Πολυτεχνείο Κρήτης

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

ΠΛΗ 102 – Αντικειμενοστραφής Προγραμματισμός

Εαρινό εξάμηνο 2022 **2η Άσκηση** **Παράδοση: 20 Μαΐου 2022**

1 Εισαγωγή

Στην άσκηση αυτή θα υλοποιήσετε το γνωστό παιχνίδι **tic-tac-toe**. Η άσκηση αυτή έχει τους παρακάτω στόχους:

1. Να εφαρμόσετε στην πράξη τις γνώσεις αντικειμενοστραφούς προγραμματισμού.
2. Να εφαρμόσετε στην πράξη τις γνώσεις δομημένου προγραμματισμού και βασικών δομών δεδομένων και αλγορίθμων.
3. Να αυξήσετε την προγραμματιστική σας εμπειρία.
4. Να βιώσετε τα οφέλη και τις δυσκολίες της ομαδικής ανάπτυξη κώδικα.
5. Να εξασκηθείτε σε επαγγελματικές πρακτικές ανάπτυξης λογισμικού, όπως το debugging και το unit testing.
6. Να μάθετε τη χρήση των βασικών βιβλιοθηκών της Java.
7. Να εξοικειωθείτε με την ανάπτυξη παραθυρικών εφαρμογών.
8. Να νιώσετε περήφανοι για την πρώτη σας σοβαρή εφαρμογή λογισμικού.

Οδηγίες. Οι ασκήσεις βοηθούν στην εξοικείωσή σας με τον προγραμματισμό και την κατανόηση της ύλης.

- Η ανάπτυξη της άσκησης θα γίνει σε ομάδες των 2 ατόμων.
- Η συζήτηση των ασκήσεων είναι θεμιτή, αλλά η σύνταξη του κώδικα πρέπει να γίνεται μόνο από τα μέλη της ομάδας.
- Γράψτε κομψό, ευανάγνωστο κώδικα και προσθέστε επεξηγηματικά σχόλια όπου χρειάζεται.
- Τα σχόλια να γράφουν με λατινικούς χαρακτήρες για να μην υπάρχει πρόβλημα με το encoding.
- Τηρήστε κατά γράμμα τις οδηγίες και τις προδιαγραφές που σας δίνονται από την άσκηση.
- Η παράδοση των λύσεων γίνεται μόνο μέσω της ιστοσελίδας `https://eclass.tuc.gr`

- Το παραδοτέο θα πρέπει να είναι ένα συμπιεσμένο αρχείο (.zip) με τα αρχεία του κώδικά σας το οποίο θα είναι ένα έγκυρο Eclipse Project (οδηγίες αποστολής στο τέλος της εκφώνησης)
- Η υποβολή παραδοτέου αυτόματα δηλώνει ότι είστε οι μοναδικοί συγγράφεις των λύσεων.
- Σε περιπτώσεις αντιγραφής οι εμπλεκόμενοι μηδενίζονται.

2 Περιγραφή της εφαρμογής

Η εφαρμογή σας θα πρέπει να ικανοποιεί τις παρακάτω απαιτήσεις:

1. Η εφαρμογή σας θα είναι ένα παιγνίδι τρίλιζας (tic-tac-toe).
2. Η εφαρμογή θα λειτουργεί με γραφική διεπαφή χρήστη (GUI).
3. Όλες οι δομές δεδομένων που θα χρησιμοποιεί (δυναμικοί πίνακες, λίστες, πίνακες κατακερματισμού, δέντρα κλπ) και όλοι οι αλγόριθμοι (αναζήτησης, ταξινόμησης κλπ) θα πρέπει να υλοποιηθούν από την εφαρμογή.
4. Θα υποστηρίζει παιχνίδια μεταξύ δύο παικτών.
5. Θα υποστηρίζει παιχνίδια μεταξύ ενός παίκτη και του υπολογιστή.
6. Θα υποστηρίζει παιγνίδια μεταξύ δύο AI παικτών.
7. Θα διατηρεί αρχείο με τους παίκτες που έχουν παίξει το παιχνίδι. Το αρχείο αυτό θα διατηρείται αποθηκευμένο σε αρχείο με όνομα **"tuctactoe.ser"**, αποθηκευμένο στο home directory του χρήστη. Στο αρχείο θα αποθηκεύονται και "παίκτες AI".
8. Η μορφή του αρχείου θα είναι σε όποιο format θέλετε, π.χ., Java serialization.
9. Για κάθε παίκτη θα τηρείται το όνομά του. Το όνομα του κάθε παίκτη θα έχει μήκος ως 20 χαρακτήρες. Θα πρέπει να μην περιλάμβανει κενά στην αρχή και στο τέλος.
10. Θα διατηρεί για κάθε παίκτη το ιστορικό του, που θα περιλαμβάνει:
    - Συνολικό αριθμό παιχνιδιών, νίκες, ήττες, ισοπαλίες.
    - Το συνολικό σκορ του παίκτη, με τον παρακάτω τύπο

      $$\text{σκορ} = 50 \times \frac{2 \star \text{νίκες} + \text{ισοπαλίες}}{\text{αριθμός παιχνιδιών}}$$

      Με βάση αυτόν τον τύπο, το σκορ ενός παίκτη είναι ένας αριθμός κινητής υποδιαστολής από 0 ως 100.
    - Τα 5 καλύτερα παιγνίδια του παίκτη.
      Ένα παιχνίδι είναι καλύτερο από ένα άλλο, αν:
      (a) Το αποτέλεσμα είναι καλύτερο (νίκη > ισοπαλία > ήττα), ή

(b) τα αποτελέσματα είναι ίσα αλλά ο αντίπαλος ήταν καλύτερος (με βάση το σκορ του κατά το χρόνο του παιχνιδιού), ή

(c) τα παραπάνω είναι ίσα αλλά το παιχνίδι είναι πιο πρόσφατο από το άλλο.

- Τα 5 πιο πρόσφατα παιχνίδια του παίκτη.
- Για κάθε παιχνίδι που διατηρείται, τα εξής στοιχεία
  - Τα ονόματα των παικτών
  - Τα σκορ των παικτών κατά το χρόνο διεξαγωγής
  - Το αποτέλεσμα
  - Το χρόνο (ημερομηνία και ώρα) παιχνιδιού.

11. Θα εμφανίζει τους 10 καλύτερους παίκτες σε μια οθόνη με τίτλο "Hall of Fame". Για κάθε παίκτη θα φαίνεται το συνολικό του σκορ και ο αριθμός των παιχνιδιών που έχει παίξει.

12. Θα υλοποιεί τουλάχιστον δύο παίκτες AI, με ονόματα Hal και Mr.Bean αντίστοιχα. Μπορείτε να υλοποιήσετε και περισσότερους τέτοιους παίκτες.

13. Ο παίκτης Mr.Bean θα παίζει τυχαίες κινήσεις.

14. Ο παίκτης Hal θα παίζει τέλεια.

3 Οδηγίες για την ομαδική ανάπτυξη

Για να μπορέσετε να υλοποιήσετε με επιτυχία την εργασία σας, θα πρέπει να ακολουθήσετε κάποιες βασικές πρακτικές της ομαδικής ανάπτυξης λογισμικού. Αυτό είναι ιδιαίτερα σημαντικό την εποχή αυτή της πανδημίας COVID, που η συνεργασία σας με το άλλο μέλος της ομάδας θα πρέπει να γίνει ηλεκτρονικά.

3.1 Ανταλλαγή κώδικα με το git

Η πιο σημαντική λειτουργία στον ομαδικό προγραμματισμό είναι η ανταλλαγή κώδικα. Για το σκοπό αυτό, οι προγραμματιστές χρησιμοποιούν συστήματα *ελέγχου εκδόσεων* (version control). Το πιο διαδεδομένο τέτοιο σύστημα είναι το **github**. Αν δεν ξέρετε ήδη τι είναι, σίγουρα θα το μάθετε! Για το σκοπό αυτό, μπορείτε να διαβάσετε την παρακάτω σελίδα
`https://guides.github.com/activities/hello-world/#intro`

Τι κάνει το github για μένα; Φανταστείτε το project σας σαν μια συλλογή από αρχεία, που αλλάζει με το χρόνο καθώς δουλεύετε. Όταν προσθέτετε μια μέθοδο, βρίσκετε κάποιο bug, ή σβήνετε κάποιον κώδικα που δε σας χρειάζεται πλέον, μπορείτε να επικυρώσετε (commit) τις αλλαγές σας σε μια καινούρια *επικύρωση*.

Μια επικύρωση είναι σαν ένα "στιγμιότυπο" του project σας.Το git διατηρεί για εσάς όλες τις επικυρώσεις, έτσι μπορείτε, αν το θελήσετε, να "γυρίσετε πίσω" στο χρόνο και να ξανα-επεξεργαστείτε μια προηγούμενη επικύρωση. Επίσης, μπορείτε και τα δύο μέλη της ομάδας να ενημερώνετε ένα κοινό repository που θα βρίσκεται στο github, με τις επικυρώσεις που παράγετε. Έτσι, οι αλλαγές καθενός θα είναι ορατές από τον άλλο.

3.2 Δοκιμασίες ενότητας (unit tests)

Έχουμε ήδη δει στη θεωρία τη σημασία και την ευκολία που προσφέρουν τα unit tests κατά την ανάπτυξη του κώδικα. Ειδικά κατά την ανάπτυξη του domain model, μπορείτε να εκτελείτε τον κώδικά σας και να ελέγχετε την ορθότητά του. Αυτό είναι διπλά χρήσιμο όταν αναπτύσσετε κώδικα από κοινού, καθώς κάποιες αλλαγές που κάνει ο ένας προγραμματιστής μπορούν να επηρεάσουν την ορθή λειτουργία κώδικα που έγραψε ο άλλος.

Οι μη-επαγγελματίες προγραμματιστές πιστεύουν—απολύτως λανθασμένα—ότι τα unit tests είναι πολύς μπελάς για λίγο όφελος, ότι ο κώδικάς τους είναι πολύ απλός για να χρειάζεται tests, και γενικά ότι είναι περιττά. Περνούν πολλά και "πικρά" χρόνια μέχρι να καταλάβουν το λάθος τους ...

Κάντε τη χάρη στον εαυτό σας να συνηθίσετε από τώρα, που είστε αρχάριοι, να γράφετε tests. Σας υπόσχομαι ότι θα είναι η καλύτερη επένδυση που θα κάνετε σαν προγραμματιστές.

4 Ο τέλειος παίκτης του tic-tac-toe

Στην ενότητα αυτή θα περιγράψουμε τις ιδέες που οδηγούν στον αλγόριθμο του τέλειου παίκτη tic-tac-toe. Για το σκοπό αυτό, θα χρησιμοποιήσουμε μια τεχνική που βασίζεται στο *δέντρο παιχνιδιού*. Μπορείτε να βρείτε περισσότερες λεπτομέρειες στην ιστοσελίδα `https://en.wikipedia.org/wiki/Game_tree`.

Το δέντρο παιχνιδιού είναι ένα (ιδεατό) διάγραμμα όλων των δυνατών κινήσεων σε ένα παιχνίδι. Στο Σχ. 1 φαίνεται το δέντρο που ξεκινά από το αρχικό ταμπλό (καμμιά κίνηση) και δείχνει τις επόμενες δύο κινήσεις (κάποιες κινήσεις που είναι συμμετρικές παραλείπονται, λόγω περιρισμένου χώρου).

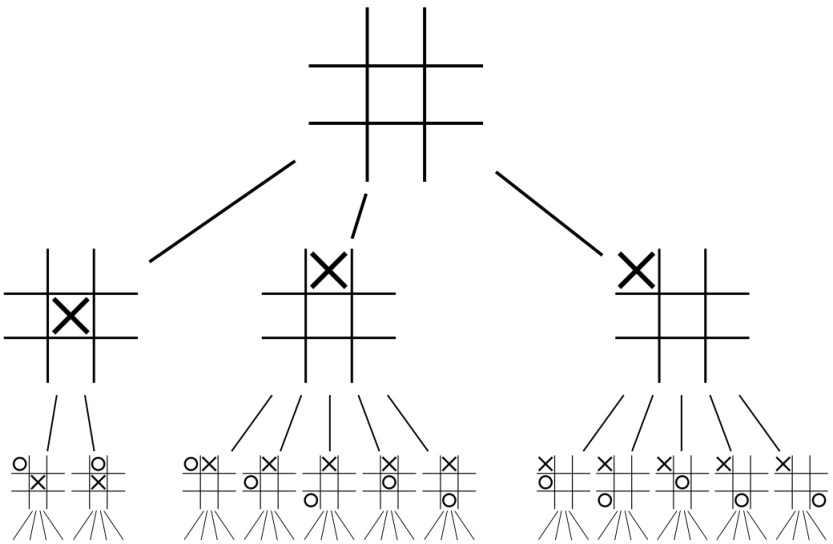

Figure 1: Τα πρώτα 3 επίπεδα του δέντρου παιχνιδιού. (*Πηγή: Wikipedia*)

Στη συνέχεια θα υποθέσουμε ότι ο 1ος παίκτης είναι πάντα ο $\times$, παρόλο που αυτό δεν είναι στους κανόνες του παιχνιδιού (ο κάθε παίκτης μπορεί να διαλέξει το δικό του σύμβολο).

4.1 Ανάλυση του tic-tac-toe

Ας θεωρήσουμε ότι έχουμε ένα ταμπλό b.

- Ας πούμε ότι $m(b)$ είναι ο αριθμός των μη-κενών θέσεων του ταμπλό b. Στο αρχικό ταμπλό b_{init} έχουμε $m(b_{\text{init}}) = 0$. Προφανώς, $0 \leq m(b) \leq 9$.

- Ένα ταμπλό είναι *νόμιμο* αν μπορούμε να φτάσουμε σε αυτό παίζοντας το παιχνίδι. Για παράδειγμα, το παρακάτω ταμπλό είναι μη-νόμιμο

∘		×
∘		×
∘		×

  διότι το παιχνίδι θα είχε ήδη τελειώσει μια κίνηση πριν (με νικητή τον $\times$). Επίσης, το παρακάτω ταμπλό είναι μη-νομιμο

∘		×
		×
		×

  διότι δεν έχει το σωστό αριθμό από κάθε σύμβολο

- Σε ένα νόμιμο ταμπλό b, θα συμβολίσουμε με $R(b)$ τον παίκτη που έκανε τρίλιζα, αν υπάρχει:

$$R(b) = \begin{cases} \times & \text{τρίλιζα από τον } \times \\ \circ & \text{τρίλιζα από τον } \circ \\ \bot & \text{δεν υπάρχει τρίλιζα} \end{cases}$$

- Ένα ταμπλό b είναι *τελικό (final)* αν ισχύει ένα από τα παρακάτω (ή και τα 2).

$$R(b) \neq \bot \quad \text{ή} \quad m(b) = 9$$

- Σε κάθε νόμιμο ταμπλό b, ας συμβολίσουμε με $p(b)$ τον παίκτη που κινεί. Αυτός καθορίζεται από το $m(b)$ ως εξής:

$$p(b) = \begin{cases} \times & \text{αν } m(b) \text{ άρτιος} \\ \circ & \text{αν } m(b) \text{ περιττός} \end{cases}$$

  Επίσης, με $\overline{p(b)}$ συμβολίζουμε τον παίκτη που δεν κινεί (τον άλλο παίκτη).

- Το ταμπλό b' είναι **επόμενο** του ταμπλό b, αν από το b πάμε στο b' με μια κίνηση. Δηλαδή, το b' είναι *παιδί* του ταμπλό b στο δέντρο του παιχνιδιού.

Για να συνοψίσουμε, για κάθε νόμιμο ταμπλό b έχουμε ορίσει τα $m(b), R(b)$. Επίσης, για ένα μη-τελικό νόμιμο ταμπλό, έχουμε τον παίκτη που κινεί $p(b)$.

4.2 Η καλύτερη κίνηση σε κάποιο ταμπλό

Τώρα, θα ορίσουμε μια τελευταία τιμή για κάθε ταμπλό: με $\text{Best}(b)$ θα συμβολίσουμε **το καλύτερο δυνατό αποτέλεσμα για τον παίκτη $p(b)$, αν και οι δύο παίκτες παίζουν τέλεια**. Το $\text{Best}(b)$ θα είναι πάντα ένα από τα παρακάτω:

$\text{Best}(b)$	Αποτέλεσμα
1	θα νικήσει ο $p(b)$
0	θα έρθει ισοπαλία
-1	θα νικήσει ο $\overline{p(b)}$

Παράδειγμα. Ας υποθέσουμε ότι έχουμε το μη-τελικό, νόμιμο ταμπλό b

$\circ$	$\times$	$\times$
$\circ$		

Βλέπουμε ότι $p(b) = \times$ και (μετά από λίγη σκέψη) $\text{Best}(b) = -1$. Αυτό συμβαίνει επειδή, όποια κίνηση και να κάνει ο παίκτης $\times$, ο παίκτης $\circ$ μπορεί να κερδίσει το παιχνίδι.

Άρα, τώρα ας δούμε την καρδιά του αλγορίθμου μας: πώς να υπολογίσουμε την τιμή $\text{Best}(b)$ για κάποιο νόμιμο ταμπλό. Η ιδέα είναι η εξής: ο παίκτης που είναι η σειρά του να κινήσει, δλδ. ο $p(b)$, θα επιδιώξει να μεταβεί σε κάποιο επόμενο ταμπλό b' που θα του εξασφαλίσει το καλύτερο δυνατό αποτέλεσμα. Το αποτέλεσμα αυτό θα είναι **το αντίθετο από την τιμή** $\text{Best}(b')$.

Οπότε, έχουμε την παρακάτω λογική, που είναι γνωστή ως αλγόριθμος minmax:

- Αν το ταμπλό b είναι τελικό, τότε θέτουμε

$$\text{Best}(b) := \begin{cases} 1 & \text{αν } R(b) = p(b), \\ 0 & \text{αν } R(b) = \bot, \\ -1 & \text{αλλιώς} \end{cases}$$

- αλλιώς,

$$\text{Best}(b) := \max\big\{ -\text{Best}(b') \mid b' \text{ επόμενο του } b \big\}$$

Καταλήγουμε ότι μπορούμε να υλοποιήσουμε μια **αναδρομική συνάρτηση** που να επιστρέφει την τιμή $\text{Best}(b)$ για κάθε νόμιμο ταμπλό b. Η συνάρτηση αυτή θα "διασχίζει" το δέντρο παιχνιδιού υπολογίζοντας για κάθε ταμπλό b την $\text{Best}(b)$.

4.3 Οι βέλτιστες κινήσεις σε ένα ταμπλό

Αν ξέρουμε για κάθε ταμπλό b την τιμή $\text{Best}(b)$, τότε μπορούμε να παίξουμε τέλεια: όταν είναι η σειρά μας με το παιχνίδι στο ταμπλό b, βλέπουμε όλες τις πιθανές *επόμενες κινήσεις μας* και διαλέγουμε (τυχαία) μια από αυτές που το ταμπλό τους b' έχει το καλύτερο δυνατό αποτέλεσμα για εμάς, δηλαδή $\text{Best}(b') = \text{Best}(b)$.

Algorithm 1: Ο αλγόριθμος BestMoves

input : Ένα νόμιμο μη-τελικό ταμπλό b
output: Το συνολο bestMoves, των βέλτιστων κινήσεων για τον παίκτη $p(\mathrm{b})$

```
begin
    bestMoves ⟵ ∅;
    foreach move: κενό κελί του b do
        bchild ⟵ επόμενο ταμπλό του b από την κίνηση move;
        if Best(bchild) == Best(b) then
            πρόσθεσε την κίνηση move στο σύνολο bestMoves;
```

4.4 Επιτάχυνση του υπολογισμού των τιμών της $\mathrm{Best}(b)$

Για καλύτερη απόδοση (επιτάχυνση του υπολογισμού), μπορείτε να υλοποιήσετε τη λογική του α−β κλαδέματος, όπως είδαμε στη θεωρία. Δεδομένου ότι ο minmax είναι πιο απλός στην υλοποίηση, μπορείτε να τον χρησιμοποιήσετε **για να γράψετε unit tests** με τα οποία θα ελέγχετε την ορθότητα της αναζήτησης α−β.

5 Σχεδίαση

Όπως μάθαμε στη θεωρία, οι γραφικές εφαρμογές υλοποιούνται με την λογική του **διαχωρισμού των θεμάτων (separation of concerns))**, και αναλύονται σε 3 μέρη: το μοντέλο (model), τις όψεις (views) και τον ελεγκτή (controller). Σύμφωνα με το πρότυπο αυτό, ο κώδικας αναλύεται σε 3 μέρη:

Model που περιγράφει τα δεδομένα και τον κώδικα που δεν αφορά τη γραφικη διεπαφή. Για να αποφασίσουμε αν κάτι ανήκει στο μοντέλο ή όχι, έχουμε τον παρακάτω κανόνα:

> Αν μια λειτουργία ή ένα δεδομένο θα υπήρχε στην εφαρμογή ακόμη και με διεπαφή κονσόλας, τότε ανήκει στο domain model.

View είναι ο κώδικας που περιγράφει την εμφάνιση των δεδομένων στην οθόνη και τη λήψη εισόδου (πληκτρολόγιο, ποντίκι αν θέλετε). Ο κανόνας εδώ είναι:

> Μια κλάση για κάθε περιοχή/ενότητα της οθόνης.

Controller Εδώ βρίσκεται η λογική της αντίδρασης της εφαρμογής σε events. Θα αναφερθούμε παρακάτω γι αυτό.

Παρακάτω δίνονται κάποιες υποδείξεις σχετικά με τη σχεδίαση.

5.1 Μοντέλο περιοχής (domain model)

Το μοντέλο περιοχής αφορά όλο τον κώδικα που δεν περιλαμβάνει θέματα γραφικής διεπαφής. Μπορείτε π.χ. να υλοποιήσετε τις παρακάτω κλάσεις.

class GameRecord μια κλάση που περιγράφει παιχνίδια που έχουν τελειώσει. Θα αποθηκεύει τους παίκτες, το αποτέλεσμα, το ατομικό σκόρ των παικτών κατά το χρόνο διεξαγωγής του παιγνιδιού και το χρόνο διεξαγωγής.

class Player μια κλάση που περιγράφει παίκτες. Για κάθε παίκτη απαιτείται το όνομα και το ιστορικό του. Το ιστορικό του περιλαμβάνει:

- Συνολικό αριθμό παιχνιδιών, νίκες, ήττες.
- Τα 5 πιο πρόσφατα παιχνίδια και υπολογισμό του σκορ σε αυτά (collection από GameRecord).
- Τα 5 καλύτερα παιχνίδια του χρήστη (collection από GameRecord).

class PlayerRoster μια κλάση-συλλογή από Players. Η κλάση αυτή θα εξασφαλίζει ότι οι παίκτες έχουν διαφορετικά μεταξύ τους ονόματα. Μπορείτε λοιπόν να την υλοποιήσετε ως Data Access Object, με τις κατάλληλες μεθόδους.

Επίσης, μπορείτε εδώ να υλοποιήσετε τις αναζητήσεις που απαιτεί το παιχνίδι, όπως οι παρακάτω:

findPlayerNames() Επιστρέφει μια συλλογή με τα ονόματα όλων των παικτών.

findPlayer(name) Επιστρέφει τον παίκτη με το δεδομένο όνομα.

findHallOfFame(n) Επιστρέφει μια συλλογή με τους n καλύτερους παίκτες.

class Board μια κλάση που θα περιγράφει την κατάσταση του ταμπλό σε ένα παιχνίδι. Στην §4 είδαμε τις λειτουργίες που μπορεί να έχει αυτή η κλάση. Μια καλή πρακτική είναι να υλοποιήσετε αυτή την κλάση ώς **immutable class**, έτσι ώστε να μπορείτε να θεωρήσετε τα διαφορετικά boards ως "τιμές".

Για να υλοποιήσετε τους παίκτες AI, θα πρέπει να ορίσετε με κάποιο τρόπο τον αλγόριθμο του κάθε παίκτη

- Κανέναν, για "ανθώπινους" παίκτες,
- "Τυχαίες κινήσεις" για παίκτες όπως ο Mr. Bean,
- "Game tree" για παίκτες όπως ο Hal.

Έχετε διάφορες επιλογές, π.χ. μπορείτε να δημιουργήσετε υποκλάσεις της Person, ή να αποθηκεύσετε σε κάθε Person κάποιο πεδίο που να ορίζει τον αλγόριθμο. *Επιλέξτε μια σχεδίαση, σύμφωνα με τις αρχές αντικειμενοστραφούς προγραμματισμού και να είστε έτοιμοι να εξηγήσετε την επιλογή σας.*

5.2 Όψεις της γραφικής διεπαφής

Ένα παράδειγμα της γραφικής διεπαφής φαίνεται στην παρακάτω Εικ. 2

Το παράθυρο αποτελείται από 4 περιοχές:

- Στην κορυφή υπάρχει μια μπάρα με 3 buttons,

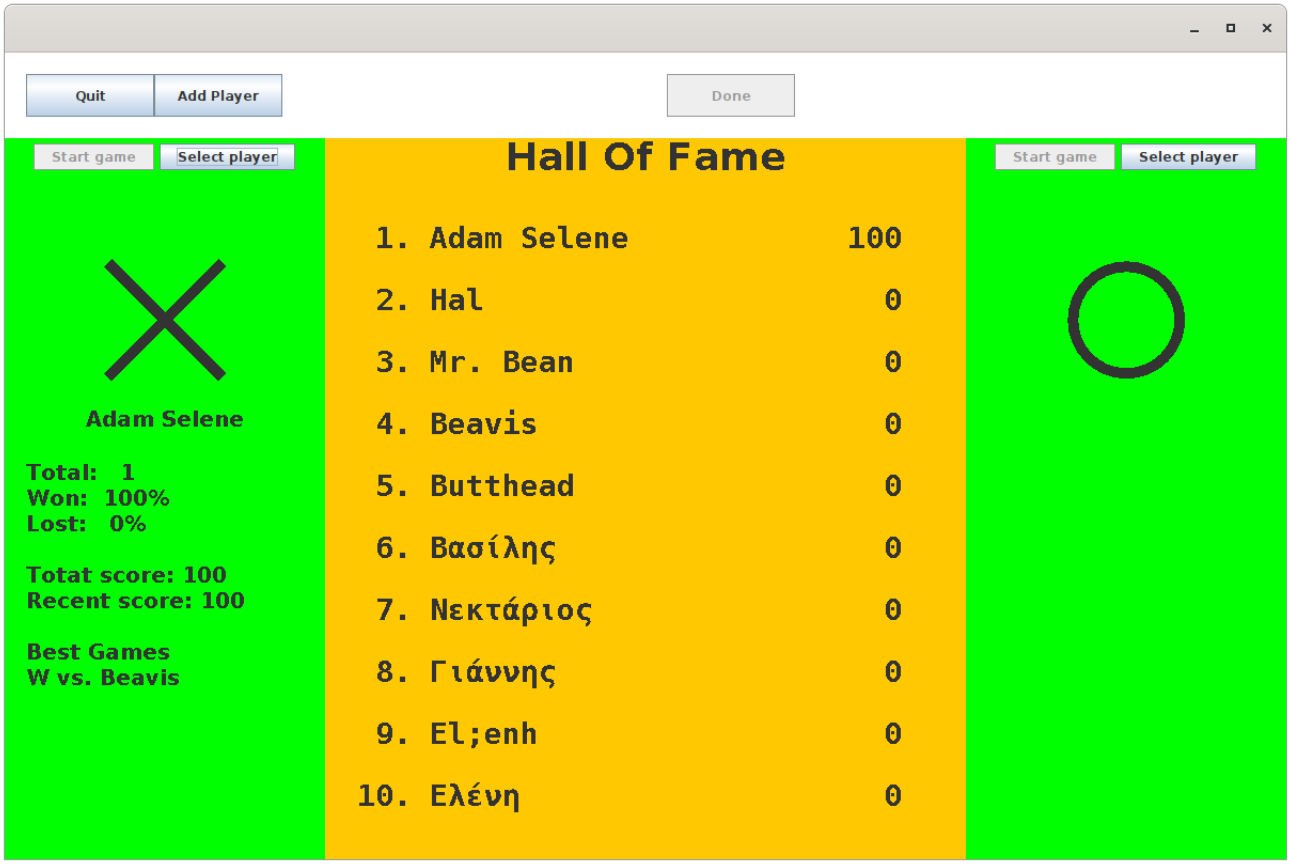

Figure 2: Αρχική οθόνη με ένα παίκτη επιλεγμένο

- Δεξιά και αριστερά υπάρχουν δύο στήλες με τις αντίθετες πλευρές του τραπεζιού. Στα αριστερά είναι ο παίκτης που παίζει X και στα δεξιά ο παίκτης που παίζει O. Υπάρχουν 2 buttons, ένα για να αλλάζει τον παίκτη και ένα για να ξεκινά νέο παιγνίδι με αυτόν τον παίκτη.

  Όταν υπάρχει επιλεγμένος παίκτης, φαίνεται το όνομά του καθώς και τα αναλυτικά στατιστικά του. Όταν δεν υπάρχει επιλεγμένος παίκτης, το παράθυρο είναι εν πολλοίς κενό.

- Στο κέντρο βρίσκεται το τραπέζι του παιγνιδιού. Όταν δεν παίζεται κάποιο παιχνίδι, εδώ εμφανίζεται το Hall Of Fame.

Κατά την εκκίνηση του παιχνιδιού οι παίκτες κινούν εναλλάξ. Σε κάθε στιγμή, στην επάνω πλευρά του τραπεζιού φαίνεται ο παίκτης που αναμένεται να κάνει κίνηση. Δείτε την Εικ. 3. Προσέξτε ότι τα buttons για αλλαγή παίκτη και εκκίνηση νέου παιχνιδιού είναι απενεργοποιημένα.

Όταν το παιχνίδι φτάσει στο τέλος του (Εικ. 4) τυπώνεται το αποτέλεσμα και ενεργοποιείται το πλήκτρο Done. Πατώντας το ο παίκτης μεταπηδά στην αρχική οθόνη.

Για παράδειγμα, τις παραπάνω οθόνες θα μπορούσαμε να υλοποιήσουμε 5 κλάσεις view:

class MainWindow το παράθυρο όλης της εφαρμογής

class PlayerPanel τα δύο πράσινα παράθυρα που αφορούν πλευρές του τραπεζιού

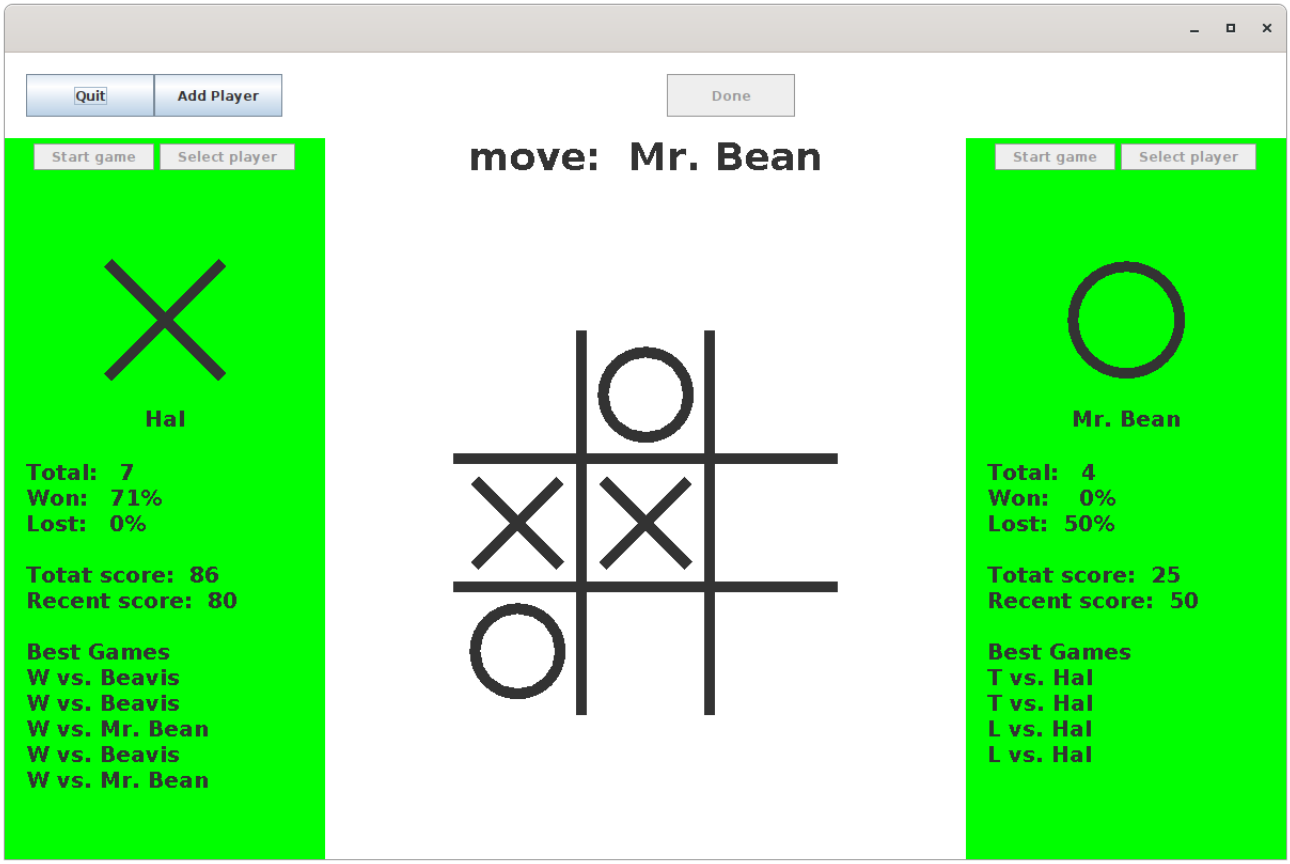

Figure 3: Οθόνη κατά τη διάρκεια του παιγνιδιού

class BannerPanel η περιοχή στην κορυφή του κυρίως παραθύρου, που περιέχει τα buttons Quit, Add Player και Done.

class HallOfFame η πορτοκαλί οθόνη που τυπώνει τους παίκτες

class GameBoard το ταμπλό του παιχνιδιού

5.2.1 Διάλογοι

Τα παράθυρα διαλόγων στις γραφικές εφαρμογές είναι πολύ διαδεδομένα, αφού έτσι διευκολύνεται η περιορισμένη αλληλεπίδραση με το χρήστη. Η βιβλιοθήκη Swing υποστηρίζει μια μεγάλη γκάμα έτοιμων διαλόγων, μέσα απο την κλάση `JOptionPane`. Διαβάστε το σχετικό tutorial

`https://docs.oracle.com/javase/tutorial/uiswing/components/dialog.html`

για να δείτε τις διάφορες επιλογές.

Εναλλακτικά, μπορείτε να χρησιμοποιήσετε το μεσαίο τραπέζι για να σας βοηθήσει στην υποστήριξη διαλόγων (αυτή η λύση είναι πιο επίπονη και δεν συνιστάται).

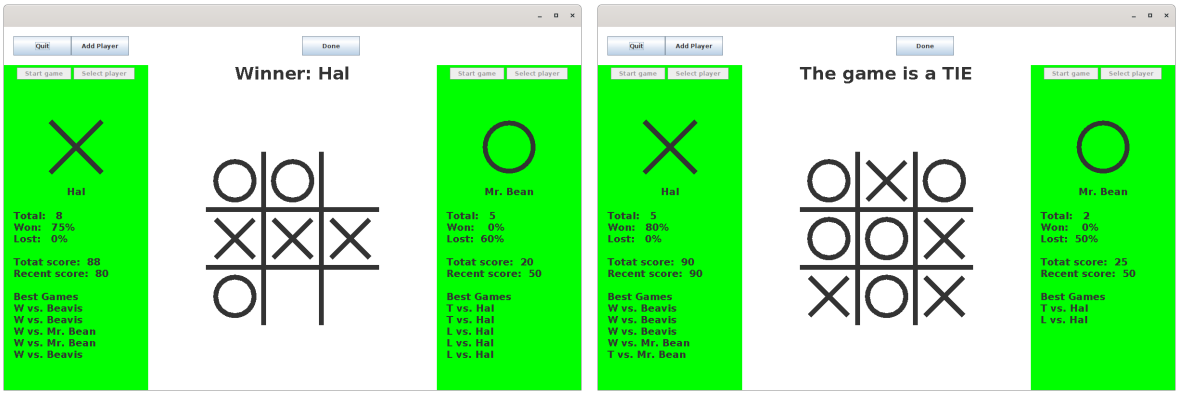

Figure 4: Τελικές οθόνες παιγνιδιού

Figure 5: Διάλογοι που χρησιμοποιεί η εφαρμογή

5.3 Έλεγχος της γραφικής διεπαφής—Controller

Σχετικά με τη σχεδίαση και υλοποίηση του controller, θα εισάγουμε δύο έννοιες που είναι πολύ χρήσιμες στη δόμηση του κώδικα.

5.3.1 Event-Condition-Action Rules (ECA rules)

Ένας τρόπος να φανταστούμε τη λογική του controller είναι ως **μια συλλογή από κανόνες ECA**. Κάθε κανόνας έχει 3 μέρη:

Event e το οποίο είναι κάποιο είδος event που παράγει το Swing, π.χ., ActionEvent, KeyboardEvent κλπ. Για κάθε είδος event υπάρχουν πολλοί κανόνες.

Condition c μια boolean συνθήκη που εξαρτάται από τα πεδία του event. **Το πιο σημαντικό πεδίο ενός event είναι το `source`** που αναφέρεται στο component που "παρήγαγε" το event (π.χ., το button που δημιούργησε το action, το παράθυρο που δέχτηκε το πάτημα πλήκτρου, κλπ). Άλλα πεδία του event μπορούν να είναι π.χ., το πλήκτρο που πατήθηκε, κλπ.

Η λογική του condition είναι ώς "προσυνθήκη για το action".

Στο Swing, καθώς κάθε widget (event source) έχει συνήθως το δικό του listener, ένα μέρος του condition είναι "δεδομένο" από τον listener που καλέστηκε.

Action A είναι ένα block κώδικα που περιλαμβάνει 3 ειδών ενέργειες:

- Αλλαγή του μοντέλου: αυτού του είδους οι ενέργειες θα πρέπει να είναι πολύ σύντομες, π.χ., κλήση μιας ή άντε 2 μεθόδων πάνω στο μοντέλο.

- Αλλαγή στα views: τα views έχουν κι αυτά κατάσταση, για παράδειγμα, ένα button είναι `enabled` ή όχι, το main view δείχνει το ένα ή το άλλο περιεχόμενο, κλπ.
- Repaint: ζητά από κάποια views να ανανεώσουν αυτό που δείχνουν. Σε μικρές εφαρμογές όπως το project μας, συνήθως αρκεί να καλέσει κανείς τη repaint() στο κυρίως παράθυρο. Σε μεγαλύτερες εφαρμογές, χρησιμοποιείται πιο στοχευμένος κώδικας.

Στο Swing, είναι μεγάλος ο πειρασμός να βάλουμε τον κώδικα των actions μέσα στους listeners που ορίζουμε στα διάφορα views. **Αποφύγετε τον πειρασμό με κάθε τρόπο**, και φτιάξτε ένα ξεχωριστό αντικείμενο, μέσα στο οποίο θα συγκεντρώσετε όλα τα actions ως μεθόδους.

5.3.2 Διάγραμμα καταστάσεων—State chart

Μια δεύτερη είναι πολύ χρήσιμη η έννοια του διαγράμματος καταστάσεων. Το διάγραμμα καταστάσεων είναι μια ιδεατή περιγραφή της συμπεριφοράς του προγράμματός μας, σε σχέση με τις αντιδράσεις του σε διάφορα events—εισόδους του χρήστη.

Ένα διάγραμμα καταστάσεων περιλαμβάνει κόμβους που ονομάζονται **καταστάσεις (states)**, και κατευθυνόμενες ακμές (βελάκια) μεταξύ καταστάσεων που ονομάζονται **μεταβάσεις (transitions)**. Επίσης, οι καταστάσεις μπορούν να ομαδοποιούνται σε **υπερ-καταστάσεις (superstates)**.

Για παράδειγμα, η εφαρμογή σας μπορεί να έχει το διάγραμμα καταστάσεων του Σχ. 6. Στο διάγραμμα αυτό μπορείτε να παρατηρήσετε την αντιστοιχία μεταξύ των μεταβάσεων και των buttons της εφαρμογής.

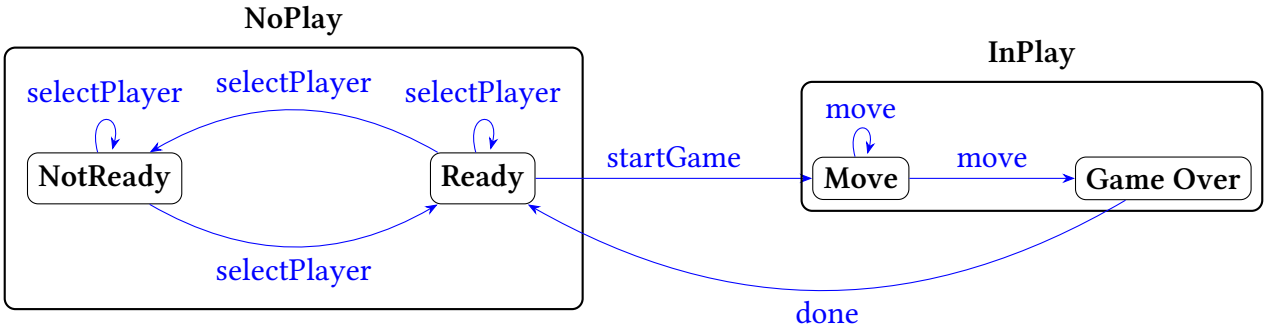

Figure 6: Ενδεικτικό διάγραμμα καταστάσεων για την εφαρμογή μας. Με μαύρο οι καταστάσεις και οι υπερ-καταστάσεις, με μπλε οι μεταβάσεις.

Γενικά:

- Οι καταστάσεις αντιστοιχούν σε διαφορετικές "φάσεις΄΄ της λειτουργίας της εφαρμογής. Αυτές **αντιστοιχούν σε συγκεκριμένες τιμές στις μεταβλητές του μοντέλου**.

  Για παράδειγμα, κατά την εκκίνηση η εφαρμογή ξεκινά από την κατάσταση **NotReady**, όπου οι δύο παίκτες (που φαίνονται στα δυο PlayerPanels) δεν έχουν ακόμη οριστεί (δλδ. τουλάχιστον ένα από τα δύο πάνελ είναι κενό).

- Οι μεταβάσεις **αντιστοιχούν σε events** που αλλάζουν (μέσα από τα actions) τις τιμές των μεταβλητών του μοντέλου.

- Όταν για μια κατάσταση δεν υπάρχει κάποια μετάβαση, το αντίστοιχο event θα πρέπει να αγνοείται. Ακόμη καλύτερα, θα πρέπει να ρυθμίζονται τα views έτσι ώστε να μην μπορούν καν να το στείλουν (π.χ., με το `setEnabled(false)` στα buttons).

Αξίζει να επαναλάβουμε ότι το διάγραμμα καταστάσεων είναι μια *ιδεατή περιγραφή της συμπεριφοράς*, που σκοπό έχει να μας βοηθά σαν προγραμματιστές να περιγράφουμε (για εμάς και τους συνεργάτες μας, στην τεκμηρίωση του κώδικα), την προσδοκώμενη συμπεριφορά του προγράμματός μας.

6 Παραδοτέα-Βαθμολογία

Θα πρέπει να παραδώσετε τον κώδικά σας που να περιλαμβάνει:

1. Την υλοποίηση του domain model (παίκτες, ιστορικό, σωστοί υπολογισμοί του σκορ κλπ) που θα σώζεται σε αρχείο
2. Την υλοποίηση του board του παιχνιδιού με τις αντίστοιχες λειτουργίες (νόμιμες κινήσεις, επόμενη κίνηση, αναγνώριση τελικού board–game over, κλπ).
3. Την υλοποίηση του αλγορίθμου AI για τον τέλειο παίκτη.
4. **Unit tests** για όλα τα παραπάνω‼!
5. Την υλοποίηση των views.
6. Την υλοποίηση του controller.

Επειδή το project σας είναι απαιτητικό, θα πρέπει να αρχίσετε αμέσως την υλοποίηση. Η βαθολόγηση του project θα γίνει με τα εξής κριτήρια:

- Επίδειξη των καλών πρακτικών του αντικειμενοστραφούς προγραμματισμού στις κλάσεις, με καλή χρήση κληρονομικότητας, πολυμορφισμού, διαχωρισμού των θεμάτων, χρήση πακέτων.
- Επίδειξη της ορθότητας των υλοποιημένων κλάσεων μέσω Unit tests. Δεν πρέπει να αρκεστείτε στο οτι το πρόγραμμά σας "φαίνεται να δουλεύει".

  Ακόμη και project που δεν δουλεύουν σωστά κατά την εκτέλεση, θα έχουν τουλάχιστον 70% του βαθμού, εφόσον υπάρχουν επαρκή unit tests για τον κώδικα του domain model.
- Καλή σχεδίαση της γραφικής διεπαφής, με καθαρό διαχωρισμό του κώδικα του controller από αυτόν των views.
- Συνολική εμπειρία **σταθερής** εκτέλεσης του κώδικα ως παιχνιδιού. Με άλλα λόγια, επικεντρωθείτε στη **σωστή εκτέλεση** και κατόπιν ασχοληθείτε με την εμφάνιση.

A Βοηθητικές παρατηρήσεις για το Swing

A.1 Πως να δείχνετε 2 views στην ίδια περιοχή

Για το σκοπό αυτό, μπορείτε να χρησιμοποιήσετε την παρακάτω λογική:

- Θα φτιάξετε ένα component που θα έχει ως "παιδιά" του τα view objects που θέλετε να χρησιμοποιούν την ίδια περιοχή
- Θα ορίσετε για το component σας το `CardLayout`. Τα layouts είναι μια πολύ σημαντική πτυχή των γραφικών βιβλιοθηκών όπως το Swing, αλλά δεν τα καλύπτουμε σε αυτό το μάθημα—όχι επειδή είναι "δύσκολα", απλά λόγω χρόνου. Σας ενθαρρύνω να τα διαβάσετε μόνοι σας από το documentation.

Ένα παράδειγμα κώδικα που μπορείτε να χρησιμοποιήσετε είναι το παρακάτω:

```
1 package tuctactoe.gui;
2
3 import java.awt.CardLayout;
4 import javax.swing.JPanel;
5
6 public class MainPanel extends JPanel {
7
8     /* The controller */
9     GameUI ui;
10
11    /* The hall—of—fame view */
12    HallOfFame hallOfFame;
13
14    /* The game board view */
15    GameBoard gameBoard;
16
17    final String HOF = "HALLOFFAME";
18    final String BOARD = "GAMEBOARD";
19
20    /* Constructor takes controller and bounds */
21    public MainPanel(GameUI ui, int width, int height) {
22        super(new CardLayout());
23
24        this.ui = ui;
25        setSize(width, height);
26        setOpaque(true);
27
28        hallOfFame = new HallOfFame(ui);
29        add(hallOfFame, HOF);
30
31        gameBoard = new GameBoard(ui, width, height);
```

```
32          add(gameBoard, BOARD);
33      }
34
35      /* Show the board if true, the hall of fame if false */
36      public void setBoardVisible(boolean flag) {
37          CardLayout cl = (CardLayout) getLayout();
38          cl.show(this, flag? BOARD: HOF);
39      }
40  }
```