ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ημίτονο και συνημίτονο.
Η σημασία τους στο Game Development

Αντωνίου Θωμάς – Σιάχος Κωνσταντίνος

ΠΑΤΡΑ 2020

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή

Πάτρα, __ / __ / ____

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1. _____, _____
2. _____, _____
3. _____, _____

Υπεύθυνη Δήλωση Φοιτητή

Βεβαιώνω ότι είμαι συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης έχω αναφέρει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επίσης βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά ειδικά για τη συγκεκριμένη εργασία.

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Πανεπιστημίου Πελοποννήσου δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα εκ μέρους του Τμήματος.

Η παρούσα εργασία αποτελεί πνευματική ιδιοκτησία των φοιτητών Αντωνίου Θωμά και Σιάχου Κωνσταντίνου που την εκπόνησαν.

Abstract (Introductory Note)

During the history of electronic gaming, programmers have been trying to create more realistic games. Not only graphic wise, but also how objects react with each other and with the environment. For this purpose, developers, used, use, and will always be using, mathematics, achieving a much more enjoyable game.

There are games in which the player must eliminate the opposing team, others in which an object must reach a specific spot. In general, many games use physics to create a more realistic feel and simultaneously functional experiences. For example, “Assassin’s Creed” and “Grand Theft Auto” use physics to realistically portray acrobatics and combat. “Rocket League” uses physics to mimic gravity and how players react with the ball and with each other.

Trigonometry in gaming can cover many different sectors and solve some otherwise difficult problems when programming one. Crashing, gravity, 3D models, object placement, trajectories… The fact that more and more people are engaging competitive gaming lately, makes precision of collision between players and objects ultimately important. Time, location and level of difficulty must always be taken into account when developing such conditions.

Περίληψη (Εισαγωγικό Σημείωμα)

Καθ’ όλη την ιστορία των ηλεκτρονικών παιχνιδιών οι προγραμματιστές προσπαθούσαν να δημιουργήσουν όλο και πιο ρεαλιστικά παιχνίδια. Όχι μόνον όσον αφορά τα γραφικά, αλλά και κατά πώς τα αντικείμενα αλληλοεπιδρούν. Οι προγραμματιστές χρησιμοποιούσαν, χρησιμοποιούν και θα χρησιμοποιούν για πάντα μαθηματικά για την επίτευξη αυτών των στόχων και τη δημιουργία πιο απολαυστικών παιχνιδιών.

Υπάρχουν παιχνίδια στα οποίο ο χρήστης χρειάζεται να εξοντώσει την αντίπαλη ομάδα, άλλα στα οποία ένα αντικείμενο πρέπει να φτάσει σε κάποιο συγκεκριμένο σημείο. Όμως γενικά, τα περισσότερα από αυτά χρησιμοποιούν physics για την δημιουργία πιο ρεαλιστικών και ταυτόχρονα λειτουργικών εμπειριών. Για παράδειγμα το Assassin’s Creed και το Grand Theft Auto, χρησιμοποιούν physics για τη ρεαλιστική απεικόνιση ακροβατικών και σκηνών μάχης. Παρομοίως το Rocket League χρησιμοποιεί physics για τη μίμηση της βαρύτητας και για το πως ο παίχτης αλληλοεπιδρά με μία μπάλα και τους υπόλοιπους παίχτες.

Συγκεκριμένα, ηλεκτρονικά παιχνίδια χρησιμοποιούν τριγωνομετρία σε διάφορους τομείς. Συγκρούσεις, βαρύτητα, μοντέλα παιχτών και αντικειμένων, σχεδίαση πίσω από τα σκηνικά και τα γραφικά, τροχιές βλημάτων. Το γεγονός, πως όλο και περισσότεροι παίχτες ασχολούνται με άκρως ανταγωνιστικά παιχνίδια, κάνει την ακρίβεια της σύγκρουσης μεταξύ ενός αντικειμένου και του παίχτη ή μεταξύ παιχτών πολύ σημαντική. Ο χρόνος, η τοποθεσία και ο βαθμός δυσκολίας, πάντα λαμβάνονται υπ’ όψιν όταν προγραμματίζουμε τέτοιες συγκρούσεις.

Ημίτονα και γραμμές

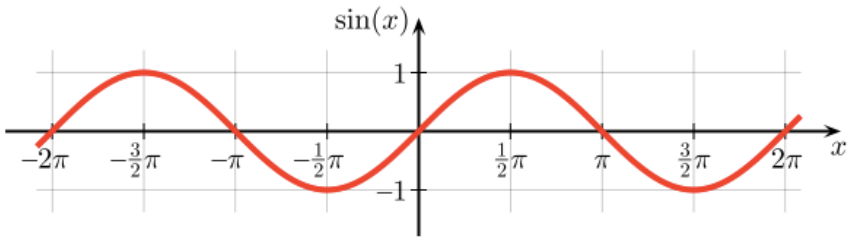
Ανέκαθεν, στα σχολικά και επιστημονικά βιβλία μας παρουσιάζεται εξ' αρχής το παρακάτω διάγραμμα, το οποίο περιγράφει ένα ημίτονο. Τα σχολικά μαθήματα περί τριγωνομετρίας δίνουν στους μαθητές να καταλάβουν ότι «Οι κύκλοι έχουν ημίτονα. Τα ημίτονα προέρχονται από κύκλους.»

Αυτό είναι λάθος. Οι κύκλοι είναι ένα παράδειγμα ημιτόνου. Το ημίτονο είναι μια φυσική κλίση, η επιτομή της ομαλότητας: κάνει τους κύκλους "κυκλικούς" με τον ίδιο τρόπο που οι γραμμές κάνουν το τετράγωνο "τετράγωνο". Ας ξεκινήσουμε, βλέποντας το ημίτονο ως το δικό του σχήμα και έπειτα θα καταλάβουμε πως "δένει" με τους κύκλους και τα συναφή.

Εδώ πρέπει να θυμόμαστε να διαχωρίζουμε την "ιδέα" από το "παράδειγμα". Τα τετράγωνα είναι παραδείγματα γραμμών.

Γραμμική κίνηση: Η γραμμική κίνηση είναι σταθερή. Πηγαίνουμε με μία σταθερή ταχύτητα και επιστρέφουμε στιγμιαία. Είναι η αφύσικη κίνηση του robot dance.

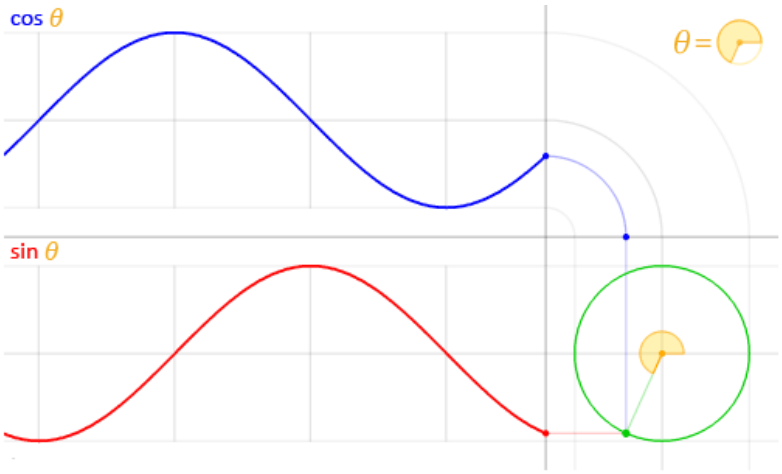
Κίνηση ημιτόνου: Το ημίτονο μεταβάλλει την ταχύτητα του, ξεκινάει γρήγορα, επιβραδύνει, σταματάει, επιστρέφει και επιταχύνει πάλι. Είναι η γοητευτική ομαλότητα του liquid dancing.



Αυτό είναι το σχηματικό διάγραμμα που μας παρουσιάζεται ανέκαθεν στα σχολικά βιβλία. Μας δίνει την αίσθηση του ημιτόνου; Όχι περισσότερο όσο ένας σκελετός μίας γάτας περιγράφει την ευκινησία της.

Το ημίτονο ξεκινάει τον κύκλο του από το ουδέτερο μέσο σημείο και κατευθύνεται προς το μέγιστο. Μία άλλη έκδοση του ημιτόνου είναι να ξεκινήσει ο κύκλος του από το μέγιστο και να πέσει προς το μέσο σημείο. Αυτό ονομάζεται συνημίτονο και είναι απλά μία άλλη έκδοση του ημιτόνου, όπως ακριβώς μία κάθετη ευθεία γραμμή είναι μια άλλη έκδοση μίας ευθείας οριζόντιας γραμμής. Ένα οριζόντιο και ένα κάθετο "ελατήριο" συνδυάζονται και μας δίνουν κυκλική κίνηση ως αποτέλεσμα.

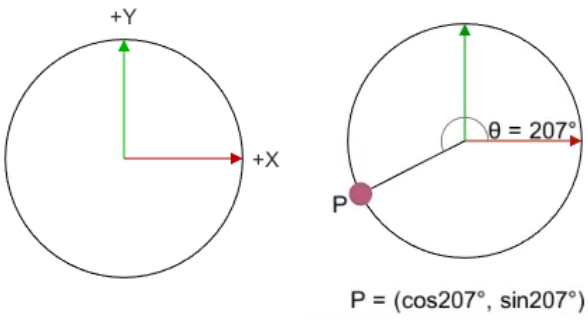
Οι κύκλοι και τα τετράγωνα είναι ένας συνδυασμός βασικών στοιχείων (ημίτονα και γραμμές). Ο κύκλος κατασκευάζεται από δύο συνδεδεμένα κύματα μίας διάστασης, κάθε ένα από τα οποία κινεί την οριζόντια και κάθετη κατεύθυνση



Γεωμετρική Ερμηνεία

Ας υποθέσουμε ότι έχουμε ένα κύκλο με ακτίνα 1, με κέντρο την αρχή των αξόνων.

Επιλέγουμε ένα σημείο P στον κύκλο. Το τμήμα της γραμμής μεταξύ αυτού του σημείου και την αρχή των αξόνων σχηματίζει μια γωνία θ μεταξύ αυτού και του άξονα x (θετική κατεύθυνση x).



Αυτό που βλέπουμε εδώ είναι ένας τρόπος να εκφράσουμε γεωμετρικά τις δύο βασικές τριγωνομετρικές συναρτήσεις: $\sin(\theta)$ (ημίτονο της γωνίας θ) και $\cos(\theta)$ (συνημίτονο της γωνίας θ). Οι συντεταγμένες (x, y) αυτού του σημείου P είναι ακριβώς $(\cos(\theta)\,\sin(\theta))$.

Οι δύο τριγωνομετρικές συναρτήσεις $\sin(\theta)$ και $\cos(\theta)$ είναι, αντίστοιχα, οι συντεταγμένες y και x ενός σημείου στον μοναδιαίο κύκλο, όπου θ είναι η γωνία από τον άξονα x (θετική κατεύθυνση x) στο τμήμα γραμμής μεταξύ του σημείου P και της αρχής των αξόνων.

Τα $\sin(\theta)$ και $\cos(\theta)$ είναι συναρτήσεις που παίρνουν μία μόνο είσοδο (γωνία) και εξάγουν μία μόνο τιμή μεταξύ -1 και 1. Αυτό το εύρος εξόδων έχει λογική: οι συντεταγμένες x και y ενός το σημείο στον κύκλο μονάδας δεν μπορεί ποτέ να φτάσει έξω από την περιοχή $[-1, 1]$.

Εάν η γωνία θ αυξάνεται με σταθερό ρυθμό, μπορούμε να σχεδιάσουμε την τιμή των συντεταγμένων x και y του σημείου μεμονωμένα, κατά το πέρασμα του χρόνου.

Αν συγκρίνουμε τα διαγράμματα δίπλα-δίπλα με τη μορφή γωνίας έναντι τιμής, παρατηρούμε ότι είναι η ίδια περιοδική καμπύλη σε σχήμα κύματος, αλλά αντισταθμίζεται με το ένα τέταρτο από την άλλη.

Η περίοδος των συναρτήσεων αυτών είναι $360°$, οπότε το $\sin 450°$ δίνει την ίδια τιμή με το $\sin 90°$. Αυτό συμβαίνει επειδή η επιπλέον $360°$ περιστροφή από τις $90°$ θα μας φέρει πίσω στην ίδια γωνία.

Ένα πρακτικό παράδειγμα στο οποίο το ημιτονοειδές κύμα είναι πολύ εύχρηστο, είναι η κίνηση ενός κέρματος (Super Mario) για το οποίο θέλουμε να δημιουργήσουμε μία αιωρούμενη κίνηση.

Κάθε φορά που χρειάζεται να κινήσουμε κάποιο αντικείμενο κατά το πέρασμα του χρόνου, απλά χρησιμοποιούμε τον αριθμό των δευτερολέπτων που έχουν περάσει από τη στιγμή που ξεκινήσαμε το παιχνίδι, ως είσοδο στην ημιτονοειδή συνάρτηση.

Σε αυτή την περίπτωση μπορούμε να ορίσουμε την θέση y του κέρματος να είναι ίση με το ημίτονο t, όπου t ο χρόνος.

Αυτό σημαίνει πως το κέρμα ξεκινάει από τη θέση μηδέν και καθώς κυλάει ο χρόνος το κέρμα αιωρείται ομαλά πάνω και κάτω.

Αν θέλουμε να κάνουμε το κέρμα να κινείται πιο γρήγορα, μπορούμε να πολλαπλασιάσουμε το χρόνο με μία μεταβλητή speed.

Πολλαπλασιάζοντας με το 2, θα κάνει την κίνηση στη διπλάσια ταχύτητα της αρχικής, όπως και πολλαπλασιάζοντας με 0.5 θα κάνει την κίνηση δύο φορές πιο αργή.

Μπορούμε επίσης να κάνουμε το κέρμα να αιωρείται πιο ψηλά και πιο χαμηλά ταυτόχρονα πολλαπλασιάζοντας ολόκληρη τη συνάρτηση $\sin(t)$ με μία μεταβλητή amplitude.

Πολλαπλασιάζοντας με το 2, το κέρμα θα αιωρείται 2 μονάδες προς τα πάνω και 2 προς τα κάτω ταυτόχρονα. Και πάλι μπορούμε να μειώσουμε την απόσταση που το κέρμα κινείται, πολλαπλασιάζοντας με ένα μικρότερο αριθμό.

Μοίρες και Ακτίνια

Η γωνία που θέτουμε στις τριγωνομετρικές συναρτήσεις μπορεί να είναι σε δύο διαφορετικές μονάδες μέτρησης: μοίρες και ακτίνια. Οι περισσότεροι είμαστε εξοικειωμένοι περισσότερο με τις μοίρες. Για παράδειγμα, η ορθή γωνία (90 μοίρες) γράφεται με αυτό τον τρόπο $90°$. Όμως το $\sin 90°$ δεν είναι το ίδιο με το $\sin 90$. Αν η βαθμολογία δεν είναι παρούσα, ως μονάδα μέτρησης γωνίας θεωρείται το ακτίνιο.

Οι $180°$ είναι ισοδύναμες με π ακτίνια, όπου π, η περίφημη μαθηματική σταθερά, η αναλογία της περιφέρειας του κύκλου προς τη διάμετρο του, περίπου ίση με 3.14. Ως εκ τούτου, μια γωνία ενός ακτινίου είναι περίπου $\frac{180°}{\pi} \approx 57.3°$, σχεδόν $60°$. Ως πράξη εχεφροσύνης, αν εισάγουμε σε ένα κομπιουτεράκι $\sin 1$ (με μονάδες μέτρησης γωνίας ρυθμισμένες σε ακτίνια) θα μας δώσει αποτέλεσμα περίπου 0.84, το οποίο είναι πράγματι πολύ κοντά στο $\sin 60° \approx 0.87$.

Παρακάτω βλέπουμε κάποιες συνήθεις αντιστοιχίσεις μοιρών σε ακτίνια:

$$30° = \frac{\pi}{6} \qquad 45° = \frac{\pi}{4} \qquad 60° = \frac{\pi}{3}$$
$$90° = \frac{\pi}{2} \qquad 180° = \pi \qquad 360° = 2\pi$$

Στο Unity, οι συναρτήσεις sin και cos καλούνται μέσω της βιβλιοθήκης Mathf ως Mathf.Sin και Mathf.Cos, αντίστοιχα. Εδώ θα πρέπει να προσέξουμε πως αυτές οι συναρτήσεις δέχονται ως είσοδο μόνο ακτίνια, οπότε αν θέλουμε να υπολογίσουμε το $\cos 45°$, **δεν** θα το κάνουμε γράφοντας:

```
1.  // συνημίτονο 45 ακτινίων!
2.  float cos45Deg = Mathf.Cos(45.0f);
```

Τα 45 ακτίνια είναι περίπου $2578°$. Μια πλήρης περιστροφή $360°$ είναι ισοδύναμη με μη περιστροφή. Αν διαιρέσουμε τις $2578°$ με $360°$ μας δίνει ένα υπόλοιπο $58°$, που είναι η ισοδύναμη γωνία $2578°$ και είναι κάτι τελείως διαφορετικό από τις $45°$.

Για να υπολογίσουμε το $cos45°$ θα το κάνουμε με αυτό τον τρόπο:

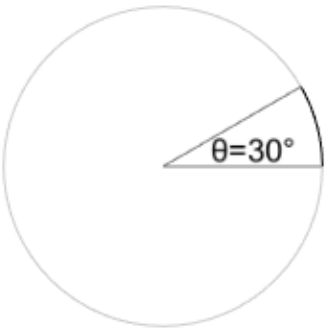
```
1.  // μετατροπή σε ακτίνια
2.  float cos45Deg = Mathf.Cos(45.0f * Mathf.PI / 180.0f);
```

Επίσης μπορούμε να χρησιμοποιήσουμε και σταθερές οι οποίες βοηθούν στην μετατροπή από μοίρες σε ακτίνια:

```
1.  float cos45Deg = Mathf.Cos(45.0f * Mathf.Deg2Rad);
```

Σε εργαλεία όπως ο επεξεργαστής Unity, η έκφραση γωνιών σε μοίρες είναι πιο φιλική προς το χρήστη, επειδή οι περισσότεροι άνθρωποι μπορούν αμέσως να φανταστούν μία γωνία 45°. Ωστόσο, όσον αφορά τα μαθηματικά και τον προγραμματισμό, πολλοί άνθρωποι, προτιμούν να χρησιμοποιούν ακτίνια.

Ένα χρήσιμο πράγμα για τα ακτίνια είναι ότι μηδενίζουν τον υπολογισμό του μήκους του τόξου με δεδομένα την ακτίνα και τη γωνία. Ας υποθέσουμε ότι θέλουμε να υπολογίσουμε το μήκος ενός τόξου 30°, ή $\frac{\pi}{6}$ ακτίνια, ενός κύκλου ακτίνας 2.



Αν υπολογιστεί χρησιμοποιώντας μοίρες, πρώτα υπολογίζεται ολόκληρη η περιφέρεια χρησιμοποιώντας τον τύπο $\boldsymbol{ακτίνια \cdot 2\pi}$, και στη συνέχεια πολλαπλασιάζεται με το κλάσμα 30° προς 360°:

$$arc = radius \times 2\pi \times \frac{30°}{360°}$$
$$= 2 \times 2\pi \times \frac{1}{12}$$
$$= \frac{\pi}{3}$$

Όταν χρησιμοποιούμε ακτίνια, ο τύπος μήκους τόξου είναι απλά ακτίνα επί γωνία σε ακτίνια:

$$arc = radius \times \frac{\pi}{6}$$
$$= 2 \times \frac{\pi}{6}$$
$$= \frac{\pi}{3}$$

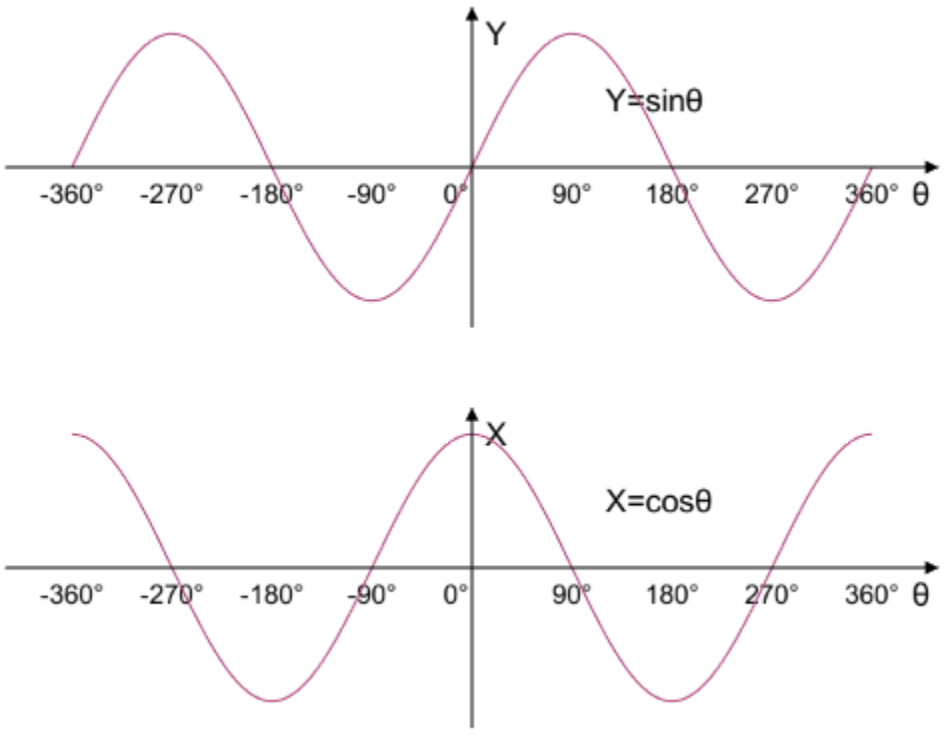
Ο τύπος περιφέρειας του κύκλου συμβαδίζει με τον τύπο μήκους τόξου σε ακτίνια. Δεδομένου ότι ένας πλήρης κύκλος είναι βασικά ένα τόξο με γωνία 2π ακτίνια, το μήκος αυτού του τόξου είναι $\boldsymbol{\alpha\kappa\tau\acute{\iota}\nu\iota\alpha \cdot 2\pi}$, ακριβώς το ίδιο με τον τύπο περιφέρειας του κύκλου.

Βασικές Ιδιότητες

Δεδομένου ότι το $(\cos\theta, \sin\theta)$ είναι συντεταγμένες ενός σημείου στον μοναδιαίο κύκλο, η απόσταση του σημείου από την αρχή των αξόνων είναι πάντα 1, ανεξάρτητα από την γωνία θ. Το Πυθαγόρειο θεώρημα δηλώνει ότι η απόσταση του σημείου (x, y) από την αρχή των αξόνων είναι $\sqrt{x^2 + y^2}$. Από εκεί μπορούμε να πάρουμε αυτήν την ταυτότητα (εξίσωση που είναι πάντα αληθής):

$$\sin^2\theta + \cos^2\theta = 1$$

Ας θυμηθούμε τη σύγκριση των γραφημάτων ημιτόνου και συνημίτονου.



Όπως φαίνεται στην εικόνα η καμπύλη συνημίτονου είναι βασικά η ημιτονοειδής καμπύλη που μετατοπίζεται προς τα αριστερά κατά $90°$, ή $\frac{\pi}{2}$ ακτίνια. Έτσι παίρνουμε αυτές τις δύο ταυτότητες που μας επιτρέπουν τη μετατροπή μεταξύ των δύο:

$$\sin\theta = \cos(\theta - \frac{\pi}{2})$$

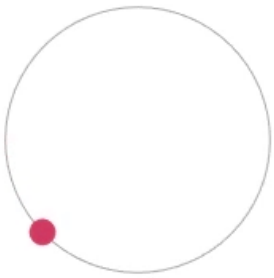
$$\cos\theta = \sin(\theta + \frac{\pi}{2})$$

Κίνηση σε Κύκλους και Σπείρες

Τώρα που έχουμε δει ότι $(\cos\theta, \sin\theta)$ είναι δισδιάστατες συντεταγμένες ενός σημείου στο μοναδιαίο κύκλο, μπορούμε να αρχίσουμε να παίζουμε με κάποια βασική κυκλική κίνηση στο Unity.

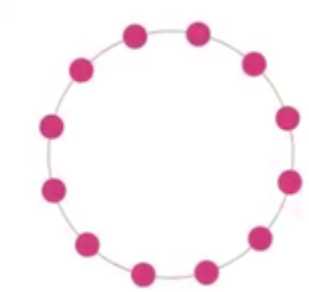
Ο παρακάτω κώδικας μετακινεί ένα αντικείμενο γύρω από έναν κύκλο με σταθερό ρυθμό:

```
obj.transform.position =
  new Vector3
  (
    Radius * Mathf.Cos(Rate * Time.time),
    Radius * Mathf.Sin(Rate * Time.time),
    0.0f
  );
```



Ο παρακάτω κώδικας μετακινεί 12 αντικείμενα γύρω από έναν κύκλο με σταθερό ρυθμό και τα αντικείμενα απέχουν εξίσου έξω από τον κύκλο:

```
float baseAngle = Rate * Time.time + angleOffset;
for (int i = 0; i < 12; ++i)
{
  float angleOffset = 2.0f * Mathf.PI * i / 12.0f;
  aObj[i].transform.position =
    new Vector3
    (
      Radius * Mathf.Cos(baseAngle + angleOffset),
      Radius * Mathf.Sin(baseAngle + angleOffset),
      0.0f
    );
}
```



Συνδυάζοντας κυκλική κίνηση με κίνηση στην κατεύθυνση z, μπορούμε να δημιουργήσουμε μια σπειροειδή κίνηση σε 3D:

```
obj.transform.position =
  new Vector3
  (
    Radius * Mathf.Cos(Rate * Time.time),
    Radius * Mathf.Sin(Rate * Time.time),
    ZSpeed * Time.time
  );
```

Απλή (Γραμμική) Αρμονική Ταλάντωση

Τι γίνεται αν βάλουμε το συνημίτονο στη συντεταγμένη x ενός αντικειμένου;

```
float x = Mathf.Cos(Rate * Time.time);
obj.transform.position = Vector3(x, 0.0f, 0.0f);
```

Το αντικείμενο κινείται προς τα δεξιά και προς τα αριστερά εναλλάξ. Στις άκρες της κίνησης του η ταχύτητά του εξασθενεί ενώ στο κέντρο φτάνει τη μέγιστη.

Η αυτού του είδους ταλαντευόμενη κίνηση, η οποία ταιριάζει με την ημιτονοειδή καμπύλη (ημιτονοειδές κύμα), ονομάζεται Απλή Αρμονική Ταλάντωση (Simple Harmonic Motion).

Δεδομένου ότι το θ ξεκινάει από το μηδέν, η συντεταγμένη x του αντικειμένου ξεκινάει από το $\cos 0 = 1$. Αν χρησιμοποιήσουμε τη συνάρτηση $\sin\theta$, η συντεταγμένη x θα αρχίσει από το $\sin 0 = 0$.

Η γωνία εισόδου που διέρχεται στις ημιτονοειδείς και συναρτήσεις συνημίτονου ονομάζεται **φάση**. Συνήθως, αν η φάση που δηλώνεται είναι σταθερό πολλαπλάσιο του χρόνου, πολλοί το γράφουν ως $\sin\omega t$, όπου ω ονομάζεται η γωνιακή συχνότητα (σε ακτίνια ανά δευτερόλεπτο) και t είναι ο χρόνος. Για παράδειγμα, το $\sin 2\pi t$ θα παράγει μια απλή αρμονική κίνηση που ταλαντώνει έναν πλήρη κύκλο κάθε δευτερόλεπτο.

Τι θα συμβεί αν μειώσουμε αυτή την κίνηση με έναν εκθετικά μειούμενο παράγοντα;

```
1.  float s = Mathf.Pow(0.5f, Decay * Time.time);
2.  float x = Mathf.Cos(Rate * Time.time);
3.  obj.transform.position = Vector3(s * x, 0.0f, 0.0f);
```

Πλέον το αντικείμενο κινείται σε μια μειούμενη ταλάντωση.

Η Κίνηση του Εκκρεμούς

Αντί να συνδέσουμε ένα ημίτονο σε μια συντεταγμένη x ενός αντικειμένου, τι γίνεται αν το συνδέσουμε στη γωνία για το παραπάνω παράδειγμα;

```
1.  float baseAngle = 1.5f * Mathf.PI; // 270 degrees
2.  float halfAngleRange = 0.25f * mathf.PI; // 45 degrees
3.  float c = Mathf.Cos(Rate * Time.time);
4.  float angle = halfAngleRange * c + baseAngle;
5.  obj.transform.position =
6.    new Vector3
7.    (
8.      Radius * Mathf.Cos(angle),
9.      Radius * Mathf.Sin(angle),
10.     0.0f
11.   );
```

a *b* *c*

Μπορούμε να πούμε πως η κίνηση αυτή θεωρείται η γωνία της κυκλικής κίνησης η οποία βρίσκεται σε μια απλή αρμονική κίνηση.

Αιωρούμενη Κίνηση

Μπορούμε να εφαρμόσουμε αρμονικές κινήσεις στις συντεταγμένες x, y και z ξεχωριστά.

```
Vector3 hover =
  new Vector3
  (
    RadiusX * Mathf.Sin(RateX * Time.time + OffsetX),
    RadiusY * Mathf.Sin(RateY * Time.time + OffsetY),
    RadiusZ * Mathf.Sin(RateZ * Time.time + OffsetZ)
  );

obj.transform.position = basePosition + hover;
```

Πλέον το αντικείμενο φαίνεται να αιωρείται.

Πηγαίνοντας ένα βήμα παραπέρα, το offset της αιώρησης μπορεί να χρησιμοποιηθεί για τον υπολογισμό της κλίσης της περιστροφής, για ένα ολοκληρωμένο αποτέλεσμα.

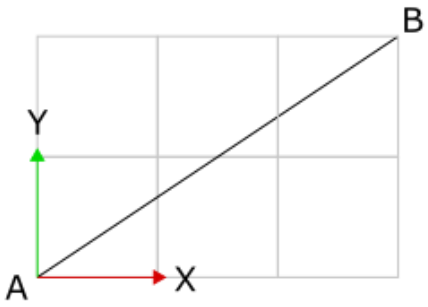
```
obj.transform.rotation =
  baseRotation
  * Quaternion.FromToRotation
    (
      Vector3.up,
      -hover + 3.0f * Vector3.up
    );
```

Γεωμετρική Ερμηνεία της Εφαπτομένης

Ας υποθέσουμε πως έχουμε ένα μοναδιαίο κύκλο, με ένα σημείο P σε αυτόν, καθώς και μία γωνία θ μεταξύ του άξονα x (θετική κατεύθυνση) και του τμήματος γραμμής που σχηματίζεται από το P και το κέντρο του κύκλου.

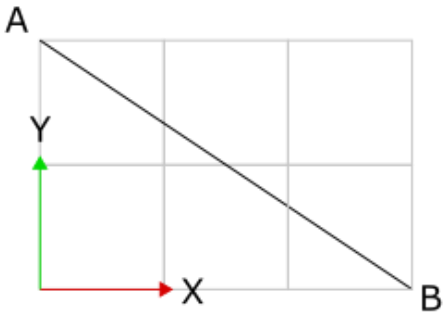
Οι συντεταγμένες (x, y) του P είναι $(\cos\theta, \sin\theta)$. Η εφαπτομένη της θ $(\tan\theta)$ είναι η κλίση του τμήματος της γραμμής μεταξύ του P και της αρχής των αξόνων.

Η κλίση μιας γραμμής είναι ο λόγος της κάθετης μεταβολής έναντι της οριζόντιας μεταβολής. Για παράδειγμα, ας δούμε αυτό το τμήμα γραμμής:



Για να μετακινηθούμε από το σημείο A στο σημείο B, περπατάμε 3 μονάδες προς την κατεύθυνση $+x$ και στη συνέχεια 2 μονάδες στην κατεύθυνση $+y$, έτσι η κλίση της γραμμής είναι $\frac{2}{3}$.

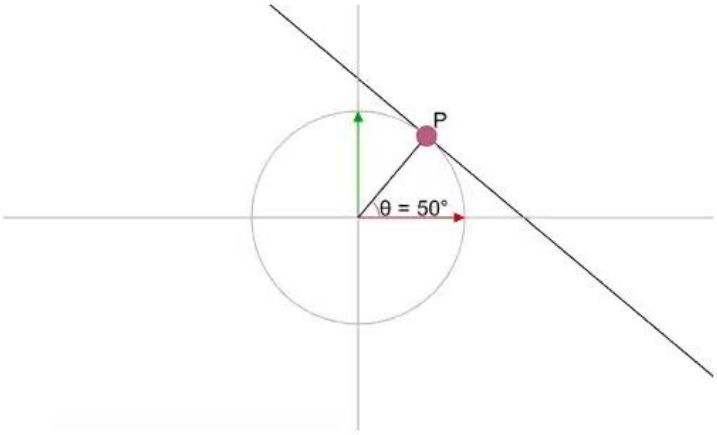
Και για ένα τμήμα γραμμής που πηγαίνει "κατηφορικά":



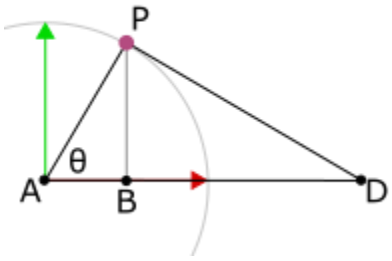
Η κλίση θα είναι $\frac{-2}{3}$, αρνητική τιμή, αφού η κάθετη μεταβολή έναντι της οριζόντιας μεταβολής είναι αρνητική.

Βλέπουμε ότι η μετάβαση από την αρχή των αξόνων στο σημείο P συνεπάγεται οριζόντια μεταβολή του $\cos\theta$ και κάθετη μεταβολή της $\sin\theta$. Έτσι η κλίση του τμήματος της γραμμής μεταξύ της αρχής των αξόνων και του σημείου P είναι $\frac{\sin\theta}{\cos\theta}$, επομένως $\tan\theta = \frac{\sin\theta}{\cos\theta}$.

Αλλά αυτό είναι απλώς μια μαθηματική έκφραση. Εδώ μπορούμε να δούμε πού το $\tan\theta$ ταιριάζει στο σχήμα ενός μοναδιαίου κύκλου. Ας σχεδιάσουμε μια εφαπτομένη γραμμή στον κύκλο στο σημείο P, δηλαδή μια γραμμή που περνάει από το σημείο P και είναι κάθετη στο τμήμα της γραμμής μεταξύ του σημείου P και του κέντρου του κύκλου:

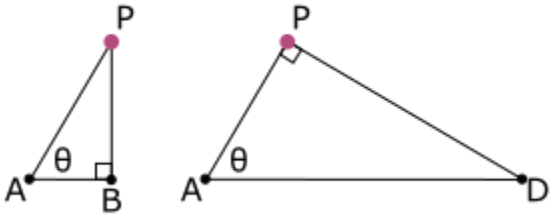


Ας δούμε μόνο το τμήμα αυτής της εφαπτόμενης γραμμής που βρίσκεται μεταξύ του P και του άξονα x και ας σημειώσουμε κάποια από τα σημεία:



Οι γωνίες $\measuredangle ABP$ και $\measuredangle APD$ είναι ορθές γωνίες. Και $\measuredangle PAB = \measuredangle PAD = \theta$. Το $\overline{AB}$ υποδηλώνει το μήκος του τμήματος της γραμμής μεταξύ του σημείου A και του σημείου B.

Τώρα, ας χωρίσουμε το σχήμα σε δύο τρίγωνα:



Δεδομένου ότι όλες οι εσωτερικές γωνίες ενός τριγώνου ανέρχονται σε $180°$ και τα δύο τρίγωνα έχουν μία γωνία θ και μία $90°$, οι γωνίες που δεν έχουν επισημανθεί στα τρίγωνα, η $\measuredangle APB$ και η $\measuredangle ADP$ είναι ακριβώς ίδια: $180° - \theta - 90°$.

Εάν δύο τρίγωνα έχουν πανομοιότυπα σύνολα γωνιών, τότε είναι όμοια, δηλαδή εάν αναλογικώς μεγαλώσουμε/μικρύνουμε, περιστρέψουμε και/ή αναποδογυρίσουμε ένα από αυτά, μπορεί να γίνει πανομοιότυπο με το άλλο.

Όταν δύο τρίγωνα είναι όμοια, ο λόγος μεταξύ των μηκών δύο πλευρών του ενός τριγώνου ισούται με τον λόγο μεταξύ των μηκών των αντίστοιχων πλευρών του άλλου τριγώνου. Έτσι:

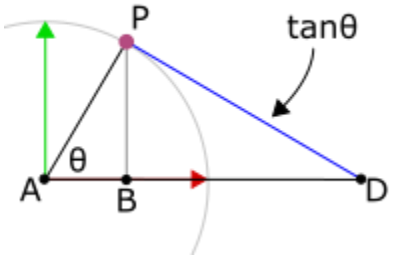
$$\frac{\overline{BP}}{\overline{AB}} = \frac{\overline{DP}}{\overline{AP}}$$

Γνωρίζουμε ότι οι συντεταγμένες του σημείου P είναι $(\cos\theta\,, \sin\theta)$, έτσι $\overline{AB} = \cos\theta$ και $\overline{BP} = \sin\theta$. Και γνωρίζουμε ότι το $\overline{AP}$ είναι ίσο με την ακτίνα του μοναδιαίου κύκλου, έτσι $\overline{AP} = 1$. Η παραπάνω εξίσωση γίνεται:

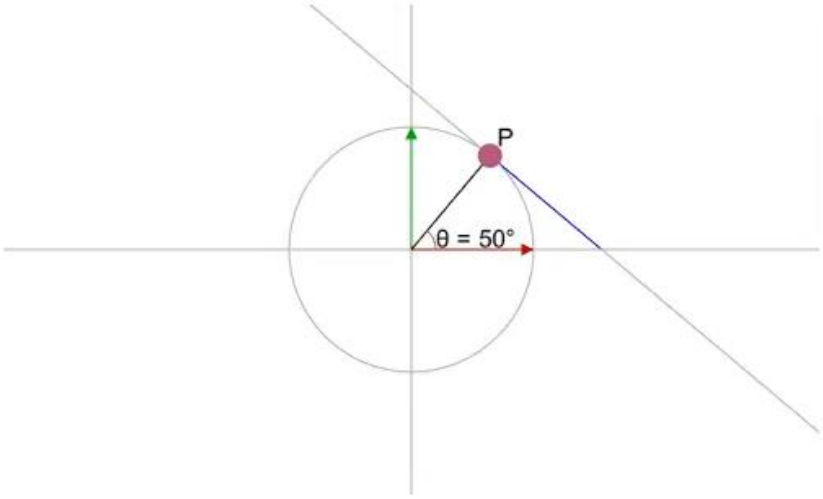
$$\frac{\sin\theta}{\cos\theta} = \frac{\overline{DP}}{1}$$

Και γνωρίζοντας ότι $\tan\theta = \frac{\sin\theta}{\cos\theta}$, έχουμε:

$$\tan\theta = \overline{DP}$$

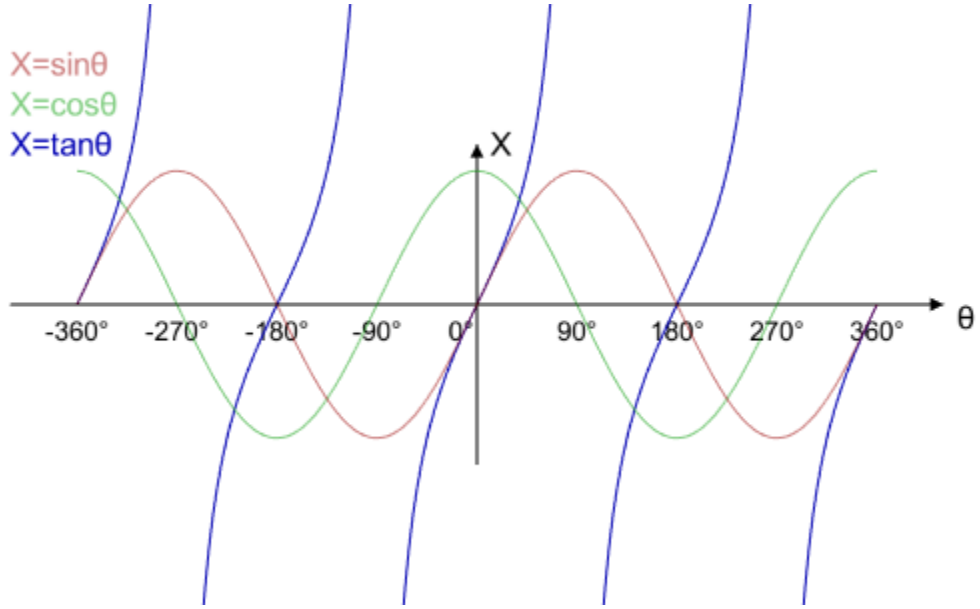


Η απόλυτη τιμή του $\tan\theta$ είναι το μήκος του τμήματος της εφαπτομένης γραμμής μεταξύ του σημείου P και του άξονα x. Απόλυτη τιμή, διότι ανάλογα με τα πρόσημα των $\sin\theta$ και $\cos\theta$, η $\tan\theta$ μπορεί να είναι θετική ή αρνητική. Στην εικόνα επισημαίνεται (με μπλε χρώμα) το τμήμα γραμμής του οποίου το μήκος είναι ίσο με την απόλυτη τιμή του $\tan\theta$:



Η Καμπύλη της Εφαπτομένης

Η γραφική παράσταση της $\tan\theta$ έχει ως εξής:



Παρατηρήστε πως, αντίθετα με τα $\sin\theta$ και $\cos\theta$, η τιμή του $\tan\theta$ δεν περιορίζεται εντός του εύρους $[-1,1]$. Από την $\tan\theta = \frac{\sin\theta}{\cos\theta}$, η απόλυτη τιμή της $\tan\theta$ προσεγγίζει το άπειρο καθώς το $\cos\theta$ προσεγγίζει το μηδέν. Επίσης, αντίθετα από τα $\sin\theta$ και $\cos\theta$, η περίοδος του $\tan\theta$ είναι π, αντί για 2π.

Κάτι άλλο που αξίζει να σημειώσουμε είναι οι σχέσεις μεταξύ των πρόσημων των τριών βασικών τριγωνομετρικών συναρτήσεων. Από την $\tan\theta = \frac{\sin\theta}{\cos\theta}$, το πρόσημο της $\tan\theta$ είναι θετικό όταν το $\sin\theta$ και $\cos\theta$ έχουν το ίδιο πρόσημο.

Με αυτό τον τρόπο μπορούμε να συνδέσουμε την εφαπτομένη καμπύλη κατά την πάροδο του χρόνου στη συντεταγμένη x ενός αντικειμένου:

```
1.  float tan = Mathf.Tan(Rate * Time.time);
2.  obj.transform.position = Vector3(tan, 0.0f, 0.0f);
```

Το αντικείμενο εισέρχεται γρήγορα από την κατεύθυνση $-x$, επιβραδύνει, και στη συνέχεια επιταχύνει πάλι προς την κατεύθυνση $+x$.

Μπορούμε να χρησιμοποιήσουμε αυτήν την κίνηση για να δημιουργήσουμε εφέ όπως για παράδειγμα διάττοντες αστέρες:

```
1.  float tan = Mathf.Tan(Rate * Time.time);
2.  obj.transform.position = center + moveDirection * tan;
```

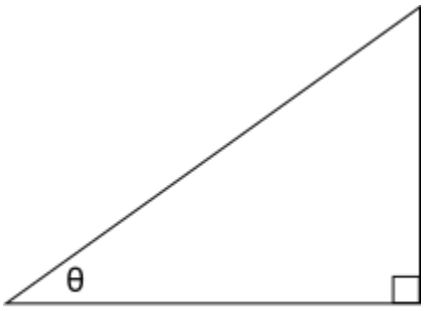
Η επιτάχυνση και η επιβράδυνση είναι διακριτική. Μπορούμε όμως να ενισχύσουμε περαιτέρω το αποτέλεσμα υψώνοντας την εφαπτομένη συνάρτηση σε μια δύναμη:

```
1.  float tan = Mathf.Tan(Rate * Time.time);
2.  float tan3 = tan * tan * tan;
3.  obj.transform.position = center + moveDirection * tan3;
```

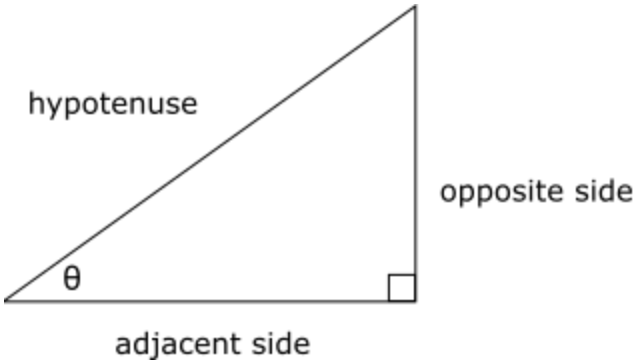
Τριγωνομετρικές συναρτήσεις, γωνίες και τρίγωνα

Έχουμε δει πώς οι τρεις βασικές τριγωνομετρικές συναρτήσεις σχετίζονται με τον μοναδιαίο κύκλο. Τώρα θα δούμε τη σχέση τους με τα τρίγωνα. Εξ' ου το όνομα "τριγωνομετρικές". Συγκεκριμένα, θα εργαστούμε με τα ορθογώνια τρίγωνα.

Παρακάτω βλέπουμε ένα ορθογώνιο τρίγωνο και μια γωνία θ:



Η πλευρά του τριγώνου μεταξύ της θ και της ορθής γωνίας ονομάζεται προσκείμενη κάθετη πλευρά, αφού είναι δίπλα στη θ. Η άλλη πλευρά δίπλα στην ορθή γωνία ονομάζεται απέναντι κάθετη πλευρά, επειδή είναι απέναντι από τη θ. Η τρίτη (και μεγαλύτερη) πλευρά που βρίσκεται απέναντι από την ορθή γωνία ονομάζεται υποτείνουσα:



Και εδώ μπορούμε να δούμε τη σχέση των τριών βασικών τριγωνομετρικών συναρτήσεων με τα μήκη των πλευρών του τριγώνου:

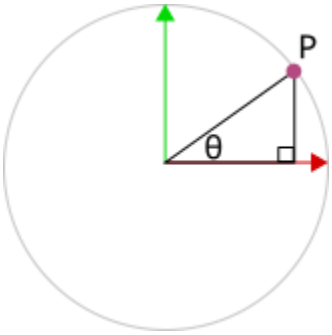
- $\sin\theta$ = το μήκος της απέναντι πλευράς προς το μήκος της υποτείνουσας.
- $\cos\theta$ = το μήκος της προσκείμενης πλευράς προς το μήκος της υποτείνουσας.
- $\tan\theta$ = το μήκος της απέναντι πλευράς προς το μήκος της προσκείμενης πλευράς.

Δηλαδή:

$$\sin\theta = \frac{απέναντι\ πλευρά}{υποτείνουσα} \quad \cos\theta = \frac{προσκείμενη\ πλευρά}{υποτείνουσα} \quad \tan\theta = \frac{απέναντι\ πλευρά}{προσκείμενη\ πλευρά}$$

Όποιο και αν είναι το μέγεθος του ορθού τριγώνου, οι παραπάνω εξισώσεις είναι πάντα αληθείς, επειδή οι αναλογίες μεταξύ των δύο πλευρών είναι ανεξάρτητες από τα απόλυτα μήκη των μεμονωμένων πλευρών.

Αν διαμορφώσουμε την κλίμακα του τριγώνου έτσι ώστε η υποτείνουσα να έχει μήκος 1, τότε μπορούμε να το τοποθετήσουμε στο σχήμα του μοναδιαίου κύκλου, με (x, y) να είναι οι συντεταγμένες του σημείου P στον κύκλο:

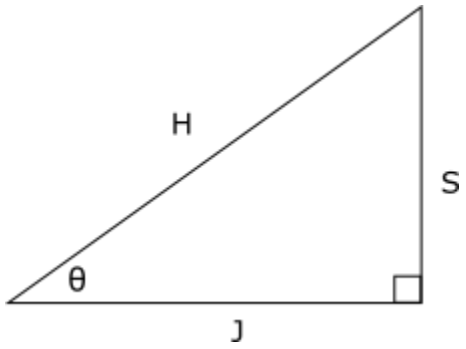


Και οι παραπάνω εξισώσεις δένουν με τις συντεταγμένες του P, $(x, y) = (\cos\theta\,, \sin\theta)$:

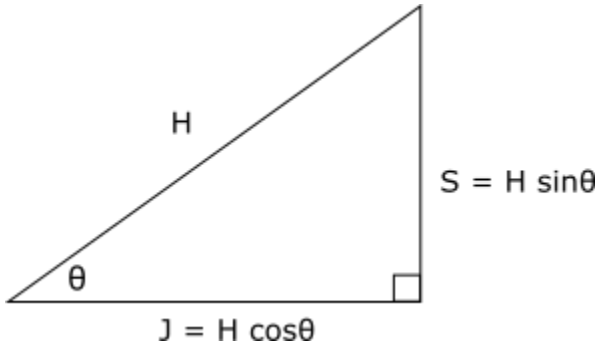
$$\sin\theta = \frac{y}{1} = Y \quad \cos\theta = \frac{x}{1} = X \quad \tan\theta = \frac{y}{x} = \frac{\sin\theta}{\cos\theta}$$

Γνωρίζοντας τις εξισώσεις για τις τριγωνομετρικές συναρτήσεις όσον αφορά τα μήκη των κάθετων πλευρών του τριγώνου, για οποιοδήποτε δεδομένο ορθογώνιο τρίγωνο με γωνία θ, αν γνωρίζουμε το μήκος οποιασδήποτε πλευράς, μπορούμε να αντλήσουμε τα μήκη των άλλων δύο πλευρών χρησιμοποιώντας τις τρεις βασικές τριγωνομετρικές συναρτήσεις.

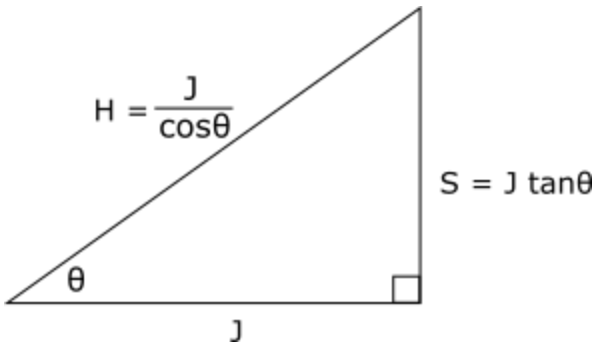
Έστω J το μήκος της προσκείμενης πλευράς, S το μήκος της απέναντι πλευράς και το H το μήκος της υποτείνουσας:



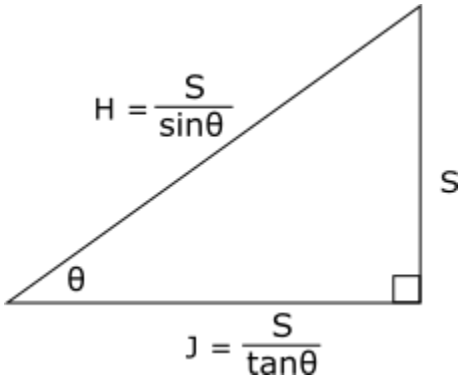
Αν γνωρίζουμε το μήκος της υποτείνουσας (H), τότε $J = H\cos\theta$ και $S = H\sin\theta$:



Αν γνωρίζουμε το μήκος της προσκείμενης πλευράς (J), τότε $S = J\tan\theta$ και $H = \frac{J}{\cos\theta}$:



Αν γνωρίζουμε το μήκος της απέναντι πλευράς (S), τότε $J = \frac{S}{\tan\theta}$, και $H = \frac{S}{\sin\theta}$:



Προσομοίωση κανονιών και πρόβλεψη τροχιών

Ας δούμε πώς μπορούμε να προσομοιώσουμε ένα κανόνι όταν μας δίνεται μια αρχική ταχύτητα, μια οριζόντια γωνία και μια γωνία ύψους. Επίσης, θα δούμε πώς μπορούμε να εμφανίσουμε τις προβλεπόμενες τροχιές ακόμη και πριν πυροδοτήσουμε το κανόνι.

Κάποιες βασικές ορολογίες της δυναμικής κίνησης, που θα χρησιμοποιήσουμε παρακάτω είναι οι εξής. Η **θέση** ενός αντικειμένου είναι εκεί όπου το αντικείμενο βρίσκεται φυσικά. Η **ταχύτητα** ενός αντικειμένου είναι ο ρυθμός αλλαγής στη θέση του (τυπικά εκφραζόμενο ως αλλαγή θέσης ανά δευτερόλεπτο). Η **επιτάχυνση** ενός αντικειμένου είναι ο ρυθμός μεταβολής της ταχύτητάς του (συνήθως εκφράζεται ως μεταβολή της ταχύτητας ανά δευτερόλεπτο).

Η μέθοδος *Euler* είναι ένας γρήγορος και εύκολος αλγόριθμος για την προσομοίωση της κίνησης αντικειμένων: Για κάθε κινούμενο αντικείμενο, αποθηκεύουμε το διάνυσμα ταχύτητας και τη θέση του. Για κάθε ενημέρωση ή βήμα χρόνου, αλλάζουμε την ταχύτητα κατά επιτάχυνση επί το χρόνο δέλτα (η χρονική διαφορά μεταξύ κάθε ενημέρωσης) και στη συνέχεια αλλάζουμε τη θέση κατά ταχύτητα επί το χρόνο δέλτα:

```
1.  velocity += acceleration * deltaTime;
2.  position += velocity * deltaTime;
```

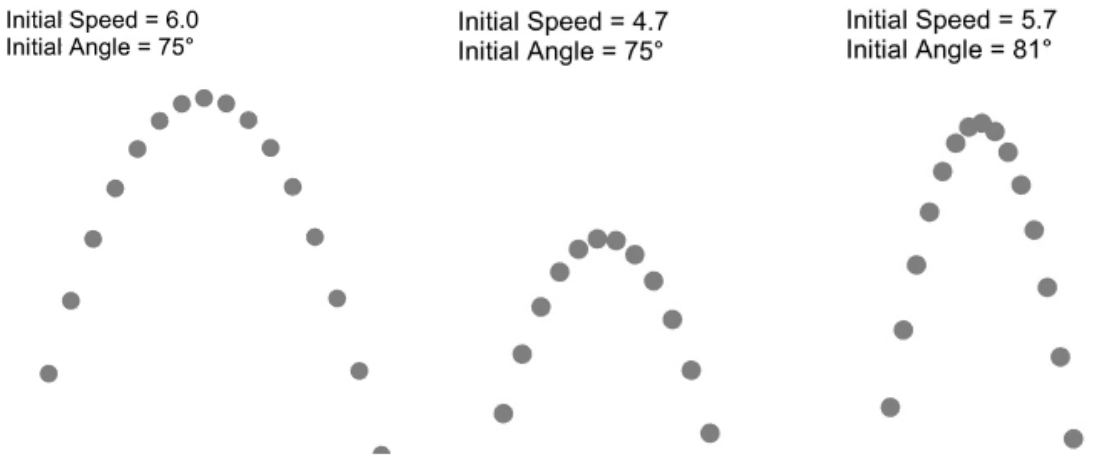
Για να προσομοιώσουμε τη βαρύτητα στο επίπεδο του εδάφους, αφήνουμε την επιτάχυνση ως ένα σταθερό προς τα κάτω διάνυσμα.

Αν προσομοιώσουμε ολόκληρη την τροχιά μέσα σε ένα καρέ με την εκτέλεση πολλαπλών βημάτων χρόνου και σημειώσουμε μια μικρή κουκκίδα μία φορά κάθε x αριθμού επαναλήψεων μπορούμε να φτιάξουμε έναν καλό πλάνο της προβλεπόμενης τροχιάς:

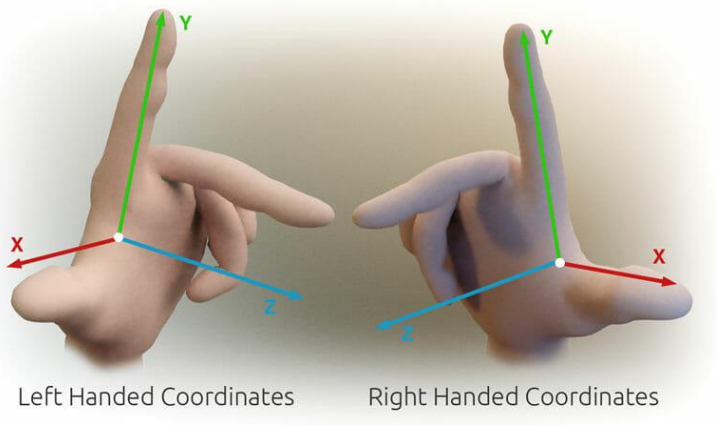
```
velocity = initialVelocity;
position = initialPosition;
for (int i = 0; i < NumIterations; ++i)
{
  velocity += acceleration * deltaTime;
  position += velocity * deltaTime;

  if (i % IterationsPerDot != 0)
    continue;

  DrawDot(position);
}
```

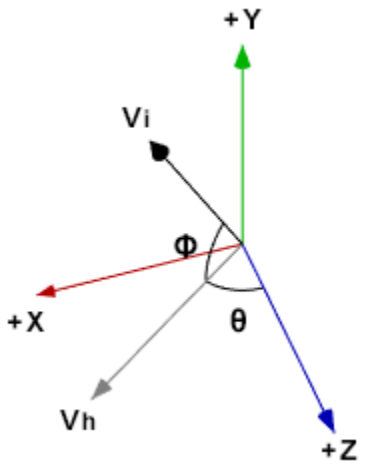


Τώρα, ας υπολογίσουμε την αρχική ταχύτητα (velocity) ενός βλήματος αν έχει εκτοξευθεί από το κανόνι σε μια αρχική ταχύτητα (speed) K (μήκος του διανύσματος αρχικής ταχύτητας), μια οριζόντια γωνία θ και μια γωνία ανύψωσης φ. Ας υποθέσουμε πως ο άξονας $+z$ είναι η κατεύθυνση του κανονιού προς τα εμπρός και ο άξονας $+x$ είναι η προς τα δεξιά του κατεύθυνση (το Unity χρησιμοποιεί αριστερόχειρο σύστημα συντεταγμένων).



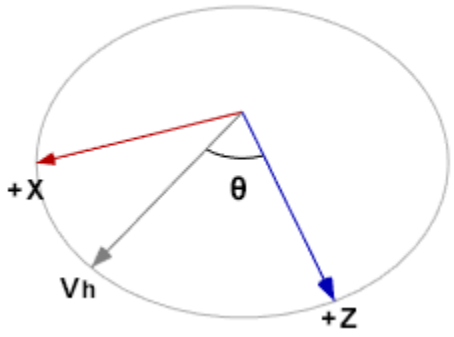
Για να υπολογίσουμε την αρχική ταχύτητα (velocity), πρέπει πρώτα να υπολογίσουμε ένα μοναδιαίο διάνυσμα προς την ίδια κατεύθυνση. Μόλις έχουμε αυτό το διάνυσμα, μπορούμε απλά να πολλαπλασιάσουμε όλα τα στοιχεία του με την επιθυμητή ταχύτητα (speed) K για να πάρουμε τον διάνυσμα της αρχικής ταχύτητας.

Το διάγραμμα παρακάτω δείχνει ένα μοναδιαίο διάνυσμα στον άξονα $+x$ με κόκκινο χρώμα, ένα μοναδιαίο διάνυσμα στον άξονα $+y$ με πράσινο χρώμα, ένα μοναδιαίο διάνυσμα στον άξονα $+z$ με μπλε χρώμα, ένα μοναδιαίο διάνυσμα στον άξονα της αρχικής ταχύτητας (velocity) με μαύρο χρώμα (με όνομα V_i), ένα μοναδιαίο διάνυσμα στην οριζόντια κατεύθυνση της αρχικής ταχύτητας (velocity) με γκρι χρώμα (με ένδειξη V_h), την οριζόντια γωνία θ (μεταξύ $+z$ και V_h), και τη γωνία ανύψωσης φ (μεταξύ V_h και V_i):

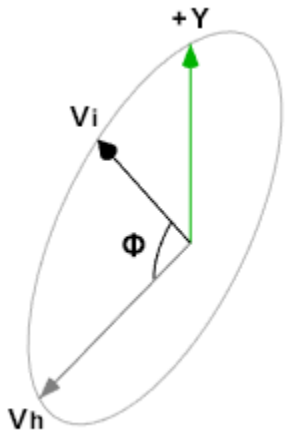


Ο στόχος είναι να βρούμε το V_i και να το πολλαπλασιάσουμε με το K. Μπορούμε να απομονώσουμε τα μοναδιαία διανύσματα και τις γωνίες από το διάγραμμα παραπάνω σε δύο διαγράμματα μοναδιαίων κύκλων.

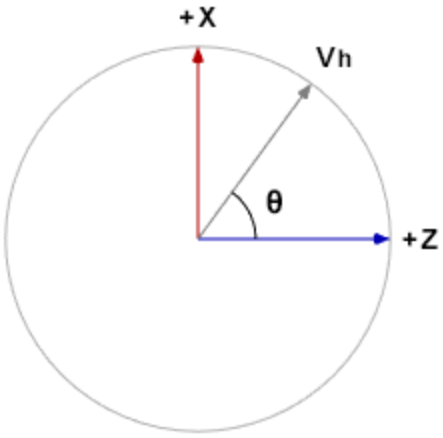
Το ένα είναι ένα οριζόντιο διάγραμμα μοναδιαίου κύκλου με $+x$, $+z$, V_h και θ:



Και το άλλο είναι ένα κάθετο διάγραμμα κύκλου με $+y$, V_h, V_i και φ:

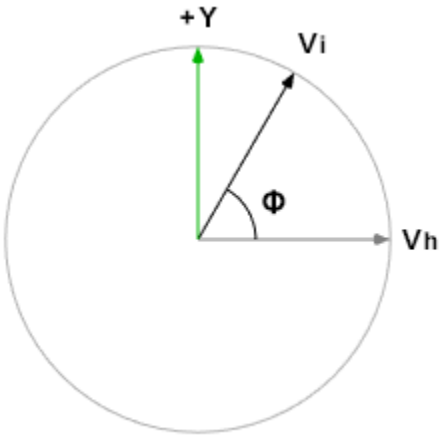


Αν δούμε το πρώτο (οριζόντιο) διάγραμμα κύκλου από διαφορετική γωνία, έχουμε τη γνωστή άποψη ενός επίπεδου κύκλου:



Όπως και προηγουμένως, η συνιστώσα του V_h κατά την κατεύθυνση $+z$ είναι μήκους $\cos\theta$, και η συνιστώσα κατά την κατεύθυνση $+x$ είναι μήκους $\sin\theta$. Έτσι έχουμε $V_h = (\sin\theta\,, 0, \cos\theta)$.

Με τον ίδιο τρόπο, το δεύτερο (κάθετο) διάγραμμα κύκλου από διαφορετική γωνία:



Η συνιστώσα του V_i κατά την κατεύθυνση V_h είναι μήκους $\cos\varphi$ και η συνιστώσα κατά την κατεύθυνση $+y$ είναι μήκους $\sin$ φ, έτσι μπορούμε τώρα να υπολογίσουμε το V_i:

$$V_i = \cos\varphi \cdot V_h + \sin\varphi \cdot (0,1,0) = (\cos\varphi \sin\theta\,, \sin\varphi\,, \cos\varphi\cos\theta)$$

Πολλαπλασιάζοντας το V_i με το K μας δίνει το διάνυσμα της αρχικής ταχύτητας:

$$V_i = K \cdot (\cos\varphi \sin\theta\,, \sin\varphi\,, \cos\varphi\cos\theta) = (K\cos\varphi \sin\theta\,, K\sin\varphi\,, K\cos\varphi\cos\theta)$$

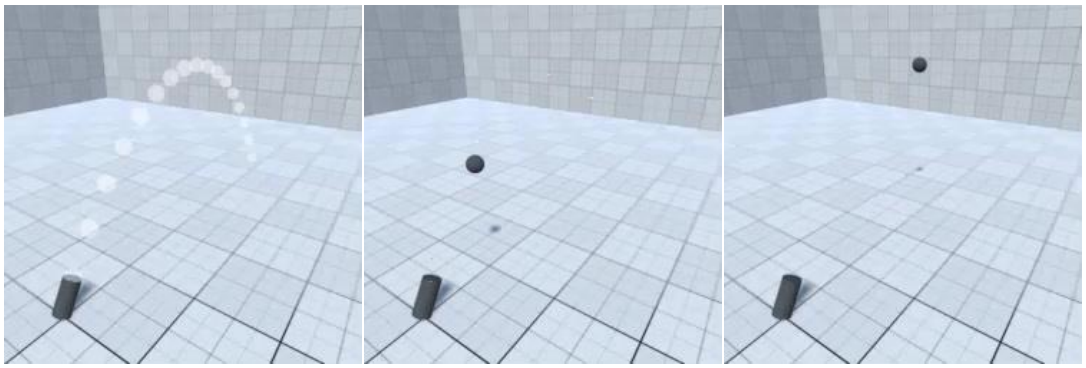
Και ο ανάλογος κώδικας:

```
Vector3 ComputeInitialVelocity()
{
  float sinTheta = Mathf.Sin(HorizontalAngle);
  float cosTheta = Mathf.Cos(HorizontalAngle);
  float sinPhi = Mathf.Sin(ElevationAngle);
  float cosPhi = Mathf.Cos(ElevationAngle);

  return
    InitialSpeed
    * new Vector3
      (
        cosPhi * sinTheta,
        sinPhi,
        cosPhi * cosTheta
      );
}
```

Το να μπορούμε να υπολογίσουμε το διάνυσμα αρχικής ταχύτητας από μια δεδομένη αρχική ταχύτητα, μια οριζόντια γωνία και μια γωνία ανύψωσης, έχουμε ό,τι χρειαζόμαστε για να προσομοιώσουμε ένα κανόνι:

```
1.  void FireCannon()
2.  {
3.    velocity = ComputeInitialVelocity();
4.    obj.transform.position = InitialPosition;
5.  }
6.
7.  void Update()
8.  {
9.    float dt = Time.deltaTime;
10.   velocity += acceleration * dt;
11.   obj.transform.position += velocity * dt;
12. }
13.
14. void DrawTrajectory()
15. {
16.   float dt = Time.fixedDeltaTime;
17.   Vector3 velocity = ComputeInitialVelocity();
18.   Vector3 position = InitialPosition;
19.   for (int i = 0; i < NumIterations; ++i)
20.   {
21.     velocity += acceleration * dt;
22.     position += velocity * dt;
23.
24.     if (i % IterationsPerDot != 0)
25.       continue;
26.
27.     DrawDot(position);
28. }
```

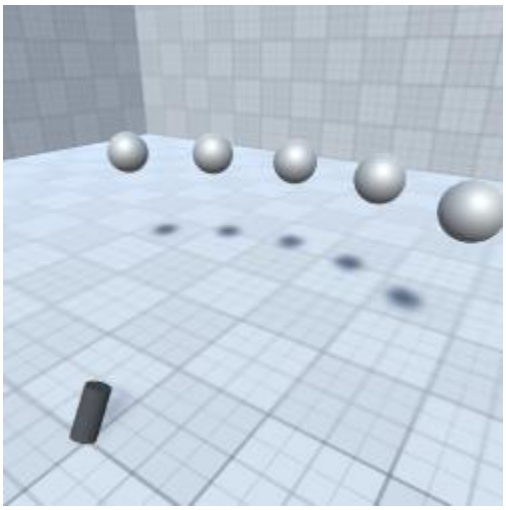


Τοποθέτηση Στόχων

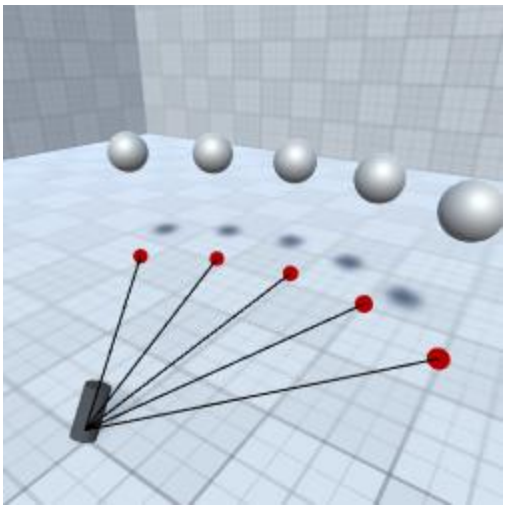
Τώρα που μπορούμε να πυροβολήσουμε, μπορούμε να τοποθετήσουμε μερικούς στόχους. Αν θέλουμε να τοποθετήσουμε έναν στόχο σε μια δεδομένη οριζόντια απόσταση μακριά από το

κανόνι, καθώς και σε μια δεδομένη γωνία ανύψωσης, πού ακριβώς πρέπει να τοποθετήσουμε τους στόχους;

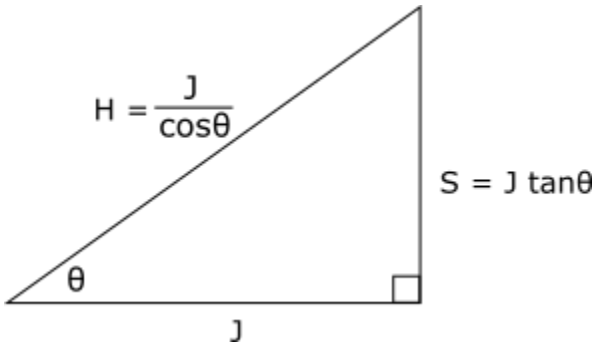
Παρακάτω είναι το επιθυμητό τελικό αποτέλεσμα. Κάθε στόχος βρίσκεται σε σταθερή οριζόντια απόσταση (στο επίπεδο xz) από το κανόνι και σε σταθερή γωνία ανύψωσης πάνω από το έδαφος. Οι στόχοι έχουν επίσης ίση οριζόντια απόσταση μεταξύ τους, δηλαδή οι οριζόντιες γωνίες τους σε σχέση με το κανόνι είναι ίσες.



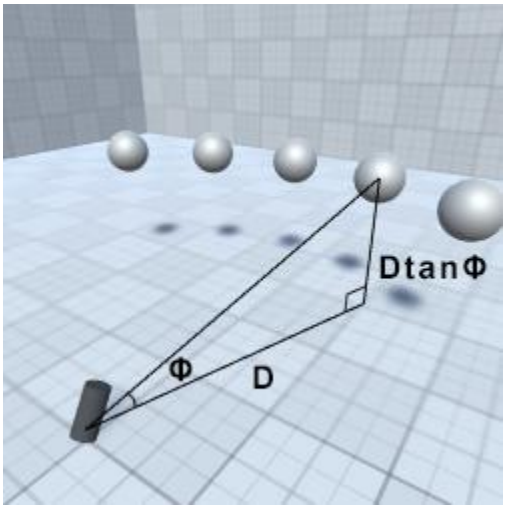
Γνωρίζουμε ήδη πώς να υπολογίσουμε ένα μοναδιαίο οριζόντιο διάνυσμα από μια οριζόντια γωνία θ. Το διάνυσμα αυτό είναι $(\cos\theta\,, 0, \sin\theta)$. Πολλαπλασιάζοντας αυτό το διάνυσμα με μια δεδομένη οριζόντια απόσταση D, παίρνουμε το διάνυσμα οριζόντιας μετατόπισης (offset) του στόχου από το κανόνι: $(D\cos\theta\,, 0, D\sin\theta)$. Έτσι, απλώνοντας διάφορες γωνίες θ και υπολογίζοντας τα διανύσματα οριζόντιας μετατόπισης για κάθε τιμή θ παίρνουμε τις συντεταγμένες xz των στόχων (που εμφανίζονται ως κόκκινες κουκίδες στην παρακάτω εικόνα):



Το τελευταίο βήμα είναι να προσδιορίσουμε τις συντεταγμένες y των στόχων, δηλαδή πόσο πάνω από το έδαδος θα πρέπει να είναι. Όπως είπαμε, αν γνωρίζουμε το μήκος της προσκείμενης πλευράς J μιας γωνίας θ ενός ορθογωνίου τριγώνου, τότε το μήκος της απέναντι πλευράς είναι $J \tan \theta$.



Αντικαθιστώντας το J με την δεδομένη οριζόντια απόσταση D και το θ με την γωνία ανύψωσης φ, ο τύπος για τη συντεταγμένη y των στόχων γίνεται $D \tan \varphi$.



Με τον παρακάτω κώδικα εν τέλη μπορούμε να τοποθετήσουμε τους στόχους στα επιθυμητά σημεία:

```
float theta = -0.5f * AngleInterval * (NumTargets - 1);
float elevationTan = Mathf.Tan(ElevationAngle);

foreach (var target in targetArray)
{
  Vector3 horizontalVec =
    HorizontalDistance
    * new Vector3
      (
        Mathf.Sin(theta),
        0.0f,
        Mathf.Cos(theta)
      );

  theta += AngleInterval;

  Vector3 verticalVec =
    HorizontalDistance
    * elevationTan
    * Vector3.up;

  target.transform.position =
    Cannon.position
    + horizontalVec
    + verticalVec;
}
```

Αντίστροφες Συναρτήσεις

Μια συνάρτηση είναι σαν ένα black box το οποίο παίρνει κάποια είσοδο και μας δίνει κάποια έξοδο. Αν μια συνάρτηση f παίρνει μία είσοδο x και δίνει μια έξοδο y, μπορούμε να την γράψουμε ως $y = f(x)$. Εν τω μεταξύ, αν μια συνάρτηση μπορεί να πάρει την έξοδο της f ως είσοδο και να μας δώσει ως αποτέλεσμα την είσοδο της f που παράγει αυτή την έξοδο, ονομάζεται αντίστροφη της f και γράφεται ως f^{-1} (αντίστροφη της f).

Με άλλα λόγια, εάν η συνάρτηση f παίρνει είσοδο x και δίνει ως έξοδο y ($y = f(x)$) τότε το f^{-1} μπορεί να πάρει ως είσοδο το y και να μας δώσει ως αποτέλεσμα το x ($x = f^{-1}(y)$).

Ένα παράδειγμα αντίστροφης συνάρτησης είναι μια συνάρτηση που προσθέτει μία μονάδα στην είσοδό της και η αντίστροφη αυτής αφαιρεί μία μονάδα από την είσοδό της:

$$y = add1(2) = 3$$

$$x = sub1(3) = 2$$

Αντίστροφες Τριγωνομετρικές Συναρτήσεις

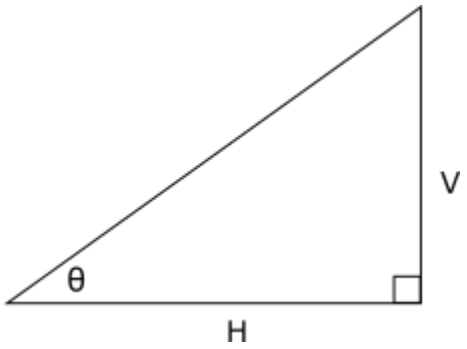
Γνωρίζουμε ήδη ότι οι τριγωνομετρικές συναρτήσεις παίρνουν ως είσοδο μία γωνία και παράγουν ως έξοδο έναν αριθμό. Μπορούμε να τροφοδοτήσουμε την έξοδο μιας τριγωνομετρικής συνάρτησης στην αντίστροφη συνάρτησή της και αυτή επιστρέφει την αρχική είσοδο της τριγωνομετρικής συνάρτησης (γωνία σε ακτίνια). Για παράδειγμα, $\sin\frac{\pi}{2} = 1$ και $\sin^{-1} 1 = \frac{\pi}{2}$.

Οι αντίστροφες τριγωνομετρικές συναρτήσεις έχουν συγκεκριμένα ονόματα. Η αντίστροφη συνάρτηση ημιτόνου, ονομάζεται arcsine. Ομοίως, οι αντίστροφες συναρτήσεις συνημίτονου και εφαπτομένης ονομάζονται arccosine και arctangent, αντίστοιχα. Στο Unity, μπορούμε να καλέσουμε αυτές τις συναρτήσεις με αυτό τον τρόπο:

```
1.  float sinAngle = Mathf.Asin(sinValue); // arcsine
2.  float cosAngle = Mathf.Acos(cosValue); // arccosine
3.  float tanAngle = Mathf.Atan(tanValue); // arctangent
```

Γωνίες Κλίσης

Αν γνωρίζουμε τον λόγο της κάθετης ανύψωσης έναντι της οριζόντιας μετατόπιση (offset) ενός λόφου σε ένα επίπεδο παιχνιδιού, πώς υπολογίζουμε τη γωνία κλίσης; Στην παρακάτω εικόνα, πώς υπολογίζουμε τη γωνία θ γνωρίζοντας την κατακόρυφη ανύψωση V και την οριζόντια μετατόπιση H;



Μπορούμε να συσχετίσουμε το θ με τα V και H χρησιμοποιώντας τη συνάρτηση της εφαπτομένης:

$$\tan\theta = \frac{V}{H}$$

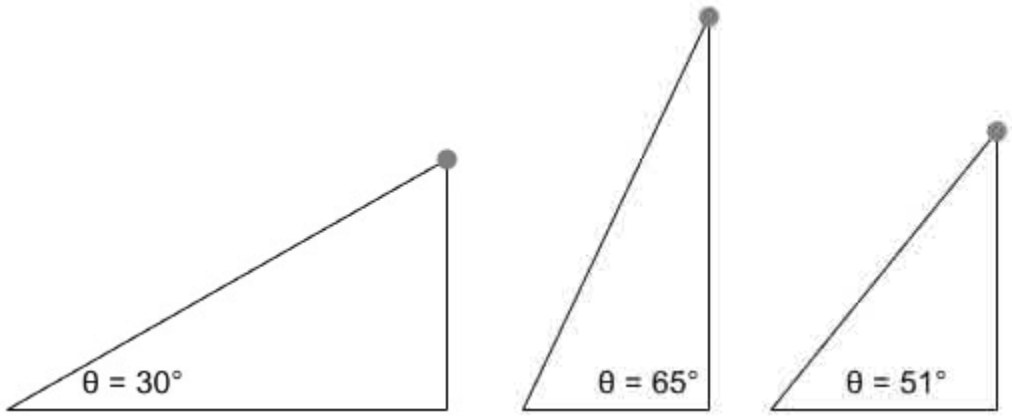
Έπειτα μπορούμε να πάρουμε το θ, δίνοντας ως είσοδο το $\frac{V}{H}$ στην αντίστροφη συνάρτηση της εφαπτομένης:

$$\theta = \tan^{-1}\frac{V}{H}$$

Εναλλακτικά, μπορούμε να θεωρήσουμε την παραπάνω εξίσωση ως αποτέλεσμα του arctangent των δύο πλευρών της προηγούμενης εξίσωσης. Γενικά, το $f^{-1}(f(x))$ απλοποιείται και μας δίνει x. Ομοίως, το $f(f^{-1}(x))$ απλοποιείται και μας δίνει y.

Η γωνία θ είναι σε ακτίνια. Όπως αναφέραμε προηγουμένως, μπορούμε να μετατρέψουμε τη μονάδα μέτρησης γωνίας σε μοίρες πολλαπλασιάζοντας με $\frac{180}{\pi}$.

Έτσι, μπορούμε να γράψουμε ένα μικρό πρόγραμμα το οποίο θα επιτρέπει στο χρήστη να μετακινήσει ένα σημείο στο επίπεδο που σχηματίζει κλίση με την αρχή των αξόνων και να χρησιμοποιώντας τις συντεταγμένες του σημείου (x, y), να υπολογίσουμε και να εμφανίουμε τη γωνία κλίσης.



Και ο ανάλογος κώδικας:

```
1.  Vector3 point = p.transform.position;
2.
3.  // υπολογισμός γωνίας κλίσης σε radians
4.  float angleRad = Mathf.Atan(point.y / point.x);
5.
6.  // μετατροπή σε μοίρες
7.  // το Mathf.Rad2Deg είναι μία σταθερά ίση με 180.0f / Pi
8.  float angleDeg = angleRad* Mathf.Rad2Deg;
9.
10. text = angleDeg + "°";
```

Πεδία Ορισμού και Εύρη Τιμών

Όταν χρησιμοποιούμε αντίστροφες τριγωνομετρικές συναρτήσεις, είναι σημαντικό να κατανοήσουμε τα πεδία ορισμού και τα εύρη τιμών τους.

Το πεδίο ορισμού μιας συνάρτησης είναι το σύνολο όλων των έγκυρων τιμών εισόδου και το εύρος τιμών μιας συνάρτησης είναι το σύνολο όλων των πιθανών τιμών εξόδου.

Για παράδειγμα, το πεδίο ορισμού της $\sin\theta$ είναι το σύνολο όλων των πραγματικών αριθμών, επειδή μπορούμε ως είσοδο να ορίσουμε κάθε γωνία. Το εύρος τιμών του $\sin\theta$ είναι $[-1,1]$, το οποίο περιγράφει όλες τις τιμές μεταξύ και συμπεριλαμβανομένων των -1 και 1. Μια παρένθεση στη θέση μίας αγκύλης, σημαίνει ότι η συγκεκριμένη τιμή δεν περιλαμβάνεται στο σύνολο τιμών (ανοιχτό/κλειστό όριο). Για παράδειγμα το $[0,10)$ δηλώνει ένα σύνολο που περιλαμβάνει όλες τις τιμές μεταξύ 0 και 10, αλλά περιλαμβάνει μόνο 0 (κλειστό όριο) και όχι το 10 (ανοιχτό όριο).

Η αντίστροφη μίας συνάρτησης θα πρέπει, οπότε, απλά να έχει ένα πεδίο ορισμού και ένα εύρος τιμών ίσο με το εύρος τιμών και το πεδίο ορισμού της αντίστοιχης συνάρτησης. Για τις αντίστροφες τριγωνομετρικές συναρτήσεις, αυτό δεν ισχύει.

Οι τριγωνομετρικές συναρτήσεις είναι περιοδικές, πράγμα που σημαίνει ότι πολλές διαφορετικές τιμές εισόδου μπορούν να έχουν την ίδια τιμή εξόδου. Για τα $\sin\theta$ και $\cos\theta$, μπορούν ακόμη και να έχουν διαφορετικές τιμές εισόδου σε μία περίοδο με αποτέλεσμα να παράγουν την ίδια τιμή εξόδου.

Τόσο το $\sin\frac{\pi}{2}$ όσο και το $\sin\frac{5\pi}{2}$ παράγουν ως αποτέλεσμα την τιμή 1. Έτσι, ποια είναι η έξοδος του $\sin^{-1}1$; Δεν μπορεί να είναι ταυτόχρονα ίσο με $\frac{\pi}{2}$, $\frac{5\pi}{2}$, ή άλλες εισόδους που κάνουν το $\sin\theta$ να δίνει ως αποτέλεσμα το 1. Τα εύρη τιμών των αντίστροφων τριγωνομετρικών συναρτήσεων είναι πιο περιορισμένα και αφού έχει συμφωνηθεί από κοινού, χρησιμοποιούνται παγκοσμίως.

Δεδομένου ότι τα εύρη τιμών των $\sin\theta$ και $\cos\theta$ είναι $[-1,1]$, είναι επίσης και τα πεδία ορισμού των $\sin^{-1}x$ και $\cos^{-1}x$. Ως εύρη τιμών των $\sin^{-1}x$ και $\cos^{-1}x$ έχουν επιλεχθεί τα $[-\frac{\pi}{2},\frac{\pi}{2}]$ και $[0,\pi]$, αντίστοιχα. Αυτά τα εύρη καλύπτουν ένα γωνιακό εύρος τιμών π ακτίνων ή 180 μοιρών.

Συνεπώς, το $\sin^{-1}1$ είναι ίσο με $\frac{\pi}{2}$, η οποία είναι η μόνη και μοναδική είσοδος που ως είσοδο στο $\sin\theta$ δίνει αποτέλεσμα ίσο με 1 και βρίσκεται εντός του εύρους τιμών $[-\frac{\pi}{2},\frac{\pi}{2}]$.

Όσον αφορά το $\tan^{-1} x$, δεδομένου ότι το εύρος τιμών του $\tan\theta$ είναι το σύνολο όλων των πραγματικών αριθμών, το πεδίο ορισμού του $\tan^{-1} x$ είναι επίσης το σύνολο όλων των πραγματικών αριθμών. Και το εύρος τιμών του $\tan^{-1} x$ έχει επιλεγεί ως $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$ όπως ακριβώς και το $\sin^{-1} x$.

Η Βοηθητική Συνάρτηση Atan2

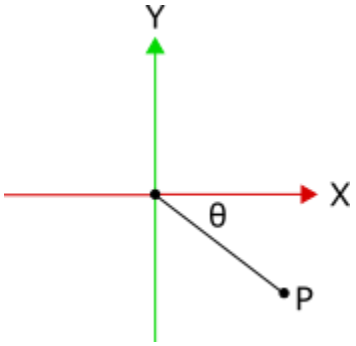
Ας υποθέσουμε πως έχουμε ένα σημείο $P = (P_x, P_y)$ στο δισδιάστατο επίπεδο και βρίσκεται στο πρώτο τεταρτημόριο, δηλαδή $P_x > 0$ και $P_y > 0$. Και ως θ, η γωνία από τον άξονα $+x$ στο τμήμα της γραμμής που συνδέει την αρχή των αξόνων και το P.

Γνωρίζουμε ότι το $\tan\theta = \frac{P_y}{P_x}$, έτσι μπορούμε να υπολογίσουμε το θ από τις συντεταγμένες του P χρησιμοποιώντας τη συνάρτηση arctangent: $\theta = \tan^{-1}\frac{P_y}{P_x}$. Δεδομένου ότι τα P_x και P_y έχουν θετική τιμή, το θ βρίσκεται στο $\left[0, \frac{\pi}{2}\right]$, το οποίο συμπεριλαμβάνεται στο εύρος τιμών του arctangent, $[-\frac{\pi}{2}, \frac{\pi}{2}]$.

Ο υπολογισμός γίνεται με τον παρακάτω κώδικα:

```
1.  float angle = Mathf.Atan(p.y / p.x);
```

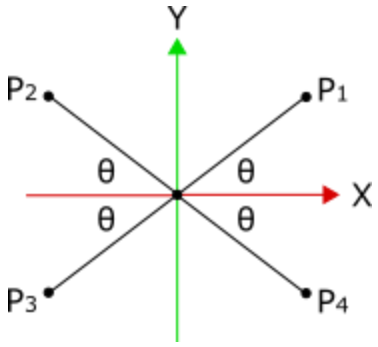
Τι συμβαίνει όμως εάν το P είναι στο τέταρτο τεταρτημόριο, δηλαδή $P_x > 0$ και $P_y < 0$; Το $\frac{P_x}{P_y}$ θα γινόταν αρνητικό και το $\tan^{-1}\frac{P_y}{P_x}$ θα έδινε μια αρνητική γωνία μέσα στο $[-\frac{\pi}{2}, 0]$, το οποίο επίσης συμπεριλαμβάνεται στο εύρος τιμών του arctangent, $[-\frac{\pi}{2}, \frac{\pi}{2}]$.



Προβλήματα προκύπτουν όταν έχουμε το P στο δεύτερο ή τρίτο τεταρτημόριο. Εάν για παράδειγμα το P βρίσκεται στο δεύτερο τεταρτημόριο, δηλαδή $P_x < 0$ και $P_y > 0$, το κλάσμα $\frac{P_y}{P_x}$ είναι

αρνητικό. Μπορούμε να βρούμε ένα σημείο $P_2 = (P_{2x}, P_{2y})$ στο δεύτερο τεταρτημόριο που έχει ως αποτέλεσμα το λόγο $\frac{P_{2y}}{P_{2x}}$ ίσο με το λόγο $\frac{P_{4y}}{P_{4y}}$ ενός σημείου $P_4 = (P_{4x}, P_{4y})$ στο τέταρτο τεταρτημόριο. Ένα τέτοιο ζεύγος σημείων ικανοποιεί την ισότητα $(P_{2x}, P_{2y}) = (-P_{4x}, -P_{4y})$.

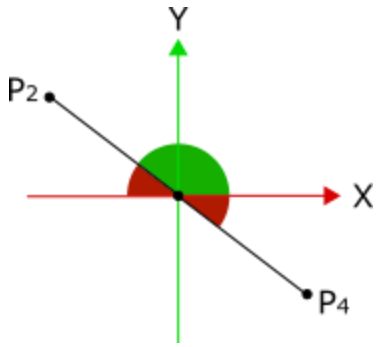
Τα δύο σημεία P_2 και P_4 στο παρακάτω σχήμα έχουν πανομοιότυπους λόγους συντεταγμένων $\frac{P_{2y}}{P_{2x}} = \frac{P_{4y}}{P_{4y}}$.



Επίσης φαίνεται στο παραπάνω σχήμα ότι οι συντεταγμένες των σημείων P_2 και P_4, σε σύγκριση με τους λόγους των συντεταγμένων των σημείων P_1 στο πρώτο τεταρτημόριο και του P_3 στο τρίτο τεταρτημόριο, διαφέρουν μόνο στα πρόσημα (αρνητικά αντί θετικά). Όλες οι οξείες γωνίες μεταξύ των γραμμικών τμημάτων που συνδέουν την αρχή των αξόνων, τα σημεία και τον άξονα x είναι πανομοιότυπα.

Ο λόγος $\frac{P_{2y}}{P_{2x}}$ είναι ίσος με το λόγο $\frac{-P_{4y}}{-P_{4x}} = \frac{P_{4y}}{P_{4x}}$. Έτσι, αν δώσουμε το $\frac{P_{2y}}{P_{2x}}$ ως είσοδο στη συνάρτηση arctangent, το $\tan^{-1}\frac{P_{2y}}{P_{2x}}$ δίνει ως αποτέλεσμα την ακριβώς ίδια αρνητική γωνία όπως και το $\tan^{-1}\frac{P_{4y}}{P_{4x}}$, επειδή μια γωνία στο τέταρτο τεταρτημόριο είναι εντός του εύρους τιμών του arctangent, ενώ μια γωνία στο δεύτερο τεταρτημόριο δεν είναι .

Όταν δίνουμε ως είσοδο στη συνάρτηση arctangent το $\frac{P_{2y}}{P_{2x}}$, αυτό που περιμένουμε να πάρουμε ως έξοδο είναι η πράσινη θετική αμβλεία γωνία στο παρακάτω σχήμα και όχι μία από τις δύο κόκκινες οξείες γωνίες. Ξεκινούμε να μετράμε γωνίες πάντα από την κατεύθυνση $+x$.



Για να γίνει αυτό, προτού συνδυάσουμε τα P_x και P_y σε ένα λόγο και τον δώσουμε ως είσοδο στη συνάρτηση arctangent, ελέγχουμε τα πρόσημα αυτών για να βρούμε σε ποιο τεταρτημόριο βρίσκεται το σημείο. Και αν βρούμε μια γωνία εκτός του εύρους τιμών $[-\frac{\pi}{2}, \frac{\pi}{2}]$, διορθώνουμε την έξοδο της συνάρτησης arctangent για να πάρουμε ως έξοδο, τη γωνία στο σωστό τεταρτημόριο. Με αυτό τον κώδικα μπορούμε να εφαρμόσουμε αυτή τη διόρθωση:

```
1.  // το εύρος τιμών της συνάρτησης είναι (-π, π]
2.  float FixedUpAtan(float py, float px)
3.  {
4.    if (px > 0.0f) // δεν χρειάζεται διόρθωση
5.    {
6.      // py > 0.0f : πρώτο τεταρτημόριο
7.      // py < 0.0f : τέταρτο τεταρτημόριο
8.      return Mathf.Atan(py / px);
9.    }
10.   else if (px < 0.0f) // χρειάζεται διόρθωση
11.   {
12.     if (py > 0.0f) // δεύτερο τεταρτημόριο
13.       return Math.PI + Mathf.Atan(py / px);
14.     else if (py < 0.0f) // τρίτο τεταρτημόριο
15.       return -Math.PI + Mathf.Atan(py / px);
16.     else // γωνία στον αρνητικό άξονα x
17.       return 2.0f * Mathf.PI;
18.   }
19.   else // άπειρο
20.   {
21.     if (py > 0.0f)
22.       return 0.5f * Mathf.PI; // ο λόγος είναι +άπειρο
23.     else if (py < 0.0f)
24.       return -0.5f * Mathf.PI; // ο λόγος είναι -άπειρο
25.     else
26.       return 0.0f; // η αρχή των αξόνων
27.   }
28. }
```

Για καλή μας τύχη, σχεδόν όλες οι τυποποιημένες βιβλιοθήκες μαθηματικών οποιασδήποτε γλώσσας προγραμματισμού παρέχουν μια βοηθητική συνάρτηση που ονομάζεται **atan2**, η οποία έχει πλήρες εύρος τιμών 360 μοιρών $(-\pi, \pi]$ και κάνει ακριβώς αυτό που κάνει ο παραπάνω κώδικας (πιθανότατα πιο αποδοτικά σε βελτιστοποιημένη μορφή). Σημειώστε ότι η σειρά των παραμέτρων είναι Y πρώτη και X δεύτερη. Η συνάρτηση atan2, ανά βιβλιοθήκη, μπορεί να έχει διαφορετική διάταξη εισόδων (arguments), αλλά το y που ακολουθείται από το x είναι το πιο κοινό.

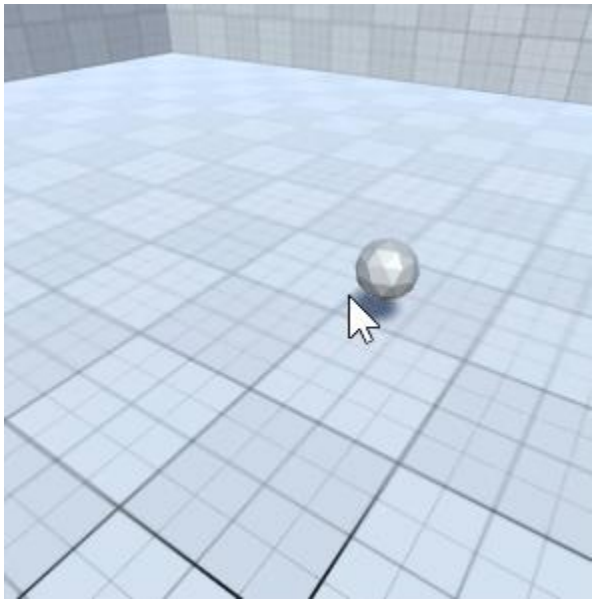
Πολλοί είναι αυτοί που έχουν εσφαλμένη αντίληψη ότι atan2 είναι απλώς μια εναλλακτική λύση για τη συνάρτηση arctangent και δεν κάνει τίποτα περισσότερο από το arctangent. Αυτό φυσικά είναι λάθος. Η συνάρτηση arctangent παίρνει μόνο μία τιμή ως είσοδο και το εύρος τιμών εξόδου της είναι $[-\frac{\pi}{2}, \frac{\pi}{2}]$. Από την άλλη πλευρά, το atan2 παίρνει δύο τιμές ως είσοδο, το P_y και το P_x προτού αυτά συνδυαστούν σε ένα λόγο, και η έξοδος έχει ένα πλήρες εύρος τιμών 360 μοιρών $(-\pi, \pi]$.

Ακολουθώντας τον κέρσορα σε τρισδιάστατο επίπεδο

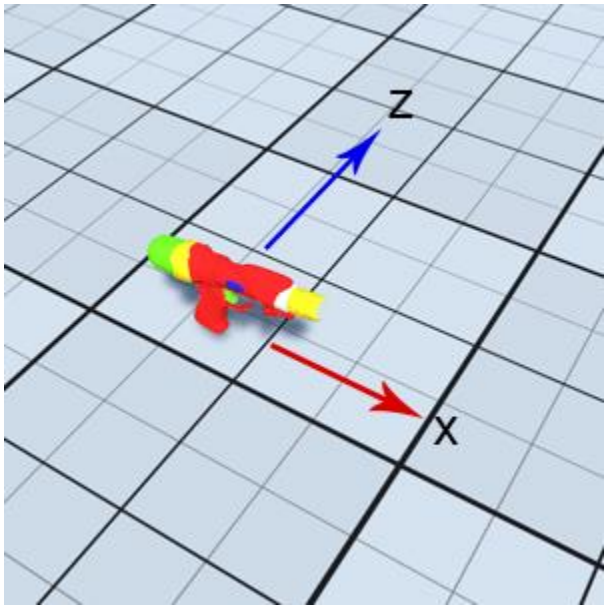
Στο παρακάτω παράδειγμα θα κάνουμε ένα αντικείμενο σε τρισδιάστατο χώρο να περιστρέφεται προς τον κέρσορα του ποντικιού.

Πρώτα, πρέπει να βρούμε τη διασταύρωση μεταξύ της ευθείας (ray) κάτω από τον κέρσορα του ποντικιού και του επιπέδου εδάφους. Στη συνέχεια, τοποθετούμε ένα αντικείμενο (object) στη διασταύρωση αυτή, δημιουργώντας έτσι ως αποτέλεσμα, το αντικειμένου να ακολουθεί τον κέρσορα του ποντικιού στο τρισδιάστατο επίπεδο. Θα χρησιμοποιήσουμε αυτό το αντικείμενο ως στόχο.

```
1.  Camera cam = Camera.current;
2.  Vector3 mouse= Input.mousePosition;
3.  Ray ray = cam.ScreenPointToRay(mouse);
4.  float rayDist;
5.  plane.Raycast(ray, out rayDist);
6.  sphere.position = ray.GetPoint(rayDist);
```



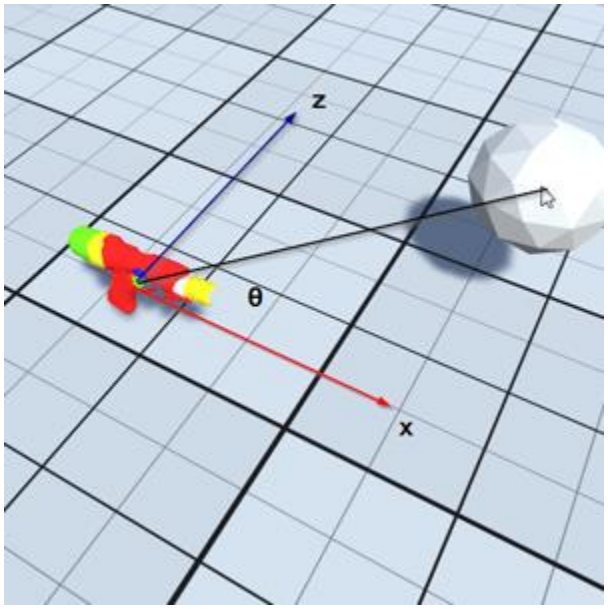
Στη συνέχεια θα χρησιμοποιήσουμε ένα τρισδιάστατο μοντέλο. Όταν αυτό δεν έχει περιστραφεί καθόλου, το διάνυσμά του προς τα εμπρός βρίσκεται στην κατεύθυνση $+x$ και το αριστερό διάνυσμα στην κατεύθυνση $+z$. Αυτό που θέλουμε να επιτύχουμε είναι να περιστρέψουμε το μοντέλο προς το στόχο μας.



Στη συνέχεια, τοποθετούμε το μοντέλο στην αρχή των αξόνων και χρειάζεται να υπολογίσουμε τις συντεταγμένες του στόχου σε σχέση με αυτό:

```
1.  Vector3 coord =
2.    sphere.transform.position
3.    - ufoBunny.transform.position;
```

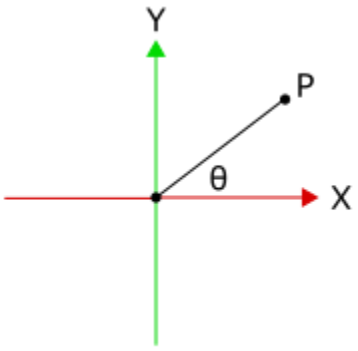
Ας σημειώσουμε στη σκηνή μια γωνία θ μεταξύ του άξονα x και του τμήματος της γραμμής που συνδέει το μοντέλο με το στόχο:



Όπως είδαμε προηγουμένως, η γωνία θ μπορεί εύκολα να υπολογιστεί χρησιμοποιώντας τη συνάρτηση atan2:

```
1.  float thetaRad = Mathf.atan2(coord.z, coord.x); // σε radians
```

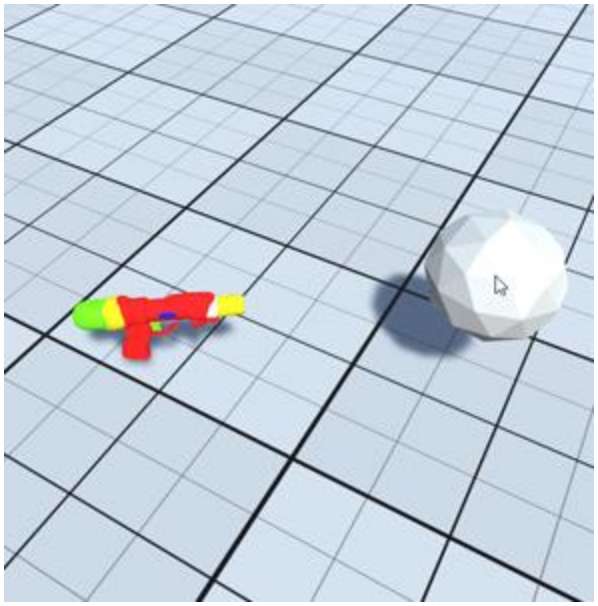
Ας θυμηθούμε το γράφημα:



Αυτό το γράφημα, μας δείχνει το επίπεδο xy. Καθώς η γωνία θ αυξάνεται, το P περιστρέφεται αριστερόστροφα γύρω από την αρχή των αξόνων. Ο άξονας περιστροφής μιας τέτοιας περιστροφής είναι $+z$. Το μοντέλο και ο στόχος βρίσκονται στο επίπεδο xz. Για να "μεταφράσουμε" το γράφημα του επιπέδου xy στο επίπεδο xz, αντιστοιχίζουμε τον άξονα $+x$ στον άξονα $+x$, τον άξονα $+y$ στον άξονα $+z$ και τον άξονα περιστροφής του άξονα $+z$ στον άξονα $-y$.

Πλέον, έχουμε τον άξονα περιστροφής και την επιθυμητή γωνία περιστροφής. Έτσι μπορούμε επιτέλους να κατασκευάσουμε ένα **quaternion** που αντιπροσωπεύει μια τέτοια περιστροφή. Το quaternion είναι ένας τύπος δεδομένων που χρησιμοποιεί το Unity για την περιστροφή αντικειμένων.

```
1.  float thetaDeg = thetaRad * Mathf.Rad2Deg; // σε μοίρες
2.  float axis = Vector3.down; // (0, -1, 0) == άξονας -Y
3.  Quaternion rot = Quaternion.AngleAxis(thetaDeg, axis);
4.  ufoBunny.transform.rotation = rot;
```



Το Unity παρέχει ήδη βοηθητικές συναρτήσεις όπως Quaternion.LookRotation και Transform.LookAt οι οποίες μπορούν να επιτύχουν το ίδιο αποτέλεσμα.

Σύνοψη

Ενώ τα μαθηματικά είναι χρήσιμα ακόμη και από καλλιτεχνικής άποψης της ανάπτυξης παιχνιδιών, οι προγραμματιστές είναι αυτοί που τα χρησιμοποιούν για τη δημιουργία χαρακτήρων, μηχανικής κλπ. Χωρίς τα μαθηματικά οι προγραμματιστές δεν θα μπορούσαν να προγραμματίσουν τα αντικείμενα ενός παιχνιδιού να κάνουν, ούτε τα πιο απλά πράγματα. Ο κώδικας ενός παιχνιδιού, συνδυασμένος με διάφορες μεταβλητές, διανύσματα και άλλα, είναι αυτά που κάνουν το Sonic να τρέχει αργά όταν ο παίχτης ακουμπάει ελαφρώς το D-pad, να τρέχει όλο και γρηγορότερα όταν το πατάει παρατεταμένα, να σταματάει όταν συναντήσει ένα συμπαγές αντικείμενο και να τρέχει διαφορετικά κάτω απ' το νερό.

Δεν είναι δύσκολο να φανταστούμε πως ένα παιχνίδι χωρίς προγραμματισμό και μαθηματικά θα ήταν απλώς ένας σωρός από όμορφη αλλά άχρηστη τέχνη. Αυτά τα δύο μαζί, μας επιτρέπουν όχι μόνο να εξομοιώσουμε τον κόσμο μας και τη φυσική του, όπως το κινούμενο νερό, αλλά και άλλους κόσμους της φαντασίας μας και μηχανικές που δίχως αυτά δεν θα μπορούσαμε να νιώσουμε πως είναι. Μόνο στο Portal έχουμε γνωρίσει πως θα ήταν, αν μπορούσαμε να ανοίξουμε μία πύλη (portal) και να μπούμε μέσα. Μόνο στο Halo, να τρέξουμε με υπεράνθρωπες ταχύτητες και να καρφώσουμε τον εχθρό με ένα ενεργειακό σπαθί.

Νερό που αγγίζει την πραγματικότητα, ανίχνευση μονοπατιών, αυτομάτως δημιουργημένα επίπεδα με αυξημένο βαθμό δυσκολίας, καίρια χτυπήματα, τεχνητή νοημοσύνη που αντιδρά στην είσοδο των παικτών, ακόμη και η ίδια η αρχιτεκτονική της μηχανής του παιχνιδιού. Τίποτα από αυτά δεν θα ήταν πιθανά για ένα προγραμματιστή, χωρίς τη χρήση μαθηματικών.

Το ημίτονο και το συνημίτονο δεν είναι παρά μία πολύ μικρή αλλά καθόλου ασήμαντη περιοχή των μαθηματικών. Τριγωνομετρία. Αν και η χρήση τους στον προγραμματισμό είναι απλή χρησιμοποιώντας δεδομένες συναρτήσεις, στις περισσότερες πλατφόρμες (IDE), για την επίτευξη κάποιων στόχων, ταυτόχρονα δεν πρόκειται να είναι ορθή για κάποιον που δεν γνωρίζει τη βάση αυτής της θεωρίας. Γι' αυτό το λόγο, θα πρέπει, όλοι οι προγραμματιστές παιχνιδιών να γνωρίσουν τι είναι και πως μπορούμε από το μηδέν να τα αξιοποιήσουμε, κατά την ανάπτυξη ενός παιχνιδιού.

Βιβλιογραφία

National Geographic Angry Birds Furious Forces: The Physics at Play in the World's Most Popular Game by Rhett Allain (Author), Peter Vesterbacka (Foreword) [ISBN-13: 978-1426211720]

Essential Mathematics for Games and Interactive Applications, 3rd Edition by Van Verth, James M. (Author), Lars M. Bishop (Author) [ISBN-13: 978-1482250923]

Concrete Mathematics: A Foundation for Computer Science, 2nd Edition by Ronald L. Graham (Author), Donald E. Knuth (Author), Oren Patashnik (Author) [ISBN-13: 978-0201558029]

Physics for Game Developers: Science, math, and code for realistic effects, 2nd Edition by David M Bourg (Author), Bryan Bywalec (Author) [ISBN-13: 978-1449392512]

Mathematics in game development: Trigonometry [https://www.softlion.nl/download/article/Trigonometry.pdf]

Trigonometry for Game Programming [https://www.raywenderlich.com/2736-trigonometry-for-game-programming-part-1-2]

Trigonometry in Game Development [https://prezi.com/fozjzdqmlhta/trigonometry-in-game-development/]

Basic trigonometry for game development [https://gist.github.com/jhonnymichel/06139d11efe23616749b43484b0e3085]

Trigonometry [https://relativity.net.au/gaming/java/Trigonometry.html]

Trigonometry in Gaming [http://library.clickteam.com/guides-tips/trigonometry-in-gaming/]

How/when do you use Trigonometry in game design? [https://forum.unity.com/threads/how-when-do-you-use-trigonometry-in-game-design.329280/]

Utility library for Unity [https://github.com/TheAllenChou/unity-cj-lib]

Different areas of Math required in various types of video game development [http://web.archive.org/web/20081019223627/http:/tetravista.net/video_game_development.php]

Camera field of view: 3D projections & trigonometry

[https://gamedev.stackexchange.com/questions/4436/camera-field-of-view-3d-projections-trigonometry]

Trigonometry for Flash Game Design

[http://www.adobepress.com/articles/article.asp?p=30617&seqNum=6]