determine the set of components that it hits.
	 * @note The overload taking rotation as an FQuat is slightly faster than the version using FRotator (which will be converted to an FQuat)..
	 * @param  OutHits		Array of hits found between ray and the world
	 * @param  PrimComp		Component's geometry to test against the world. Transform of this component is ignored
```

```
	* @param Start		Start location of the trace

	* @param End		End location of the trace

	* @param Rot		Rotation of PrimComp geometry for test against the world (rotation remains constant over sweep)

	* @param Params		Additional parameters used for the trace

	* @return TRUE if OutHits contains any blocking hit entries

	*/

	bool ComponentSweepMulti(TArray<struct FHitResult>& OutHits, class UPrimitiveComponent* PrimComp, const FVector& Start, const FVector& End, const FQuat& Rot,	const FComponentQueryParams& Params) const;

	bool ComponentSweepMulti(TArray<struct FHitResult>& OutHits, class UPrimitiveComponent* PrimComp, const FVector& Start, const FVector& End, const FRotator& Rot, const FComponentQueryParams& Params) const;


	// COMPONENT OVERLAP


	/**

	* Test the collision of the supplied component at the supplied location/rotation using object types, and determine the set of components that it overlaps

	* @note The overload taking rotation as an FQuat is slightly faster than the version using FRotator (which will be converted to an FQuat)..

	* @param OutOverlaps	Array of overlaps found between component in specified pose and the world

	* @param PrimComp	Component's geometry to test against the world. Transform of this component is ignored

	* @param Pos		Location of PrimComp geometry for test against the world

	* @param Rot		Rotation of PrimComp geometry for test against the world

	*	@param		ObjectQueryParams	List of object types it's looking for. When this enters, we do object query with component shape

	* @return TRUE if any hit is found

	*/
```

bool ComponentOverlapMulti(TArray<struct FOverlapResult>& OutOverlaps, const class UPrimitiveComponent* PrimComp, const FVector& Pos, const FQuat& Rot, const FComponentQueryParams& Params = FComponentQueryParams::DefaultComponentQueryParams, const FCollisionObjectQueryParams& ObjectQueryParams=FCollisionObjectQueryParams::DefaultObjectQueryParam) const;

bool ComponentOverlapMulti(TArray<struct FOverlapResult>& OutOverlaps, const class UPrimitiveComponent* PrimComp, const FVector& Pos, const FRotator& Rot, const FComponentQueryParams& Params = FComponentQueryParams::DefaultComponentQueryParams, const FCollisionObjectQueryParams& ObjectQueryParams=FCollisionObjectQueryParams::DefaultObjectQueryParam) const;

/**

* Test the collision of the supplied component at the supplied location/rotation using a specific channel, and determine the set of components that it overlaps

* @note The overload taking rotation as an FQuat is slightly faster than the version using FRotator (which will be converted to an FQuat)..

* @param OutOverlaps Array of overlaps found between component in specified pose and the world

* @param PrimComp Component's geometry to test against the world. Transform of this component is ignored

* @param Pos Location of PrimComp geometry for test against the world

* @param Rot Rotation of PrimComp geometry for test against the world

* @param TraceChannel The 'channel' that this query is in, used to determine which components to hit

* @return TRUE if OutOverlaps contains any blocking results

*/

bool ComponentOverlapMultiByChannel(TArray<struct FOverlapResult>& OutOverlaps, const class UPrimitiveComponent* PrimComp, const FVector& Pos, const FQuat& Rot, ECollisionChannel TraceChannel, const FComponentQueryParams& Params = FComponentQueryParams::DefaultComponentQueryParams, const FCollisionObjectQueryParams& ObjectQueryParams=FCollisionObjectQueryParams::DefaultObjectQueryParam) const;

bool ComponentOverlapMultiByChannel(TArray<struct FOverlapResult>& OutOverlaps, const class UPrimitiveComponent* PrimComp, const FVector& Pos, const FRotator& Rot, ECollisionChannel TraceChannel, const FComponentQueryParams& Params = FComponentQueryParams::DefaultComponentQueryParams, const FCollisionObjectQueryParams& ObjectQueryParams=FCollisionObjectQueryParams::DefaultObjectQueryParam) const;

```
/**
 * Interface for Async. Pretty much same parameter set except you can optional set delegate to be called when execution is completed and you can set UserData if you'd like
 * if no delegate, you can query trace data using QueryTraceData or QueryOverlapData
 * the data is available only in the next frame after request is made - in other words, if request is made in frame X, you can get the result in frame (X+1)
 *
 *	@param	InTraceType		Indicates if you want multiple results, single result, or just yes/no (no hit information)
 * @param Start		Start location of the ray
 * @param End		End location of the ray
 * @param TraceChannel	The 'channel' that this ray is in, used to determine which components to hit
 * @param Params		Additional parameters used for the trace
 *	@param	ResponseParamResponseContainer to be used for this trace
 *	@param	InDeleagte		Delegate function to be called - to see example, search FTraceDelegate
 *	Example can be void MyActor::TraceDone(const FTraceHandle& TraceHandle, FTraceDatum & TraceData)
 *	Before sending to the function,
 *
 *	FTraceDelegate TraceDelegate;
 *	TraceDelegate.BindRaw(this, &MyActor::TraceDone);
 *
 *	@param	UserData		UserData
 */
FTraceHandle AsyncLineTraceByChannel(EAsyncTraceType InTraceType, const FVector& Start,const FVector& End, ECollisionChannel TraceChannel, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam, const FCollisionResponseParams& ResponseParam =
```

```
FCollisionResponseParams::DefaultResponseParam, FTraceDelegate * InDelegate=NULL, uint32
UserData = 0 );


	DEPRECATED(4.12, "bMultiTrace option replaced with required EAsyncTraceType enum.")

	FTraceHandle	AsyncLineTraceByChannel(const FVector& Start, const FVector& End,
ECollisionChannel TraceChannel, const FCollisionQueryParams& Params =
FCollisionQueryParams::DefaultQueryParam, const FCollisionResponseParams& ResponseParam =
FCollisionResponseParams::DefaultResponseParam, FTraceDelegate * InDelegate = NULL, uint32
UserData = 0, bool bMultiTrace = false)

	{

		return AsyncLineTraceByChannel(bMultiTrace ? EAsyncTraceType::Multi :
EAsyncTraceType::Single, Start, End, TraceChannel, Params, ResponseParam, InDelegate, UserData);

	}


	/**

	 * Interface for Async. Pretty much same parameter set except you can optional set delegate to
be called when execution is completed and you can set UserData if you'd like

	 * if no delegate, you can query trace data using QueryTraceData or QueryOverlapData

	 * the data is available only in the next frame after request is made - in other words, if request is
made in frame X, you can get the result in frame (X+1)

	 *

	 *	@param	InTraceType		Indicates if you want multiple results, single hit
result, or just yes/no (no hit information)

	 * @param Start		Start location of the ray

	 * @param End		End location of the ray

	 *	@param	ObjectQueryParams	List of object types it's looking for

	 * @param Params		Additional parameters used for the trace

	 *	@param	InDeleagte		Delegate function to be called - to see example,
search FTraceDelegate

	 *		Example can be void
MyActor::TraceDone(const FTraceHandle& TraceHandle, FTraceDatum & TraceData)

	 *		Before sending to the function,
```

```
	*
	*                                                   FTraceDelegate TraceDelegate;
	*                                                   TraceDelegate.BindRaw(this,
&MyActor::TraceDone);
	*
	*	@param	UserData		UserData
	*/
	FTraceHandle	AsyncLineTraceByObjectType(EAsyncTraceType InTraceType, const FVector&
Start,const FVector& End, const FCollisionObjectQueryParams& ObjectQueryParams, const
FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam, FTraceDelegate *
InDelegate=NULL, uint32 UserData = 0 );


	DEPRECATED(4.12, "bMultiTrace option replaced with required EAsyncTraceType enum.")
	FTraceHandle	AsyncLineTraceByObjectType(const FVector& Start,const FVector& End, const
FCollisionObjectQueryParams& ObjectQueryParams, const FCollisionQueryParams& Params =
FCollisionQueryParams::DefaultQueryParam, FTraceDelegate * InDelegate=NULL, uint32 UserData = 0,
bool bMultiTrace=false )
	{
		return AsyncLineTraceByObjectType(bMultiTrace ? EAsyncTraceType::Multi :
EAsyncTraceType::Single, Start, End, ObjectQueryParams, Params, InDelegate, UserData);
	}



	/**
	 * Interface for Async trace
	 * Pretty much same parameter set except you can optional set delegate to be called when
execution is completed and you can set UserData if you'd like
	 * if no delegate, you can query trace data using QueryTraceData or QueryOverlapData
	 * the data is available only in the next frame after request is made - in other words, if request is
made in frame X, you can get the result in frame (X+1)
	*
```

```
	* @param InTraceType Indicates if you want multiple results, single hit result, or just yes/no (no hit information)
	* @param Start Start location of the shape
	* @param End End location of the shape
	* @param TraceChannel The 'channel' that this trace is in, used to determine which components to hit
	* @param CollisionShape CollisionShape - supports Box, Sphere, Capsule
	* @param Params Additional parameters used for the trace
	* @param ResponseParamResponseContainer to be used for this trace
	* @param InDeleagte Delegate function to be called - to see example, search FTraceDelegate
	*                   Example can be void MyActor::TraceDone(const FTraceHandle& TraceHandle, FTraceDatum & TraceData)
	*                   Before sending to the function,
	*
	*                   FTraceDelegate TraceDelegate;
	*                   TraceDelegate.BindRaw(this, &MyActor::TraceDone);
	*
	* @param UserData UserData
	*/
	FTraceHandle AsyncSweepByChannel(EAsyncTraceType InTraceType, const FVector& Start, const FVector& End, ECollisionChannel TraceChannel, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam, const FCollisionResponseParams& ResponseParam = FCollisionResponseParams::DefaultResponseParam, FTraceDelegate * InDelegate = NULL, uint32 UserData = 0);


	DEPRECATED(4.12, "bMultiTrace option replaced with required ETraceDatumType enum.")
	FTraceHandle AsyncSweepByChannel(const FVector& Start, const FVector& End, ECollisionChannel TraceChannel, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam, const FCollisionResponseParams& ResponseParam = FCollisionResponseParams::DefaultResponseParam, FTraceDelegate * InDelegate = NULL, uint32 UserData = 0, bool bMultiTrace = false)
```

```
	{
		return AsyncSweepByChannel(bMultiTrace ? EAsyncTraceType::Multi : EAsyncTraceType::Single, Start, End, TraceChannel, CollisionShape, Params, ResponseParam, InDelegate, UserData);
	}

	/**
	 * Interface for Async trace
	 * Pretty much same parameter set except you can optional set delegate to be called when execution is completed and you can set UserData if you'd like
	 * if no delegate, you can query trace data using QueryTraceData or QueryOverlapData
	 * the data is available only in the next frame after request is made - in other words, if request is made in frame X, you can get the result in frame (X+1)
	 *
	 *	@param	InTraceType		Indicates if you want multiple results, single hit result, or just yes/no (no hit information)
	 *  @param  Start		Start location of the shape
	 *  @param  End		End location of the shape
	 *	@param	ObjectQueryParams	List of object types it's looking for
	 *  @param	CollisionShape		CollisionShape - supports Box, Sphere, Capsule
	 *  @param  Params		Additional parameters used for the trace
	 *	@param	InDeleagte		Delegate function to be called - to see example, search FTraceDelegate
	 *							Example can be void MyActor::TraceDone(const FTraceHandle& TraceHandle, FTraceDatum & TraceData)
	 *							Before sending to the function,
	 *
	 *							FTraceDelegate TraceDelegate;
	 *							TraceDelegate.BindRaw(this, &MyActor::TraceDone);
	 *
```

```
	*	@param	UserData	UserData
	*/
	FTraceHandle	AsyncSweepByObjectType(EAsyncTraceType InTraceType, const FVector& Start, const FVector& End, const FCollisionObjectQueryParams& ObjectQueryParams, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam, FTraceDelegate * InDelegate = NULL, uint32 UserData = 0);

	DEPRECATED(4.12, "bMultiTrace option replaced with required ETraceDatumType enum.")
	FTraceHandle	AsyncSweepByObjectType(const FVector& Start, const FVector& End, const FCollisionObjectQueryParams& ObjectQueryParams, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam, FTraceDelegate * InDelegate = NULL, uint32 UserData = 0, bool bMultiTrace = false)
	{
		return AsyncSweepByObjectType(bMultiTrace ? EAsyncTraceType::Multi : EAsyncTraceType::Single, Start, End, ObjectQueryParams, CollisionShape, Params, InDelegate, UserData);
	}

	// overlap functions

	/**
	 * Interface for Async trace
	 * Pretty much same parameter set except you can optional set delegate to be called when execution is completed and you can set UserData if you'd like
	 * if no delegate, you can query trace data using QueryTraceData or QueryOverlapData
	 * the data is available only in the next frame after request is made - in other words, if request is made in frame X, you can get the result in frame (X+1)
	 *
	 *  @param  Pos		Location of center of shape to test against the world
	*	@param	bMultiTrace		true if you'd like to do multi trace, or false otherwise
```

```
	* @param  TraceChannel    The 'channel' that this query is in, used to determine which components to hit
	* @param	CollisionShape		CollisionShape - supports Box, Sphere, Capsule
	* @param  Params		Additional parameters used for the trace
	*		@param		ResponseParamResponseContainer to be used for this trace
	*		@param		InDeleagte			Delegate function to be called - to see example, search FTraceDelegate
	*                                                       Example can be void MyActor::TraceDone(const FTraceHandle& TraceHandle, FTraceDatum & TraceData)
	*                                                       Before sending to the function,
	*
	*                                                       FTraceDelegate TraceDelegate;
	*                                                       TraceDelegate.BindRaw(this, &MyActor::TraceDone);
	*
	*		@param UserData				UserData
	*/
	FTraceHandle	AsyncOverlapByChannel(const FVector& Pos, const FQuat& Rot, ECollisionChannel TraceChannel, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam, const FCollisionResponseParams& ResponseParam = FCollisionResponseParams::DefaultResponseParam, FOverlapDelegate * InDelegate = NULL, uint32 UserData = 0);


	/**
	* Interface for Async trace
	* Pretty much same parameter set except you can optional set delegate to be called when execution is completed and you can set UserData if you'd like
	* if no delegate, you can query trace data using QueryTraceData or QueryOverlapData
	* the data is available only in the next frame after request is made - in other words, if request is made in frame X, you can get the result in frame (X+1)
	*
	* @param  Pos		Location of center of shape to test against the world
```

```
	*		@param		ObjectQueryParams	List of object types it's looking for
	*	@param	CollisionShape		CollisionShape - supports Box, Sphere, Capsule
	*	@param	Params		Additional parameters used for the trace
	*		@param		InDeleagte			Delegate function to be called - to see example, search FTraceDelegate
	*											Example can be void MyActor::TraceDone(const FTraceHandle& TraceHandle, FTraceDatum & TraceData)
	*											Before sending to the function,
	*
	*											FTraceDelegate TraceDelegate;
	*											TraceDelegate.BindRaw(this, &MyActor::TraceDone);
	*
	*		@param UserData				UserData
	*/
	FTraceHandle	AsyncOverlapByObjectType(const FVector& Pos, const FQuat& Rot, const FCollisionObjectQueryParams& ObjectQueryParams, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam, FOverlapDelegate * InDelegate = NULL, uint32 UserData = 0);


	/**
	 * Query function
	 * return true if already done and returning valid result - can be hit or no hit
	 * return false if either expired or not yet evaluated or invalid
	 * Use IsTraceHandleValid to find out if valid and to be evaluated
	 */
	bool QueryTraceData(const FTraceHandle& Handle, FTraceDatum& OutData);


	/**
	 * Query function
	 * return true if already done and returning valid result - can be hit or no hit
```

```
	 * return false if either expired or not yet evaluated or invalid
	 * Use IsTraceHandleValid to find out if valid and to be evaluated
	 */
	bool QueryOverlapData(const FTraceHandle& Handle, FOverlapDatum& OutData);
	/**
	 * See if TraceHandle is still valid or not
	 *
	 * @param	Handle			TraceHandle that was returned when request Trace
	 * @param	bOverlapTrace	true if this is overlap test Handle, not trace test handle
	 *
	 * return true if it will be evaluated OR it has valid result
	 * return false if it already has expired Or not valid
	 */
	bool IsTraceHandleValid(const FTraceHandle& Handle, bool bOverlapTrace);

	/** NavigationSystem getter */
	FORCEINLINE UNavigationSystem* GetNavigationSystem() { return NavigationSystem; }
	/** NavigationSystem const getter */
	FORCEINLINE const UNavigationSystem* GetNavigationSystem() const { return
NavigationSystem; }

	/** AISystem getter. if AISystem is missing it tries to create one and returns the result.
	 *	@NOTE the result can be NULL, for example on client games or if no AI module or
AISystem class have not been specified
	 *	@see UAISystemBase::AISystemClassName and
UAISystemBase::AISystemModuleName*/
	UAISystemBase* CreateAISystem();

	/** AISystem getter */
	FORCEINLINE UAISystemBase* GetAISystem() { return AISystem; }
```

```
	/** AISystem const getter */
	FORCEINLINE const UAISystemBase* GetAISystem() const { return AISystem; }

	/** Avoidance manager getter */
	FORCEINLINE class UAvoidanceManager* GetAvoidanceManager() { return AvoidanceManager; }
	/** Avoidance manager getter */
	FORCEINLINE const class UAvoidanceManager* GetAvoidanceManager() const { return
AvoidanceManager; }

	/** Returns an iterator for the controller list. */
	FConstControllerIterator GetControllerIterator() const;

	/** @return Returns an iterator for the pawn list. */
	FConstPawnIterator GetPawnIterator() const;

	/** @return Returns the number of Pawns. */
	int32 GetNumPawns() const;

	/** @return Returns an iterator for the player controller list. */
	FConstPlayerControllerIterator GetPlayerControllerIterator() const;

	/** @return Returns the first player controller, or NULL if there is not one. */
	APlayerController* GetFirstPlayerController() const;

	/*
	 *	Get the first valid local player via the first player controller.
	 *
	 *  @return Pointer to the first valid ULocalPlayer, or NULL if there is not one.
	 */
```

```
        ULocalPlayer* GetFirstLocalPlayerFromController() const;

        /** Register a CameraActor that auto-activates for a PlayerController. */
        void RegisterAutoActivateCamera(ACameraActor* CameraActor, int32 PlayerIndex);

        /** Get an iterator for the list of CameraActors that auto-activate for PlayerControllers. */
        FConstCameraActorIterator GetAutoActivateCameraIterator() const;

        /** Returns a reference to the game viewport displaying this world if one exists. */
        UGameViewportClient* GetGameViewport() const;

private:
        /** Begin async simulation */
        void StartAsyncSim();

        friend FStartAsyncSimulationFunction;

public:

        /**
         * Returns the default brush for the persistent level.
         * This is usually the 'builder brush' for editor builds, undefined for non editor instances and may be NULL.
         */
        ABrush* GetDefaultBrush() const;

        /** Returns true if the actors have been initialized and are ready to start play */
        bool AreActorsInitialized() const;
```

```
        /** Returns true if gameplay has already started, false otherwise. */
        bool HasBegunPlay() const;

        /**
         * Returns time in seconds since world was brought up for play, IS stopped when game pauses, IS dilated/clamped
         *
         * @return time in seconds since world was brought up for play
         */
        float GetTimeSeconds() const;

        /**
        * Returns time in seconds since world was brought up for play, IS NOT stopped when game pauses, IS dilated/clamped
        *
        * @return time in seconds since world was brought up for play
        */
        float GetUnpausedTimeSeconds() const;

        /**
        * Returns time in seconds since world was brought up for play, does NOT stop when game pauses, NOT dilated/clamped
        *
        * @return time in seconds since world was brought up for play
        */
        float GetRealTimeSeconds() const;

        /**
        * Returns time in seconds since world was brought up for play, IS stopped when game pauses, NOT dilated/clamped
```

```
	*
	* @return time in seconds since world was brought up for play
	*/
	float GetAudioTimeSeconds() const;

	/**
	 * Returns the frame delta time in seconds adjusted by e.g. time dilation.
	 *
	 * @return frame delta time in seconds adjusted by e.g. time dilation
	 */
	float GetDeltaSeconds() const;

	/** Helper for getting the time since a certain time. */
	float TimeSince( float Time ) const;

	/** Helper for getting the mono far field culling distance. */
	float GetMonoFarFieldCullingDistance() const;

	/** Creates a new physics scene for this world. */
	void CreatePhysicsScene();

	/** Returns a pointer to the physics scene for this world. */
	FPhysScene* GetPhysicsScene() const { return PhysicsScene; }

	/** Set the physics scene to use by this world */
	void SetPhysicsScene(FPhysScene* InScene);

	/**
	 * Returns the default physics volume and creates it if necessary.
```

```
	 *
	 * @return default physics volume
	 */
	APhysicsVolume* GetDefaultPhysicsVolume() const;

	/** Returns true if a DefaultPhysicsVolume has been created. */
	bool HasDefaultPhysicsVolume() const { return DefaultPhysicsVolume != nullptr; }

	/** Add a physics volume to the list of those in the world. DefaultPhysicsVolume is not tracked. Used internally by APhysicsVolume. */
	void AddPhysicsVolume(APhysicsVolume* Volume);

	/** Removes a physics volume from the list of those in the world. */
	void RemovePhysicsVolume(APhysicsVolume* Volume);

	/** Get an iterator for all PhysicsVolumes in the world that are not a DefaultPhysicsVolume. */
	FConstPhysicsVolumeIterator GetNonDefaultPhysicsVolumeIterator() const;

	/** Get the count of all PhysicsVolumes in the world that are not a DefaultPhysicsVolume. */
	int32 GetNonDefaultPhysicsVolumeCount() const;

	/**
	 * Returns the current (or specified) level's level scripting actor
	 *
	 * @param	OwnerLevel	the level to get the level scripting actor for.  Must correspond to one of the levels in GWorld's Levels array;
	 *				Thus, only applicable when editing a multi-level map.  Defaults to the level currently being edited.
	 *
```

```
	 * @return		A pointer to the level scripting actor, if any, for the specified level, or NULL if no level scripting actor is available
	 */
	class ALevelScriptActor* GetLevelScriptActor( class ULevel* OwnerLevel=NULL ) const;

	/**
	 * Returns the AWorldSettings actor associated with this world.
	 *
	 * @return AWorldSettings actor associated with this world
	 */
	AWorldSettings* GetWorldSettings( bool bCheckStreamingPersistent = false, bool bChecked = true ) const;

	/**
	 * Returns the current levels BSP model.
	 *
	 * @return BSP UModel
	 */
	UModel* GetModel() const;

	/**
	 * Returns the Z component of the current world gravity.
	 *
	 * @return Z component of current world gravity.
	 */
	float GetGravityZ() const;

	/**
	 * Returns the Z component of the default world gravity.
```

```
     *
     * @return Z component of the default world gravity.
     */
    float GetDefaultGravityZ() const;

    /**
     * Returns the name of the current map, taking into account using a dummy persistent world
     * and loading levels into it via PrepareMapChange.
     *
     * @return        name of the current map
     */
    const FString GetMapName() const;

    /** Accessor for bRequiresHitProxies. */
    bool RequiresHitProxies() const
    {
        return bRequiresHitProxies;
    }

    /**
     * Inserts the passed in controller at the front of the linked list of controllers.
     *
     * @param        Controller        Controller to insert, use NULL to clear list
     */
    void AddController( AController* Controller );

    /**
     * Removes the passed in controller from the linked list of controllers.
     *
```

```
	 * @param	Controller	Controller to remove
	 */
	void RemoveController( AController* Controller );

	/**
	 * Inserts the passed in pawn at the front of the linked list of pawns.
	 *
	 * @param	Pawn	Pawn to insert, use NULL to clear list
	 */
	void AddPawn( APawn* Pawn );

	/**
	 * Removes the passed in pawn from the linked list of pawns.
	 *
	 * @param	Pawn	Pawn to remove
	 */
	void RemovePawn( APawn* Pawn );

	/**
	 * Adds the passed in actor to the special network actor list
	 * This list is used to specifically single out actors that are relevant for networking without having to scan the much large list
	 * @param	Actor	Actor to add
	 */
	void AddNetworkActor( AActor* Actor );

	/**
	 * Removes the passed in actor to from special network actor list
	 * @param	Actor	Actor to remove
```

```
	 */
	void RemoveNetworkActor( AActor* Actor );

	/** Add a listener for OnActorSpawned events */
	FDelegateHandle AddOnActorSpawnedHandler( const FOnActorSpawned::FDelegate& InHandler
);

	/** Remove a listener for OnActorSpawned events */
	void RemoveOnActorSpawnedHandler( FDelegateHandle InHandle );

	/**
	 * Returns whether the passed in actor is part of any of the loaded levels actors array.
	 * Warning: Will return true for pending kill actors!
	 *
	 * @param	Actor	Actor to check whether it is contained by any level
	 *
	 * @return	true if actor is contained by any of the loaded levels, false otherwise
	 */
	bool ContainsActor( AActor* Actor );

	/**
	 * Returns whether audio playback is allowed for this scene.
	 *
	 * @return true if current world is GWorld, false otherwise
	 */
	virtual bool AllowAudioPlayback();

	//~ Begin UObject Interface
	virtual void Serialize( FArchive& Ar ) override;
```

```
        virtual void FinishDestroy() override;
        virtual void PostLoad() override;
        virtual bool PreSaveRoot(const TCHAR* Filename) override;
        virtual void PostSaveRoot( bool bCleanupIsRequired ) override;
        virtual UWorld* GetWorld() const override;
        virtual FPrimaryAssetId GetPrimaryAssetId() const override;
        static void AddReferencedObjects(UObject* InThis, FReferenceCollector& Collector);
#if WITH_EDITOR
        virtual bool Rename(const TCHAR* NewName = NULL, UObject* NewOuter = NULL,
ERenameFlags Flags = REN_None) override;
        virtual void GetAssetRegistryTags(TArray<FAssetRegistryTag>& OutTags) const override;
#endif
        virtual void PostDuplicate(bool bDuplicateForPIE) override;
        //~ End UObject Interface

        /**
         * Clears all level components and world components like e.g. line batcher.
         */
        void ClearWorldComponents();

        /**
         * Updates world components like e.g. line batcher and all level components.
         *
         * @param       bRerunConstructionScripts       If we should rerun construction scripts on
actors
         * @param       bCurrentLevelOnly                       If true, affect only the current level.
         */
        void UpdateWorldComponents(bool bRerunConstructionScripts, bool bCurrentLevelOnly);
```

```
	/**
	 * Updates cull distance volumes for a specified component or a specified actor or all actors
	* @param ComponentToUpdate If specified just that Component will be updated
	 * @param ActorToUpdate If specified (and ComponentToUpdate is not specified), all Components owned by this Actor will be updated
	 */
	void UpdateCullDistanceVolumes(AActor* ActorToUpdate = nullptr, UPrimitiveComponent* ComponentToUpdate = nullptr);

	/**
	 * Cleans up components, streaming data and assorted other intermediate data.
	 * @param bSessionEnded whether to notify the viewport that the game session has ended
	 * @param NewWorld Optional new world that will be loaded after this world is cleaned up. Specify a new world to prevent it and it's sublevels from being GCed during map transitions.
	 */
	void CleanupWorld(bool bSessionEnded = true, bool bCleanupResources = true, UWorld* NewWorld = nullptr);

	/**
	 * Invalidates the cached data used to render the levels' UModel.
	 *
	 * @param	InLevel		Level to invalidate. If this is NULL it will affect ALL levels
	 */
	void InvalidateModelGeometry( ULevel* InLevel );

	/**
	 * Discards the cached data used to render the levels' UModel.  Assumes that the
	 * faces and vertex positions haven't changed, only the applied materials.
	 *
	 * @param	bCurrentLevelOnly		If true, affect only the current level.
```

```
	 */
	void InvalidateModelSurface(bool bCurrentLevelOnly);

	/**
	 * Commits changes made to the surfaces of the UModels of all levels.
	 */
	void CommitModelSurfaces();

	/** Purges all sky capture cached derived data and forces a re-render of captured scene data. */
	void UpdateAllSkyCaptures();

	/** Returns the active lighting scenario for this world or NULL if none. */
	ULevel* GetActiveLightingScenario() const;

	/** Propagates a change to the active lighting scenario. */
	void PropagateLightingScenarioChange();

	/**
	 * Associates the passed in level with the world. The work to make the level visible is spread across several frames and this
	 * function has to be called till it returns true for the level to be visible/ associated with the world and no longer be in
	 * a limbo state.
	 *
	 * @param Level					Level object we should add
	 * @param LevelTransform		Transformation to apply to each actor in the level
	 */
	void AddToWorld( ULevel* Level, const FTransform& LevelTransform = FTransform::Identity );
```

```
	/**
	 * Dissociates the passed in level from the world. The removal is blocking.
	 *
	 * @param Level				Level object we should remove
	 */
	void RemoveFromWorld( ULevel* Level, bool bAllowIncrementalRemoval = false );

	/**
	 * Updates sub-levels (load/unload/show/hide) using streaming levels current state
	 */
	void UpdateLevelStreaming();

private:
	void UpdateLevelStreamingInner( ULevelStreaming* StreamingLevel );

public:
	/**
	 * Flushes level streaming in blocking fashion and returns when all levels are loaded/ visible/ hidden
	 * so further calls to UpdateLevelStreaming won't do any work unless state changes. Basically blocks
	 * on all async operation like updating components.
	 *
	 * @param FlushType				Whether to only flush level visibility operations (optional)
	 */
	void FlushLevelStreaming(EFlushLevelStreamingType FlushType = EFlushLevelStreamingType::Full);

	/**
```

```
	 * Triggers a call to ULevel::BuildStreamingData(this,NULL,NULL) within a few seconds.
	 */
	void TriggerStreamingDataRebuild();

	/**
	 * Calls ULevel::BuildStreamingData(this,NULL,NULL) if it has been triggered within the last few seconds.
	 */
	void ConditionallyBuildStreamingData();

	/** @return whether there is at least one level with a pending visibility request */
	bool IsVisibilityRequestPending() const;

	/** Returns whether all the 'always loaded' levels are loaded. */
	bool AreAlwaysLoadedLevelsLoaded() const;

	/** Requests async loading of any 'always loaded' level. Used in seamless travel to prevent blocking in the first UpdateLevelStreaming.  */
	void AsyncLoadAlwaysLoadedLevelsForSeamlessTravel();

	/**
	 * Returns whether the level streaming code is allowed to issue load requests.
	 *
	 * @return true if level load requests are allowed, false otherwise.
	 */
	bool AllowLevelLoadRequests();

	/** Creates instances for each parameter collection in memory.  Called when a world is created. */
	void SetupParameterCollectionInstances();
```

```
	/** Adds a new instance of the given collection, or overwrites an existing instance if there is one. */
	void AddParameterCollectionInstance(class UMaterialParameterCollection* Collection, bool bUpdateScene);

	/** Gets this world's instance for a given collection. */
	UMaterialParameterCollectionInstance* GetParameterCollectionInstance(const UMaterialParameterCollection* Collection);

	/** Updates this world's scene with the list of instances, and optionally updates each instance's uniform buffer. */
	void UpdateParameterCollectionInstances(bool bUpdateInstanceUniformBuffers);

	/** Gets the canvas object for rendering to a render target.  Will allocate one if needed. */
	UCanvas* GetCanvasForRenderingToTarget();
	UCanvas* GetCanvasForDrawMaterialToRenderTarget();

	/** Struct containing a collection of optional parameters for initialization of a World. */
	struct InitializationValues
	{
		InitializationValues()
			: bInitializeScenes(true)
			, bAllowAudioPlayback(true)
			, bRequiresHitProxies(true)
			, bCreatePhysicsScene(true)
			, bCreateNavigation(true)
			, bCreateAISystem(true)
			, bShouldSimulatePhysics(true)
			, bEnableTraceCollision(false)
```

```
                , bTransactional(true)
                , bCreateFXSystem(true)
        {
        }

        /** Should the scenes (physics, rendering) be created. */
        uint32 bInitializeScenes:1;

        /** Are sounds allowed to be generated from this world. */
        uint32 bAllowAudioPlayback:1;

        /** Should the render scene create hit proxies. */
        uint32 bRequiresHitProxies:1;

        /** Should the physics scene be created. bInitializeScenes must be true for this to be
considered. */
        uint32 bCreatePhysicsScene:1;

        /** Should the navigation system be created for this world. */
        uint32 bCreateNavigation:1;

        /** Should the AI system be created for this world. */
        uint32 bCreateAISystem:1;

        /** Should physics be simulated in this world. */
        uint32 bShouldSimulatePhysics:1;

        /** Are collision trace calls valid within this world. */
        uint32 bEnableTraceCollision:1;
```

```
		/** Should actions performed to objects in this world be saved to the transaction buffer.
*/
		uint32 bTransactional:1;

		/** Should the FX system be created for this world. */
		uint32 bCreateFXSystem:1;

		InitializationValues& InitializeScenes(const bool bInitialize) { bInitializeScenes =
bInitialize; return *this; }
		InitializationValues& AllowAudioPlayback(const bool bAllow) { bAllowAudioPlayback =
bAllow; return *this; }
		InitializationValues& RequiresHitProxies(const bool bRequires) { bRequiresHitProxies =
bRequires; return *this; }
		InitializationValues& CreatePhysicsScene(const bool bCreate) { bCreatePhysicsScene =
bCreate; return *this; }
		InitializationValues& CreateNavigation(const bool bCreate) { bCreateNavigation =
bCreate; return *this; }
		InitializationValues& CreateAISystem(const bool bCreate) { bCreateAISystem = bCreate;
return *this; }
		InitializationValues& ShouldSimulatePhysics(const bool bInShouldSimulatePhysics) {
bShouldSimulatePhysics = bInShouldSimulatePhysics; return *this; }
		InitializationValues& EnableTraceCollision(const bool bInEnableTraceCollision) {
bEnableTraceCollision = bInEnableTraceCollision; return *this; }
		InitializationValues& SetTransactional(const bool bInTransactional) { bTransactional =
bInTransactional; return *this; }
		InitializationValues& CreateFXSystem(const bool bCreate) { bCreateFXSystem = bCreate;
return *this; }
	};

	/**
	 * Initializes the world, associates the persistent level and sets the proper zones.
```

```
	 */
	void InitWorld(const InitializationValues IVS = InitializationValues());

	/**
	 * Initializes a newly created world.
	 */
	void InitializeNewWorld(const InitializationValues IVS = InitializationValues());

	/**
	 * Static function that creates a new UWorld and returns a pointer to it
	 */
	static UWorld* CreateWorld( const EWorldType::Type InWorldType, bool
bInformEngineOfWorld, FName WorldName = NAME_None, UPackage* InWorldPackage = NULL, bool
bAddToRoot = true, ERHIFeatureLevel::Type InFeatureLevel = ERHIFeatureLevel::Num );

	/**
	 * Destroy this World instance. If destroying the world to load a different world, supply it here to
prevent GC of the new world or it's sublevels.
	 */
	void DestroyWorld( bool bInformEngineOfWorld, UWorld* NewWorld = nullptr );

	/**
	 * Marks all objects that have this World as an Outer as pending kill
	 */
	void MarkObjectsPendingKill();

	/**
	 *  Interface to allow WorldSettings to request immediate garbage collection
	 */
	DEPRECATED(4.18, "Use GEngine->PerformGarbageCollectionAndCleanupActors instead.")
```

```
	void PerformGarbageCollectionAndCleanupActors();


	/**
	 *  Requests a one frame delay of Garbage Collection
	 */
	DEPRECATED(4.18, "Use GEngine->DelayGarbageCollection instead.")
	void DelayGarbageCollection();


	/**
	 * Updates the timer (as a one-off) that is used to trigger garbage collection; this should only be used for things
	 * like performance tests, using it recklessly can dramatically increase memory usage and cost of the eventual GC.
	 *
	 * Note: Things that force a GC will still force a GC after using this method (and they will also reset the timer)
	 */
	DEPRECATED(4.18, "Use GEngine->SetTimeUntilNextGarbageCollection instead.")
	void SetTimeUntilNextGarbageCollection(float MinTimeUntilNextPass);


	/**
	 * Returns the current desired time between garbage collection passes (not the time remaining)
	 */
	DEPRECATED(4.18, "Call GEngine->GetTimeBetweenGarbageCollectionPasses instead")
	float GetTimeBetweenGarbageCollectionPasses() const;


	/**
	 *	Remove NULL entries from actor list. Only does so for dynamic actors to avoid resorting.
	 *	In theory static actors shouldn't be deleted during gameplay.
	 */
```

```
        void CleanupActors();

public:

        /** Get the event that broadcasts TickDispatch */
        FOnNetTickEvent& OnTickDispatch() { return TickDispatchEvent; }
        /** Get the event that broadcasts TickFlush */
        FOnNetTickEvent& OnTickFlush() { return TickFlushEvent; }
        /** Get the event that broadcasts TickFlush */
        FOnTickFlushEvent& OnPostTickFlush() { return PostTickFlushEvent; }

        /**
         * Update the level after a variable amount of time, DeltaSeconds, has passed.
         * All child actors are ticked after their owners have been ticked.
         */
        void Tick( ELevelTick TickType, float DeltaSeconds );

        /**
         * Set up the physics tick function if they aren't already
         */
        void SetupPhysicsTickFunctions(float DeltaSeconds);

        /**
         * Run a tick group, ticking all actors and components
         * @param Group - Ticking group to run
         * @param bBlockTillComplete - if true, do not return until all ticks are complete
         */
        void RunTickGroup(ETickingGroup Group, bool bBlockTillComplete);
```

```
	/**
	 * Mark a component as needing an end of frame update
	 * @param Component - Component to update at the end of the frame
	 * @param bForceGameThread - if true, force this to happen on the game thread
	 */
	void MarkActorComponentForNeededEndOfFrameUpdate(UActorComponent* Component, bool bForceGameThread);

	/**
	* Clears the need for a component to have a end of frame update
	* @param Component - Component to update at the end of the frame
	*/
	void ClearActorComponentEndOfFrameUpdate(UActorComponent* Component);

	/**
	 * Updates an ActorComponent's cached state of whether it has been marked for end of frame update based on the current
	 * state of the World's NeedsEndOfFrameUpdate arrays
	 * @param Component - Component to update the cached state of
	 */
	void UpdateActorComponentEndOfFrameUpdateState(UActorComponent* Component) const;

	bool HasEndOfFrameUpdates();

	/**
	 * Send all render updates to the rendering thread.
	 */
	void SendAllEndOfFrameUpdates();
```

```
        /** Do per frame tick behaviors related to the network driver */
        void TickNetClient( float DeltaSeconds );

        /**
         * Issues level streaming load/unload requests based on whether
         * local players are inside/outside level streaming volumes.
         *
         * @param OverrideViewLocation Optional position used to override the location used to calculate current streaming volumes
         */
        void ProcessLevelStreamingVolumes(FVector* OverrideViewLocation=NULL);

        /**
         * Transacts the specified level -- the correct way to modify a level
         * as opposed to calling Level->Modify.
         */
        void ModifyLevel(ULevel* Level);

        /**
         * Ensures that the collision detection tree is fully built. This should be called after the full level reload to make sure
         * the first traces are not abysmally slow.
         */
        void EnsureCollisionTreeIsBuilt();

#if WITH_EDITOR
        /** Returns the SelectedLevelsChangedEvent member. */
        FOnSelectedLevelsChangedEvent& OnSelectedLevelsChanged() { return SelectedLevelsChangedEvent; }
```

```
/**
 * Flag a level as selected.
 */
void SelectLevel( ULevel* InLevel );

/**
 * Flag a level as not selected.
 */
void DeSelectLevel( ULevel* InLevel );

/**
 * Query whether or not a level is selected.
 */
bool IsLevelSelected( ULevel* InLevel ) const;

/**
 * Set the selected levels from the given array (Clears existing selections)
 */
void SetSelectedLevels( const TArray<class ULevel*>& InLevels );

/**
 * Return the number of levels in this world.
 */
int32 GetNumSelectedLevels() const;

/**
 * Return the selected level with the given index.
 */
ULevel* GetSelectedLevel( int32 InLevelIndex ) const;
```

```
	/**
	 * Return the list of selected levels in this world.
	 */
	TArray<class ULevel*>& GetSelectedLevels();

	/** Shrink level elements to their minimum size. */
	void ShrinkLevel();
#endif // WITH_EDITOR

	/**
	 * Returns an iterator for the level list.
	 */
	FConstLevelIterator		GetLevelIterator() const;

	/**
	 * Return the level with the given index.
	 */
	ULevel* GetLevel( int32 InLevelIndex ) const;

	/**
	 * Does the level list contain the given level.
	 */
	bool ContainsLevel( ULevel* InLevel ) const;

	/**
	 * Return the number of levels in this world.
	 */
	int32 GetNumLevels() const;
```

```
/**
 * Return the list of levels in this world.
 */
const TArray<class ULevel*>& GetLevels() const;

/**
 * Add a level to the level list.
 */
bool AddLevel( ULevel* InLevel );

/**
 * Remove a level from the level list.
 */
bool RemoveLevel( ULevel* InLevel );

/** Returns the FLevelCollection for the given InType. If one does not exist, it is created. */
FLevelCollection& FindOrAddCollectionByType(const ELevelCollectionType InType);

/** Returns the index of the first FLevelCollection of the given InType. If one does not exist, it is created and its index returned. */
int32 FindOrAddCollectionByType_Index(const ELevelCollectionType InType);

/** Returns the FLevelCollection for the given InType, or null if a collection of that type hasn't been created yet. */
FLevelCollection* FindCollectionByType(const ELevelCollectionType InType);

/** Returns the FLevelCollection for the given InType, or null if a collection of that type hasn't been created yet. */
const FLevelCollection* FindCollectionByType(const ELevelCollectionType InType) const;
```

```
/** Returns the index of the FLevelCollection with the given InType, or INDEX_NONE if a collection of that type hasn't been created yet. */
int32 FindCollectionIndexByType(const ELevelCollectionType InType) const;

/**
 * Returns the level collection which currently has its context set on this world. May be null.
 * If non-null, this implies that execution is currently within the scope of an FScopedLevelCollectionContextSwitch for this world.
 */
const FLevelCollection* GetActiveLevelCollection() const;

/**
 * Returns the index of the level collection which currently has its context set on this world. May be INDEX_NONE.
 * If not INDEX_NONE, this implies that execution is currently within the scope of an FScopedLevelCollectionContextSwitch for this world.
 */
int32 GetActiveLevelCollectionIndex() const { return ActiveLevelCollectionIndex; }

/** Sets the level collection and its context on this world. Should only be called by FScopedLevelCollectionContextSwitch. */
void SetActiveLevelCollection(int32 LevelCollectionIndex);

/** Returns a read-only reference to the list of level collections in this world. */
const TArray<FLevelCollection>& GetLevelCollections() const { return LevelCollections; }

/**
 * Creates a new level collection of type DynamicDuplicatedLevels by duplicating the levels in DynamicSourceLevels.
 * Should only be called by engine.
```

```
	 *
	 * @param MapName The name of the soure map, used as a parameter to UEngine::Experimental_ShouldPreDuplicateMap
	 */
	void DuplicateRequestedLevels(const FName MapName);

	/** Handle Exec/Console Commands related to the World */
	bool Exec( UWorld* InWorld, const TCHAR* Cmd, FOutputDevice& Ar=*GLog );

private:
	/** Utility function to handle Exec/Console Commands related to the Trace Tags */
	bool HandleTraceTagCommand( const TCHAR* Cmd, FOutputDevice& Ar );

	/** Utility function to handle Exec/Console Commands related to persistent debug lines */
	bool HandleFlushPersistentDebugLinesCommand( const TCHAR* Cmd, FOutputDevice& Ar );

	/** Utility function to handle Exec/Console Commands related to logging actor counts */
	bool HandleLogActorCountsCommand( const TCHAR* Cmd, FOutputDevice& Ar, UWorld* InWorld );

	/** Utility function to handle Exec/Console Commands related to demo recording */
	bool HandleDemoRecordCommand( const TCHAR* Cmd, FOutputDevice& Ar, UWorld* InWorld );

	/** Utility function to handle Exec/Console Commands related to playing a demo recording*/
	bool HandleDemoPlayCommand( const TCHAR* Cmd, FOutputDevice& Ar, UWorld* InWorld );

	/** Utility function to handle Exec/Console Commands related to stopping demo playback */
	bool HandleDemoStopCommand( const TCHAR* Cmd, FOutputDevice& Ar, UWorld* InWorld );
```

```
        /** Utility function to handle Exec/Console Command for scrubbing to a specific time */
        bool HandleDemoScrubCommand(const TCHAR* Cmd, FOutputDevice& Ar, UWorld* InWorld);

        /** Utility function to handle Exec/Console Command for pausing and unpausing a replay */
        bool HandleDemoPauseCommand(const TCHAR* Cmd, FOutputDevice& Ar, UWorld* InWorld);

        /** Utility function to handle Exec/Console Command for setting the speed of a replay */
        bool HandleDemoSpeedCommand(const TCHAR* Cmd, FOutputDevice& Ar, UWorld* InWorld);

public:

        // Destroys the current demo net driver
        void DestroyDemoNetDriver();

        /** Returns true if we are currently playing a replay */
        bool IsPlayingReplay() const { return (DemoNetDriver ? DemoNetDriver->IsPlaying() : false); }

        // Start listening for connections.
        bool Listen( FURL& InURL );

        /** @return true if this level is a client */
        bool IsClient() const;

        /** @return true if this level is a server */
        bool IsServer() const;

        /** @return true if the world is in the paused state */
        bool IsPaused() const;
```

```
	/** @return true if the camera is in a moveable state (taking pausedness into account) */
	bool IsCameraMoveable() const;

	/**
	 * Wrapper for DestroyActor() that should be called in the editor.
	 *
	 * @param	bShouldModifyLevel		If true, Modify() the level before removing the actor.
	 */
	bool EditorDestroyActor( AActor* Actor, bool bShouldModifyLevel );

	/**
	 * Removes the actor from its level's actor list and generally cleans up the engine's internal state.
	 * What this function does not do, but is handled via garbage collection instead, is remove references
	 * to this actor from all other actors, and kill the actor's resources.  This function is set up so that
	 * no problems occur even if the actor is being destroyed inside its recursion stack.
	 *
	 * @param	ThisActor				Actor to remove.
	 * @param	bNetForce				[opt] Ignored unless called during play. Default is false.
	 * @param	bShouldModifyLevel		[opt] If true, Modify() the level before removing the actor.  Default is true.
	 * @return							true if destroyed or already marked for destruction, false if actor couldn't be destroyed.
	 */
	bool DestroyActor( AActor* Actor, bool bNetForce=false, bool bShouldModifyLevel=true );

	/**
```

```
	* Removes the passed in actor from the actor lists. Please note that the code actually doesn't physically remove the
	* index but rather clears it so other indices are still valid and the actors array size doesn't change.
	*
	* @param	Actor				Actor to remove.
	* @param	bShouldModifyLevel	If true, Modify() the level before removing the actor if in the editor.
	*/
	void RemoveActor( AActor* Actor, bool bShouldModifyLevel );

	/**
	 * Spawn Actors with given transform and SpawnParameters
	 *
	 * @param	Class				Class to Spawn
	 * @param	Location			Location To Spawn
	 * @param	Rotation			Rotation To Spawn
	 * @param	SpawnParameters		Spawn Parameters
	 *
	 * @return	Actor that just spawned
	 */
	AActor* SpawnActor( UClass* InClass, FVector const* Location=NULL, FRotator const* Rotation=NULL, const FActorSpawnParameters& SpawnParameters = FActorSpawnParameters() );

	/**
	 * Spawn Actors with given transform and SpawnParameters
	 *
	 * @param	Class				Class to Spawn
	 * @param	Transform			World Transform to spawn on
	 * @param	SpawnParameters		Spawn Parameters
	 *
```

```
	 * @return		Actor that just spawned
	 */
	AActor* SpawnActor( UClass* Class, FTransform const* Transform, const
FActorSpawnParameters& SpawnParameters = FActorSpawnParameters());

	/**
	 * Spawn Actors with given absolute transform (override root component transform) and
SpawnParameters
	 *
	 * @param	Class					Class to Spawn
	 * @param	AbsoluteTransform		World Transform to spawn on - without
considering CDO's relative transform, thus Absolute
	 * @param	SpawnParameters			Spawn Parameters
	 *
	 * @return	Actor that just spawned
	 */
	AActor* SpawnActorAbsolute( UClass* Class, FTransform const& AbsoluteTransform, const
FActorSpawnParameters& SpawnParameters = FActorSpawnParameters());

	/** Templated version of SpawnActor that allows you to specify a class type via the template
type */
	template< class T >
	T* SpawnActor( const FActorSpawnParameters& SpawnParameters = FActorSpawnParameters()
)
	{
		return CastChecked<T>(SpawnActor(T::StaticClass(), NULL, NULL,
SpawnParameters),ECastCheckedType::NullAllowed);
	}

	/** Templated version of SpawnActor that allows you to specify location and rotation in
addition to class type via the template type */
```

```
template< class T >
T* SpawnActor( FVector const& Location, FRotator const& Rotation, const
FActorSpawnParameters& SpawnParameters = FActorSpawnParameters() )
{
	return CastChecked<T>(SpawnActor(T::StaticClass(), &Location, &Rotation,
SpawnParameters),ECastCheckedType::NullAllowed);
}

/** Templated version of SpawnActor that allows you to specify the class type via parameter
while the return type is a parent class of that type */
template< class T >
T* SpawnActor( UClass* Class, const FActorSpawnParameters& SpawnParameters =
FActorSpawnParameters() )
{
	return CastChecked<T>(SpawnActor(Class, NULL, NULL,
SpawnParameters),ECastCheckedType::NullAllowed);
}

/**
 *  Templated version of SpawnActor that allows you to specify the rotation and location in
addition
 *  class type via parameter while the return type is a parent class of that type
 */
template< class T >
T* SpawnActor( UClass* Class, FVector const& Location, FRotator const& Rotation, const
FActorSpawnParameters& SpawnParameters = FActorSpawnParameters() )
{
	return CastChecked<T>(SpawnActor(Class, &Location, &Rotation,
SpawnParameters),ECastCheckedType::NullAllowed);
}
/**
```

```
	 *  Templated version of SpawnActor that allows you to specify whole Transform
	 *  class type via parameter while the return type is a parent class of that type
	 */
	template< class T >
	T* SpawnActor(UClass* Class, FTransform const& Transform,const FActorSpawnParameters& SpawnParameters = FActorSpawnParameters())
	{
		return CastChecked<T>(SpawnActor(Class, &Transform, SpawnParameters), ECastCheckedType::NullAllowed);
	}

	/** Templated version of SpawnActorAbsolute that allows you to specify absolute location and rotation in addition to class type via the template type */
	template< class T >
	T* SpawnActorAbsolute(FVector const& AbsoluteLocation, FRotator const& AbsoluteRotation, const FActorSpawnParameters& SpawnParameters = FActorSpawnParameters())
	{
		return CastChecked<T>(SpawnActorAbsolute(T::StaticClass(), FTransform(AbsoluteRotation, AbsoluteLocation), SpawnParameters), ECastCheckedType::NullAllowed);
	}

	/**
	 *  Templated version of SpawnActorAbsolute that allows you to specify whole absolute Transform
	 *  class type via parameter while the return type is a parent class of that type
	 */
	template< class T >
	T* SpawnActorAbsolute(UClass* Class, FTransform const& Transform,const FActorSpawnParameters& SpawnParameters = FActorSpawnParameters())
	{
```

```
		return CastChecked<T>(SpawnActorAbsolute(Class, Transform, SpawnParameters), ECastCheckedType::NullAllowed);
	}

	/**
	 * Spawns given class and returns class T pointer, forcibly sets world transform (note this allows scale as well). WILL NOT run Construction Script of Blueprints
	 * to give caller an opportunity to set parameters beforehand.  Caller is responsible for invoking construction
	 * manually by calling UGameplayStatics::FinishSpawningActor (see AActor::OnConstruction).
	 */
	template< class T >
	T* SpawnActorDeferred(
		UClass* Class,
		FTransform const& Transform,
		AActor* Owner = nullptr,
		APawn* Instigator = nullptr,
		ESpawnActorCollisionHandlingMethod CollisionHandlingOverride = ESpawnActorCollisionHandlingMethod::Undefined
		)
	{
		if( Owner )
		{
			check(this==Owner->GetWorld());
		}
		FActorSpawnParameters SpawnInfo;
		SpawnInfo.SpawnCollisionHandlingOverride = CollisionHandlingOverride;
		SpawnInfo.Owner = Owner;
		SpawnInfo.Instigator = Instigator;
		SpawnInfo.bDeferConstruction = true;
```

```
		return (Class != nullptr) ? Cast<T>(SpawnActor(Class, &Transform, SpawnInfo)) : nullptr;
	}

	/**
	 * Returns the current Game Mode instance cast to the template type.
	 * This can only return a valid pointer on the server and may be null if the cast fails. Will always return null on a client.
	 */
	template< class T >
	T* GetAuthGameMode() const
	{
		return Cast<T>(AuthorityGameMode);
	}

	/**
	 * Returns the current Game Mode instance, which is always valid during gameplay on the server.
	 * This will only return a valid pointer on the server. Will always return null on a client.
	 */
	AGameModeBase* GetAuthGameMode() const { return AuthorityGameMode; }

	/** Returns the current GameState instance cast to the template type. */
	template< class T >
	T* GetGameState() const
	{
		return Cast<T>(GameState);
	}

	/** Returns the current GameState instance. */
```

```
	AGameStateBase* GetGameState() const { return GameState; }

	/** Sets the current GameState instance on this world and the game state's level collection. */
	void SetGameState(AGameStateBase* NewGameState);

	/** Copies GameState properties from the GameMode. */
	void CopyGameState(AGameModeBase* FromGameMode, AGameStateBase* FromGameState);


	/** Spawns a Brush Actor in the World */
	ABrush*		SpawnBrush();

	/**
	 * Spawns a PlayerController and binds it to the passed in Player with the specified RemoteRole and options
	 *
	 * @param Player - the Player to set on the PlayerController
	 * @param RemoteRole - the RemoteRole to set on the PlayerController
	 * @param URL - URL containing player options (name, etc)
	 * @param UniqueId - unique net ID of the player (may be zeroed if no online subsystem or not logged in, e.g. a local game or LAN match)
	 * @param Error (out) - if set, indicates that there was an error - usually is set to a property from which the calling code can look up the actual message
	 * @param InNetPlayerIndex (optional) - the NetPlayerIndex to set on the PlayerController
	 * @return the PlayerController that was spawned (may fail and return NULL)
	 */
	APlayerController* SpawnPlayActor(class UPlayer* Player, ENetRole RemoteRole, const FURL& InURL, const TSharedPtr<const FUniqueNetId>& UniqueId, FString& Error, uint8 InNetPlayerIndex = 0);
	APlayerController* SpawnPlayActor(class UPlayer* Player, ENetRole RemoteRole, const FURL& InURL, const FUniqueNetIdRepl& UniqueId, FString& Error, uint8 InNetPlayerIndex = 0);
```

```
	/** Try to find an acceptable position to place TestActor as close to possible to PlaceLocation. Expects PlaceLocation to be a valid location inside the level. */
	bool FindTeleportSpot( AActor* TestActor, FVector& PlaceLocation, FRotator PlaceRotation );

	/** @Return true if Actor would encroach at TestLocation on something that blocks it.  Returns a ProposedAdjustment that might result in an unblocked TestLocation. */
	bool EncroachingBlockingGeometry( AActor* TestActor, FVector TestLocation, FRotator TestRotation, FVector* ProposedAdjustment = NULL );

	/** Begin physics simulation */
	void StartPhysicsSim();

	/** Waits for the physics scene to be done processing */
	void FinishPhysicsSim();

	/** Spawns GameMode for the level. */
	bool SetGameMode(const FURL& InURL);

	/**
	 * Initializes all actors and prepares them to start gameplay
	 * @param InURL commandline URL
	 * @param bResetTime (optional) whether the WorldSettings's TimeSeconds should be reset to zero
	 */
	void InitializeActorsForPlay(const FURL& InURL, bool bResetTime = true);

	/**
	 * Start gameplay. This will cause the game mode to transition to the correct state and call BeginPlay on all actors
```

```
	 */
	void BeginPlay();

	/**
	 * Looks for a PlayerController that was being swapped by the given NetConnection and, if found, destroys it
	 * (because the swap is complete or the connection was closed)
	 * @param Connection - the connection that performed the swap
	 * @return whether a PC waiting for a swap was found
	 */
	bool DestroySwappedPC(UNetConnection* Connection);

	//~ Begin FNetworkNotify Interface
	virtual EAcceptConnection::Type NotifyAcceptingConnection() override;
	virtual void NotifyAcceptedConnection( class UNetConnection* Connection ) override;
	virtual bool NotifyAcceptingChannel( class UChannel* Channel ) override;
	virtual void NotifyControlMessage(UNetConnection* Connection, uint8 MessageType, class FInBunch& Bunch) override;
	//~ End FNetworkNotify Interface

	/** Welcome a new player joining this server. */
	void WelcomePlayer(UNetConnection* Connection);

	/**
	 * Used to get a net driver object by name. Default name is the game net driver
	 * @param NetDriverName the name of the net driver being asked for
	 * @return a pointer to the net driver or NULL if the named driver is not found
	 */
	FORCEINLINE_DEBUGGABLE UNetDriver* GetNetDriver() const
```

```
	{
		return NetDriver;
	}

	/**
	 * Returns the net mode this world is running under.
	 * @see IsNetMode()
	 */
	ENetMode GetNetMode() const;

	/**
	* Test whether net mode is the given mode.
	* In optimized non-editor builds this can be more efficient than GetNetMode()
	* because it can check the static build flags without considering PIE.
	*/
	bool IsNetMode(ENetMode Mode) const;

private:

	/** Private version without inlining that does *not* check Dedicated server build flags (which should already have been done). */
	ENetMode InternalGetNetMode() const;

	// Sends the NMT_Challenge message to Connection.
	void SendChallengeControlMessage(UNetConnection* Connection);
	void SendChallengeControlMessage(const FEncryptionKeyResponse& Response, TWeakObjectPtr<UNetConnection> WeakConnection);

public:
```

```
#if WITH_EDITOR
        /** Attempts to derive the net mode from PlayInSettings for PIE*/
        ENetMode AttemptDeriveFromPlayInSettings() const;
#endif

        /** Attempts to derive the net mode from URL */
        ENetMode AttemptDeriveFromURL() const;

        /**
         * Sets the net driver to use for this world
         * @param NewDriver the new net driver to use
         */
        void SetNetDriver(UNetDriver* NewDriver)
        {
                NetDriver = NewDriver;
        }

        /**
         * Returns true if the game net driver exists and is a client and the demo net driver exists and is a server.
         */
        bool IsRecordingClientReplay() const;

        /**
         * Sets the number of frames to delay Streaming Volume updating,
         * useful if you preload a bunch of levels but the camera hasn't caught up yet
         */
        void DelayStreamingVolumeUpdates(int32 InFrameDelay)
```

```
	{
		StreamingVolumeUpdateDelay = InFrameDelay;
	}

	/**
	 * Transfers the set of Kismet / Blueprint objects being debugged to the new world that are not already present, and updates blueprints accordingly
	 * @param	NewWorld	The new world to find equivalent objects in
	 */
	void TransferBlueprintDebugReferences(UWorld* NewWorld);

	/**
	 * Notifies the world of a blueprint debugging reference
	 * @param	Blueprint	The blueprint the reference is for
	 * @param	DebugObject The associated debugging object (may be NULL)
	 */
	void NotifyOfBlueprintDebuggingAssociation(class UBlueprint* Blueprint, UObject* DebugObject);

	/** Broadcasts that the number of levels has changed. */
	void BroadcastLevelsChanged();

	/** Returns the LevelsChangedEvent member. */
	FOnLevelsChangedEvent& OnLevelsChanged() { return LevelsChangedEvent; }

	/** Returns the actor count. */
	int32 GetProgressDenominator();

	/** Returns the actor count. */
```

```
	int32 GetActorCount();


public:


	/**
	 * Finds the audio settings to use for a given view location, taking into account the world's default
	 * settings and the audio volumes in the world.
	 *
	 * @param	ViewLocation			Current view location.
	 * @param	OutReverbSettings		[out] Upon return, the reverb settings for a camera at ViewLocation.
	 * @param	OutInteriorSettings		[out] Upon return, the interior settings for a camera at ViewLocation.
	 * @return 							If the settings came from an audio volume, the audio volume object is returned.
	 */
	class AAudioVolume* GetAudioSettings( const FVector& ViewLocation, struct FReverbSettings* OutReverbSettings, struct FInteriorSettings* OutInteriorSettings );


	/** Returns the audio device handle for this world.*/
	uint32 GetAudioDeviceHandle() const { return AudioDeviceHandle; }


	/** Sets the audio device handle to the active audio device for this world.*/
	void SetAudioDeviceHandle(const uint32 InAudioDeviceHandle);


	/**
	 * Returns the audio device associated with this world, or returns the main audio device if there is none.
	 *
	 * @return Audio device to use with this world.
```

```
	*/
	class FAudioDevice* GetAudioDevice();

	/** Return the URL of this level on the local machine. */
	virtual FString GetLocalURL() const;

	/** Returns whether script is executing within the editor. */
	bool IsPlayInEditor() const;

	/** Returns whether script is executing within a preview window */
	bool IsPlayInPreview() const;

	/** Returns whether script is executing within a mobile preview window */
	bool IsPlayInMobilePreview() const;

	/** Returns whether script is executing within a vulkan preview window */
	bool IsPlayInVulkanPreview() const;

	/** Returns true if this world is any kind of game world (including PIE worlds) */
	bool IsGameWorld() const;

	/** Returns true if this world is any kind of editor world (including editor preview worlds) */
	bool IsEditorWorld() const;

	/** Returns true if this world is a preview game world (editor or game) */
	bool IsPreviewWorld() const;

	/** Returns true if this world should look at game hidden flags instead of editor hidden flags for the purposes of rendering */
```

```
	bool UsesGameHiddenFlags() const;

	// Return the URL of this level, which may possibly
	// exist on a remote machine.
	virtual FString GetAddressURL() const;

	/**
	 * Called after GWorld has been set. Used to load, but not associate, all
	 * levels in the world in the Editor and at least create linkers in the game.
	 * Should only be called against GWorld::PersistentLevel's WorldSettings.
	 *
	 * @param bForce		If true, load the levels even is a commandlet
	 */
	void LoadSecondaryLevels(bool bForce = false, TSet<FString>* CookedPackages = NULL);

	/** Utility for returning the ULevelStreaming object for a particular sub-level, specified by package name */
	ULevelStreaming* GetLevelStreamingForPackageName(FName PackageName);

#if WITH_EDITOR
	/**
	 * Called when level property has changed
	 * It refreshes any streaming stuff
	 */
	void RefreshStreamingLevels();

	/**
	 * Called when a specific set of streaming levels need to be refreshed
```

```
	 * @param LevelsToRefresh A TArray<ULevelStreaming*> containing pointers to the levels to refresh
	 */
	void RefreshStreamingLevels( const TArray<class ULevelStreaming*>& InLevelsToRefresh );

	void IssueEditorLoadWarnings();

#endif

	/**
	 * Jumps the server to new level.  If bAbsolute is true and we are using seemless traveling, we
	 * will do an absolute travel (URL will be flushed).
	 *
	 * @param URL the URL that we are traveling to
	 * @param bAbsolute whether we are using relative or absolute travel
	 * @param bShouldSkipGameNotify whether to notify the clients/game or not
	 */
	virtual bool ServerTravel(const FString& InURL, bool bAbsolute = false, bool bShouldSkipGameNotify = false);

	/** seamlessly travels to the given URL by first loading the entry level in the background,
	 * switching to it, and then loading the specified level. Does not disrupt network communication or disconnect clients.
	 * You may need to implement GameModeBase::GetSeamlessTravelActorList(), PlayerController::GetSeamlessTravelActorList(),
	 * GameModeBase::PostSeamlessTravel(), and/or GameModeBase::HandleSeamlessTravelPlayer() to handle preserving any information
	 * that should be maintained (player teams, etc)
	 * This codepath is designed for worlds that use little or no level streaming and GameModes where the game state
	 * is reset/reloaded when transitioning. (like UT)
```

```
	 * @param URL - the URL to travel to; must be on the same server as the current URL
	 * @param bAbsolute (opt) - if true, URL is absolute, otherwise relative
	 * @param MapPackageGuid (opt) - the GUID of the map package to travel to - this is used to find the file when it has been auto-downloaded,
	 *                               so it is only needed for clients
	 */
	void SeamlessTravel(const FString& InURL, bool bAbsolute = false, FGuid MapPackageGuid = FGuid());

	/** @return whether we're currently in a seamless transition */
	bool IsInSeamlessTravel();

	/** this function allows pausing the seamless travel in the middle,
	 * right before it starts loading the destination (i.e. while in the transition level)
	 * this gives the opportunity to perform any other loading tasks before the final transition
	 * this function has no effect if we have already started loading the destination (you will get a log warning if this is the case)
	 * @param bNowPaused - whether the transition should now be paused
	 */
	void SetSeamlessTravelMidpointPause(bool bNowPaused);

	/** @return the current detail mode, like EDetailMode but can be outside of the range */
	int32 GetDetailMode();

	/** Updates the timer between garbage collection such that at the next opportunity garbage collection will be run. */
	DEPRECATED(4.18, "Call GEngine->ForceGarbageCollection instead")
	void ForceGarbageCollection( bool bFullPurge = false );

	/** asynchronously loads the given levels in preparation for a streaming map transition.
```

```
	* This codepath is designed for worlds that heavily use level streaming and GameModes where the game state should
	* be preserved through a transition.
	* @param LevelNames the names of the level packages to load. LevelNames[0] will be the new persistent (primary) level
	*/
	void PrepareMapChange(const TArray<FName>& LevelNames);

	/** @return true if there's a map change currently in progress */
	bool IsPreparingMapChange();

	/** @return true if there is a map change being prepared, returns whether that change is ready to be committed, otherwise false */
	bool IsMapChangeReady();

	/** cancels pending map change (@note: we can't cancel pending async loads, so this won't immediately free the memory) */
	void CancelPendingMapChange();

	/** actually performs the map transition prepared by PrepareMapChange()
	 * it happens in the next tick to avoid GC issues
	 * if a map change is being prepared but isn't ready yet, the transition code will block until it is
	 * wait until IsMapChangeReady() returns true if this is undesired behavior
	 */
	void CommitMapChange();

	/**
	 * Sets NumLightingUnbuiltObjects to the specified value.  Marks the worldsettings package dirty if the value changed.
	 * @param	InNumLightingUnbuiltObjects		The new value.
```

```
	 */
	void SetMapNeedsLightingFullyRebuilt(int32 InNumLightingUnbuiltObjects, int32 InNumUnbuiltReflectionCaptures);

	/** Returns TimerManager instance for this world. */
	inline FTimerManager& GetTimerManager() const
	{
		return (OwningGameInstance ? OwningGameInstance->GetTimerManager() : *TimerManager);
	}

	/**
	 * Returns LatentActionManager instance, preferring the one allocated by the game instance if a game instance is associated with this.
	 *
	 * This pattern is a little bit of a kludge to allow UWorld clients (for instance, preview world in the Blueprint Editor
	 * to not worry about replacing features from GameInstance. Alternatively we could mandate that they implement a game instance
	 * for their scene.
	 */
	inline FLatentActionManager& GetLatentActionManager()
	{
		return (OwningGameInstance ? OwningGameInstance->GetLatentActionManager() : LatentActionManager);
	}

	/** Sets the owning game instance for this world */
	inline void SetGameInstance(UGameInstance* NewGI)
	{
		OwningGameInstance = NewGI;
```

```
	}
	/** Returns the owning game instance for this world */
	inline UGameInstance* GetGameInstance() const
	{
		return OwningGameInstance;
	}

	/** Returns the OwningGameInstance cast to the template type. */
	template<class T>
	T* GetGameInstance() const
	{
		return Cast<T>(OwningGameInstance);
	}

	/** Returns the OwningGameInstance cast to the template type, asserting that it is of the correct type. */
	template<class T>
	T* GetGameInstanceChecked() const
	{
		return CastChecked<T>(OwningGameInstance);
	}

	/** Retrieves information whether all navigation with this world has been rebuilt */
	bool IsNavigationRebuilt() const;

	/** Request to translate world origin to specified position on next tick */
	void RequestNewWorldOrigin(FIntVector InNewOriginLocation);

	/** Translate world origin to specified position  */
```

```
        bool SetNewWorldOrigin(FIntVector InNewOriginLocation);

        /** Sets world origin at specified position and stream-in all relevant levels */
        void NavigateTo(FIntVector InLocation);

        /** Gets all matinee actors for the current level */
        void GetMatineeActors( TArray<AMatineeActor*>& OutMatineeActors );

        /** Updates all physics constraint actor joint locations.  */
        virtual void UpdateConstraintActors();

        /** Gets all LightMaps and ShadowMaps associated with this world. Specify the level or leave
null for persistent */
        void GetLightMapsAndShadowMaps(ULevel* Level, TArray<UTexture2D*>&
OutLightMapsAndShadowMaps);

public:
        /** Rename this world such that it has the prefix on names for the given PIE Instance ID */
        void RenameToPIEWorld(int32 PIEInstanceID);

        /** Given a level script actor, modify the string such that it points to the correct instance of the
object. For replays. */
        bool RemapCompiledScriptActor(FString& Str) const;

        /** Given a PackageName and a PIE Instance ID return the name of that Package when being
run as a PIE world */
        static FString ConvertToPIEPackageName(const FString& PackageName, int32 PIEInstanceID);

        /** Given a PackageName and a prefix type, get back to the original package name (i.e. the
saved map name) */
```

```
	static FString StripPIEPrefixFromPackageName(const FString& PackageName, const FString& Prefix);

	/** Return the prefix for PIE packages given a PIE Instance ID */
	static FString BuildPIEPackagePrefix(int32 PIEInstanceID);

	/** Given a loaded editor UWorld, duplicate it for play in editor purposes with OwningWorld as the world with the persistent level. */
	static UWorld* DuplicateWorldForPIE(const FString& PackageName, UWorld* OwningWorld);

	/** Given a string, return that string with any PIE prefix removed */
	static FString RemovePIEPrefix(const FString &Source);

	/** Given a package, locate the UWorld contained within if one exists */
	static UWorld* FindWorldInPackage(UPackage* Package);

	/** If the specified package contains a redirector to a UWorld, that UWorld is returned. Otherwise, nullptr is returned. */
	static UWorld* FollowWorldRedirectorInPackage(UPackage* Package, UObjectRedirector** OptionalOutRedirector = nullptr);
};

/** Global UWorld pointer. Use of this pointer should be avoided whenever possible. */
extern ENGINE_API class UWorldProxy GWorld;

/** World delegates */
class ENGINE_API FWorldDelegates
{
public:
```

```
	DECLARE_MULTICAST_DELEGATE_TwoParams(FWorldInitializationEvent, UWorld* /*World*/, const UWorld::InitializationValues /*IVS*/);

	DECLARE_MULTICAST_DELEGATE_ThreeParams(FWorldCleanupEvent, UWorld* /*World*/, bool /*bSessionEnded*/, bool /*bCleanupResources*/);

	DECLARE_MULTICAST_DELEGATE_OneParam(FWorldEvent, UWorld* /*World*/);


	/**
	 * Post UWorld duplicate event.
	 *
	 * Sometimes there is a need to duplicate additional element after
	 * duplicating UWorld. If you do this using this event you need also fill
	 * ReplacementMap and ObjectsToFixReferences in order to properly fix
	 * duplicated objects references.
	 */
	typedef TMap<UObject*, UObject*> FReplacementMap; // Typedef needed so the macro below can properly digest comma in template parameters.
	DECLARE_MULTICAST_DELEGATE_FourParams(FWorldPostDuplicateEvent, UWorld* /*World*/, bool /*bDuplicateForPIE*/, FReplacementMap& /*ReplacementMap*/, TArray<UObject*>& /*ObjectsToFixReferences*/);


#if WITH_EDITOR
	DECLARE_MULTICAST_DELEGATE_FiveParams(FWorldRenameEvent, UWorld* /*World*/, const TCHAR* /*InName*/, UObject* /*NewOuter*/, ERenameFlags /*Flags*/, bool& /*bShouldFailRename*/);
#endif // WITH_EDITOR


	// Delegate type for level change events
	DECLARE_MULTICAST_DELEGATE_TwoParams(FOnLevelChanged, ULevel*, UWorld*);


	// delegate for generating world asset registry tags so project/game scope can add additional tags for filtering levels in their UI, etc
```

```
	DECLARE_MULTICAST_DELEGATE_TwoParams(FWorldGetAssetTags, const UWorld*, TArray<UObject::FAssetRegistryTag>&);

	DECLARE_MULTICAST_DELEGATE_TwoParams(FOnWorldTickStart, ELevelTick, float);
	static FOnWorldTickStart OnWorldTickStart;

	DECLARE_MULTICAST_DELEGATE_ThreeParams(FOnWorldPostActorTick, UWorld* /*World*/, ELevelTick/**Tick Type*/, float/**Delta Seconds*/);
	static FOnWorldPostActorTick OnWorldPostActorTick;

	// Callback for world creation
	static FWorldEvent OnPostWorldCreation;

	// Callback for world initialization (pre)
	static FWorldInitializationEvent OnPreWorldInitialization;

	// Callback for world initialization (post)
	static FWorldInitializationEvent OnPostWorldInitialization;

#if WITH_EDITOR
	// Callback for world rename event (pre)
	static FWorldRenameEvent OnPreWorldRename;
#endif // WITH_EDITOR

	// Post duplication event.
	static FWorldPostDuplicateEvent OnPostDuplicate;

	// Callback for world cleanup start
	static FWorldCleanupEvent OnWorldCleanup;
```

```
	// Callback for world cleanup end
	static FWorldCleanupEvent OnPostWorldCleanup;

	// Callback for world destruction (only called for initialized worlds)
	static FWorldEvent OnPreWorldFinishDestroy;

	// Sent when a ULevel is added to the world via UWorld::AddToWorld
	static FOnLevelChanged			LevelAddedToWorld;

	// Sent when a ULevel is removed from the world via UWorld::RemoveFromWorld or
	// LoadMap (a NULL object means the LoadMap case, because all levels will be
	// removed from the world without a RemoveFromWorld call for each)
	static FOnLevelChanged			LevelRemovedFromWorld;

	// Called after offset was applied to a level
	DECLARE_MULTICAST_DELEGATE_FourParams(FLevelOffsetEvent, ULevel*,  UWorld*, const
FVector&, bool);
	static FLevelOffsetEvent		PostApplyLevelOffset;

	// called by UWorld::GetAssetRegistryTags()
	static FWorldGetAssetTags GetAssetTags;

#if WITH_EDITOR
	// Delegate called when levelscript actions need refreshing
	DECLARE_MULTICAST_DELEGATE_OneParam(FRefreshLevelScriptActionsEvent, UWorld*);

	// Called when changes in the levels require blueprint actions to be refreshed.
	static FRefreshLevelScriptActionsEvent RefreshLevelScriptActions;
```

```
#endif

private:
	FWorldDelegates() {}
};

////////////////////////////////////////////////////////////////////////////
// UWorld inlines:

FORCEINLINE_DEBUGGABLE float UWorld::GetTimeSeconds() const
{
	return TimeSeconds;
}

FORCEINLINE_DEBUGGABLE float UWorld::GetUnpausedTimeSeconds() const
{
	return UnpausedTimeSeconds;
}

FORCEINLINE_DEBUGGABLE float UWorld::GetRealTimeSeconds() const
{
	checkSlow(!IsInActualRenderingThread());
	return RealTimeSeconds;
}

FORCEINLINE_DEBUGGABLE float UWorld::GetAudioTimeSeconds() const
{
	return AudioTimeSeconds;
```

```
}

FORCEINLINE_DEBUGGABLE float UWorld::GetDeltaSeconds() const
{
	return DeltaTimeSeconds;
}

FORCEINLINE_DEBUGGABLE float UWorld::TimeSince(float Time) const
{
	return GetTimeSeconds() - Time;
}

FORCEINLINE_DEBUGGABLE bool UWorld::ComponentOverlapMulti(TArray<struct FOverlapResult>& OutOverlaps, const class UPrimitiveComponent* PrimComp, const FVector& Pos, const FRotator& Rot, const FComponentQueryParams& Params, const FCollisionObjectQueryParams& ObjectQueryParams) const
{
	// Pass through to FQuat version.
	return ComponentOverlapMulti(OutOverlaps, PrimComp, Pos, Rot.Quaternion(), Params, ObjectQueryParams);
}

FORCEINLINE_DEBUGGABLE bool UWorld::ComponentOverlapMultiByChannel(TArray<struct FOverlapResult>& OutOverlaps, const class UPrimitiveComponent* PrimComp, const FVector& Pos, const FRotator& Rot, ECollisionChannel TraceChannel, const FComponentQueryParams& Params /* = FComponentQueryParams::DefaultComponentQueryParams */, const FCollisionObjectQueryParams& ObjectQueryParams/* =FCollisionObjectQueryParams::DefaultObjectQueryParam */) const
{
	// Pass through to FQuat version.
	return ComponentOverlapMultiByChannel(OutOverlaps, PrimComp, Pos, Rot.Quaternion(), TraceChannel, Params);
}
```

```
FORCEINLINE_DEBUGGABLE bool UWorld::ComponentSweepMulti(TArray<struct FHitResult>& OutHits, class UPrimitiveComponent* PrimComp, const FVector& Start, const FVector& End, const FRotator& Rot, const FComponentQueryParams& Params) const
{
	// Pass through to FQuat version.
	return ComponentSweepMulti(OutHits, PrimComp, Start, End, Rot.Quaternion(), Params);
}

FORCEINLINE_DEBUGGABLE ENetMode UWorld::GetNetMode() const
{
	// IsRunningDedicatedServer() is a compile-time check in optimized non-editor builds.
	if (IsRunningDedicatedServer())
	{
	return NM_DedicatedServer;
	}

	return InternalGetNetMode();
}

FORCEINLINE_DEBUGGABLE bool UWorld::IsNetMode(ENetMode Mode) const
{
#if UE_EDITOR
	// Editor builds are special because of PIE, which can run a dedicated server without the app running with -server.
	return GetNetMode() == Mode;
#else
	// IsRunningDedicatedServer() is a compile-time check in optimized non-editor builds.
	if (Mode == NM_DedicatedServer)
	{
```

```
        return IsRunningDedicatedServer();
        }
        else
        {
        return !IsRunningDedicatedServer() && (InternalGetNetMode() == Mode);
        }
#endif
}
```