ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικόν και Καποδιστριακόν Πανεπιστήμιον Αθηνών
ΙΔΡΥΘΕΝ ΤΟ 1837

Εισαγωγή στον Προγραμματισμό

Εργασία #3

Ιανουάριος 2024

Στόχος της τρίτης και τελευταίας εργασίας είναι να εξοικειωθούμε με την υλοποίηση μεγαλύτερων και πιθανώς πιο πολύπλοκων προγραμμάτων, ολοκληρώνοντας την εισαγωγή μας στην γλώσσα C. Συγκεκριμένα, στοχεύουμε σε εμπειρία με τα ακόλουθα:

1. Γραφή μεγαλύτερων προγραμμάτων, αποτελούμενα από πολλά αρχεία.
2. Χρήση όλων των βασικών προγραμματιστικών δομών.
3. Αναζήτηση σε χώρο καταστάσεων.

Υποβολή Εργασίας. Όλες οι υποβολές των εργασιών θα γίνουν μέσω GitHub και συγκεκριμένα στο github.com/progintro [2]. Προκειμένου να ξεκινήσεις, μπορείς να δεχτείς την άσκηση με αυτήν την: πρόσκληση [3]. Σε αντίθεση με τις προηγούμενες εργασίες, αυτή είναι **προαιρετικά ομαδική με ομάδες μέχρι 2 άτομα**. Εάν θέλετε να δουλέψετε ως ομάδα, πρέπει ένας/μία από εσάς να αποδεχτεί την πρόσκληση και να δημιουργήσει την ομάδα σας **με συγκεκριμένο όνομα**. Αφού η ομάδα δημιουργηθεί ο/η συνεργάτης σας μπορεί να αποδεχτεί την πρόσκληση και να επιλέξει να προστεθεί στην ομάδα σας. **Προσοχή: μην επιλέξετε λάθος ομάδα, βεβαιωθείτε ότι κάνατε join την σωστή.** Αν επιλέξετε να εργαστείτε ως ομάδα, το repository για την εργασία θα είναι κοινό ανάμεσά σας και θα μπορέσετε να συνεργαστείτε σαν επαγγελματίες software engineers - προσοχή στα conflicts! Τέλος, για να είναι ξεκάθαρο ποια άτομα συνεργάστηκαν για την εργασία, **κάθε repository πρέπει να έχει ένα AUTHORS αρχείο** το οποίο θα περιέχει τα στοιχεία σας στην ακόλουθη μορφή:

```
$ cat AUTHORS
sdi2400998,mourmourakis-2006,ΘΑΝΟΣ ΜΟΥΡΜΟΥΡΑΚΗΣ
sdi2400999,xifias-2006,ΚΩΣΤΑΣ ΞΙΦΙΑΣ
```

δηλαδή μία γραμμή για κάθε άτομο, με πρώτο το sdi σας, μετά το github username και τέλος το όνομά σας. **Υποβολές χωρίς σωστό AUTHORS αρχείο δεν θα εξεταστούν**. Αυτό ισχύει και για ατομικές υποβολές.

1. Νέα Μηχανή Σκακιού (chess engine - 100 Μονάδες)

Τα περισσότερα παιχνίδια έχουν παρεμφερή δομή: (1) το παιχνίδι ξεκινάει σε μια αρχική κατάσταση, (2) ένας από τους παίκτες παίζει πρώτος επιλέγοντας μια από τις δυνατές κινήσεις που έχει στην διάθεσή του αλλάζοντας την κατάσταση του παιχνιδιού και (3) στην συνέχεια δίνει την σειρά του στον επόμενο παίκτη. Η εναλλαγή των παικτών συνεχίζεται μέχρι κάποιο κριτήριο (διαφορετικό για κάθε παιχνίδι) να μας υποδείξει ότι το παιχνίδι τελείωσε με νίκη, ισοπαλία ή ήττα για την κάθε πλευρά.

Η πολυπλοκότητα του κάθε παιχνιδιού [12] έχει άμεση σχέση με δύο παραμέτρους: (1) πόσες δυνατές κινήσεις έχει σε κάθε γύρο ο παίκτης και (2) πόσους γύρους μπορεί να έχει ένα παιχνίδι. Αυτές οι δύο παράμετροι μας επιτρέπουν να υπολογίσουμε τον χώρο καταστάσεων του παιχνιδιού (δηλαδή σε πόσες διαφορετικές καταστάσεις μπορεί να βρεθεί το παιχνίδι μας). Ένας τρόπος να φανταστούμε αυτόν τον χώρο καταστάσεων είναι σαν ένα m-αδικό δέντρο βάθους n, όπου m οι επιλογές μας σε κάθε θέση και n ο αριθμός των επιλογών (γύρων) του παιχνιδιού. Ένα m-αδικό δέντρο βάθους n έχει περίπου m^n διαφορετικές καταστάσεις—όμως πόσο γρήγορα μεγαλώνει ένα τέτοιο δέντρο; Για να δούμε μερικά παραδείγματα παιχνιδιών.

Ίσως έχετε παίξει τρίλιζα. Στην τρίλιζα ξεκινάμε με ένα πλέγμα 3x3 και στην αρχή έχουμε 9 επιλογές για το που θα τοποθετήσουμε το ο ή το x. Στην χειρότερη περίπτωση, μετά από 9 γύρους το παιχνίδι μας θα έχει τελειώσει καθώς και οι 9 θέσεις θα έχουν γεμίσει. Επομένως, ένα απλοϊκό άνω όριο για τον χώρο καταστάσεων είναι $9^9 = 387.420.489$ (στην πραγματικότητα ο χώρος καταστάσεων είναι πολύ μικρότερος—γύρω στις 765 [4]—καθώς σε κάθε κίνηση μειώνεται ο αριθμός των επιλογών μας, τα παιχνίδια μπορεί να ολοκληρωθούν πολύ πιο γρήγορα από 9 κινήσεις και κάποια παιχνίδια καταλήγουν στην ίδια κατάσταση με διαφορετική αρχική σειρά κινήσεων). Παρόλο που ο αριθμός αυτός είναι αρκετά μεγάλος για έναν άνθρωπο - θα μας έπαιρνε πολλές ώρες να παίξουμε όλες τις δυνατές παρτίδες τρίλιζας και να υπολογίσουμε τα αποτελέσματά τους - ο ίδιος υπολογισμός μπορεί να γίνει σε λιγότερο από ένα δευτερόλεπτο από έναν σύγχρονο υπολογιστή. Εάν διατρέξουμε ολόκληρο τον χώρο καταστάσεων ενός παιχνιδιού τότε μπορούμε να πούμε στην επιστήμη των υπολογιστών ότι το "λύσαμε" [17], δηλαδή για οποιαδήποτε θέση του παιχνιδιού μπορούμε να πούμε με βεβαιότητα ποια είναι η βέλτιστη κίνηση.

Χάρη στην εξέλιξη των υπολογιστών, έχουμε καταφέρει να λύσουμε αρκετά δημοφιλή παιχνίδια με μεγάλους χώρους καταστάσεων. Για παράδειγμα η "ντάμα" (Checkers / Draughts) με χώρο καταστάσεων γύρω στο 10^{20} έχει λυθεί επίσημα από το 2007, όπου ερευνητές κατάφεραν να ολοκληρώσουν τον υπολογισμό ύστερα από δεκαετίες προσπάθειας [8]. Φυσικά κανένας δεν θυμάται την καλύτερη κίνηση για όλες τις 10^{20} καταστάσεις οπότε παρόλο που το πρόβλημα λύθηκε οι άνθρωποι μπορούν ακόμα να απολαμβάνουν αυτό το παιχνίδι σε παρτίδες μεταξύ τους.

Ένα παιχνίδι που μας διαφεύγει μέχρι στιγμής είναι το σκάκι (chess) [5]. Ένα από τα αρχαιότερα και πιο δημοφιλή παιχνίδια στρατηγικής στον κόσμο, έχει τις ρίζες του στη βορειοδυτική Ινδία του 6ου αιώνα. Θεωρούνταν ανέκαθεν τόσο πολύπλοκο

Εικόνα 1: Το 1996, ο Garry Kasparov αντιμετώπισε για πρώτη φορά τον υπερυπολογιστή Deep Blue της IBM σε ένα αγώνα σκακιού. Παρά την ήττα του σε ένα παιχνίδι, ο Kasparov κέρδισε πειστικά τον αγώνα με σκορ 4-2. Την επόμενη χρονιά (1997), αντιμετώπισε την ανανεωμένη έκδοση του Deep Blue και αυτήν την φορά υπερίσχυσε ο Deep Blue 3,5-2,5, σηματοδοτώντας την πρώτη φορά που ένας υπολογιστής κέρδισε έναν παγκόσμιο πρωταθλητή στο σκάκι σε επίσημο αγώνα. Η εξέλιξη της τεχνολογίας υπήρξε ραγδαία από τότε και σήμερα ένα απλό chess app στο κινητό μας μπορεί να κερδίσει τον παγκόσμιο πρωταθλητή.

και πνευματικό που ιστορικά συνδέθηκε με τη στρατηγική, τη φιλοσοφία, ακόμα και την εκπαίδευση ηγετών.

Για δεκαετίες ερευνητές στον χώρο της τεχνητής νοημοσύνης προσπαθούσαν να φτιάξουν συστήματα που έπαιζαν σκάκι και παρόλες τις επιτυχίες στον χώρο, τα συστήματα δεν ήταν σε θέση να φτάσουν το επίπεδο των πρωταθλητών στον χώρο. Όλα αυτά μέχρι το 1997. Το 1997, ο Deep Blue—ένας από τους ισχυρότερους υπερυπολογιστές της εποχής—κέρδισε τον τότε παγκόσμιο πρωταθλητή Garry Kasparov σε μια πολυδιαφημισμένη κόντρα [9] (Σχήμα 1). Υπήρξαν τότε πολλές προβλέψεις πως το σκάκι θα έπαυε να είναι ενδιαφέρον και θα σταματούσε ο κόσμος να ασχολείται. Το μέλλον όμως διέψευσε τις προβλέψεις και το σκάκι σήμερα γνωρίζει πρωτόγνωρη άνθηση με περισσότερα μέλη και παρτίδες από ποτέ (κάποιες παρτίδες λαμβάνουν χώρα ακόμα και εν ώρα μαθήματος). Το παιχνίδι συνεχίζει να έχει φίλους πέρα από σύνορα, ηλικίες και πολιτισμούς. Μάλιστα, έχουν βγει πολλές καινούριες εκδοχές σκακιού που συνδυάζουν ανθρώπους και υπολογιστές προκειμένου να παίξουν Advanced (Centaur) Chess [6].

Γιατί όμως το σκάκι δεν έχει λυθεί ακόμα; Ένας από τους λόγους είναι το τεράστιο μέγεθος του χώρου καταστάσεων (με ένα πλέγμα μόλις 8x8!). Ο διάσημος

μαθηματικός Claude Shannon εκτίμησε τις δυνατές παρτίδες στο 10^{120} και τις πιθανές θέσεις της σκακιέρας στο 10^{40} [16]—μεγέθη που είναι ακόμα άπιαστα ακόμα και σήμερα για τα ισχυρότερα υπολογιστικά συστήματα. Πως επομένως εξερευνούν οι μηχανές σκακιού έναν τέτοιο χώρο σήμερα; Η απάντηση είναι με τεχνικές αναζήτησης / ευριστικές [14, 15] και με ιδιαίτερα αποδοτικές υλοποιήσεις—οι περισσότεροι πυρήνες μηχανών σκακιού σήμερα συνεχίζουν να γράφονται σε γλώσσες όπως η C και η C++ λόγω των προτερημάτων που τους προσφέρει σε ταχύτητα (οι ίδιες υλοποιήσεις μπορούν να γίνουν compile και να τρέξουν σε browsers, προκειμένου να μπορούν να χρησιμοποιηθούν και από web εφαρμογές—όπως θα δούμε σε αυτήν την εργασία).

Πόσο εύκολο είναι να φτιάξουμε μια μηχανή σκακιού σήμερα; Μπορεί να κερδίσει σχετικά απλούς αντιπάλους που απλά παίζουν τυχαίες κινήσεις; Αυτό θα είναι και το αντικείμενο αυτής της άσκησης, όπου καλείστε να υλοποιήσετε τον πυρήνα μιας μηχανής σκακιού. Ευτυχώς, δεν χρειάζεται να ξεκινήσετε από το μηδέν, μπορείτε να βρείτε στο διαδίκτυο πολλές πληροφορίες για το πως δομούνται σκακιστικές μηχανές καθώς και ποιες βελτιστοποιήσεις είναι δημοφιλείς [1]. Οι τεχνικές προδιαγραφές ακολουθούν.

Τεχνικές Προδιαγραφές

- Repository Name: progintro/hw3-<YourTeamName>
- README Filepath: README.md
- Όλα τα αρχεία κώδικα (.c και .h) που θα υλοποιήσετε πρέπει να τοποθετηθούν μέσα σε έναν φάκελο `src`.
- Το αρχείο C που θα υποβληθεί πρέπει να μεταγλωττίζεται χωρίς ειδοποιήσεις για λάθη και με κωδικό επιστροφής (exit code) που να είναι 0. Συγκεκριμένα, το αρχείο σας **πρέπει** να μπορεί να μεταγλωττιστεί επιτυχώς με την ακόλουθη εντολή σε ένα από τα μηχανήματα του εργαστηρίου (linuxXY.di.uoa.gr):

  ```
  make
  ```

  Στο repository της άσκησης σας δίνεται μια πιθανή οργάνωση κώδικα μαζί με ένα Makefile [13]. Μπορείτε να αλλάξετε τα πάντα στο project σας, αλλά θέλουμε να επιβεβαιώσετε πως η εντολή make συνεχίζει να δουλεύει και να παράγει εκτελέσιμα από την μηχανή σκακιού σας.
- Το πρόγραμμά σας πρέπει να προσφέρει τις ακόλουθες δύο διεπαφές:

  1. **Μέσω γραμμής εντολών.** Να δέχεται 3 ορίσματα από την γραμμή εντολών: fen moves timeout. Το πρώτο όρισμα (fen) θα είναι η τρέχουσα κατάσταση της σκακιέρας σε Forsyth-Edwards Notation [11]. Το δεύτερο όρισμα (moves) θα είναι όλες οι δυνατές κινήσεις για αυτήν την θέση σε standard algebraic notation [7] και χωρισμένες με τον κενό χαρακτήρα " ". Τέλος, το 3ο όρισμα

θα είναι ο χρόνος σε δευτερόλεπτα που έχει το πρόγραμμά σας για να αποφασίσει ποια από τις δοθείσες κινήσεις επιλέγει. Το πρόγραμμά σας πρέπει να τυπώνει την θέση (index) της κίνησης που επιλέγει και στην συνέχεια να επιστρέφει με exit code 0.

2. **Μέσω κλήσης συνάρτησης (για χρήση σε Web Assembly [18])**. Το πρόγραμμά σας πρέπει να περιέχει μια συνάρτηση `int choose_move(char * fen, char * moves, int timeout)` η οποία θα δέχεται ορίσματα με σημασιολογία όμοια με παραπάνω και θα επιστρέφει το index της κίνησης που επιλέγει. Όταν γίνεται compile σε Web Assembly, το πρόγραμμά σας θα πρέπει να αποφεύγει να τυπώνει στο stdout/stderr καθώς αυτά δεν είναι επιτρεπτά σε αυτό το περιβάλλον.

- Το πρόγραμμά σας πρέπει να κερδίζει αποφασιστικά (άνω του 60%) των παιχνιδιών με αντίπαλο μια μηχανή Random, δηλαδή μια μηχανή που επιλέγει κινήσεις τυχαία.
- Το πρόγραμμά σας όταν μεταγλωττιστεί δεν θα πρέπει να ξεπερνά το 1MB στον δίσκο.
- Το πρόγραμμά / συνάρτησή σας πρέπει να ολοκληρώνει την εκτέλεση μέσα στον αριθμό των δευτερολέπτων που δίνονται ως όρισμα.

Παρακάτω παραθέτουμε την αλληλεπίδραση με μια ενδεικτική λύση:

```
$ make TARGET=engine
$ ./engine "rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1" \
           "a3 a4 b3 b4 c3 c4 d3 d4 e3 e4 f3 f4 g3 g4 h3 h4 Na3 Nc3 Nf3 Nh3" \
           3
9
```

Παρατηρούμε ότι δώσαμε στο πρόγραμμά μας (το οποίο ονομάσαμε engine) 3 ορίσματα: (1) την αρχική μας σκακιέρα σε FEN format, (2) τις κινήσεις που μπορεί να παίξει χωρισμένες με κενό (20 στον αριθμό) και (3) 3 δευτερόλεπτα για να αποφασίσει ποια κίνηση επιθυμεί να παίξει. Το πρόγραμμά μας αποφάσισε να παίξει την κίνηση στην θέση 9 (προσοχή οι κινήσεις ξεκινάνε από το 0) και επομένως επέλεξε την κίνηση e4 (να κινήσει επομένως το λευκό πιόνι κατά δύο τετράγωνα μπροστά). Αυτό ήταν! Αν το πρόγραμμά σας υποστηρίζει την παραπάνω δυνατότητα (να τυπώνει τον σωστό ακέραιο), είναι σε θέση να χρησιμοποιηθεί ως μηχανή σκακιού. Το πόσο καλά θα τα πάει εξαρτάται από τις επιλογές που θα κάνει.

Στο αρχείο README.md πρέπει να προσθέσετε την περιγραφή του project σας καθώς και ποιες πηγές χρησιμοποιήσατε κατά την υλοποίηση. Περιμένουμε να δούμε write ups υψηλής ποιότητας, καθώς αυτό είναι ένα project που θα μπορούσατε να δημοσιεύσετε μετά το πέρας της άσκησης και να το βάλετε στο portfolio σας. Σκακιστικές μηχανές που επιτυγχάνουν υψηλότερες επιδόσεις, θα πάρουν bonus μονάδες

με μετρική που θα ανακοινωθεί στην συνέχεια, (πιθανώς να χρησιμοποιήσουμε την μετρική ELO [10]).

Καλή Συνέχεια!

Αναφορές

[1] Community Effort. Chess Programming Wiki. https://www.chessprogramming.org/Main_Page.

[2] DIT. Οργανισμός για το μάθημα (GitHub progintro). https://github.com/progintro.

[3] DIT. Πρόσκληση για Εργασία 3. https://classroom.github.com/a/KoToqcBN.

[4] Gary Fredericks. Tic Tac Toe Visualizations. https://gfredericks.com/blog/76.

[5] Quantiacs. Chess. https://en.wikipedia.org/wiki/Chess.

[6] Wikipedia. Advanced (Centaur) Chess. https://en.wikipedia.org/wiki/Advanced_chess.

[7] Wikipedia. Algebraic Notation. https://en.wikipedia.org/wiki/Algebraic_notation_(chess).

[8] Wikipedia. Checkers. https://en.wikipedia.org/wiki/Checkers.

[9] Wikipedia. Deep Blue. https://en.wikipedia.org/wiki/Deep_Blue_(chess_computer).

[10] Wikipedia. ELO. https://en.wikipedia.org/wiki/Elo_rating_system.

[11] Wikipedia. Forsyth-Edwards Notation. https://en.wikipedia.org/wiki/Forsyth%E2%80%93Edwards_Notation.

[12] Wikipedia. Game Complexity. https://en.wikipedia.org/wiki/Game_complexity.

[13] Wikipedia. Make and Makefiles. https://en.wikipedia.org/wiki/Make_(software).

[14] Wikipedia. Minimax. https://en.wikipedia.org/wiki/Minimax.

[15] Wikipedia. Negamax. https://en.wikipedia.org/wiki/Negamax.

[16] Wikipedia. Shannon Number. https://en.wikipedia.org/wiki/Shannon_number.

[17] Wikipedia. Solved Game. https://en.wikipedia.org/wiki/Solved_game.

[18] Wikipedia. Web Assembly. https://en.wikipedia.org/wiki/WebAssembly.