ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

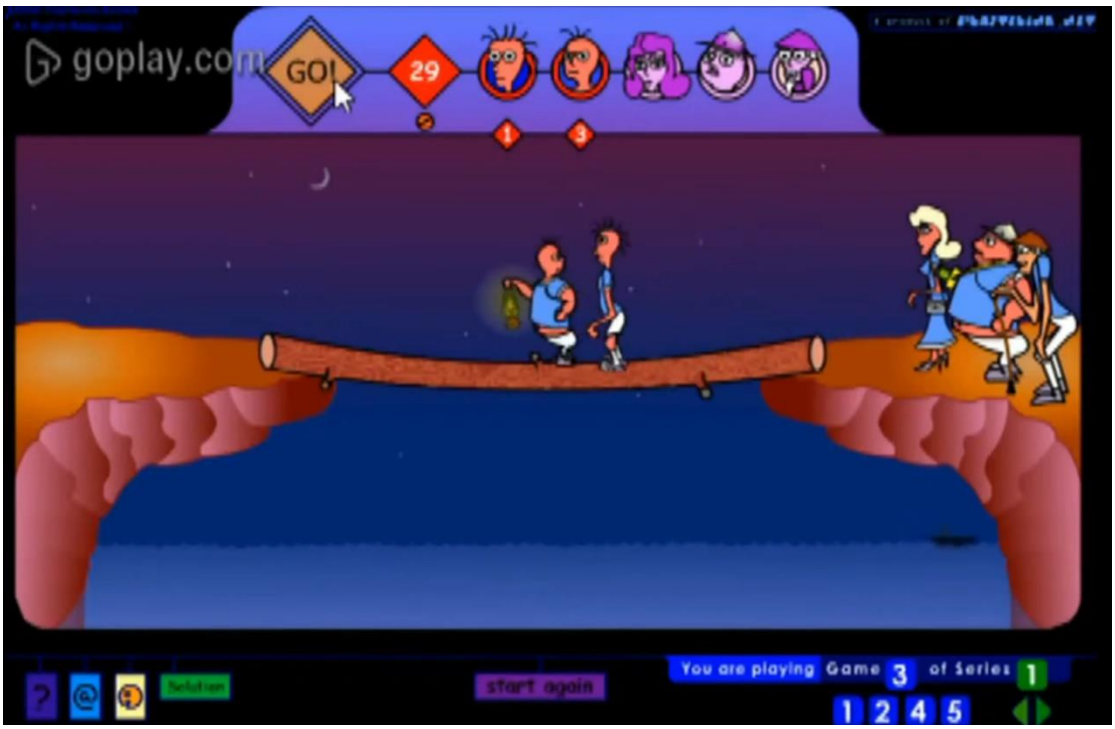
ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ

ΧΕΙΜΕΡΙΝΟ ΕΞΑΜΗΝΟ 2023-24

ΓΙΩΡΓΟΣ ΚΟΥΡΟΣ-3190095

ΚΩΝΣΤΑΝΤΙΝΟΣ ΑΝΔΡΙΝΟΠΟΥΛΟΣ-3190009

ΕΡΓΑΣΙΑ #1 – ΔΙΑΣΧΙΣΗ ΠΟΤΑΜΟΥ ΑΛΓΟΡΙΘΜΟΣ Α*



ΠΕΡΙΓΡΑΦΗ ΕΥΡΕΤΙΚΗΣ ΣΥΝΑΡΤΗΣΗΣ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ (calculate_heuristic)

Καταρχάς η συνάρτηση παίρνει ως παραμέτρους ένα αντικείμενο State που αναπαριστά μία κατάσταση (κόμβο) στο παιχνίδι καθώς και την μεταβλητή game_state που μας δείχνει προς ποια μεριά πρέπει να κινηθούμε βάση της κατάστασης (state) που βρισκόμαστε τώρα (τιμές που παίρνει το game_state -> AtoB ή BtoA όπου A είναι η δεξιά μεριά όπου αρχίζουν όλοι οι παίχτες και B η απέναντι μεριά). Να σημειωθεί ότι στην υλοποίησή μας δεν υπάρχει κάποια μεταβλητή που να υποδηλώνει την λάμπα, εναλλακτικά διασχίζουν απέναντι την γέφυρα 2 άτομα και πάντα πρέπει να γυρνάει ένα πίσω ώστε να γυρίσει υποθετικά η λάμπα στην αρχική μεριά.

Αν βρισκόμαστε σε κατάσταση AtoB η ευρετική παίρνει την τιμή του βραδύτερου παίχτη στη μεριά A όπου αυτό το νούμερο αναπαριστά τον χρόνο που μας απομένει για να φτάσουμε σε τελική κατάσταση. Ο υπολογισμός αυτός αποτελεί πάντα υποεκτίμηση του πραγματικού χρόνου που μας απομένει για να φτάσουμε σε goal state, καθώς υποθέτουμε πάντα ότι βρισκόμαστε ένα βήμα πριν την τελική κατάσταση, δηλαδή ότι έχουν μείνει 2 παίχτες στην μεριά A (αρχική) με την λάμπα. Στην πραγματικότητα, ανάλογα και με τον αριθμό των παικτών N και τους χρόνους τους, θα υπάρχουν περισσότεροι παίχτες ή ακριβώς 2 στην πλευρά A. (αποδεκτή ευρετική)

Αντίστοιχα, σε κατάσταση BtoA παίρνει την τιμή του βραδύτερου παίχτη στην A και την τιμή του γρηγορότερου στην B (θέλουμε να ευνοήσουμε τον γρήγορο ώστε οι επιστροφές να καταναλώνουν όσο το δυνατόν λιγότερο χρόνο) και τις αθροίζει. Το άθροισμα αυτό αναπαριστά τον χρόνο που μας απομένει για να φτάσουμε σε τελική. Ο υπολογισμός αυτός αποτελεί πάντα υποεκτίμηση του πραγματικού χρόνου που μας απομένει να φτάσουμε σε goal state, καθώς υποθέτουμε πάντα ότι βρισκόμαστε 2 βήματα πριν την τελική κατάσταση, δηλαδή ότι έχει μείνει ένας παίχτης στην μεριά A και πρέπει να γυρίσει ένας από την B για να τον πάρει (λόγω λάμπας), όποτε συνολικά 2 βήματα. Στην πραγματικότητα, θα υπάρχουν περισσότεροι παίχτες στην A ή ακριβώς 1 οπότε θα χρειαστούμε πολύ παραπάνω χρόνο για να φτάσουμε σε τελική κατάσταση ή τον χρόνο που υποθέτει η ευρετική (αποδεκτή ευρετική)

ΑΠΟΤΕΛΕΣΜΑΤΑ ΓΙΑ ΔΙΑΦΟΡΕΤΙΚΕΣ ΤΙΜΕΣ ΤΟΥ N (N= # ΜΕΛΗ ΤΗΣ ΟΙΚΟΓΕΝΕΙΑΣ)

- Για N=5 (1,3,6,8,12) με χρονικό όριο 30 το πρόγραμμα βρίσκει λύση περίπου σε 1 sec
- Για N=4 (3,6,12,13) με χρονικό όριο 35 το πρόγραμμα βρίσκει λύση περίπου σε 1 sec
- Για N=6(1,3,6,8,9,12) με χρονικό όριο 40 το πρόγραμμα βρίσκει λύση περίπου σε 1,5 sec
- Για N=7(2,4,6,8,9,10,15) με χρονικό όριο 50 το πρόγραμμα δεν βρίσκει λύση και η εκτύπωση του μηνύματος ήταν περίπου στα 8 sec
- Για το ίδιο N με τους ίδιους χρόνους αλλά με χρονικό όριο 60 το πρόγραμμα βρίσκει λύση σε περίπου 7 sec

ΑΡΧΕΙΟ main.py : Από αυτό το αρχείο εκτελείται το πρόγραμμα. Περιέχει hardcoded το παράδειγμα της εκφώνησης που πρέπει οπωσδήποτε να βρίσκει λύση το πρόγραμμα εντός 30 λεπτών (επιλογή 0). Στην επιλογή 1 λαμβάνει σαν εισόδους τον αριθμό N παικτών και ύστερα το όνομα και τον χρόνο διάσχισης του καθενός καθώς και το χρονικό όριο μέσα στο οποίο πρέπει να βρει βέλτιστη λύση ο αλγόριθμος. Κάθε παίκτης είναι ένα λεξικό της μορφής {'name' : …, 'speed' : …, 'side' : …}. Ύστερα δημιουργείται το initial state (starting_state) ως ένα αντικείμενο της κλάσης State του αρχείου state.py το οποίο θα περιγραφεί παρακάτω. Μετά δημιουργούμε ένα αντικείμενο της κλάσης Tree_Generator και εκτελούμαι την μέθοδο της generate_path(starting_state) ώστε να βρεθεί η βέλτιστη λύση και να εκτυπωθεί το path της.

ΑΡΧΕΙΟ state.py : Κάθε αντικείμενο state αναπαριστά μια κατάσταση (node) του δέντρου. Στον κώδικα υπάρχουν σχόλια που εξηγούν την λειτουργία κάθε μεταβλητής. Εδώ βρίσκεται και η ευρετική συνάρτηση (calculate_heuristic) για την οποία δόθηκε εξήγηση παραπάνω. Κάθε κατάσταση είναι μια λίστα από λεξικά (παίκτες) και που βρίσκονται (μεριά Α αρχική ή Β τελική). Η μέθοδος change_state μας καθοδηγεί ώστε να ξέρουμε κάθε φορά σε ένα state που βρισκόμαστε, προς τα που θα πρέπει να κινηθούμε για το επόμενο state, με λίγα λόγια αναπαριστά την λάμπα στο παιχνίδι, σαν ένα flag που μας υποδεικνύει σε ποια μεριά βρισκόμαστε άρα και προς τα που θα πρέπει να κινηθούμε. Η χρήση της θα περιγραφεί περαιτέρω στο αρχείο tree_generator.py. Η generate_children παράγει όλα τα πιθανά παιδιά του εκάστοτε state. Πρώτα πρέπει να διαπιστώσει το game_state ώστε να ξέρει τι παιδιά θα παράξει (προς τα που κινούμαστε). Αν είμαστε στην Α και πρέπει να πάμε στην Β (AtoB game_state) δημιουργούμαι όλους τους πιθανούς συνδυασμούς παικτών ανά 2 που βρίσκονται στην πλευρά Α και ύστερα για κάθε συνδυασμό δημιουργούμε νέα λίστα με παίκτες (current_state_copy[]) διαμορφωμένη και την προσθέτουμε στην λίστα των πιθανών παιδιών (outcomes[]). Αν είμαστε στην Β και πρέπει να πάμε στην Α (BtoA game_state) ακολουθούμε την ίδια διαδικασία αλλά χωρίς να βρίσκουμε όλους τους πιθανούς συνδυασμούς αφού μόνο ένας γυρνάει πίσω. Υπάρχουν αναλυτικά σχόλια στον κώδικα ώστε να είναι ευανάγνωστος.

<u>ΑΡΧΕΙΟ tree_generator.py :</u> Περιλαμβάνει τρεις μεταβλητές με σχόλια στον κώδικα σχετικά με το τι αναπαριστά η κάθε μία. (path, μέτωπο αναζήτησης, κλειστό σύνολο). Η μέθοδος calculate_cost παίρνει σαν παράμετρο ένα αντικείμενο state και υπολογίζει για αυτό τις τιμές του κόστους, της ευρετικής, του χρόνου που μας απομένει και της τιμής f. Χρησιμοποιούμε επαναληπτική διαδικασία ώστε να διαπιστώσουμε ποιος μετακινήθηκε σε αυτό το state συγκρίνοντας το με το father του. Ύστερα υπολογίζουμε τον μέγιστο χρόνο από αυτούς που μετακινήθηκαν (μπορεί να είναι και ένας αν στο state του father είχαμε game_state BtoA οπότε μόνο ένας μετακινήθηκε). Για περαιτέρω σχετικά με αυτήν την μέθοδο, τα σχόλια στον κώδικα είναι εκτενή και επεξηγηματικά. Μετά έχουμε την μέθοδο generate_path η οποία χρησιμοποιεί τον αλγόριθμο A* ώστε να βρει αν υπάρχει την βέλτιστη λύση. Ξεκινώντας, ελέγχει αν το μέτωπο αναζήτησης είναι άδειο (initial_state case) και αν είναι του προσθέτει το current_state που πήρε σαν παράμετρο στο μέτωπο αναζήτησης (open_set) και υπολογίζει για αυτό τις τιμές του h, f, g, total_time (αυτή η συνθήκη χρησιμοποιείται ουσιαστικά μόνο στην αρχή την πρώτη φορά που καλείται η μέθοδος). Ύστερα προσθέτουμε το current_state στο κλειστό σύνολο, αποθηκεύουμε στην μεταβλητή outcomes[] μια λίστα με όλα τα πιθανά παιδιά του current_state. Αλλάζουμε το game_state ώστε να δημιουργήσουμε για κάθε παιδί της λίστας outcomes[] ένα αντικείμενο state (new_state) που αναπαριστά αυτό το παιδί, το οποίο μετά προσθέτουμε στο μέτωπο αναζήτησης. Στη συνέχεια αφαιρούμε από το μέτωπο αναζήτησης το current_state που μόλις εξετάσαμε και καλούμε στην μεταβλητή current_state την μέθοδο astar_helper() η οποία εξετάζει στο μέτωπο αναζήτησης ποιο state έχει μικρότερο score (μικρότερη f τιμή) και το επιστρέφει στην μεταβλητή current_state. Αν αυτό το νέο current_state είναι τελική κατάσταση, τότε αν είναι εκτός χρονικού ορίου δεν βρέθηκε λύση. Διαφορετικά προσθέτουμε το goal_state στο path[] και με μια while βάζουμε τους προγόνους του και αυτούς στο path[]. Ύστερα αντιστρέφουμε την λίστα path και την εκτυπώνουμε. Αν αυτό το νέο current_state δεν είναι τελική κατάσταση, τότε ελέγχουμε αν βρίσκεται στο κλειστό σύνολο (closed_set). Αν όχι τότε καλούμαι αναδρομικά ξανά την generate_path για αυτό το current_state και συνεχίζει με αυτόν τον τρόπο η διαδικασία εύρεσης της λύσης. Αν είναι στο κλειστό σύνολο τότε το βγάζουμε από το μέτωπο αναζήτησης και βρίσκουμε το επόμενο state που δεν έχουμε εξετάσει στο μέτωπο αναζήτησης και καλούμαι αναδρομικά ξανά την generate_path και συνεχίζουμε. Τέλος, σχετικά με την astar_helper() που μας βοηθάει να επιλέξουμε το καλύτερο παιδί στο μέτωπο, ουσιαστικά τρέχει επαναληπτικά το μέτωπο αναζήτησης και αποθηκεύσει σε νέα λίστα για κάθε state το score της f τιμής. Παίρνει το min από αυτή την λίστα και τελικά επιστρέφει το state για το οποίο αντιστοιχεί αυτή η min τιμή του f. Και σε αυτό το αρχείο υπάρχουν σε αρκετά σημεία λεπτομερή σχόλια κώδικα ώστε να είναι πιο κατανοητός και λογικός.