ΠΛΗ311 Τεχνητή Νοημοσύνη
Εαρινό Εξάμηνο 2022 - Διδάσκων: Γιώργος Χαλκιαδάκης

Χρήστος Ιωαννίδης
AM:2018030006
Παπαδόπουλος Στέφανος
AM:2018030169

<u>Αναφορά 1ης Προγραμματιστικής Εργασίας</u>

Δομή αναφοράς:

- **A***
  - **Διαδικασία σχεδιασμού A***
  - **A* Δομή Κώδικα**
  - **Πειραματισμοί κατα την διάρκεια υλοποίησης του αλγορίθμου A***

- **Δεύτερη ευρετική συνάρτηση :Heuristic_2()**
  - **Διαδικασία σχεδιασμού Heuristic_2()**
  - **Heuristic_2() Προκλήσεις και δυσκολίες κατά την διάρκεια της υλοποίησης**
  - **Heuristic_2() Δομή υλοποιημένου κώδικα**
  - **Heuristic_2() Γιατί είναι καλύτερη από την ευκλείδεια ευρετική:**
  - **Παρατηρήσεις και αποτελέσματα:**

- **IDA***
  - **Διαδικασία σχεδιασμού IDA***
  - **Πειραματισμοί κατά την διάρκεια υλοποίησης του αλγορίθμου IDA***
  - **IDA* Δομή Κώδικα**

- **<u>Συγκρίσεις μεταξύ των αλγορίθμων</u>**

Σημείωση: Τροποποιήσεις έχουν γίνει μόνο στα αρχεία **IDAstar.py, Astar.py, main.py**

Διαδικασία σχεδιασμού A*

Ο σχεδιασμός ακολούθησε τα εξής βήματα:

- Αρχικά κάναμε έρευνα στον ακριβή τρόπο λειτουργίας του A*, για να καταλάβουμε την αρχή λειτουργίας.
- Είδαμε επίσης διάφορες υλοποιήσεις σε C++ και ψευδοκώδικα. Δεν αντιγράψαμε καμία από αυτές γιατί δεν δούλευαν στο περιβάλλον του προβλήματος.
- Πειραματιστήκαμε με πιθανές recursive υλοποιήσεις, χωρίς επιτυχία.
- Φτιάξαμε έναν λειτουργικό αλγόριθμο με υλοποιήσεις for/while loop, χωρίς να λάβουμε υπόψη τις ειδικές περιπτώσεις “ενημέρωσης” nodes.
- Συμπεριλάβαμε τούτες τις ειδικές περιπτώσεις.
- Υλοποιήσαμε την de rigeur λύση δυναμικού προγραμματισμού του A* με τις 2 λίστες open/closed (ξανά δουλεύοντας από την λογική του αλγόριθμου, όχι “μεταφράζοντας” υπάρχοντα κώδικα, γιατί δεν μπορούσαμε να κάνουμε αυτές τις υλοποιήσεις να δουλέψουν). Αντί της κοινής αντιμετώπισης στην οποία το κόστος του κόμβου αποθηκεύεται στο αντικείμενο του (πχ this_node.set_fcost(69)) χρησιμοποιούμε μια 3η λίστα που με όλα τα κόστη των κόμβων στην open στην μέθοδο που υλοποιεί τον αλγόριθμο.
- Συμπεριλάβαμε μια απλή περίπτωση επίλυσης “ισοπαλιών”, και κρίναμε πως λόγω των δεδομένων του προβλήματος, δεν ήταν απαραίτητο να επεκτείνουμε αυτήν την λειτουργία.

A* Δομή Κώδικα

- **evaluation_function(*node_current*):** Η μέθοδος που καθορίζει το συνολικό κόστος του κόμβου *node_current*. f=g+w*h, όπου g η ήδη υλοποιημένη μέθοδος για το μήκος του μονοπατιού μέχρι τον κόμβο, h μια κάποια ευρετική συνάρτηση, και w ένας σταθερός θετικός ακέραιος.
- **open_list:** Η λίστα με τους κόμβους (nodes) που μπορούμε να εξερευνήσουμε. Κρατάται ταξινομημένη σε σειρά αύξουσα ως προς το **evaluation_function** (f = g+w*h) έκαστου κόμβου.
- **closed_list:** Η λίστα με τους nodes που έχουν ήδη εξερευνηθεί.
- **loopy_astar(*node_current*):** Η μέθοδος που περιλαμβάνει ολόκληρο τον αλγόριθμο. Βάζουμε τον *node_current* στην λίστα **open_list**.Σε κάθε loop εξερευνούμε τον node με το χαμηλότερο **evaluation_function** που βρίσκεται στην λίστα **open_list**. Εξερευνούμε σημαίνει πως κοιτάμε όλους τους successors (“γείτονες” ή “παιδιά”) του και τους προσθέτουμε στην λίστα **open_list**, εκτός 4 περιπτώσεων: (1), πηγαίνοντας από node στον successor χτυπάμε κάποιο εμπόδιο, (2) O successor υπάρχει ήδη στην λίστα **closed_list** (έχει εξερευνηθεί), (3) O successor υπάρχει ήδη στην **open_list**, και (4) πηγαίνοντας από node στον successor χτυπάμε τον στόχο. Στις περιπτώσεις (1) και (2), απλά αγνοούμε αυτόν τον successor. Στην περίπτωση (4) τερματίζουμε των αλγόριθμο. Στην περίπτωση (3) κρατάμε τον node με το μικρότερο **evaluation_function** στην **open_list** και αγνοούμε ή αφαιρούμε τον άλλον. Μεταφέρουμε τον node που μόλις εξερευνήσαμε στο **closed_list**.

- **execute_search:** Η μέθοδος που καλεί την **loopy_astar** με όρισμα *node_current* τον αρχικό κόμβο.

Πειραματισμοί κατα την διάρκεια υλοποίησης του αλγορίθμου A*:
Πειραματισμοί με πιθανή recursive υλοποίηση (αποφασίσαμε εναντίον της για να κάνουμε πιο εύκολη την λύση δυναμικού προγραμματισμού)

Διαδικασία σχεδιασμού Heuristic_2() all the math,thoughts and experimentation before implementation
Η ιδέα της ευρετικής συνάρτησης είναι να έρθει όσο το δυνατόν πιο κοντά στην πραγματική μικρότερη απόσταση που θα μπορούσαμε να διανύσουμε από αυτό το node, χωρίς την χρήση εκτενών υπολογισμών, ιδανικά πολυπλοκότητας O(1), και ιδανικά από την πλευρά της υποτίμησης αντί της υπερτίμησης ούτως ώστε να εξασφαλίσουμε το καλύτερο τελικό μονοπάτι. Με τα δεδομένα του προβλήματος, κάναμε κάποιες αποφάσεις για τα πράγματα που θα υπολογίζονται και δεν θα υπολογίζονται.

Πρώτον, δεν θα υπολογίζουμε τα εμπόδια (θέσεις, μεγέθη, ούτω καθεξής) στην συνάρτηση, πρώτον γιατί αυτό θα απαιτούσε πολυπλοκότητα τουλάχιστον O(n), δεύτερον γιατί αυτές οι πολυπλοκότητες θα λύνονται από τον A* αλγόριθμο ούτως ή άλλως, τρίτον γιατί δεν είμαστε σίγουροι πως οι πληροφορίες του πού βρίσκονται όλα τα εμπόδια είναι πληροφορίες που ο αλγόριθμος "υποτίθεται" πως θα μπορεί να γνωρίζει.

Δεύτερον, δεν θα υπολογίζουμε τις διαστάσεις του στόχου, γιατί εν τέλει υπολογίζουμε πως αυτή η πληροφορία δεν θα κάνει μεγάλη διαφορά, αφού θα πρέπει να είμαστε ήδη πολύ κοντά στον στόχο για να μετράει, και σε αυτό το σημείο δεν υπάρχουν πολλές περιπλοκές που θα μας καθυστερούσαν σημαντικά στις περισσότερες πρακτικές περιπτώσεις.

Τρίτον, θα υπολογίζουμε την κατεύθυνση της τροχιάς (prograde vector) του κάθε node μαζί με την θέση του ως προς τον στόχο, γιατί η κατεύθυνση, δεδομένου του τρόπου κίνησης στο πρόβλημα, επηρεάζει σημαντικά την ελάχιστη πραγματική απόσταση που θα πρέπει να διανύσουμε (αντίθετα από την πολύ κοινή περίπτωση του grid movement που υπάρχει σε όλες τις εκδοχές του αλγορίθμου που βρήκαμε σε εξωτερικές πηγές).

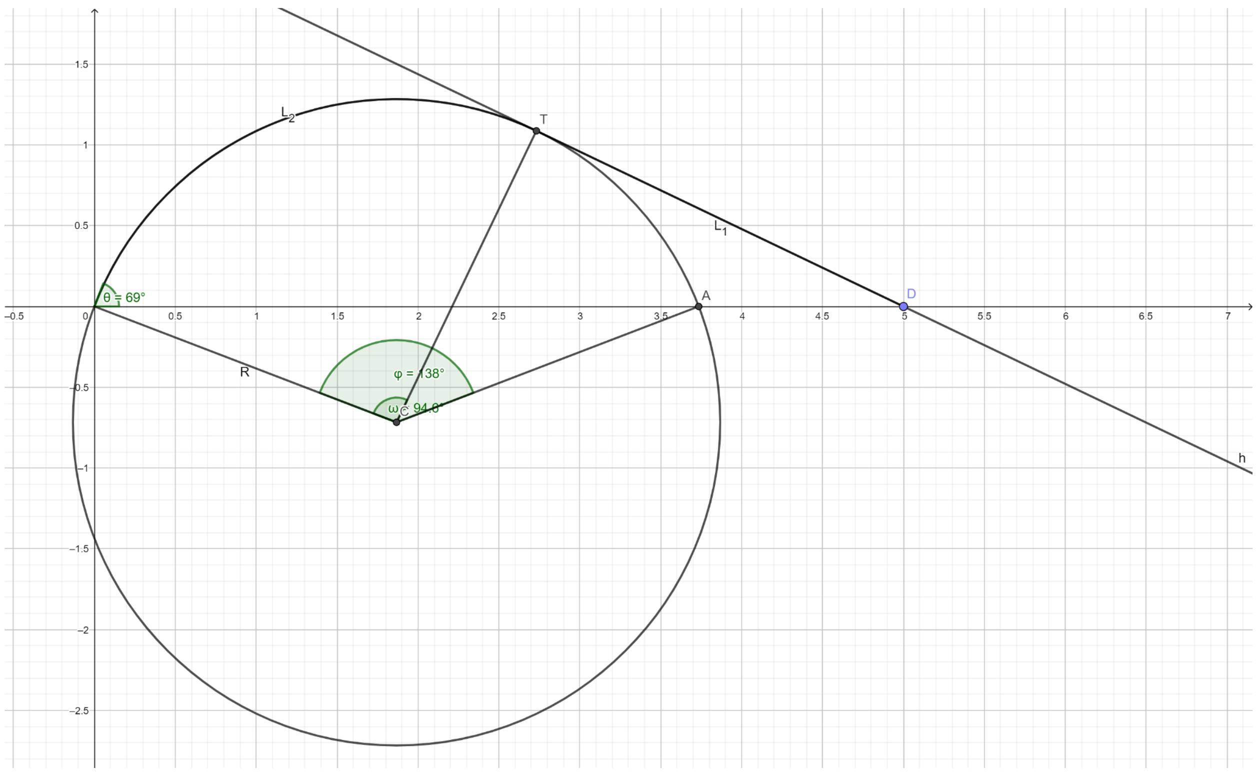
Τέταρτον, δεν θα προσπαθήσουμε να εκτιμήσουμε το μήκος της διαδρομής που θα πρέπει να πάρουμε στην περίπτωση που η απόσταση από τον στόχο είναι αρκετά μικρή, και η γωνία μεταξύ της κατεύθυνσης μας και της ίσιας γραμμής προς τον στόχο αρκετά μεγάλη που θα "χάσουμε" τον στόχο, δηλαδή, δεν θα μπορέσουμε να στρίψουμε αρκετά γρήγορα για να τον πετύχουμε. Σε αυτήν την περίπτωση η συνάρτηση απλά θα επιστρέφει έναν flat μεγάλο αριθμό για να αποθαρρύνει την εξερεύνηση αυτών των μονοπατιών. Ο λόγος είναι πως σε οποιοδήποτε *πρακτικό* σενάριο που μπορούμε να φανταστούμε, καθώς και όλα τα σενάρια που μας δίνονται, μια τέτοια μανούβρα δεν είναι ποτέ απαραίτητη για να βρούμε το καλύτερο μονοπάτι.

Με αυτές τις πληροφορίες, αποφασίσαμε πως θα υπολογίσουμε την *μαθηματικά ακριβή* απόσταση, χωρίς εκτιμήσεις ή απλοποιήσεις. Προφανώς, στην γενική περίπτωση, αν απέχουμε κάποια (ευκλείδεια) απόσταση και προχωράμε σε κάποια γωνία από τον στόχο, αν δεν έχουμε εμπόδια, η καλύτερη δυνατή τροχιά που μπορούμε να πάρουμε συνίσταται από μια όσο τον δυνατόν πιο κλειστή στροφή προς τον στόχο μέχρι να τον “κοιτάμε”, και ύστερα μια ευθεία γραμμή από εκεί προς τον στόχο. Το μήκος αυτής της τροχιάς συνίσταται από το μήκος του κυκλικού τόξου μέχρι να φτάσουμε στο σημείο που “κοιτάμε” τον στόχο, συν το μήκος της ευθείας γραμμής από εκεί μέχρι τον στόχο:

Ο node που θέλουμε να υπολογίσουμε βρίσκεται στο σημείο $(0, 0)$
Ο στόχος μας βρίσκεται στο σημείο $(D, 0)$

Τα δεδομένα μας είναι τα εξής:

1. Η (ευκλείδεια) απόσταση D από τον στόχο
2. Η γωνία μεταξύ της κατεύθυνσής μας και του στόχου θ, $0 \leq \theta \leq \pi$
3. Η ακτίνα στροφής μας (η ακτίνα της πιο κλειστής στροφής που μπορούμε να κάνουμε) $R \approx 12.903$



(Στις γεωμετρικές απεικονίσεις χρησιμοποιούμε $R = 2$ για να έχουμε πιο φιλικά νούμερα, αλλά φυσικά οι τύποι είναι ακριβώς οι ίδιοι)

Ασχολούμαστε πρώτα με την περίπτωση $0 \leq \theta \leq \pi/2$

Αρχικά υπολογίζουμε το σημείο $C = (x_R, y_R)$ γύρω από το οποίο στρίβουμε:

${x_R}^2 + {y_R}^2 = R^2$, το σημείο στο οποίο είμαστε είναι στην περιφέρεια του κύκλου, και

$tan(\pi/2 - \theta) = y_R/x_R$, από το ορθογώνιο τρίγωνο $(0, 0), (x_R, 0), (x_R, y_R)$

Άρα

$y_R = - R/\sqrt{1 + tan(\theta)^2}$ και

$x_R = -y_R tan(\theta)$

Στην συνέχεια υπολογίζουμε το σημείο $\mathrm{T} = (x_T, y_T)$ στο οποίο σταματάμε να στρίβουμε και αρχίζουμε να κινούμαστε σε ευθεία γραμμή, καθώς και την κλίση a αυτής της ευθείας:

$y_T = ax_T + b$, το σημείο T ανήκει στην ευθεία και
$a = -b/D$, το σημείο $(D, 0)$ ανήκει στην ευθεία και
$(y_T - y_R)^2 + (x_T - x_R)^2 = R^2$, το σημείο T ανήκει στον κύκλο και
$(y_T - y_R)a + (x_T - x_R) = 0$, η ευθεία εφάπτεται του κύκλου

Απαλείφουμε την άχρηστη μεταβλητή b με την 1η και 2η εξίσωση:

$y_T = ax_T - aD$ και
$(y_T - y_R)^2 + (x_T - x_R)^2 = R^2$ και
$(y_T - y_R)a + (x_T - x_R) = 0$

Παρατηρούμε πως το σύστημα απαρτίζεται από 2 γραμμικές εξισώσεις και 1 τετραγωνική. Άρα είναι επιλύσιμο με την τετραγωνική φόρμουλα. Αφού απορρίψουμε την 1 από της 2 λύσεις που δεν αποκρίνεται στα δεδομένα, καταλήγουμε στα εξής:

$x_T = ((x_R^{\ 3} + y_R C) - 2x_R^{\ 2}D + x_R B + R^2 D)/(A + y_R^{\ 2} + D^2)$ και
$a = -(C - y_R(x_R - D))/(A - R^2 + D^2)$ και
$y_T = a(x_T - D)$

Με A, B, C τις βοηθητικές σταθερές:

$A = x_R^{\ 2} - 2x_R D$ και
$B = y_R^{\ 2} - R^2 + D^2$ και
$C = R\sqrt{A + B}$

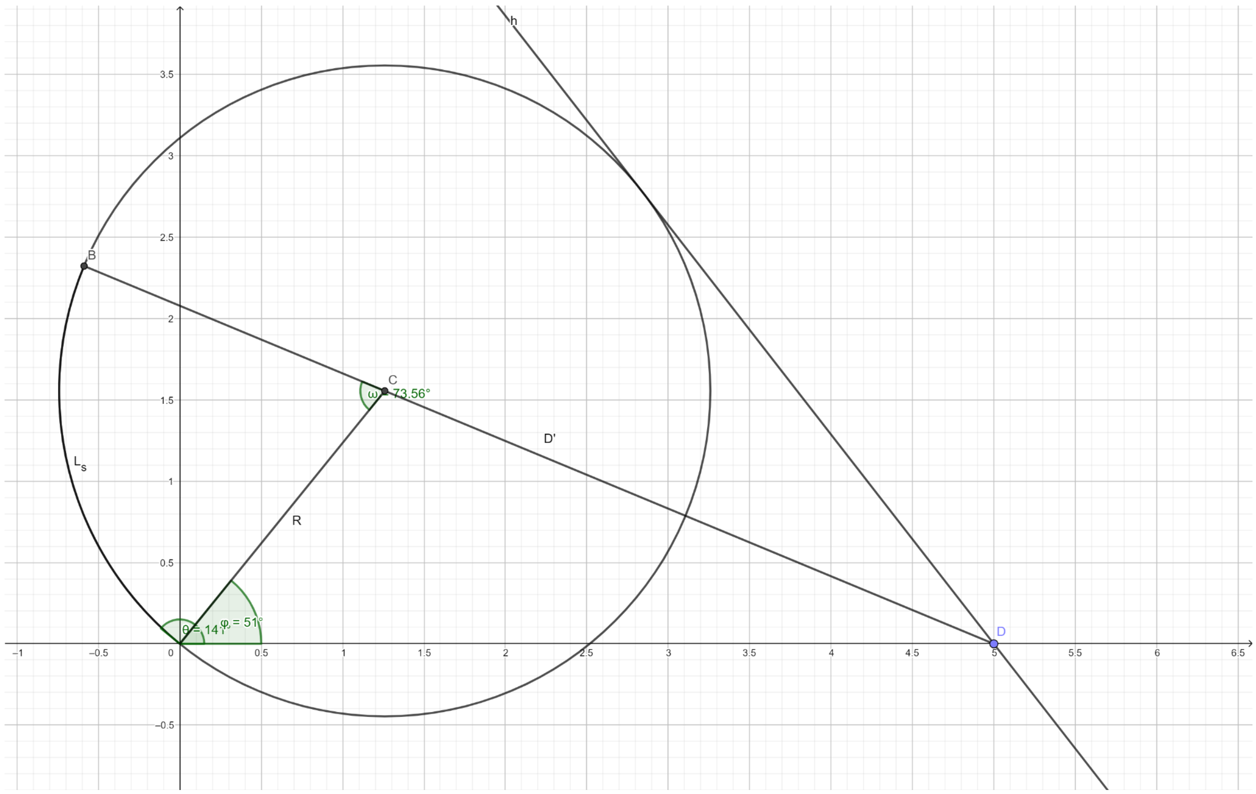
Τα μήκη που μας ενδιαφέρουν είναι λοιπόν:

$L_1 = \sqrt{(D - x_T)^2 + y_T^{\ 2}}$, το μήκος της ευθείας γραμμής και
$L_2 = \omega R$, το μήκος του κυκλικού τόξου

Με ω την γωνία $(0, 0)$, (x_R, y_R), (x_T, y_T), δηλαδή την γωνία που στρίβουμε πάνω στον κύκλο

Αυτή η γωνία υπολογίζεται ως εξής

$atan(a) = \theta - (2\theta/\varphi)\omega$, και
$sin(\varphi/2) = x_R/R$
Άρα
$\omega = \theta(\theta - atan(a))/asin(x_R/R)$
Άρα
$L_1 = \sqrt{(D - x_T)^2 + y_T^2}$ και
$L_2 = R\theta(\theta - atan(a))/asin(x_R/R)$ και προφανώς
$L(D, \theta) = L_1 + L_2$ το μήκος που μας ενδιαφέρει



(Στις γεωμετρικές απεικονίσεις χρησιμοποιούμε $R = 2$ για να έχουμε πιο φιλικά νούμερα, αλλά φυσικά οι τύποι είναι ακριβώς οι ίδιοι). Τονίζουμε επίσης πως το πάνω μέρος του κύκλου είναι ακριβές ημικύκλιο.
Τώρα κοιτάμε την περίπτωση $\pi/2 < \theta \leq \pi$
Σε αυτήν την περίπτωση θα μετακινηθούμε πρώτα μια απόσταση L_s πάνω στον κύκλο περιστροφής μας έτσι ώστε να βρεθούμε με $\theta = \pi/2$ σε κάποια απόσταση D' από τον στόχο, και μετά θα ακολουθήσουμε τα προηγούμενα βήματα.

Ορίζουμε $\varphi = \theta - \pi/2$, την “excess” γωνία

Χρειάζεται να υπολογίσουμε ξανά το κέντρο του κύκλου $C = (x_R, y_R)$. Παρομοίως με πριν, προκύπτει:

$y_R = R/\sqrt{1 + tan(\varphi)^2}$, και
$x_R = y_R/tan(\varphi)$

Από το σχήμα είναι προφανές πως:
$D' = R + \sqrt{y_R^2 + (x_R - D)^2}$ και
$L_s = \omega R$
Με ω αυτήν την φορά την γωνία του αρχικού τόξου που διαγράφουμε πριν βρεθούμε με $\theta = \pi/2$

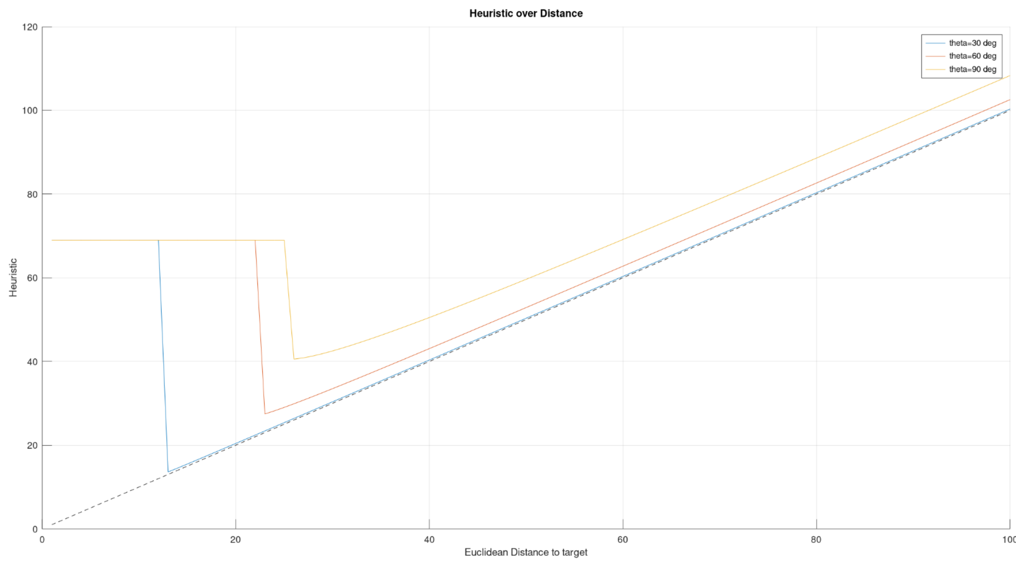
Για να υπολογίσουμε την γωνία ω, εστιάζουμε στο τρίγωνο $(0, 0)$, (x_R, y_R), $(D, 0)$ και ορίζουμε άγνωστη γωνία δ την (x_R, y_R), $(D, 0)$, $(0, 0)$. Τώρα έχουμε:

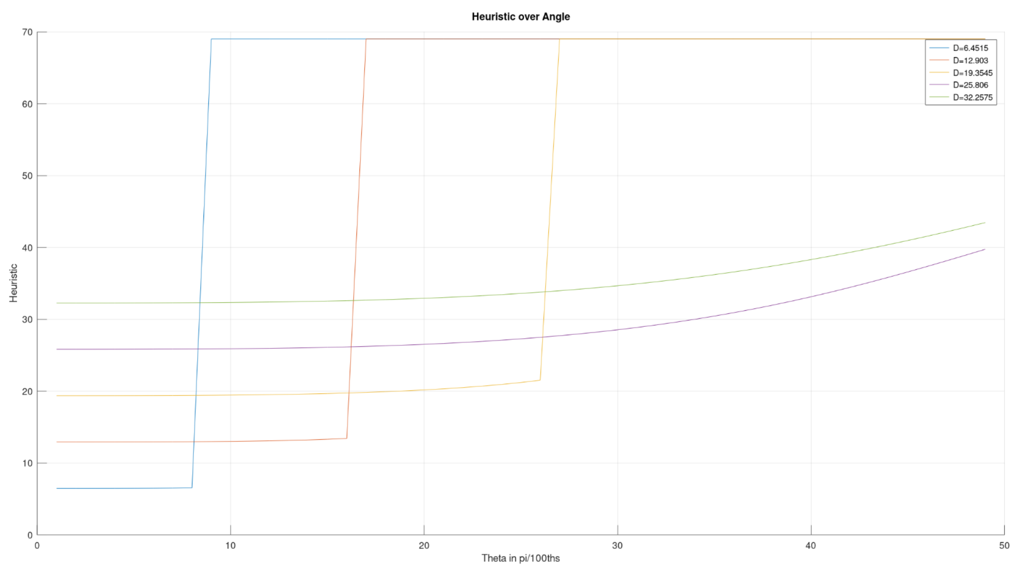
$\varphi + (\pi - \omega) + \delta = \pi$, άθροισμα των γωνιών τριγώνου και
$R/sin(\delta) = D/sin(\pi - \omega)$, νόμος ημιτόνων

Απαλείφουμε την άχρηστη μεταβλητή δ και έχουμε:
$Rsin(\pi - \omega) = Dsin(\omega - \varphi)$
Με το οποίο εύκολα καταλήγουμε σε:
$sin(\omega) = (Dcos(\varphi) - R)/\sqrt{D^2 + R^2 - 2RDcos(\varphi)}$

Με την γωνία ω υπολογίζουμε το μήκος τόξου L_s, και με την απόσταση D' υπολογίζουμε την συνάρτηση $L(D', \pi/2)$ όπως περιγράφεται στην 1η περίπτωση. Άρα το μήκος που μας ενδιαφέρει είναι:
$L(D, \theta) = L_s + L(D', \pi/2)$

Αυτή είναι η βασική ιδέα της συνάρτησης που χρησιμοποιούμε. Υστέρως, προσθέτουμε κάποιες απλές υποπεριπτώσεις για την εκτίμηση των διαστάσεων του στόχου, και για αποφυγή σύγκρουσης με τα άκρα του δρόμου (το ένα εμπόδιο που μπορούμε να συμπεριλάβουμε εύκολα, από πολλές κινήσεις πριν, με O(1) πολυπλοκότητα, και φαίνεται λογικό πως “υποτίθεται” ότι μπορούμε να το γνωρίζουμε)





Heuristic_2() Προκλήσεις και δυσκολίες κατά την διάρκεια της υλοποίησης:

Προφανώς υπήρχε το μαθηματικό πρόβλημα, το οποίο λύθηκε, και επικυρώθηκε με χρήση Matlab για την υλοποίηση της συνάρτησης ταυτόχρονα με Geogebra για να κατασκευή του γεωμετρικού προβλήματος. Το πρόβλημα, όπως αναφέρεται πάνω, έπρεπε να χωριστεί σε 2 υποκατηγορίες, με την επίλυση της 2ης να βασίζεται στην 1η.

Heuristic_2() Δομή υλοποιημένου κώδικα:

- **heuristic_helper(***euc_distance, angle***):** Η μέθοδος που υλοποιεί όλους τους υπολογισμούς που περιγράφηκαν παραπάνω. Καθώς οι υπολογισμοί έχουν ήδη περιγραφεί, και άρα δεν τους επαλαναμβάνουμε εδώ
- **heuristic_function_2(***node_current***):** Η ευρετική συνάρτηση που καλείται από την **evaluation_function**. Χρησιμοποιούμε την “σπιτική” μας **heuristic_function** (η οποία είναι μια απλή ευκλείδεια ευρετική) για να βρούμε την απόσταση του *node_current* από τον στόχο, και δοσμένες μεθόδους από την κλάση SearchBaseClass(ABC) του αρχείου base_class.py για να βρούμε την γωνία θ του *node_current* που χρειαζόμαστε για να υπολογίσουμε την καινούρια μας ευρετική. Στην συνέχεια καλούμε την μέθοδο **heuristic_helper**, με ορίσματα *euc_distance* την απόσταση που βρήκαμε, και *angle* την γωνία που βρήκαμε.
- **r:** Η ακτίνα στροφής (turn radius) μας. Μια κρίσιμη σταθερά μέσα στην μέθοδο **heuristic_helper**, η οποία αναπαριστά την ακτίνα του κύκλου που θα διαγραφόταν αν το υποθετικό αυτοκίνητο (?) που ελέγχουμε έκανε την πιο κλειστή στροφή που μπορούσε. Εκτιμήθηκε πειραματικά (μ’άλλα λόγια, από τις εξαγόμενες εικόνες) ως περίπου 12.903

Heuristic_2() Γιατί είναι καλύτερη από την ευκλείδεια ευρετική:

Η καινούρια ευρετική συνάρτηση υλοποιήθηκε στο πρόβλημά μας, και συγκρίθηκε στα 3 διαφορετικά σενάρια με όλα τα απαραίτητα βάρη, έναντι της ευκλείδειας ευρετικής. Παρατηρήθηκε πως με τα καλύτερα βάρη για κάθε σενάριο, η καινούρια ευρετική είχε ίδια απόδοση (αριθμός nodes που εξερευνήθηκαν μέχρι να βρούμε το καλύτερο ή σχεδόν (<1% διαφορά) καλύτερο μονοπάτι, περισσότεροι nodes => μικρότερη απόδοση) με την ευκλείδεια στο 1ο σενάριο (9 nodes αμφότερες), περίπου 3 φορές καλύτερη απόδοση στο 2ο σενάριο (19 έναντι 60 nodes), και περίπου 2 φορές καλύτερη απόδοση στο 3ο σενάριο (79 έναντι 169 nodes). Επομένως, η δεύτερη ευρετική είναι αποδεδειγμένα καλύτερη από την απλή ευκλείδεια, σε αυτό το πρόβλημα, κάτι αναμενόμενο αφού υπολογίζει την μικρότερη απόσταση λαμβάνοντας υπόψην τους περιορισμούς κίνησης που προέρχονται από την φύση του προβλήματος. Είναι επίσης αξιόλογο το οτι καθώς η καινούρια ευρετική τείνει στην ευκλείδεια καθώς η γωνία τείνει στο 0, για σχετικά “απλά” σενάρια όπως το 1ο, η απόδοση των 2 ταυτίζεται και είναι, εν τέλει, πολύ κοντά στην καλύτερη δυνατή απόδοση (στην οποία δεν εξερευνούμε κανέναν node που δεν ανήκει στο βέλτιστο μονοπάτι)

Παρατηρήσεις και αποτελέσματα:

Η καινούρια ευρετική συνάρτηση υλοποιήθηκε στο πρόβλημά μας, και συγκρίθηκε στα 3 διαφορετικά σενάρια με όλα τα απαραίτητα βάρη, έναντι της ευκλείδειας ευρετικής. Παρατηρήθηκε πως με τα καλύτερα βάρη για κάθε σενάριο, η καινούρια ευρετική είχε ίδια απόδοση (αριθμός nodes που εξερευνήθηκαν μέχρι να βρούμε το καλύτερο ή σχεδόν (<1% διαφορά) καλύτερο μονοπάτι, περισσότεροι nodes => μικρότερη απόδοση) με την ευκλείδεια στο 1ο σενάριο (9 nodes αμφότερες), περίπου 3 φορές καλύτερη απόδοση στο 2ο σενάριο (19 έναντι 60 nodes), και περίπου 2 φορές καλύτερη απόδοση στο 3ο σενάριο (79 έναντι 169 nodes). Επομένως, η δεύτερη ευρετική είναι αποδεδειγμένα καλύτερη από την απλή ευκλείδεια,

σε αυτό το πρόβλημα, και παρ’όλο που η συνάρτηση είναι σαφώς πιο περίπλοκη από $\sqrt{\Delta x^2 + \Delta y^2}$, η υπολογιστική της πολυπλοκότητα παραμένει O(1).

Διαδικασία σχεδιασμού IDA*

Για τον σχεδιασμο του αλγορίθμου χρησιμοποιηθηκε σαν βαση ο κώδικας του DFS
Και απο εκει και περα ακολουθησαν οι εξης τροποποιήσεις

- Η def recursive_DFS(self, node_current) μετατράπηκε σε def recursive_search(self,limit,node_current) με ουσιαστικη αλλαγη την προσθηκη της μεταβλητης **limit** η οποια αντιπροσωπεύει το οριο μεχρι το οποίο θα επεκταθουν κομβοι σε αυτην την κληση.
- Η recursive_search πλεον επιστρέφει περα απο το αν βρέθηκε μονοπάτι και το καινουριο limit
- Μεσα στην execute_search δημιουργηθηκε ενας βροχος ο οποιος καλει την συνάρτηση αναζήτησης εως οτου βρεθει μονοπάτι, καθε φορα με το καινουριο limit που επέστρεψε η προηγουμενη κληση
- Μέσα στην recursive_search πλέον γίνεται αναδρομική προσπέλαση των κόμβων.Απο αυτους τους κομβους επεκτείνονται μονο οσοι εχουν εκτιμώμενο κόστος μικρότερο από το limit και απο αυτους που δεν τηρουν τις προϋποθέσεις. για επέκταση επιλέγεται ο κομβος με το μικρότερο εκτιμώμενο κοστος για να οριστει το καινουριο limit.

Πειραματισμοί κατα την διάρκεια υλοποίησης του αλγορίθμου IDA*

Σε γενικές γραμμές η υλοποίηση του αλγορίθμου ήταν αρκετα straightforward και δεν χρειάστηκαν πολλοί πειραματισμοί. Παρ’όλα αυτά έγινε προσπάθεια τροποποίησης του γραφήματος που εμφανίζεται κατά την διάρκεια της εκτέλεσης έτσι ώστε σε κάθε κλήση της συνάρτησης αναζήτησης με καινούριο limit αυτό να καθαρίζει και έτσι να φαίνονται καλύτερα τα στάδια εκτέλεσης.Παρ’όλα αυτα η προσπάθεια αυτή αποδείχθηκε αποτυχής λόγω της μεγάλης πολυπλοκότητας του κώδικα. Ήταν πολύ δύσκολο να βρεθούν και να τροποποιηθουν οι κατάλληλες μεταβλητές χωρίς να χαλάσει κάτι άλλο.

IDA* Δομή Κώδικα

- **limit:** Αντιπροσωπεύει το όριο το οποίο δεν πρέπει να υπερβαίνουν οι κόμβοι προκειμένου να επεκταθούν. Αυξάνεται με καθε κλήση της συνάρτησης αναζήτησης.

- **execute_search(args….) :** Τρέχει έναν βρόχο ο οποίος καλεί την συνάρτηση αναζήτησης εως ότου βρεθεί μονοπάτι.Σε κάθε επιστροφή της συνάρτησης αναζήτησης εφόσον δεν έχει βρεθεί μονοπάτι ξανακαλεί την συνάρτηση αναζήτησης με το καινούριο limit που επέστρεψε η προηγούμενη κλήση της.

- **recursive_search(self,limit,node_current) :** Αναδρομική συνάρτηση η οποία εκτελεί την αναζήτηση. Εξερευνά και επεκτείνει κόμβους με εκτιμώμενο κόστος μικρότερο από το limit. Απο τους κόμβους που εξερευνήθηκαν αλλα δεν επεκτάθηκαν επιλέγει τον κομβο με το μικρότερο εκτιμώμενο κοστος για και το ορίζει ως το καινούριο limit το οποίο και επιστρέφει μαζί με την πληροφορία του αν βρέθηκε μονοπάτι.

<u>Συγκρίσεις μεταξύ των αλγορίθμων</u>

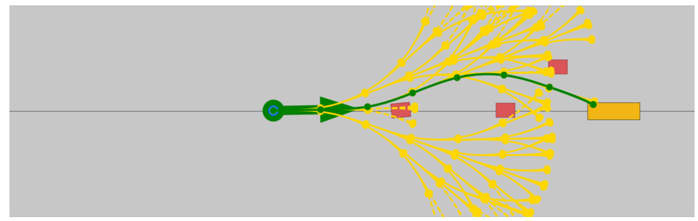
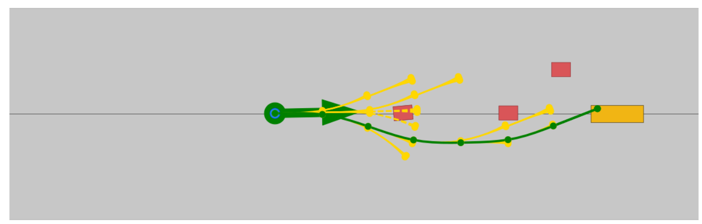
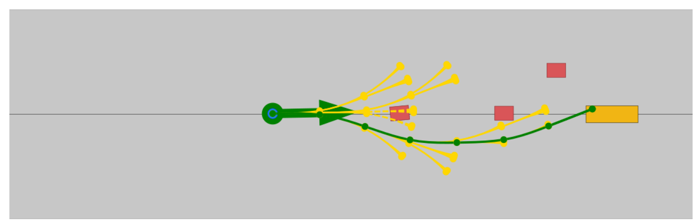
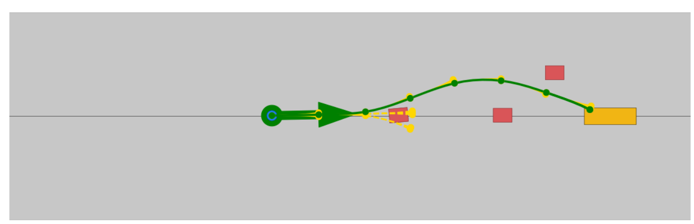
Παρακάτω παρατίθεται ενας πίνακας με κάποια αξιοσημείωτα δεδομένα απο την έξοδο του προγράμματος που θα μας βοηθήσουν να συγκρίνουμε ποιο w ειναι καλύτερο για τον A* καθώς και να συγκρίνουμε τους 2 αλγορίθμους

	<Scenario 1>	<Scenario 2>	<Scenario 3>
A* (w=0) (Dijkstra)	Visited Nodes number: 264 Estimated Cost:32.500075 Real Cost:35.144471	Visited Nodes number: 1217 Estimated Cost:52.500061 Real Cost:55.392724	Visited Nodes number: 584 Estimated Cost:57.537469 Real Cost:69.084362
A* (w=1)	Visited Nodes number: 41 Estimated Cost:32.500075 Real Cost:35.0774	Visited Nodes number: 299 Estimated Cost:52.500061 Real Cost:55.392724	Visited Nodes number: 288 Estimated Cost:57.537469 Real Cost:69.084362
A* (w=2)	**Visited Nodes number: 23** Estimated Cost:32.500075 Real Cost:35.144471	**Visited Nodes number: 48** Estimated Cost:52.500061 Real Cost:55.392724	**Visited Nodes number: 180** Estimated Cost:57.537469 Real Cost:69.084362
A* (w=3)	Visited Nodes number: 23 Estimated Cost:32.50007568247 Real Cost:35.14447138766	Visited Nodes number: 48 Estimated Cost:52.50006102209 Real Cost:55.39272454877	Visited Nodes number: 180 Estimated Cost:57.537469 Real Cost:69.084362
IDA*:	**Visited Nodes number: 468** Estimated Cost:32.50005538456 Real Cost:35.077439345	**Visited Nodes number: 13190** Estimated Cost:52.50004233225 Real Cost:55.35498293764	**Visited Nodes number: 18537** Estimated Cost:57.5371282 Real Cost:69.084362

<u>Από τα παραπάνω δεδομένα μπορούμε να κάνουμε τις εξής παρατηρήσεις</u>

- Η περίπτωση με την ελάχιστη απόσταση για τις διάφορες περιπτώσεις του A* ειναι αυτή με w=1. Παρόλα αυτα θεωρούμε ως βέλτιστη την περίπτωση με **w=2**. Αυτο το επιλέξαμε επειδή μπορεί να δώσει τελικό κόστος διαδρομής μόνο ελαφρώς αυξημένο της προηγούμενης περίπτωσης ενώ βλέπουμε τεράστια διαφορά στους κόμβους που χρειάστηκε να επισκεφτεί προκειμένου να βρει μονοπάτι.
- Ο Αλγόριθμος **IDA*** παρόλο που σε πολλες περιπτώσεις δίνει ελαφρώς καλύτερη απόσταση, ο αριθμός το κόμβων που επρεπε να επισκεφτεί στην πορεία είναι τάξεις μεγεθους μεγαλύτερος από αυτόν του A* αυτό συμβαίνει επειδή λόγω της επανάληψης επισκέπτεται πολλούς κόμβους ξανά και ξανά.
- Η 2η ευρετική μας, **heuristic_function_2**, αποτελεί καλή ευρετική για τον αλγόριθμο A* σε αυτό το πρόβλημα για οποιοδήποτε πρακτικό σενάριο. Όταν λέμε καλή, εννοούμε καλύτερη από την de facto απλή ευκλείδεια ευρετική που θα χρησιμοποιούνταν σε αυτού του είδους το πρόβλημα (αφου η κίνηση δεν είναι βασισμένη σε κάποιο grid, αλλά “ελεύθερη”). Έως και 3 φορές πιο γρήγορη αναλόγως του σεναρίου, συγκεκριμένα.

Παρακάτω παρατίθενται σαν παραδείγματα τα διαγράμματα εξόδου του προγράμματος για το πρώτο σενάριο για να γίνουν εμφανείς και οπτικά οι παραπάνω παρατηρήσεις.

SCENARIO 1	
A* w=0	
A* w=1	
IDA*	
A* w=2	

Παράδειγμα εξόδου αρχείου:

```
<Scenario 1>
A* (w=0)
	Visited Nodes number: 264

Path:(60.0,0.06)->(64.5,0.15000000000000002)->(68.98824564428476,0.4286351005712119)->(73.2866125261574,1.731155761324
7845)->(77.5409446101468,3.1853082435222184)->(82.01118323423854,3.427808939564313)->(86.35552861233279,2.287992656
692106)->(90.53789678357063,0.6382358859652373)
	Heuristic Cost (initial node):32.50007568247887
	Estimated Cost:32.50007568247887
	Real Cost:35.14447138766721
A* (w=1)
	Visited Nodes number: 41

Path:(60.0,0.06)->(64.49579743107746,-0.03895423906396718)->(68.84281301190049,-1.1685441925210989)->(73.1518926735628
4,-2.4514056003865288)->(77.6359481281286,-2.70785170242066)->(82.12419377241336,-2.4292166018494483)->(86.422560654
286,-1.1266959410958757)->(90.61581698464897,0.5062772209220701)
	Heuristic Cost (initial node):32.50007568247887
	Estimated Cost:32.50007568247887
	Real Cost:35.077439345714
A* (w=2)
	Visited Nodes number: 23

Path:(60.0,0.06)->(64.5,0.15000000000000002)->(68.98824564428476,0.4286351005712119)->(73.2866125261574,1.731155761324
7845)->(77.5409446101468,3.1853082435222184)->(82.01118323423854,3.427808939564313)->(86.35552861233279,2.287992656
692106)->(90.53789678357063,0.6382358859652373)
	Heuristic Cost (initial node):32.50007568247887
	Estimated Cost:32.50007568247887
	Real Cost:35.14447138766721
A* (w=3)
	Visited Nodes number: 23

Path:(60.0,0.06)->(64.5,0.15000000000000002)->(68.98824564428476,0.4286351005712119)->(73.2866125261574,1.731155761324
7845)->(77.5409446101468,3.1853082435222184)->(82.01118323423854,3.427808939564313)->(86.35552861233279,2.287992656
692106)->(90.53789678357063,0.6382358859652373)
	Heuristic Cost (initial node):32.50007568247887
	Estimated Cost:32.50007568247887
	Real Cost:35.14447138766721
IDA*:
	Visited Nodes number: 468

Path:(60.0,0.06)->(64.49579743107746,-0.03895423906396718)->(68.84281301190049,-1.1685441925210989)->(73.1518926735628
4,-2.4514056003865288)->(77.6359481281286,-2.70785170242066)->(82.12419377241336,-2.4292166018494483)->(86.422560654
286,-1.1266959410958757)->(90.61581698464897,0.5062772209220701)
	Heuristic Cost (initial node):32.50005538456819
	Estimated Cost:32.50005538456819
	Real Cost:35.077439345714
<Scenario 2>
………………………………………………
```