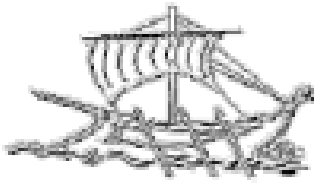


ΑΕΙ ΠΕΙΡΑΙΑ Τ.Τ.
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ Τ.Ε.

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Σχεδίαση και Εφαρμογή Βασικών Χαρακτηριστικών Τεχνητής Νοημοσύνης σε Περιβάλλον Παιχνιδιού

Ανδρέου Φρίξος

Εισηγητής: Δρ Παναγιώτης Γιαννακόπουλος

ΑΘΗΝΑ
ΙΟΥΝΙΟΣ 2016

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Σχεδίαση και Εφαρμογή Βασικών Χαρακτηριστικών Τεχνητής Νοημοσύνης σε Περιβάλλον Παιχνιδιού

Ανδρέου Φρίξος
Α.Μ. 39441

Εισηγητής:

Δρ. Παναγιώτης Γιαννακόπουλος

Εξεταστική Επιτροπή:

Γιαννακόπουλος Παναγιώτης, Καθηγητής
Ζαχάρης Νικόλαος, Αναπληρωτής Καθηγητής Εφαρμογών
Φατούρος Σταύρος, Επίκουρος Καθηγητής

Ημερομηνία εξέτασης:

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

Ο/Η κάτωθι υπογεγραμμένος/η ……, του ……, με αριθμό μητρώου …….. φοιτητής/τρια του Τμήματος Μηχανικών Η/Υ Συστημάτων Τ.Ε. του Α.Ε.Ι. Πειραιά Τ.Τ. πριν αναλάβω την εκπόνηση της Πτυχιακής Εργασίας μου, δηλώνω ότι ενημερώθηκα για τα παρακάτω:

«Η Πτυχιακή Εργασία (Π.Ε.) αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο του συγγραφέα, όσο και του Ιδρύματος και θα πρέπει να έχει μοναδικό χαρακτήρα και πρωτότυπο περιεχόμενο.

Απαγορεύεται αυστηρά οποιοδήποτε κομμάτι κειμένου της να εμφανίζεται αυτούσιο ή μεταφρασμένο από κάποια άλλη δημοσιευμένη πηγή. Κάθε τέτοια πράξη αποτελεί προϊόν λογοκλοπής και εγείρει θέμα Ηθικής Τάξης για τα πνευματικά δικαιώματα του άλλου συγγραφέα. Αποκλειστικός υπεύθυνος είναι ο συγγραφέας της Π.Ε., ο οποίος φέρει και την ευθύνη των συνεπειών, ποινικών και άλλων, αυτής της πράξης.

Πέραν των όποιων ποινικών ευθυνών του συγγραφέα σε περίπτωση που το Ίδρυμα του έχει απονείμει Πτυχίο, αυτό ανακαλείται με απόφαση της Συνέλευσης του Τμήματος. Η Συνέλευση του Τμήματος με νέα απόφασης της, μετά από αίτηση του ενδιαφερόμενου, του αναθέτει εκ νέου την εκπόνηση της Π.Ε. με άλλο θέμα και διαφορετικό επιβλέποντα καθηγητή. Η εκπόνηση της εν λόγω Π.Ε. πρέπει να ολοκληρωθεί εντός τουλάχιστον ενός ημερολογιακού 6μήνου από την ημερομηνία ανάθεσης της. Κατά τα λοιπά εφαρμόζονται τα προβλεπόμενα στο άρθρο 18, παρ. 5 του ισχύοντος Εσωτερικού Κανονισμού.»

ΕΥΧΑΡΙΣΤΙΕΣ

Η παρούσα πτυχιακή εργασία ολοκληρώθηκε μετά από επίπονες προσπάθειες και εκτενή έρευνα στο εν λόγω αντικείμενο τόσο σε θεωρητικό όσο και σε πρακτικό επίπεδο. Θα ήθελα να ευχαριστήσω πρωτίστως τον εισηγητή και επιβλέπων καθηγητή μου, κ. Παναγιώτη Γιαννακόπουλο, για την υποστήριξη που μου παρείχε στην επιλογή του θέματος αλλά και για την πολύτιμη βοήθεια κατά την διάρκεια της εργασίας μου. Επίσης θα ήθελα να ευχαριστήσω τον βασικό προγραμματιστή Τ.Ν της Epic *Games Mieszko Zielnski* που μέσα από τις συμβουλές του βοήθησε σημαντικά στην ολοκλήρωση της πτυχιακής εργασίας μου.

Τέλος, θα ήθελα ακόμη να ευχαριστήσω την οικογένεια μου αλλά και τους φίλους μου για όλη την υποστήριξη και εμπιστοσύνη που μου έδειξαν κατά την διάρκεια των σπουδών μου.

ΠΕΡΙΛΗΨΗ

Η παρούσα πτυχιακή εργασία ασχολείται με την σχεδίαση, ανάπτυξη και εφαρμογή των βασικών αισθήσεων του ανθρώπου όπως η όραση και η ακοή σε ένα περιβάλλον παιχνιδιού με στόχο μια όσο το πιο δυνατόν ρεαλιστική αντίδραση από την αντίθεση που θα βρει ο παίχτης σε ένα παιχνίδι. Τα παραπάνω δημιουργήθηκαν βάσει γνωστών ιστορικών πηγών και αλγορίθμων που έχουν παρουσιαστεί στον τομέα της τεχνητής νοημοσύνης. Το πρακτικό κομμάτι ολοκληρώθηκε εξ ολοκλήρου στην παιχνιδομηχανή *Unreal Engine 4*. Αξιοσημείωτο εργαλείο για την επιτυχή ολοκλήρωση της υπήρξε και το εργαλείο *Blueprint (Visual Scripting Language)* το οποίο αποτελεί μία σαφή καινοτομία της μηχανής.

ABSTRACT

The present thesis concerns the designing, development and implementation of a human's basic senses such as vision and hearing in a game environment in order to accomplish a realistic behavior from the opposition which a player may encounter. All of the above were created through referencing widely known historic sources and algorithms. The practical part of this thesis was completed exclusively in the Unreal Engine 4 game engine. A key feature for the successful completion of this thesis was a tool name Blueprint (Visual Scripting Language) which constitutes a clear innovation of this particular engine.

ΕΠΙΣΤΗΜΟΝΙΚΗ ΠΕΡΙΟΧΗ: Τεχνητή Νοημοσύνη για Ψηφιακά Παιχνίδια.

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Αλγόριθμος, Λήψη Αποφάσεων, Αντίληψη, Καταστάσεις, Οντότητα, Τεχνητή Νοημοσύνη, Δέντρο Συμπεριφοράς, Παιχνίδια.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

ΣΥΝΤΟΜΟΓΡΑΦΙΕΣ - ΕΠΕΞΗΓΗΣΗ ΟΡΩΝ

AI,T.N: *Artificial Intelligence*, Τεχνητή Νοημοσύνη.

KME: Κεντρική Μονάδα Επεξεργασίας.

BP: *Blueprint.*

Ram*: Random Access Memory*.

FSM: *Finite State Machines*.

BT*: Behavior Tree* - Δέντρο Συμπεριφοράς.

AAA: *Triple A* - Ένας όρος που χρησιμοποιείται για να δείξει πώς ένα προιόν κατέχει την μέγιστη ποιότητα.

EQS*: Environmental Query System.*

UI: *User Interface.*

[A]Behavior Tree: Αποτελείται από μια πληθώρα κόμβων με τον πρώτο κόμβο να ονομάζεται *root.* Δέχεται τις τιμές του από το *BlackBoard* και εκτελεί τους κόμβους που βρίσκονται κάτω από τον κόμβο root. Έπειτα, βασικά στοιχεία αποτελούν και οι κόμβοι *Selector* και *Sequence* καθώς και τα παρακλάδια*(branches)* που οδηγούν τους κόμβους.

[B]Selector: Εκτελεί τους κόμβους από κάτω του με σειρά προτεραιότητας από τα αριστερά στα δεξιά και θα σταματήσει να εκτελεί τα *branche*s από κάτω του μόλις κάποιο από τα παιδιά - κόμβους του επιτύχουν. Αν τα παιδιά - κόμβους αποτύχουν τότε αποτυγχάνει και ο *Selector*.

[Γ]Sequence*:* Εκτελεί τους κόμβους από τα αριστερά προς τα δεξιά και θα σταματήσει να εκτελείται μόλις κάποια από τα παιδιά - κόμβους του αποτύχει στην εκτέλεση του. Αν όλα τα παιδιά - κόμβοι του επιτύχουν τότε και ο κόμβος *Sequence* επιτυγχάνει.

[Δ]BlackBoard: Αποτελεί ένα μέρος όπου αποθηκεύονται δεδομένα που μπορούν να διαβαστούν και να γραφτούν για σκοπούς λήψης αποφάσεων. Το *BlackBoard* μπορεί να μοιράζεται από μία οντότητα ή από μία ομάδα πολλών οντοτήτων με κύριο σκοπό την ευκολία για έλεγχο στις τιμές τους. Συνήθως συνδέεται άμεσα με το Δέντρο Συμπεριφοράς.

[E]Services:Εισάγονται πάνω από τους κόμβους σε ένα Δέντρο Συμπεριφοράς και εκτελούνται σύμφωνα με την προκαθορισμένη συχνότητα αρκεί να εκτελείται και το ίδιο κλαδί*(branch)* που βρίσκονται. Χρησιμοποιούνται συνήθως για ελέγχους και να αλλάζουν τις τιμές που εμπεριέχονται στο *BlackBoard*.

[Z]Decorators: Ή αλλιώς Συνθήκες εισάγονται σε έναν κόμβο Έργου*(Task)* ή απλό κόμβο με μόνο σκοπό να αναλύσουν αν κάποιο branch μπορεί να εκτελεστεί ή όχι.

***[H]Tasks*:** Είναι οι κόμβοι που ουσιαστικά πραγματοποιούν μία ενέργεια όπως την μετακίνηση μίας οντότητας ή την αλλαγή στις τιμές του *BlackBoard*.

ΚΕΦΑΛΑΙΟ 1

ΕΙΣΑΓΩΓΗ

Σε αυτό το κεφάλαιο αναλύεται το αντικείμενο της πτυχιακής εργασίας και γίνεται μια ιστορική αναδρομή γύρω από τις μεθόδους που έχουν παρουσιαστεί σε αυτό το πεδίο.

Όσων αφορά τον τομέα της Τεχνητής Νοημοσύνης για παιχνίδια έχουν υπάρξει πολλές αναφορές σε μία πληθώρα τεχνικών για το πώς είναι δυνατόν να εφαρμοστούν σωστά. Τα τελευταία χρόνια λοιπόν έχουν συγγραφτεί πολλά βιβλία εκ των οποίων ένα αποτέλεσε το "κλειδί" για την ολοκλήρωση της εργασίας. Το βιβλίο αυτό ονομάζεται *"AI Game Programming Wisdom"* (5) και παραθέτει μία συμπτυγμένη έκδοση διαφόρων τεχνικών από αρκετούς επαγγελματίες. Πέρα από το συγκεκριμένο βιβλίο όμως εξετάστηκαν και επιμέρους δουλειές γνωστών ανθρώπων στον χώρο για την καλύτερη επεξήγηση και σαφήνεια σε δύσκολα σημεία. Αν και η παρούσα πτυχιακή εργασία υλοποιείται μέσα στην *Unreal Engine* η οποία αποτελεί σχετικά ένα μοντέρνο πρόγραμμα υλοποίησης ψηφιακών παιχνιδιών, η μελέτη της θεωρίας της Τεχνητής Νοημοσύνης κρίθηκε απαραίτητη για την σωστή κατανόηση βασικών ιδεών που επρόκειτο να εφαρμοστούν μέσα στο παιχνίδι μας. Πέρα όμως του πλουσίου υλικού που μπορεί κάποιος εύκολα να ανατρέξει, υπάρχουν αρκετά άρθρα και παρουσιάσεις στο διαδίκτυο από ειδικευόμενους ανθρώπους, όπως ο κ. *Donald Kehoe* (1) πάνω στο θέμα που συνεισέφεραν σημαντικά στο τελικό αποτέλεσμα.

Ο κύριος σκοπός της εργασίας αποτελείται από την σωστή εκμαίευση της θεωρίας της Τ.Ν και η εφαρμογή τους σε μία *Game Engine* όπου υπάρχει λίγη αν όχι ελάχιστη βιβλιογραφία. Τέλος, θα προσπαθήσουμε να δημιουργήσουμε ένα ολοκληρωμένο γενικό σύστημα που θα μπορεί να αποδειχτεί χρήσιμο σε διάφορες επιδείξεις Τ.Ν ψηφιακών παιχνιδιών.

Τα ερευνητικά πεδία που θα εξετάσουμε ξεχωριστά αφορούν:

- Βασικές Αρχές και Γνωστές Τεχνικές Υλοποίησης Τ.Ν για Ψηφιακά Παιχνίδια.
- Σχεδίαση και Υλοποίηση ενός Περιβάλλοντος Παιχνιδιού.
- Σχεδίαση και Υλοποίηση των Οντοτήτων καθώς και Ανάλυση των Εργαλείων της *Unreal Engine*.

1.1 Εισαγωγή στον τομέα της Τεχνητής Νοημοσύνης.

Η τεχνητή νοημοσύνη έχει απασχολήσει εδώ και ένα μεγάλο χρονικό διάστημα πολλά πεδία στην καθημερινή ζωή του ανθρώπου.(2)Στον κόσμο των ψηφιακών παιχνιδιών όμως η Τ.Ν ως χαρακτηριστικό ξεκίνησε να εφαρμόζεται πιο μετά από τα πρώτα παιχνίδια που δημιουργήθηκαν, με μόνο στόχο μια πιστευτή και μοναδική **εμπειρία** για τον παίχτη.(4) Αυτή ακριβώς η ανάγκη, ώθησε τους προγραμματιστές ψηφιακών παιχνιδιών στην μεγάλη πρόοδο που έχει επιτευχθεί μέχρι σήμερα. Παρόλα αυτά, δεν μπορεί να μην γίνει η παρατήρηση για τον μεγάλο διαχωρισμό που έχει επέλθει μεταξύ της ανάπτυξης Τ.Ν σε ακαδημαϊκό επίπεδο με αυτή που παρατηρείται στον κόσμο των ψηφιακών παιχνιδιών.(8)(2) Η μεγάλη διαφοροποίηση βρίσκεται στο γεγονός πώς ο καθένας πλησιάζει το ίδιο πρόβλημα αλλά με διαφορετικό ύφος.(8)(2)

Πριν ξεκινήσουμε όμως να αναφερόμαστε περεταίρω στην διαφοροποίηση που υπάρχει στον επιστημονικό κόσμο καλό θα ήταν να αναλύσουμε τι θεωρείται αρχικά η Τ.Ν για τα παιχνίδια.
Στο πιο βασικό επίπεδο της επίπεδο, η Τ.Ν νοημοσύνη για παιχνίδια αποτελείται κυρίως από το **"φαίνεσθε"** και όχι το **"είναι"**. Δηλαδή, αναφερόμαστε στο να παρουσιάσουμε ως μοναδικό στόχο προς τον παίχτη μια ρεαλιστική συμπεριφορά. Με άλλα λόγια, η Τ.Ν είναι κυρίως "τεχνητή" και λιγότερο "νοημοσύνη" στον κόσμο των παιχνιδιών.(1)
Αντιθέτως, αυτό που πραγματικά παρατηρείται και συμβαίνει είναι σε επιτυχημένα παραδείγματα η σωστή **"μίμηση"** ανθρώπινης συμπεριφοράς αντί μιας πραγματικά έξυπνης αντίδρασης.

Η παραδοσιακή έρευνα επάνω στην Τ.Ν αναζητά την δημιουργία και την ανάλυση μίας νοημοσύνης μέσω τεχνικών μέσων.(1) Ένα από αυτά τα παραδείγματα αποτελεί το *Kismet* του *MIT* το οποίο θέλει να δείξει κάποια συναισθήματα όπως και να αλληλεπιδρά με την κοινωνία.(1)
Όπως ήδη προαναφέρθηκε, ο σκοπός της Τ.Ν για παιχνίδια είναι να λειτουργεί μέσα στα πλαίσια του χώρου - εμπειρίας παιχνιδιού (*gameplay*). Δεν χρειάζεται να

δείχνει συναίσθημα, απλά να προκαλεί ολοένα και περισσότερο τον παίχτη έτσι ώστε αυτός να την ξεπεράσει.(1)

1.2 Ιστορική αναδρομή.

Η ιδέα της Τεχνητής Νοημοσύνης υπήρχε πολύ πριν εδραιωθεί ως όρος στην καθημερινή ζωή των ανθρώπων. Οι ρίζες της Τ.Ν ξεκινούν από την Αρχαία Ελλάδα και την ιστορία της. Τα πρώτα "έξυπνα" αντικείμενα εμφανίστηκαν σε λογοτεχνικά έργα με αναφορά σε συσκευές που εκτελούσαν και "εφάρμοζαν" ένα είδος Νοημοσύνης. (2)

Η Τ.Ν όπως προαναφέρθηκε δεν αποτελούσε χαρακτηριστικό των πρώτων παιχνιδιών. Το πρώτο παιχνίδι που φτιάχτηκε ποτέ ήταν το 1958 από τον *William Higginbotham* που εργαζόταν στο *Brookhaven National Laboratory*.(11)(2) Το παιχνίδι ονομαζόταν *"Tennis for Two"* και δημιουργήθηκε όταν συνέδεσε έναν παλμογράφο με έναν αναλογικό υπολογιστή.(11) "Το πρώτο παιχνίδι που "έτρεξε" σε υπολογιστή ήταν το *"Spacewar"* από τον *Steve Russell* του *MIT*. Αποτελούσε από πολλούς ίσως το πρώτο παιχνίδι που φτιάχτηκε για υπολογιστή καθώς δημιουργήθηκε για να εκτελείται στον PDP-1, μεγάλο υπολογιστικό σύστημα.(*mainframe computer*). (11)(2)

Όχι πολύ μετά το 1970 και το *"Computer Space"* της *ATARI*, οι σχεδιαστές παιχνιδιών (*game designers*) ξεκίνησαν να κατανοούν την ιδέα της Τ.Ν και πώς αυτή μπορεί να εφαρμοστεί μέσα στα παιχνίδια της εποχής.(11)(2)

Εκείνη την περίοδο είχαν κατακτήσει τον κόσμο τα *Arcade* παιχνίδια με τα νομίσματα. Παιχνίδια σαν τα ευρέως γνωστά *Pong*, *Space Invaders* και *Donkey Kong* παρείχαν έναν στοιχειώδη ανταγωνισμό για τον παίχτη.(4) Αλλά από αυτά τα παιχνίδια έλειπαν βασικά θεμέλια και κανόνες της Τ.Ν. Το κυριότερο ίσως ήταν η έλλειψη της **λήψεως αποφάσεων** από τους εχθρούς. Μερικές φορές υπήρχαν και οι εξαιρέσεις έτσι ώστε η συμπεριφορά των αντιπάλων να φανεί απρόβλεπτη στα μάτια του παίχτη, αλλά ο πυρήνας των εν λόγω παιχνιδιών ήταν οι απλοί κανόνες και οι ήδη "φτιαγμένοι" κώδικες και γεγονότα (*pre scripted code and events*).(4)(2) Η αρχική Τ.Ν δεν ήταν κάτι παραπάνω από **αποθηκευμένα πρότυπα** (*stored patterns*) που ξεγελούσαν το ανθρώπινο μάτι κάνοντας το να πιστεύει πώς αντιμετωπίζει έναν πραγματικό αντίπαλο.(4) Παρόμοιο παράδειγμα

αποτέλεσε και το *Pac Man* καθώς κάθε φάντασμα - εχθρός του παίχτη είχε έναν συγκεκριμένο στόχο όπως π.χ. να κυνηγάει τον παίχτη ή να τον αποφεύγει. Πολύ συχνά όμως ο παίχτης υπέπιπτε στην ψευδαίσθηση πώς είχε να αντιμετωπίσει 4 - 5 διαφορετικούς εχθρούς.(6)

Καθώς όμως η τάση για τα παιχνίδια τύπου *arcade* άρχισε να ξεθωριάζει γύρω στα τέλη του 1980 η ανάπτυξη των Η/Υ έπαιξε καθοριστικό ρόλο στην κατεύθυνση που θα έπαιρνε και η ανάπτυξη των ψηφιακών παιχνιδιών. Σύντομα ο προσωπικός Η/Υ έγινε διαθέσιμος και έπειτα δημιουργήθηκε περιεχόμενο παιχνιδιών βασισμένο σε άλλα πράγματα όπως τα γραφικά. (4)(2)

Όμως η Τ.Ν αποτέλεσε και με την σειρά της ένας ολόκληρος τομέας που έπρεπε να εξελιχθεί. Η ανάγκη ήταν μεγάλη, τόσο μεγάλη που εδραίωσε την ιδέα της Τ.Ν ως ένα επίσημο χαρακτηριστικό στα παιχνίδια από το παρελθόν έως και σήμερα. Όντας λοιπόν ένα βασικό χαρακτηριστικό, επόμενο ήταν να προχωρήσει και η εξέλιξη της Τ.Ν παραπάνω.

Στο άμεσο παρελθόν εμφανίστηκαν παιχνίδια που ώθησαν σε μεγάλο βαθμό και έθεσαν υψηλά τον πήχη σε θέματα πιστευτής συμπεριφοράς. Μερικά παραδείγματα που "έλαμψαν" ήταν τα παιχνίδια στρατηγικής (*Real Time Strategy - RTS*). Το συγκεκριμένο είδος εμφανίστηκε και αυτό με την σειρά του στα τέλη του 1980 και γρήγορα ταυτίστηκε με τον όρο "ποιοτική Τ.Ν."(4) Οι τεχνικές αυτών των παιχνιδιών ήταν αρκετά απαιτητικές αλλά και τόσο εντυπωσιακές που σύντομα εξελιχτήκαν σε ένα επιστημονικό πεδίο από μόνες τους (2)

Ένα επίσης σημαντικό παράδειγμα αλλά και τεχνολογικό επίτευγμα αποτελεί και το παιχνίδι *The Sims* που διαθέτει πλέον μια δεκαπεντάχρονη εμπειρία στο να προσπαθεί να μιμηθεί την καθημερινή ζωή των ανθρώπων από το να περιτριγυρίζουν μόνοι στον δρόμο έως και το να διακοσμούν το ίδιο τους το σπίτι! Το εν λόγω εγχείρημα προκάλεσε ιδιαίτερη εντύπωση καθώς παρουσιάζει διάφορες πτυχές που μέχρι πριν δεν είχαν εμφανιστεί στον ίδιο βαθμό. Μερικά από αυτά είναι η έννοια της **μνήμης**, μια εξελιγμένη μορφή **λήψης αποφάσεων** αλλά και η γνωστή **Τ.Ν συμπεριφοράς** (*behavioral AI*) που ασχολείται με την αποφυγή εμποδίων ή αντικειμένων.

1.3 Οι δύο πτυχές της Τεχνητής Νοημοσύνης.

Πώς δημιουργήθηκε όμως ένα τέτοιο τεράστιο χάσμα μεταξύ της ακαδημαϊκής Τ.Ν και την ανάπτυξη Τ.Ν για παιχνίδια;
Η απάντηση βρίσκεται σε **δύο** βασικές παραδοχές:
Πρώτον, η Τ.Ν αποτελεί έναν τεράστιο κλάδο της επιστήμης των Η/Υ. Δεν είναι πάντα εύκολο να εφαρμόσεις τις ίδιες αρχές μέσα σε ένα ψηφιακό παιχνίδι. (8)
Δεύτερον, ο κόσμος και πιο ο συγκεκριμένα ο παίχτης δεν θα ενδιαφερθεί πραγματικά για το τι συμβαίνει πίσω από τα γραφικά και το *gameplay*. (8) Δεν θα αγοράσουν ένα *video game* βάσει των πολύπλοκων αλγορίθμων που χρησιμοποιήθηκαν για να λυθεί ένα μαθηματικό πρόβλημα ορθά.(8)(2) Αυτοί οι σχεδιαστές παιχνιδιών μάλιστα, χρησιμοποίησαν διαφορετική προσέγγιση από αυτή που αναλύεται στον επιστημονικό πεδίο της Τ.Ν. Είναι όμως άξιο ερώτησης να αναρωτηθεί κάποιος αν εκμεταλλεύτηκαν βασικές ιδέες και παραδοχές της ακαδημαϊκής Τ.Ν για να φτάσουν στο επιθυμητό αποτέλεσμα.

Ένα σημαντικό παράδειγμα που μας δίνει μια μικρή οπτική για το πώς ίσως κάποιος θα πρέπει να σχεδιάσει μία Τ.Ν για το παιχνίδι του είναι το ψηφιακό σκάκι. Αν και το ψηφιακό σκάκι αποτελεί έναν φανταστικό τομέα στην εκπαίδευση και στην έρευνα της Τ.Ν, στον κόσμο των παιχνιδιών συμβαίνει το αντίθετο.(8) Θα ήταν καταστροφικό για τον κόσμο των παιχνιδιών αν ο υπολογιστής νικούσε τον μέτριο παίχτη χωρίς καμία δυσκολία. Θα χανόταν η κύρια μορφή ψυχαγωγίας που προσφέρει ένα παιχνίδι, θα χανόταν η **διασκέδαση**, ένας όρος που καθορίζει τι πραγματικά είναι ένα παιχνίδι.(8) Αντίθετα η Τ.Ν στα παιχνίδια είναι εκεί για να **αφήσει** τον παίχτη να νικήσει, αλλά με έναν ψυχαγωγικό τρόπο.(2) Παρόλα αυτά, η υπερβολική επένδυση στο κομμάτι της Τ.Ν έτσι να ώστε να "αγγίξει" τα ακαδημαϊκά επίπεδα είναι ανέφικτη.(2) Από τα παλαιότερα κιόλας χρόνια που η υπολογιστική δύναμη υστερούσε σημαντικά, οι προγραμματιστές "πάλευαν" να χωρέσουν μια γραμμή κώδικα στα παιχνίδια λόγω περιορισμού της μνήμης *(RAM).* (3) (2)
Ακόμη και στην σήμερον ημέρα το ποσοστό των κύκλων ΚΜΕ *(CPU cycles)* είναι εμφανέστατα περιορισμένο για την ανάπτυξη Τ.Ν ακαδημαϊκού επιπέδου. Η Τ.Ν

για παιχνίδια όμως δεν χρειάζεται να έχει **βάθος**.(3)(2) Ένα διαφορετικό παράδειγμα αποτέλεσε και το *Pac Man*. Ο τρόπος με τον οποίο το *Pac Man* χειρίζεται τις διασταυρώσεις στο μονοπάτι των εχθρών είναι να προσθέτει μία δόση τύχης στις αποφάσεις που παίρνουν (*randomness in decision making when encountered with junction*) έτσι ώστε να μην ακολουθούν την ίδια διαδρομή.(6) (2) Αποτελεί έργο των προγραμματιστών να προσθέτουν μια **παραλλαγή** στον ίδιο αλγόριθμο εύρεσης μονοπατιού (*Path-Finding*) για να επιτύχουν μία τέτοια τυχαία συμπεριφορά.(6) (2)

Ακριβώς για αυτόν τον λόγο το επίπεδο της Τ.Ν για παιχνίδια είναι σαφώς κατώτερο από οτιδήποτε έχει καταφέρει μέχρι τώρα η ακαδημαϊκή Τ.Ν.(8)Αλλά με το πέρασμα του χρόνου και την συνεχή ανάπτυξη Τ.Ν για παιχνίδια ολοένα και περισσότερες τεχνικές βασισμένες σε έρευνες από ακαδημαϊκή προέλευση βρίσκουν τον δρόμο τους μέσα στα παιχνίδια.(8) Το γεγονός αυτό αποδεικνύεται καθώς η Τ.Ν παιχνιδιών έχει εκμεταλλευτεί πλήρως βασικές θεωρίες και εργαλεία από την ακαδημαϊκή πτυχή της Τ.Ν. Μερικές από αυτές τις παντοδύναμες μεθόδους και εργαλεία αποτελούν οι:

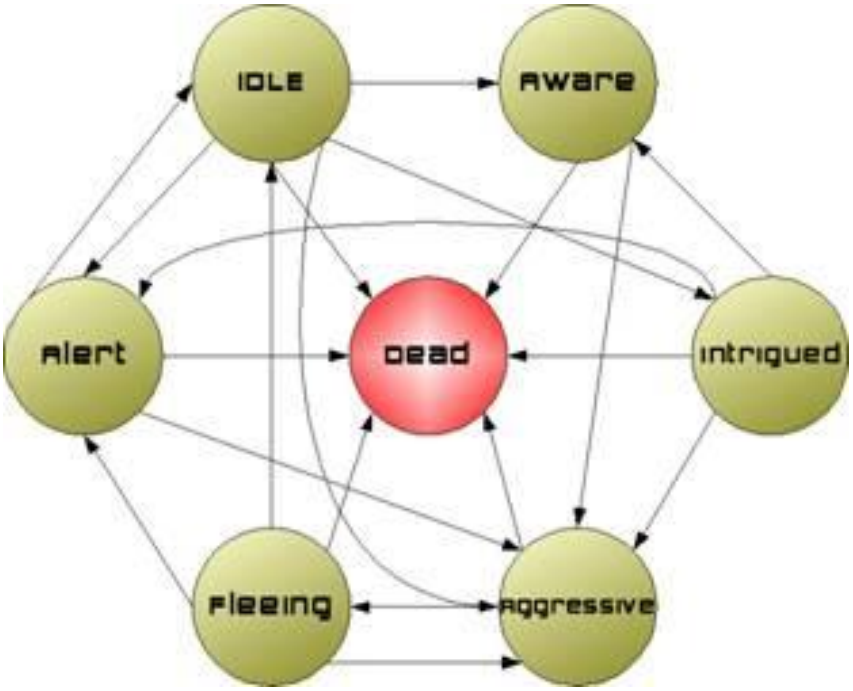
- Μηχανές Πεπερασμένων Καταστάσεων (*FSM - Finite State Machines*)
- Δέντρα Αποφάσεων (*Decision Trees or Behavior Trees*)
- Ασαφής Λογική (*Fuzzy Logic*)

(7)

Μέσω των παραπάνω τεχνικών είναι εύλογο να σκεφτεί κανείς πώς έχει επέλθει η πλέον σημαντική εξέλιξη στον κόσμο της Τ.Ν για παιχνίδια. Οι παραπάνω τεχνικές αποτέλεσαν εργαλεία ζωτικής σημασίας για να ολοκληρωθεί η παρούσα πτυχιακή εργασία και συνεπώς ακολουθεί η ανάλυση δύο από αυτών.

1.4 Μηχανή Πεπερασμένων Καταστάσεων *(FSM).*

Η *FSM* είναι ένας τρόπος σχεδίασης και περάτωσης μιας οντότητας που διακατέχεται από ξεχωριστές **μοναδικές καταστάσεις***(states*) κατά τη διάρκεια της ζωής της.(1) Ειδικότερα, αποτελείται από ένα μαθηματικό μοντέλο που χρησιμοποιείται σε πολλούς κλάδους της επιστήμης των Η/Υ. Μία κατάσταση μπορεί να παρουσιάζει μία "φυσική" αντίδραση που πραγματοποιεί η οντότητα εκτελώντας μέσα στο παιχνίδι μία κίνηση (*animation*). Επίσης μία κατάσταση μπορεί να αναπαριστά κάποια συναισθήματα. Αυτά τα συναισθήματα μπορούν να φανούν στο περιεχόμενο του παιχνιδιού μέσω προκαθορισμένων συμπεριφορών.(*predetermined behaviors*). (1)
Παρακάτω ακολουθεί ένα παράδειγμα με διάφορες καταστάσεις που μπορεί να υπάρξουν σε μία οντότητα ενός παιχνιδιού δράσης.



Εικόνα 1.1: Καταστάσεις μίας Μηχανής Πεπερασμένων Καταστάσεων(*FSM*). (1)

Οι παραπάνω καταστάσεις μπορούν να υποδηλώνουν τα εξής χαρακτηριστικά μέσα στο παιχνίδι:

- Άεργος - *Idle*: Σε αυτήν την κατάσταση η οντότητα παραμένει άεργη. Μπορεί να παραμένει ακίνητη με χαμηλή αντίληψη για το συμβαίνει γύρω του μέχρι να δεχτεί κάποιο έναυσμα.
- Ενήμερος - *Aware*: Η οντότητα συνεχώς ψάχνει για εχθρούς. Εφόσον δεχτεί το έναυσμα όπως π.χ να παρατηρήσει μία ανοιχτή πόρτα, μπορεί να αυξήσει την ακτίνα που ακούει. (1) Περισσότερο από ότι μπορούσε στην άεργη κατάσταση.
- Περίεργος - *Intrigued*: Η οντότητα κατανοεί πώς κάτι δεν πάει καλά. Για να αναπαράγει την συγκεκριμένη συμπεριφορά οπτικά ίσως αφήσει τον προκαθορισμένο μονοπάτι που περπατάει ψάχνοντας για στοιχεία.(1)
- Ειδοποίηση - *Alert*: Η οντότητα έχει καταλάβει τον παίχτη, και τρέχει προς το μέρος του ή ειδοποιεί τους άλλους φρουρούς. Αν τρέξει και φτάσει στην ακτίνα που μπορεί να επιτεθεί στον παίχτη προχωράει στην επόμενη κατάσταση.(1)
- Επιθετικός - *Aggressive*: Εφόσον η οντότητα φτάσει στην ακτίνα του παίχτη ξεκινάει την επίθεση. Η οντότητα αλλάζει κατάσταση αν φτάσει στην κατάσταση "fleeing" που μπορεί να συμβεί όταν φτάσει σε χαμηλούς πόντους ζωής ή αν πεθάνει.(1)
- Φυγή - *Fleeing*: Σε αυτή την κατάσταση η οντότητα τρέχει μέχρι να αποφύγει τον παίχτη και να ξανά επιστρέψει σε μία διαφορετική κατάσταση.(1)
- Νεκρός - *Dead*: Η οντότητα παύει να υπάρχει. Σε συγκεκριμένα παιχνίδια μία τέτοια κατάσταση μπορεί να αποτελέσει έναυσμα για άλλες οντότητες δίπλα της που θα δράσουν ανάλογα.(1)

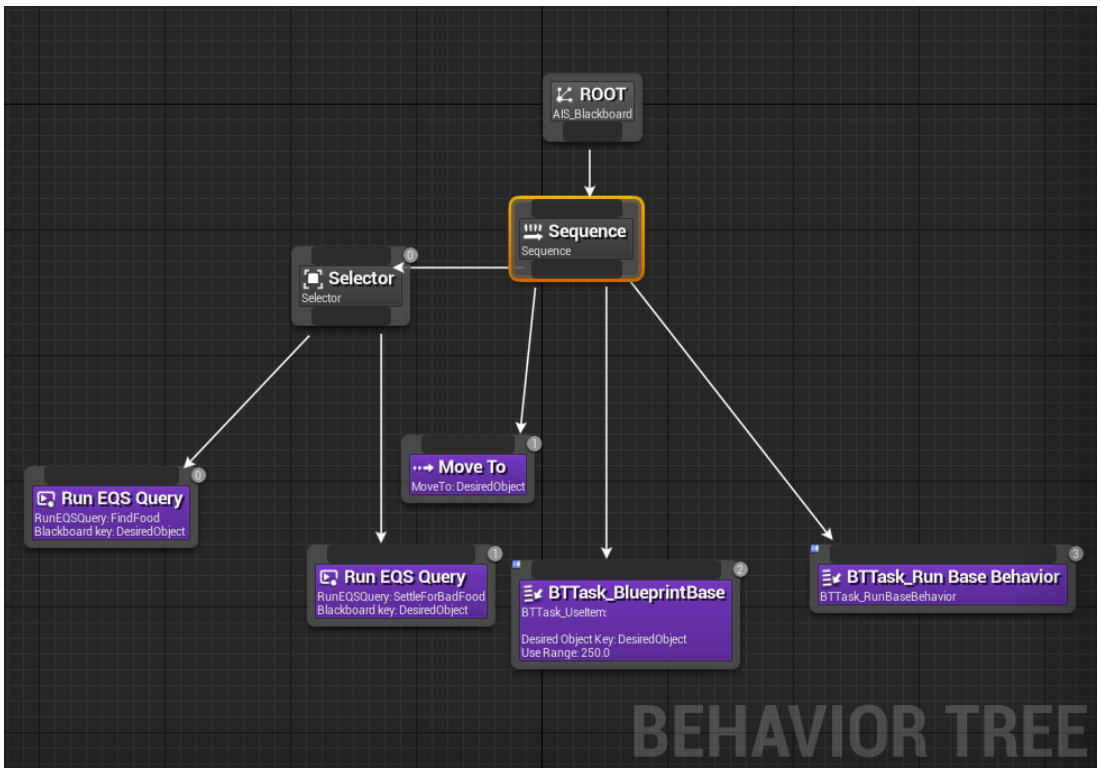
Αυτές οι καταστάσεις μπορούν να εφαρμοστούν μέσω **δύο τρόπων** σε ένα παιχνίδι. Κάθε κατάσταση να συνδέεται με μία μεταβλητή η οποία θα αλλάζει τιμή δεδομένου κάποιων κανόνων ή μέσω άλλου τρόπου αντικειμενοστραφούς προγραμματισμού (*Object Oriented Programming*). (1)

Πριν προχωρήσουμε στο πώς οι *FSM* και τα *Behavior Trees* αντιστοιχούν στην *Unreal Engine* καλό θα ήταν να αναλύσουμε πώς μία οντότητα μπορεί να αντιληφθεί τι συμβαίνει γύρω της. Η απάντηση βρίσκεται στις **αισθήσεις** του ανθρώπινου παράγοντα που θα αναλυθούν στο Κεφάλαιο 1.6.

1.5 Δέντρα Συμπεριφοράς - *Behavior Trees.*

Σε αντίθεση με τις *FSM* τα δέντρα συμπεριφοράς αποτελούνται από κόμβους. Μέσω λοιπόν μίας **ιεραρχικής αρχιτεκτονικής** που τα διακατέχουν, είναι υπεύθυνα για την λήψη αποφάσεων καθώς και συμπεριφορά της οντότητας μας. Κάθε Ιεραρχία κόμβων που παρατηρούμε αποτελεί ουσιαστικά και μία συμπεριφορά που θα εκτελεί η οντότητα μας. Αυτά τα δέντρα αν και στην πρώτη όψη μπορούν να φανούν απλά, πολύ γρήγορα μπορούν να γίνουν αρκετά περίπλοκα. Ανάλογα λοιπόν τι περιμένουμε από την οντότητα μας θα πρέπει να είμαστε σε θέση να στηριχτούμε στα *BT* καθώς είναι τόσο δυνατά που μπορούμε να προσθέσουμε έναν μεγάλο αριθμό συμπεριφορών. Το καλύτερο σημείο όμως αφορά την σχετικά εύκολη εκτέλεση τους. Στην *Unreal Engine* τα *BT* αποτελούν βασικό εργαλείο για την υλοποίηση μίας οντότητας συνεπώς θα είναι συνετό να κρατήσουμε μερικά πράγματα.

Ακολουθεί ένα *BT* μέσα στην *Unreal Engine*.



Εικόνα 1.2: Αναπαράσταση ενός *Behavior Tree.*

Παρατηρώντας λοιπόν την δομή της εικόνας 1.2 παρατηρούμε ένα δέντρο κόμβων. Η αρχή του δέντρου ονομάζεται root και οι κόμβοι ***Selector***[Β] και ***Sequence***[Γ] ονομάζονται *Composite Nodes*. Αυτό το είδος κόμβων τοποθετείται

πάνω από τα ***Tasks*** [H] και επιλέγουν την ιεραρχία που θα τρέξουν. Πέρα από αυτούς τους κόμβους υπάρχουν και άλλοι κόμβοι όπως οι **Decorators**[Z] και οι **Services**[E] που έχουν την δυνατότητα να αλλάξουν την ροή που εκτελείται το δέντρο μας. Αυτοί οι κόμβοι μας επιτρέπουν να "κλειδώνουμε" ορισμένες συμπεριφορές και αντιδράσεις της οντότητας μας.

Ειδικότερα παρατηρούμε πώς αρχικά το δέντρο εκτελείται από τον κόμβο *root* και καθώς κατεβαίνει μέσω των κλαδιών*(branches)* διαλέγει μέσω των κόμβων *Selector* και *Sequence* με ποια σειρά θα εκτελεστούν τα *Tasks*.

Πολύ σημαντική πληροφορία επίσης είναι πώς το *BT* εκτελείται για όσο "τρέχει" το παιχνίδι. Αναλυτικότερα εκτελείται ανά **καρέ(frame)** κατά την διάρκεια του παιχνιδιού και αν η οντότητα προβεί σε μία απόφαση και ενέργεια μπορεί να μεταπηδήσει από τον ένα κόμβο στον άλλο. Τέλος θα πρέπει να κρατήσουμε τις τρεις καταστάσεις που μπορεί να κατέχει ένα δέντρο καθώς εκτελείται. Την επιτυχία, την αποτυχία και την σωστή εκτέλεση του.

1.6 Αντίληψη - *Perception* στην Τεχνητή Νοημοσύνη.

Για να είναι σε θέση μία οντότητα να συμπεριφέρεται ορθά και να λαμβάνει τις κατάλληλες αποφάσεις απαιτείται ένα **σύστημα αντίληψης**.(1) Στην πιο απλή του μορφή ένα τέτοιο σύστημα εμπεριέχει και αναζητά από την οντότητα τις ανάλογες πληροφορίες για την θέση που μπορεί να βρίσκεται ο παίχτης έτσι ώστε να προβεί και στην ανάλογη ενέργεια. Άλλοι πιο πολύπλοκοι κλάδοι της Τ.Ν όμως συναντούν αρκετά εμπόδια στην εκτέλεση τους σχετικά με τα καθήκοντα που πρέπει να πραγματοποιήσει η οντότητα. Ένα απλό παράδειγμα αποτελεί ένα δύσβατο μονοπάτι που πρέπει να διασχίσει η οντότητα για να φτάσει στον παίχτη χωρίς στο τέλος να φτάνει στον προορισμό της.(1) Ο κύριος παράγοντας σε αυτή την περίπτωση θα ήταν ο χάρτης / επίπεδο που δυσκολεύει την δουλειά της οντότητας.

Παρόλα αυτά η πραγματική πρόκληση κρύβεται στην προσαρμογή της οντότητας στην πίστα καθώς και όλα τα στοιχεία που μπορεί να την συνοδεύουν.(1)

Η συγκεκριμένη τεχνική ονομάζεται προσαρμοστική Τ.Ν (*Adaptive A.I*) και εφαρμόζεται πιο συχνά στα παιχνίδια στρατηγικής. Συνήθως αυτό που συμβαίνει

είναι πώς η οντότητα αναλύει τις κινήσεις του παίχτη και δρα ανάλογα, σε πολύ γρήγορο χρονικό διάστημα.

Πέρα από τεχνικές όπως η προσαρμοστική Τ.Ν, η οντότητα πρέπει να είναι σε θέση να "καταλαβαίνει" τι συμβαίνει γύρω της.(1) Χρειάζεται λοιπόν, τις βασικές αισθήσεις του ανθρώπου για να το καταφέρει αυτό. Το θετικό με αυτή την υπόθεση είναι πώς στον κόσμο ενός παιχνιδιού ο δημιουργός - προγραμματιστής έχει πρόσβαση σε οτιδήποτε ο ίδιος έχει φτιάξει. Αυτό μετατρέπει την όλη διαδικασία σε μία πιο απλή παραδοχή από αυτή της πραγματικής Τ.Ν.

Μερικές από τις αισθήσεις και πώς αυτές ίσως θα μπορούσαν να εισαχθούν στο παιχνίδι μας ακολουθούν παρακάτω:

Όραση: Η όραση μπορεί να επιτευχθεί σε προγραμματιστικά επίπεδα μέσω ενός απλού διανύσματος (*Vector*) που ξεκινάει από το μοντέλο της οντότητας και καταλήγει σε μία συγκεκριμένη απόσταση. Αν ο παίχτης εισέλθει στην διάμετρο του διανύσματος τότε μέσω μίας αλλαγής μεταβλητής θα αναγνωριστεί από την οντότητα που με την σειρά της θα προβεί στην ανάλογη ενέργεια, να τρέξει προς το μέρος του, να επιτεθεί κτλ.
Πέρα από το πόσο σημαντικά μπορεί να δει η οντότητα μας θα πρέπει σίγουρα να υπολογίσουμε και την γωνία της όρασης της.(1) Το ανθρώπινο μάτι είναι σε θέση να διακρίνει 62 μοίρες από το αριστερό μάτι και 62 από το δεξί. Με κύριο σκοπό τον ρεαλισμό στο παιχνίδι μας θα ήταν συνετό να χρησιμοποιήσουμε και αντίστοιχες τιμές για την οντότητα μας.
Ένα εξίσου σημαντικό πρόβλημα αποτελεί και το τι ακριβώς θέλουμε να βλέπει και να αναγνωρίζει η οντότητα μας. Θα αντιδρά στην όψη του παίχτη ή και σε κάποιο άλλο αντικείμενο που εισέρχεται στην εμβέλεια της όρασης της; Το συγκεκριμένο ζήτημα μπορεί να αντιμετωπιστεί μέσω της έννοιας της ταμπέλας *(label)* στην πιο απλή μορφή αναγνώρισης που θα κατέχει η οντότητα. Ένα απλό παράδειγμα περιλαμβάνει μία οντότητα, τον παίχτη και ένα δέντρο. Αν εισάγουμε μία ταμπέλα *"Player"* μέσα στις ιδιότητες της οντότητας, θα είναι πλέον σε θέση να "αναγνωρίσει" τι βλέπει απέναντι της. Αν θέλουμε π.χ. η οντότητα μας να εκτελεί μία απλή σκοπιά μεταξύ δύο δέντρων θα εισάγαμε ταμπέλες και για τα δέντρα με

αποτέλεσμα να βλέπει τα δύο αντικείμενα (*game objects*) και να περιφέρεται ανάμεσα τους.

Αναφερόμενος όμως σε απλές μορφές αναγνώρισης του περιβάλλοντος γύρω της μια οντότητα θα πρέπει να είναι σε θέση να μην βλέπει μέσα από τοίχους καθώς κάτι τέτοιο θα οδηγούσε σε μία δυσάρεστη εμπειρία για τον παίχτη. Το πρόβλημα αυτό αντιμετωπίζεται μέσω μίας πολύ γνωστής τεχνικής εν ονόματι ίχνος ακτίνας (*ray tracing*).(1) Το ***ray tracing*** είναι ένας μαθηματικός τρόπος να καταλάβουμε αν η ακτίνα μας συμπίπτει με κάποιο άλλο *game object* σε μία προκαθορισμένη απόσταση.

Ήχος: Ο ήχος θα μπορούσε εύκολα να αντιμετωπιστεί με τους παραπάνω τρόπους. Στην περίπτωση όμως ενός *Vector*, ο ήχος θα απεικονιζόταν διαφορετικά. Στην δίκη μου εργασία επέλεξα αυτό το *vector* να είναι στην ουσία μία σφαίρα που φτάνει μέχρι ένα συγκεκριμένο σημείο. Η οντότητα αποτελεί το κέντρο της σφαίρας και αν κάποιος ήχος ακουστεί μέσα στην εμβέλεια της θα αναγκάσει την οντότητα να προβεί στην ανάλογη ενέργεια. Όπως και στην όραση απαιτείται μία λίστα με τις άλλες οντότητες που θα μπορεί να αναγνωρίσει η δικής μας οντότητα. Επίσης, στην περίπτωση που ο κόσμος μας εμπεριέχει κάποια ηχομονωτικά υλικά θα αξιοποιούσαμε με τον ίδιο τρόπο την τεχνική *ray casting.*

Πέρα από αυτές τις δύο αισθήσεις μία οντότητα μπορεί να αξιοποιήσει και τις άλλες αισθήσεις του ανθρώπου αν οι κανόνες και ο σχεδιασμός του παιχνιδιού το επιτρέπουν:

Αφή: Οι περισσότερες παιχνιδομηχανές (*game engines*) έχουν αντιμετωπίσει αυτό το πρόβλημα μέσω της έννοιας της **σύγκρουσης / τριβής** (*collision*).(1) Αυτό συμβαίνει γιατί οι μοντέρνες *game engines* έχουν χτισμένο μέσα τους ένα απόλυτα ρεαλιστικό σύστημα Φυσικής που εφαρμόζει τους κανόνες της μέσα στο παιχνίδι. Αυτά τα συστήματα αποτελούν έργο τεράστιων εταιριών στον κόσμο της επιστήμης Η/Υ όπως η *Nvidia* με την τεχνολογία *GeForce | PhysX* που έχει ενσωματωθεί σε όλες τις σύγχρονες *game engines*. Συνεπώς αν μία οντότητα συγκρουστεί με μία άλλη θα παρατηρηθεί ένα είδος αφής που μπορεί ο προγραμματιστής να εκμεταλλευτεί για τα επιθυμητά αποτελέσματα.

Όσφρηση: Η ιδέα της μυρωδιάς σε ένα παιχνίδι μπορεί να πραγματοποιηθεί με την προσθήκη ενός αόρατου ίχνους που με την σειρά του θα μπορεί να αναγνωριστεί από τον παίχτη. Λίγα παιχνίδια προσδίδουν τέτοιες ιδιότητες στο *A.I* παρόλα αυτά σίγουρα αποτελεί έναν τρόπο να αποδώσεις ρεαλισμό στο *gameplay*.

Τελειώνοντας με το πώς μία οντότητα μπορεί να αντιληφθεί τι συμβαίνει γύρω της στο περιβάλλον μας απομένει ένα πολύ σημαντικό ερώτημα. Ένα ερώτημα που αποτελεί την βάση της Τ.Ν για παιχνίδια. Πώς μπορεί μία οντότητα που δεν ελέγχεται από τον ανθρώπινο παράγοντα να **μετακινηθεί** στο ίδιο το περιβάλλον που πλέον αντιλαμβάνεται;

Η απάντηση έρχεται από την ακαδημαϊκή έρευνα της Τ.Ν και αλγόριθμους που υπήρξαν εδώ και καιρό.

1.7 Πλοήγηση της Τ.Ν *(A.I Navigation).*

Στον χώρο της πλοήγησης για Τ.Ν έχουν αναπτυχθεί πολλά μοντέλα με εταιρίες παραγωγής ψηφιακών παιχνιδιών να αναπτύσσουν και τα δικά τους μοντέλα για να ταιριάξουν στις ανάγκες του εκάστοτε παιχνιδιού.(10) Μία από αυτές είναι η *Valve* που για την ανάπτυξη ενός από τα πιο δημοφιλή παιχνίδια που φτιάχτηκαν ποτέ, το *Half-Life 2,* δημιούργησε εξ ολοκλήρου το δικό της σύστημα Τ.Ν.

Παρόλα αυτά θα αρκεστούμε με δύο από τις πιο γνωστές τεχνικές πλοήγησης Τ.Ν που υπάρχουν εδώ και αρκετά χρόνια στον χώρο. Αποδείχθηκαν επίσης, **καθοριστικές** για την ολοκλήρωση της παρούσας πτυχιακής εργασίας.

Σύγκρουση και Στροφή (Crash and Turn)

Η τεχνική *crash and turn* αποτελείται από δύο πολύ απλές παραδοχές:

1. Μετακινήσου προς την κατεύθυνση του στόχου σου. (1)

2. Αν συναντήσεις κάποιο εμπόδιο στρίψε προς την κατεύθυνση που σε φέρνει πιο κοντά στον στόχο σου. Αν δεν βρεις καμία τέτοια κατεύθυνση διάλεξε μία στην τύχη και συνέχισε να διαλέγεις μέχρι να φτάσεις στον στόχο σου. (1)

Αυτή η τεχνική έχει "λάμψει" κατά την εξέλιξη των παιχνιδιών καθώς αποτελεί μία πολύ απλή λύση. Ο αριθμός των παιχνιδιών που χρησιμοποιεί αυτή την τεχνική

είναι πολύ μεγάλος.(1) Ο λόγος που χρησιμοποίησα αυτή την τεχνική θα αναλυθεί παρακάτω αλλά ας υποθέσουμε πώς ο στόχος που πρέπει να κινηθεί η οντότητα δεν αφορά τον παίχτη αλλά κάποια αντικείμενα που τον ανταμείβουν.

Ο Αλγόριθμος A* *(A Star Algorithm)*

Ο αλγόριθμος A* βασίζεται επάνω στον γνωστό αλγόριθμο του *Edsger W. Dijkstra*. Κατά καιρούς έχουν παρουσιαστεί πολλές εναλλακτικές λύσεις για τον αλγόριθμο του *Dijkstra* με μία από αυτές να συνθέτει τον αλγόριθμο A*. Στην ουσία υπολογίζει την **γρηγορότερη** διαδρομή μεταξύ κόμβων(*nodes*) σε ένα σχεδιάγραμμα.(9) Η τελική μορφή του A* αποτελείται και από στοιχεία του γνωστού αλλά όχι και τόσο αποτελεσματικού σε κάποιες περιπτώσεις, πρώτου καλύτερου ευρήματος (*Best First Search*).(9) Ως πρώτο βήμα υπολογίζει ποιός κόμβος είναι πιο κοντά. Εφόσον επιλέξει την πρώτη διαδρομή ακολουθεί την ίδια διαδικασία και για τον επόμενο κόμβο μέχρι να φτάσει στον τελικό επιθυμητό κόμβο.
Ας θεωρήσουμε και ορίσουμε τις παρακάτω μεταβλητές για την καλύτερη κατανόηση του A*:

- Η συνάρτηση g(n) αποτελεί το ακριβές κόστος που χρειάζεται για να φτάσει από τον ένα κόμβο στον άλλο.
- Η συνάρτηση h(n) υπολογίζει την ευρετική (*heuristic*) διαδρομή από τον κόμβο n μέχρι τον τελικό κόμβο.

Τέλος πριν η οντότητα μας προχωρήσει στον επόμενο κόμβο εκτελεί μία επανάληψη με την παρακάτω μαθηματική πράξη για να φτάσει στο τέλος με την βέλτιστη διαδρομή. Ακόμα καλύτερα εξετάζει κάθε φορά ποιος κόμβος **n** έχει το μικρότερο αποτέλεσμα της παρακάτω πράξης και με την σειρά προχωράει σε αυτόν:

- $f(n) = g(n) + h(n)$

Με αυτόν τον τρόπο επιτυγχάνεται η υλοποίηση του αλγορίθμου A*.

ΚΕΦΑΛΑΙΟ 2

ΕΙΣΑΓΩΓΗ

Σε αυτό το κεφάλαιο αναλύονται ο σχεδιασμός και οι τεχνικές που χρησιμοποιήθηκαν για την δημιουργία του περιβάλλοντος που θα κινούνται οι οντότητες.

2.1 Σχεδίαση του *Game Environment.*

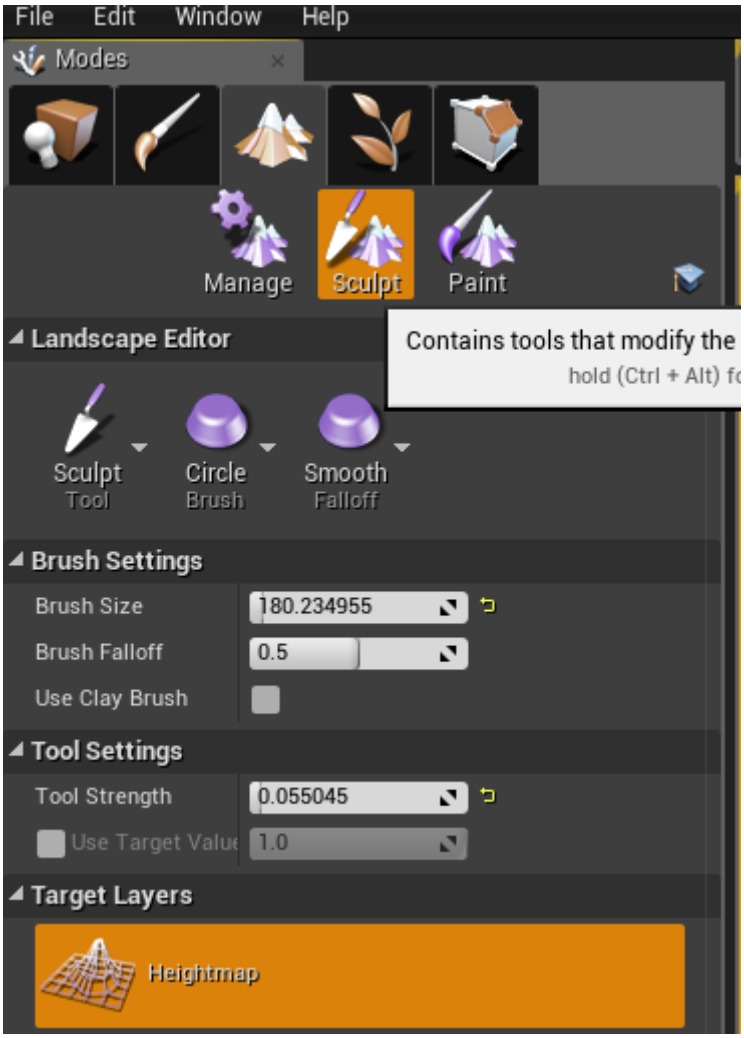
Προχωρώντας στην υλοποίηση της παρούσας πτυχιακής εργασίας ,η ροή εργασίας μου ξεκίνησε με την δημιουργία του χώρου που θα επιδεικνύονταν οι δυνατότητες των οντοτήτων που θα είχα σκοπό να φτιάξω. Συνεπώς το πρώτο βήμα στην εργασία μου ήταν να δημιουργήσω ένα *game environment*.
Το συγκεκριμένο *game environment* πρέπει φυσικά να κατέχει ένα "θέμα", εφόσον μελετούμε τον κόσμο των παιχνιδιών δεν θα πρέπει να χαθεί η εμπειρία που προσφέρουμε στον παίχτη δημιουργώντας μόνο κουτιά και πατώματα.
Η επιλογή μου λοιπόν ήταν να εμπλουτίσω τον χώρο μου με ένα μεσαιωνικό θέμα καθώς αποτελούσε τεράστια πηγή έμπνευσης για μένα στα νεανικά μου χρόνια. Πέρα από το θέμα της πίστας μου, θα ήταν σοφό να χτιστούν δύο διαφορετικοί χώροι για να επιδείξουν τα δύο είδη Τ.Ν που σκοπεύω να δημιουργήσω.
Έτσι στην δημιουργία του *game environment* λειτούργησα με δύο στόχους στο μυαλό μου:

- **Μεσαιωνικό** Ύφος
- **Δύο** διαφορετικούς χώρους για επίδειξη της Τ.Ν

Συνεχίζοντας λοιπόν στο πρακτικό κομμάτι θα αναφέρω τα στάδια που εργάστηκα για να "φέρω" το περιβάλλον μου στην ζωή.

2.2 Γλυπτική στην *Unreal Engine 4*.

Όσο περίεργο και αν μπορεί να ακούγεται όλα τα *game environment* διαθέτουν ως "ραχοκοκαλιά" ένα είδος γλυπτικής. Αναφερόμενος λοιπόν σε αυτή την γλυπτική και στην δημιουργία *3D* μοντέλων για τα παιχνίδια επιστρατεύονται συνήθως επαγγελματίες *Animators* και *3D Artists* για να ολοκληρώσουν αυτό το είδος της δουλειάς. Όπως είναι επόμενο ένα από τα πιο γνωστά εργαλεία σε αυτούς τους τομείς αποτελεί το *3DS Max* της *Autodesk* που εδώ και πολλά χρονιά θεωρείται από τις κορυφαίες εταιρείες στον τομέα της. Παρόλα αυτά η *Unreal Engine* μας παρέχει με ουκ ο λίγα εργαλεία για να δουλέψουμε έτσι ώστε να φτιάξουμε και εμείς ένα *AAA game environment*. Ξεκινώντας λοιπόν ας παρατηρήσουμε μερικές εικόνες μέσα από την *Unreal Engine* και τα *menu* της καθώς και παράλληλα εικόνες από την δική μου ροή εργασίας μέχρι να φτάσω στο τελικό αποτέλεσμα.



Εικόνα 2.2.1: Αρχικό μενού στην *Unreal Engine* για την δημιουργία ενός τοπίου.

Συνεχίζοντας, θα πρέπει να δώσουμε την περισσότερη σημασία στο ***sculpt tool*.** Συνεπώς στο συγκεκριμένο σημείο θα αναλύσουμε μερικά διαφορετικές **λειτουργίες (*modes*)** που διαθέτει που θα μας βοηθήσουνε να φτάσουμε στο επιθυμητό αποτέλεσμα:

***Sculpt*:** Είναι σε θέση να ανυψώσει ένα πάτωμα(*plane*) ή να το κατεβάσει.

***Smooth*:** Λειαίνει τις επιφάνειες για ένα πιο ελκυστικό αποτέλεσμα.

Flatten: Κάνει επίπεδη μία επιφάνεια ασχέτως του ύψους της.

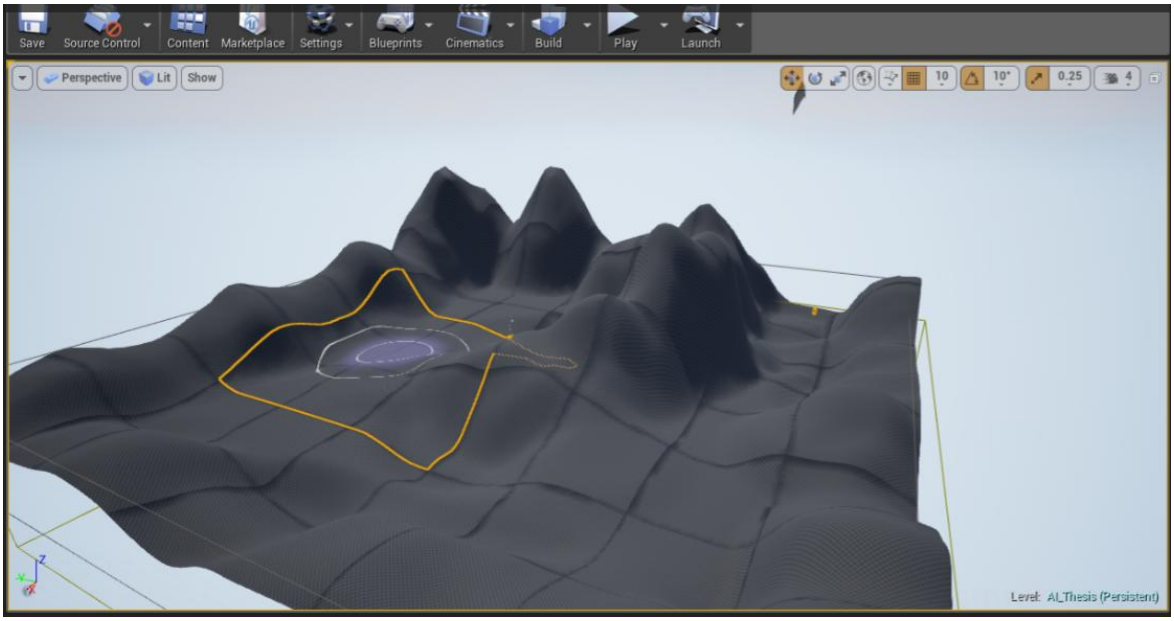
Ramp: Δημιουργεί ένα υψωμένο επίπεδο(*heightmap*) σε μορφή ράμπας.

***Erosion*:** Προκαλεί διάβρωση είτε από ήλιο είτε από νερό για ένα πιο ρεαλιστικό αποτέλεσμα.

Noise: Από τα πιο σημαντικά εργαλεία, προσθέτει θόρυβο στο plane μας ή στο *heightmap* για ρεαλιστισμό.

Εικόνα 2.2.2: Διάφορα *modes* του *sculpt tool*.

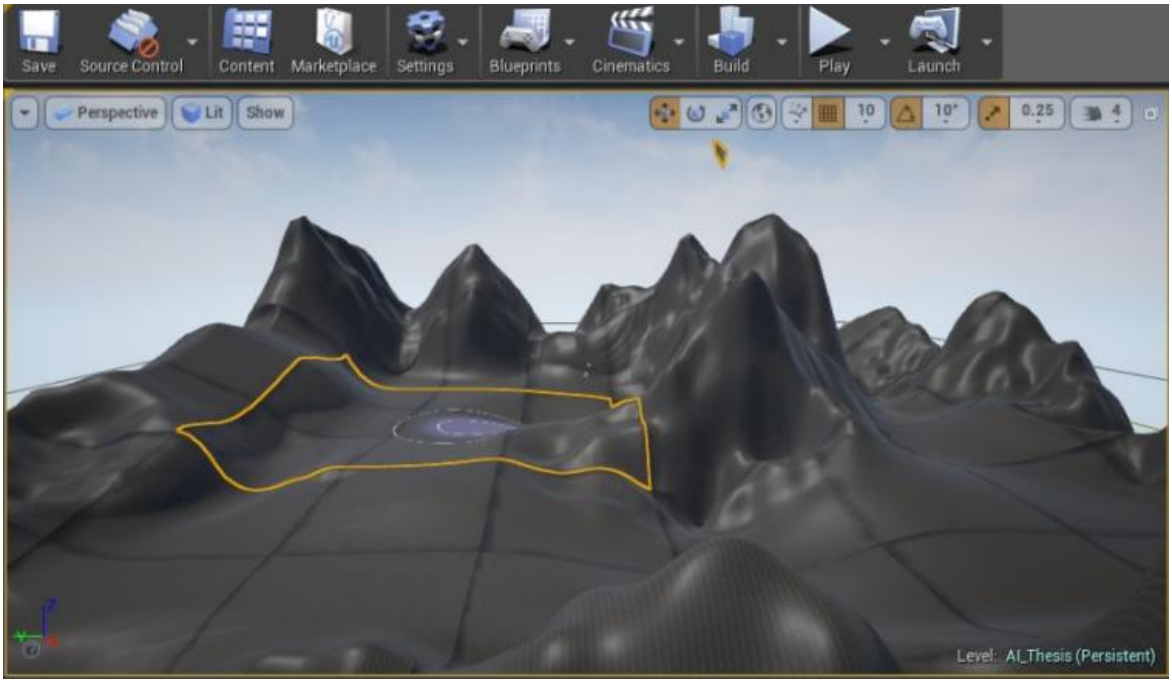
Έχοντας λοιπόν κατανοήσει αυτά τα στοιχεία θα είμαστε σε θέση να καταλάβουμε και την εξέλιξη στο δικό μου τοπίο. Ακολουθεί η εικόνα από την αρχή του *project* μου.



Εικόνα 2.2.3: Αρχικό στάδιο υλοποίησης του *Game Environment (*1/3).

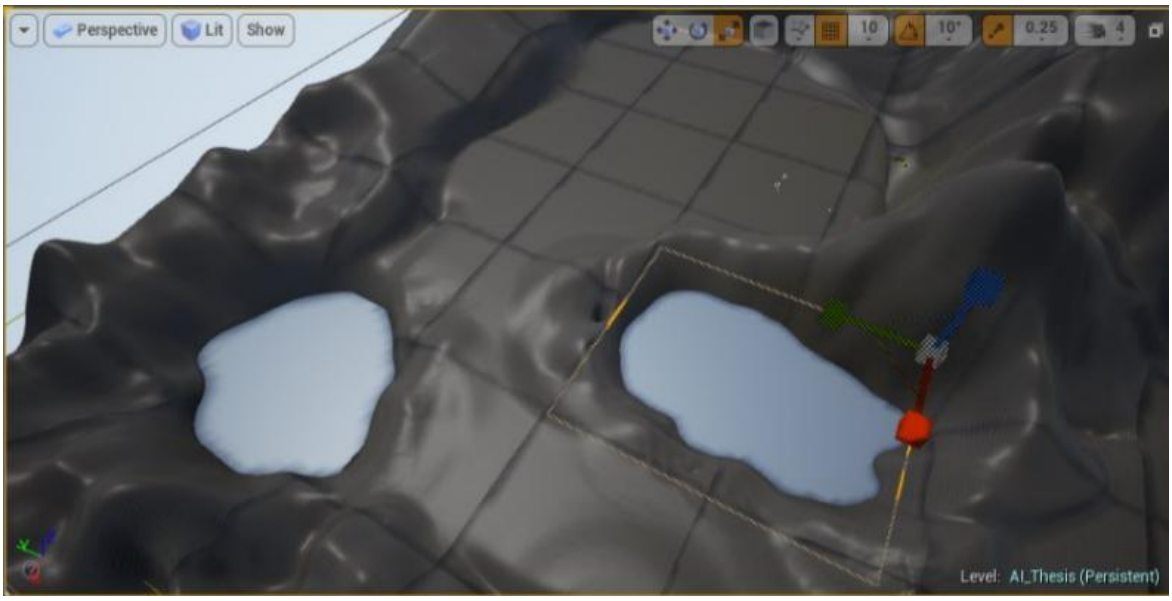
Όπως διακρίνουμε στο παραπάνω σχήμα ξεκινάμε με ένα απλό γκρι τοπίο που σταθερά και αργά ανυψώνουμε(*heightmapping*) το επίπεδο για την δημιουργία μίας οροσειράς που θα περιβάλλει τον δρόμο μας. Η διαδικασία αυτή γίνεται μέσω του ***sculpt tool*** και μίας **βούρτσας(***brush*) που μπορούμε να αλλάξουμε και να μεταποιήσουμε το μέγεθος της.

Συνεχίζοντας, η πιστά μας διαμορφώνεται προς την τελική της μορφή.



Εικόνα 2.2.4: Αρχικό στάδιο υλοποίησης του *Game Environment* (2/3)

Τέλος, τοποθετούμε δύο λίμνες "κατεβάζοντας" το επίπεδο προς τα κάτω με το ίδιο *brush* όπως φαίνεται παρακάτω με κύριο σκοπό την δημιουργία μίας γέφυρας σε εκείνο το σημείο.



Εικόνα 2.2.5: Αρχικό στάδιο υλοποίησης του *Game Environment* (3/3)

Τελειώνοντας με το *sculpting* του τοπίου μας είμαστε σε θέση να προχωρήσουμε στο παρακάτω βήμα που αφορά το **"βάψιμο"** *(paint)* του τοπίου μας. Η επόμενη διαδικασία είναι πολύ χρονοβόρα και απαιτεί ιδιαίτερη προσοχή στην εκτέλεση της.

2.3 Painting a Landscape in Unreal Engine.

Για να καταφέρουμε αρχικά να "βάψουμε" το τοπίο μας πρέπει να διαθέτουμε τα κατάλληλα **υλικά***(materials*) που θα δρουν ως τα χρώματα μας. Παρακάτω ακολουθούν μερικές εικόνες κατά την δημιουργία των υλικών αλλά και η επεξήγηση. Ένα υλικό αποτελείται από **τρία *maps*** τα οποία του προσδίδουν την τελική όψη . Αυτά τα *maps* είναι τα εξής:

- ✓ ***Diffuse Map*:** Αυτό το *map* δίνει το ***χρώμα*** στο υλικό που θέλουμε. Αν θέλαμε να δώσουμε το παράδειγμα της σχεδίασης μίας καφέ πόρτας, το *diffuse map* θα αποτελούταν από ένα καφέ χρώμα.

- ✓ ***Normal Map***: Στην ουσία αν θέλουμε να αποδώσουμε μία **γεωμετρία** στο υλικό μας, το πραγματοποιούμε μέσω του *normal map*. Στο παράδειγμα σχεδίασης μίας τραχιάς πέτρας με πολλές γωνίες θα "περνούσαμε" όλη την πληροφορία των γεωμετρικών εξογκωμάτων μέσα στο *normal map*.

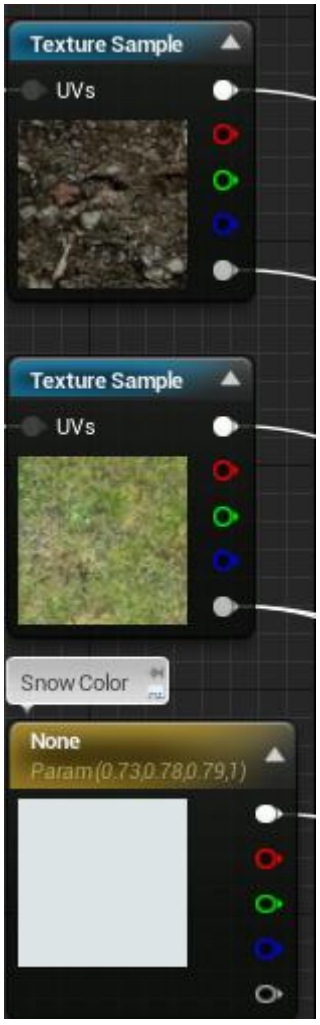
- ✓ ***Specular Map*:** Πρόκειται για την πληροφορία που διακατέχει το πόσο ***γυαλίζει*** και ***φωτίζει*** η επιφάνεια του υλικού μας. Ένα άτοπο παράδειγμα της λάθος χρήσης αυτού του *map* θα ήταν να στην επιφάνεια ενός δέντρου που θέλουμε να τοποθετήσουμε στο παιχνίδι έναν μεγάλο δείκτη φωτεινότητας και γυαλίσματος μέσω του *specular map*. Όπως είναι φυσιολογικό και από την πραγματική ζωή το ξύλο ως υλικό ενός δέντρου δεν έχει την ιδιότητα να γυαλίζει τόσο πολύ υπό το φως του ηλίου αντίθετα με την περίπτωση του αλουμινίου. Θέλει αρκετή προσοχή για να προσδώσουμε τον ρεαλισμό που χρειάζεται.

Η σωστή πρακτική αυτών των *maps* είναι να δημιουργούνται σε εξωτερικά προγράμματα ειδικευόμενα σε αυτές τις διαδικασίες. Το πιο γνωστό εργαλείο σε αυτόν τον τομέα είναι το *Photoshop* . Υπάρχουν επίσης πολλά άλλα ειδών *maps* που τελειοποιούν το οπτικό αποτέλεσμα ενός υλικού αλλά απαιτούν βαθύτατες γνώσεις από τον επαγγελματία που τις χρησιμοποιεί. Τα πιο γνωστά όμως *maps* που δίνουν την τελική μορφή σε ένα υλικό είναι τα τρία που προαναφέρθηκαν.

- Αξίζει να σημειωθεί πώς σε προγράμματα όπως το *3DS Max* το *normal map* της *Unreal Engine* θεωρείται ως *Bump Map* και το *normal* αποτελεί κάτι διαφορετικό. Για λόγους **ευκολίας** μέσα στην *Unreal Engine* τα δύο αυτά *maps* έχουν ενωθεί σε ένα.

Τελειώνοντας με την επεξήγηση των *maps* ακολουθούν μερικές τεχνικές από το *workflow* μου μέχρι να φτάσω στο τελικό αποτέλεσμα καθώς και το αλλαγμένο πλέον τοπίο.

Στην εργασία αποφάσισα να δουλέψω με αρκετά *textures* και χωρίς αυτά να εμπεριέχουν το *specular map*. Ο λόγος πίσω από αυτή την απόφαση είναι πώς τα *textures* μου, μου χρειάστηκαν για να βάψω το έδαφος της πίστας μου και όχι κάποιο αντικείμενο με λεπτομέρεια. Τα ***textures*** στην προκειμένη φάση θα ήταν σωστό να τα θεωρήσουμε ως το **χρώμα** του υλικού μας, δηλαδή τα *diffuse maps*. Τα *textures* που χρησιμοποίησα είναι τα παρακάτω:



- Το πρώτο *diffuse map* του texture αποτελεί ένα είδος **λάσπης** αφού αναφερόμαστε σε εξοχή.

- Το δεύτερο είναι το **γρασίδι**.

- Το τρίτο είναι το **χιόνι** που ήθελα να προσθέσω στις βουνοκορφές.

Εικόνα 2.3.1: *Diffuse Maps* των *Textures.* (1/2)

Στην συνέχεια έχουμε:

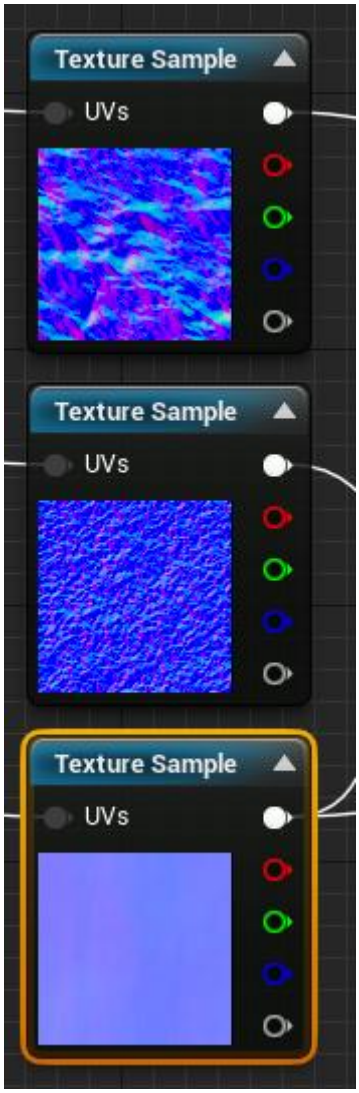
- Ένα *diffuse map texture* που θα χρησιμοποιηθεί για την δημιουργία ενός **χωμάτινου δρόμου**.

- Ένα *diffuse map texture* που θα χρησιμοποιηθεί για την δημιουργία ενός **πλακόστρωτου δρόμου**.

- Τέλος ένα πιο βραχώδες *texture* που θα τοποθετήσουμε κυρίως στα **βουνά**. Όπως είναι ευνόητο αυτό το map θα έχει αρκετές γεωμετρικές διαταραχές.

Εικόνα 2.3.2: *Diffuse Maps* των *Textures.* (2/2)

Φυσικά ακολουθεί και η δημιουργία των *normal maps* όπου θα παραθέσουμε μία απλή αντιστοίχηση της εικόνας 2.3.3.



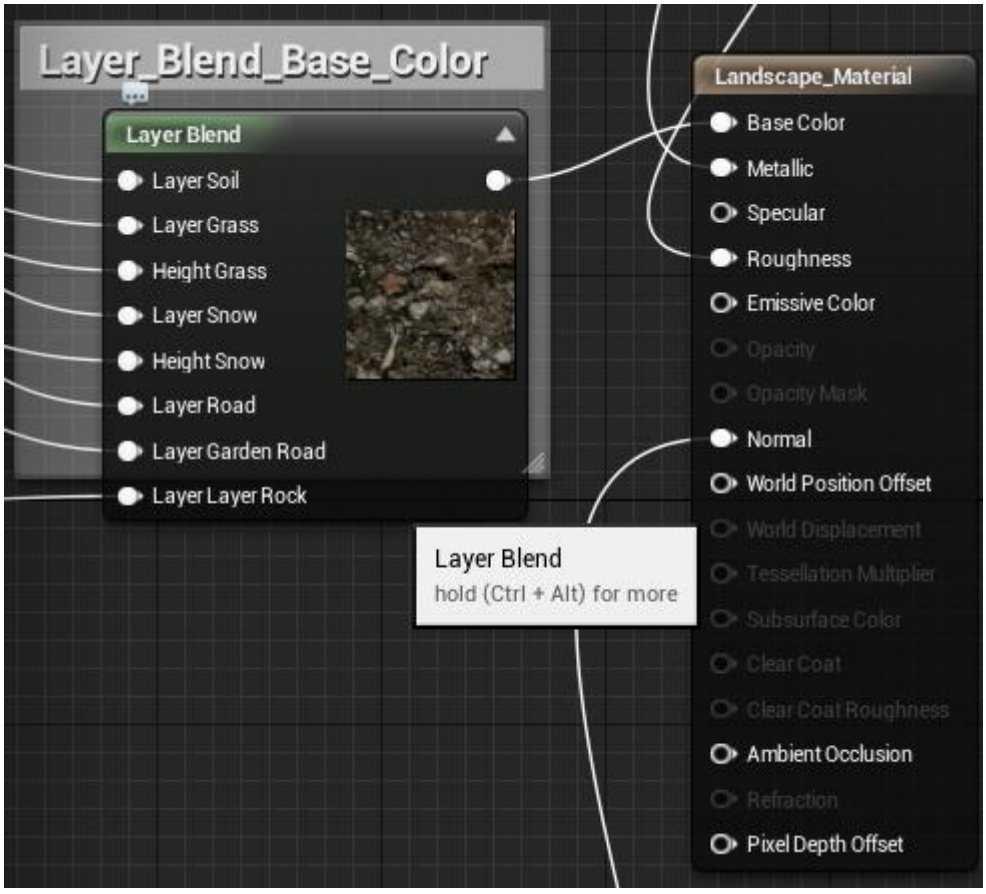
- Το αντίστοιχο *normal map* του *texture* της **λάσπης** που ορίσαμε στην αρχή.

- Το αντίστοιχο *normal map* του texture του **γρασιδιού**.

- Το αντίστοιχο *normal map* του texture του **χιονιού** που όπως φαίνεται δεν θα εμπεριέχει κάποιου είδους γεωμετρική αλλοίωση αλλά θα είναι επίπεδο.

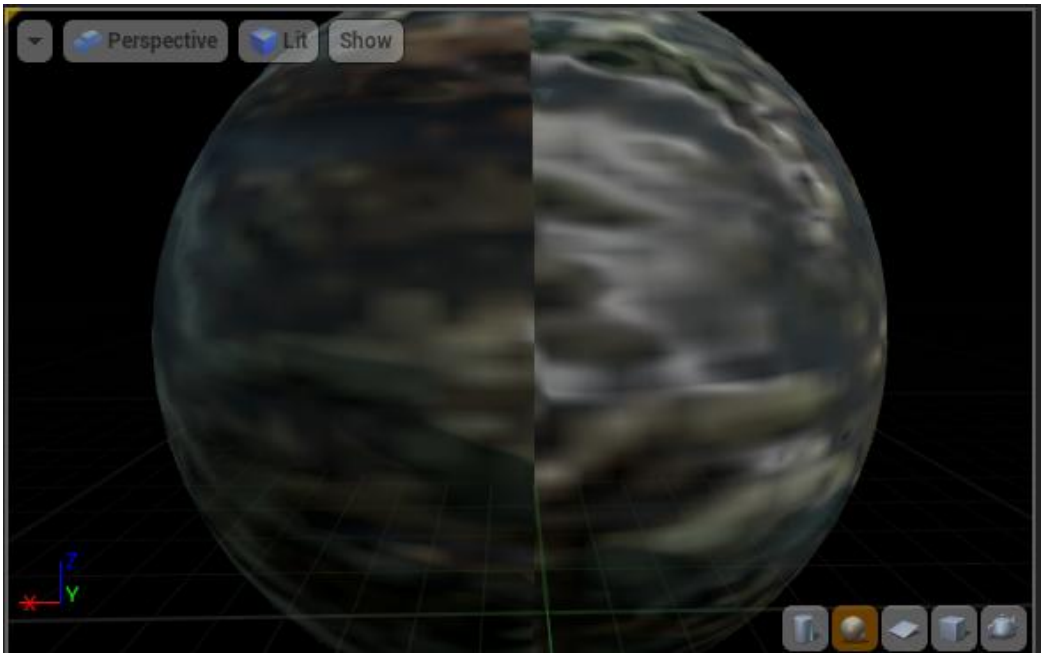
Εικόνα 2.3.3: *Normal Maps* των *Textures*.

Έχοντας στήσει τα *materials* είμαστε πλέον σε θέση να τα ενώσουμε όλα σε ένα είδος πίνακα(*atlas / Layer Blend*) που προσφέρει η *Unreal Engine* για ευκολία κατά την βαφή του τοπίου μας.
Ο τελικός μας πίνακας λοιπόν θα έχει την παρακάτω μορφή

: 

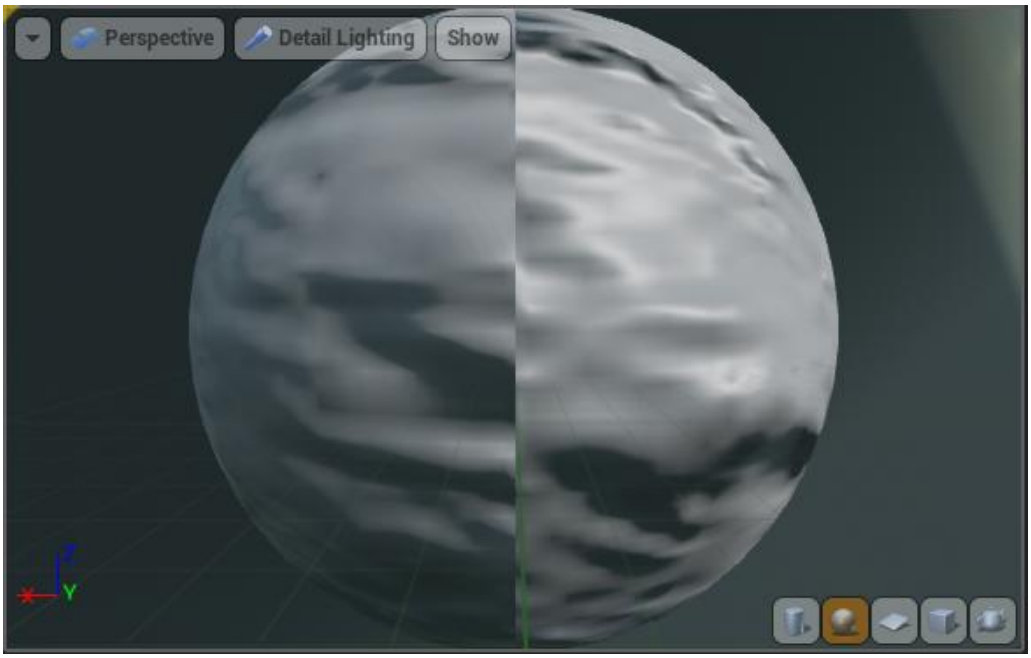
Εικόνα 2.3.4: Πίνακας *Layer Blend* που συνδυάζει όλα τα υλικά μας.

Και παρακάτω ακολουθούν δύο εικόνες που θα μας βοηθήσουν να κατανοήσουμε τι πραγματικά κατασκευάσαμε. Πρώτα θα εξετάσουμε το υλικό μας ως **ολόκληρο**, δηλαδή με την φωτεινότητα, την γεωμετρία και το χρώμα του και έπειτα θα εξετάσουμε μόνο την **γεωμετρία** του.
Η επόμενη εικόνα μπορεί να μας προσδίδει μία διστακτική άποψη στον τρόπο που σχεδιάσαμε και χτίσαμε το υλικό μας αλλά κάτι τέτοιο είναι φυσιολογικό καθώς εδώ πέρα απεικονίζονται **όλα** τα υλικά μας σε μία εικόνα.



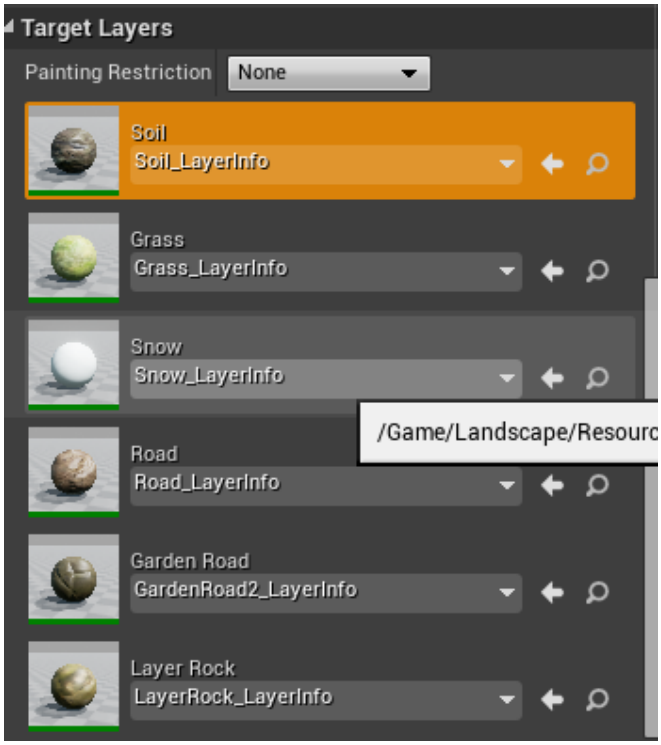
Εικόνα 2.3.5: Οπτική Αναπαράσταση του υλικού μας στην *Unreal Engine*.

Για να παρατηρήσουμε όμως και την εργασία μας επάνω στην γεωμετρία μέσω των *Normal Maps* θα ενεργοποιήσουμε τον κατάλληλο διακόπτη *"Detail Lighting"*.



Εικόνα 2.3.6: Οπτική Αναπαράσταση της γεωμετρίας του υλικού μας στην *Unreal Engine*.

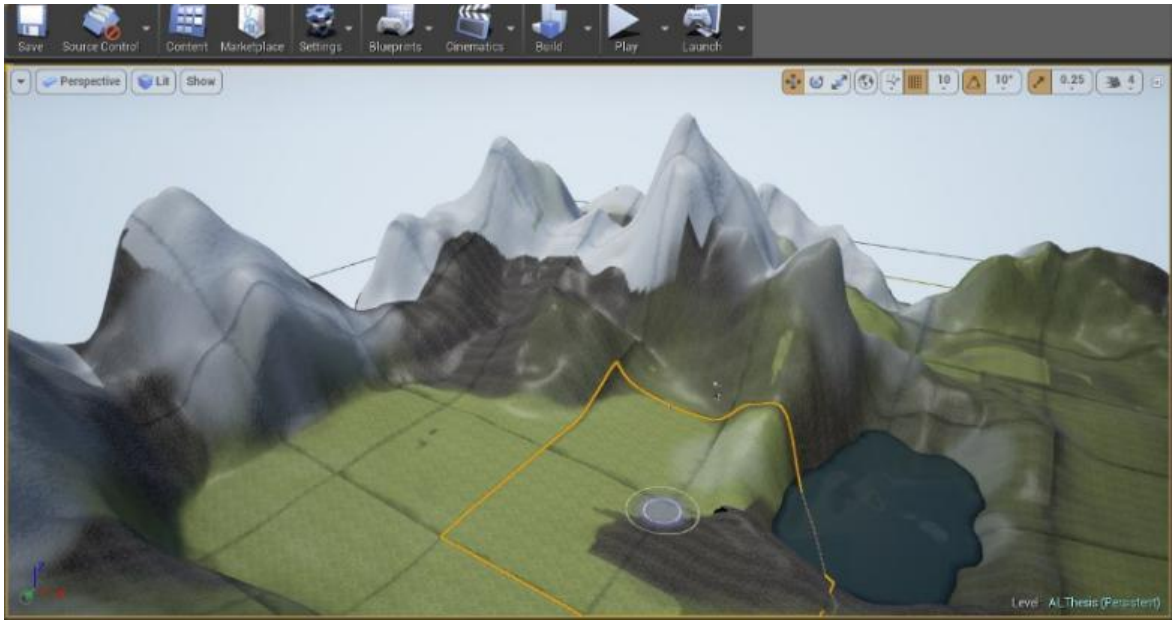
Με την κατάλληλη βαφή πλέον έτοιμη είμαστε σε θέση να ξεκινήσουμε την διαδικασία επάνω στο τοπίο μας.



- Βλέπουμε πώς στο *menu* του εργαλείου *Paint* της *Unreal Engine* έχουν ήδη διαμορφωθεί οι επιλογές που μπορούμε να έχουμε όταν βάφουμε με το εργαλείο *brush*.

Εικόνα 2.3.7: Τα Υλικά ως *Layers* στο εργαλείο *Paint*.

Το τελικό μας αποτέλεσμα θα είναι το παρακάτω μετά το βάψιμο έχει πάρει την εξής μορφή:



Εικόνα 2.3.8: Τοπίο μετά την εφαρμογή των Υλικών.

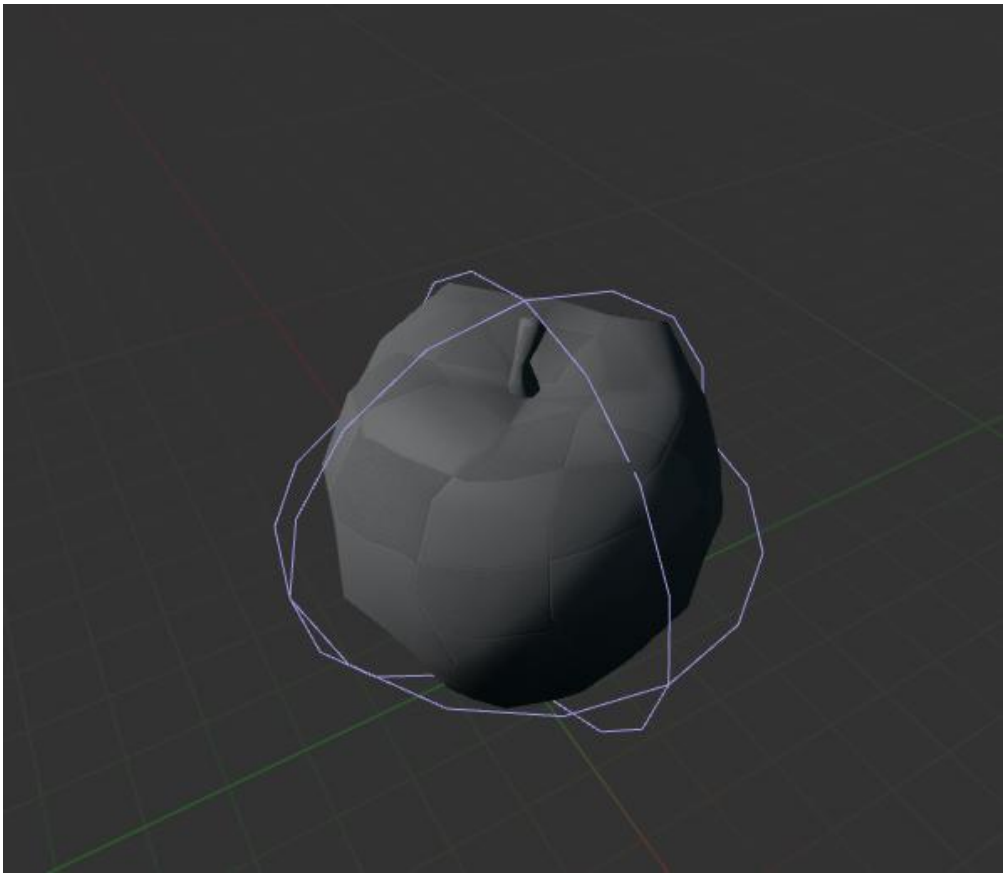
2.4 Προσθήκη Αντικείμενων *(Props)* και τελική υλοποίηση της σκηνής.

Το τελευταίο μας στάδιο αφορά την **προσθήκη αντικειμένων** όπως δέντρα, βράχους, πύργοι και άλλα για να φέρουμε το μεσαιωνικό επίπεδο στο τελικό στάδιο. Τα παρακάτω *3D* μοντέλα βρέθηκαν και χρησιμοποιήθηκαν από το επίσημο *marketplace* της *Unreal Engine*. Διαδίδονται δωρεάν και χωρίς καμία επιβάρυνση για **εκπαιδευτική** ή **εμπορική χρήση**.

- Ο χαρακτήρας αυτών των αντικειμένων είναι καθαρά διακοσμητικός για να "βυθίσουν" τον παίχτη στο μεσαιωνικό ύφος που θέλουμε να επιτύχουμε. Παρόλα αυτά, είναι πολύ σημαντικό να κατανοήσουμε πώς όλα αυτά τα αντικείμενα διαθέτουν κάποιες ιδιότητες. Οποιοδήποτε αντικείμενο μέσα στην μηχανή μπορεί να **κληρονομήσει** ιδιότητες ή να **παραχωρήσει** σε άλλα αντικείμενα τις δικές τους. Η διαδικασία αυτή ορθώς μπορεί να θυμίζει την παρόμοια ιδέα που υπάρχει στις μοντέρνες γλώσσες αντικειμενοστραφούς προγραμματισμού καθώς πάνω σε μία τέτοια γλώσσα βασίζεται και έχει προγραμματιστεί όλη η μηχανή, την C++.

Πριν όμως προχωρήσουμε στην σχεδίαση των οντοτήτων μας καλό είναι να αναγνωρίσουμε από τώρα τα αντικείμενα που "παίζουν" έναν ουσιαστικό ρόλο στην ολοκλήρωση της εργασίας.
Στο πρώτο Κεφάλαιο αναφέραμε την τεχνική ***Crash and Turn***. Για το αντικείμενο που ακολουθεί λοιπόν μέσα στην *Unreal* εφαρμόσαμε αυτή ακριβώς την τεχνική για το πώς θα αντιδρά η οντότητα μας στην όψη της τροφής της.

Ξεκινάμε λοιπόν με το **μήλο**:



Εικόνα 2.4.1: Το αντικείμενο του μήλου στην *Unreal Engine.*

Το εν λόγω *3D* μοντέλο αποτελεί την **τροφή** που θα αναζητά ο χαρακτήρας μας. Μέσω της όρασης που θα έχουμε αποδώσει στην οντότητα μας, θα είναι σε θέση να αναγνωρίζει το μήλο και όταν φτάσει σε μία **κατάσταση πείνας** θα το απορροφήσει. Αξίζει να σημειωθεί πώς το μήλο ανήκει στην ***Good Food Class***, που θα το εκμεταλλευτούμε στο επόμενο κεφαλαίο.

Το ίδιο μοντέλο έχει χρησιμοποιηθεί για δύο σκοπούς:

- **Κόκκινο Μήλο:** Είναι η τροφή που θα αναζητά καθώς θα λύσει το πρόβλημα της πείνας του.
- **Καφέ Μήλο:** Είναι η τροφή που θα αποφεύγει καθώς όχι μόνο δεν θα λύσει το πρόβλημα της πείνας του αλλά θα βλάψει και τη υγεία της οντότητας.

Είναι εξίσου χρήσιμο να αναγνωρίσουμε πώς σε αυτό το σημείο χρησιμοποιώντας ένα ίδιο αντικείμενο αλλάζοντας θεωρητικά μόνο το χρώμα του δημιουργούμε μία οντότητα που είναι σε θέση να **επιλέγει** με τι μπορεί να τραφεί και με τι όχι.

Έχοντας αναλύσει την επιλογή που είναι σε θέση να πραγματοποιήσει πλέον η οντότητα μας προχωράμε στην έννοια της **μνήμης** που διακατέχει την οντότητα μας.

Το επόμενο αντικείμενο είναι ένα δέντρο που με μία προκαθορισμένη συχνότητα θα πετάει(*spawn*) τα μήλα στο έδαφος έτσι ώστε να μπορεί να επιδράσει πάνω τους η οντότητα μας. Το κόλπο στην προκειμένη περίπτωση είναι αν η οντότητα μας έχει χάσει τον δρόμο της και έχει φτάσει στο σημείο πείνας, να καταφέρει να θυμηθεί ακριβώς που υπάρχει η τροφή του. Αυτό επιτυγχάνεται με το να θυμάται που ακριβώς βρίσκεται το αντικείμενο (δέντρο) που δημιουργεί την τροφή του έτσι ώστε να φτάσει στο σημείο που βρίσκεται.

Συνεπώς το δέντρο που θα μπορεί να θυμάται η οντότητας μας είναι το παρακάτω:

Εικόνα 2.4.2: Το αντικείμενο του δέντρου που χαράζεται στην μνήμη της οντότητας μας, στην *Unreal Engine.*

2.5 Προσθήκη φυλλωμάτων και *post process effects.*

Παρακάτω ακολουθούν μερικές διαδικασίες που αξίζει να αναφερθούν καθώς αποδείχτηκαν ιδιαίτερα χρονοβόρες.

Έπειτα λοιπόν από την προσεκτική τοποθέτηση των αντικειμένων μας στην σκηνή, απαιτήθηκε και η χρήση του εργαλείου **φυλλωμάτων**(*foliages*) της *Unreal Engine* που ουσιαστικά μας δίνει την δυνατότητα να προσθέσουμε βλάστηση με έναν πιο **παραγωγικό** τρόπο. Το αποτέλεσμα όπως είναι εμφανές αποδείχθηκε πολύ ευχάριστο στο μάτι:

Εικόνα 2.5.1: *Foliages* στην *Unreal Engine*.

Έπειτα χρειάστηκε και λίγη γνώση επάνω στα *post process effects*.

Τα ***Post Process Effects*** που μας προσφέρει η *Unreal Engine* υπάρχουν έτσι ώστε να προσαρμόσουμε όλη την **αισθητική** της σκηνής μας και του τοπίου μας. Μεταξύ άλλων μας επιτρέπει να μεταποιήσουμε τον φωτισμό, τα χρώματα για να επιτύχουμε τον στόχο μας μέσω πολύ γνωστών τεχνικών όπως είναι το *ambient occlusion* και *tone mapping*. Χωρίς ιδιαίτερες λεπτομέρειες ας παρατηρήσουμε πώς αλλάζει η σκηνή μας πριν και μετά την χρήση των *post process effects*:

Εικόνα 2.5.2: Το τοπίο μας **πριν** τα *Post Process Effects.*

Εικόνα 2.5.3 Το τοπίο μας **μετά** τα *Post Process Effects.*

Τέλος, ακολουθεί μία εικόνα που χαρακτηρίζει πλέον το περιβάλλον μας ως έτοιμο έτσι ώστε να δουλέψουμε στον "πυρήνα" της παρούσα εργασίας που είναι η Τ.Ν.

Εικόνα 2.5.4: Η τελική Σκηνή του τοπίου.

ΚΕΦΑΛΑΙΟ 3

ΕΙΣΑΓΩΓΗ

Σε αυτό το κεφάλαιο ορίζουμε τους στόχους καθώς και το **θεωρητικό υπόβαθρο** που θα χρειαστούν οι οντότητες μας πριν την πρακτική αντιστοίχιση και εφαρμογή του πρακτικού κομματιού μέσα στην *Unreal Engine*.

3.1 Σχεδίαση της Τ.Ν και των Οντοτήτων.

Πριν προχωρήσουμε όμως στο πώς όλα τα παραπάνω μπορούν να εισαχθούν σε μια *game engine* όπως είναι η *Unreal* καλό θα ήταν να κατανοηθεί ορθά το τι ακριβώς περιμένουμε από την Τ.Ν που θα εισάγουμε στο περιβάλλον μας.

Το πρώτο κομμάτι θα αφορά έναν **φρουρό** (*Guard A.I*) που ως οντότητα θα είναι υπεύθυνη για:

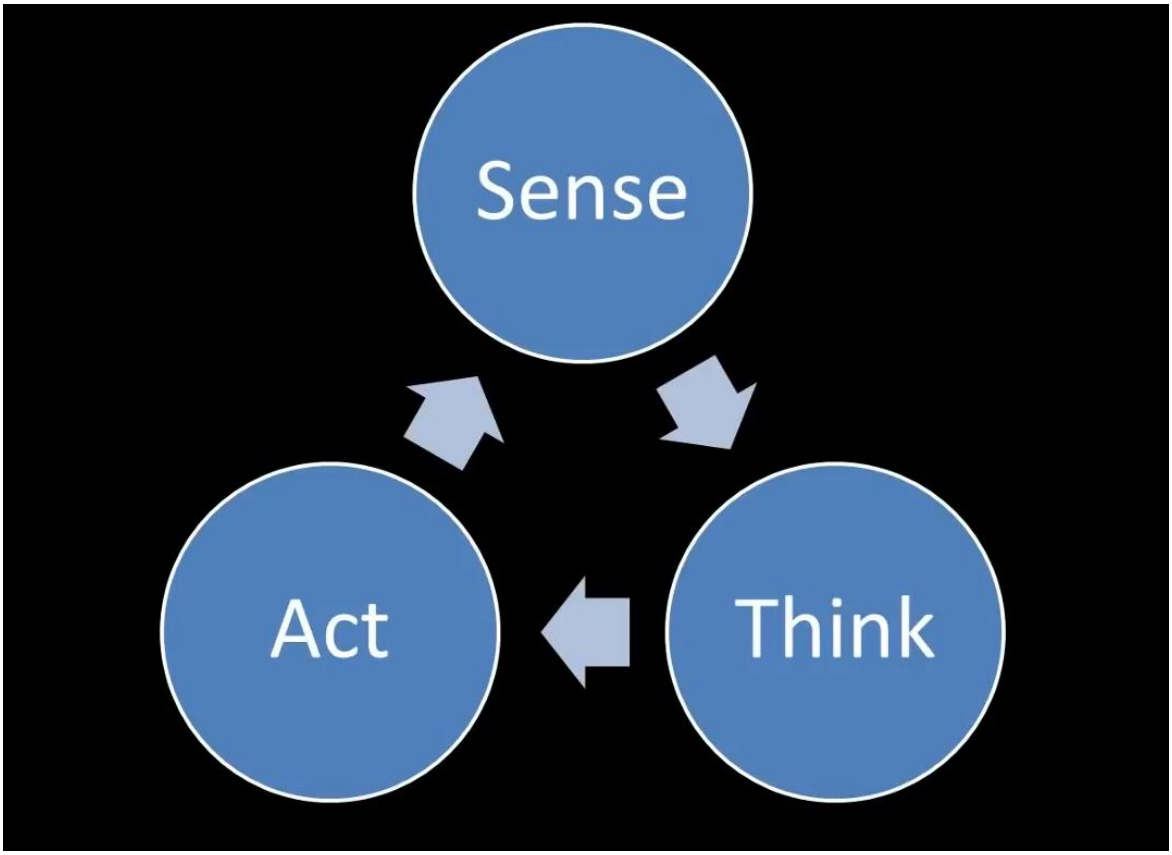
- Την **φύλαξη** μιας προκαθορισμένης θέσης.
- **Αντίδραση** σε οποιαδήποτε κίνηση ή ήχο πραγματοποιήσει ο παίχτης.

Το δεύτερο κομμάτι θα αφορά μία οντότητα που δεν θα απασχολείται από τις πράξεις του παίχτη αλλά αντιθέτως:

- Θα **περιφέρεται τυχαία** γύρω από έναν χώρο με κύριο σκοπό να αναζητά τα εφόδια του, στην περίπτωση μας, ένα είδος τροφής.
- Θα είναι σε θέση εάν έχει ξεφύγει από την πορεία του να **"θυμάται"** σε ποια σημεία θα πρέπει να επιστρέψει για να βρει την τροφή του.
- Θα είναι σε θέση να **ξεχωρίζει** ποια τροφή θα πρέπει να καταναλώσει και ποια να αποφύγει καθώς θα εισάγουμε **δύο ειδών** τροφές έτσι να δώσουμε στην οντότητα μας την αίσθηση της επιλογής.

Έχοντας λοιπόν ορίσει τους στόχους μας ως προς το τι ζητάμε στα πλαίσια της Τ.Ν ήρθε η στιγμή να εξηγήσουμε πώς θα καταφέρουμε να φτάσουμε στο επιθυμητό αποτέλεσμα.

Αρχικά ας μελετήσουμε σε μία πιο απλή μορφή το σύστημα που θα βρίσκεται "πίσω" από την οντότητα μας με το παρακάτω σχήμα:



Εικόνα 3.1: Κυκλική διαδικασία Αντίληψης, Λήψης Απόφασης και Ενέργειας της οντότητας.

Πρώτα λοιπόν η οντότητα μας θα πρέπει να είναι σε θέση να **καταλαβαίνει** τι συμβαίνει γύρω της.
Έπειτα θα πρέπει να είναι σε θέση να **πάρει μία απόφαση** (*decision making*) για το τι θα κάνει στην συνέχεια.
Τέλος θα προβεί σε μία **ενέργεια** σύμφωνα με την απόφαση που πήρε.

Η παραπάνω κυκλική διαδικασία θα εκτελείται σε μία **επανάληψη** κατά την διάρκεια της εκτέλεσης του παιχνιδιού μας.
Αναλυτικότερα έχουμε τα εξής σημεία σε κάθε κλάδο:

Αντίληψη - Αίσθηση (*Sensory*):
Τι γνωρίζουμε για το περιβάλλον παιχνιδιού;

- Τι βρίσκεται γύρω μας;
- Έχουμε ακούσει ή δει κάτι;
- Έχουμε φτάσει στο στάδιο της πείνας;
- Νιώθουμε συμπάθεια για κάποιον ή για κάτι (Επιλογή της Τροφής);

Όλες αυτές οι πληροφορίες πρέπει με κάποιον τρόπο να τις **αποθηκεύσουμε** στην μηχανή μας. Αυτό μπορεί να επιτευχθεί με την γνωστή μέθοδο της αποθήκευσης σε μεταβλητές μέσα σε ένα σύστημα εν ονόματι ***Blackboard*** [Δ].
Εφόσον όμως εκμεταλλευτούμε τις μεταβλητές ως ένα μέσο για την αποθήκευση τιμών θα χρειαστεί να σχεδιάσουμε κάποια ερωτήματα για την οντότητα μας. Μία σωστή αντιμετώπιση που χρησιμοποιούμε για αυτά τα ερωτήματα είναι η απάντηση μέσω ορισμένων **υπηρεσιών** που ονομάζονται ***Services***[Ε] και πραγματοποιούν συγκεκριμένους ελέγχους. Μέσω αυτών των υπηρεσιών είμαστε σε θέση να "αναγκάζουμε" την οντότητα μας να αντιληφθεί κάτι. Υπάρχουν όμως μερικά προβλήματα που αξίζει να σημειωθούν τα οποία προκύπτουν με την χρήση τέτοιων συνθηκών:

- Εάν είναι μεγάλες σε αριθμό τότε απαιτούν αρκετή υπολογιστική δύναμη και δεν αποτελούν τις βέλτιστες επιλογές, παρομοίως με αυτό που συμβαίνει σε γλώσσες όπως η C και η C++.
- Συνήθως αυτές οι υπηρεσίες βασίζονται σχεδόν πάντα στις εσωτερικά προσαρμοσμένες *physics engines*. Κάτι που επίσης δεν αποτελεί πάντα την βέλτιστη λύση.
- Οι υπηρεσίες (*Services*) μπορούν να μην έχουν σταθερή συχνότητα στον τρόπο που λαμβάνουν απαντήσεις και συνεπώς να εμφανιστεί κάποιο ανεπιθύμητο αποτέλεσμα, αν και με την *Unreal Engine* μας δίνεται η δυνατότητα να αλλάξουμε αυτή την συχνότητα.

Σκέψη - Λήψη Αποφάσεων *(Decision Making)*:

Η λήψη αποφάσεων παρά την ίσως παρεξηγημένη φύση της ως στοιχείο της *Unreal Engine* έχει την πιο εύκολη υλοποίηση στο τεχνικό κομμάτι. Μερικά στοιχεία που την απαρτίζουν είναι τα εξής:

- Λογική Απόφασης (*Decision Logic*)
- Απλή υλοποίηση
- Απαιτεί εντατική σχεδίαση

Η συγκεκριμένη διαδικασία χρειάζεται πιο πολύ σκέψη στο σχεδιαστικό κομμάτι. Αυτό συμβαίνει γιατί θέλουμε η οντότητα μας να αναγνωρίζει κάποια στοιχεία ή αντικείμενα μέσα στο παιχνίδι. Δηλαδή αφορά πιο πολύ "λογικά" προβλήματα στον τρόπο που παίζεται το παιχνίδι παρά πώς αυτά κατασκευάζονται. Ένα καλό παράδειγμα αποτελεί η αναγνώριση του παίχτη από την οντότητα και πώς αντιδρά όταν τον βλέπει. Αν δεν έχουμε σκεφτεί ποια απόφαση θα πάρει η οντότητα μόλις συναντήσει τον παίχτη θα έχουμε σοβαρό πρόβλημα στην εμπειρία που προσφέρουμε στον ίδιο τον παίχτη.

Αναφερόμενος όμως στην σχετικά εύκολη υλοποίηση της, η απάντηση βρίσκεται στην **υποθετική λογική**(*conditional logic*). Σύμφωνα με αυτά που γνωρίζουμε για το περιβάλλον που δημιουργούμε αναθέτουμε και κάποιες κατάλληλες συνθήκες. Στις συγκεκριμένες συνθήκες απαντάμε με την γνωστή μέθοδο Σωστού **ή** Λάθους *(True or False)* ή ακόμη και Σωστού **και** Λάθους. Ανάλογα την απάντηση που θα πάρουμε λοιπόν θα προβούμε και στην αντίστοιχη απόφαση που έχουμε θέσει.
Για την απλούστερη κατανόηση θα ήταν σοφό να το παρομοιάσουμε με τις συνθήκες Αν - Αλλιώς *(If - Else)* της γλώσσας δομημένου προγραμματισμού (C) και όχι μόνο. Αυτές οι συνθήκες ονομάζονται **Διακοσμητές** *(Decorators)* [Z] στην *Unreal Engine* και ο στόχος που υπηρετούν είναι να πραγματοποιούν **τακτικά ελέγχους** πριν από την εκτέλεση κάθε απόφασης έτσι ώστε να κριθεί αν θα παρθεί ή όχι.

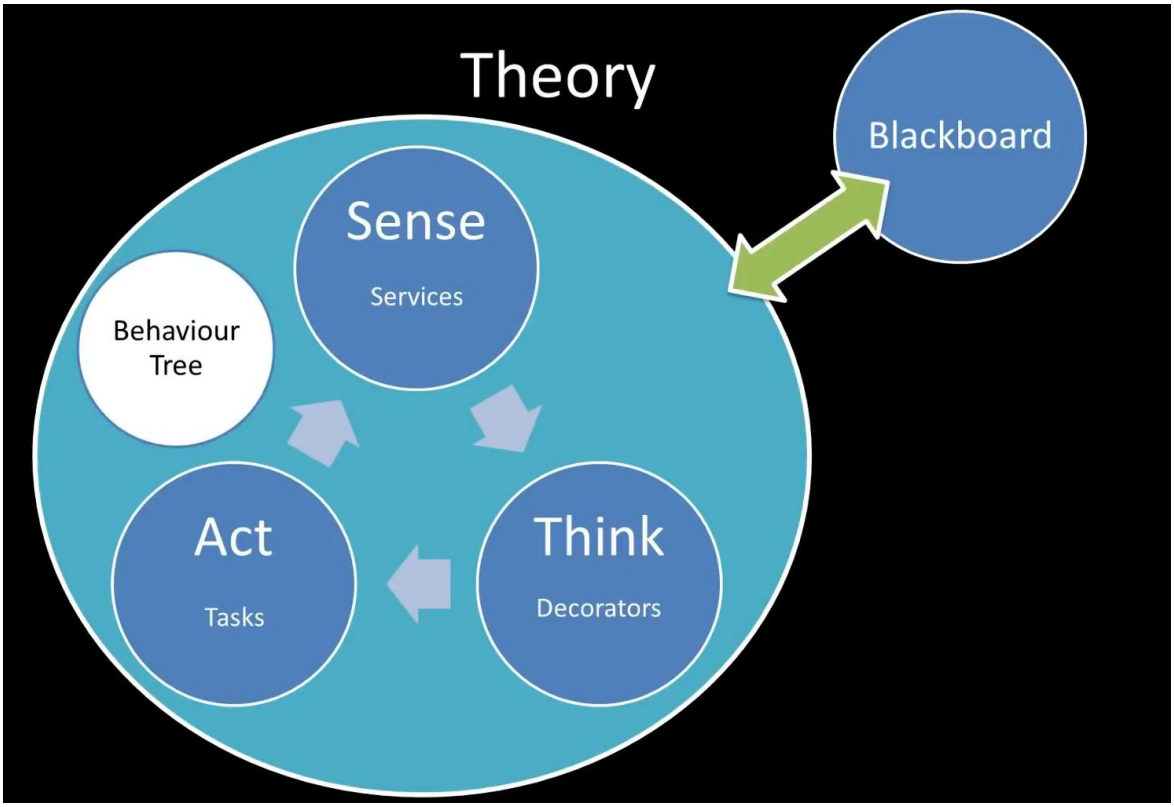
Ενέργεια *(Action)*:

Η ενέργεια που πραγματοποιεί μια οντότητα αποτελεί ίσως το πιο βασικό κομμάτι στον κύκλο και στην εμπειρία που ζει ο παίχτης. Αυτό συμβαίνει διότι αν η οντότητα μας αντιδράσει θα το δει ο παίχτης. Συνεπώς μέσω ενός έστω **οπτικού αποτελέσματος** θα καταλάβει ότι η ενέργεια που έκανε ο ίδιος είχε ένα αντίκτυπο. Μερικά σημεία που θα μπορούσαμε να σημειώσουμε είναι τα εξής:

- Η ενέργεια κατέχει την μεγαλύτερη διάρκεια εκτέλεσης στην κυκλική διαδικασία.
- Πολύ συχνά μπορεί να αντιμετωπίσει κάποιο πρόβλημα και παύει να τρέχει. Ένα παράδειγμα είναι πώς η οντότητα μπορεί να τρέχει από το σημείο Α στο σημείο Β που είναι μία μεγάλη απόσταση στο παιχνίδι μας. Καθώς διανύει αυτή την απόσταση είναι πιθανό να συναντήσει κάποιο εμπόδιο και να μην ολοκληρώσει το προκαθορισμένο έργο που της έχει ανατεθεί.
- Πρέπει οπωσδήποτε να δώσει κάποιου είδους **ανατροφοδότησης** στον παίχτη. Αν από την προοπτική του ο παίχτης δεν αντιληφθεί κάτι τέτοιο τότε ο όλος ο κύκλος καταρρέει.

Μία ενέργεια μέσα στην *Unreal Engine* μπορεί να υλοποιηθεί μέσω των ***Έργων (Tasks)*** [H]. Τα *Tasks* έχουν πολλές μορφές και η μηχανή ήδη είναι εξ αρχής εξοπλισμένη με διάφορα *tasks* που μπορούν να κάνουν την εργασία μας πιο εύκολη. Βέβαια η πρόκληση είναι "κρυμμένη" στο να δημιουργήσουμε τα δικά μας ανεξάρτητα *tasks*.

Έχοντας αναφέρει πλέον την βασική θεωρία πίσω από την οντότητα μας ήρθε η ώρα να ανακαλύψουμε πώς όλα τα παραπάνω στοιχεία συνδέονται. Τα **δέντρα συμπεριφοράς** *(Behavior Trees)* [A] είναι αυτά που συνδυάζουν όλες τις πληροφορίες που επισημάναμε και τις μεταφέρουν στη οντότητα μας. Ακολουθεί ένα σχήμα για την καλύτερη κατανόηση αυτής της ιδέας:



Εικόνα 3.2: Θεωρητικό Σχήμα του συστήματος T.N της Unreal Engine.

Παρατηρούμε πώς τα ***Services***, ***Decorators*** και τα ***Tasks*** ανήκουν μέσα στο ***Behavior Tree*** και αυτά αντιστοίχως εξυπηρετούν την **Αντίληψη**, την **Σκέψη** και την **Ενέργεια**. Για να εκτελεστούν όμως ομαλά θα χρειαστεί να δεχτούν κάποιες τιμές. Αυτές οι τιμές εισέρχονται στο *Behavior Tree* ως μεταβλητές από το *BlackBoard*.

Κλείνοντας το κεφαλαίο ας προσπαθήσουμε να αναλύσουμε σε επιμέρους κομμάτια την υλοποίηση της οντότητας μας. Συνεπώς έχουμε:

- **AI Character**: Αποτελεί το **σώμα** της οντότητας μας. Στην ουσία πρόκειται για ένα *2D* ή *3D* μοντέλο με ένα σκελετικό πλέγμα*(skeletal mesh)* που υπακούει εντολές.
- ***Animation BluePrint***: Αποτελείται από *State Machines*, *Blend Trees* κτλ που "κινούν" τον *AI Character* με κύριο σκοπό να επιστρέψουν μία οπτικοακουστική αναπαράσταση στα μάτια του παίχτη.

- ***AI Controller***: Αποτελεί το **κεφάλι** της οντότητας μας. Οι αρμοδιότητες του είναι πολύ περιορισμένες. Πρακτικά αυτό που κάνει είναι να φέρνει σε **υπόσταση**(*instantiate*) το *Behavior Tree* έτσι να το εκμεταλλευτεί ο *AI Character*.
- ***Behavior Tree***: Εφόσον δημιουργηθεί από τον *AI Controller* και "συνδεθεί" με τον *AI Character* αποτελεί πλέον τον **εγκέφαλο** της οντότητας. Την οδηγεί στο να παίρνει τις κατάλληλες αποφάσεις.
- ***BlackBoard***: Αποτελεί πλέον την **μνήμη** της οντότητας μας. Μέσω των αποθηκευμένων τιμών του μπορεί να καθοδηγήσει την συμπεριφορά της οντότητας μέσω του *Behavior Tree* στο σωστό αποτέλεσμα.

ΚΕΦΑΛΑΙΟ 4

ΕΙΣΑΓΩΓΗ

Σε αυτό το κεφάλαιο εφαρμόζουμε τις αρχές που θέσαμε στο Κεφάλαιο 3 μέσα στην *Unreal Engine*. Αναλυτικότερα ερευνούμε βασικά στάδια και εργαλεία όπως τα *Behavior Trees* που θα μας οδηγήσουν στην ολοκλήρωση του πρακτικού μέρους της εργασίας.

4.1 Πλοήγηση των Οντοτήτων - *Navmesh.*

Ένα κρίσιμο ερώτημα ασχέτως το τι αρμοδιότητες θα αναθέσουμε στις οντότητες μας θα πρέπει αρχικά να στήσουμε το περιβάλλον έτσι ώστε να μπορούν να κινηθούν σε αυτό. Μέσα στην *Unreal Engine* αυτό αποτέλεσε μία πολύ εύκολη δουλειά μέσω του συστήματος ***Navmesh*** που σε ένα μέρος αξιοποιεί τον Αλγόριθμο A* που αναλύσαμε στο Κεφάλαιο 1.

Το *Navmesh* λειτουργεί επίσης μέσω πολλών εργαλείων και βιβλιοθηκών των ***Recast/Detour***. Αποτελεί μία αυτοματοποιημένη διαδικασία που διαχωρίζει την γεωμετρία του εδάφους σε σημεία που μπορούν να κινούνται οι οντότητες μας. Αποτελεί την βέλτιστη επιλογή επίσης καθώς αυτός ο διαχωρισμός πραγματοποιείται πολύ **γρήγορα**.

Όντας η καλύτερη λύση, το *Navmesh* θα έχει την εξής μορφή μέσα στο τοπίο:

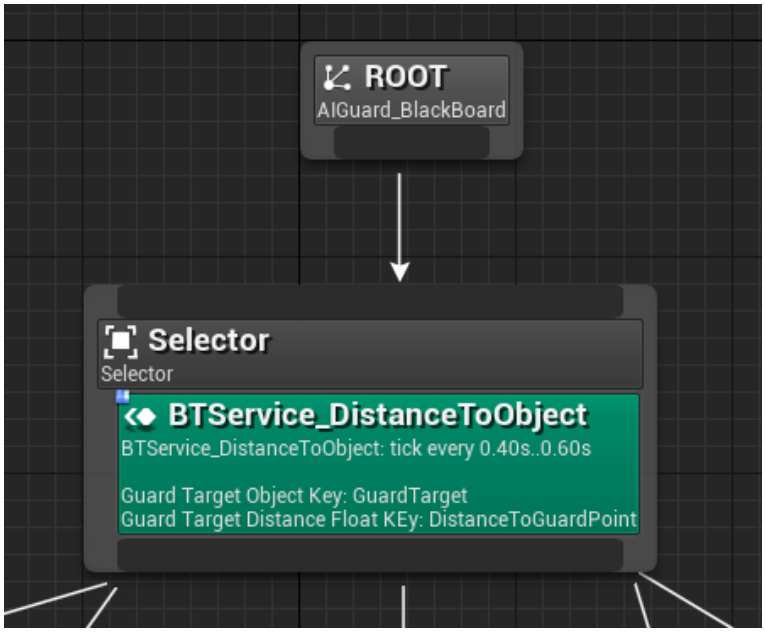
4.1.1: Αναπαράσταση του *Navmesh* μέσα στο τοπίο μας.

Στην παραπάνω εικόνα διακρίνουμε τον διαχωρισμό της γεωμετρίας του εδάφους. Το πράσινο χρώμα αποτελεί το έδαφος που οι οντότητες μας και εμείς θα μπορούμε να κινηθούμε. Τα κομμάτια που δεν επικαλύπτονται είναι τα δύσβατα μέρη. Αυτά αποτελούν τις πλαγιές αλλά και ένα μικρό μέρος δίπλα στα δέντρα μας. Αυτό γίνεται καθώς η γεωμετρία στις πλαγιές είναι πολύ απότομη έτσι ώστε να γίνει προσβάσιμη μέσω του *Navmesh* αλλά και τα *3D* μοντέλα που καταναλώνουν παραπάνω χώρο μέσα στο περιβάλλον από ότι φαίνεται.
Το "χτίσιμο"*(build)* του *Navmesh* πραγματοποιείται σε ***real time*** και είναι εύκολα **προσαρμοζόμενο** αν θέλουμε να αλλάξουμε το τοπίο όπως επιθυμούμε.

4.2 Υλοποίηση της Πρώτης Οντότητας - Φρουρού.

Φτάνοντας σε αυτό το μέρος του κεφαλαίου της παρούσας εργασίας θα ξεκινήσουμε με τις επιμέρους διαδικασίες που χρειάστηκαν για την δημιουργία της πρώτης οντότητας, δηλαδή αυτής που θα "φυλάσσει" ένα σημείο και κατά συνέπεια θα αντιδρά στην όψη ή σε έναν ήχο που θα εκτελεί ο παίχτης. Αρχικά, θα αναλύσουμε το δέντρο συμπεριφοράς της οντότητας του φρουρού*(GuardAI).*

Στην εικόνα που ακολουθεί βλέπουμε τον πρώτο κόμβο*(root)* και έπειτα θα παρατηρήσουμε τον κόμβο ***Selector*** [B]**.**

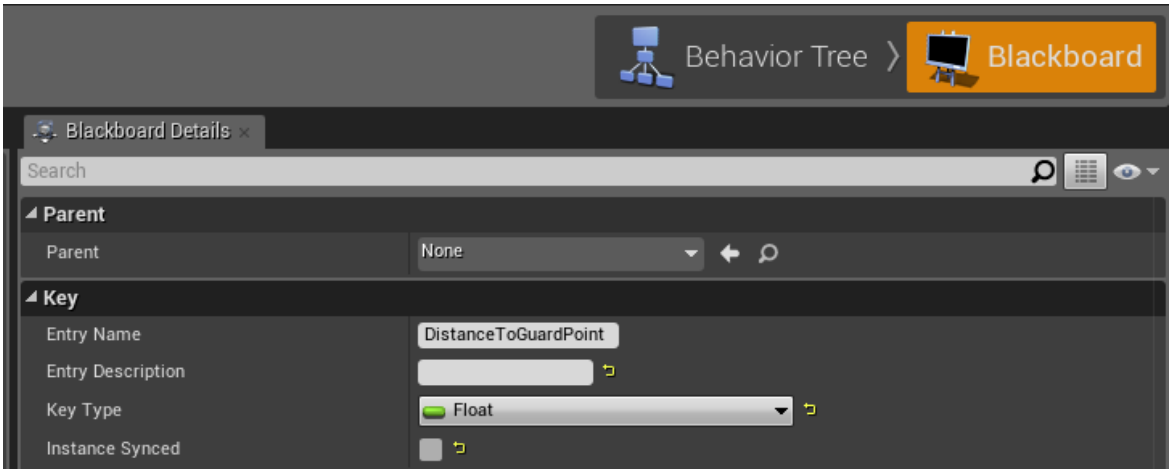


:

Εικόνα 4.2.1: *Selector Node in GuardAI's Behavior Tree*.

Η δουλεία της Υπηρεσίας*(Service)* εν ονόματι ***BTService_DistanceToObject*** είναι όπως προαναφέραμε να πραγματοποιεί συγκεκριμένους ελέγχους. Για λόγους ευκολίας μετονομάσαμε την Υπηρεσία με την ιδιότητα που πρέπει να εκτελεί έτσι ώστε να καταλαβαίνουμε εύκολα τις αρμοδιότητες της. Ας μην ξεχνάμε πώς η δουλεία ενός *Selector Node* είναι να εκτελεί τους κόμβους από κάτω με σειρά προτεραιότητας από τα **αριστερά στα δεξιά.** Αν όμως στην προκειμένη περίπτωση η Υπηρεσία μας δεν είναι "εγκρίνει" τους κανόνες που έχουμε θέσει ο *Selector Node* θα αποτύχει και οι κόμβοι παρακάτω δεν θα εκτελεστούν.

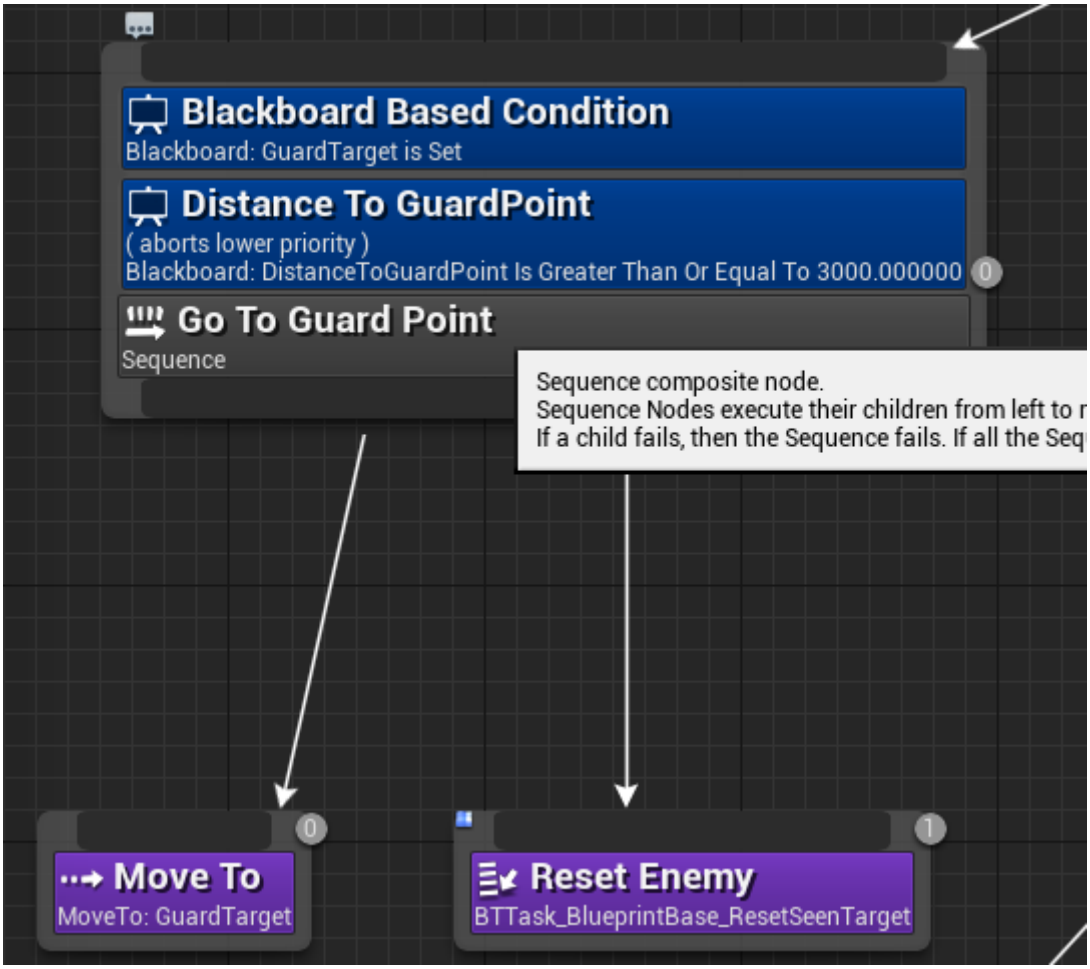
Αν παρατηρήσουμε καλύτερα την εικόνα 4.2.1 θα καταλάβουμε πώς η συχνότητα που πραγματοποιούνται αυτοί οι έλεγχοι είναι το μέσο μεταξύ των τιμών 0.40 δευτερολέπτων και 0.60 δευτερολέπτων. Έπειτα πραγματοποιούμε δύο απλούς ελέγχους. Στην **πρώτη συνθήκη** ορίζουμε πως το σημείο που πρέπει να φυλάει ο φρουρός μας ως *GuardTarget* πρέπει να υπάρχει μέσα στο περιβάλλον. Και η **δεύτερη συνθήκη** πολύ απλά μετράει την απόσταση που απέχει ο φρουρός μας από το σημείο που φυλάει.
Ο λόγος που χρειάζεται να αποθηκεύουμε μία τέτοια μεταβλητή είναι απλός:
Αν ο φρουρός μας ξεπερνάει την απόσταση που του έχουμε ορίσει επειδή κυνηγάει τον παίχτη, θα τον αναγκάσουμε να γυρίσει πίσω έτσι ώστε να ξανά φυλάξει το σημείο του.
Αναφερθήκαμε ήδη όμως στην αποθήκευση μεταβλητών και για αυτό ήρθε να γνωρίσουμε και το ***Blackboard*** μέσα στην Unreal Engine:



Εικόνα 4.2.2: Αποθήκευση μεταβλητής στο Blackboard.

Όπως αναγράφεται στο πάνω δεξιά μέρος της εικόνας 4.2.2 παρατηρούμε πώς όντως το *Behavior Tree* και το *Blackboard* συνδυάζονται πολύ στενά.
Χωρίς την εισαγωγή τιμών στο *Blackboard* οποιαδήποτε τιμή και να προσπαθήσουμε να εισάγουμε στο *Behavior Tree* δεν θα γίνει δεκτή καθώς δεν θα μπορεί να την αναγνωρίσει.

Συνεχίζοντας στους επόμενους κόμβους θα εξετάσουμε τον επόμενο κόμβο προς εκτέλεση, δηλαδή τον πρώτο από τα αριστερά:



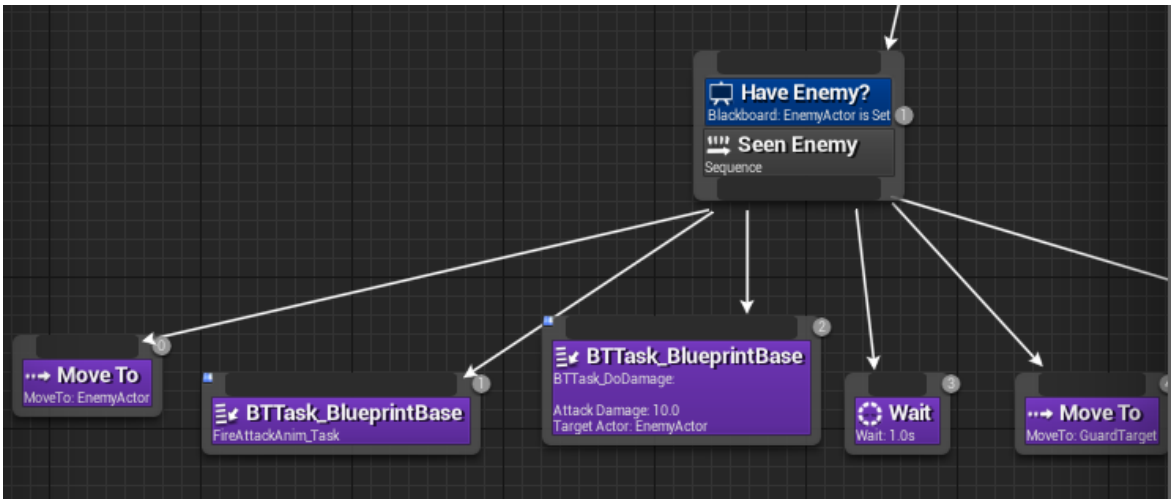
Εικόνα 4.2.3: Πρώτη Σειρά Κόμβων

Το παραπάνω σχήμα αποτελείται από τους Διακοσμητές*(Decorators)* οι οποίοι διακρίνονται από το μπλε χρώμα. Στην ουσία αυτή η σειρά κόμβων ελέγχει αν το σημείο που φυλάει ο φρουρός μας έχει οριστεί και αν η απόσταση του φρουρού από το σημείο φύλαξης, ξεπερνάει ή είναι ίσο με ***3000 Unreal Units(30 cm)*** θα

αναγκάσει την οντότητα να μετακινηθεί πίσω στο σημείο και να "ξεχάσει" πώς είδε τον παίχτη.

Οι δύο τελευταίοι κόμβοι είναι Έργα*(Tasks)* και ξεχωρίζουν μέσω του μωβ χρώματος.

Επόμενοι κόμβοι στην σειρά είναι οι εξής:



Εικόνα 4.2.4: Δεύτερη σειρά Κόμβων.

Ακολουθώντας την ίδια διαδικασία μέσω ενός Διακοσμητή ερωτούμε την οντότητα μας αν έχει δει τον παίχτη. Αν δει τον παίχτη, δηλαδή η μεταβλητή *EnemyActor* γίνει *Set* τότε θα εκτελεστούν με σειρά από αριστερά στα δεξιά τα παρακάτω έργα:

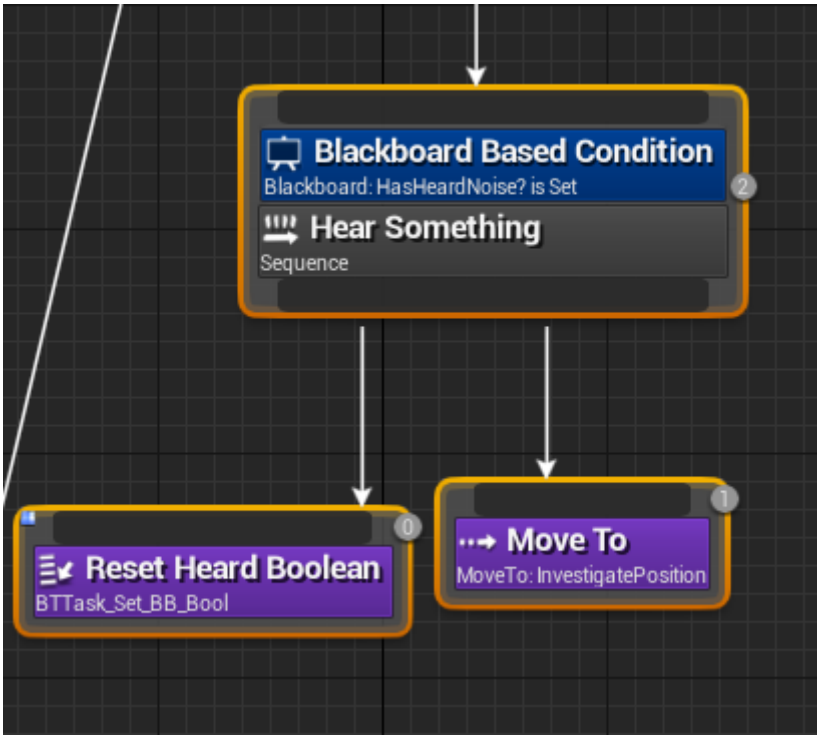
- Προχώρα προς τον παίχτη.
- Πραγματοποίησε ένα είδος Κίνησης
- Προχώρα στο να δημιουργήσεις Ζημιά στον παίχτη
- Περίμενε 1 δευτερόλεπτο
- Γύρνα πίσω στο σημείο που φρουρείς.

Παρατηρούμε λοιπόν μία σειρά συμπεριφορών που έχουμε αναθέσει στην οντότητα μας μέσω των Έργων.

Πώς μπορεί όμως η οντότητα μας να δει που βρίσκεται ο παίχτης; Αυτό πραγματοποιείται στο πρώτο παράδειγμα με την προσθήκη ενός συστατικού(*Component)* πάνω στην οντότητα μας και πιο συγκεκριμένα στο "κεφάλι" της οντότητας μας, τον *AI Controller*, που ονομάζεται ***PawnSensing***.

Το *PawnSensing Component* ενεργοποιεί τις βασικές αισθήσεις στην οντότητα μας όπως η **όραση** και η **ακοή.** Αναλυτικότερα ο τρόπος που θέτουμε το πόσο μακριά μπορεί να βλέπει και να ακούει παρουσιάζεται στο Παράρτημα Α'.

Μετά έχουμε την σειρά των κόμβων που οδηγούν στην συμπεριφορά της οντότητας μόλις ακούσει κάτι:

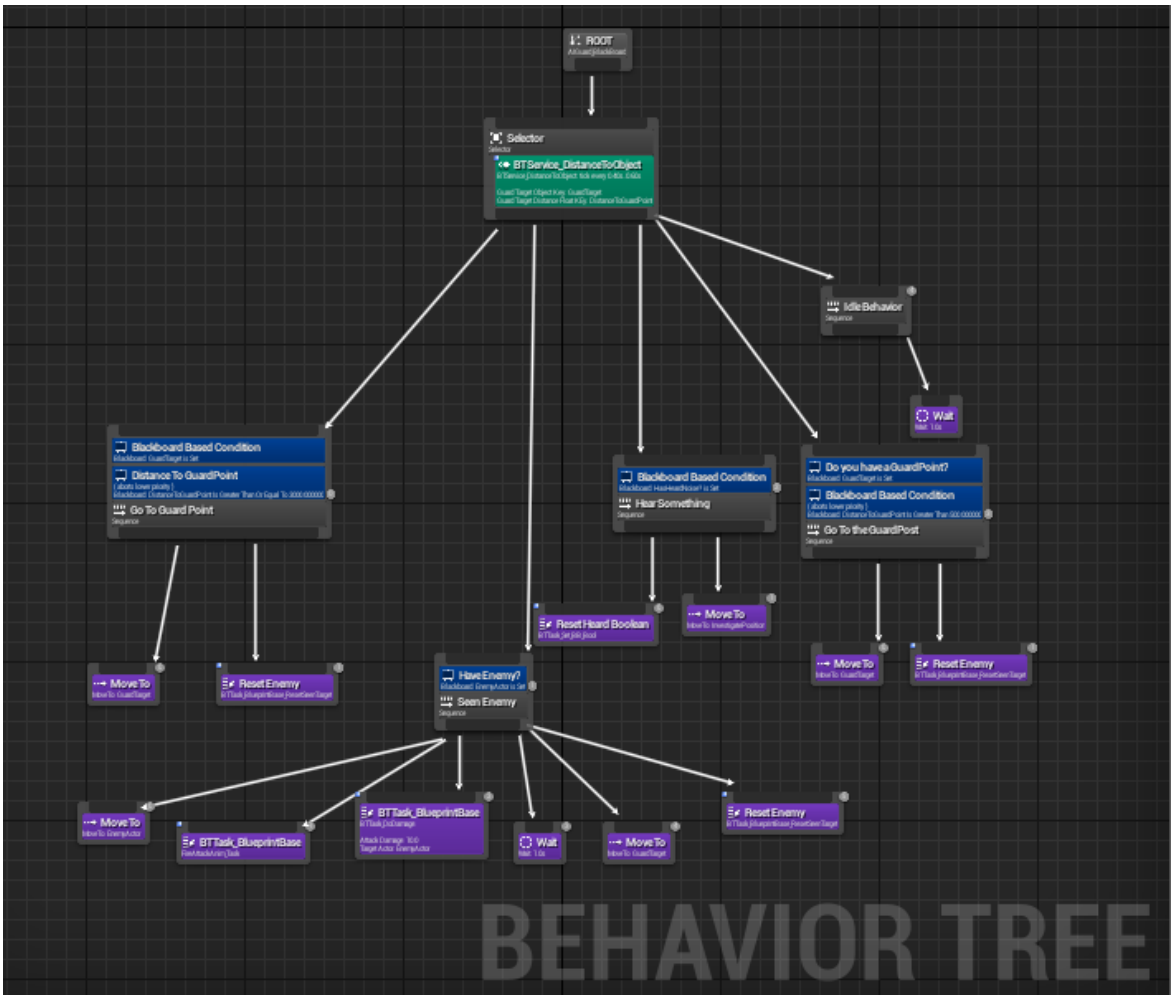


Εικόνα 4.2.5: Τρίτη σειρά Κόμβων.

Αν λοιπόν η λογική μεταβλητή *HasHeardNoise?* έχει οριστεί από το άκουσμα ενός ήχου τότε θα χρειαστεί να αλλάξουμε την τιμή της μεταβλητής πάλι έτσι να σταματήσει να ακολουθάει τον παίχτη. Κάτι τέτοιο θα συνέβαινε αφού ο ακριβώς επόμενος κόμβος που βρίσκεται δεξιά δεν θα έβγαινε ποτέ από την επανάληψη αναγκάζοντας την οντότητα να κυνηγάει τον παίχτη που προκάλεσε τον ήχο. Αυτό αποτρέπεται μέσω του *Task Reset Heard Boolean*. Τέλος έχουμε προσθέσει δύο ακόμα σειρές κόμβων που πραγματοποιούν ελέγχους και πάλι για την απόσταση ως προληπτικό μέτρο αλλά και την τελευταία στην ιεραρχία σειρά των κόμβων που ονομάζεται ***Idle*** έτσι ώστε αν δεν συμβαίνει

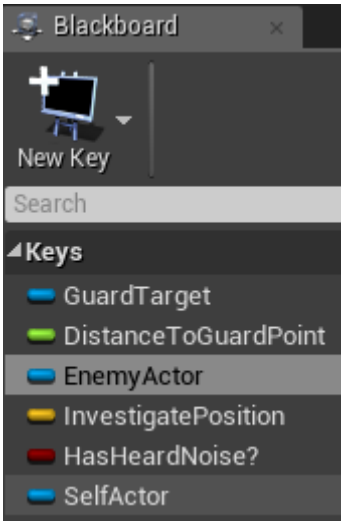
τίποτα από τα προηγούμενα η οντότητα μας να παραμένει άεργη. Αυτή η διαδικασία μπορεί πολύ ορθά να μας θυμίζει μία *FSM.*

Το τελικό μας Δέντρο Συμπεριφοράς θα έχει την παρακάτω μορφή:



Εικόνα 4.2.6: Τελικό *Behavior Tree* του *GuardAI.*

Το οποίο λειτουργεί χάρη στις παρακάτω μεταβλητές του *Blackboard:*



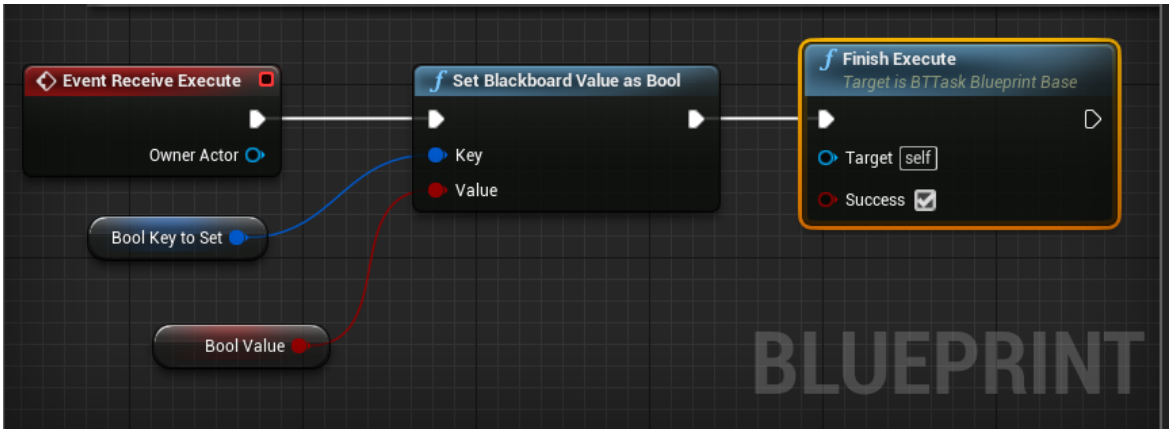
Εικόνα 4.2.7: Μεταβλητές του *GuardAI*.

Αναλύοντας λοιπόν το πώς η οντότητα του φρουρού μας μπορεί να πάρει αποφάσεις και να δράσει αντίστοιχα σε κάποια γεγονότα θα πρέπει επίσης να αναλύσουμε τι πραγματικά συμβαίνει πίσω από το *Behavior Tree*. Οι Υπηρεσίες, οι Διακοσμητές αλλά και τα Έργα λειτουργούν βάσει των προγραμματιστικών σχέσεων που υπάρχουν κάτω από την επιφάνεια τους. Αυτές οι προγραμματιστικές σχέσεις ονομάζονται ***Blueprints***.

4.3 *Blueprints.*

Τα *Blueprints* αποτελούν μία σαφή καινοτομία εκ μέρους των προγραμματιστών της *Unreal Engine*. Πρόκειται για μία μορφή προγραμματισμού που ονομάζεται *Visual Scripting*. Ο λόγος πίσω από την ανάγκη να δημιουργηθεί ένα σύστημα σαν το *Blueprint* κρύβεται στην απλή παραδοχή πώς στον κόσμο της ανάπτυξης ψηφιακών παιχνιδιών δεν είναι όλοι προγραμματιστές. Αυτός ο κόσμος αποτελείται από μουσικούς, *3D artists*, *game designers* και πολλούς άλλους. Επομένως η ανάγκη για ένα εύχρηστο σύστημα που θα μπορεί να χρησιμοποιήσει κάποιος χωρίς προγραμματιστικό υπόβαθρο και θα διαθέτει παρόμοιες αν όχι ίδιες υπηρεσίες με γλώσσες προγραμματισμού όπως η C++ (στην οποία έχει χτιστεί) ήταν αναμενόμενη. Στην ουσία αποτελείται από ένα πολύ φιλικό ***interface*** που λειτουργεί και αυτό με **κόμβους συναρτήσεων***(functions),* **γεγονότων***(events)* και άλλων στοιχείων που συνδέονται μέσω **καλωδίων***(wires)* για να μας αποδώσει ένα πανίσχυρο σύστημα ανάπτυξης *gameplay*.

Για την καλύτερη κατανόηση των *BP* ας αναλύσουμε ένα Έργο.



Εικόνα 4.3.1: Blueprint του Έργου Reset Heard Boolean της εικόνας 4.2.5.

Ένα πράγμα που ίσως παρατηρούμε από την αρχή είναι πως το *Blueprint* ακολουθεί μία **σειριακή πορεία**. Η άσπρη κουκίδα που ενώνει του κόμβους σηματοδοτεί την επιτυχή εκτέλεση του εν λόγω κόμβου και την συνέχεια προς την εκτέλεση του επόμενου.

Συνεπώς αυτό το σύστημα υποδεικνύει έναν σειριακό τρόπο που λειτουργούν τα *Blueprints*.

Ξεκινώντας, διακρίνουμε έναν κόμβο με τον τίτλο "*Event Receive Execute*". Αυτός ο αρχικός κόμβος που ξεχωρίζει μέσω του κόκκινου χρώματος μας ενημερώνει πώς όταν εκτελεστεί ένα *event* θα συμβούν και τα παρακάτω:

- Δημιούργησε μία συνάρτηση με όνομα "*Set Blackboard Value as Bool*" που θα δέχεται δύο ορίσματα, ένα κλειδί από το *Blackboard* τύπου *Boolean* που επρόκειτο να πάρει μία τιμή και παρουσιάζεται με μπλε χρώμα. Και μία τιμή μεταβλητή τύπου ***Boolean***.
- Εκχώρησε την τιμή της μεταβλητής μέσα στο κλειδί που δέχτηκες από το *Blackboard*.
- Αν η εκχώρηση είναι επιτυχής θα κριθεί μέσα από τον επόμενο κόμβο - συνάρτηση, "*Finish Execute*", που μέσω ενός απλού *checkbox* με όνομα *Success* που έχουμε εισάγει θα σημάνει το τέλος της σειριακής αυτής διαδικασίας.

Αυτός το παράδειγμα είναι ένας απλοϊκός τρόπος για να κατανοήσει κανείς πώς λειτουργούν τα *Blueprints*. Είναι πραγματικά αξιοσημείωτο όμως το γεγονός πώς μέσω αυτών των συστημάτων μπορούμε να δημιουργήσουμε **πολύπλοκες σχέσεις και μαθηματικές πράξεις** μεταξύ των αντικειμένων που υπάρχουν σε ένα παιχνίδι. Το παραπάνω έργο λοιπόν, καταφέρνει να αλλάξει την τιμή μέσα στο *Blackboard* μας και με την σειρά του να αλλάξει και την ροή που θα εκτελεστεί το *Behavior Tree*. Ως αποτέλεσμα παρατηρούμε πώς τα *Blueprints* αποτελούν το τελικό κομμάτι που συνδέει όλες τις έννοιες που έχουμε αναλύσει έως τώρα. Περισσότερα και αρκετά πιο περίπλοκα *Blueprints* παραθέτονται στο παράρτημα Β'.

4.4 Υλοποίηση της Δεύτερης Οντότητας.

Ξεκινώντας με την σχεδίαση της δεύτερης οντότητας θα πρέπει αρχικά να δούμε πώς θα αντιλαμβάνεται το περιβάλλον γύρω της. Στην πρώτη οντότητα αξιοποιήσαμε το *Component PawnSensing* έτσι ώστε να προσδώσουμε της αισθήσεις της ακοής και της όρασης. Εδώ θα πραγματοποιήσουμε την ίδια διαδικασία αλλά με διαφορετικό τρόπο. Προσθέσαμε ένα *Component* που λειτουργεί διαφορετικά και μας δίνει περισσότερη **ευελιξία** ως το πόσο μακριά θα βλέπει η οντότητα μας. Αυτό έγινε καθώς σε αυτό το σημείο η οντότητα μας θα πρέπει να είναι σε θέση να αναγνωρίζει περισσότερα αντικείμενα εκτός από τον παίχτη, με αποτέλεσμα να εισάγουμε ένα καλύτερο σύστημα.

Ακολουθεί μία εικόνα για να κατανοήσουμε πώς λειτουργεί:

Εικόνα 4.4.1: Αναπαράσταση του *AIPerception Component.*

Παραπάνω διακρίνουμε δύο κύκλους που αφορούν την όραση. Ο **πράσινος** κύκλος μας δείχνει πόσο μακριά μπορεί να βλέπει η οντότητα μας και ο **μωβ** κύκλος μας δείχνει που τελειώνει η όραση του. Σε αυτό το σημείο αξίζει να αναφερθεί πώς **δεν** έχουμε εισάγει την αίσθηση της ακοής εδώ.

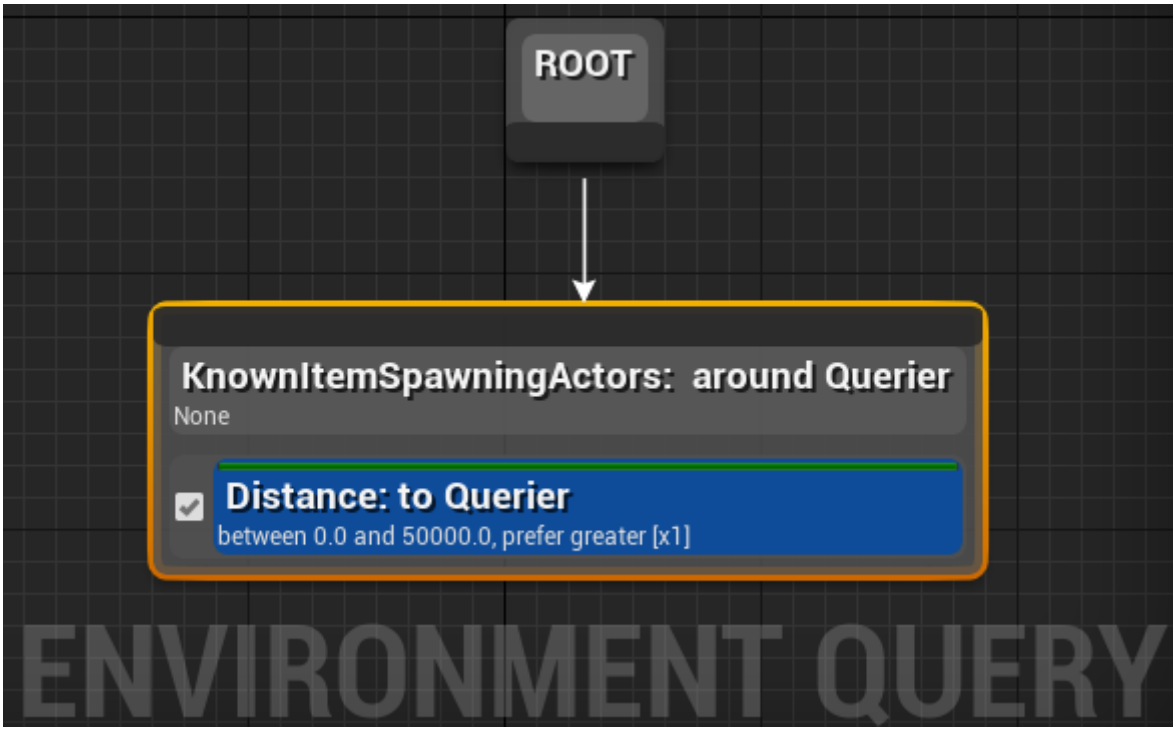
4.5 *Environmental Query System.*

Προχωρώντας, θα ήταν συνετό να αναλογιστούμε πώς η οντότητα μας θα πλησιάζει τα δέντρα που του παρέχουν την τροφή με έναν βέλτιστο τρόπο.

Το *EQS* αποτελεί ένα "λογικό" σύστημα που φτιάχτηκε για να επιτρέπει στους χρήστες της *Unreal Engine* να δημιουργούν **αντικείμενα/κεφάλαια ερωτήσεων***(query assets).* Ο σκοπός όμως είναι μέσω κάποιων **πρότυπων ερωτήσεων**(query templates) να κάνουμε ερωτήσεις στα *query assets* για το περιβάλλον παιχνιδιού. Αποτελεί ένα πειραματικό χαρακτηριστικό μέσα στην *Unreal Engine* και για αυτό θέλει ειδική προσοχή στην διαχείριση του.

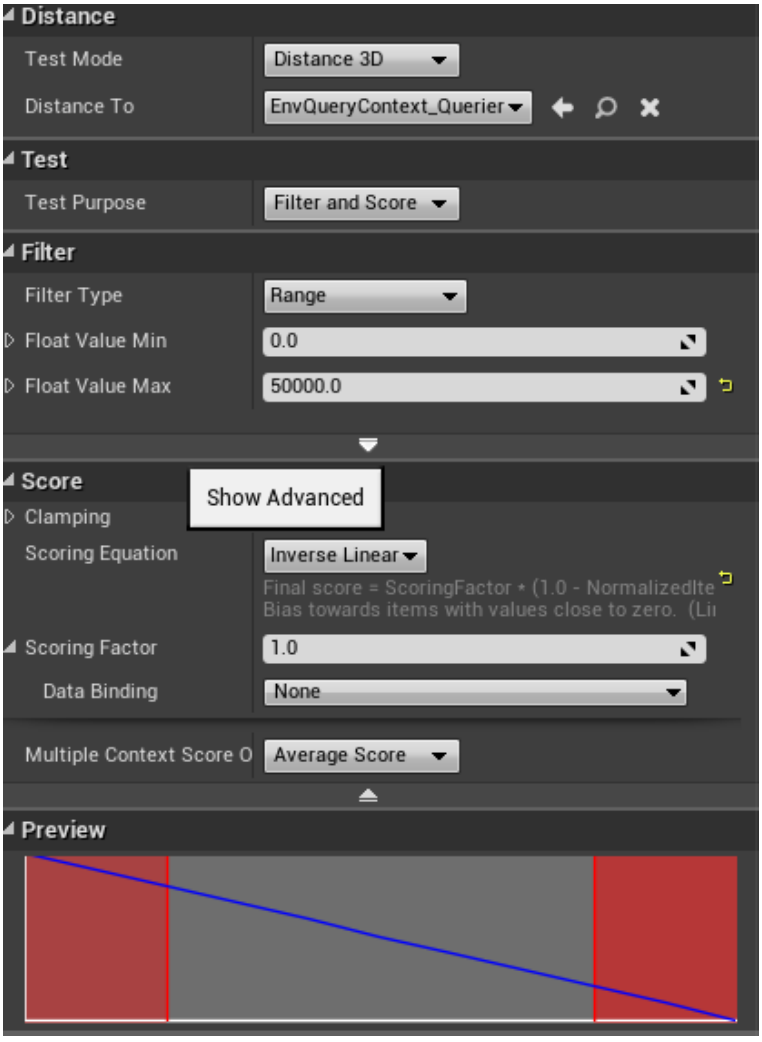
Για την πλήρη κατανόηση του ένας ορισμός δεν είναι αρκετός, συνεπώς θα προσπαθήσουμε να καταλάβουμε μέσω του *EQS* που απαιτήθηκε να δημιουργήσουμε για τις ανάγκες της δεύτερης οντότητας.

Ας ξεκινήσουμε με την δημιουργία του *EQS* μας. Αναφερόμενο στο δέντρο που πρέπει να γυρίζει πίσω ο παίχτης για να αναζητήσει την τροφή του, θα δημιουργήσουμε ένα *query template*. Ο σκοπός αυτού το *query template* είναι να πραγματοποιεί "έξυπνες" ερωτήσεις γύρω στο περιβάλλον μας έτσι ώστε η οντότητα μας να αντιληφθεί που είναι τοποθετημένα αυτά τα δέντρα. Το *query template* θα έχει την εξής μορφή:



Εικόνα 4.5.1: *EQS* των αντικείμενων που δημιουργούν τροφή.

Όπως και στα *Behavior Trees* παρατηρούμε πώς και εδώ ο αρχικός μας κόμβος είναι το *root*. Έπειτα μετά από τον κόμβο root τραβήξαμε ένα βέλος για να δημιουργήσουμε έναν ***Generator***. Οι *Generators* διαθέτουν εφτά διαφορετικά είδη και ουσιαστικά αποτελούν το group των αντικειμένων που επιθυμούμε να τεστάρουμε. Πρόκειται για ένα **σύνολο** που φιλτράρουμε έτσι ώστε να διακρίνουμε αν διαθέτουν τις κατάλληλες προϋποθέσεις. Ο *Generator* που επιλέξαμε στην περίπτωση μας ονομάζεται ***KnownItemSpawningActors*** και πρόκειται για έναν αποκλειστικά φτιαγμένο*(custom)* από εμάς *Generator*. Η *Unreal Engine* μας παρέχει την δυνατότητα να φτιάχνουμε τους δικούς μας *Generators* για περεταίρω ευελιξία. Τα δέντρα που επιθυμούμε να δοκιμάσουμε διαθέτουν μία συγκεκριμένη ιδιότητα. Την ιδιότητα να παράγουν την απαραίτητη τροφή έτσι ώστε να τραφεί η οντότητα μας όταν φτάσει στο σημείο πείνας.
Ακριβώς από κάτω έχουμε εισάγει στον *Generator* μας ένα **τεστ** που απεικονίζεται στην εικόνα 4.5.1 με **μπλε** χρώμα. Μέσα στο τεστ μας λοιπόν έχουμε ορίσει τις εξής προϋποθέσεις:



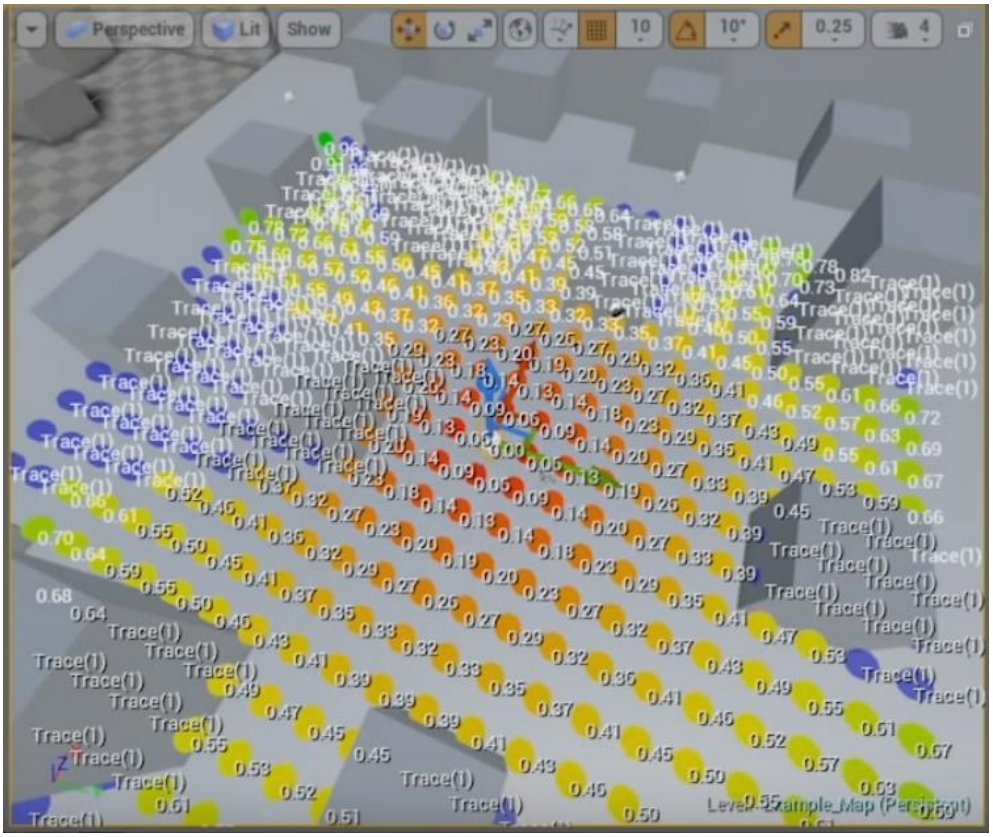
Εικόνα 4.5.2: Τεστ *EQS* βασισμένο στην απόσταση από τον παίχτη.

Στην εικόνα 4.5.2 λοιπόν παρατηρούμε πώς ως **φίλτρο** έχουμε εισάγει μία τιμή τύπου *float* με ελάχιστη τιμή το 0 και μέγιστη το 50000. Με απλά λόγια αυτό το τεστ "ρωτάει" ένα query asset αν υπάρχει αυτό το σύνολο των δέντρων μέσα στο εύρος των τιμών που αναθέσαμε. Αυτή η διαδικασία αποτελεί τη έννοια του φιλτραρίσματος όπως αναγράφεται και στην εικόνα. Έπειτα ακολουθεί μία καινούρια έννοια ως τεστ. Η έννοια του ***score***.

Το *score* είναι από τα πιο σημαντικά χαρακτηριστικά του *EQS* καθώς ειδικά στην δική μας περίπτωση "αναγνωρίζει" πιο από τα δέντρα του συνόλου βρίσκεται κοντά στο *query asset* με στόχο ο παίχτης μας να διανύσει την μικρότερη δυνατή απόσταση.

Στο δικό μας παράδειγμα έχουμε ορίσει ως παράγοντα του *scoring* το 1.0. Αυτό μας δείχνει πώς η κοντινότερη απόσταση στο *query asset* θα είναι στο 0 και όσο αυξάνεται θα γίνει μέγιστη στο 1. Έχοντας επιλέγει όμως της **αντίστροφης γραμμικής εξίσωσης***(inverse linear)* θα συμβεί το ακριβώς αντίθετο. Η κίνηση αυτή έγινε για λόγους *debug*. Τι ακριβώς όμως είναι το *query asset* μέσα στο τοπίο μας που προκαλεί αυτούς του ελέγχους;

Η απάντηση έρχεται μέσω του ***EQS_TestingPawn***. Πρόκειται για ένα αντικείμενο σαν όλα τα άλλα που δέχεται ως όρισμα το *query template* που δημιουργήσαμε προηγουμένως. Τέλος το *EQS* θα έχει μία παρόμοια μορφή με αυτήν της εικόνας παρακάτω:



Εικόνα 4.5.3: *EQS_TestPawn* που λειτουργεί με *Grid Generator.*

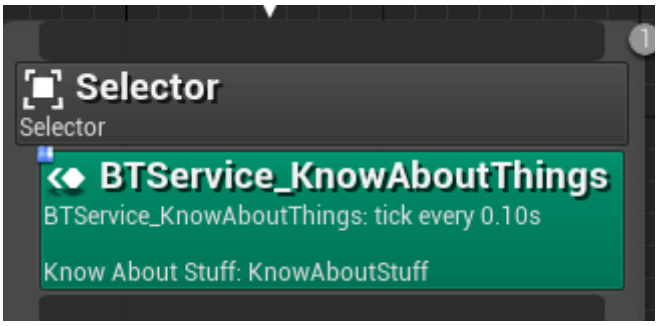
Στην παραπάνω εικόνα παρατηρούμε μία πληθώρα κύκλων με αναγραφόμενες συγκεκριμένες τιμές επάνω τους. Στο κέντρο του κύκλου βρίσκεται το *EQS_TestPawn*. Οι αναγραφόμενες αυτές τιμές προέρχονται από το πεδίο *Score* που αναλύσαμε πριν και έχουν ως ελάχιστο το 0 και ως μέγιστο το 1. Επίσης αξίζει να σημειωθεί πώς οι κύκλοι με το **μπλε** χρώμα δεν διαθέτουν τις κατάλληλες προϋποθέσεις και για αυτό έχουν **αποτύχει** στο τεστ που έχουμε εισάγει. Οι κύκλοι με **πράσινο** χρώμα διαθέτουν τις καλύτερες προϋποθέσεις ενώ αυτές με το **κόκκινο** που είναι δίπλα στο κέντρο αποτελούν τις χειρότερες επιλογές. Αυτό θα μπορούσε εύκολα να αλλάξει αλλάζοντας την επιλογή της εξίσωσης του τεστ σε *Inverse Linear* κτλ..Διακρίνουμε πέρα από τις τιμές λοιπόν και μία ευκολία στην αναγνώριση των καλύτερων επιλογών μέσω του χρωματικού συστήματος.

Ανακεφαλαιώνοντας, είμαστε σε θέση να καταλάβουμε πώς το *EQS* αποτελεί ένα πανίσχυρο σύστημα ως προς την ανάπτυξη της Τ.Ν. Τα δύο στοιχεία που το προκαλούν αυτό είναι η δυνατότητα της ανάπτυξης των δικών μας ***Generators*** καθώς και ο ορισμός του δικού μας κέντρου***(Query_Context)***. Μέσω αυτών των

στοιχείων καταφέρνουμε να "χτίσουμε" πολύπλοκες σχέσεις μεταξύ των οντοτήτων και του περιβάλλοντος μας. Ένα καλό παράδειγμα θα αποτελούσε κάποιες οντότητες που θα προσπαθούσαν να ξεφύγουν από την όψη του παίχτη αναζητώντας τα τυφλά σημεία. Αυτά τα σημεία θα μπορούσαν να τα αναγνωρίσουν και να μετακινηθούν προς αυτά με την βοήθεια ενός *EQS*.

4.6 Μνήμη στην T.N.

Έχοντας αναλύσει τον τρόπο με τον οποίο η οντότητα μας θα καταφτάνει στο κοντινότερο σημείο γύρω του που του παρέχει τροφή αφήνουμε αναπάντητο ακόμα ένα ερώτημα. Πώς καταφέρνει η οντότητα μας να θυμάται που ακριβώς βρίσκονται τα δέντρα μέσα στο περιβάλλον μας;
Η έννοια της **μνήμη**ς φυσικά παρέχεται από την άλλη μία φορά πολύτιμη βοήθεια μίας Υπηρεσίας.



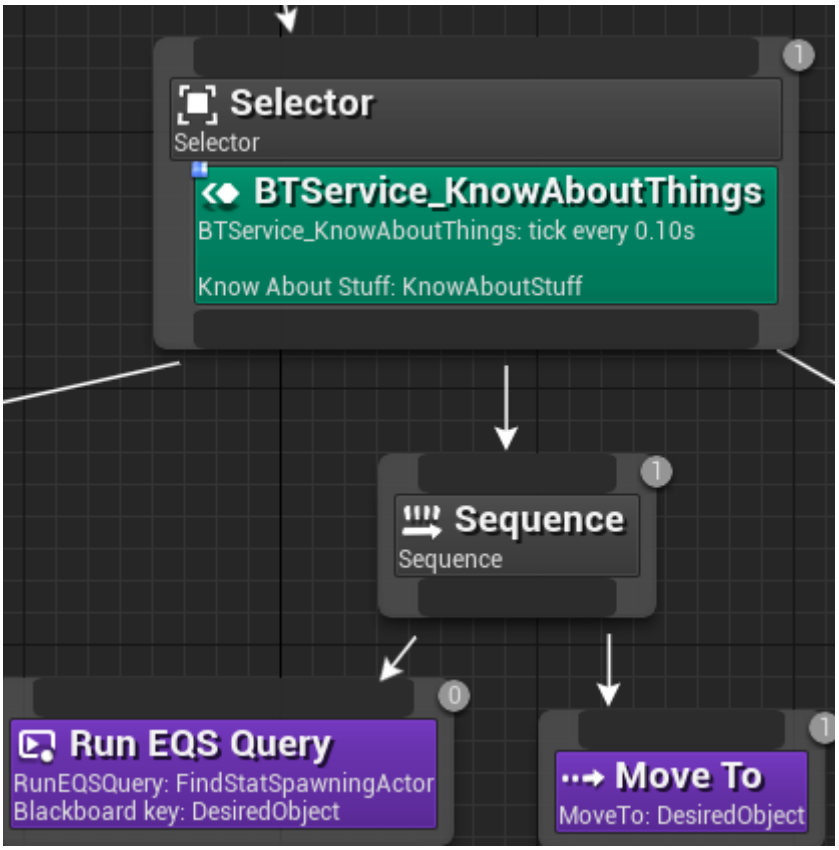
Εικόνα 4.6.1: Υπηρεσία με λειτουργία μνήμης.

Πριν λοιπόν εκτελεστεί το *EQS* και αναζητήσει η οντότητα μας την κοντινότερη απόσταση των δέντρων από την ίδια εκτελούμε μία υπηρεσία που πραγματοποιεί έναν έλεγχο. Ο έλεγχος υλοποιήθηκε σε *Blueprint* που παραθέτεται στο Παράρτημα Β'. Παρατηρούμε όμως πώς η συχνότητα της Υπηρεσίας *KnowAboutThings* είναι στα 0.10 δευτερόλεπτα. Ουσιαστικά αυτό που **διατάζουμε** την οντότητα να μας κάνει είναι καθώς περιφέρεται τυχαία, ανά 1/10 του δευτερολέπτου να προσέξει αν μέσω της όρασης της έχει δει πουθενά ένα κόκκινο μήλο. Η διαδικασία είναι ευκολότερη από ότι παρουσιάζεται καθώς αυτό που εννοούμε πραγματικά είναι:

- Έχεις δει ένα αντικείμενο της κλάσης *Good Food Class*;

- Αν ναι, τότε θέσε την τιμή της *Boolean* μεταβλητής *KnowAboutStuff* ως *true.*

Εφόσον λοιπόν περάσουμε από την Υπηρεσία και η μεταβλητή KnowAboutStuff γίνει **true** συμβαίνουν τα εξής στο *Behavior Tree*:



Εικόνα 4.6.2: Σειρά Κόμβων για τον καθορισμό της Μνήμης.

- Εκτέλεσε το *EQS* έτσι ώστε να αναγνωρίσει που είδε τελευταία φορά το δέντρο*(StatSpawningActor)* που μας παρέχει τα μήλα*(DesiredObjects).*
- Και στην συνέχεια μετακινήσου προς αυτό.

Έχοντας τοποθετήσει αυτή την σειρά κόμβων με την κατάλληλη σειρά σε όλο το σύνολο του *Behavior Tree* θα παρατηρήσουμε πως τα αποτελέσματα θα είναι εντυπωσιακά. Εισάγουμε λοιπόν μία μεταβλητή που αν γίνει *true* εφόσον η οντότητα μας δει ένα μήλο θα τον αναγκάσει να μετακινηθεί πίσω στο μέρος, δηλαδή στο δέντρο, που τα μήλα θα παρέχουν την απαραίτητη τροφή στον παίχτη. Σε αυτό το σημείο το σημείο θα πρέπει να αναφέρουμε πώς η συχνότητα που δημιουργούνται τα μήλα είναι ορισμένη από εμάς.

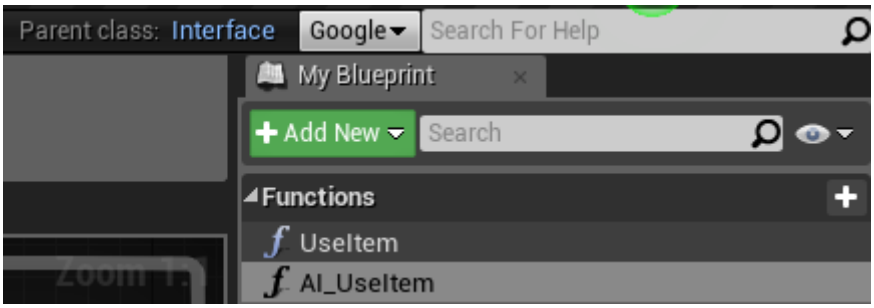
Προχωρώντας όμως ας μην ξεχνάμε πώς μέσα στο περιβάλλον μας έχουμε ορίσει και τα καφέ μήλα ως κακή τροφή ενώ συγκαταλέγονται στην ίδια κλάση. Συνεπώς

ήρθε η ώρα να κατανοήσουμε πώς η οντότητα αλληλεπιδρά με τα αντικείμενα γύρω της.

4.7 Interfaces

Ειδικότερα, τα ***Blueprint Interfaces*** μας παρέχουν μία κοινή μέθοδο από τις γλώσσες προγραμματισμού, **αλληλεπίδρασης** με όλα τα αντικείμενα που συνυπάρχουν μέσα στον κόσμο μας. Αρκεί να μοιράζονται μία συγκεκριμένη ιδιότητα. Έστω λοιπόν πως παραθέτουμε μία ιδιότητα, ονομαζόμενη ως *OnTouchFallDown,* που προκαλεί κάποιο αντικείμενο να πέσει στο έδαφος μόλις το ακουμπήσει*(collide)* ο παίχτης. Αν εισάγουμε αυτή την ιδιότητα σε ένα δέντρο και μία καρέκλα, όσο περίεργο και να φαίνεται αυτά τα αντικείμενα θα αντιδράσουν με τον ίδιο τρόπο. Ίσως όχι ακριβώς με τον ίδιο τρόπο καθώς όπως έχουμε αναφέρει η μηχανή κάνει εξαιρετική χρήση της Φυσικής αλλά η ιδέα παραμένει η ίδια.

Τα *Interfaces* μέσα στην *Unreal Engine* είναι πολύ εύκολο να φτιαχτούν. Η διαδικασία απαιτεί ελάχιστα βήματα. Φυσικά οτιδήποτε αποτελεί λόγο δράσης της οντότητας μας πρέπει να συμπεριληφθεί ως μία λήψη απόφασης. Συνεπώς θα εισάγουμε τον όρο της επιλογής της τροφής ως ένα **΄Εργο** μέσα στο *Behavior Tree*. Αρχικά όμως ας αναλύσουμε την δημιουργία ενός *Interface* και πώς αυτό με την σειρά του θα δρομολογηθεί μέσα στο *Behavior Tree*.

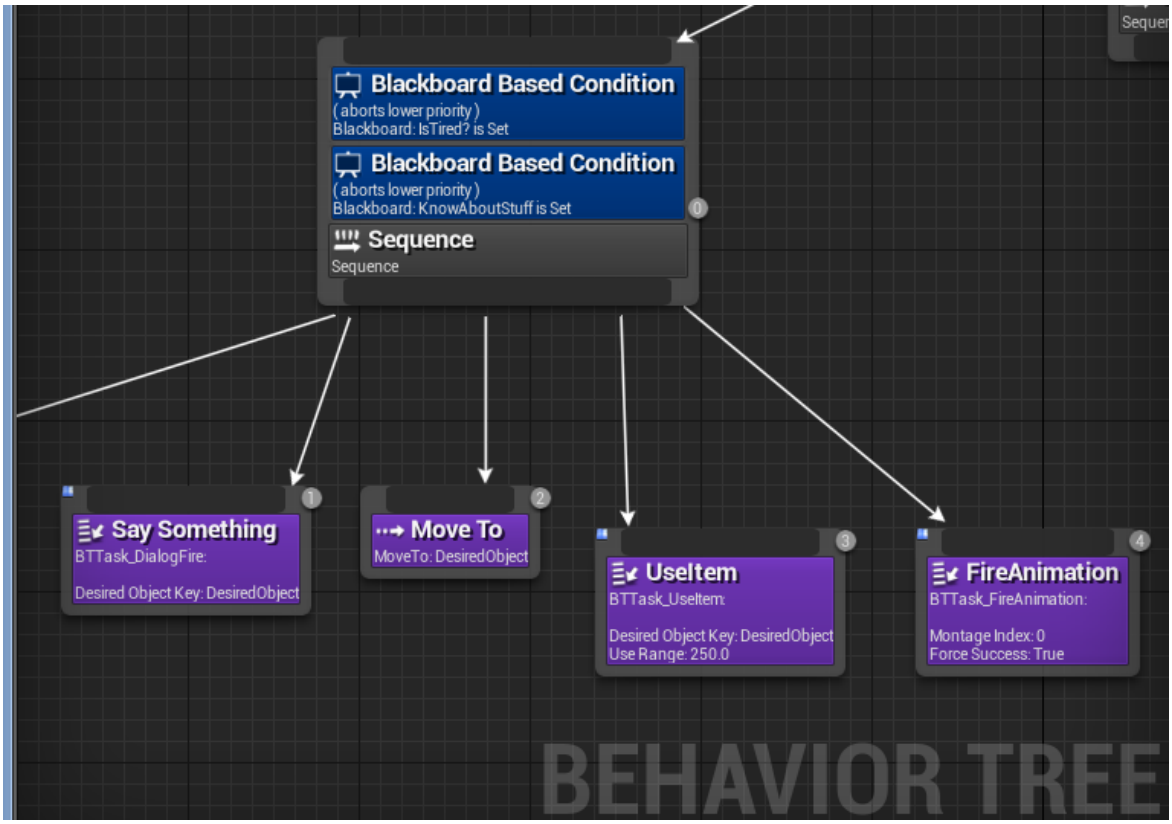


Εικόνα 4.7.1: Συναρτήσεις ως Ορίσματα μέσα στο *BP Interface.*

Από κάτω ακολουθεί το όρισμα που δώσαμε μέσα στο *Interface*. Αναλυτικότερα πρόκειται για ένα αντικείμενο που θα είναι σε θέση να αξιοποιήσει τις παραπάνω συναρτήσεις.

Εικόνα 4.7.2: Το αντικείμενο*(Actor)* του μήλου ως *Input.*

Παρακάτω ακολουθεί η εικόνα με την τοποθέτηση του έργου που χρησιμοποιεί το *Interface* μας μέσα στο *Behavior Tree*:



Εικόνα 4.7.3: Η πρώτη σειρά κόμβων στο *BT* της δεύτερης οντότητας.

Το Έργο λοιπόν ***UseItem*** είναι εκείνο που μέσω κώδικα σε *Blueprint* που παραθέτεται στο παράρτημα Β' μας δίνει τον όρο της **επιλογής** ως προς την τροφή που θα επιλέξουμε. Ουσιαστικά όσων αφορά το προγραμματιστικό κομμάτι θέτουμε κάποια κριτήρια έτσι ώστε να επιτρέψουμε ή όχι στην οντότητα μας να καταναλώσει το μήλο. Παρατηρώντας τα ορίσματα του αναγνωρίζουμε το *DesiredObject* ως το μήλο που επιθυμεί να καταναλώσει η οντότητα μας αλλά και την απόσταση που πρέπει να έχει από αυτό έτσι ώστε να την καταναλώσει. Ένας τυπικός αριθμός που οπτικά δίνει την ψευδαίσθηση πώς η οντότητα μας είναι δίπλα στο μήλο και το καταναλώνει είναι τα *250 Unreal Units(25cm)*.

Δύο άλλοι επίσης κόμβοι που πρόσθεσα αφορούν την οπτικοακουστική ανατροφοδότηση ως προς τα μάτια του παίχτη έτσι ώστε να κατανοήσει πώς κάτι συνέβη.

Ο κόμβος ***Say Something*** χρησιμοποιεί ένα ηχητικό αρχείο που ακούγεται καθώς ο παίχτης πλησιάζει το μήλο ενώ ο ***FireAnimation*** αποτελείται από ένα *clip* που αναγκάζει την οντότητα μας να πέφτει στο έδαφος μόλις επίσης καταναλώνει ένα μήλο.

4.8 Απεικόνιση Ιδιοτήτων.

Αναφερόμενος στην πείνα που θα αναγκάσει τον παίχτη να αναζητήσει τροφή θα ήταν ορθό να παρουσιάσουμε και ένα είδος οπτικής πληροφορίας που θα βλέπουμε.

Σε αυτό το σημείο έγινε χρήση του εργαλείου ***UI Widget*** που μας παρέχει η *Unreal Engine*. Αυτό το εργαλείο "τραβάει" πληροφορίες από την οντότητα μας σχετικά με την πείνα, την υγεία του και άλλες ιδιότητες που πραγματοποιήθηκαν μέσω *Blueprint* και παραθέτονται στο Παράρτημα Β'.

Το *UI* μας αναπαρίσταται πάνω από την οντότητα ως εξής:

Εικόνα 4.8.1: Αναπαράσταση των *Stats* μέσω *UI Widget.*

Παρατηρούμε μία πληθώρα από χαρακτηριστικά που έχουμε εισάγει στον παίχτη. Η λογική είναι πώς όταν το πεδίο ***"Hunger"*** φτάσει στο **0** θα ξεκινήσει να μειώνεται και το πεδίο ***"Stamina"*** και έπειτα το πεδίο ***"Health"***. Όταν φτάσουμε στο σημείο 0 για την πείνα τότε ο παίχτης θα αρχίσει να καταναλώνει την τροφή του. Όλη η διαδικασία μεταφέρθηκε σε ένα οπτικό αποτέλεσμα για να καταλαβαίνουμε πότε η οντότητα μας έχει φτάσει στο σημείο πείνας. Το εν λόγω σύστημα μας βοηθάει να καταλάβουμε πότε κάτι δεν λειτουργεί σωστά. Οι **ιδιότητες** αυτές έχουν προγραμματιστεί έτσι ώστε να **μειώνονται σταδιακά**, οδηγώντας την οντότητα στην αναζήτηση τροφής.

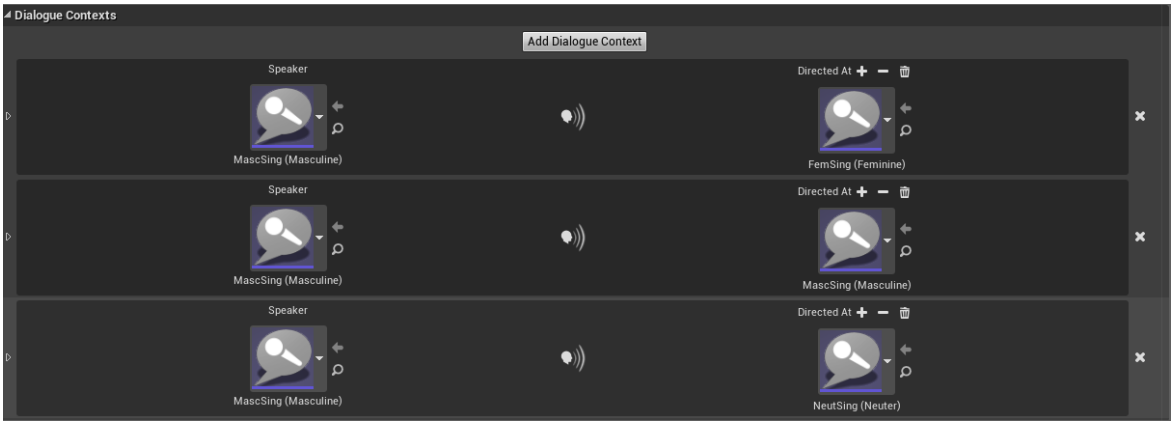
4.9 Ήχος και *Animation.*

Ξεκινώντας την συγγραφή της παρούσας πτυχιακής εργασίας, ένας από τους πρώτους στόχους μου ήταν να προσθέσω ένα σύστημα που θα έδινε την ψευδαίσθηση στον παίχτη πώς η οντότητα μας μπορεί να αναγνωρίσει **καταστάσεις** και **ανθρώπους** γύρω της. Αυτός ο στόχος λοιπόν ταυτίστηκε σύντομα με την εισαγωγή ενός **ηχητικού συστήματος** που μας παραθέτει η *Unreal Engine*. Η παραδοχή θα ήταν απλή:
Να χρησιμοποιηθούν τρία διαφορετικά κομμάτια ήχου*(Audio Clips)* για τρείς διαφορετικές περιπτώσεις.
Το πρώτο *Audio Clip* θα αναφέρονταν σε κάποια οντότητα γένους **αρσενικού**, το δεύτερο σε γένος **θηλυκού** και το τρίτο σε ένα **ουδέτερο** γένος. Αρχικά σκέφτηκα το εξής παράδειγμα που έπειτα οδηγήθηκε ορθά και στην υλοποίηση του.
Έστω πώς η οντότητα μας αναζητά χαρακτήρες ασχέτως του φύλου τους, μέσα στο περιβάλλον μας. Εφόσον είχε επιτύχει τον στόχο της, θα ακούγαμε τρία διαφορετικά *Audio Clips* ανάλογα την περίσταση.

- Αν η οντότητα που αναζητούσαμε ήταν γένος αρσενικού τότε θα ακούγαμε την έκφραση **Τον** βρήκα.
- Αντίστοιχα αν ήταν γένος θηλυκού θα ακούγαμε την έκφραση **Την** βρήκα
- Τέλος, αν αναζητούσαμε ένα ουδέτερο αντικείμενο θα ακούγαμε την έκφραση **Το** βρήκα.

Χρησιμοποιώντας λοιπόν έξυπνα τα ηχητικά μας κομμάτια θα είμαστε σε θέση να δημιουργήσουμε μία οντότητα που πέρα των άλλων αρμοδιοτήτων της θα μπορεί να **αναγνωρίζει** το φύλο των χαρακτήρων ή αντικείμενων δίπλα της. Παρόλο που δημιούργησα τρία ηχητικά κλιπ μέσα στο ηχητικό μας σύστημα, το μόνο που κατάφερα να χρησιμοποιήσω είναι η φράση "**Το** βρήκα" καθώς η οντότητα μας αναγνωρίζει ένα μήλο που πρέπει να καταναλώσει. Το καλύτερο σημείο όμως δεν κρύβεται εκεί. Αντιθέτως το εν λόγω σύστημα θα μπορούσε κάλλιστα να αποτελέσει την αρχή για ένα **σύστημα διαλόγων** μέσα σε ένα παιχνίδι όπου η Τ.Ν μας θα είναι σε θέση να ανταποκρίνεται στον παίχτη ανάλογα το φύλο του χαρακτήρα που παίζει.

Συνεχίζοντας λοιπόν ακολουθούν μερικές εικόνες από την λειτουργία του συστήματος:



Εικόνα 4.9.1: Το σύστημα *Dialogue Wave(1/2).*

Αριστερά παρατηρούμε την λίστα με τα "ηχεία" *(speakers)* καθώς και την τονικότητα της φωνή μας που μπορεί να χωριστεί σε τρία μέρη:

- *Masculine*
- *Feminine*
- *Neuter*

Δεξιά διακρίνουμε την στήλη *Directed At* που στην ουσία επισημαίνει που απευθυνόμαστε. Στην πρώτη περίπτωση απευθυνόμαστε σε ένα γυναικείο πρόσωπο, στην δεύτερη σε ένα αντρικό και στην συνεχεία σε ένα ουδέτερο.

Εφόσον έχουμε αντιστοιχήσει ορθά τις λίστες μας εισάγουμε το ηχητικό κύμα*(Sound Wave)* που **ηχογραφήθηκε** από εμένα και "παίζει" την ανάλογη

φράση. Στην περίπτωση λοιπόν που εμείς απευθυνόμαστε σε ένα γυναικείο πρόσωπο θα εισάγουμε το *Sound Wave* που περιέχει την έκφραση Την βρήκα:

Εικόνα 4.9.2: Το σύστημα *Dialogue Wave*(2/2).

Ας μην ξεχνάμε όμως πώς η οντότητα μας αναζητά ένα αντικείμενο ουδετέρου γένους. Εφόσον "στήσαμε" το σύστημα διαλόγων, το συνδέσαμε με την οντότητα μας μέσω ενός *Blueprint* που παραθέτεται στο Παράρτημα Β'.

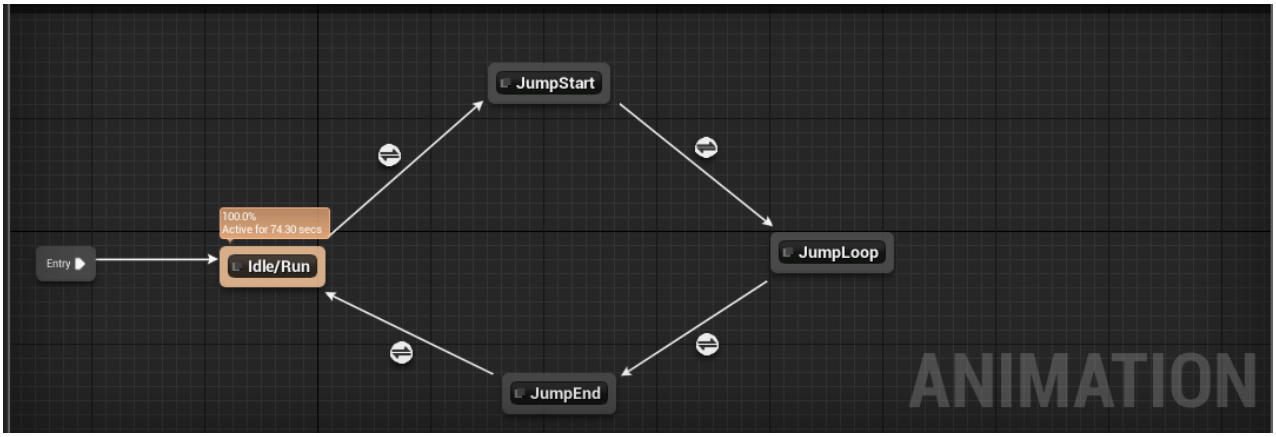
Το ***Animation*** μέσα στην *Unreal Engine* αποδείχθηκε αρκετά δύσκολη διαδικασία. Κάτι που θεωρείται λογικό αν αναλογιστεί κανείς πώς οι *3D Animators* αποτελούν έναν τεράστιο ξεχωριστό κλάδο στον χώρο εργασίας των παιχνιδιών. Έχοντας όμως καλύψει την ηχητική ανατροφοδότηση ξεκίνησα να εργάζομαι επάνω στην κίνηση της οντότητας μας. Το σχέδιο μου απλά αποτελείται από μία **κίνηση(ξάπλωμα)** που πραγματοποιεί η Τ.Ν μας μόλις έρθει σε **επαφή** με την τροφή της για να καταφέρω δείξω το αποτέλεσμα αυτής της πράξης.

Το *Animation* λοιπόν αποτελείται από τα **τρία βασικά στοιχεία**. Το *3D* μοντέλο που θα πραγματοποιεί τις κινήσεις μας:

- Το *3D* μοντέλο που θα πραγματοποιεί τις κινήσεις μας.
- Το *Animation Tab* που μας επιτρέπει να προσθέτουμε και να βλέπουμε τα *animations* στο μοντέλο μας, αν φυσικά η ανατομία του μας το επιτρέπει.
- Το *Animgraph* που αποτελείται από διάφορα ***State Machines*** και τα ***AnimBlueprints*** που μας επιτρέπουν να συνδέσουμε τις κινήσεις του χαρακτήρα μας με το προγραμματιστικό κομμάτι του παιχνιδιού.

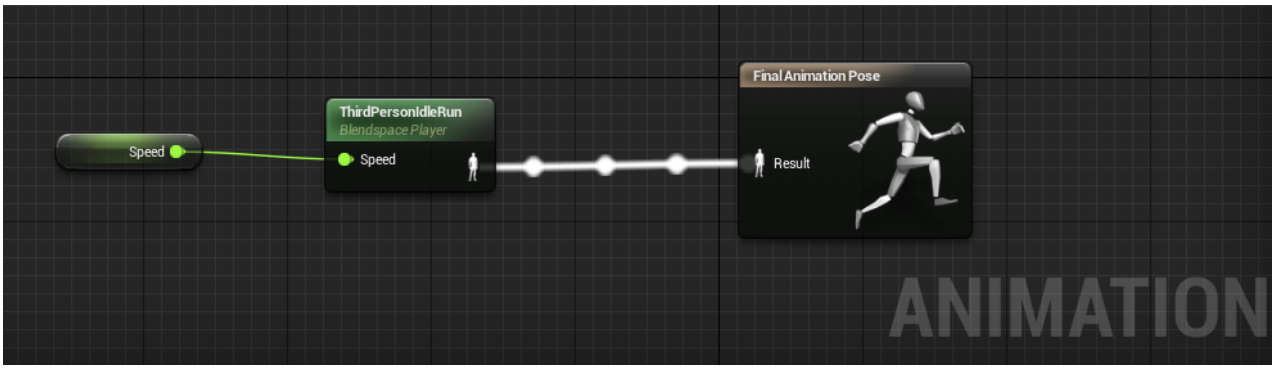
Αν και τα *Behavior Trees* παρουσιάζουν **σημαντικές ομοιότητες** με τα *State Machines,* παρατηρήσαμε πώς το κομμάτι του *Animation* αποτελείται από αυτές.

Παρακάτω ακολουθεί ένα *Animgraph* καθώς και τις απαραίτητες καταστάσεις που μεταδίδονται στο παίχτης μας:



Εικόνα 4.9.3: Οι καταστάσεις*(states)* στο *Animation.*

Στην εικόνα 5.9.3 παρατηρούμε τα διάφορα states που μπορεί να προβεί ο χαρακτήρας μας κατά την διάρκεια της κίνησης του.
Αναλύοντας τον κόμβο ***Idle/Run*** θα αναγνωρίσουμε άλλο ένα Blueprint που μέσω της μεταβλητής *Speed* θα καθορίσει και την ταχύτητα που θα τρέξει ο παίχτης.



Εικόνα 4.9.4: *Idle/Run Node.*

Τέλος μπορούμε να αναλύσουμε τα *animations* μας ***καρέ-καρέ*** για τυχόν σφάλματα που μπορεί να παρουσιάζονται. Η παρακάτω εικόνα αναλύει τους κόμβους από *JumpStart --> Jump Loop --> JumpEnd.*

Εικόνα 4.9.5: Οπτική Αναπαράσταση των *Jump Nodes.*

Στο κάτω μέρος της εικόνας παρατηρούμε τον Editor για τα καρέ έτσι ώστε να αναλύσουμε καλύτερα την κίνηση μας.

ΚΕΦΑΛΑΙΟ 5

ΕΙΣΑΓΩΓΗ

Σε αυτό το κεφάλαιο παραθέτονται η συνοπτική περιγραφή της εργασίας μετά το πέρας της καθώς και σκέψεις και συμπεράσματα μετά την ολοκλήρωση της.

5.1 Συνοπτική Περιγραφή

Η παρούσα πτυχιακή εργασία ολοκληρώθηκε μετά από επίπονες προσπάθειες και πολύ χρόνο. Αρχικά μελετήσαμε το υπόβαθρο της Τ.Ν καθώς και ιστορικούς αλγόριθμους που χρησιμοποιούνται μέχρι σήμερα και πώς αυτοί μπορούν να εφαρμοστούν σε ένα περιβάλλον παιχνιδιού μετά από **διεξοδική έρευνα** των πηγών μου.

Το επόμενο κομμάτι αφορούσε έναν αρκετά **δημιουργικό τομέα** που είναι η δημιουργία ενός ρεαλιστικού περιβάλλοντος μέσα στην *Unreal Engine* καθώς και την ανάλυση των εργαλείων που παρέχει η μηχανή όπως την δημιουργία των δικών μας υλικών , την μορφοποίηση του περιβάλλοντος μας αλλά και την προσθήκη των *Post Process Effects*. Ο μοναδικός μας στόχος ήταν ένα **θεαματικό αποτέλεσμα** που επετεύχθη .

Εφόσον το περιβάλλον μας ήταν έτοιμο πραγματοποιήσαμε τον **σχεδιασμό** των δύο οντοτήτων. Ο σχεδιασμός περιελάμβανε την ανάθεση αρμοδιοτήτων στις οντότητες μας αλλά και βασικά χαρακτηριστικά της Τ.Ν όπως οι έννοιες της επιλογής, της μνήμης της όραση και της ακοής. Αλλά και την αντιστοίχιση των απαραίτητων εργαλείων της *Unreal Engine* για την σωστή **υλοποίηση** τους.

Προχωρώντας στο τελευταίο μέρος της εργασίας εξετάσαμε πως αυτά τα εργαλεία "συνεργάζονται" και γνωρίσαμε κρίσιμες έννοιες όπως τι είναι τα *Behavior Trees, το Blackboard, τα Blueprints, τα Interfaces* και τα *Environmental Query Systems* που αποδείχθηκαν σημαντικά για την σωστή και βέλτιστη λειτουργία των οντοτήτων μας.

Έπειτα αναφερθήκαμε σε δευτερεύοντες διαδικασίες όπως η εισαγωγή ηχητικών εφέ και *animations* που αποδειχθήκαν αρκετά απαιτητικές με μόνο στόχο την **ανατροφοδότηση** στα μάτια του παίχτη, οπτική και ακουστική.

5.2 Συμπεράσματα.

Τελειώνοντας την παρούσα πτυχιακή εργασία και συνοψίζοντας θα ήταν σωστό να αναλογιστούμε τους αρχικούς μας στόχους και εάν καταφέραμε να τους επιτύχουμε. Πράγματι, καταφέραμε να μεταφέρουμε ορθώς την θεωρητική γνώση που μελετήσαμε μέσα στο περιβάλλον της *Unreal Engine*. Επίσης, σπουδαιότερο κατόρθωμα της πτυχιακής εργασίας αποτέλεσε το γεγονός πώς καταφέραμε να "χτίσουμε" ένα **γενικό σύστημα** που θα μπορέσει να αποδειχθεί χρήσιμο σε ποικίλες καταστάσεις ανάπτυξης Τ.Ν και όχι σε μεμονωμένα περιστατικά.
Αναφερόμενος όμως σε ψηφιακά παιχνίδια και από τι πραγματικά απαρτίζονται, συναντήσαμε αρκετές δυσκολίες σε άλλα κομμάτια ζωτικής σημασίας όπως το *animation* και ο *ήχος* που δυστυχώς μπορούν εύκολα να καταστρέψουν την εμπειρία που προσφέρουμε στον παίχτη. Όπως αναλύσαμε στην αρχή ένα ψηφιακό παιχνίδι αποτελείται κυρίως από την διασκέδαση και την εμπειρία που προσφέρουμε στον παίχτη και όχι από τους πολύπλοκους αλγόριθμους που αυτός δεν βλέπει. Συνεπώς θα πρέπει να αναλογιστεί κανείς πώς ακόμα και το πιο ισχυρό σύστημα Τ.Ν κρίνεται αδιάφορο αν δεν παρέχει την οπτικοακουστική ανατροφοδότηση που θα ενθουσιάσει τον παίχτη. Όσο βασικό χαρακτηριστικό να είναι πλέον η Τ.Ν, ένα ψηφιακό παιχνίδι απαιτεί τεράστια ομαδική προσπάθεια, έτσι ώστε να φανούν όλοι οι κλάδοι του, γεγονός που καταστεί την δημιουργία ενός τέτοιου έργου πολύ σημαντική και δύσκολη υπόθεση.

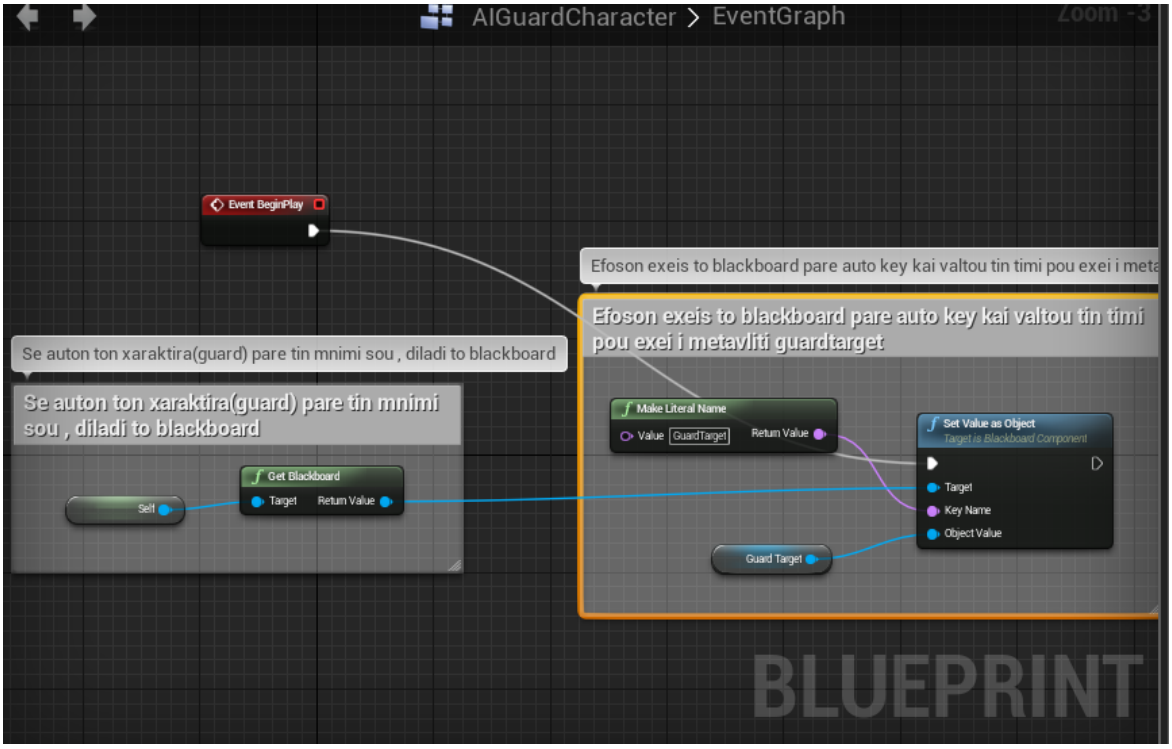
Παρόλη την **τεχνολογική ανάπτυξη** που βιώνει όμως η ανθρωπότητα τα τελευταία χρόνια στον κόσμο των υπολογιστικών συστημάτων παρατηρούμε πώς η Τεχνητή Νοημοσύνη για τα ψηφιακά παιχνίδια δεν έχει εξελιχθεί ανάλογα. Τα πιο δημοφιλή παιχνίδια δεν γίνονται γνωστά λόγω της "έξυπνης" Τ.Ν αλλά αντιθέτως για τα υπέροχα γραφικά που μπορούν να δελεάσουν τον παίχτη. Με το πέρασμα του χρόνου όμως τα *game environments* γίνονται ολοένα και πιο ρεαλιστικά με την Τ.Ν να μην καταφέρνει να φτάνει τα ίδια υψηλά στάνταρ. Παρατηρούμε ολοένα και περισσότερους πανέμορφους ψηφιακούς κόσμους να δρουν θετικά στην έκβαση ενός τελικού προϊόντος. Κόσμοι όμως που δεν κατοικούνται από χαρακτήρες - οντότητες του ίδιου βελησνεκούς που θα καταφέρουν να οδηγήσουν τον **ρεαλισμό** στο επόμενο επίπεδο. Μία σοφή παραδοχή για την επίτευξη του ρεαλισμού θα ήταν η σχεδίαση και η ανάπτυξη της

Τ.Ν χωρίς να υπολογίζουμε τον παίχτη. Με αυτόν τον τρόπο ο παίχτης δεν θα αποτελεί το κέντρο του κόσμου, αλλά θα πρέπει προσαρμοστεί στον κόσμο του παιχνιδιού όπως προσαρμόζεται και στην πραγματική κοινωνία και όχι αντίστροφα όπως συμβαίνει σε πολλά παιχνίδια.

Την συγκεκριμένη περίοδο που διανύουμε όμως κάτι τέτοιο είναι δυνατόν να αλλάξει. Έχοντας εισέλθει στην εποχή της **Εικονικής Πραγματικότητας**, μία ωραία εμπειρία μέσα σε ένα παιχνίδι θα πάψει να αποτελείται μόνο από ένα όμορφο τοπίο. Είναι ήδη γνωστό πώς στα επόμενα χρόνια η έννοια της ηλεκτρονικής ψυχαγωγίας θα αλλάξει ριζικά. Θα έχει να κάνει σε μεγάλο βαθμό με την **αντιμετώπιση** και **την αντίθεση** που θα βρίσκει ο παίχτης μέσα στο παιχνίδι. Η ανάγκη πλέον για μία πιστευτή συμπεριφορά και αντίδραση απέναντι στον παίχτη έχει γίνει μεγαλύτερη από ποτέ. Μέσω λοιπόν ανανεωμένων προγραμμάτων όπως η *Unreal Engine* αλλά που έχει αλλάξει σημαντικά τον τρόπο που υλοποιείται μία μορφή τεχνητής νοημοσύνης αλλά και περεταίρω τεχνολογικών ανακαλύψεων όπως το *Virtual Reality Headset* από διάφορες εταιρείες το μέλλον του πεδίου της Τεχνητής Νοημοσύνης για παιχνίδια διαγράφεται λαμπρό.

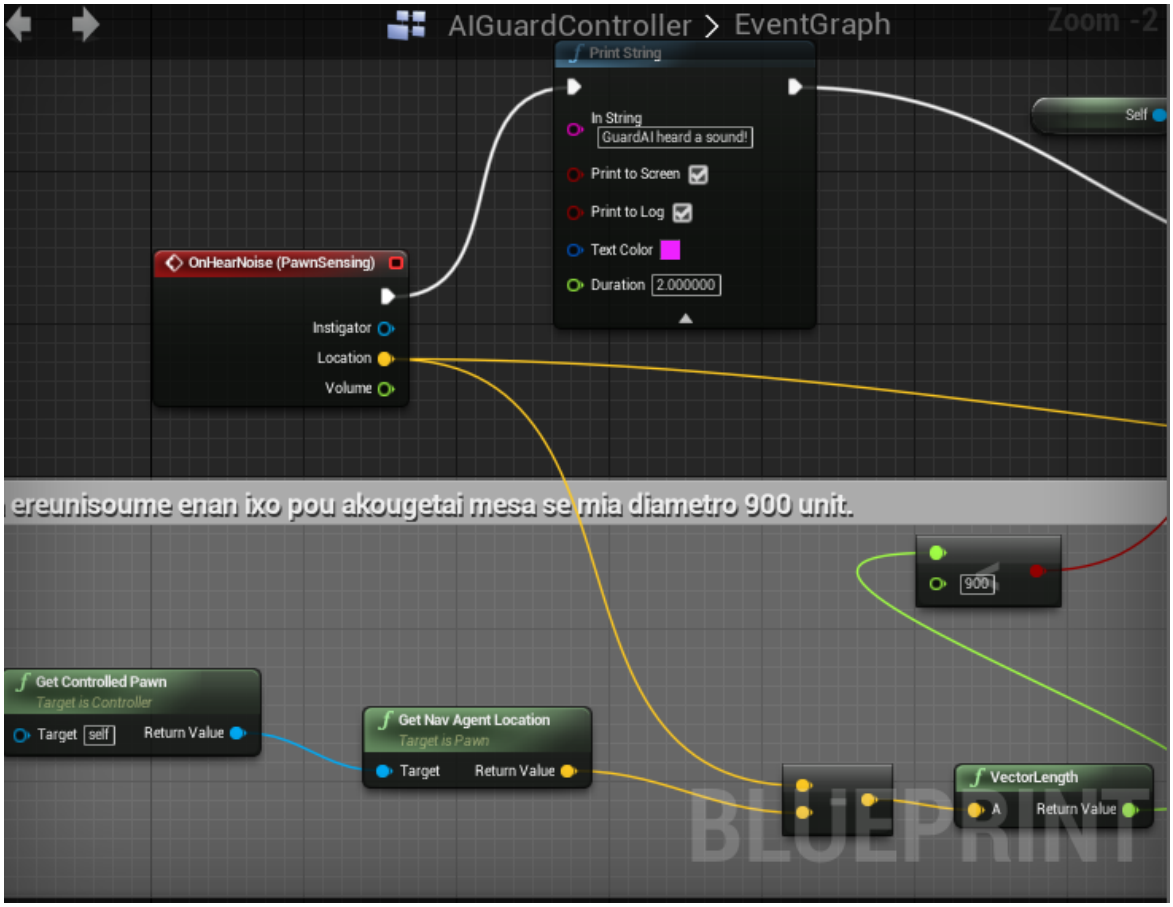
ΠΑΡΑΡΤΗΜΑ Α’

Στο παράρτημα αυτό παρατίθενται τα κρίσιμα Blueprints για την ορθή λειτουργία της πρώτης οντότητας μας, την *AI_Guard.*

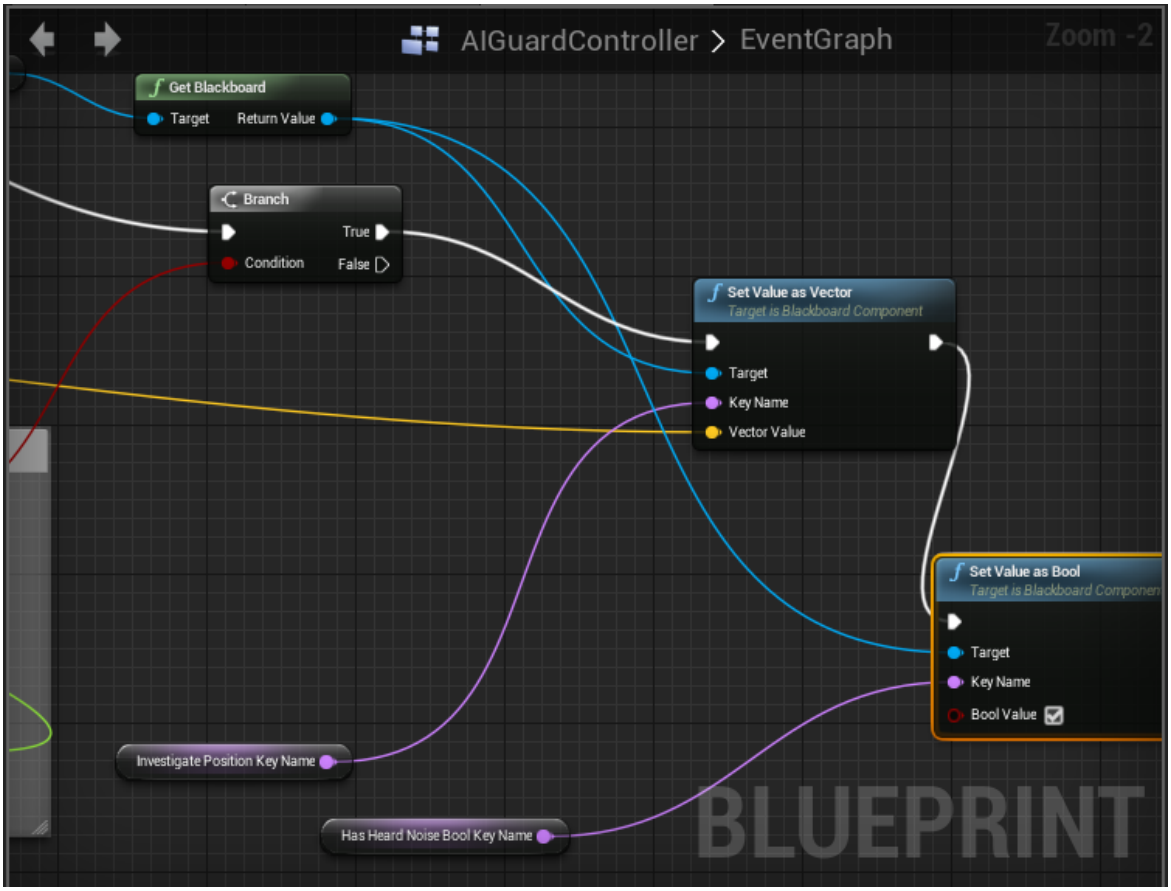


Εικόνα 6.1: Αναπαράσταση του *Blueprint* που έχουμε εισάγει μέσα στην οντότητα του φρουρού*(AIGuardCharacter).*

Στην εικόνα 6.1 ορίζουμε στο "σώμα" της οντότητας μας, την μεταβλητή που έχουμε δημιουργήσει μέσα στο *Blackboard* έτσι ώστε να θέσουμε το σημείο που πρέπει να φυλάει.

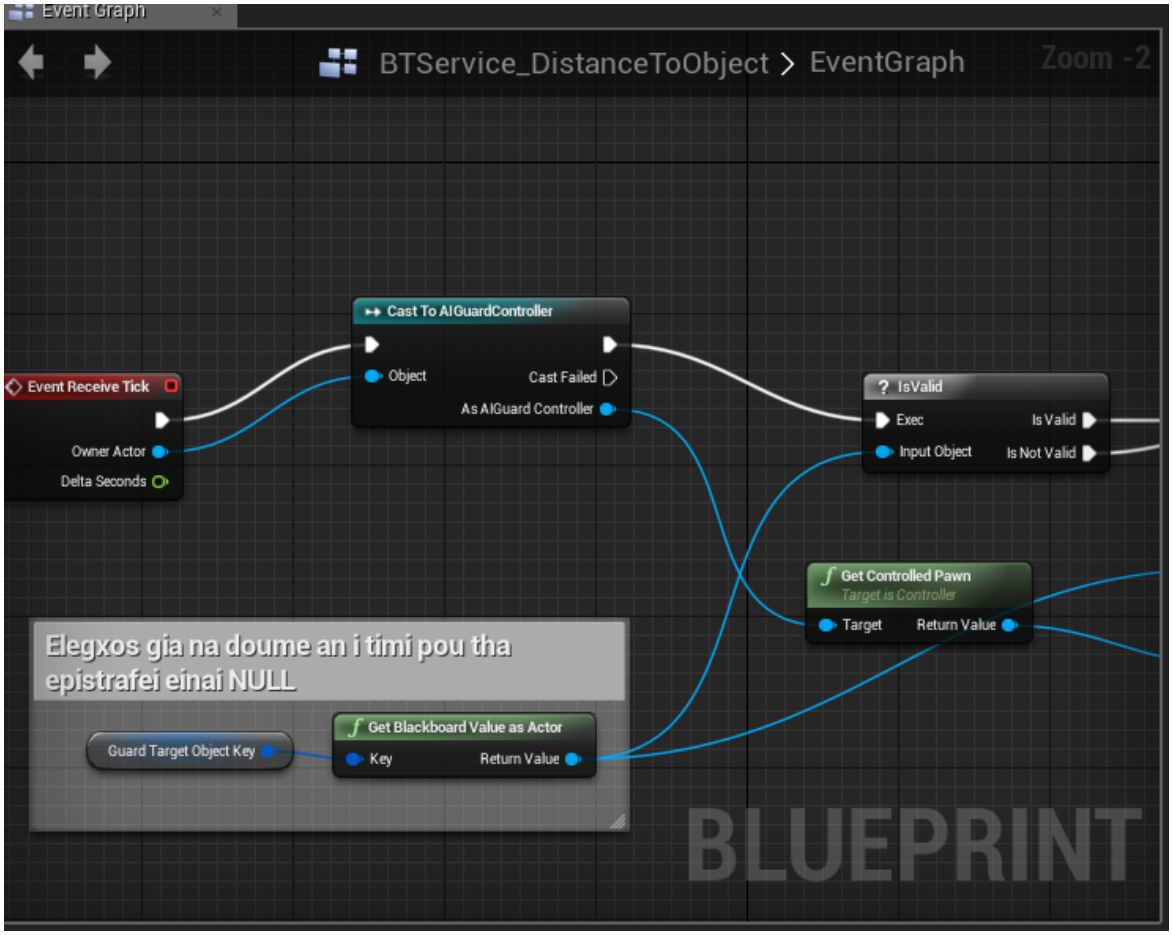


Εικόνα 6.2.1.: *Blueprint* για τον έλεγχο του ήχου καθώς και την αναγραφή μηνύματος αν αυτό επιτευχθεί.



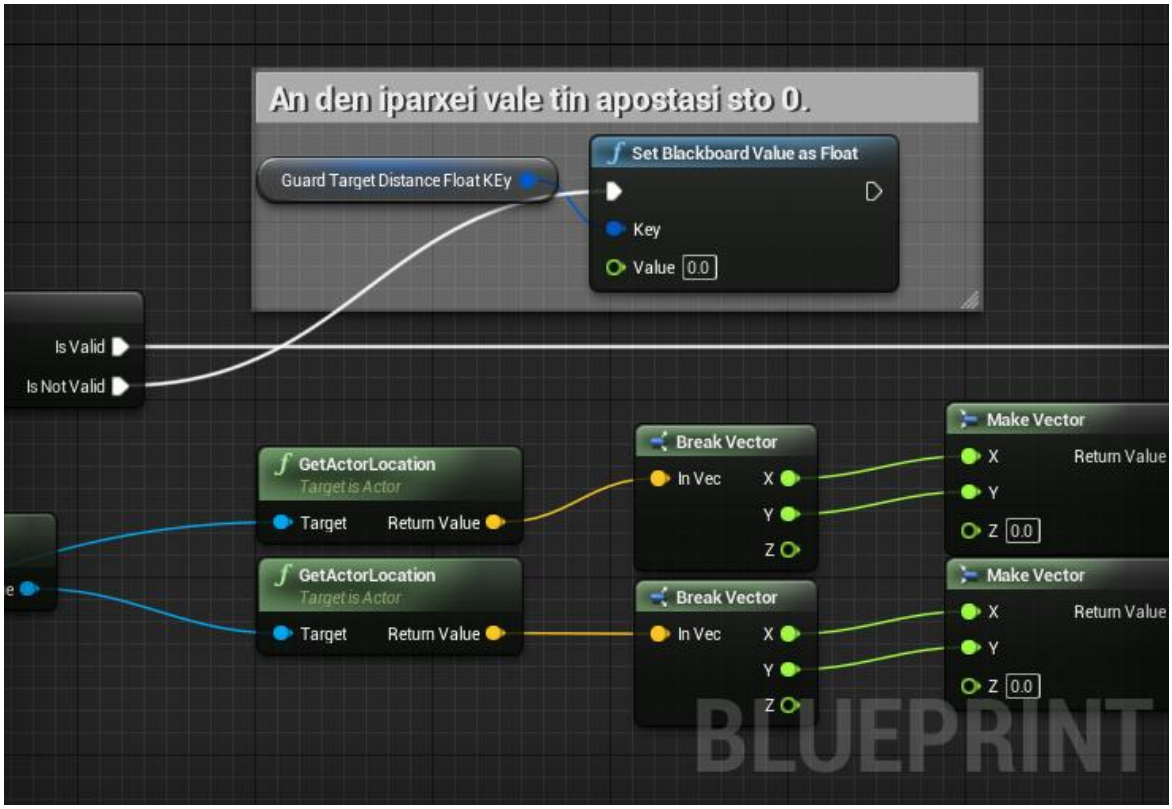
Εικόνα 6.2.2: Αναπαράσταση του Blueprint που ευθύνεται για την αναγνώριση της θέσης του παίχτη από την Οντότητα.

Η παραπάνω εικόνα αποτελεί την συνέχεια της 6.2.1 και ουσιαστικά ορίζει το σημείο που πρέπει να τρέξει ο Φρουρός μας μόλις δει τον παίχτη. Αυτό επιτυγχάνεται μέσω ενός *Vector.*

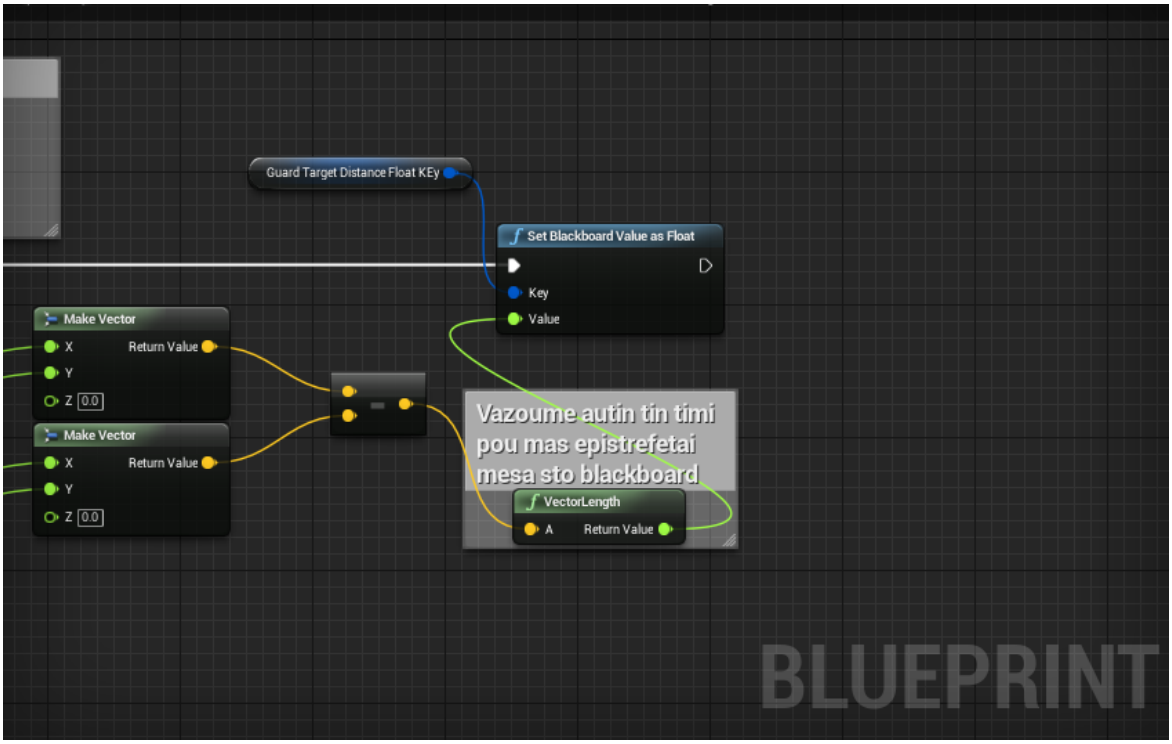


Εικόνα 6.3.1: Αναπαράσταση της Υπηρεσίας *Distance to Object.*

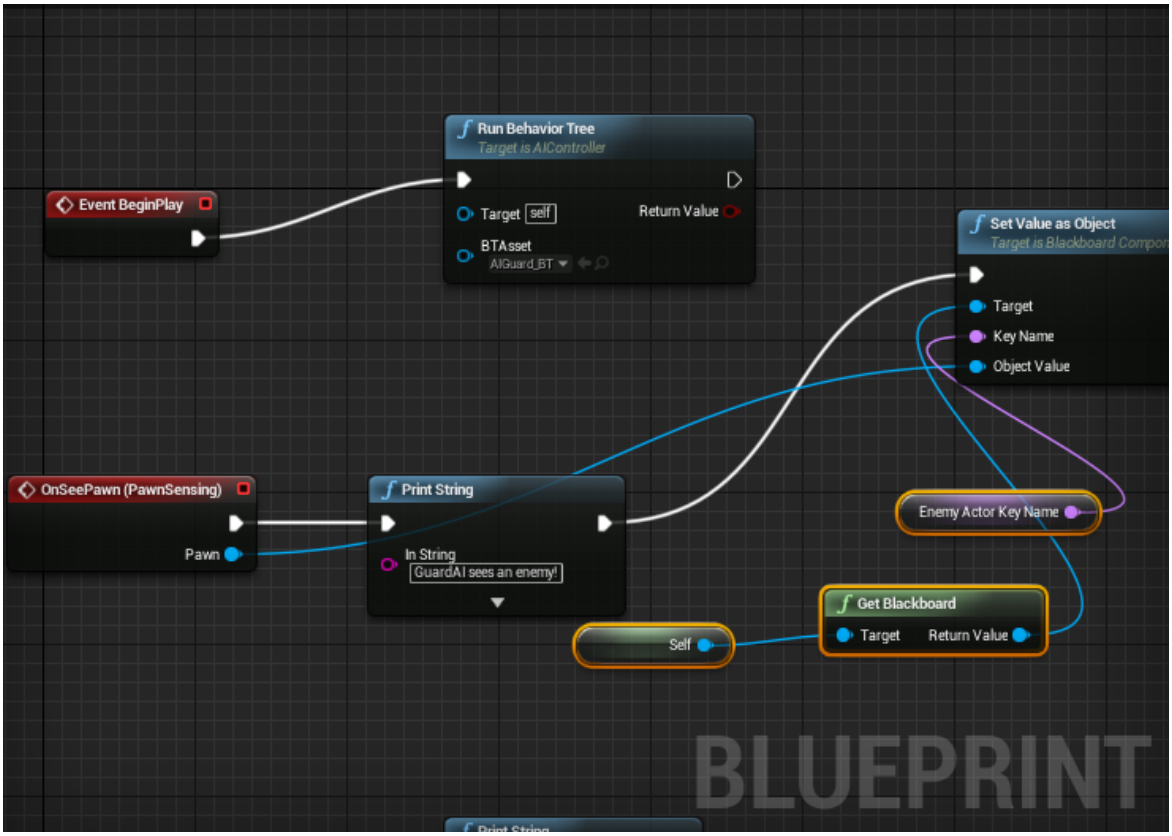
Κατευθύνει και μετράει την απόσταση του Φρουρού μας από το σημείο που πρέπει να φυλάξει. Αυτό γίνεται μέσω την αφαίρεση των δύο *Vector* χωρίς όμως να υπολογίζεται η διάσταση Z. Το αποτέλεσμα θα εκχωρηθεί στο τέλος μέσα στο *Blackboard* που θα μας δώσει την δυνατότητα να προχωρήσουμε από τα κριτήρια της Υπηρεσίας ή να αποτύχουμε.



Εικόνα 6.3.2: Συνέχεια της εικόνας 6.3.

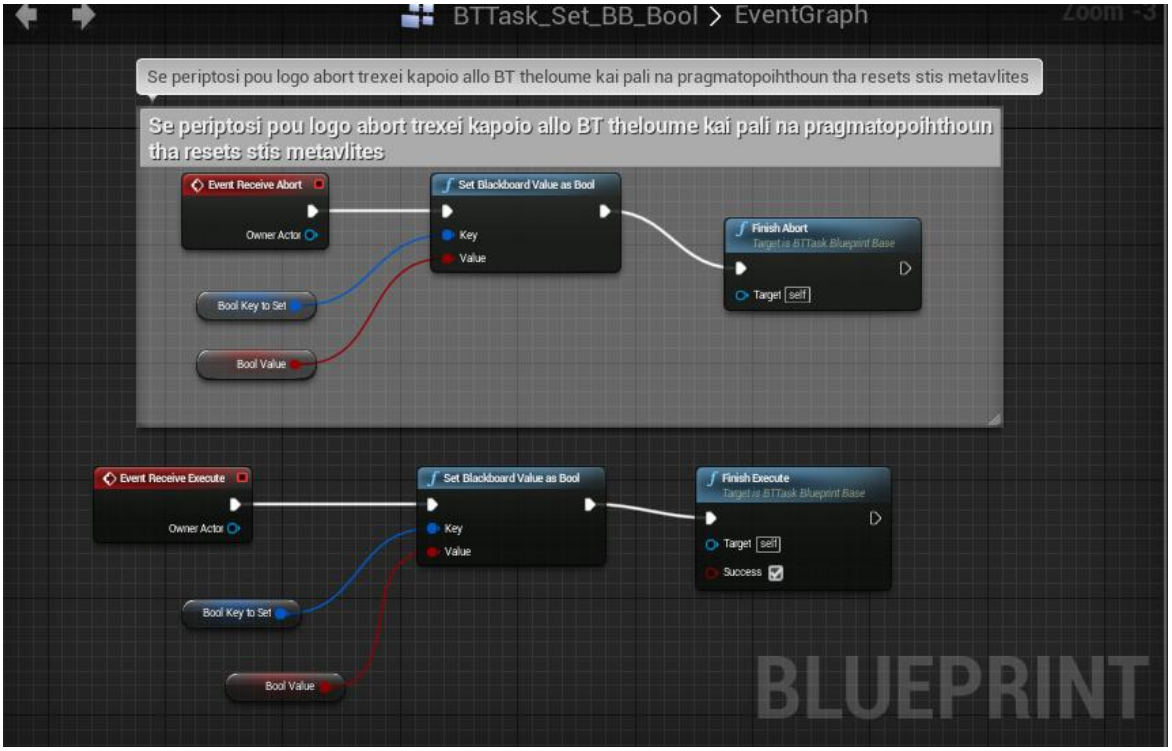


Εικόνα 6.3.3: Συνέχεια της εικόνας 6.3.1



Εικόνα 6.4: Αναπαράσταση του *BP* που βρίσκεται μέσα στον *AI_Controller* μας.

Παρατηρούμε πώς το *Behavior Tree* εκτελείται σε αυτό το κομμάτι παράλληλα με την εκτύπωση ενός μηνύματος εάν η οντότητα αντιληφθεί τον παίχτης μας.



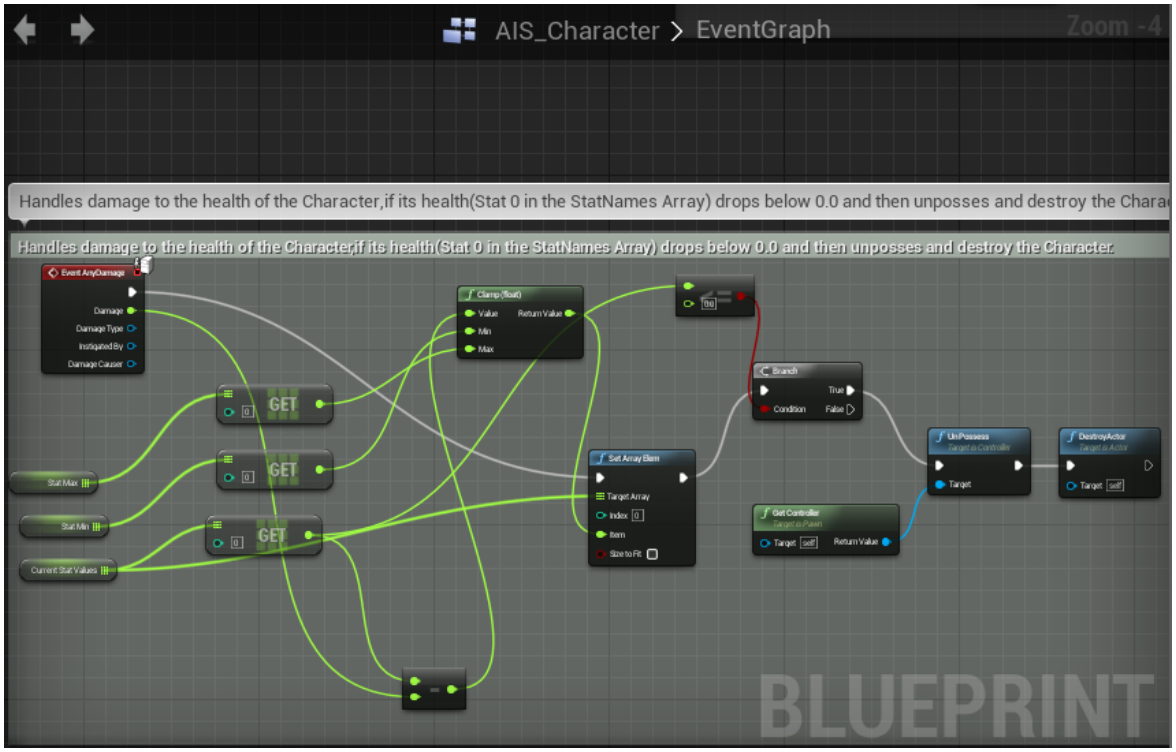
Εικόνα 6.5: Επανεκκίνηση των μεταβλητών.

.

Σε αυτή την εικόνας κάνουμε ***reset*** σε ορισμένες *Boolean* μεταβλητές ανεξαρτήτως αν θα εκτελεστούν τα Έργα σε ορισμένους κόμβους. Αυτό γίνεται γιατί παρότι δεν εισερχόμαστε σε ορισμένους κόμβους θα χρειαστεί αυτές οι μεταβλητές να πηγαίνουν στην αρχική τους κατάσταση.

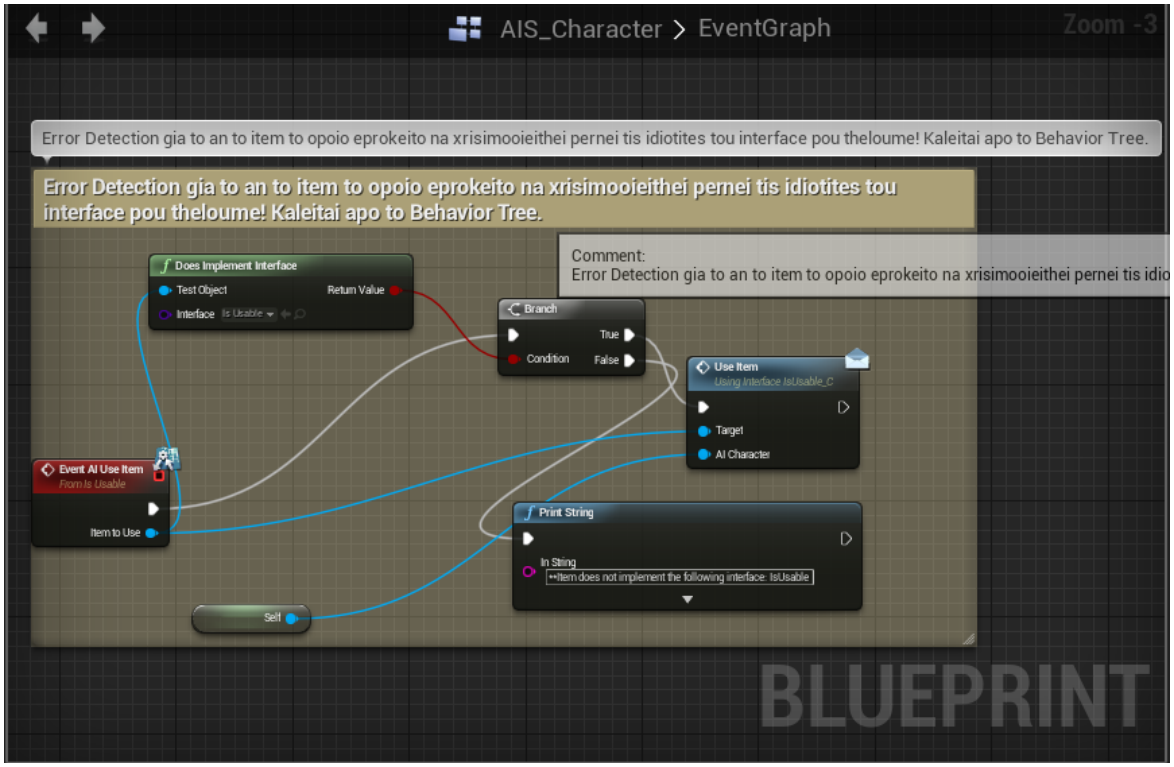
ΠΑΡΑΡΤΗΜΑ Β’

Σε αυτό το παράρτημα παραθέτονται τα *Blueprints* που χρησιμοποιήθηκαν για την λειτουργία της δεύτερης οντότητας. Λόγω του τεράστιου αριθμού των εικόνων έχουν συμπεριληφθεί μόνο τα κρίσιμα μέρη.



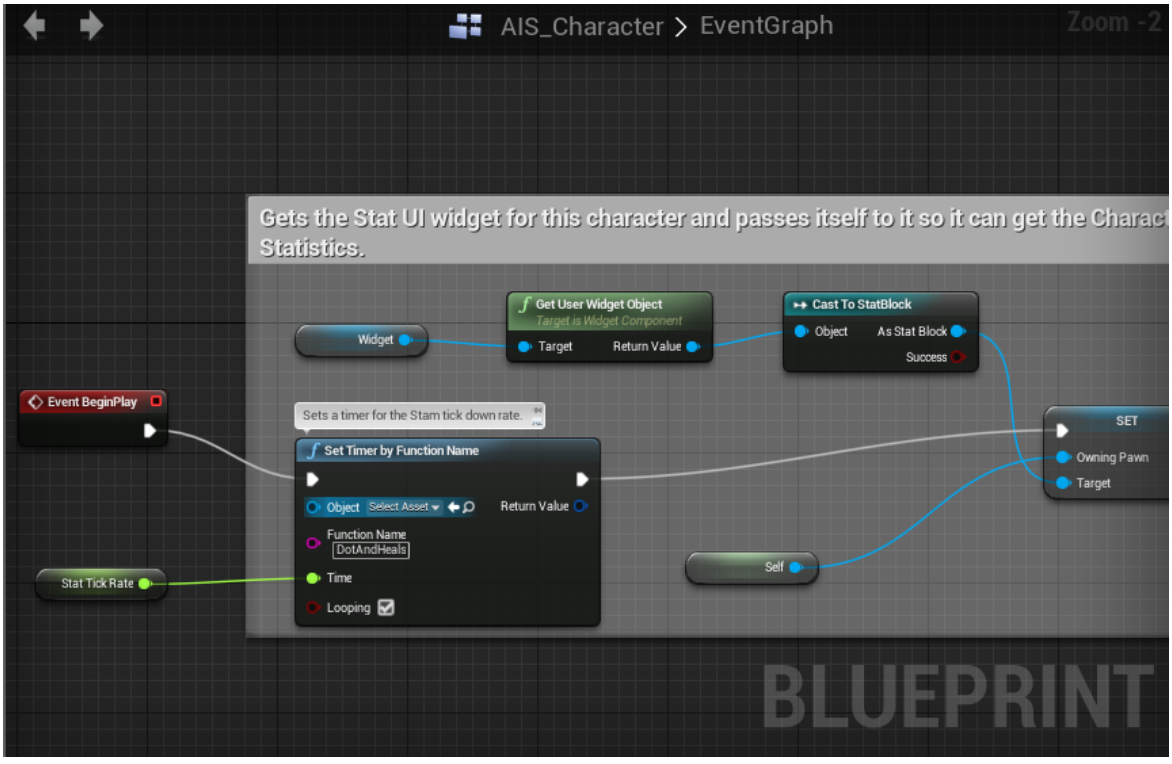
Εικόνα 7.1.1: Αναπαράσταση του *BP* που χειρίζεται την ζημιά της δεύτερης οντότητας.

Παρατηρούμε ήδη στην παραπάνω εικόνα ένα αρκετά πιο πολύπλοκο σύστημα σε σχέση με αυτό που είδαμε στην πρώτη οντότητα. Το εν λόγω *BP* χειρίζεται την ζημία που παθαίνει η οντότητα με το πέρασμα του χρόνου έτσι ώστε να τον αναγκάσει να φτάσει στο στάδιο πείνας και να αναζητήσει την τροφή.
Αυτό το σύστημα βρίσκεται μέσα στον *AIS_Character* μας αλλά δεν είναι το μόνο.

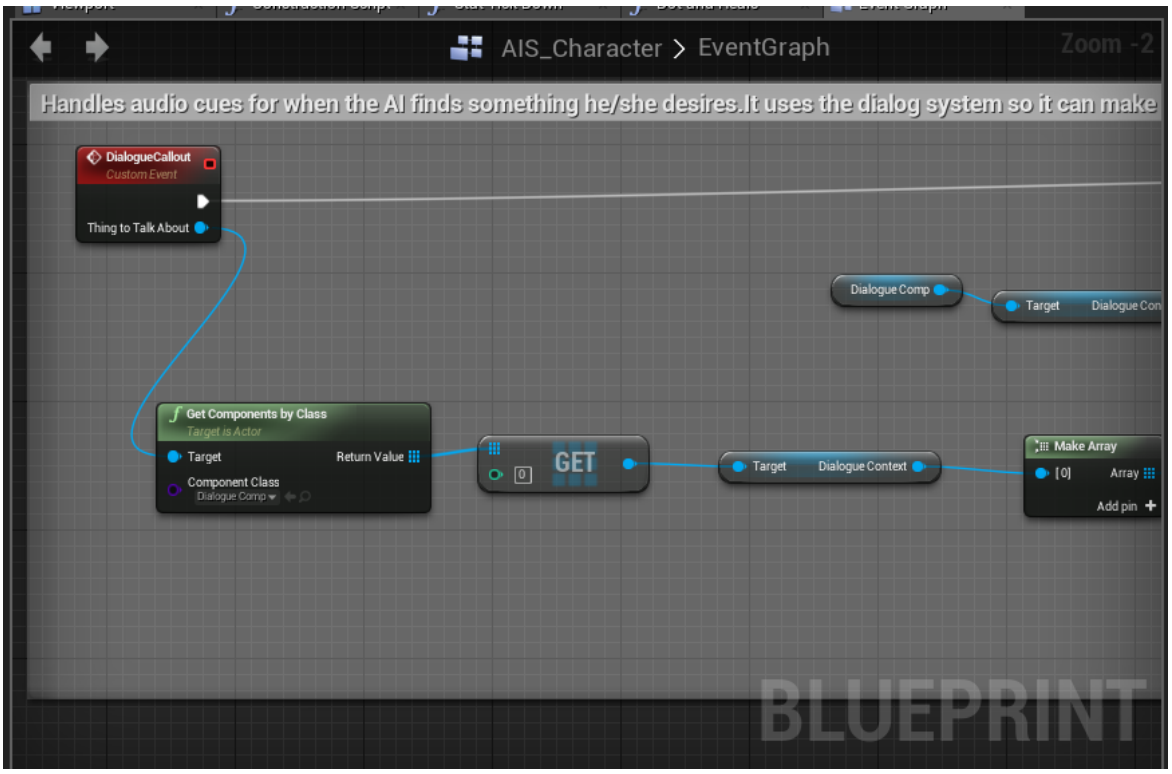


Εικόνα 7.1.2: Έλεγχος Σφαλμάτων για την τροφή.

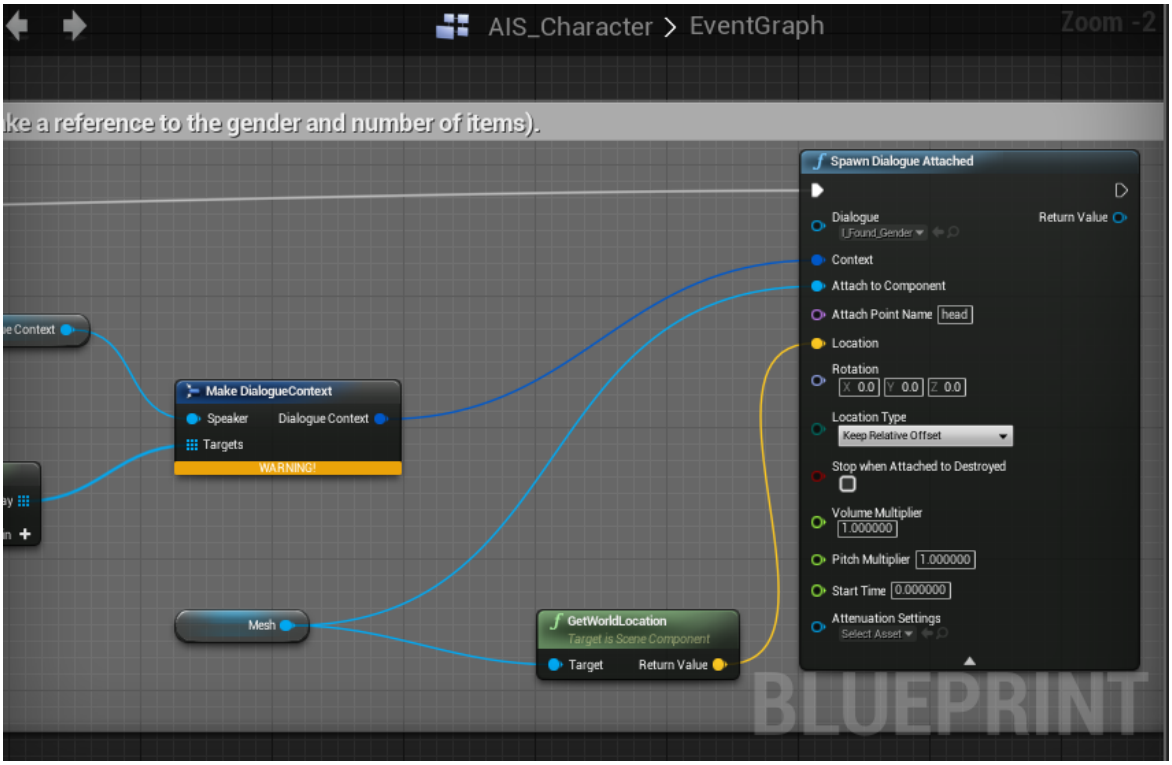
Σε αυτό το σημείο ελέγχουμε αν το μήλο που πρόκειται να καταναλωθεί διαθέτει τις κατάλληλες ιδιότητες έτσι ώστε η οντότητα μας να αλληλεπιδράσει μαζί του.



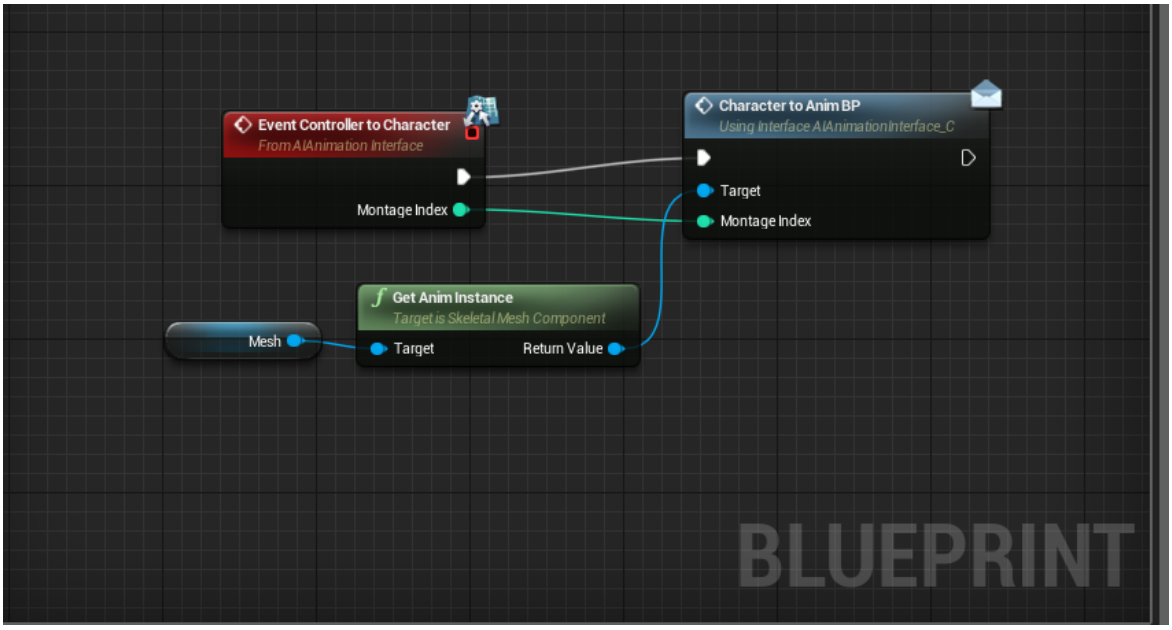
Εικόνα 7.1.3: Εισαγωγή του *User Interface* μέσω *BP.*



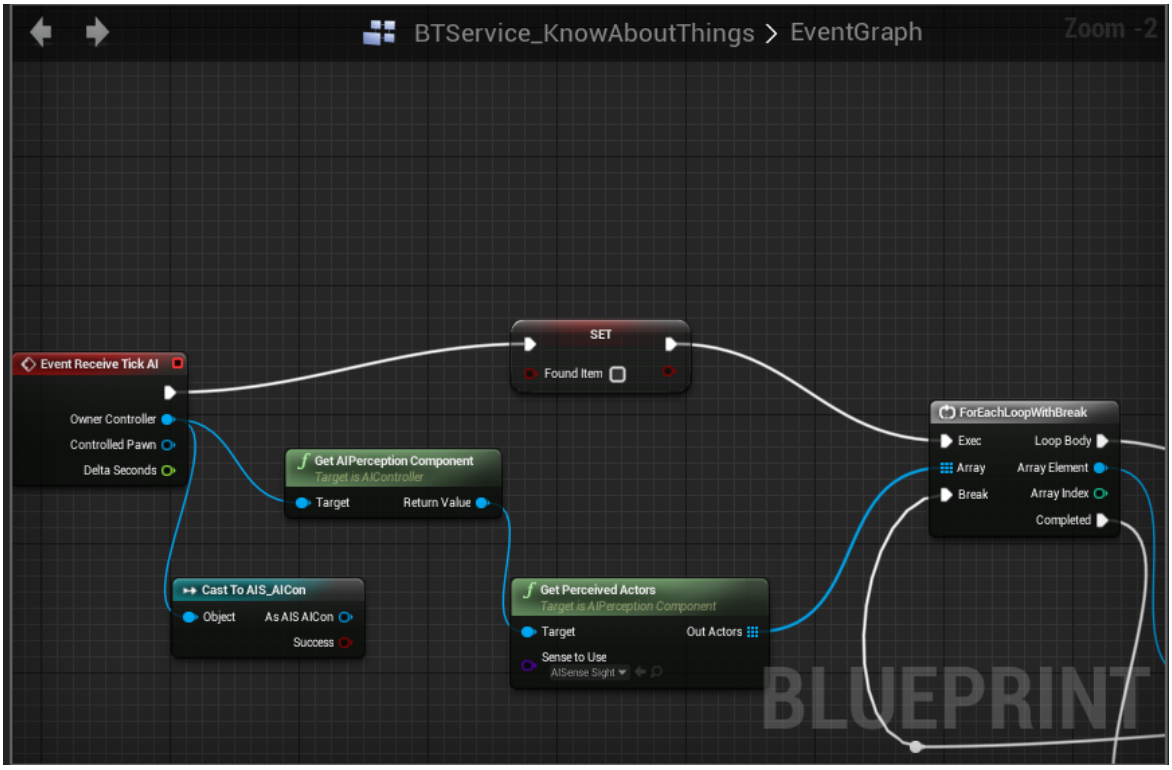
Εικόνα 7.1.4: Εισαγωγή του συστήματος διαλόγου μέσω *BP*. (1/2)



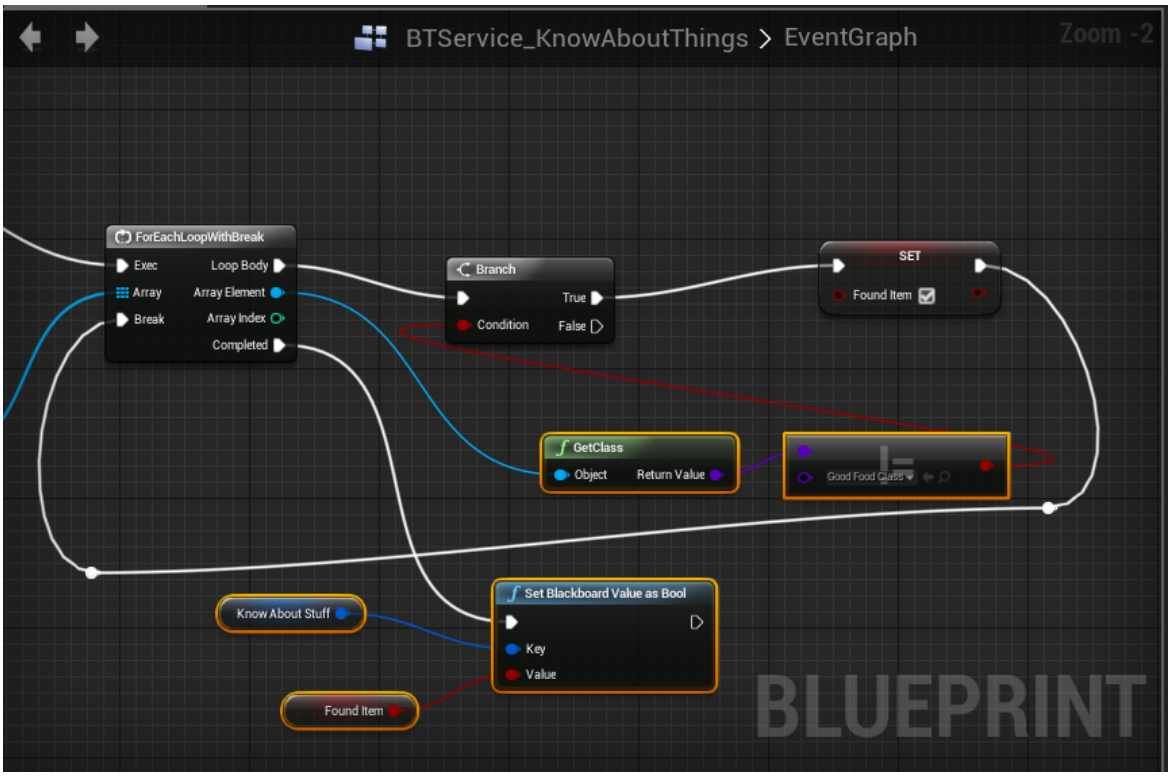
Εικόνα 7.1.4: (2/2)



Εικόνα 7.1.5: Εισαγωγή του *Animation System* μέσω *BP.*

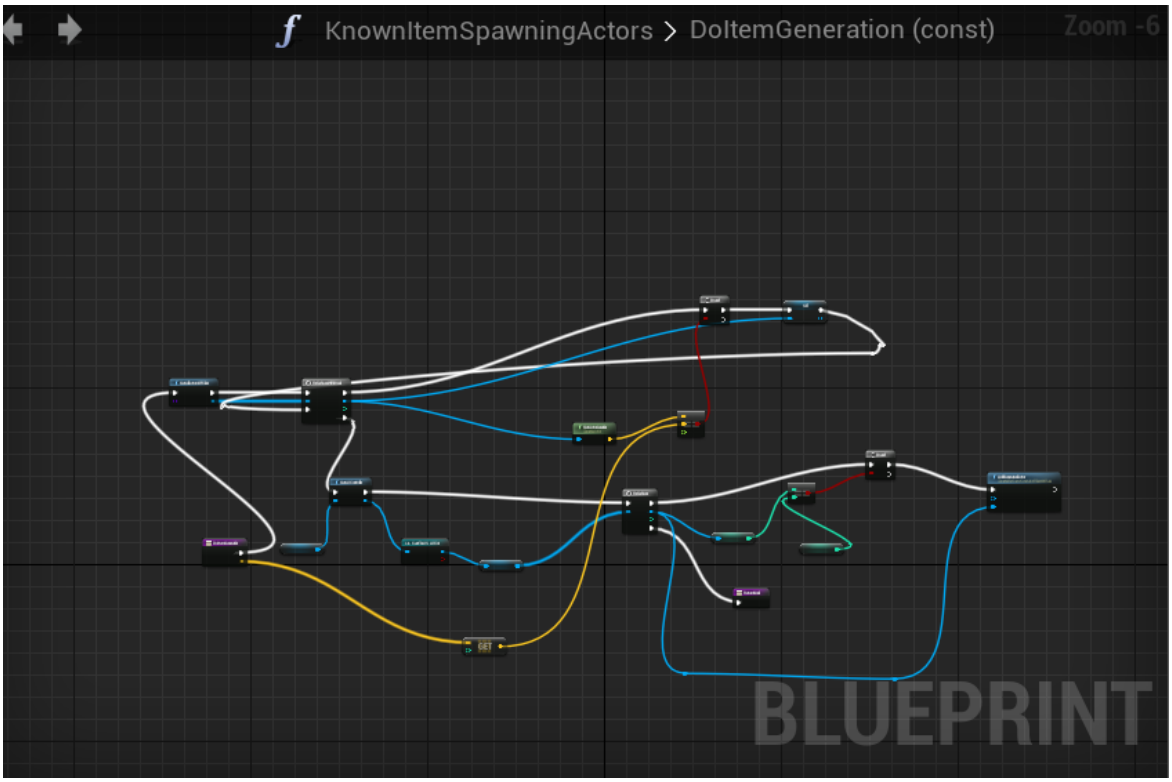


Εικόνα 7.2.1: Αναπαράσταση της Υπηρεσίας *KnowAboutThings.*



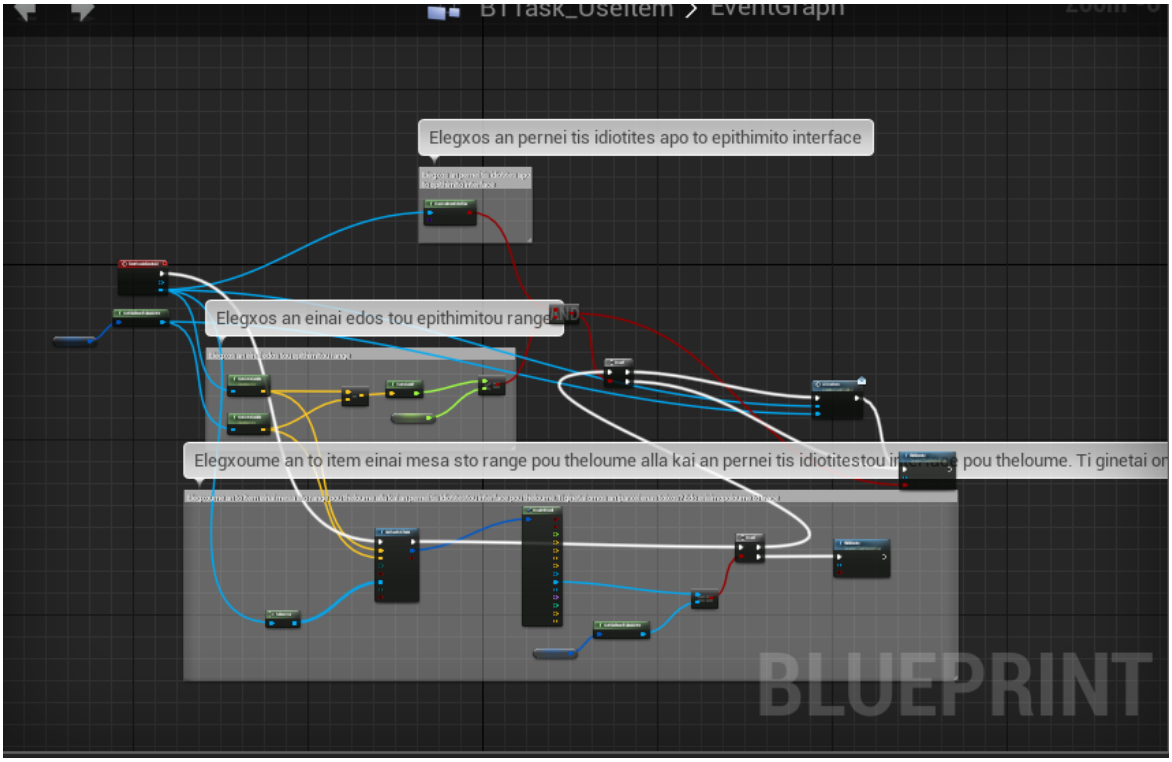
Εικόνα 7.2.2: Αναπαράσταση της Υπηρεσίας *KnowAboutThings.*

Η Υπηρεσία *KnowAboutThings* που παρουσιάζεται στις εικόνες 4.6.1 και 4.6.2 πραγματοποιεί μία σειρά επαναλήψεων εφόσον ήδη έχουμε καλέσει την δεύτερη οντότητα μας με μόνο σκοπό την ενημέρωση της μεταβλητής *Know About Stuff* που αποτελεί κύριο σημείο στην σωστή λειτουργία του *Behavior Tree.* Η ενημέρωση αυτή θα πραγματοποιηθεί μόλις η οντότητα μας αντικρίσει την τροφή του.



Εικόνα 7.3: Αναπαράσταση του *Generator KnownItemSpawningActor.*

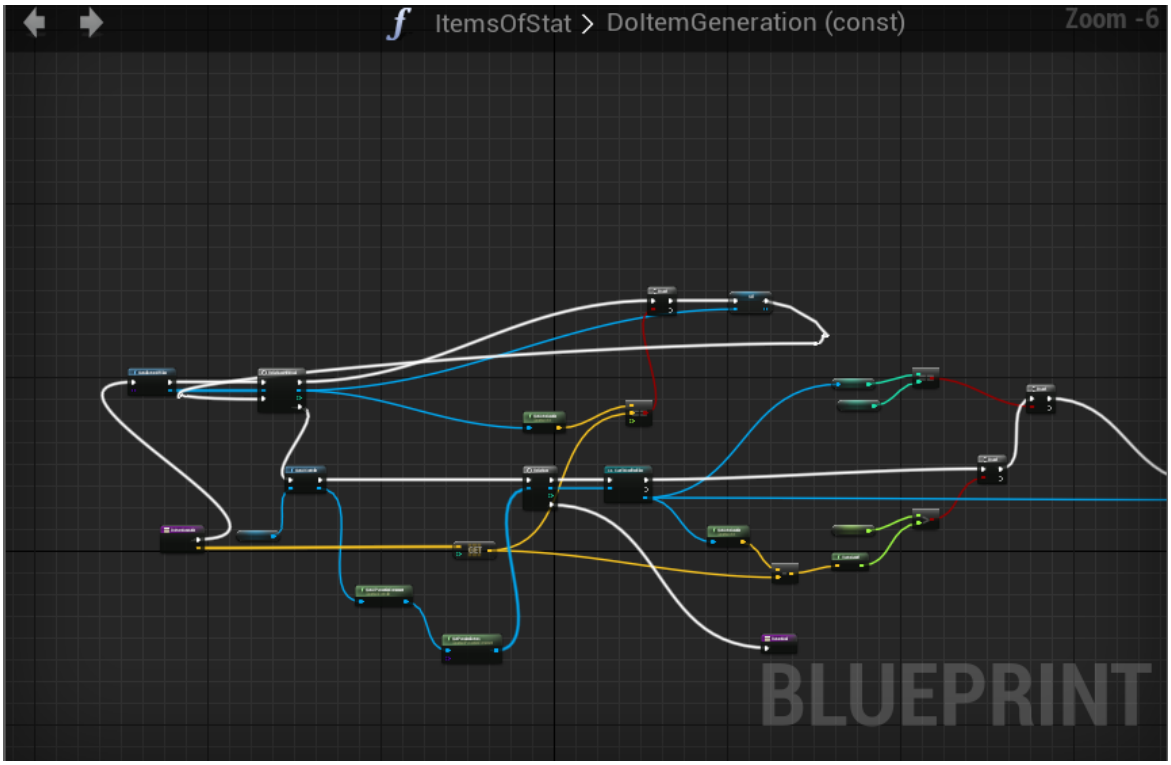
Σε αυτό το *BP* καθορίζουμε τις **θέσεις** των αντικειμένων που αξίζουν στα μάτια της οντότητας μας. Πρόκειται για τον δικό μας *Generator* που χρησιμοποιήθηκε σε συνδυασμό με το *EQS*. Ειδικότερα , θα επιστρέψει μέσα στο *Behavior Tree* το καλύτερο αντικείμενο έτσι ώστε η οντότητα μας να προβεί σε αυτό. Καλύτερο αντικείμενο θεωρούμε αυτό που κατέχει τις καλύτερες προδιαγραφές στο τεστ που πραγματοποιήσαμε μέσα στο *EQS*. Τα αντικείμενα που εξετάζονται εδώ είναι τα δέντρα.



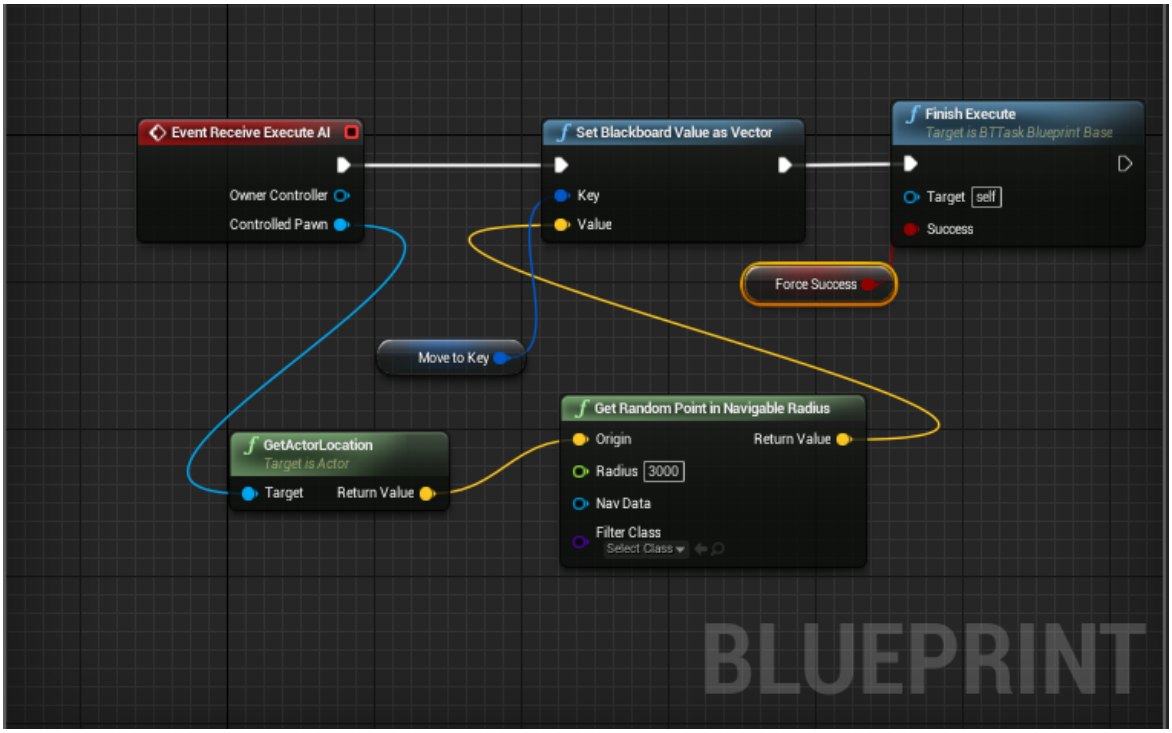
Εικόνα 7.4: Αναπαράσταση του Έργου *Use Item.*

Μερικοί έλεγχοι στην εικόνα 7.4 που αφορούν την αλληλεπίδραση του παίχτη με το μήλο έχουν κάνουν με τα εξής:

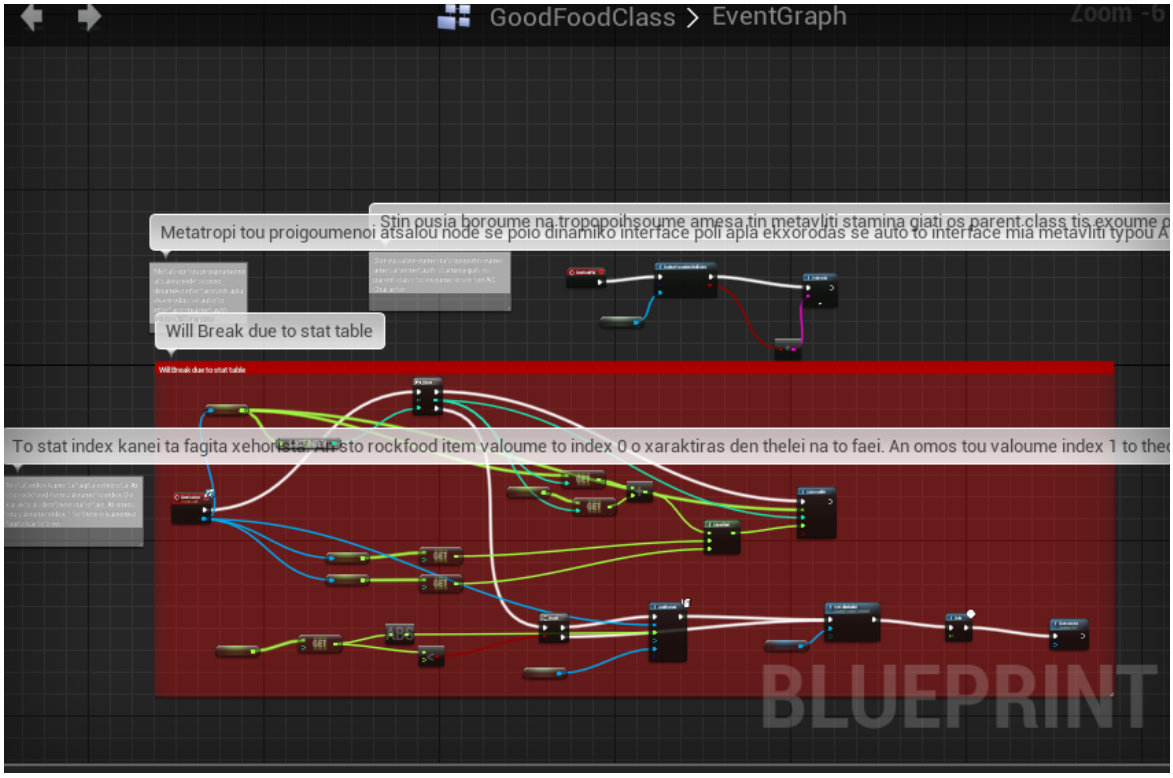
- Αν το μήλο που συναντήσαμε κληρονομεί το απαραίτητο *Interface* έτσι ώστε να προβεί στην κατανάλωση του.
- Αν το μήλο είναι στην επιθυμητή ακτίνα.
- Εφόσον το μήλο μπορεί να καταναλωθεί, ελέγχουμε αν η οντότητα μας μπορεί να φτάσει σε εκείνο το σημείο. Αυτό το επιτυγχάνουμε μέσω του *ray-tracing* που αναλύσαμε στο κεφάλαιο 1.



Εικόνα 7.5: Ο δεύτερος *Custom Generator* των Μήλων.



Εικόνα 7.6: Αναπαράσταση του Έργου της τυχαίας κατεύθυνσης και μετακίνησης της οντότητας μέσω αναφορά στο *navmesh.*



Εικόνα 7.7: Το *Blueprint* της κλάσης του μήλου μας, *Good Food Class*.

ΒΙΒΛΙΟΓΡΑΦΙΑ

(1; 2)

Αναφορά σε Ιστότοπο.

1. **Kehoe Donald.** Designing Artificial Intelligence for Games (Part 1). *Intel Developer Zone.* [Ηλεκτρονικό] Intel, 20 06 2009. [Παραπομπή: 10 05 2016.] https://software.intel.com/en-us/articles/designing-artificial-intelligence-for-games-part-1.

2. **Xu, Siyuan.** My Angel Cafe. [Ηλεκτρονικό] 19 12 2011. https://sites.google.com/site/myangelcafe/articles/history_ai.

(3)

Αναφορά σε Περιοδικό Επιστημονικού Άρθρου.

3. **Woodcock, S.** Game AI: The State of the Industry. *Game Developer Magazine.* 1999, Τόμ. Volume 3, Issue 33.

(4; 5; 6; 7; 8; 9; 10)

Αναφορά σε Βιβλίο.

4. **Tozour, P.** *The Evolution of Game AI.* s.l. : Charles River Media, 2002. σσ. 3-15.

5. **Rabin, S.** *AI Game Programming Wisdom.* s.l. : Charles River Media, 2002. σσ. 49-59. **ISBN-10:** 1584500778

6. **Woodcock, S.** Arcade AI doesn't have to Be Dumb! [συγγρ. βιβλίου] S. Rabin. AI *Game Programming Wisdom Volume 2.* s.l. : Charles River Media, 2004, σσ. 49-59. **ISBN-10:** 1584502894

7. **Rabin, B.** Common Game AI Techniques. [συγγρ. βιβλίου] S. Rabin. *Game Programming Wisdom volume 2.* s.l. : Charles River Media, 2004, σσ. 5-17. **ISBN-10:** 1584502894

8. **Baekkelund, C.** Academic AI Research and Relations with the Game Industry. [συγγρ. βιβλίου] S. Rabin. AI *Game Programming Wisdom.* s.l. : Charles River Media, σσ. 77-88. **ISBN-10:** 1584500778

9. **Matthews, J.** Basic A* Pathfinding Made Simple. Basic A* Pathfinding Made Simple. [συγγρ. βιβλίου] S. Rabin. *AI Gaming Programming Wisdom.* s.l. : Charles River Media, 2003, σσ. 101-113. **ISBN-10:** 1584500778

10. **Higgins, D.** Pathfinding Design Architecture. [συγγρ. βιβλίου] S. Rabin. *AI Game Programming Wisdom.* s.l. : Charles River Media, 2002, σσ. 122-132. **ISBN-10:** 1584500778

(11)

Αναφορά σε Ηλεκτρονική Πηγή.

11. **Wexler, J.**Artifical Intelligence in Games, University of Rochester

http://www.cs.rochester.edu/~brown/242/assts/termprojs/games.pdf. [Ηλεκτρονικό]