Οικονομικό Πανεπιστήμιο Αθηνών

Τμήμα Πληροφορικής

Τεχνητή Νοημοσύνη

(1η Εργασία – κατασκευή Othello)

Υπεύθυνος καθηγητής: Ι. Ανδρουτσόπουλος

Ομάδα εργασίας:

3040024 Γώγος Αναστάσιος
3040162 Πιλίτσογλου Γεώργιος
3050251 Καλλιγάς Θεόδωρος

Βασικά χαρακτηριστικά εργασίας:

1. **Χρησιμοποιήσαμε γλώσσα JAVA**
2. **Έχει υλοποιηθεί γραφική διεπαφή**
3. **Χρήση αλγορίθμου mini-max με πριόνισμα A-B**
4. **Είναι δυνατή η επιλογή των παρακάτω:**
    a. **Ποιος θα <u>παίξει πρώτος</u> (CPU ή χρήστης)**
    b. **<u>Χρώμα</u> (πούλια) που επιθυμεί ο χρήστης**
    c. **<u>Βάθος Αναζήτησης</u> του αλγορίθμου mini-max**
5. **Κατάλληλα μηνύματα όταν κάποιος δεν έχει καμία επιτρεπτή επόμενη κίνηση**
6. **Μήνυμα τέλους παιχνιδιού**

Τι περιέχεται στο φάκελο που παραδώσαμε:

1. Othello description.doc

(αυτό το έγγραφο)

2. Φάκελος "Othello"

(Στο φάκελο που αυτό περιέχονται τα **αρχεία πηγαίου κώδικα** στη γλώσσα java και τα αντίστοιχα **.class αρχεία** που παράχθηκαν κατά τη μεταγλώττιση. Επίσης, δίνονται και **τρία ακόμα αρχεία (.bat)** τα οποία είναι εντελώς βοηθητικά στη μεταγλώττιση και την εκτέλεση του παιχνιδιού. Συγκεκριμένα:

Αρχείο	**Compile_me.bat**	**Run_me.bat**	**Run_me(more memory).bat**
Εντολές	javac game.java javac tree.java javac MyPopup.java javac Board.java javac MyFrame.java javac Processor.java pause	java Processor pause	java –Xmx300m Processor

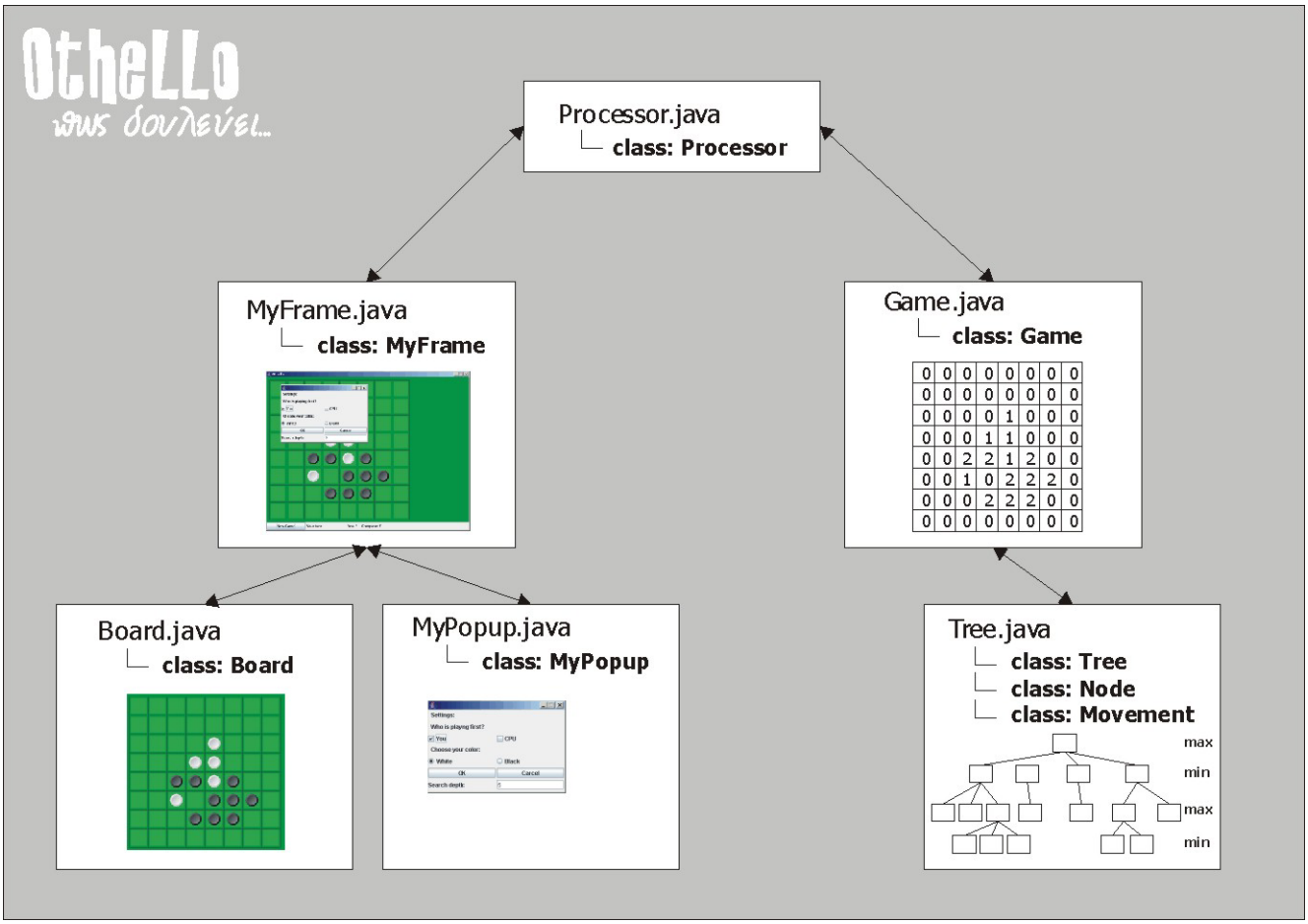
Άρα μπορούμε κάθε φορά να χρησιμοποιούμε αυτά τα αρχεία αντί να πληκτρολογούμε κάθε φορά τις ίδιες εντολές από τη γραμμή εντολών.)

3. Φάκελος "Resources"

(σε αυτό το φάκελο περιέχονται μερικές σελίδες από το διαδίκτυο (σωσμένες σε .mht μορφή) που μας φάνηκαν ιδιαίτερα βοηθητικές.)

Πως δομούνται τα αρχεία και οι τάξεις...

Το παρακάτω σχήμα θα μας βοηθήσει ιδιαίτερα στο να καταλάβουμε το πως αυτά τα αρχεία και οι υλοποιήσεις των τάξεων στη JAVA τελικώς συνδέονται μεταξύ τους.



Η **τάξη Processor** στο αντίστοιχο αρχείο είναι θα λέγαμε η **κεντρική τάξη ελέγχου** που χρησιμοποιεί και διαχειρίζεται τις υπόλοιπες ώστε να βγαίνει το τελικό αποτέλεσμα του παιχνιδιού Othello. Όπως είναι φανερό και στο παραπάνω σχήμα, στην ουσία η δουλειά της τάξης αυτής είναι να **συνδέει το γραφικό περιβάλλον** (παιχνίδι που βρίσκεται σε εξέλιξη και αντιλαμβάνεται ο χρήστης) **με το παιχνίδι που αντιλαμβάνεται ο Η/Υ**. Επίσης, είναι αυτή που επιβλέπει το παιχνίδι και φροντίζει ώστε να προχωρά ομαλά πχ:

- Δίνει κάθε φορά τη σειρά στο χρήστη ή στη CPU
- Αν τελειώσει η παρτίδα εμφανίζει μήνυμα τέλους του παιχνιδιού
- Αν ο χρήστης ζητήσει να ξεκινήσει καινούργια παρτίδα στέλνει τα κατάλληλα μηνύματα αρχικοποιήσεων

Σημαντικό: Η αναπαράσταση του παιχνιδιού γίνεται με τη χρήση ενός πίνακα 8x8. Και τα γραφικά και ο πυρήνας που υλοποιεί το παιχνίδι για τη μεριά του υπολογιστή χρησιμοποιούν έναν τέτοιο πίνακα.

Άρα η Processor είναι ο διαμεσολαβητής που η βασικότερη δουλειά του είναι να παίρνει το παιχνίδι (σε μορφή πίνακα) όταν έχει παίξει ο χρήστης και να το παραδίδει στον «υπολογιστή» και έπειτα να κάνει το αντίστροφο μέχρις ότου τερματίσει η παρτίδα. Περιέχει επίσης τους handlers για τα γραφικά.

Οι τάξεις **Board** (extends *JPanel*) και **MyPopup** (extends *JFrame*) είναι αυτές που όταν ενωθούν από δίνουν το τελικό αποτέλεσμα των γραφικών. Η πρώτη είναι τα γραφικά ή αλλιώς το ταμπλώ του παιχνιδιού και η δεύτερη είναι το αναδυόμενο (pop up) παράθυρο που εμφανίζεται όταν ο χρήστης θέλει να ξεκινήσει καινούργια παρτίδα. Έπειτα, η **MyFrame** (extends *JFrame*) είναι αυτή που διαχειρίζεται τις δύο παραπάνω τάξεις.

Η τάξη **Game** φτιάχτηκε με σκοπό να της ζητείται από τον Processor η επόμενη κίνηση που επιλέγει ο υπολογιστής να παίξει κάθε φορά που ο χρήστης κάνει την κίνησή του. Έτσι κάθε φορά που είναι η σειρά του υπολογιστή να παίξει, η Game φτιάχνει ένα αντικείμενο της τάξης **Tree** αναθέτοντας στην τελευταία την δύσκολη δουλειά της εύρεσης επόμενης κίνησης.

Η **Tree**, όπως δηλώνει και η λέξη, είναι η υλοποίηση ενός δένδρου. Σε αυτό τη βοηθούν οι τάξεις **Node** (κόμβος δένδρου) και **Movement**. *(Η τελευταία είναι μια πολύ απλή υλοποίηση που κρατά σε δύο μεταβλητές x και y κινήσεις του παιχνιδιού, δηλαδή κρατά το τετράγωνο που τοποθετήθηκε πούλι. Αυτή η τάξη κρίθηκε αναγκαίο να κατασκευασθεί γιατί υπήρχαν περιπτώσεις που μια μέθοδος έπρεπε να επιστρέψει κίνηση (δηλ. x και y). Επειδή μια μέθοδος πάντα επιστρέφει μία τιμή το πρόβλημα λύθηκε επιστρέφοντας ένα αντικείμενο τύπου Movement.)*

Αρχικά η **Tree** φτιάχνει τη ρίζα του δένδρου δείχνοντας σε ένα Node που κατασκευάζει η ίδια. Στη συνέχεια καλείται **ο αλγόριθμος mini-max με πριόνισμα A-B ο οποίος υλοποιείται μέσα στην Tree**, χρησιμοποιεί την τάξη Node, και **κατασκευάζει ο ίδιος τα παιδιά των κόμβων** (επόμενες κινήσεις παιχνιδιού). Τέλος, η Tree επιστρέφει στην Game την κίνηση που επέλεξε να παίξει ο υπολογιστής.

Η τάξη **Node** υλοποιεί μια τυπική τάξη κόμβου δένδρου. Ένας κόμβος αποθηκεύει πρώτα από όλα το παιχνίδι (πίνακας 8 επί 8), την κατηγορία του (min/max), την κίνηση με την οποία έφτασε σε αυτή την κατάσταση το παιχνίδι (x,y), το χρώμα του παίχτη και της cpu. Τέλος περιέχει τη **μέθοδο Heuristic** που **καλείται μόνο από τους κόμβους που βρίσκονται σε βάθος ίσο με το βάθος αναζήτησης που έχει ορίσει ο χρήστης** και αποτελεί μια συνάρτηση αξιολόγησης βάση της οποίας λειτουργεί στη συνέχεια ο mini-max.

Ευριστικές και κριτήρια αξιολόγησης:

Το παιχνίδι χωρίζεται σε **τρεις φάσεις**:

1. έχουν τοποθετηθεί λιγότερα από 16 πούλια (ίσο με τις θέσεις του «ασφαλούς» τετραγώνου στο κέντρο)
2. έχουν τοποθετηθεί περισσότερα από 16 πούλια και λιγότερα από 56
3. έχουν τοποθετηθεί περισσότερα από 56 πούλια

Σε κάθε φάση χρησιμοποιούνται διαφορετικά βάρη στις συναρτήσεις αξιολόγησης.

Οι καταστάσεις του παιχνιδιού αξιολογούνται με βάση τα εξής:

1. **Απλό μέτρημα των τετραγώνων που έχει ο κάθε παίχτης -** (χρησιμοποιείται προς το τέλος του παιχνιδιού με αυξημένο βάρος)
2. **Μέτρημα των τετραγώνων με ειδικά βάρη** – (αυξημένο βάρος στις γωνίες και λιγότερο στις πλευρές, αρνητικό βάρος στην επικίνδυνη περιοχή= τετράγωνα δίπλα στα εξωτερικά τετράγωνα)
3. **Μέτρημα των δυνατών επόμενων κινήσεων** – (ο υπολογιστής κινείται προς τέτοια κατεύθυνση που δεν αφήνει στο χρήστη το περιθώριο πολλών επιλογών για επόμενες κινήσεις)
4. **Μέτρημα των ελεύθερων – κενών θέσεων γύρω από ένα τετράγωνο**

   Αυτή η ιδέα βρέθηκε στο διαδίκτυο και βρίσκεται στις σελίδες (.mht) που παραδώσαμε ως το υλικό που μας βοήθησε. Πιο συγκεκριμένα, επειδή το παιχνίδι ξεκινά με 4 πούλια στο κέντρο, αρχικοποιείται ένας πίνακας 8x8 που για κάθε τετράγωνο δηλώνει πόσες κενές θέσεις υπάρχουν γύρω του. Π.χ στη θέση [0][0] που αποτελεί γωνία θα έχει την τιμή 3, γιατί αρχικά 3 τετράγωνα γύρω από τη γωνία είναι άδεια.

Ο πίνακας αυτός αρχικοποιείται με τον εξής κώδικα (τον οποίο χρησιμοποιήσαμε όπως τον βρήκαμε...)

```
short liberties[][] = {
                    { 3, 5, 5, 5, 5, 5, 5, 3 },
                    { 5, 8, 8, 8, 8, 8, 8, 5 },
                    { 5, 8, 7, 6, 6, 7, 8, 5 },
                    { 5, 8, 6, 5, 5, 6, 8, 5 },
                    { 5, 8, 6, 5, 5, 6, 8, 5 },
                    { 5, 8, 7, 6, 6, 7, 8, 5 },
                    { 5, 8, 8, 8, 8, 8, 8, 5 },
                    { 3, 5, 5, 5, 5, 5, 5, 3 }
```

Η ιδέα λέει ότι σε κάθε κίνηση ο πίνακας αυτός ανανεώνεται. Έτσι χρειάστηκε να υλοποιήσουμε τον δικό μας κώδικα για αυτή τη δουλειά επειδή εκείνος που βρισκόταν στην ιστοσελίδα ήταν σε κάποια μορφή ψευδοκώδικα.

```
public void update_liberties()
{
            int x= last_movement_X;
            int y= last_movement_Y;

            for (int i=x-1 ; i<=x+1; i++)
            {
                  for (int j=y-1 ; j<=y+1 ; j++)
                  {
                        if ((i<0) || (i>7) || (j<0) || (j>7)) continue;

                  if (i != x || j != y)
                  {
                              liberties[i][j]--;
                  }
                  }
            }
}
```

Η χρήση αυτού του στοιχείου μας βοήθησε κυρίως στην αξιολόγηση των πλευρικών τετραγώνων. Μπορεί κανείς να έχει πολλά πλευρικά πούλια αλλά αν δεν καλύπτονται από πλαϊνά του ίδιου χρώματος, τότε ο αντίπαλος μπορεί να του πάρει όλη την πλευρά.

Μια ματιά στην ευριστική που χρησιμοποιούμε...

```
public int heuristic()
{
            int h=0;

            int phase= this.phase();

            if (phase==1)
            {
                  int h0= 10 * h_simpleCount();          //απλό μέτρημα τετραγώνων
                  int h1= 50 * h_dangerous_corners();    //σημεία γύρω από γωνίες
                  int h2= 10 * h_number_of_next_moves();//αριθμός επομενων κινήσεων
                  int h3= 20 * h_dangerous_zone();      //τετράγωνα δίπλα στις πλευρές
                  int h4= 30 * h_acmes();               //αυξημένο βάρος στις πλευρές
                  int h5= 50 * h_count_corners();       //αυξημένο βάρος στις γωνίες
                  h= h0+h1+h2+h3+h4+h5;
```

```
		}
		else if (phase==2)
		{
			int h0= 30 * h_simpleCount();
			int h1= 50 * h_dangerous_corners();
			int h2= 30 * h_number_of_next_moves();
			int h3= 20 * h_dangerous_zone();
			int h4= 30 * h_acmes();
			int h5= 50 * h_count_corners();
			h= h0+h1+h2+h3+h4+h5;
		}
		else if (phase==3)
		{
			int h1= 50 * h_dangerous_corners();
			int h2= 50 * h_number_of_next_moves();
			int h3= 20 * h_dangerous_zone();
			int h4= 30 * h_acmes();
			int h5= 50 * h_count_corners();
			int h6= 75 * h_simpleCount();
			h= h1+h2+h3+h4+h5+h6;
		}

		return h;
}
```

Είναι φανερό αυτό που προαναφέρθηκε ότι το παιχνίδι χωρίζεται σε 3 φάσεις. Από εκεί και πέρα χρησιμοποιούνται οι διάφορες συναρτήσεις αξιολόγησης με βάρη, στα οποία καταλήξαμε δοκιμάζοντας. Δηλαδή, ελέγχαμε αν πράγματι δημιουργήσαμε δυνατότερο παίχτη και έτσι αναδείχθηκε ο καλύτερος. Φυσικά σηκώνει πολύ συζήτηση για το αν υπάρχει κάτι καλύτερο και δυστυχώς, λόγω έλειψης χρόνου, δεν καταφέραμε να κάνουμε αυτή την αναζήτηση βαρών που σχεδιάζαμε να κάνουμε. **Θέλαμε, τροποποιώντας την τάξη Processor, να βάλουμε το παιχνίδι μας να παίζει εναντίον του εαυτού του πολλές φορές και κάθε φορά να αλλάζει τυχαία τα βάρη στην ευριστική του χαμένου. Με αυτό τον τρόπο θα καταφέρναμε να αναδείξουμε τον καλύτερο δυνατό παίκτη, δεδομένων φυσικά των συναρτήσεων αξιολόγησης που εφαρμόσαμε και γενικότερα του τρόπου σκέψης που χρησιμοποιήσαμε.**

Κριτική εργασίας:

Πιστεύουμε ότι καταφέραμε να υλοποιήσουμε όλα εκείνα που ζητούσε η εκφώνηση της εργασίας. Στόχος μας βέβαια δεν ήταν μόνο αυτός. Μέσα από την ενασχόλησή μας με το θέμα, μάθαμε αρκετά γύρω από το παιχνίδι του Othello και θελήσαμε, όταν πλέον είχαμε φτιάξει έναν «παίχτη» που έπαιζε μόνος του, να τον κάνουμε καλύτερο και αρκετά ανταγωνιστικό! Σε σύγκριση με τα πρωταρχικά στάδια, η βελτίωση που έχει σημειώσει ο παίχτης μας είναι πράγματι ραγδαία. Βέβαια, δεν καταφέραμε να τον φτάσουμε στο επίπεδο που φανταζόμασταν. Δυστυχώς, ο χρόνος που χρειάζεται για την εκλέπτυνση είναι πολύ μεγαλύτερος από αυτόν της ανάπτυξης. Φυσικά, ο κώδικας υπάρχει και γιατί όχι, έξω από τα πλαίσια του μαθήματος μας περιμένει για να τον βελτιώσουμε!