Πολυτεχνειο Κρητης
Τμημα Ηλεκτρονικων Μηχανικων και Μηχανικων Υπολογιστων

**ΠΛΗ 417 – Τεχνητή Νοημοσύνη – 2020**

Εαρινό Εξάμηνο 2020 **Εργασία Προγραμματισμού** Παράδοση: **27/5/2020**
Ομαδική (max 2 άτομα/ομάδα)

# Παίζοντας το Παιχνίδι TUC-ANTS

Επιμέλεια εργασίας: Βογιατζής Αντώνης, Νίκος Τρίγκας, Γεώργιος Χαλκιαδάκης

**Βαθμός: 22% του μαθήματος**

## 1 Εισαγωγή

Στα πλαίσια της εργασίας προγραμματισμού θα σχεδιάσετε και θα υλοποιήσετε ένα πρόγραμμα που παίζει το παιχνίδι TUC-ANTS. Στο παρόν κείμενο υπάρχει ένας μικρός οδηγός με τους κανόνες του παιχνιδιού, καθώς και μια μικρή περιγραφή του κώδικα που σας δίνεται ως βάση.

Μετά την παράδοση των εργασιών θα διεξαχθεί ένα πρωτάθλημα TUC-ANTS μεταξύ όλων των agents. Θα δοθεί επιπλέον βαθμολογία (bonus) επί του βαθμού της εργασίας, 5% και 10% στους δύο πρώτους. Η ημερομηνία διεξαγωγής και η ακριβής μορφή του πρωταθλήματος θα ανακοινωθούν έγκαιρα.

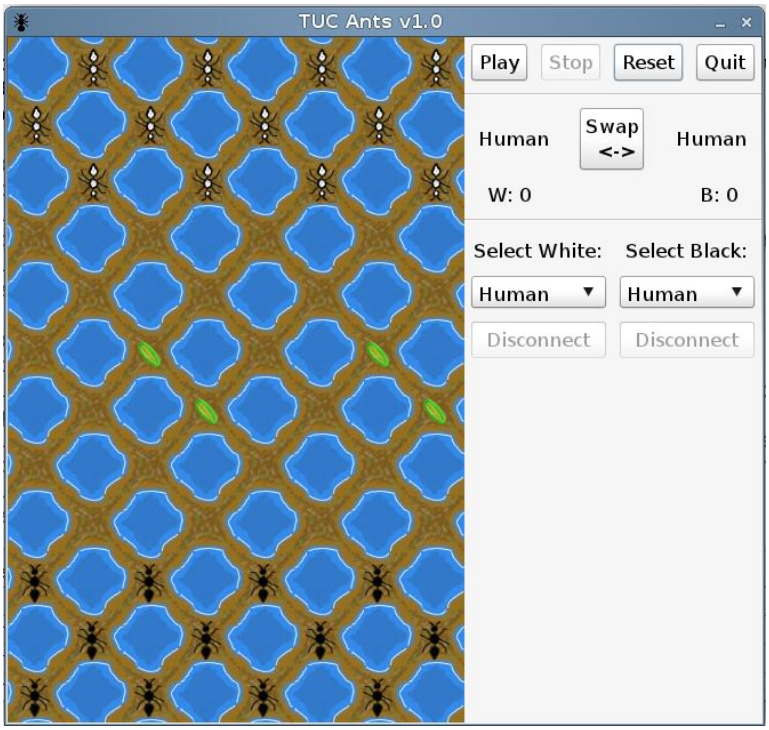
Η εργασία είναι ομαδική και είστε ελεύθεροι να χρησιμοποιήσετε όποια γλώσσα προγραμματισμού θέλετε αρκεί να υπάρχει συμβατότητα επικοινωνίας με τον server του παιχνιδιού που θα σας δοθεί.

## 2 Το Παιχνίδι

Το παιχνίδι TUC-ANTS είναι εμπνευσμένο από το κλασσικό παιχνίδι Ντάμα (Checkers ή Draughts). Η επινόηση του παιχνιδιού της Ντάμας χρονολογείται περίπου στον 13ο μ.Χ. αιώνα, ωστόσο υπάρχουν εικασίες ότι μπορεί να είναι πολύ παλαιότερο. Εδώ θα περιγράψουμε την παραλλαγή TUC-ANTS που επινοήθηκε για τις ανάγκες της εργασίας προγραμματισμού αποβλέποντας σε κάποιο παιχνίδι που εξελίσσεται κάνοντας σταθερά βήματα προόδου προς τη λήξη, ώστε να αποφευχθούν ατέρμονες παρτίδες.

Το παιχνίδι παίζεται σε μία σκακιέρα διαστάσεων $12 \times 8$, όπου είναι τοποθετημένα τετράγωνα «εδάφους» και «νερού», εναλλάξ. Υπάρχουν δύο παίκτες, ο λευκός και ο μαύρος, καθένας από τους οποίους έχει 12 μυρμήγκια με το χρώμα του τοποθετημένα στις τρεις πρώτες γραμμές από την πλευρά του, μόνο στα τετράγωνα με έδαφος. Ολόκληρη η παρτίδα εξελίσσεται μόνο στα τετράγωνα εδάφους της σκακιέρας, κάτι που υπονοεί ότι όλες οι κινήσεις είναι διαγώνιες και όχι οριζόντιες ή κάθετες. Οι παίκτες παίζουν εναλλάξ και, κατά σύμβαση, η πρώτη κίνηση ανήκει στο μαύρο παίκτη.

Οι επιτρεπτές κινήσεις είναι πάντα διαγώνιες και πάντα προς την πλευρά του αντιπάλου. Δεν υπάρχουν κινήσεις προς τα πίσω ή προς τα πλάγια. Κάθε παίκτης, όταν είναι σειρά του να παίξει, πρέπει να κάνει μία κίνηση με ένα μυρμήγκι, εφόσον έχει διαθέσιμες κινήσεις. Ως κίνηση ορίζεται η διαγώνια μετακίνηση προς τα εμπρός (δεξιά ή αριστερά) κατά μία θέση ενός μυρμηγκιού του παίκτη που παίζει. Εφόσον η γειτονική αυτή θέση είναι κενή, το μυρμήγκι τοποθετείται εκεί και δίνεται η σειρά στον αντίπαλο για να παίξει. Αν όμως η γειτονική αυτή θέση είναι κατειλημμένη από μυρμήγκι του αντιπάλου και η αμέσως επόμενη θέση στην ίδια κατεύθυνση είναι κενή, τότε το πιόνι του παίκτη κάνει άλμα πάνω από το μυρμήγκι του αντιπάλου, το «αιχμαλωτίζει», και προσγειώνεται στην κενή θέση αμέσως μετά. Με το όρο «αιχμαλωτίζει» εννοούμε την απομάκρυνση του αντίπαλου μυρμηγκιού από την σκακιέρα του παιχνιδιού. Οι κινήσεις που οδηγούν σε αιχμαλωσία μυρμηγκιών του αντιπάλου έχουν προτεραιότητα. Δηλαδή, εάν σε κάποιο στάδιο του παιχνιδιού ο παίκτης έχει μεταξύ των διαθέσιμων κινήσεων κάποιες που αιχμαλωτίζουν αντίπαλα μυρμήγκια, οφείλει να παίξει μία από αυτές (όποια θέλει). Απλές μετακινήσεις μυρμηγκιών κατά



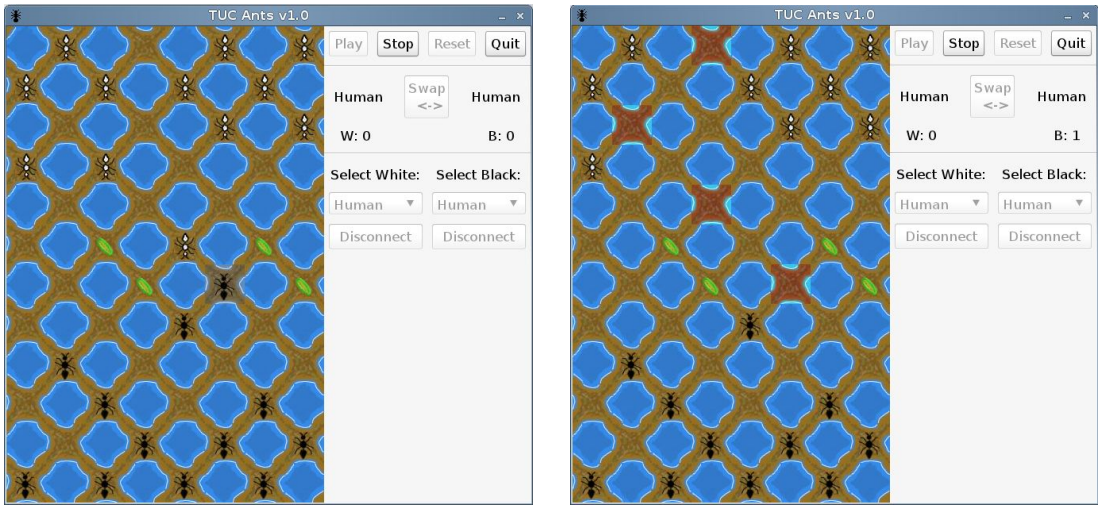

Σχήμα 1: Αρχική κατάσταση του παιχνιδιού TUC-ANTS.

μια θέση διαγώνια επιτρέπονται μόνο όταν δεν υπάρχουν κινήσεις αιχμαλωσίας. Επιπρόσθετα, μετά από μια κίνηση ενός μυρμηγκιού που οδήγησε στην αιχμαλώτιση ενός αντίπαλου μυρμηγκιού, το ίδιο αυτό μυρμήγκι του παίκτη υποχρεούται να προχωρήσει και σε νέα αιχμαλώτιση (εφόσον υπάρχει) στον ίδιο γύρο από τη θέση που προσγειώθηκε, και ούτω καθ' εξής από τη νέα θέση, έως ότου προσγειωθεί σε θέση όπου δεν μπορεί να συνεχίσει με επιπλέον αιχμαλωτίσεις. Αυτό σημαίνει ότι σε έναν γύρο κάποιο μυρμήγκι μπορεί να αιχμαλωτίσει διαδοχικά μέχρι και πέντε αντίπαλα μυρμήγκια.

Κάθε μυρμήγκι του παίκτη που καταφέρνει να φτάσει στην αντίπαλη πλευρά γίνεται «βασιλίσσα», δίνει έναν (1) βαθμό στον παίκτη, και απομακρύνεται από τη σκακιέρα. Επίσης, στα κεντρικά τετράγωνα της σκακιέρας εμφανίζονται στην αρχή του παιχνιδιού πηγές φαγητού με πιθανότητα 1/2. Εάν κάποιο μυρμήγκι βρεθεί σε ένα τέτοιο τετράγωνο, έχει πιθανότητα 1/2 να εκμεταλλευτεί την τροφή και να κερδίσει έναν (1) επιπλέον βαθμό και πιθανότητα 1/2 να καταστρέψει την τροφή και να μην κερδίσει τίποτα.

Το παιχνίδι τερματίζεται όταν έχουν απομακρυνθεί όλα τα μυρμήγκια από τη σκακιέρα (είτε επειδή αιχμαλωτίστηκαν, είτε επειδή έγιναν βασίλισσες). Στόχος του κάθε παίκτη είναι να βγάλει περισσότερες βασίλισσες μαζεύοντας το περισσότερο φαγητό, δηλαδή να πετύχει μεγαλύτερη βαθμολογία από τον αντίπαλο. Ο παίκτης με τη μεγαλύτερη βαθμολογία στο τέλος της παρτίδας ανακηρύσσεται νικητής. Αν η βαθμολογία είναι ίδια, η παρτίδα λήγει με ισοπαλία.

Στο Σχήμα 1 βλέπουμε την αρχική κατάσταση του παιχνιδιού. Τα μυρμήγκια του μαύρου καταλαμβάνουν τις τρεις κάτω σειρές, ενώ εκείνα του λευκού τις τρεις πάνω. Ο προσανατολισμός της σκακιέρας πρέπει να είναι τέτοιος ώστε ο κάθε παίκτης να έχει τετράγωνο εδάφους στην πλευρά του στα αριστερά του. Ο παίκτης που ξεκινάει πρώτος είναι ο μαύρος. Στο Σχήμα 2 βλέπουμε μια ενδιάμεση κατάσταση, όπου παίζει ο μαύρος και η βαθμολογία είναι 0–0. Σ' αυτήν την κατάσταση, ο μαύρος παίκτης πρέπει υποχρεωτικά να παίξει με το μυρμήγκι που είναι χρωματισμένο, καθώς η κίνηση αυτή οδηγεί σε αιχμαλώτιση και μάλιστα τριπλή αιχμαλώτιση. Στο τέλος της κίνησης έχουν αιχμαλωτισθεί τρία λευκά μυρμήγκια και το μαύρο μυρμήγκι έγινε βασιλίσσα, εφόσον έφτασε στην αντίπαλη πλευρά, οπότε και η βαθμολογία διαμορφώνεται σε 0–1 υπέρ του μαύρου.




Σχήμα 2: (α) Πριν από μια τριπλή αιχμαλώτιση από τα μαύρα, (β) Μετά τη τριπλή αιχμαλώτιση.

Συνοψίζοντας, οι κανόνες έχουν ως εξής:

- Οι παίκτες παίζουν εναλλάξ και ο μαύρος παίκτης παίζει πρώτος.
- Ο λευκός παίκτης μετακινεί λευκά μυρμήγκια και ο μαύρος παίκτης μαύρα.
- Ως κίνηση ορίζεται η διαγώνια κίνηση ενός μυρμηγκιού του παίκτη που παίζει.
- Η κίνηση είναι πάντα προς την αντίπαλη πλευρά (δεξιά ή αριστερά).
- Το τετράγωνο προορισμού μετά από μια κίνηση πρέπει να είναι κενό.
- Η απλή κίνηση μετακινεί το μυρμήγκι κατά μία θέση.
- Η κίνηση αιχμαλώτισης γίνεται με άλμα πάνω από ένα αντίπαλο μυρμήγκι.
- Η κίνηση αιχμαλώτισης μετακινεί το μυρμήγκι κατά δύο θέσεις.
- Οι κινήσεις αιχμαλώτισης έχουν προτεραιότητα έναντι των απλών κινήσεων.
- Αν υπάρχουν διαδοχικές αιχμαλωτίσεις με το ίδιο μυρμήγκι, πρέπει να γίνουν όλες μαζί.
- Τα μυρμήγκια που αιχμαλωτίζονται απομακρύνονται από τη σκακιέρα διαπαντός.
- Το τετράγωνο φαγητού δίνει ένα βαθμό στο πρώτο μυρμήγκι που θα το επισκευθεί, με πιθανότητα 1/2.
- Κάθε μυρμήγκι που φτάνει στην απέναντι πλευρά γίνεται βασίλισσα και απομακρύνεται.
- Μια παρτίδα λήγει όταν δεν υπάρχουν άλλα μυρμήγκια (ή διαθέσιμες κινήσεις).
- Νικητής είναι ο παίκτης με τις περισσότερες βασιλίσσες (μεγαλύτερη βαθμολογία).

# 3 Διαδικαστικά

Η διαδικασία που πρέπει να ακολουθήσετε για να ολοκληρώσετε την εργασία είναι η παρακάτω:

- Διαβάστε προσεκτικά το παρόν κείμενο και τον κώδικα που το συνοδεύει.
- Βεβαιωθείτε ότι μπορείτε να τρέξετε τον κώδικα και κατανοήστε τη λειτουργία του.
- Υλοποιήστε τον αλγόριθμο αναζήτησης minimax .

- Υλοποιήστε μια καλή συνάρτηση αξιολόγησης και μια μέθοδο αποκοπής.
- Βελτιώστε την αναζήτηση υλοποιώντας α–β pruning.
- Βελτιώστε περαιτέρω την αναζήτηση με singular extensions, forward pruning, κλπ.
- Βεβαιωθείτε ότι το πρόγραμμά σας συνεργάζεται με τον server χωρίς προβλήματα.
- Παραδώστε την εργασία σας μέχρι την ημερομηνία παράδοσης.
- Παραδώστε μια σύντομη αναφορά (5 σελίδες max) για τις τεχνικές που χρησιμοποιήσατε και υλοποιήσατε.
- Παρουσιάστε την εργασία σας στο διδάσκοντα και στους βοηθούς.
- Λάβετε μέρος στο πρωτάθλημα και κερδίστε με τον παίκτη σας!

# 4 Εκπόνηση

Ο αλγόριθμος αναζήτησης minimax και το α–β pruning καθώς και άλλες τεχνικές για παιχνίδια έχουν καλυφθεί εκτενώς στο μάθημα. Επίσης, τα βιβλία του μαθήματος περιέχουν όλες τις λεπτομέρειες και τον ψευδοκώδικα των αλγορίθμων αναζήτησης που χρειάζεστε για την εργασία.

Σκεφτείτε τον τρόπο με τον οποίο θα δημιουργήσετε το δέντρο της αναζήτησης. Η μέθοδος που θα ακολουθήσετε μπορεί να επηρεάσει σημαντικά την ταχύτητα του παίκτη σας. Προφανώς, δεν θα υπάρχει αρκετός χρόνος για να να ψάξει το πρόγραμμα σας ολόκληρο το δέντρο αναζήτησης για την καλύτερη κίνηση. Έτσι θα πρέπει να ψάχνετε για την καλύτερη δυνατή κίνηση σε λογικά πλαίσια πραγματικού χρόνου.

Ακολουθούν κάποια σχόλια και προτάσεις για την εκπόνηση της εργασίας.

- Βεβαιωθείτε ότι μετά την υλοποίηση του α–β pruning η αναζήτησή σας καταλήγει ακριβώς στις ίδιες τιμές με τον minimax. Θα ήταν σκόπιμο να έχετε ένα μηχανισμό για να ενεργοποιείτε και να απενεργοποιείτε το α–β pruning κατά βούληση.
- Ο απλός αλγόριθμος α–β pruning εξετάζει τα παιδιά κάθε κόμβου με τη σειρά δημιουργίας, αλλά μπορείτε να υλοποιήσετε κάτι πιο έξυπνο. Όπως ξέρετε, η σειρά με την οποία γίνονται επεκτάσεις κόμβων έχει μεγάλη επιρροή στην αποτελεσματικότητα του κλαδέματος.
- Μια αρχική (απλή) συνάρτηση αξιολόγησης κατάστασης (state) που μπορείτε να χρησιμοποιήσετε σε συνδυασμό με κάποια μέθοδο αποκοπής είναι η παρακάτω:

  $$V(\text{state}) = |\text{my (queens + pieces + gath. food)}| - |\text{opponent's (queens + pieces + gath. food)}|$$

- Μπορείτε να χρησιμοποιήσετε οποιαδήποτε τεχνική βελτίωσης βρείτε διαθέσιμη στο διαδίκτυο ή σε βιβλία αρκεί να εξηγήσετε στην αναφορά σας τι κάνει, πως το κάνει, γιατί την επιλέξατε και πως την προσαρμόσατε στον κώδικά σας.
- Στην αναφορά σας περιγράψτε τις ιδιαιτερότητες του κώδικά σας και τις τυχόν προεκτάσεις ή έξυπνες λύσεις που υλοποιήθηκαν. Θα πρέπει απαραίτητα να υπάρχουν αναφορές των πηγών σας· τυφλή αντιγραφή (plagiarism) θα επιφέρει άμεσο μηδενισμό στο μάθημα.

# 5 Υλοποίηση

Μπορείτε να γράψετε τον κώδικα σας σαν συνέχεια του κώδικα TUC-ANTS που σας δίνεται (ενότητα Υλικό Εργαστηρίου στο courses). Εκεί υπάρχει υλοποιημένη όλη η απαραίτητη λειτουργικότητα του παιχνιδιού, ώστε να ελαττωθεί ο φόρτος υλοποίησης. Συγκεκριμένα, περιλαμβάνεται ο κώδικας ενός βασικού παίκτη (client) και του server του παιχνιδιού σε C.

Ο κώδικας παρέχει τις βασικές δομές δεδομένων και τους βασικούς μηχανισμούς για το παιχνίδι. Επίσης, υπάρχουν υλοποιημένες και βοηθητικές συναρτήσεις (π.χ. έλεγχος αν μια κίνηση είναι έγκυρη) χωρίς αυτό όμως να σημαίνει ότι δεν μπορεί κάποιος να υλοποιήσει τις δικές του συναρτήσεις. **Προσοχή!** Δεν επιτρέπονται αλλαγές στο πρωτόκολλο επικοινωνίας. Αν αλλάξετε τον τρόπο επικοινωνίας, η συμμετοχή σας στο τουρνουά δεν θα είναι εφικτή.

## 5.1 Υποδομή

Οι δομές που χρησιμοποιούνται στην υπάρχουσα υλοποίηση βρίσκονται στα αρχεία `board.h` και `move.h`. Στο πρώτο αρχείο ορίζεται η δομή `Position` που αναπαριστά μια πλήρη κατάσταση του παιχνιδιού. Η δομή αυτή εμπεριέχει έναν δισδιάστατο πίνακα `board` μεγέθους `BOARD_ROWS`×`BOARD_COLUMNS` που αναπαριστά την κατάσταση της σκακιέρας (0 για λευκό μυρμήγκι, 1 για μαύρο μυρμήγκι, 2 για κενό τετράγωνο, 3 για τετράγωνο με φαγητό), έναν μονοδιάστατο πίνακα `score` με την τρέχουσα βαθμολογία (μία τιμή για κάθε παίκτη) και μία μεταβλητή `turn` που δηλώνει την ταυτότητα του παίκτη (0 για τον λευκό, 1 για τον μαύρο) που καλείται να παίξει στην εν λόγω σκακιέρα. Στο δεύτερο αρχείο ορίζεται η δομή `Move` που χρησιμοποιείται για να αναπαριστά μια κίνηση του παιχνιδιού. Σ΄ αυτήν περιέχονται μέσα σ΄ έναν δισδιάστατο πίνακα `tile` οι συντεταγμένες της θέσης από την οποία ξεκινάει το πιόνι στο οποίο αναφερόμαστε, καθώς και όλες οι συντεταγμένες των θέσεων από τις οποίες θα περάσουμε μέχρι να καταλήξουμε στις συντεταγμένες της θέσης προορισμού (συμπεριλαμβανομένων). Επίσης, στη δομή περιέχεται και η ταυτότητα `color` του παίκτη στον οποίο ανήκει η κίνηση. Οι θέσεις μετρώνται από το πάνω αριστερά τετράγωνο όπως εικονίζεται η σκακιέρα στο GUI, ξεκινώντας από το $(0, 0)$.

Εδώ αξίζει να σημειωθεί ότι το μέγεθος του πίνακα που αποθηκεύει τις συντεταγμένες της κίνησης είναι `tile[2][6]`, εφόσον μπορούν να γίνουν πέντε το πολύ διαδοχικά άλματα. Όπως γίνεται αντιληπτό, πολλές κινήσεις (σχεδόν όλες) δεν θα γεμίζουν ολόκληρο αυτόν τον πίνακα. Για να δείξουμε σ΄ αυτήν την περίπτωση ότι η κίνησή μας ολοκληρώνεται μετά από κάποιες συντεταγμένες, πρέπει να ορίσουμε το `row` της επόμενης θέσης ως `-1`. Για παράδειγμα, έστω ότι θέλουμε να αποθηκεύσουμε την κίνηση από τη θέση $(5, 0)$ στην $(4, 1)$ στη συγκεκριμένη δομή. Τα `tile[0][0]` και `tile[1][0]` θα είναι οι τιμές των `row` και `column` αντίστοιχα της θέσης απ΄ όπου θα ξεκινήσει η κίνηση, δηλαδή `5` και `0`. Τα `tile[0][1]` και `tile[1][1]` θα είναι οι τιμές των `row` και `column` αντίστοιχα της θέσης στην οποία θέλουμε να μετακινηθούμε, δηλαδή `4` και `1`. Τέλος, επειδή δεν έχουμε γεμίσει τον πίνακα, αλλά έχουμε τελειώσει την κίνησή μας, θα πρέπει να θέσουμε το `tile[0][2]` στην τιμή `-1`. Σε περίπτωση που δεν έχουμε καμία κίνηση διαθέσιμη, τότε πρέπει να στείλουμε την κενή (null) κίνηση, η οποία είναι στην ουσία είναι μία κίνηση με `tile[0][0]` ίσο με `-1`.

Είστε ελεύθεροι να κάνετε όποιες αλλαγές θέλετε σ΄ αυτές τις δομές, ακόμα και να τις αντικαταστήσετε με δικές σας. Πρέπει όμως να προσέχετε ώστε οι αλλαγές σας να μην επηρεάσουν την επικοινωνία του παίκτη σας με τον server του παιχνιδιού.

## 5.2 Χρήσιμες Διεργασίες

Στο αρχείο `board.h` υπάρχουν δηλωμένες μερικές χρήσιμες διεργασίες και συναρτήσεις ελεύθερες προς χρήση τις οποίες και περιγράφουμε συνοπτικά παρακάτω.

- `void doMove( Position * pos, Move * moveToDo )`
  Η διεργασία αυτή, εκτελεί την κίνηση που ορίζεται μέσα στο `moveToDo` πάνω στην κατάσταση που ορίζεται στο `pos`. Πρέπει να χρησιμοποιείται μόνο όταν είμαστε σίγουροι ότι η κίνηση είναι σωστή, καθώς δεν εμπεριέχει ελέγχους.

- `int canJump( char row, char col, char player, Position * pos )`
  Αυτή η συνάρτηση ελέγχει αν από τη θέση (`row`, `col`) της κατάστασης pos είναι εφικτή κίνηση άλματος για τον παίκτη `player`. Επιστρέφει 0 αν δεν μπορεί να υπάρξει άλμα, 1 αν υπάρχει άλμα μόνο προς τα αριστερά της σκακιέρας, 2 αν υπάρχει μόνο προς τα δεξιά και 3 αν υπάρχει και προς τις δύο κατευθύνσεις. Αξίζει να σημειωθεί ότι η θέση (`row`, `col`) δεν χρειάζεται να περιέχει απαραίτητα μυρμήγκι, πρέπει όμως να είμαστε σίγουροι ότι

αποτελεί έγκυρη θέση της σκακιέρας μας. Συνεπώς, θα μπορούσατε να χρησιμοποιήσετε τη συνάρτηση για ελέγχους σε πολλαπλά άλματα πριν αυτά γίνουν στην σκακιέρα.

- `int canJumpTo( char row, char col, char player, Position * pos, char rowDest, char colDest )`
  Στην ουσία είναι ίδια με την παραπάνω συνάρτηση, μόνο που δέχεται ακόμη μία θέση, την (`rowDest`, `colDest`) και μας επιστρέφει 1, αν το άλμα στο συγκεκριμένο τετράγωνο είναι εφικτό ή 0 σε αντίθετη περίπτωση. Δεν γίνεται έλεγχος αν η (`rowDest`, `colDest`) αποτελεί έγκυρη θέση της σκακιέρας, οπότε χρησιμοποιήστε τη με προσοχή.

- `int canMove( Position * pos, char player )`
  Αυτή η συνάρτηση, λαμβάνοντας υπόψη την σκακιέρα του `pos` και τον παίκτη player επιστρέφει 1 αν ο παίκτης αυτός έχει τουλάχιστον μία νόμιμη κίνηση στην εν λόγω σκακιέρα, ή 0 σε περίπτωση που δεν έχει καμία.

- `int isLegal( Position * pos, Move * moveToCheck )`
  Η συνάρτηση αυτή επιστρέφει 1, αν η κίνηση `moveToCheck` είναι νόμιμη στην σκακιέρα `pos`, διαφορετικά επιστρέφει 0. Σε αντίθεση με τις `canJump` και `canJumpTo`, πραγματοποιεί όλους τους απαραίτητους ελέγχους.

## 5.3 Επικοινωνία

Για να μπορούν δύο προγράμματα να παίξουν μεταξύ τους, έχει υλοποιηθεί μια απλή client–server αρχιτεκτονική. Ο server επικοινωνεί με δύο clients που αντιπροσωπεύουν τους δύο παίκτες. Οι clients μπορούν να βρίσκονται στο ίδιο ή ακόμα και σε διαφορετικά μηχανήματα. Ο server αρχικά ενημερώνει τους παίκτες για την κατάσταση του παιχνιδιού (σκακιέρα, βαθμολογία, παίκτης) και για το χρώμα τους. Στη συνέχεια, σε κάθε στιγμή ο server στέλνει στον ένα από τους δύο clients (σ΄ αυτόν που έχει σειρά να παίξει) την τελευταία κίνηση του αντιπάλου του. Ο client πρέπει να αποφασίσει την κίνηση του **μέσα σε λογικά χρονικά πλαίσια** και να στείλει ένα μήνυμα με την κίνησή του, πίσω στο server. Ο κάθε client οφείλει να ενημερώνει εσωτερικά την κατάσταση του παιχνιδιού βάσει της αρχικής κατάστασης και των κινήσεων που έχουν γίνει από τον ίδιο και τον αντίπαλο. Όλες οι ρουτίνες επικοινωνίας μεταξύ client και server είναι υλοποιημένες. **Δεν επιτρέπονται αλλαγές σ΄ αυτές τις ρουτίνες!!!** Υπάρχει ένας απλός client στο αρχείο `client.c` που παίζει με τυχαίες κινήσεις. Αυτός ο κώδικας μπορεί να χρησιμοποιηθεί ως βάση για να υλοποιηθεί ο παίκτης σας. Λογικά, δεν θα χρειαστεί να πειράξετε κανένα άλλο αρχείο.

Σε περίπτωση που επιθυμείτε να δουλέψετε σε κάποια άλλη γλώσσα πέραν της C, πρέπει να υλοποιήσετε το ίδιο πρωτόκολλο επικοινωνίας στην πλευρά του δικού σας client ώστε να είστε συμβατοί με τον server. Σ΄ αυτήν την περίπτωση θα σας βοηθήσει ο υπάρχων κώδικας, οι βοηθοί του μαθήματος και η λεπτομερής παρουσίαση της επικοινωνίας που ακολουθεί.

Μέσα στο αρχείο `comm.h` ορίζονται επτά (7) τύποι μηνυμάτων που αποστέλλονται μεταξύ του server και του client και καθορίζουν την επικοινωνία, καθώς και οι συναρτήσεις που αποστέλλουν και λαμβάνουν δεδομένα. Όλα τα μηνύματα που ανταλλάσσονται, εκτός από αυτό της μεταφοράς του ονόματος του παίκτη, έχουν σταθερό μήκος. Ο server είναι αυτός που έχει τον απόλυτο έλεγχο της επικοινωνίας και ο ρόλος του client είναι να υπακούει και να αντιδρά σωστά στις όποιες οδηγίες του. Μετά τη δημιουργία σύνδεσης μεταξύ server και client (μετά το accept), ο client πρέπει να περιμένει την αποδοχή κάποιου μηνύματος (από τα επτά που αναφέρθηκαν παραπάνω). Η αντίδρασή του ανάλογα με το μήνυμα του server πρέπει να είναι η ακόλουθη:

- `NM_NEW_POSITION`
  Ο server μας ενημερώνει ότι θα μας αποστείλει νέα δομή `Position`. Η αντίδρασή μας πρέπει να είναι η εκτέλεση της διεργασίας `void getPosition( Position * posToGet, int mySocket )`. Στην ουσία έτσι μεταφέρεται όλη η δομή του `Position`, μαζί με την τρέχουσα βαθμολογία και το χρώμα του παίκτη που έχει σειρά να παίξει. Το μήνυμα έχει τη μορφή `2 0 2 0 2 0 ... 1 2 0 0 1`, όπου τα πρώτα `BOARD_ROWS*BOARD_COLUMNS` bytes είναι τα περιεχόμενα της σκακιέρα μας, τα επόμενα δύο η τρέχουσα βαθμολογία του λευκού και μαύρου παίκτη αντίστοιχα και τέλος η ταυτότητα του παίκτη που έχει την κίνηση.

- `NM_COLOR_W`
  Ο server μάς ενημερώνει ότι το χρώμα μας από εδώ και στο εξής είναι το λευκό. Το μόνο που έχουμε να κάνουμε είναι να ενημερώσουμε τον παίκτη μας γι΄ αυτήν την αλλαγή. Στον έτοιμο κώδικα αυτό αποθηκεύεται στη μεταβλητή `myColor` μέσα στο `client.c`.

- `NM_COLOR_B`
  Αντίστοιχα με το παραπάνω μήνυμα, αλλάζουμε το χρώμα του παίκτη μας σε μαύρο.

- `NM_REQUEST_MOVE`
  Ο server ζητάει την κίνησή μας. Πρέπει να αποφασίσουμε ποια θα είναι αυτή και αφού γεμίσουμε τη δομή της κίνησης, να την αποστείλουμε με την βοήθεια της συνάρτησης `void sendMove( Move * moveToSend, int mySocket )`. Στην ουσία μεταφέρεται ολόκληρος ο δισδιάστατος πίνακας `tile` της δομής `Move`. Αν θέλουμε να στείλουμε την κίνηση (5,0) στο (4,1) το μήνυμα θα έχει τη μορφή `5 0 4 1 -1 X X X`. Το χρώμα του παίκτη μας δεν αποστέλλεται, καθώς ο server ξέρει πολύ καλά ποιοι είμαστε! Προφανώς, δεν μπορείτε να στείλετε illegal move ως δήθεν αντίπαλος...! Στη συνέχεια, πριν επεξεργαστούμε το επόμενο μήνυμα του server, θα πρέπει να ενημερώσουμε την σκακιέρα με την κίνησή μας.

- `NM_PREPARE_TO_RECEIVE_MOVE`
  Ο server μας ενημερώνει ότι ο αντίπαλός μας έχει επιλέξει την κίνησή του και τη λαμβάνουμε εκτελώντας τη συνάρτηση `void getMove( Move * moveToGet, int mySocket )`. Στη συνέχεια, πριν αναζητήσουμε το επόμενο μήνυμα, πρέπει να παίξουμε αυτήν την κίνηση με το αντίπαλο χρώμα στην σκακιέρα μας για να την ενημερώσουμε.

- `NM_REQUEST_NAME`
  Ο server θέλει να μάθει το όνομά μας. Αυτό αποθηκεύεται στο string `agentName` μέσα στο `client.c`. Βάλτε το όνομα που επιθυμείτε, έχοντας υπόψη τον περιορισμό των `MAX_NAME_LENGTH` χαρακτήρων (16 στην περίπτωσή μας). Η αποστολή γίνεται με χρήση της `void sendName( char textToSend[ MAX_NAME_LENGTH +1 ], int mySocket )`. Το μήνυμα που αποστέλλεται έχει μπροστά ένα byte που δηλώνει το μέγεθος του ονόματος. Για παράδειγμα, αν θέλουμε να στείλουμε το `test`, το μήνυμά μας θα είναι `4 t e s t`.

- `NM_QUIT`
  Ο server μας ζητάει να αποσυνδεθούμε και να αναστείλουμε τη λειτουργία μας, συνεπώς θα κλείσουμε το socket της επικοινωνίας και θα τερματίσουμε.

Μετά το πέρας της επεξεργασίας κάθε μηνύματος (εκτός από αυτό του τερματισμού) περιμένουμε νέο μήνυμα από τον server για να συνεχίσουμε.

## 5.4 Client

Ο client είναι σχεδιασμένος για να συνδέεται στον server και να παίζει το ρόλο ενός παίκτη. Ο server είναι αυτός που αποφασίζει ποια πλευρά θα παίξει ο κάθε ένας client. Για να τρέξει ο C client απλά γράφουμε:

```
./client [-i server-ip] [-p server-port].
```

- `server-ip`: ορίζει την ip address του μηχανήματος στο οποίο τρέχει ο server, π.χ. `147.27.14.5`, (default=`127.0.0.1` – το ίδιο μηχάνημα με τον client).

- `server-port`: ορίζει το port στο οποίο θα συνδεθεί ο client (default=6001).

### 5.5 Server

Το γραφικό περιβάλλον guiServer που δίδεται επιτρέπει τη διεξαγωγή παρτίδων TUC-ANTS μεταξύ ενός ενσωματωμένου παίκτη που παίζει εντελώς τυχαία, διαφόρων εξωτερικών παικτών που μπορούν να συνδεθούν μέσω δικτύου, και χρηστών (ανθρώπων). Ο χρήστης έχει την δυνατότητα να αλλάξει τους παίκτες κατά την διάρκεια μιας παρτίδας, χρησιμοποιώντας οποιεσδήποτε από τις διαθέσιμες επιλογές παικτών. Επίσης, μπορεί να αποσυνδέσει και να συνδέσει όποιον εξωτερικό παίκτη θέλει. Οπτικά, ο χρήστης υποβοηθείται με το να επισημαίνονται με έντονο χρώμα πάνω στην σκακιέρα το μυρμήγκι που έχει επιλεγεί για κίνηση καθώς και η τελευταία κίνηση του κάθε παίκτη. Εναλλακτικά, μπορεί να χρησιμοποιηθεί ο ascii server (`./server -h`).

## 6 Επίλογος

Όπως θα έχετε διαπιστώσει, η εργασία είναι αρκετά ανοικτή με την έννοια ότι σας δίνει την ελευθερία να κάνετε πολλές αυθαίρετες επιλογές και να επικεντρωθείτε στα σημεία που σας ενδιαφέρουν περισσότερο. Επίσης, σας δίνει τη δυνατότητα για μελλοντικές επεκτάσεις με μεθόδους μηχανικής μάθησης ώστε ο παίκτης να βελτιώνει την απόδοσή του σταδιακά. Θα είμαστε στη διάθεσή σας για ό,τι προκύψει. Ξεκινήστε σήμερα κιόλας, βάλτε το μυαλό και τη φαντασία σας να δουλέψουν και διασκεδάστε ασκώντας τη δημιουργικότητά σας!

**Καλή επιτυχία!!!**