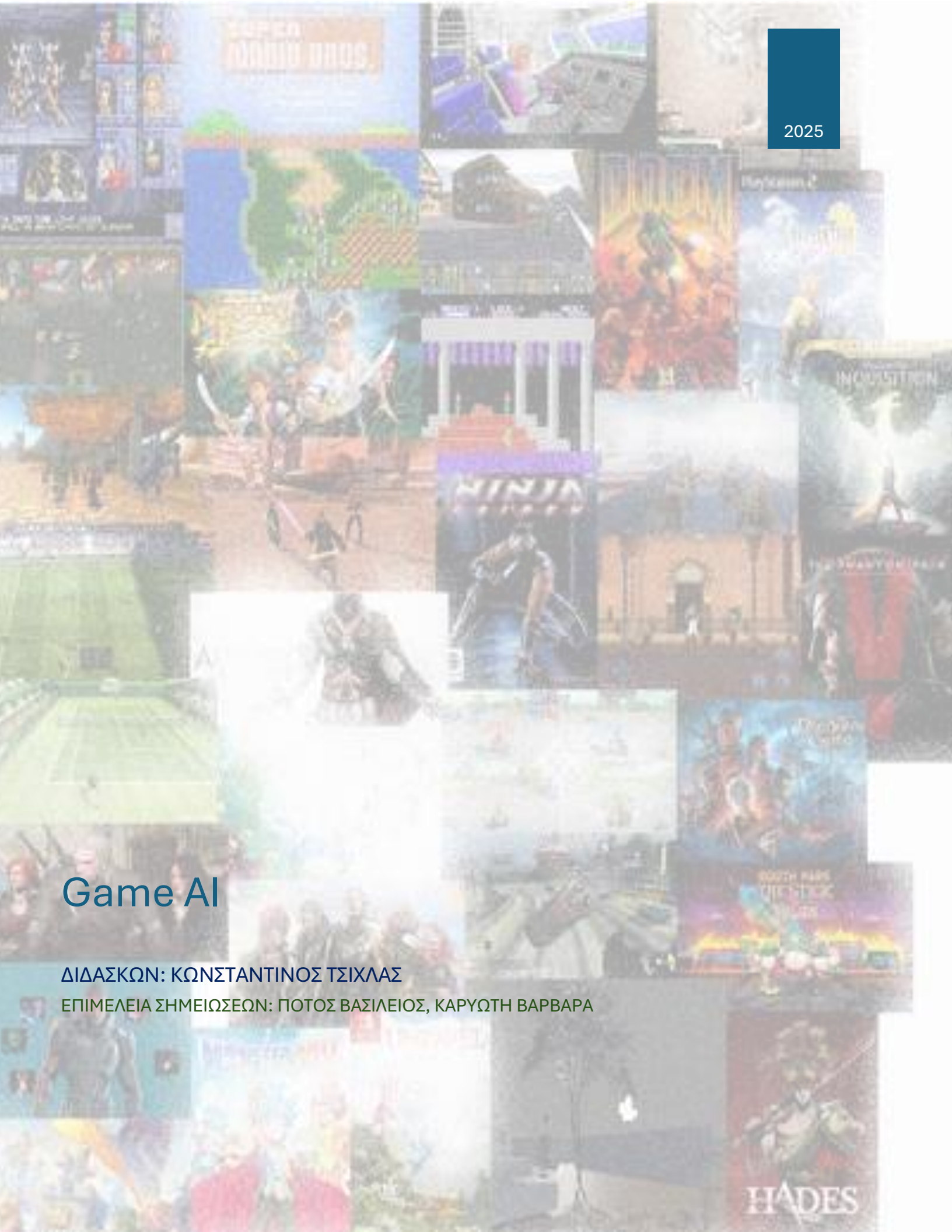


Game AI

ΔΙΔΑΣΚΩΝ: ΚΩΝΣΤΑΝΤΙΝΟΣ ΤΣΙΧΛΑΣ

ΕΠΙΜΕΛΕΙΑ ΣΗΜΕΙΩΣΕΩΝ: ΠΟΤΟΣ ΒΑΣΙΛΕΙΟΣ, ΚΑΡΥΩΤΗ ΒΑΡΒΑΡΑ

Περιεχόμενα

Εισαγωγή στο AI σε βιντεοπαιχνίδια

Εισαγωγή

Έχουμε πει πολλά για τις μηχανές βιντεοπαιχνιδιών και το πως λειτουργούν σε χαμηλό επίπεδο. Στο παρόν κεφάλαιο θα μιλήσουμε για την τεχνητή νοημοσύνη (AI) στα βιντεοπαιχνίδια αναφέροντας μερικές φορές και θέματα που αφορούν τη σχεδίαση του παιχνιδιού. Ο γενικός ορισμός της τεχνητής νοημοσύνης είναι η μορφή νοημοσύνης η οποία εκτίθενται από έναν υπολογιστή. Αποτελεί έναν ή ένα σύνολο αλγορίθμων οι οποίοι χρησιμοποιούν κάποιες εισόδους τις οποίες επεξεργάζονται προκειμένου να παράγουν κάποια έξοδο. Όσον αφορά τα παιχνίδια, το AI συνήθως εφαρμόζεται στους χαρακτήρες που δεν ελέγχονται από τον παίκτη (Non-Player Characters: NPCs), αλλά μπορεί να χρησιμοποιηθεί και σε άλλες λειτουργίες, όπως τη δημιουργία ενός διαδικαστικά παραγόμενου κόσμου ή την εύρεση της βέλτιστης διαδρομής σε έναν χώρο. Να σημειωθεί ότι συνήθως μιλάμε για AI το οποίο αναπτύχθηκε από μία ομάδα ανθρώπων σε μορφή κώδικα και όχι σε μηχανική μάθηση. Η μηχανική μάθηση αν και πολύ ισχυρή και χρήσιμη για τον τομέα μας, είναι υπερβολικά βαριά για να χρησιμοποιηθεί σε εφαρμογές πραγματικού χρόνου, τουλάχιστον για τώρα. Υπάρχει η δυνατότητα να χρησιμοποιηθεί μηχανική μάθηση για την εκπαίδευση ενός AI το οποίο στη συνέχεια εφαρμόζεται στο παιχνίδι, ωστόσο τα αποτελέσματα μπορεί να μην είναι επαρκή ενώ η φύση της μηχανικής μάθησης δεν μας επιτρέπει να ελέγξουμε το πως πραγματικά δουλεύει και να κάνουμε μικρο-αλλαγές στη συμπεριφορά τέτοιων AI.

Γνωρίζουμε ότι το AI έχει αμέτρητες χρήσεις στον πραγματικό κόσμο, τι είναι λοιπόν αυτό που το κάνει να ξεχωρίζει στον τομέα των βιντεοπαιχνιδιών; Ένα καλό AI ενός βιντεοπαιχνιδιού πρέπει να ακολουθεί δύο κανόνες: να φαίνεται έξυπνο και να κάνει το παιχνίδι πιο ευχάριστο. Ας εξετάσουμε τους δύο αυτούς κανόνες.

- **Να φαίνεται έξυπνο**: Με απλά λόγια, αφορά την *ουσία* των πραγμάτων. Τα περισσότερα AI που χρησιμοποιούνται σήμερα εκτός των βιντεοπαιχνιδιών σχεδιάζονται με τέτοιο τρόπο ώστε να σκέφτονται σαν άνθρωποι, να δρουν σαν άνθρωποι, να σκέφτονται ακολουθώντας κάποια λογική, και να δρουν λογικά. Για εμάς όμως δεν μας ενδιαφέρει το AI μας όντως να εκπληρώνει αυτές τις ιδέες, αρκεί να *φαίνεται* σαν να τις εκπληρώνει.

- **Να κάνει το παιχνίδι πιο ευχάριστο**: Αυτό θα πρέπει να είναι προφανές από τις πρώτες διαλέξεις. Κάθε χαρακτηριστικό και περιεχόμενο του παιχνιδιού μας θα πρέπει να τονώσει την ευχαρίστηση και απόλαυση που νιώθει ο παίκτης ενώ παίζει. Φυσικά το AI δεν είναι εξαίρεση σε αυτόν τον κανόνα. Ο παίκτης θα αλληλεπιδράσει με το AI που χτίσαμε όσο με κάθε άλλο κομμάτι του παιχνιδιού, για αυτό πρέπει να βεβαιωθούμε ότι οι αλληλεπιδράσεις αυτές είναι ευχάριστες. Για παράδειγμα, το AI ενός εχθρού θα πρέπει να είναι ικανό να αντιμετωπίσει τον παίκτη παρουσιάζοντας μία ευχάριστη πρόκληση, χωρίς όμως να είναι τόσο απαιτητικό που το στοχευμένο μας κοινό δεν μπορεί να ανταπεξέλθει.

Τρόποι Ανάπτυξης AI Βιντεοπαιχνιδιών

Θα μιλήσουμε σε περισσότερη λεπτομέρεια για το πως σχεδιάζουμε τεχνητή νοημοσύνη για τα παιχνίδια μας παρακάτω. Για την ώρα, ας δούμε πώς αναπτύσσουμε ένα τέτοιο σύστημα. Μία από τις πιο διαδεδομένες τεχνικές ανάπτυξης τεχνητής νοημοσύνης είναι η χρήση μηχανών πεπερασμένων καταστάσεων (finite state machines). Αυτή η τεχνική δηλώνει ότι το AI χωρίζεται σε καταστάσεις, όπου η κάθε κατάσταση αποτελεί κάποια συμπεριφορά. Έτσι, αν φανταστούμε το AI ενός εχθρού ως ένα γράφημα, τότε ο κάθε κόμβος αποτελεί μία κατάσταση ενώ οι ακμές δείχνουν από ποιες καταστάσεις και σε ποιες άλλες μπορούμε να πάμε. Για παράδειγμα, ο εχθρός να ξεκινά σε μία κατάσταση αδράνειας, αλλά όταν εντοπίσει τον παίκτη να αρχίσει να τον καταδιώκει μέχρι να χάσει την οπτική επαφή.

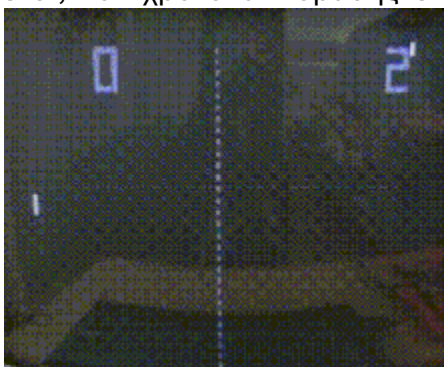
Σημαντικό είναι να αναφέρουμε ότι κάθε AI που έχουμε αναπτύξει συνήθως τρέχει σε κάθε καρέ και πολλές φορές δεν έχουμε μόνο έναν εχθρό αλλά δεκάδες. Για τον λόγο αυτό εφαρμόζουμε πολλές τεχνικές για να ελαχιστοποιήσουμε το βάρος στον επεξεργαστή. Ένα από αυτά τα κόλπα είναι να μειώσουμε τα καρέ στα οποία εκτελείται το AI μας. Για παράδειγμα, όταν ένας εχθρός ο οποίος κυνηγάει τον παίκτη βρίσκεται κοντά του θέλουμε να έχει μία τακτικά ανανεωμένη διαδρομή προς τον παίκτη ώστε να μην κινείται ρομποτικά. Αντιθέτως, όταν ο εχθρός βρίσκεται πιο μακριά μπορούμε να υπολογίζουμε καινούριο μονοπάτι μετά από κάποιο χρονικό διάστημα, ενώ όταν βρίσκεται πολύ μακριά μπορούμε να τον βγάλουμε από την κατάσταση καταδιώξης. Μία άλλη τεχνική, για να πουλήσουμε την ευφυΐα της τεχνητής μας νοημοσύνης δεν έχει καν να κάνει με το ίδιο το AI. Κάποιο ενδιαφέρον animation, σχετικό voice line ή άλλη οπτικό ή ακουστικό ερέθισμα μπορεί να εμφανίσει περισσότερη ευφυία από έναν αλγόριθμο.

Παράδειγμα AI σε 2 κλασικά παιχνίδια

Ας δούμε, στη συνέχεια, πως δουλεύει το AI σε δύο κλασικά παιχνίδια, το Pong και το PacMan.

Αρχικά, το Pong είναι ένα από τα πρώτα βιντεοπαιχνίδια που βγήκε στην αγορά και μπορεί να περιγραφεί ως μία top-down προσομοίωση του τένις. Η πρώτη έκδοση του Pong ήταν σχεδιασμένη για δύο παίκτες, με τον καθένα να χειρίζεται την δική του πλακέτα με το αντίστοιχο χειριστήριο.

Επόμενες εκδόσεις του παιχνιδιού παρουσίασαν την δυνατότητα να παίξει ένας μόνο παίκτης, με την δεύτερη πλατφόρμα να ελέγχεται από τον υπολογιστή. Η πιο απλή υλοποίηση του τεχνητού παίκτη είναι η αντίπαλη πλατφόρμα να ακολουθεί με άριστη ακρίβεια την κάθετη θέση της μπάλας. Ωστόσο αυτό δεν θα ήταν πολύ ευχάριστο για τον παίκτη, καθώς η πλατφόρμα θα κινούνταν με μεγαλύτερη ταχύτητα και ακρίβεια από αυτή ενός κοινού παίκτη. Οι σχεδιαστές έπρεπε να υλοποιήσουν μία τεχνητή νοημοσύνη η οποία μιμείται τις ικανότητες και αδυναμίες ενός πραγματικού παίκτη. Για να το πετύχουν αυτό, πρόσθεσαν δύο απλές αστοχίες στον αντίπαλο. Αρχικά, ο υπολογιστής υπολογίζει τη θέση στην οποία θα φτάσει η σφαίρα όταν βρεθεί στη μεριά του αντιπάλου. Στη συνέχεια, αντί να μετακινείται η πλατφόρμα αμέσως στη θέση αυτή, περιμένει κάποιο χρονικό διάστημα πριν αρχίσει να κινείται, προσομοιώνοντας, έτσι, τον χρόνο αντίδρασης ενός πραγματικού παίκτη. Στη

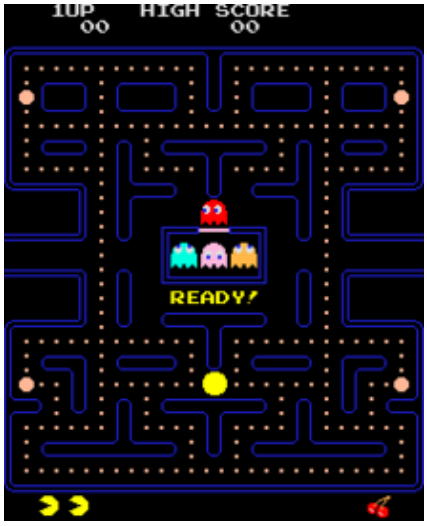


συνέχεια, αντί να κινείται στην ακριβή θέση στην οποία θα βρίσκεται η μπάλα, προσθέτει ένα τυχαίο σφάλμα, προσομοιώνοντας τις ανακρίβειες ενός πραγματικού παίκτη. Το αποτέλεσμα είναι το θεμελιωδώς ίδιο παιχνίδι, αλλά με μόνο ένα πραγματικό παίκτη.

Στη συνέχεια, ας μιλήσουμε για το PacMan. Έχοντας βγει 8 χρόνια μετά την πρώτη έκδοση του Pong, είναι αρκετά πιο πολύπλοκο. Ο παίκτης χειρίζεται τον σφαιρικό κίτρινο χαρακτήρα με μόνο στόχο να φάει όλα τα σφαιρίδια στο χάρτη, εκ των οποίων τα μεγαλύτερα, τα οποία βρίσκονται στις τέσσερις γωνίες του χάρτη, του επιτρέπουν να φάει και τους εχθρούς φαντάσματα. Όταν το πετύχει αυτό, ο χάρτης επαναφέρεται στην αρχική του μορφή και το παιχνίδι συνεχίζεται με όλο και πιο δύσκολους εχθρούς. Όταν ένα φάντασμα αγγίζει τον

παίκτη τότε αυτός χάνει μία ζωή και ο ίδιος μαζί με τα φαντάσματα επαναφέρονται στις αρχικές τους θέσεις, χωρίς όμως το επίπεδο δυσκολίας ή τα σφαιρίδια να αλλάζουν.

Ως παιχνίδι σχεδιασμένο για έναν παίκτη, το AI των φαντασμάτων είναι πιο περίπλοκο από αυτό του Pong προκειμένου να διατηρήσει το ενδιαφέρον των παικτών. Μπορούμε το AI των φαντασμάτων να το χωρίσουμε σε τρεις διακριτές καταστάσεις: κυνήγι, διασπορά, και φόβος. Στην κατάσταση κυνηγιού, ο στόχος των φαντασμάτων είναι να αγγίξουν τον παίκτη με αποτέλεσμα να τον φάνε. Κάθε φάντασμα επιλέγει ένα κελί (ένα μπλοκ 8 επί 8 pixel) σαν στόχο και κινείται προς αυτό. Προκειμένου να κινούνται με πιο ενδιαφέρον τρόπο, το κάθε φάντασμα επιλέγει ένα διαφορετικό στόχο. Ο Blinky, το κόκκινο φάντασμα, είναι το πιο απλό, επιλέγει πάντα το κελί στο οποίο βρίσκεται ο PacMan και κινείται προς αυτό. Ο Inky, το κυανό φάντασμα, λειτουργεί σαν βοηθός του Blinky. Αρχικά, υπολογίζει ένα διάνυσμα από τον Blinky προς δύο κελιά μπροστά από τον PacMan, και μετά διπλασιάζει το μήκος αυτού του διανύσματος. Το καινούριο κελί στο οποίο καταλήγει είναι ο στόχος του. Η ιδέα είναι ο Blinky και ο Inky να δουλέψουν μαζί για να περικυκλώσουν τον παίκτη. Η Pinky, το ροζ φάντασμα, επιχειρεί να ξαφνιάσει τον παίκτη κλείνοντάς του τον δρόμο. Ο στόχος της είναι 4 κελιά μπροστά από τον παίκτη. Τέλος ο Clyde, το πορτοκαλί φάντασμα, είναι το πιο φοβιτσιάρικο. Ο στόχος του είναι ο ίδιος με αυτόν του Blinky, δηλαδή η θέση του παίκτη. Ωστόσο, όταν βρίσκεται σε απόσταση μικρότερη από 8 κελιά από τον παίκτη, τότε ο στόχος του αλλάζει στο “σπίτι” (παρακάτω για τα σπίτια) του στην κάτω αριστερά γωνία του χάρτη. Το αποτέλεσμα είναι ο Clyde να διατηρεί την απόστασή του από τον παίκτη, αλλά δεν προσπαθεί να τον φάει.



Όταν έχει περάσει ένα χρονικό διάστημα, το οποίο αλλάζει με κάθε επίπεδο, τα φαντάσματα αλλάζουν από κατάσταση κυνηγιού στην κατάσταση διασποράς. Σε αυτή την κατάσταση, κάθε φάντασμα αλλάζει τον στόχο του από τον PacMan στο σπίτι του. Το σπίτι του Blinky είναι στην πάνω αριστερά γωνία του χάρτη, του Inky στην κάτω δεξιά, της Pinky στην πάνω αριστερά, και του Clyde στην κάτω αριστερά. Αυτή η κατάσταση υπάρχει προκειμένου να δώσει λίγο χρόνο στον παίκτη να ξεφύγει, αλλιώς το παιχνίδι θα ήταν πολύ άδικο, όπως είναι στο πέμπτο επίπεδο και μετά, φυσικά για να ζητηθεί ο παίκτης να ρίξει δεύτερο κέρμα για να συνεχίσει να παίζει (μην ξεχνάτε ότι μιλάμε για ένα παιχνίδι που αρχικά βγήκε σε arcade).

Τέλος, έχουμε την κατάσταση φόβου. Τα φαντάσματα μπαίνουν σε αυτή την κατάσταση όταν ο PacMan τρώει ένα μεγάλο σφαιρίδιο, αφού πλέον δεν μπορούν να τον φάνε, αλλά μπορεί αυτός. Σε αυτή την κατάσταση τα φαντάσματα δεν έχουν κάποιο στόχο, κινούνται τυχαία.

Όταν ο παίκτης φάει ένα φάντασμα τότε αυτό τρέχει στο κεντρικό του σπίτι, από όπου ξεκινάει στην αρχή του παιχνιδιού, για να αναβιωθεί.

Όλοι αυτοί οι μηχανισμοί οδηγούν σε ένα πολύ ενδιαφέρον και ευχάριστο παιχνίδι, το οποίο αγαπήθηκε από όλο τον κόσμο.

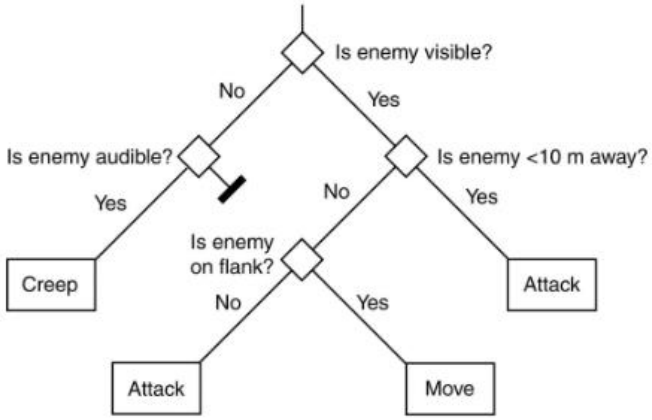
Χρήσιμα εργαλεία για AI βιντεοπαιχνιδιών

Αν ένα παιχνίδι από το 1980 μπορούσε να έχει τόσο πολύπλοκο AI, μπορείτε να φανταστείτε τι γίνεται στα παιχνίδια για κονσόλες που βγήκαν αργότερα. Παιχνίδια όπως το Goldeneye 007 (1997) εκμεταλλεύτηκαν ένα μικρό πλήθος καταστάσεων για το AI των χαρακτήρων. Αυτό που το έκανε να ξεχωρίζει, ωστόσο, είναι ότι το συνδύασαν με ένα σύστημα οπτικής προσομοίωσης. Ο κάθε χαρακτήρας είχε τη δυνατότητα να δει άλλους χαρακτήρες και να παρατηρήσει σε τι κατάσταση βρισκόντουσαν. Αν κάποιος έβλεπε ότι ένας χαρακτήρας είναι νεκρός τότε θα έμπαινε ο ίδιος σε κάποια κατάσταση στην οποία θα κινείται πιο προσεκτικά, ή θα καταδιώκει τον παίκτη. Παιχνίδια όπως τα Thief και Metal Gear Solid βασίζονται σε προσομοιώσεις όρασης, καθώς είναι παιχνίδια stealth όπου παίζει πολύ σημαντικό ρόλο αν οι NPC μπορούν να δουν τον παίκτη. Η υλοποίηση μίας οπτικής προσομοίωσης μπορεί να γίνει με τη χρήση Raycast σε συνδυασμό με σκιές για να ελεγχθεί αν ένα αντικείμενο είναι ορατό σε έναν χαρακτήρα. Για τον ήχο ο χαρακτήρας ελέγχει την απόστασή του από την πηγή του ήχου για να ελέγξει αν τον ακούει.

Στα μέσα της δεκαετίας του 1990 παιχνίδια στρατηγικής πραγματικού χρόνου όπως το Warcraft ήταν εξαιρετικά δημοφιλή. Σημαντικό χαρακτηριστικό αυτών των παιχνιδιών ήταν οι πολύπλοκοι -για την εποχή- χάρτες τους. Οι χάρτες αυτοί απαιτούν τη χρήση κάποιας μεθόδους εύρεσης διαδρομής ώστε οι NPC να μπορούν να περιηγηθούν σε αυτούς. Τέτοια συστήματα συνήθως απλοποιούνται σε αναζήτηση συντομότερης διαδρομής σε ένα γράφημα, όπου υπάρχει τουλάχιστον ένας κόμβος σε κάθε μικρή περιοχή.

Το Halo (2001) ήταν από τα πρώτα βιντεοπαιχνίδια τα οποία εκμεταλλεύτηκαν δέντρα απόφασης για την υλοποίηση της τεχνητής τους νοημοσύνης. Σήμερα, αυτή η τεχνική είναι πολύ συνηθισμένη για την υλοποίηση της συμπεριφοράς των NPC. Κάθε κάποιο χρονικό διάστημα, το οποίο δεν είναι απαραίτητα κάθε καρέ του παιχνιδιού, το AI ακολουθεί ένα δέντρο για να αποφασίσει ποια θα είναι η επόμενή του κίνηση. Ακολουθώντας το παρακάτω παράδειγμα, ο εχθρός ελέγχει αν μπορεί να δει τον παίκτη. Αν ναι, τότε προχωράει στον επόμενο έλεγχο στο δέντρο, ο οποίο ρωτάει αν ο παίκτης βρίσκεται λιγότερο από 10 μέτρα μακριά του. Αν ναι, τότε ο επόμενος κόμβος του λέει να επιτεθεί, αν όχι συνεχίζει στον επόμενο έλεγχο. Υλοποιώντας ένα επεκτάσιμο σύστημα με το οποίο ο σχεδιαστής μπορεί εύκολα να δηλώσει ελέγχους και καταστάσεις, μπορεί

να σχεδιάσει πολύπλοκο και ρεαλιστικό AI το οποίο εντυπωσιάζει τον παίκτη και αυξάνει την ευχαρίστησή του.



Με διαφορετικές τεχνολογίες και υλοποίησεις αναπτύσσουμε όλο και πιο εντυπωσιακά AI τα οποία μπορεί και να ξεπεράσουν τις ικανότητες των πιο αφοσιωμένων παικτών. Χαρακτηριστικό παράδειγμα είναι το AlphaStar, ένα πρόγραμμα σχεδιασμένο για να παίζει StarCraft II, το οποίο κατάφερε να φτάσει στους κορυφαίους 0.15% των παικτών στο σύστημα κατάταξης των ευρωπαικών διακομιστών. Μάλλον το πιο γνωστό παράδειγμα πραγματικών παικτών εναντίων υπολογιστή το βλέπουμε στο σκάκι. Ο πρώτος υπολογιστής σκακιού, ο οποίος κατάφερε να νικήσει άνθρωπο ήταν ο MANIAC I το 1956. Σήμερα, η πιο ισχυρή μηχανή σκακιού είναι το Stockfish με βαθμολογία 3642. Για χάρη σύγκρισης, ο καλύτερος παίκτης σήμερα, ο Magnus Carlsen έχει βαθμολογία μόλις 2831.
Σημερινά παιχνίδια μεγάλων εταιρειών χρησιμοποιούν όλο και πιο ανεπτυγμένα AI για περισσότερο ρεαλισμό και πιο ενδιαφέρον παιχνίδι. Το The Last of Us Part II χρησιμοποιεί προχωρημένο AI για την συμπεριφορά των NPC το οποίο τους επιτρέπει να φέρονται ρεαλιστικά και να αντιδρούν όχι μόνο με τον παίκτη αλλά και με το περιβάλλον. Το Middle-Earth: Shadow of Mordor, χρησιμοποιεί ένα σύστημα ονόματι Nemesis το οποίο επιτρέπει στους χαρακτήρες του παιχνιδιού να θυμούνται τις προηγούμενες αλληλεπιδράσεις τους με τον παίκτη και τους άλλους χαρακτήρες. Με το σύστημα αυτό οι χαρακτήρες σχολιάζουν αυτές τις συναντήσεις δίνοντάς τους περισσότερη προσωπικότητα. Το No Man’s Sky χρησιμοποιεί ένα σύστημα διαδικαστικής παραγωγής για τη δημιουργία του κόσμου του με το οποίο ο κόσμος είναι μεγαλύτερος από κάθε άλλο παιχνίδι.

Λειτουργία του AI των Πρακτόρων στα Βιντεοπαιχνίδια

Όπως είδαμε, υπάρχουν πολλές μορφές τεχνητής νοημοσύνης, αλλά αυτή που μας ενδιαφέρει περισσότερο αφορά τους πράκτορες, δηλαδή τους εχθρούς, τους συμμάχους και άλλους NPC. Στην πιο απλή του μορφή, η συμπεριφορά ενός πράκτορα υπολογίζεται από τον ακόλουθο κύκλο:

Αντίληψη → Σκέψη → Δράση

Στο βήμα της αντίληψης, ο πράκτορας μαζεύει την πληροφορία την οποία χρειάζεται: την απόστασή του από τον παίκτη, ένα μονοπάτι προς τον στόχο του, αν μπορεί να δει κάποιον άλλο χαρακτήρα ή αντικείμενο, τι εργαλεία κρατάει, ποια είναι η εικονική ώρα και τι πρέπει να κάνει αυτή την ώρα σύμφωνα με το πρόγραμμά του. Ανάλογα με την πολυπλοκότητα του AI που χρειαζόμαστε, ο πράκτορας μπορεί να μαζέψει περισσότερη ή λιγότερη πληροφορία. Σε ένα παιχνίδι όπως τα Hitman, ο πράκτορας του κάθε χαρακτήρα χρειάζεται να ξέρει πολλά πράγματα, όπως τι είναι επόμενο στο πρόγραμμά του, που πρέπει να βρίσκεται, αν βλέπει κάτι ύποπτο, και άλλα. Αντιθέτως, στο Pong, το μόνο που χρειάζεται είναι η τωρινή του θέση, και μία πρόβλεψη για το που πρόκειται να φτάσει η μπάλα.

Στο βήμα της σκέψης, ο πράκτορας χρησιμοποιεί τα δεδομένα που μάζεψε στο προηγούμενο βήμα και τα επεξεργάζεται για να φτάσει σε κάποια δράση. Αν θυμηθούμε το δέντρο απόφασης του Halo, μπορούμε να φανταστούμε το βήμα σκέψης σαν όλους τους κόμβους ενός δέντρου απόφασης οι οποίοι δεν είναι φύλλα. Κάθε ερώτηση και έλεγχος που χρειάζεται να κάνει ο πράκτορας για να φτάσει σε κάποιο φύλλο αποτελούν μέρος το βήματος σκέψης. Είναι πολύ σημαντικό να προσέχουμε αν χρησιμοποιούμε κάποιο αναδρομικό σύστημα, να σιγουρευτούμε ότι δεν υπάρχει η πιθανότητα να βρεθεί ο πράκτορας σε έναν άπειρο βρόχο, γιατί τότε οι έλεγχοί του δεν θα τελειώσουν ποτέ και το παιχνίδι θα κρασάρει. Όταν ολοκληρωθούν οι υπολογισμοί που χρειάζονται τότε παράγεται η ανάλογη έξοδος και ο πράκτορας προχωράει στο βήμα δράσης.

Τέλος, στο βήμα δράσης ο πράκτορας εκτελεί την δράση στην οποία κατέληξε χρησιμοποιώντας τα δεδομένα τα οποία παρήγαγε το προηγούμενο βήμα. Για παράδειγμα, στο βήμα σκέψης ο πράκτορας αποφασίζει ότι θέλει να κινηθεί προς τον παίκτη με κάποια ταχύτητα, ωστόσο το βήμα δράσης είναι υπεύθυνο για να πραγματοποιήσει αυτή την κίνηση, ενώ παίζει το κατάλληλο animation.

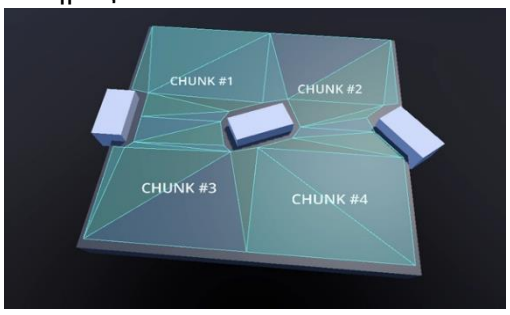
Κοινές τεχνικές AI σε βιντεοπαιχνίδια

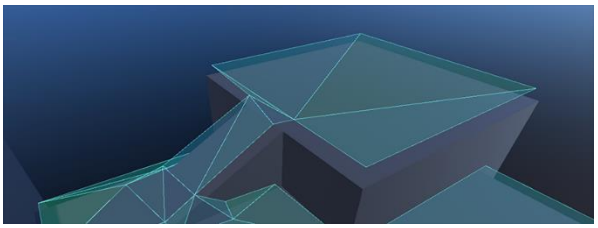
Ας δούμε κάποιες τεχνικές που συναντάμε συχνά στα σύγχρονα παιχνίδια. Ένα πολύ συνηθισμένο πρόβλημα είναι αυτό της κίνησης πολλών χαρακτήρων. Όταν έχουμε λίγους NPC είναι αρκετά εύκολο να υπολογίσουμε την διαδρομή την οποία θα πρέπει να ακολουθήσουν για να φτάσουν στον στόχο τους. Το πρόβλημα εμφανίζεται όταν έχουμε πολλούς χαρακτήρες με τον ίδιο στόχο, γιατί σε αυτή την περίπτωση θα χρησιμοποιήσουν όλοι την ίδια διαδρομή με αποτέλεσμα να δημιουργείται ένας συνωστισμός. Η λύση σε αυτό το πρόβλημα ονομάζεται συσσώρευση (flocking). Δεν έχουμε τον χρόνο να αναλύσουμε την κάθε λεπτομέρεια αυτής τεχνικής αλλά η βασική ιδέα είναι ότι κάθε οντότητα ακολουθεί τρεις κανόνες προκειμένου η συνολική ομάδα οντοτήτων να κινείται με κάπως φυσικό τρόπο. Οι κανόνες είναι οι ακόλουθοι:

- Διαχωρισμός: η κάθε οντότητα αλλάζει την κατεύθυνσή της προκειμένου να αποφευχθεί ο συνωστισμός.
- Ευθυγράμμιση: η κάθε οντότητα αλλάζει την κατεύθυνσή της προς τη μέση κατεύθυνση της ομάδας.
- Συνοχή: η κάθε οντότητα αλλάζει την κατεύθυνσή προς τη μέση θέση της ομάδας.

Μία άλλη τεχνική η οποία επιτυγχάνει παρόμοια αποτελέσματα είναι οι σχηματισμοί. Αυτή η τεχνική λειτουργεί παρόμοια με την συσσώρευση, με τη διαφορά ότι οι οντότητες κρατάνε τις σχετικές θέσεις μεταξύ τους ώστε να σχηματίζουν μία διάταξη σαν έναν στρατιωτικό σχηματισμό.

Όσον αφορά την εύρεση της διαδρομής ο πιο συνήθης αλγόριθμος που χρησιμοποιούμε είναι ο A*. Είναι ένας αρκετά έξυπνος αλγόριθμος ο οποίος χρησιμοποιεί τη γνώση του προορισμού ώστε να επιστρέψει την συντομότερη διαδρομή σε καλύτερο χρόνο συγκριτικά με άλλους αλγορίθμους. Να σημειωθεί ότι ο A* είναι αρκετά καλός για στατικό περιβάλλον αλλά όχι για δυναμικό με κινούμενα σημεία σύγκρουσης. Χρησιμοποιείται συνήθως εντός ενός στατικού πλέγματος πλοήγησης το οποίο ορίζεται από τους σχεδιαστές, ώστε να βρίσκει την κατάλληλη διαδρομή γύρω από στατικά αντικείμενα (εμπόδια) με τα οποία συγκρούεται ο χαρακτήρας.





Ένα άλλο ζήτημα είναι το πως οργανώνουμε τους κανόνες συμπεριφοράς. Ανάλογα με το τι θέλουμε να υλοποιήσουμε μπορούμε να εφαρμόσουμε διαφορετικές τεχνικές οργάνωσης συμπεριφοράς για την τεχνητή μας νοημοσύνη.

Για παράδειγμα, μπορούμε να χρησιμοποιήσουμε αναδυόμενη συμπεριφορά σύμφωνα με την οποία εφαρμόζονται απλοί κανόνες οι οποίοι οδηγούν σε πολύπλοκες αλληλεπιδράσεις. Κλασικό παράδειγμα είναι το παιχνίδι της ζωής του John Horton Conway, σύμφωνα με το οποίο έχουμε ένα πλέγμα πλακιδίων όπου κάθε πλακίδιο μπορεί στον επόμενο κύκλο να είναι σε κατάσταση ζωής ή θανάτου ανάλογα με το σε τι κατάσταση βρίσκονται τα πλακίδια γύρω του. Τέτοια συστήματα είναι χρήσιμα για απλές προσομοιώσεις αλλά δεν χρησιμεύουν πολύ για σύγχρονα παιχνίδια.

Ένα σύστημα το οποίο χρησιμοποιείται στα σύγχρονα παιχνίδια είναι η ιεραρχία εντολών σύμφωνα με την οποία το AI παίρνει αποφάσεις σε διαφορετικά επίπεδα. Για παράδειγμα, σε ένα παιχνίδι στρατηγικής μπορούμε να χωρίσουμε το AI μας σε επίπεδο συνολικής στρατηγικής, επίπεδο τακτικού λόχου, και επίπεδο τακτικής στρατιώτη.

Αν έχουμε ένα παιχνίδι με πολλές οντότητες τότε μπορεί να έχουν κακή απόδοση αν λειτουργούν μεμονωμένα. Σε τέτοιες περιπτώσεις, υλοποιούμε έναν διαχειριστή, ο οποίος φτιάχνει εργασίες, δίνει προτεραιότητες και τις αναθέτει σε ομάδες. Για παράδειγμα, σε ένα παιχνίδι ποδοσφαίρου ο διαχειριστής θα ελέγξει ποιος από τους NPC θα πρέπει να κινηθεί προς τον παίκτη για να του κλέψει την μπάλα.

Τέλος, πρέπει να αναφέρουμε και εδώ μία γνωστή τεχνική, την τεχνική Level of Detail (LoD) αλλά για τεχνητή νοημοσύνη. Όσο πιο μακριά είναι μία οντότητα, τόσο λιγότερο θα φαίνονται οι ατέλειες της, και άρα τόσο λιγότερους πόρους χρειαζόμαστε να αφιερώσουμε σε αυτή σε σχέση με το AI της.

Υποσχόμενες AI Τεχνικές

Προφανώς αυτές δεν είναι οι μόνες τεχνικές που μπορούμε να χρησιμοποιήσουμε για την υλοποίηση της τεχνητής νοημοσύνης μας στα παιχνίδια. Υπάρχουν πολλές άλλες τεχνικές οι οποίες δεν είναι ευρέως χρησιμοποιούμενες ή δεν έχουν χρησιμοποιηθεί ακόμα λόγω κόστους. Υπάρχουν διάφοροι λόγοι για τους οποίους μπορεί να αποφεύγουμε αυτές τις τεχνικές. Τα σύγχρονα βιντεοπαιχνίδια παίρνουν πολύ χρόνο για να αναπτυχθούν και έχουν μεγάλο κόστος. Οπότε, όταν έχουμε να διαλέξουμε ανάμεσα σε δύο τεχνικές με την δεύτερη να παρέχει ελαφρώς καλύτερα αποτελέσματα αλλά να είναι πολύ πιο χρονοβόρα και απαιτητική στην υλοποίησή της, τότε μάλλον θα προτιμήσουμε την ευκολότερη. Άλλες τεχνικές μπορεί να αποφεύγονται επειδή έχουν πολύ περιορισμένη χρήση ή είναι

δυσκολονόητες σε σημείο που καθυστερεί η επικοινωνία ανάμεσα στους υλοποιητές οι οποίοι δουλεύουν στους ίδιους μηχανισμούς. Ανεξαρτήτως, όλα αυτά δεν σημαίνουν ότι οι τέτοιος τεχνικές πρέπει να αποφεύγονται ή ότι δεν έχουν χρησιμοποιηθεί ποτέ. Σίγουρα υπάρχουν ειδικές περιπτώσεις στις οποίες είναι χρήσιμες.

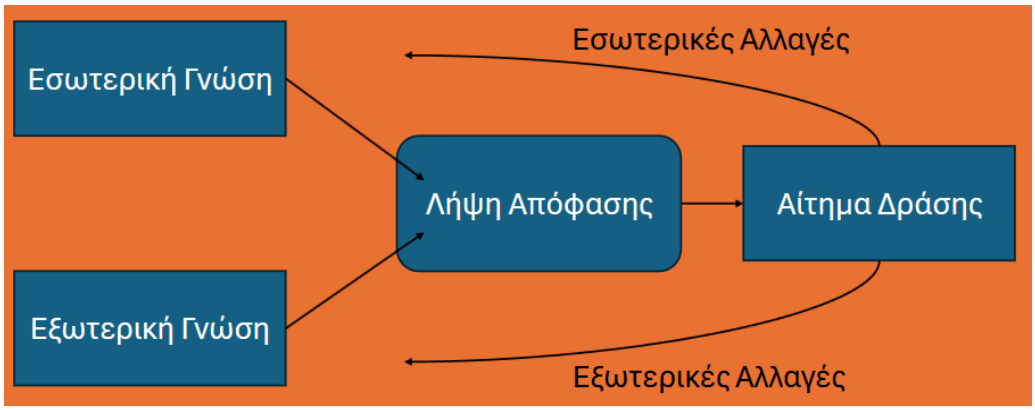
Μία από αυτές τις τεχνικές είναι τα δίκτυα Bayesian, με τα οποία μπορούμε να μοντελοποιήσουμε τις γνώσεις του παίκτη από το AI. Ένα τέτοιο σύστημα θα μπορούσε να χρησιμοποιηθεί για το AI του αντιπάλου σε ένα παιχνίδι στρατηγικής όπου ο αντίπαλος καλείται να δρα σαν πραγματικό άνθρωπο. Ένα άλλο πεδίο τεχνικής νοημοσύνης για το οποίο ακούγονται πολλά τελευταία είναι οι γενετικοί αλγόριθμοι. Αν και ισχυροί, παραείναι ακριβοί για να χρησιμοποιηθούν στα σημερινά παιχνίδια, ενώ ο χρόνος εκπαίδευσης σε συνδυασμό με την αδυναμία να κατανοήσουμε πως πραγματικά λειτουργεί ο αλγόριθμος, με αποτέλεσμα να είναι δύσκολες οι τροποποιήσεις του, τους καθιστά κατώτερους σε σύγκριση με πιο παραδοσιακές τεχνικές τεχνητής νοημοσύνης.

Μοντέλο AI

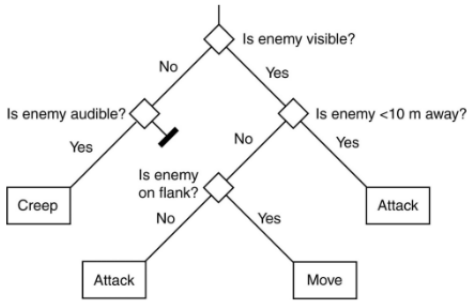
Θυμίζουμε τη βασική δομή μίας τεχνητής νοημοσύνης για βιντεοπαιχνίδια. Πρώτα η σχετική οντότητα συλλέγει όλη τη πληροφορία που την ενδιαφέρει και εκτελεί κάποιες αποφάσεις πάνω σε αυτή. Για ένα παιχνίδι δράσης αυτή η πληροφορία μπορεί να είναι η θέση του παίκτη και της οντότητας, καθώς και την κατάσταση στην οποία βρίσκονται ο καθένας από αυτούς. Στη συνέχεια επεξεργάζεται τα δεδομένα που σύλλεξε για να φτιάξει ένα σχέδιο πράξεων. Συνεχίζοντας από το προηγούμενο παράδειγμα, αν η οντότητα γνωρίζει που είναι ο παίκτης μπορεί να επιχειρήσει να του επιτεθεί, ενώ αν όχι τότε μπορεί να αρχίσει να τον ψάχνει. Σε κάθε περίπτωση η οντότητα θα πρέπει να υπολογίσει μία διαδρομή προς τον στόχο της (π.χ., τον ίδιο τον παίκτη ή το τελευταίο σημείο στο οποίο τον είδε). Τέλος, η οντότητα δρα πάνω στο σχέδιο που έφτιαξε στο προηγούμενο βήμα. Όταν δηλαδή υπολογίσει την διαδρομή την οποία πρέπει να ακολουθήσει, αρχίζει να κινείται πάνω σε αυτή εφαρμόζοντας κανόνες φυσικής, σύγκρουσης και άλλα.

Πηγαίνοντας ένα βήμα πιο βαθιά, μπορούμε να επεκτείνουμε το μοντέλο που έχουμε σχηματίσει αναλύοντας το βήμα αντίληψης. Αν φανταστούμε τη σκηνή του παιχνιδιού μας σαν ένα δέντρο, τότε η οντότητα την οποία αναλύουμε είναι ένας από τους κόμβους αυτού του δέντρου. Έτσι χωρίζουμε τις γνώσεις της οντότητας σε εσωτερικές και εξωτερικές. Ό,τι σχετίζεται με την ίδια οντότητα και τα παιδιά της θα το θεωρούμε εσωτερική γνώση, ενώ οτιδήποτε άλλο θα το λέμε εξωτερική γνώση. Η εσωτερική γνώση της οντότητας μας μπορεί

να είναι η ζωή της, η κατάστασή της, ή το όπλο που χρησιμοποιεί, ενώ το ύψος του χώρου, η θέση του παίκτη, και οι καταστάσεις άλλων οντοτήτων αποτελούν εξωτερική γνώση.



Αναφέραμε προηγουμένως ένα πολύ σημαντικό σύστημα για την υλοποίηση του βήματος σκέψης, τα δέντρα απόφασης. Σε επίπεδο κώδικα, τα δέντρα απόφασης απλοποιούνται σε καταστάσεις οι οποίες περιγράφονται με τη χρήση if-then-else ή case. Αποτελούν ένα σύστημα κανόνων όπου συγκεκριμένες δράσεις εκτελούνται σε συγκεκριμένες συνθήκες. Είναι μία απλή αλλά ισχυρή τεχνική η οποία υλοποιείται και κατανοείται εύκολα και γρήγορα. Χάρη στην ευελιξία του συστήματος μπορεί εύκολα να χρησιμοποιηθεί με άλλα συστήματα ή τεχνικές. Θυμίζουμε το παράδειγμα από πριν. Προφανώς το συγκεκριμένο δέντρο είναι δυαδικό, οπότε εκμεταλλεύεται μόνο if-else statements, αλλά σε ένα πραγματικό παιχνίδι τίποτα δεν μας εμποδίζει από το να χρησιμοποιήσουμε όλα τα εργαλεία ελέγχου συνθηκών που ξέρουμε. Για παράδειγμα, αντί ο χαρακτήρας να έχει μόνο έναν έλεγχο για την απόστασή του από τον παίκτη (is enemy <10m away) μπορούμε να χρησιμοποιήσουμε if-elseif-else για να εξετάσουμε πολλές διαφορετικές αποστάσεις. Ή μπορούμε να εκμεταλλευτούμε cases για να ελέγξουμε αν ο παίκτης βρίσκεται σε κάποια συγκεκριμένη κατάσταση. Το κύριο πρόβλημα αυτής της υλοποίησης είναι η hard-coded φύση του, που σημαίνει ότι αν πρέπει να κάνουμε κάποιες αλλαγές στο AI θα πρέπει να γίνουν στον κώδικα, το οποίο μπορεί να μην είναι θέμα για τον προγραμματιστή ο οποίος έφτιαξε το σύστημα, αλλά αν κάποιος σχεδιαστής θέλει να κάνει μία αλλαγή, ή αν κάποιος άλλος προγραμματιστής του αναθετηθεί να ενημερώσει τον κώδικα τότε θα πρέπει να ξοδέψουν χρόνο να μάθουν πως πραγματικά λειτουργεί αυτός ο ιστός συνθηκών.



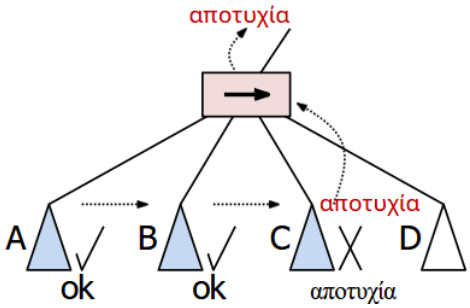

```
if(enemy.isVisible()) {
  if(distance(player, enemy) < 10){
    attack();
  }
  else{
    if(enemy.isOnFlank()){
      move();
    }
    else{
      attack();
    }
  }
}
else{ … … …
```

Έχουμε λοιπόν ένα ευνόητο και αποδοτικό σύστημα με το οποίο μπορούμε να υλοποιήσουμε όποια συμπεριφορά χρειαζόμαστε αλλά έχει όμως ένα σχεδιαστικό μειονέκτημα: είναι προβλέψιμο. Όσο πολύπλοκο και όσες καταστάσεις και αν βάλουμε στο δέντρο μας, στις ίδιες συνθήκες θα καταλήγει πάντα στο ίδιο αποτέλεσμα. Τέτοια AI μπορεί εύκολα να χειραγωγηθούν από τον παίκτη και πολύ γρήγορα οι αλληλεπιδράσεις με αυτά θα καταλήξουν βαρετές. Μπορούμε να μετριάσουμε αυτό το πρόβλημα εφαρμόζοντας ένα βαθμό τυχαιότητας στο AI μας. Όταν ένας χαρακτήρας κινείται ελεύθερα στον χώρο μπορεί να διαλέγει που να στρίψει τυχαία, ενώ όταν ένας εχθρός πολεμάει τον παίκτη μπορεί να χρησιμοποιεί μία τυχαία επίθεση όταν έχει περάσει ένα τυχαίο χρονικό διάστημα. Να θυμίσουμε ότι το δέντρο θα ελέγχεται πολλές φορές το δευτερόλεπτο και αν σε κάθε βρόχο ελέγχου ο πράκτορας παίρνει μία καινούρια τυχαία απόφαση τότε δεν θα λειτουργεί σωστά. Για αυτό θα πρέπει να χρησιμοποιήσουμε κάποιο σύστημα το οποίο θα επιμένει στην απόφαση ή δράση στην οποία βρίσκεται ο πράκτορας. Για παράδειγμα αν ο εχθρός βρίσκεται ήδη στο animation επίθεσης δεν θέλουμε να το ξαναρχίσει σε κάθε βρόχο. Θέλουμε να περιμένουμε μέχρι να τελειώσει το animation και μετά να ξαναξεκινήσει ο βρόχος απόφασης.

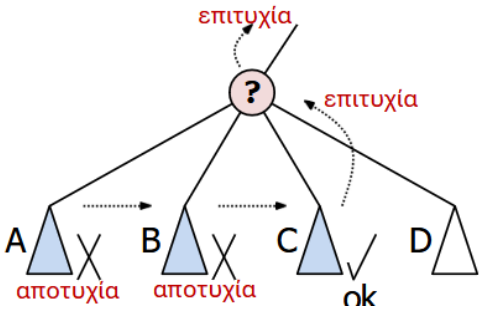
Δένδρα Συμπεριφοράς

Μία άλλη μέθοδος δίνεται από τα δένδρα συμπεριφοράς (behavior trees) τα οποία μπορούν να υλοποιήσουν πλάνα δραστηριοτήτων για τον NPC. Ένα τέτοιο δένδρο που περιγράφει συμπεριφορές θα μπορούσατε ως σχεδιαστές να το επικοινωνήσετε στους υλοποιητές. Προφανώς θυμίζουμε ότι διαφορετικές εταιρείες έχουν διαφορετικές ροές εργασίας, οπότε το παρακάτω σχέδιο είναι ενδεικτικό:

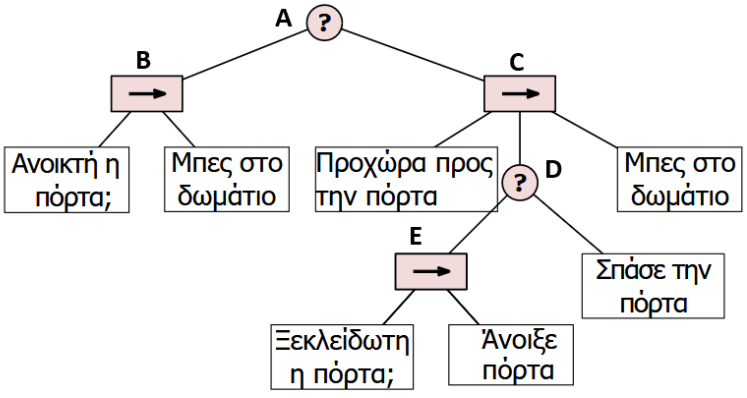
Ο σειριακός κόμβος AND ξεκινάει από το αριστερότερο παιδί και συνεχίζει στα πιο δεξιά. Αν το A επιστρέψει επιτυχία τότε ο έλεγχος προχωράει στο B. Αν το B πετύχει τότε προχωράει στο C. Αν το C αποτύχει τότε ο έλεγχος επιστρέφει στον γονέα του κόμβου για να ελεγχθεί ο επόμενος αδελφός.



Ο σειριακός κόμβος OR από την άλλη πλευρά ελέγχει όλα τα παιδιά μέχρι κάποιο από αυτά να επιστρέψει επιτυχία.



Ας δούμε πως αυτό το σύστημα μπορεί να χρησιμοποιηθεί για την υλοποίηση ενός εχθρού ο οποίος προσπαθεί να μπει σε ένα δωμάτιο. Ο πράκτορας ξεκινάει από τον κόμβο A και ελέγχει το αριστερότερο παιδί πρώτα. Το αριστερότερο παιδί είναι ένας κόμβος AND. Αρχικά ελέγχει



αν η πόρτα είναι ανοιχτή, αν ναι τότε θα επιχειρίσει να μπει στο δωμάτιο. Αν τα καταφέρει τότε ο έλεγχος τελειώνει εδώ. Αν οποιοδήποτε από αυτά τα βήματα αποτύχει τότε ενημερώνει τον κόμβο A ότι απέτυχε και προχωράει στον κόμβο C. Ο κόμβος C είναι άλλος ένας κόμβος AND. Αρχικά ο πράκτορας θα προσπαθήσει να κινηθεί προς την πόρτα. Αν τα καταφέρει τότε συνεχίζει στον κόμβο D. Όταν είναι αρκετά κοντά στην πόρτα θα ελέγξει αν είναι ξεκλείδωτη και αν ναι τότε θα προσπαθήσει να την ανοίξει, αλλιώς θα προσπαθήσει να την σπάσει. Αν οποιαδήποτε από αυτές τις συνθήκες πετύχει τότε ο πράκτορας θα προσπαθήσει να μπει στο δωμάτιο. Αν σε οποιοδήποτε από τα παιδιά του C επιστρέψει αποτυχία τότε τόσο ο B όσο και ο C επέστρεψαν επίσης αποτυχία, οπότε ο κόμβος A θα επιστρέψει και αυτός αποτυχία και θα συνεχίσει το υπόλοιπο δέντρο (δεν φαίνεται). Σε επίπεδο κώδικα προφανώς υπάρχουν πολλές λεπτομέρειες που μένουν έξω, αλλά αυτό το σχήμα είναι επαρκές για να εξηγήσει ο σχεδιαστής αυτό που περιμένει να αναπτύξουν οι υλοποιητές.

Μηχανές Πεπερασμένων Καταστάσεων

Μία εξίσου ισχυρή τεχνική είναι οι μηχανές πεπερασμένων καταστάσεων. Οργανώνουμε τις διάφορες συμπεριφορές ή καταστάσεις του πράκτορα ως κόμβους σε ένα γράφημα. Μεταβαίνουμε από την μία κατάσταση στην άλλη ανάλογα με κάποιες συνθήκες. Οι καταστάσεις μπορεί να είναι πολύ απλές και να περιέχουν μόνο ένα animation, ή μπορεί να έχουν το δικό τους δέντρο απόφασης για πιο πολύπλοκα AI. Με σωστή υλοποίηση μπορούμε να εκφράσουμε τις καταστάσεις και τις μεταβάσεις σε ένα γράφημα το οποίο ο σχεδιαστής μπορεί εύκολα να παραποιήσει σε κάποιο editor. Καθώς η λογική της κάθε κατάστασης είναι υλοποιημένη μέσα σε αυτή σημαίνει ότι μπορούμε να εντοπίσουμε και να επιδιορθώσουμε σφάλματα πιο εύκολα καθώς θα χρειάζεται να εξετάσουμε μόνο το script αυτής της κατάστασης και όχι ένα γενικό script το οποίο περιέχει όλο το AI. Το κύριο πρόβλημα αυτής της τεχνικής είναι ότι για πολύπλοκα AI θα χρειαστούμε ένα μεγάλο και πιθανόν δυσανάγνωστο γράφημα με πολλές καταστάσεις. Βεβαίως, υπάρχουν τεχνικές που προσπαθούν να μειώσουν το πρόβλημα, όπως οι ιεραρχικές μηχανές πεπερασμένων καταστάσεων, που εκφράζουν μεταβάσεις μεταξύ ομάδων καταστάσεων.

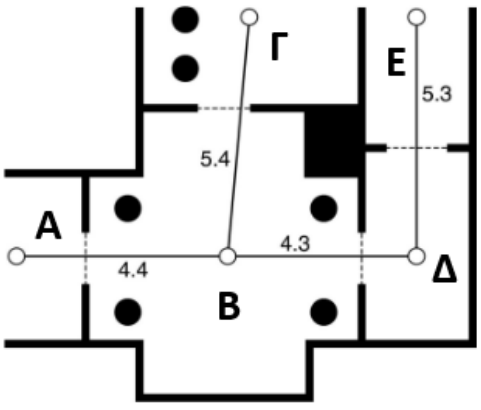
Εύρεση Διαδρομής

Όπως έχουμε ήδη αναφέρει, ένα πολύ σύνηθες πρόβλημα στο AI για παιχνίδια είναι η εύρεση διαδρομής. Είναι πολύ σπάνιο σήμερα να βρούμε παιχνίδι το οποίο δεν χρειάζεται κάποιου είδους πλοήγηση για τους χαρακτήρες του. Και ακόμα και αν βρούμε λύση για μία οντότητα, αυτό το σύστημα δεν είναι ικανό όταν έχουμε δεκάδες ή και εκατοντάδες οντότητες με διαφορετικούς στόχους.

Ένας τρόπος να αντιμετωπίσουμε αυτό το πρόβλημα είναι η αναπαράσταση ενός χάρτη με τη μορφή γραφημάτων. Η μέθοδος αυτή λειτουργεί καλά σε άκαμπτους χώρους. Για την εύρεση διαδρομής πάνω σε αυτή την αναπαράσταση μπορούμε να χρησιμοποιήσουμε τον αλγόριθμο του Dijkstra. Φυσικά, στον τομέα των βιντεοπαιχνιδιών, η βέλτιστη διαδρομή δεν είναι απαραίτητα αυτή με την μικρότερη απόσταση. Ένα παιχνίδι αποτελεί μία μορφή προσομοίωσης και για αυτό θα πρέπει να τροποποιήσουμε τον αλγόριθμο ώστε να λάβουμε υπόψιν άλλες μεταβλητές, όπως τη ταχύτητα της οντότητας, το κόστος ενέργειας ή κάποιου πόρου, τον κίνδυνο της διαδρομής, και ότι άλλο είναι σχετικό στο παιχνίδι μας. Ας υποθέσουμε ότι έχουμε έναν κόσμο ο οποίος αναπαριστάται από εξάγωνα πλακίδια με το κάθε πλακίδιο να έχει έναν βαθμό κινδύνου και πόρων. Ποια είναι η καλύτερη διαδρομή την οποία πρέπει να ακολουθήσουμε για να φτάσουμε τον στόχο μας; Πρέπει να δώσουμε προτεραιότητα στη συλλογή πόρων; Μήπως προτιμάμε την πιο ασφαλή διαδρομή, ή μάλλον την πιο γρήγορη; Συνήθως θέλουμε έναν συνδυασμό αυτών.

Η εύρεση διαδρομής είναι ένας τομέας στον οποίο πρέπει να δώσουμε βάση, καθώς παίζει σημαντικό ρόλο στο πόσο ευφυής φαίνεται η τεχνητή μας νοημοσύνη. Όσες καταστάσεις και αν έχει η μηχανή καταστάσεων μας δεν σημαίνει τίποτα αν το πρώτο πράγμα που βλέπει ο παίκτης είναι ένας εχθρός να κολλάει σε ένα βαρέλι.

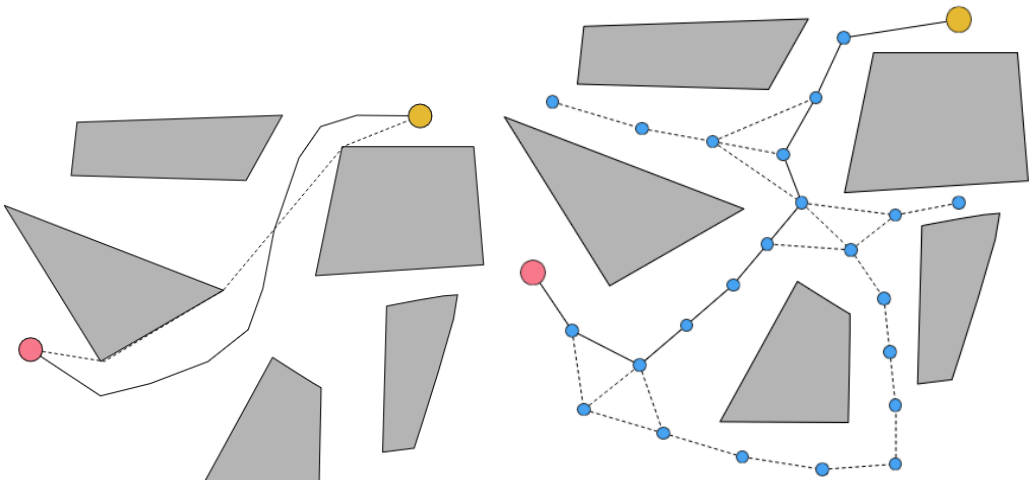
Όπως έχουμε αναφέρει, πολλά παιχνίδια χρησιμοποιούν κάποια παραλλαγή του A* αλγορίθμου. Ο άλλος αλγόριθμος που αναφέραμε είναι αυτός του Dijkstra, ο οποίος μπορεί να χρησιμοποιηθεί σε συνδυασμό με τον A* για πιο αποδοτικό AI. Ας χρησιμοποιήσουμε την παρακάτω εικόνα σαν παράδειγμα. Έστω ότι βρισκόμαστε στο δωμάτιο Α και θέλουμε να φτάσουμε στο Δ. Τότε μπορούμε να χρησιμοποιήσουμε τον αλγόριθμο του Djikstra για να βρούμε ότι η βέλτιστη διαδρομή στον χώρο των δωματίων είναι η Α → Β → Δ. Συνεπώς όταν εφαρμώσουμε τον A* για μεγαλύτερη απόδοση μπορούμε να αγνοήσουμε τα δωμάτια Γ και Ε για να σώσουμε υπολογιστικούς πόρους. Επιπροσθέτως να θυμίσουμε ότι αυτοί οι αλγόριθμοι δεν εφαρμόζονται στην πραγματική γεωμετρία του χώρου αλλά σε μία απλοποιημένη γεωμετρία η οποία ονομάζεται “πλέγμα πλοήγησης”. Όσο πιο απλή είναι η γεωμετρία μας τόσο πιο γρήγορος θα είναι ο αλγόριθμος.



Πλέγματα

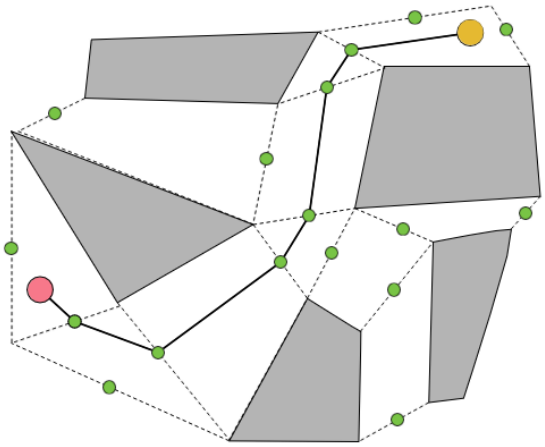
Υπάρχουν πολλοί τρόποι να σχεδιάσουμε ένα πλέγμα, ανάλογα με τις απαιτήσεις του παιχνιδιού μας. Παιχνίδια στα οποία δεν υπάρχουν προσβάσιμες περιοχές στις ίδιες συντεταγμένες αλλά σε διαφορετικό υψόμετρο (για παράδειγμα δεν υπάρχουν σπηλιές κάτω από μία πόλη) μπορεί να χρησιμοποιούν ένα δισδιάστατο πλέγμα φτιαγμένο είτε από πολύγωνα, ή από κελιά τα οποία επισημαίνονται ως προσπελάσιμα ή απροσπέλαστα. Κάθε αντικείμενο στον εικονικό κόσμο μπορεί να καταλάβει ένα ή περισσότερα κελιά. Θυμίζει κάπως το σύστημα μάχης του Dungeons & Dragons. Χαρακτηριστικό τέτοιων πλεγμάτων είναι ότι η αναζήτηση γίνεται γρήγορα και ότι μπορούμε εύκολα να αναπαραστήσουμε τον επίπεδο ή και μη επίπεδο χώρο αλλά με μικρές αλλαγές στο υψόμετρο. Τέτοιο σύστημα χρησιμοποιεί το Warcraft III.

Ένα πρόβλημα με την εύρεση διαδρομής είναι ότι οι αλγόριθμοι είναι σχεδιασμένοι για να βρίσκουν την βέλτιστη και όχι την πιο φυσική. Αυτό έχει ως αποτέλεσμα οι χαρακτήρες μας να φαίνονται σαν να κινούνται πολύ ρομποτικά ή να κολλάνε στους τοίχους. Στην παρακάτω εικόνα φαίνεται με την μαύρη γραμμή η προτεινόμενη διαδρομή σε αυτό τον χώρο, ενώ με τη διακεκομμένη γραμμή φαίνεται η διαδρομή που θα παράξει ο A*. Ένας τρόπος να αντιμετωπίσουμε αυτό το πρόβλημα είναι η εφαρμογή ενός γραφήματος για την αναπαράσταση του χώρου, αλλά μπορείτε να φανταστείτε πόσο μεγάλο και πολύπλοκο θα ήταν ένα τέτοιο γράφημα. Θα μπορούσαμε να χρησιμοποιήσουμε έναν αλγόριθμο για να παράξει ένα τέτοιο γράφημα αυτόματα αλλά για πιο πολύπλοκους χώρους η ανθρώπινη παρέμβαση είναι αναπόφευκτη.



Μία καλύτερη λύση είναι τα πλέγματα πλοήγησης. Σχεδιάζουμε τον χώρο μας σε ένα πολυγωνικό πλέγμα ελεύθερων χώρων. Επιπλέων η διαδρομή που ψάχνει ο A* γίνεται

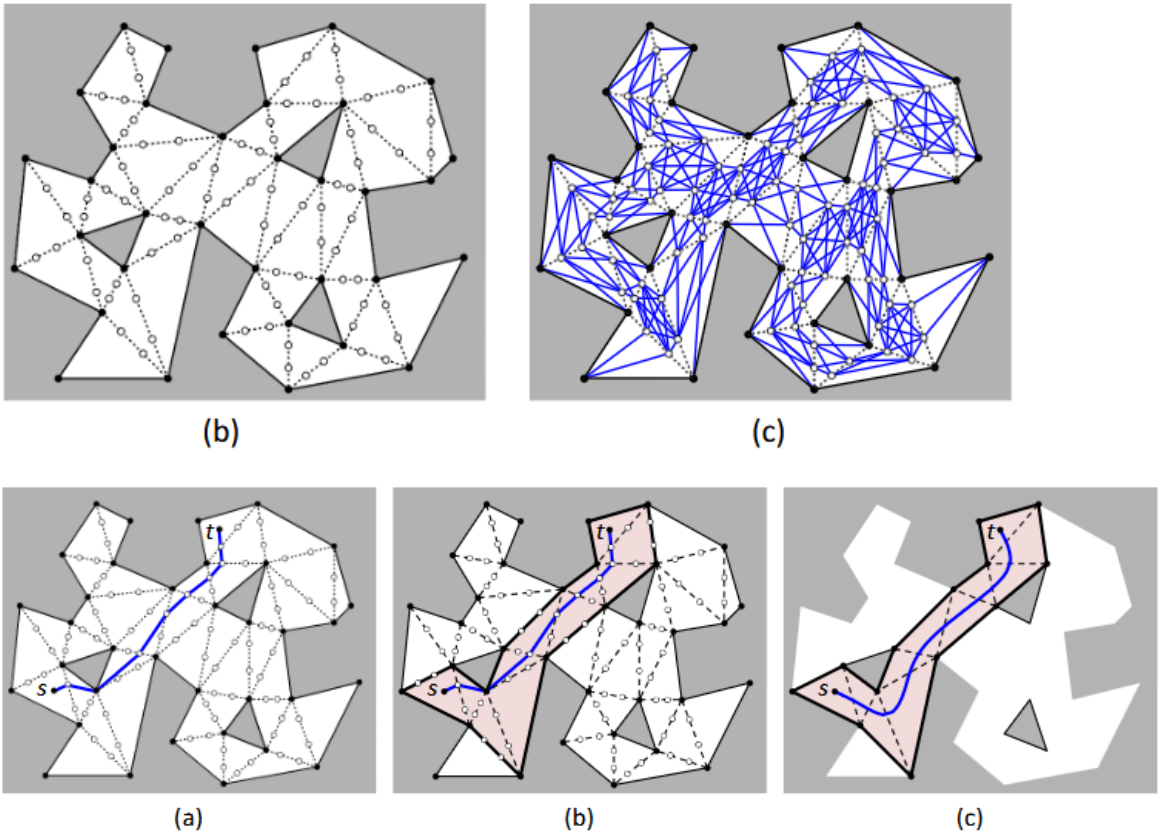
μεταξύ τον ακμών και των κέντρων των χώρων αυτών. Τα πλέγματα πλοήγησης μπορούν επίσης να έχουν διαφορετικούς τύπου εδάφους, ή να καθορίζονται από διαφορετικές περιοχές οι οποίες συνδέονται μέσω “πυλών”. Όταν η οντότητα και ο στόχος της βρίσκονται μέσα στην ίδια υποπεριοχή τότε μία ευθεία είναι αρκετή.



Τα πλέγματα πλοήγησης συνήθως δημιουργούνται από ανθρώπους, αλλά μπορούν να παραχθούν και από αλγορίθμους. Ένας τέτοιος αλγόριθμος πρώτα θα βρει τις προσιτές επιφάνειες αναπαριστώντας τον χώρο σαν έναν πολυγωνικό χάρτη. Στη συνέχεια θα απλοποιήσει τα σύνορα ή και εσωτερικές υποπεριοχές προκειμένου να έχει την πιο απλή γεωμετρία η οποία ακόμα αναπαριστά ικανοποιητικά τον χώρο. Τέλος, τριγωνοποιεί τον χάρτη. Ανεξαιρέτως του πως δημιουργήθηκε το πλέγμα, σχεδόν πάντα προϋπολογίζεται και φορτώνεται στην σκηνή ως μία στατική γεωμετρία, καθώς η παραγωγή καινούργιου πλέγματος είναι πολύ ακριβή για να γίνει σε πραγματικό χρόνο.

Όπως αναφέραμε, για την χρήση των πλεγμάτων πλοήγησης για εύρεση διαδρομής διακριτοποιούμε τις ακμές σε σημεία σχηματίζοντας ένα γράφημα το οποίο χρησιμοποιείται από το παιχνίδι για την εύρεση διαδρομών. Το πρόβλημα της αφύσικης κίνησης υπάρχει ακόμα, αλλά χάρη στο πιο απλό γράφημα η λύση του είναι πολύ πιο εύκολη, καθώς τώρα

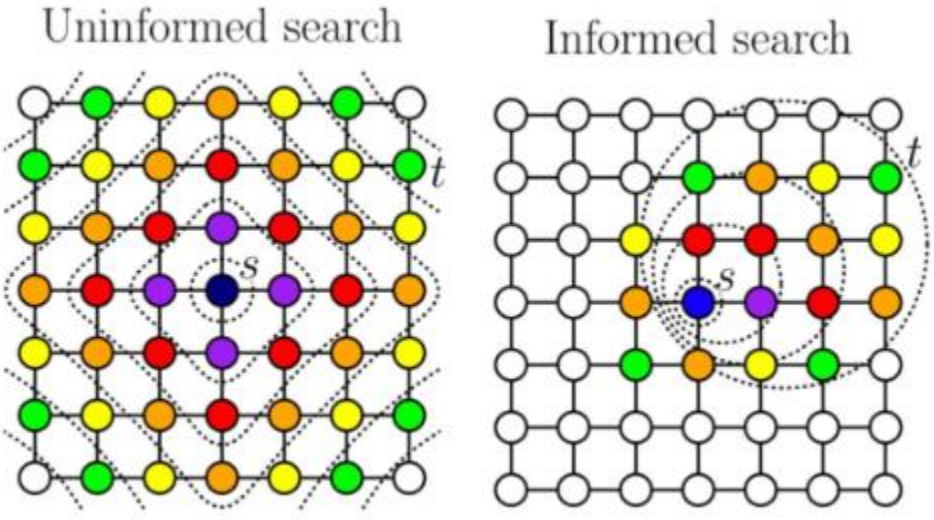
έχουμε λίγα προκαθορισμένα σημεία και μία πολύπλοκη γεωμετρία. Στη Unity υπάρχει αυτόματο σύστημα παραγωγής πλεγμάτων πλοήγησης μέσω του συστήματος δημιουργίας εδάφους. Χρησιμοποιεί τις ιδιότητες του συστήματος πλοήγησης του πράκτορα, όπως τη μέγιστη κλίση που μπορεί να ανέβει, για να αφαιρέσει από το πλέγμα τις μη προσβάσιμες περιοχές.



Υπολογισμός Συντομότερης Διαδρομής

Πλέον έχουμε μετατρέψει τον χώρο σε ένα γράφημα το οποίο θα χρησιμοποιήσουμε για την εύρεση της συντομότερης διαδρομής. Αφού το πρόβλημα ανάγεται σε αναζήτηση διαδρομής σε γράφημα μπορούμε να εκμεταλλευτούμε ιδιότητες των γραφημάτων, όπως τις κατευθυνόμενες ακμές ή βάρη. Η αναζήτηση σε γράφημα μπορεί να κατηγοριοποιηθεί ως μη-ενημερωμένη ή ενημερωμένη. Στην μη ενημερωμένη αναζήτηση επιλέγουμε τους κόμβους οι οποίοι βρίσκονται πιο κοντά στον αρχικό κόμβο και συνεχίζουμε την αναζήτηση αγνοώντας κόμβους οι οποίοι βρίσκονται ήδη στη λίστα. Μία καλύτερη εναλλακτική είναι η ενημερωμένη αναζήτηση, η οποία παίζει ρόλο και η απόσταση προς τον προορισμό και αγνοεί κόμβους οι οποίοι βρίσκονται πιο μακριά. Η απόσταση συνήθως υπολογίζεται

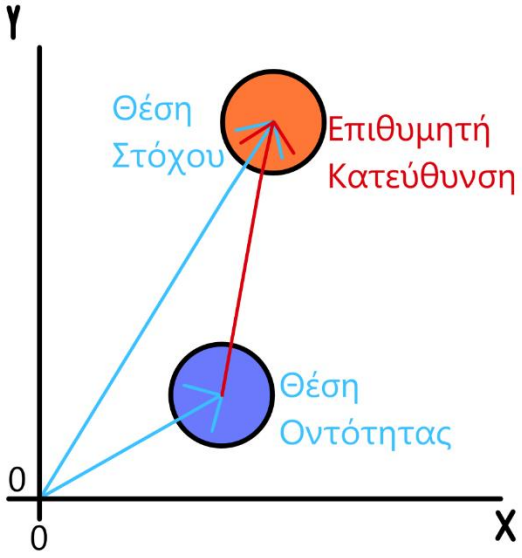
χρησιμοποιώντας Ευκλείδεια απόσταση, ωστόσο αν η κίνηση στο παιχνίδι μας είναι περιορισμένη σε πλακίδια τότε η απόσταση Manhattan μπορεί να είναι πιο ακριβής. Η χρήση της μεθόδου που χρησιμοποιούμε για την απόσταση από τον εξεταζόμενο κόμβο προς τον στόχο είναι κατά βάση ευρετική και εξαρτάται από πολλούς παράγοντες. Ο αλγόριθμος A* χρησιμοποιεί το άθροισμα της απόστασης του ελεγχόμενου κόμβου από την αρχή και την ευρετική απόσταση για να επιλέξει τον επόμενο κόμβο.



Κίνηση

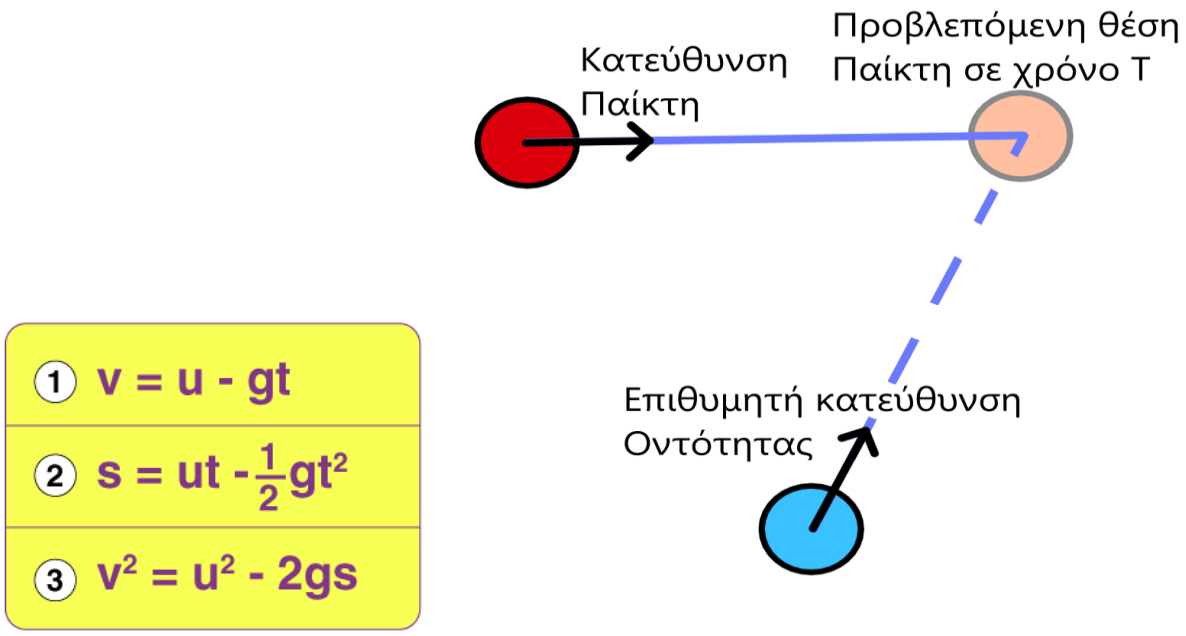
Η πιο απλή εφαρμογή της κίνησης είναι μία κατεύθυνση και μία σταθερή ταχύτητα. Αν γνωρίζουμε την κατεύθυνση ενός αντικειμένου και τον στόχο του, τότε πως μπορούμε να μεταβάλουμε την κατεύθυνσή του ώστε να κινείται προς τον στόχο αυτό; Ας υποθέσουμε ότι η οντότητα που κινείται και ο στόχος της περιγράφονται σε ένα σύστημα αξόνων, όπως στο παρακάτω σχήμα. Αν θυμηθούμε τις πράξεις διανυσμάτων, αν αφαιρέσουμε τη θέση της οντότητας από τη θέση του στόχου τότε το διάνυσμα που προκύπτει οδηγεί από τη θέση της οντότητας στον στόχο. Αυτό το διάνυσμα μπορούμε να το κανονικοποιήσουμε ώστε να έχει μέτρο 1 και τέλος αρκεί να το πολλαπλασιάσουμε με το μέτρο της ταχύτητας της οντότητας για να κινηθεί προς τον στόχο της. Αν θέλουμε η οντότητα να απομακρύνεται από τον στόχο, τότε το διάνυσμα που ψάχνουμε είναι απλώς το αντίθετο αυτού που υπολογίσαμε πριν, οπότε αρκεί είτε να πολλαπλασιάσουμε το διάνυσμα με -1 ή να αφαιρέσουμε τη θέση του στόχου από τη θέση της οντότητας. Φυσικά αυτή είναι μία πολύ απλή μέθοδος για να βρούμε την κατεύθυνση, αλλά δεν είναι η μόνη. Αντί αυτού μπορούμε να υπολογίσουμε την επιθυμητή κατεύθυνση και να εφαρμόσουμε έναν περιστροφικό μετασχηματισμό στην τωρινή κατεύθυνση χρησιμοποιώντας LERP για να αλλάξουμε κατεύθυνση πιο ομαλά. Η ταχύτητά

μας επίσης δεν πρέπει να είναι σταθερή, αντιθέτως αρκεί να εφαρμόσουμε μία αθροιστική τιμή στην τωρινή ταχύτητα της οντότητάς μας μέχρι να φτάσει στην μέγιστη ταχύτητά της. Σημασία έχει ότι η κίνηση περιγράφεται ως μία κατεύθυνση και μία ταχύτητα, το πως υπολογίζουμε αυτά τα δύο εξαρτάται από την εφαρμογή μας.

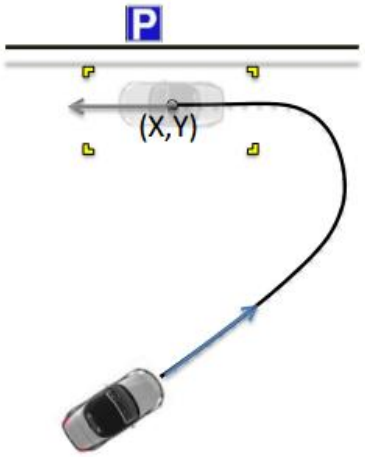


Αφού μιλάμε για τεχνητή νοημοσύνη μπορεί διάφοροι παράγοντες να επηρεάσουν την κατεύθυνση ή την ταχύτητα της οντότητας. Για παράδειγμα, όταν η οντότητα φτάσει στον στόχο της μάλλον δεν θέλουμε να σταματάει αμέσως, αλλά να ελατώνει ταχύτητα πιο ομαλά. Σε αυτή την περίπτωση θα πρέπει να εφαρμόσουμε μία δύναμη η οποία θα ασκείται στην αντίθετη κατεύθυνση από αυτή την οποία χρησιμοποιεί η οντότητα ώστε να επιβραδύνει. Στην πραγματικότητα θα αφαιρούμε ένα μικρό ποσό της ταχύτητάς της κάθε κάποιο χρονικό διάστημα, μάλλον ανά καρέ. Επιπροσθέτως μπορεί να θέλουμε η οντότητα να φτάσει σε κάποιο σημείο σε ένα συγκεκριμένο χρονικό διάστημα. Γνωρίζοντας την απόσταση που πρέπει να διανύσει η οντότητα, τον χρόνο στον οποίο πρέπει να την διανύσει, και την τωρινή της ταχύτητα, μπορούμε να θυμηθούμε τους τύπου κίνησης της Μηχανικής και να λύσουμε ως προς την επιτάχυνση. Η κινηματική φυσική παίζει σημαντικό ρόλο στην μοντελοποίηση ρεαλιστικών συστημάτων στα βιντεοπαιχνίδια. Για παράδειγμα, ας υποθέσουμε ότι ο παίκτης κινείται με σταθερή ταχύτητα προς μία κατεύθυνση και στο παιχνίδι μας υπάρχουν αστυνομικοί οι οποίοι τον καταδιώκουν σε οχήματα. Πως θα βρούμε την κατεύθυνση που πρέπει να ακολουθήσει ο αστυνομικός για να βρεθεί στον δρόμο του παίκτη; Γνωρίζοντας τις ταχύτητες και κατευθύνσεις των οχημάτων το πρόβλημα θυμίζει εκείνο το θέμα με τα δύο τρένα τα οποία κινούνται αντίθετα από και προς δύο σταθμούς και πρέπει να βρούμε σε πιο σημείο θα συναντηθούν και σε πόσο χρόνο. Στην δική μας περίπτωση μπορούμε να χρησιμοποιήσουμε μία προσέγγιση. Ας υπολογίσουμε τον χρόνο που θα χρειαζόταν ο καταδιώκτης να φτάσει τον παίκτη αν αυτός ήταν ακίνητος. Αυτό ο χρόνος T ισούται με την απόσταση του καταδιώκτη από τον παίκτη διά την ταχύτητά του. Τώρα υπολογίζουμε την

προβλεπόμενη θέση του παίκτη σε χρόνο T, η οποία θα είναι η ταχύτητα του παίκτη επί τον χρόνο T. Έχοντας τη θέση στην οποία θέλουμε να καταλήξει ο καταδιώκτης και γνωρίζοντας την ίδια του τη θέση μπορούμε να βρούμε την κατεύθυνση όπως τη βρήκαμε πριν και να εφαρμόσουμε μία ταχύτητα σε αυτή τη κατεύθυνση. Όταν ο καταδιώκτης φτάσει στον στόχο του ή ο παίκτης απομακρυνθεί αρκετά επαναλαμβάνουμε την διαδικασία για να βρούμε καινούργιο στόχο. Αυτή η προσέγγιση δεν είναι τέλεια αλλά είναι αρκετή για το απλό μας παράδειγμα. Ένα μεγαλύτερο παιχνίδι όπως τα Grand Theft Auto μάλλον έχουν έναν πιο πολύπλοκο αλγόριθμο για να βρουν τον στόχο του καταδιώκτη. Όλα αυτά τα προβλήματα είναι αναμφίβολα δύσκολα να μοντελοποιηθούν, ωστόσο είναι απαραίτητα για να κάνουν το παιχνίδι μας πιο ρεαλιστικό, προκλητικό, και να φαίνεται λιγότερο ρομποτικό.

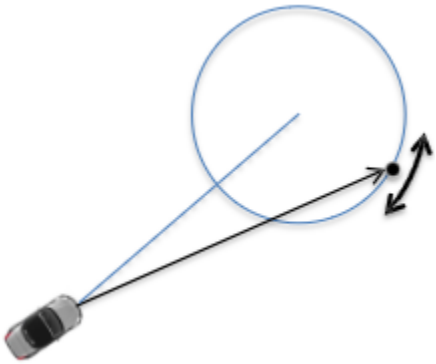


Μέχρι στιγμής έχουμε μιλήσει για απλές ευθείες κινήσεις, αλλά αυτές οι κινήσεις δεν είναι πάντα επαρκείς για το παιχνίδι μας. Ας υποθέσουμε ότι θέλουμε να προγραμματίσουμε την τεχνητή νοημοσύνη ενός αυτοκινήτου το οποίο πρέπει να παρκάρει. Τώρα δεν θέλουμε μόνο το αυτοκίνητο να φτάσει στην θέση του πάρκινγκ, αλλά θέλουμε και να βρίσκεται σε μία συγκεκριμένη κατεύθυνση, παράλληλη ή κάθετη με την θέση του πάρκινγκ. Για να το πετύχουμε αυτό μπορούμε να υπολογίσουμε μία διαδρομή η οποία όταν καταλήξει στη θέση του πάρκινγκ το αυτοκίνητο θα βρίσκεται στην επιθυμητή κατεύθυνση. Έτσι το αυτοκίνητο θα ακολουθήσει τη διαδρομή. Αυτή η λύση προϋποθέτει ότι δεν υπάρχουν εξωτερικές επιρροές. Τι θα γίνει αν ένα άλλο όχημα πλησιάσει το αυτοκίνητο; Μάλλον θέλουμε ένα ή και τα δύο οχήματα να εκτελέσουν κάποια μανούβρα για να αποφύγουν σύγκρουση ή ίσως να χαμηλώσουν ταχύτητα. Μπορούμε να χρησιμοποιήσουμε το προηγούμενο παράδειγμα για να προβλέψουμε την θέση του άλλου οχήματος σε χρόνο T αλλά τώρα αντί να φτάσουμε σε αυτό τον στόχο θέλουμε να τον αποφύγουμε.



Ξέρουμε πως να κινούμαστε σε έναν στόχο αλλά τι θα πρέπει να κάνουμε όταν δεν υπάρχει απλός στόχος όπως σε έναν μεγάλο δρόμο; Μπορούμε να χρησιμοποιήσουμε το πλέγμα διαδρομής αλλά τότε όλα τα οχήματα θα κινούνται με ακριβώς τον ίδιο τρόπο, το οποίο θα φαίνεται ρομποτικό. Για να δώσουμε λίγη ζωή στα οχήματά μας χρησιμοποιούμε την έννοια της περιπλάνησης. Πιο ακριβές θα ήταν να εφαρμόζουμε τυχαίες δυνάμεις στροφής πάνω στον ίδιο τον πράκτορα, αλλά υπάρχει ένας πιο εύκολος τρόπος ο οποίος παρέχει και καλύτερα αποτελέσματα. Αφού χρησιμοποιούμε ήδη σύστημα στόχου μπορούμε να προσθέσουμε λίγη τυχαιότητα στον στόχο αυτό. Έτσι αντί να στοχεύουμε ακριβώς τον επόμενο κόμβο στον γράφο, στοχεύουμε κάπου τυχαία αλλά κοντά στον κόμβο αυτό. Ο καινούριος μας στόχος διαλέγει ένα τυχαίο σημείο πάνω σε έναν κύκλο με κέντρο τον επόμενο κόμβο. Αφού ο πράκτορας πλέον δεν έχει στόχο ακριβώς τον επόμενο κόμβο αλλά κάπου κοντά σε αυτόν, θέλουμε να διαλέξουμε καινούργιο στόχο όταν βρισκόμαστε αρκετά κοντά στον κόμβο, όχι ακριβώς πάνω σε αυτόν.

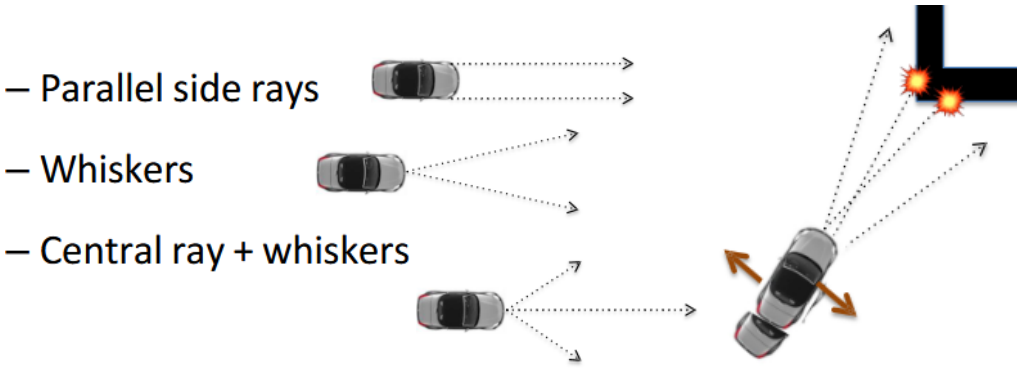
Φυσικά, όλα αυτά συνδέονται με κάποια μηχανή πεπερασμένων καταστάσεων για λήψη απόφασης ή παρόμοιο σύστημα ώστε να ακολουθούνται διαφορετικές συμπεριφορές σε διαφορετικές καταστάσεις. Για παράδειγμα, ένα περιπολικό ακολουθεί τον δρόμο μέχρι να αντιληφθεί τον παίκτη, οπότε τότε αλλάζει σε κατάσταση καταδίωξης. Πολύ πιθανό να πρέπει να συνδυάσουμε κινήσεις. Για παράδειγμα ας υποθέσουμε ότι ένα περιπολικό καταδιώκει τον παίκτη και ένα άλλο όχημα βρίσκεται ανάμεσά τους. Τότε δεν θέλουμε το περιπολικό να τρέξει ευθεία και να συγκρουστεί με το άλλο όχημα, αλλά να το αποφύγει. Σε τέτοιες περιπτώσεις μπορούμε να αθροίσουμε την δύναμη στροφής που παράγει η κάθε κατάσταση ή να εφαρμόσουμε αυτή με τη μεγαλύτερη προτεραιότητα. Προφανώς εμείς ορίζουμε τι προτεραιότητα έχει η κάθε κατάσταση.



Ένα γνωστό σύστημα κίνησης είναι τα boids. Το μοντέλο αυτό αποτελείται από τρεις κανόνες: διαχωρισμός, ευθυγράμμιση, και συσσώρευση, και έχει στόχο να προσομοιώσει την κίνηση ενός σμήνος πουλιών. Σύμφωνα με τον διαχωρισμό, ο πράκτορας προσπαθεί να στρίψει ώστε να αποφύγει τους κοντινούς του πράκτορες. Ο κανόνας ευθυγράμμισης ορίζει ότι ο πράκτορας θα στρίψει ώστε η κατεύθυνσή του να αντιστοιχεί με την μέση κατεύθυνση και ταχύτητα του σμήνος. Τέλος σύμφωνα με τον κανόνα συσσώρευσης ο πράκτορας θα στρίψει ώστε να κινηθεί προς το κέντρο μάζας του σμήνους για να μην απομακρυνθεί πολύ από αυτό. Τα αποτελέσματα αυτών των κανόνων αθροίζονται και η τελική δύναμη εφαρμόζεται στον πράκτορα.

Έχουμε πει πως ο πράκτορας προσπαθεί να αποφύγει συγκρούσεις, αλλά πως καταλαβαίνει ότι πρόκειται να συγκρουστεί; Είπαμε ήδη ότι μπορούμε να χρησιμοποιήσουμε τον αλγόριθμο εύρεσης στόχου κατά την καταδίωξη αλλά αυτό δεν είναι πάντα αρκετό. Ο αλγόριθμος αυτός επιστρέφει ένα μόνο σημείο στον χώρο, ωστόσο στην πραγματικότητα εμείς δεν δουλεύουμε με σημεία αλλά με αντικείμενα τα οποία έχουν όγκο. Θα μπορούσαμε να ελέγξουμε έναν σφαιρικό χώρο με κέντρο το σημείο που προβλέψαμε αλλά αυτό δεν θα είναι ακριβές για αντικείμενα τα οποία δεν έχουν σφαιρικό σχήμα. Άλλα γεωμετρικά σχήματα είναι πολύ πιο δύσκολο να μοντελοποιηθούν. Αφού ένα σημείο δεν είναι αρκετό χρειαζόμαστε μία άλλη μέθοδο. Πολύ συχνά παιχνίδια χρησιμοποιούν ακτίνες για έλεγχο σύγκρουσης. Μία ακτίνα με κέντρο το όχημά μας και κατεύθυνση ίδια με αυτή του μοντέλου του οχήματος είναι αρκετή για να ελέγξουμε αν υπάρχει κάποιο αντικείμενο ακριβώς μπροστά από το όχημα. Για άλλες κατευθύνσεις χρησιμοποιούμε άλλες ακτίνες. Οι ακτίνες είναι σχετικά φτηνές οπότε δεν υπάρχει πρόβλημα με το πόσο τις χρησιμοποιούμε. Επιπροσθέτως, έχοντας διαφορετικό μήκος ακτίνων για διαφορετικές κατευθύνσεις

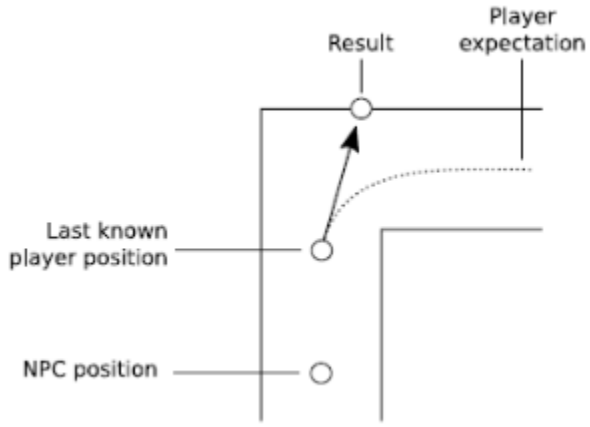
μπορούμε να μοντελοποιήσουμε τον χώρο όρασης του οδηγού. Είναι πιο πιθανό ένας οδηγός να αποφύγει ένα αντικείμενο το οποίο βρίσκεται μπροστά του αλλά μακριά παρά ένα το οποίο βρίσκεται δεξιά ή αριστερά του (αφού κινείται ευθεία). Όσον αφορά το ποια δύναμη θα εφαρμόσουμε, η πιο απλή υλοποίηση είναι η κάθε ακτίνα να παράξει μία δύναμη και να εφαρμόσουμε το άθροισμά τους, δηλαδή τον μέσο όρο. Αυτό μπορεί να προκαλέσει προβλήματα αν το άθροισμα των δυνάμεων είναι κοντά στο μηδέν. Σε τέτοιες περιπτώσεις θα πρέπει να διαλέξουμε μία δύναμη τυχαία.



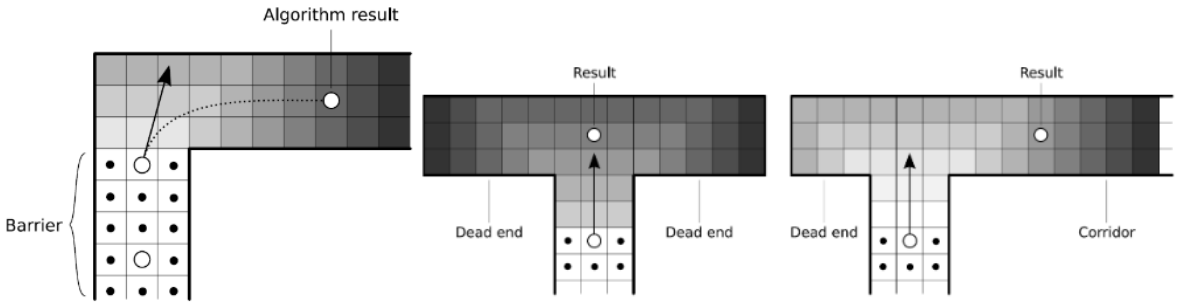
Ανάλυση του DISHONORED 2

Τέλος, ας μελετήσουμε το σύστημα κίνησης ενός παιχνιδιού και συγκεκριμένα το σύστημα καταδίωξης του DISHONORED 2.

Οι εχθρικοί NPC όταν καταλάβουν ότι ο παίκτης βρίσκεται στην περιοχή τους αλλάζουν sthn κατάσταση καταδίωξης. Αν ο χαρακτήρας έχει άμεση οπτική επαφή με τον παίκτη τότε επιτίθενται, αλλιώς προσπαθεί να τον βρει. Το πετυχαίνει αυτό τρέχοντας προς ένα σημείο που πιστεύει ότι είναι ο παίκτης, αν τον βρει επιτίθεται, αλλιώς μπαίνει σε κατάσταση αργής αναζήτησης στον κοντινό χώρο. Στο πρώτο παιχνίδι της σειράς, DISHONORED, χρησιμοποιήθηκε η τεχνική ψίχουλων. Ο παίκτης όσο κινείται αφήνει πίσω του “ψίχουλα” τα οποία οι εχθροί ακολουθούν. Η τεχνική αυτή έφερε καλά αποτελέσματα αλλά ήταν άδικη για τον παίκτη. Στο δεύτερο παιχνίδι αρχικά χρησιμοποίησαν την τεχνική dead reckoning. Όταν χανόταν η οπτική επαφή με τον παίκτη εφαρμοζόταν μία γραμμική προέκταση με βάση τη θέση αυτή. Ο επόμενος στόχος του εχθρού ήταν το αποτέλεσμα αυτής της προέκτασης. Αν και αρχικά δούλεψε, οι play testers ανέφεραν ότι η καταδίωξη σταματούσε υπερβολικά γρήγορα.



Αντί αυτού, χρησιμοποίησαν έναν χάρτη επιρροής. Ο χάρτης χωρίζεται σε κελιά με το κάθε κελί να έχει μία τιμή θερμότητας αντιστρόφως ανάλογα με το πόσο κοντά είναι στο αρχικό σημείο ελέγχου, δηλαδή στον εχθρό, ενώ τα εμπόδια όπως οι τοίχοι διατηρούσαν μηδενική θερμότητα. Ο νέος στόχος αποτελεί το σημείο στο οποίο αντιστοιχεί η μέση θέση των θερμότερων κελιών στο χάρτη. Αυτό το σύστημα είναι επίσης εξαιρετικά ρυθμίσιμο, επιτρέποντας στους σχεδιαστές να αλλάξουν τη δυσκολία του παιχνιδιού αλλάζοντας τρεις απλές μεταβλητές. Με τη μεταβλητή θερμότητας ο σχεδιαστής ρυθμίζει το πόσο γρήγορα τα κελιά κρυώνουν στη διάρκεια του αλγορίθμου, αυξάνοντας ή μειώνοντας έτσι την πιθανότητα ενός πράκτορα να εξερευνήσει μεγάλους διαδρόμους. Η μεταβλητή “μέγιστο πλήθος θερμών κελιών ανά βήμα” (M_h) χρησιμοποιείται παρομοίως αλλά για μεγάλα δωμάτια ρυθμίζοντας το πόσο εύκολα σταματάει η διάδοση της θερμότητας του αλγορίθμου σε αυτά. Τέλος, η μεταβλητή μέγιστου πλήθους βημάτων ορίζει πόσο μακριά επιτρέπεται να τρέξει ο αλγόριθμος.



Αν και ο αλγόριθμος αυτός ήταν αποτελεσματικός δεν ήταν τέλειος. Ένα πρόβλημα ήταν ότι δεν εφαρμόζεται καλά σε νέα πληροφορία όπως κάποιον ήχο τον οποίο ο εχθρός θεωρεί ύποπτο. Για κάθε καινούρια πληροφορία, θα πρέπει ο αλγόριθμος να ξανατρέξει από την αρχή. Ένα άλλο πρόβλημα του αλγορίθμου ήταν ότι υπήρχε η περίπτωση περιοχές καταδίωξης να χαθούν. Αφού ο αλγόριθμος αγνοούσε κελιά “πίσω” από τον εχθρό, σε κυκλικές περιοχές όπως φαίνεται παρακάτω πιθανές έξοδοι χάνονται. Οι προγραμματιστές δεν είχαν τον χρόνο να το λύσουν αυτό το πρόβλημα στον αλγόριθμο. Ωστόσο λύθηκε σε

επίπεδο σχεδιασμού επιπέδων. Αφού ήξεραν ότι ο αλγόριθμος παρουσίαζε πρόβλημα σε τέτοιες περιπτώσεις, απέφυγαν στα επίπεδα να εμφανιστούν αυτές οι περιπτώσεις.