ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ
ΣΧΟΛΗ ΟΙΚΟΝΟΜΙΑΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Ανάπτυξη και αξιολόγηση αλγορίθμου δημιουργίας ομάδων σε περιβάλλον παιχνιδιών

Τοπαλίδης Ιάσονας
2022201600174

Επιβλέπων:
Κωνσταντίνος Βασιλάκης
Καθηγητής

ΤΡΙΠΟΛΗ, ΣΕΠΤΕΜΒΡΙΟΣ 2021

ΠΕΡΙΛΗΨΗ

Στα online παιχνίδια πολλών χρηστών αλλά και σε άλλες ομαδικές δραστηριότητες που προϋποθέτουν συνεργασία μεταξύ χρηστών, ένα σημαντικό ζητούμενο είναι η κατάλληλη διαμόρφωση ομάδων. Οι ομάδες θα πρέπει να δημιουργηθούν κατά τρόπο ώστε αφ' ενός να γίνονται σεβαστές οι επιθυμίες των χρηστών για συνεργασία (ή αποφυγή συνεργασίας) μεταξύ τους και αφ' ετέρου να μεγιστοποιείται η ικανότητα συνεργασίας και η παραγωγικότητα των ομάδων.

Στην παρούσα πτυχιακή εργασία θεωρούμε ένα διαδικτυακό περιβάλλον συνεργασίας, όπου οι χρήστες εργάζονται ανά ζεύγη και αφού συνεργαστούν μεταξύ τους, ο καθένας τους βαθμολογεί την ποιότητα της εργασίας και τη συνεργατικότητα του άλλου. Επιπρόσθετα, οι χρήστες έχουν τη δυνατότητα να τοποθετηθούν θετικά, αρνητικά ή ουδέτερα ως προς το ενδεχόμενο συνεργασίας με άλλους χρήστες. Στο περιβάλλον αυτό, αναπτύχθηκε ένας αλγόριθμος ο οποίος χρησιμοποιεί τεχνικές γραμμικού προγραμματισμού, προκειμένου να σχηματίσει τα βέλτιστα δυνατά ζευγάρια ανάλογα με βαθμολογίες που έχουν αποδώσει οι παίκτες μεταξύ τους καθώς και τις δηλώσεις πρόθεσης συνεργασίας. Στο κείμενο της πτυχιακής παρουσιάζονται αναλυτικά η δομή του συστήματος, οι αλγόριθμοι και τεχνικές που χρησιμοποιήθηκαν, και αναλύεται η απόδοση του συστήματος με βάση τον χρόνο εκτέλεσης του αλγόριθμου διαμόρφωσης ζευγαριών για διάφορες περιπτώσεις. Τέλος, παρουσιάζονται ενδεικτικές επεκτάσεις του συστήματος.

Λέξεις-Κλειδιά: Ομάδες χρηστών, γραμμικός προγραμματισμός, πρόβλημα μεγιστοποίησης, μέθοδος simplex, Java, glpk, scpsolver.

ABSTRACT

In the context of multiplayer online games, but in many other group activities that entail collaboration between user, appropriate group formation is a critical issue. Groups should be formulated in a way that on the one hand user collaboration (or collaboration avoidance) preferences are respected, and on the other hand collaboration potential and productivity of groups is maximized.

In the current thesis, we consider an internet-based collaboration environment, where users work in pairs and, after their collaboration, each of them rates the quality of work and the quality of collaboration of his/her partner. Moreover, users have the option to express a positive, negative or neutral opinion towards the possibility of cooperating with individual users. For such an environment, we have developed an algorithm that employs linear programming techniques, in order to create an optimal user pair formulation, considering the ratings that users have assigned to other users and collaboration intention statements. We present in detail the system structure, the algorithms and techniques employed, and we analyze the performance of the system in terms of the time needed to formulate user pair assignments in various indicative cases. Finally, potential future extensions are outlined.

Keywords: user groups, linear programming, maximization problem, μέθοδος simplex method, Java, glpk, scpsolver.

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΕΥΡΕΤΗΡΙΟ ΕΙΚΟΝΩΝ

1 ΕΙΣΑΓΩΓΗ

Η δημιουργία συμβατών ζευγαριών ή η κατανομή ατόμων σύμφωνα με προτιμήσεις τους είναι ένα ζήτημα που εμφανίζεται σε πολλές περιπτώσεις, όπως ενδεικτικά:

- Δημιουργία ομάδων για παιχνίδια
- Δημιουργία ζευγαριών για υπηρεσίες γνωριμιών
- Κατανομή ατόμων σε βέλτιστες θέσεις εργασίας.

Η λύση του προβλήματος δεν είναι εμφανής, εύκολη ή βέλτιστη τις περισσότερες φορές, καθώς μπορεί να υπάρχουν αντικρουόμενες προτιμήσεις από πλευράς χρηστών (π.χ. ο χρήστης Α επιθυμεί να συνεργαστεί με τον Β αλλά ο Β δεν επιθυμεί να συνεργαστεί με τον Α), ενώ παράλληλα η βελτιστοποίηση στόχων σε συλλογικό επίπεδο μπορεί να οδηγεί σε ημιβέλτιστες αποφάσεις σε επίπεδο μεμονωμένων ζευγών (π.χ. για να βελτιστοποιείται το επίπεδο συνεργασιμότητας του συνόλου των αναθέσεων μπορεί να χρειάζεται να δημιουργηθεί κάποιο ζεύγος όπου ο ένας από τους χρήστες δεν επιθυμεί να συνεργαστεί με τον άλλο).

Στην παρούσα πτυχιακή εργασία αντιμετωπίζουμε το πρόβλημα της δημιουργίας ζευγαριών παικτών από έναν γύρο παιχνιδιού στον επόμενο, με χρήση τεχνικών γραμμικού προγραμματισμού. Πιο συγκεκριμένα, θεωρούμε ένα διαδικτυακό περιβάλλον συνεργασίας, όπου οι χρήστες εργάζονται ανά ζεύγη και αφού συνεργαστούν μεταξύ τους, ο καθένας τους βαθμολογεί την ποιότητα της εργασίας και τη συνεργατικότητα του άλλου. Επιπρόσθετα, οι χρήστες έχουν τη δυνατότητα να τοποθετηθούν θετικά, αρνητικά ή ουδέτερα ως προς το ενδεχόμενο συνεργασίας με άλλους χρήστες. Στο περιβάλλον αυτό, αναπτύχθηκε ένας αλγόριθμος ο οποίος χρησιμοποιεί τεχνικές γραμμικού προγραμματισμού, προκειμένου να σχηματίσει τα βέλτιστα δυνατά ζευγάρια ανάλογα με βαθμολογίες που έχουν αποδώσει οι παίκτες μεταξύ τους καθώς και τις δηλώσεις πρόθεσης συνεργασίας.

Η εργασία αποτελείται από έναν εξυπηρετητή (server) ο οποίος αναμένει αιτήματα διαμόρφωσης ζευγαριών παικτών, όπου κάθε αίτημα περιλαμβάνει, τους υποψήφιους παίκτες μαζί με τις βαθμολογίες τους στα επιμέρους κριτήρια και τις προτιμήσεις του κάθε παίκτη σχετικά με το με ποιους άλλους παίκτες επιθυμεί να συνεργαστεί. Μόλις ο εξυπηρετητής λάβει τα δεδομένα σε μορφή αίτησης από κάποιον εξυπηρετούμενο (client) διαμορφώνει με

βάση τα στοιχεία που έχει ένα πρόγραμμα βελτιστοποίησης γραμμικού προγραμματισμού, η επίλυση του οποίου είναι το επιθυμητή σύνολο αναθέσεων.

Το πρόβλημα γραμμικού προγραμματισμού που δημιουργείται είναι ένα πρόβλημα μεγιστοποίησης μιας αντικειμενικής συνάρτησης και για τη διαμόρφωσή του χρησιμοποιείται ένα σύνολο παραμέτρων. Τέτοιες παράμετροι είναι τα βάρη των πιθανών ζευγαριών και οι προτιμήσεις των παιχτών, που προκύπτουν από τις βαθμολογίες που βάζουν οι παίχτες μεταξύ τους και που αντλούνται από το περιεχόμενο της αίτησης. Κατόπιν, το πρόβλημα επιλύεται με χρήση της μεθόδου Simplex, υπολογίζοντας το βέλτιστο σύνολο ζευγαριών παικτών.

Το υπόλοιπο κείμενο της πτυχιακής διαρθρώνεται σε κεφάλαια ως ακολούθως: στο κεφάλαιο 2 θα αναλυθούν οι τεχνολογίες , το προγραμματιστικό περιβάλλον και οι μαθηματικές βιβλιοθήκες που χρησιμοποιήθηκαν. Στο κεφάλαιο 3 θα αναλυθούν οι τεχνολογίες, οι γλώσσες προγραμματισμού, υπηρεσίες (services) και βιβλιοθήκες που χρησιμοποιήθηκαν καθώς και η δομή του προγράμματος, η δομή των συναρτήσεων που υλοποιήθηκαν και η δομή των αιτημάτων και των απαντήσεων του server. Στο κεφάλαιο 4 θα παρουσιαστούν παραδείγματα εκτέλεσης του αλγορίθμου, χρόνοι εκτέλεσης του αλγορίθμου καθώς και ειδικές περιπτώσεις εκτέλεσης. Τέλος στο κεφάλαιο 5 παρατίθεται μία σύνοψη της πτυχιακής εργασίας και σκιαγραφούνται πιθανές μελλοντικές επεκτάσεις του συστήματος.

2 ΥΠΟΒΑΘΡΟ ΚΑΙ ΤΕΧΝΟΛΟΓΙΕΣ

Σε αυτό το κεφάλαιο δίδεται μία συνοπτική περιγραφή των τεχνολογιών και εργαλείων (προγραμματιστικό περιβάλλον, γλώσσα προγραμματισμού, μαθηματικές βιβλιοθήκες κ.λπ.) που χρησιμοποιήθηκαν για τη δημιουργία του συστήματος και την υλοποίηση και την αξιολόγηση του αλγορίθμου.

2.1 Java

Η γλώσσα προγραμματισμού που χρησιμοποιήθηκε για την ανάπτυξη του server καθώς και για την υλοποίηση του είναι η Java. Η Java είναι μία γλώσσα προγραμματισμού γενικού σκοπού. Η πρώτη δημόσια έκδοση της Java κυκλοφόρησε από τη Sun Microsystems το 1996. Βασικό χαρακτηριστικό της γλώσσας είναι η λειτουργία Write Once, Run Anywhere (WORA) [1]. Αυτό σημαίνει πως τα προγράμματα γραμμένα σε αυτή την γλώσσα μόλις ολοκληρωθούν και περάσουν από τον μεταγλωττιστή (compiler) μπορούν να «τρέχουν» σε οποιαδήποτε πλατφόρμα, με βασική προϋπόθεση να είναι διαθέσιμη για την πλατφόρμα η εικονική μηχανή της Java (Java virtual machine, JVM).

Στην επιλογή αυτής της γλώσσας έναντι άλλων αντικειμενοστραφών γλωσσών όπως η C++ ή η Python συντέλεσαν οι εξής κύριοι παράγοντες:

1. Η Java προλαμβάνει σφάλματα και στο επίπεδο μεταγλώττισης (compile-time) σε αντίθεση με την Python που όλα τα σφάλματα εκδηλώνονται στο επίπεδο εκτέλεσης (run-time).
2. Η Java είναι εύκολα μεταφέρσιμη από ένα σύστημα σε ένα άλλο όπως και η Python, αλλά σε αντίθεση με την Python είναι πιο γρήγορη στον χρόνο εκτέλεσης καθώς είναι μεταγλωττιζόμενη γλώσσα που περιλαμβάνει περαιτέρω τεχνικές βελτιστοποίησης του χρόνου εκτέλεσης (π.χ. just-in-time compiler [2], hotspot technology [3]), ενώ η Python είναι διερμηνευόμενη γλώσσα.

2.2 Python

Για την ανάπτυξη του δοκιμαστικού προγράμματος δημιουργίας τυχαίων παιχτών, με τυχαίες βαθμολογίες και αξιολογήσεις, έγινε χρήση της γλώσσας προγραμματισμού Python. Η

Python είναι μια διερμηνευόμενη γλώσσα προγραμματισμού υψηλού επιπέδου γενικής χρήσης. Η σχεδιαστική της φιλοσοφία δίνει έμφαση στην αναγνωσιμότητα του κώδικα με τη χρήση εσοχής σε αντίθεση με άλλες γλώσσες στις οποίες ο κώδικας διαχωρίζεται με χρήση του ερωτηματικού «;» και τα μπλοκ κώδικα οριοθετούνται είτε με χαρακτήρες (π.χ. {}) είτε με δεσμευμένες λέξεις (π.χ. begin-end). Οι γλωσσικές της κατασκευές καθώς και η αντικειμενοστρεφής προσέγγιση στοχεύουν να βοηθήσουν τους προγραμματιστές να γράψουν σαφή, λογικό κώδικα τόσο για έργα μικρής όσο και για έργα μεγάλης κλίμακας. Η Python είναι γλώσσα με δυναμικούς τύπους μεταβλητών (dynamically typed)[1] και διαθέτει συλλογή απορριμμάτων (garbage collector)[2]. Υποστηρίζει πολλά παραδείγματα προγραμματισμού, συμπεριλαμβανομένου του δομημένου (ιδιαίτερα διαδικαστικού), αντικειμενοστραφούς και λειτουργικού προγραμματισμού [4].

Ο λόγος που επιλέχθηκε αυτή η γλώσσα για την δημιουργία του δοκιμαστικού προγράμματος, είναι η ευκολία και η απλότητα που προσφέρει, για γρήγορη δημιουργία δοκιμαστικών προγραμμάτων.

2.3 Eclipse

Το ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) που χρησιμοποιήθηκε για την ανάπτυξη του αλγορίθμου είναι το Eclipse. Το Eclipse είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) ανοιχτού κώδικα γραμμένο κυρίως σε Java, με κύρια χρήση την ανάπτυξη εφαρμογών γραμμένες σε Java, αλλά με χρήση επεκτάσεων υποστηρίζονται και άλλες γλώσσες προγραμματισμού ανάμεσα στις οποίες είναι οι: C, C++, C#, Fortran, Julia, Lua, Pearl, PHP, Python, Rust κ.ά. [5] Το περιβάλλον προγραμματισμού Eclipse διαθέτει πολλά σημαντικά χαρακτηριστικά, όπως π.χ. συνεργασία με αποθετήρια διαχείρισης εκδόσεων κώδικα (GitLab [6], GitHub [7], SVN [8] κ.λπ.), έλεγχο του κώδικα για ζητήματα που χρήζουν προσοχής από τον προγραμματιστή μέσω του SonarLint [9], πλήρη υποστήριξη του Maven [10], ενσωματωμένο περιβάλλον αποσφαλμάτωσης (debugging) κ.ο.κ.

[1] Ο έλεγχος του προγράμματος γίνεται κατά την εκτέλεση και όχι κατά την μετάφραση (compile)
[2] Αυτόματη διαχείριση μνήμης

2.4 Αρχιτεκτονική REST

Η αρχιτεκτονική REST (Representational State Transfer) δημιουργήθηκε για να καθοδηγήσει τον σχεδιασμό και την ανάπτυξη της αρχιτεκτονικής για τον Παγκόσμιο Ιστό. Το μοτίβο REST καθορίζει ένα σύνολο περιορισμών για το πώς θα πρέπει να συμπεριφέρεται η αρχιτεκτονική ενός κατανεμημένου συστήματος σε κλίμακα Διαδικτύου, όπως ο Ιστός. Η αρχιτεκτονική δομή REST δίνει έμφαση στην επεκτασιμότητα των αλληλεπιδράσεων μεταξύ των τμημάτων μιας δομής, ομοιόμορφες διεπαφές, ανεξάρτητη ανάπτυξη επεκτάσεων και τη δημιουργία μιας πολυεπίπεδης αρχιτεκτονικής για τη διευκόλυνση της προσωρινής αποθήκευσης στοιχείων για τη μείωση του χρόνου απόκρισης μεταξύ αιτήματος και απάντησης που λαμβάνει ο χρήστης, την επιβολή ασφάλειας και την ενσωμάτωση παλαιών συστημάτων.

Η αρχιτεκτονική REST χρησιμοποιείται σε όλη τη βιομηχανία λογισμικού και είναι ένα ευρέως αποδεκτό σύνολο οδηγιών για τη δημιουργία stateless,[3] αξιόπιστων API. Ένα API που υπακούει στους περιορισμούς REST περιγράφεται ανεπίσημα ως RESTful. Τα RESTful API συνήθως βασίζονται σε μεθόδους Hypertext Transfer Protocol (HTTP) για πρόσβαση σε πόρους μέσω παραμέτρων που κωδικοποιούνται με Uniform Resource Locator (URL) και τη χρήση JSON ή XML για τη μετάδοση δεδομένων.

Ο στόχος του REST είναι να αυξήσει την απόδοση, την επεκτασιμότητα, την απλότητα, την τροποποίηση, τη μεταφερσιμότητα και την αξιοπιστία. Αυτό επιτυγχάνεται με την τήρηση των αρχών REST, όπως αρχιτεκτονική πελάτη - διακομιστή, statelessness, προσωρινή μνήμη, χρήση πολυεπίπεδου συστήματος, υποστήριξη κώδικα κατά παραγγελία και χρήση ομοιόμορφης διεπαφής. Αυτές οι αρχές πρέπει να τηρούνται για να ορίζεται το σύστημα ως RESTful [11].

2.5 JSON

Το JSON (JavaScript Object Notation) είναι μια ανοιχτή τυπική μορφή αρχείου και μορφή ανταλλαγής δεδομένων που χρησιμοποιεί αναγνώσιμο από τον άνθρωπο κείμενο για την

[3] Stateless : Κάθε αίτημα από τον πελάτη προς τον διακομιστή πρέπει να περιέχει όλες τις απαραίτητες πληροφορίες για την κατανόηση του αιτήματος.

αποθήκευση και τη μετάδοση αντικειμένων δεδομένων που αποτελούνται από ζεύγη χαρακτηριστικών-τιμών και πίνακες (ή άλλες σειριοποιήσιμες τιμές). Είναι μια κοινή μορφή δεδομένων με ένα ευρύ φάσμα λειτουργιών στην ανταλλαγή δεδομένων, συμπεριλαμβανομένης της επικοινωνίας εφαρμογών Ιστού με διακομιστές.

Το πρότυπο JSON είναι μια μορφή δεδομένων ανεξάρτητη από τις διάφορες γλώσσες προγραμματισμού. Προέρχεται από την JavaScript, αλλά πολλές σύγχρονες γλώσσες προγραμματισμού περιλαμβάνουν κώδικα για τη δημιουργία και ανάλυση δεδομένων μορφής JSON.

Το πρότυπο JSON εξελίχθηκε από την ανάγκη για stateless 4πρωτόκολλο, σε πραγματικό χρόνο από διακομιστή σε πρόγραμμα περιήγησης χωρίς τη χρήση πρόσθετων προγράμματος περιήγησης όπως εφαρμογές Flash ή Java [12].

2.6 Swagger

Η αρχιτεκτονική REST απαιτεί την ύπαρξη ενός εξυπηρετητή διαδικτύου (web server) που θα υποδέχεται τα αιτήματα και θα μεριμνά για την εκτέλεσή τους, και αντίστοιχα κώδικα που θα μπορεί να χρησιμοποιηθεί από τους εξυπηρετούμενους (clients) για την υποβολή των αιτημάτων. Οι δύο κώδικες θα πρέπει να είναι συγχρονισμένοι ως προς τη δομή δεδομένων που χρησιμοποιούν για την αναπαράσταση των αιτημάτων και των απαντήσεων.

Για τη δημιουργία του εξυπηρετητή διαδικτύου έγινε χρήση του Swagger [13]. Το Swagger είναι ένα εργαλείο που επιτρέπει την συγγραφή δικών μας διεπαφών προγραμματισμού εφαρμογών (APIs) σε επίπεδο διαδικτύου με τρόπο αναγνώσιμο για τις μηχανές και με μεγάλο βαθμό αναγνωσιμότητας από τους ανθρώπους/προγραμματιστές. Μία προδιαγραφή Swagger μπορεί να γραφεί είτε σε γλώσσα YAML [14], είτε σε μορφή JSON, με τη YAML να χρησιμοποιείται άμεσα από ανθρώπους/προγραμματιστές λόγω

[4] Stateless : Κάθε αίτημα από τον πελάτη προς τον διακομιστή πρέπει να περιέχει όλες τις απαραίτητες πληροφορίες για την κατανόηση του αιτήματος.

υψηλής αναγνωσιμότητας και τη μορφή JSON να χρησιμοποιείται κυρίως από μηχανές, για διαλειτουργικότητα.

Το Swagger προσφέρει επιπλέον τη δυνατότητα αυτόματης δημιουργίας κώδικα (Swagger Codegen) τόσο για τους εξυπηρετητές όσο και για τους εξυπηρετούμενους σε πολλές γλώσσες προγραμματισμού. Μία προδιαγραφή σε Swagger περιλαμβάνει τον ορισμό των διεπαφών, συμπεριλαμβάνοντας τις παραμέτρους που δέχονται, τα αποτελέσματα που επιστρέφουν και σχετική τεκμηρίωση, καθώς και προσδιορισμό των τύπων δεδομένων που χρησιμοποιούνται. Καθώς η γλώσσα προγραμματισμού που επιλέχθηκε για την υλοποίηση της πτυχιακής είναι η Java, ο server που χρησιμοποιήθηκε από το Swagger Codegen είναι αυτός του Spring framework και συγκεκριμένα το Spring Boot Application.

2.7 Spring framework

Το Spring είναι ένα ευέλικτο, ελαφρύ, ασφαλές και εύκολα επεκτάσιμο πλαίσιο (framework) γραμμένο σε Java, με τις εξής ιδιότητες:

- Ασύγχρονη λειτουργία, για καλύτερη διαχείριση των υπολογιστικών πόρων
- Εύκολη συγγραφή/μεταφορά κώδικα σε περιβάλλον υπολογιστικού νέφους (cloud)
- Εύκολη δημιουργία διαδικτυακών εφαρμογών (web applications)
- κ.ά.

Το Spring Framework έχει δημιουργήσει πολλά υπο-έργα, κάθε ένα από τα οποία έχει συγκεκριμένη εστίαση. Από τα έργα αυτά σταχυολογούνται ενδεικτικά τα ακόλουθα:

- Spring Boot: Το Spring Boot διευκολύνει τη δημιουργία αυτόνομων εφαρμογών με βάση το Spring Framework.
- Spring Cloud: Το Spring Cloud παρέχει εργαλεία στους προγραμματιστές για να χτίσουν γρήγορα μερικά από τα κοινά μοτίβα σε κατανεμημένα συστήματα (π.χ. διαχείριση διαμόρφωσης, ανακάλυψη υπηρεσιών, διακόπτες κυκλώματος, έξυπνη δρομολόγηση κ.ά.).

- Spring Security: Το Spring Security είναι ένα ισχυρό και εξαιρετικά προσαρμόσιμο πλαίσιο ελέγχου ταυτότητας και ελέγχου πρόσβασης. Είναι το προκαθορισμένο πρότυπο για τη διασφάλιση εφαρμογών που βασίζονται στο Spring.
- Spring GraphQL: Το Spring GraphQL παρέχει υποστήριξη για εφαρμογές Spring που βασίζονται σε GraphQL Java.[5]
- κ.ά.

Το Spring Boot, που χρησιμοποιείται σε αυτή την εργασία, είναι ένα εργαλείο που καθιστά την ανάπτυξη διαδικτυακών εφαρμογών και μικροϋπηρεσιών με το Spring Framework γρηγορότερη και ευκολότερη μέσω τριών βασικών δυνατοτήτων:

- Αυτόματη διαμόρφωση (Autoconfiguration)
- Προσωπική προσέγγιση (Opinionated approach)
- Αυτόνομες εφαρμογές (Standalone applications)

Με τον όρο *αυτόματη διαμόρφωση (autoconfiguration)* εννοείται ότι η αρχικοποίηση των εφαρμογών γίνεται με προκαθορισμένες εξαρτήσεις (dependencies) χωρίς να χρειάζεται χειροκίνητη παρέμβαση. Χάρη στις δυνατότητες αυτόματης διαμόρφωσης το Spring Boot ενσωματώνει πακέτα τόσο του ίδιου το Spring Framework όσο και τρίτων κατασκευαστών, με τρόπο που βοηθάει στην αποφυγή σφαλμάτων. Με τον όρο *προσωπική προσέγγιση* εννοείται ότι το Spring Boot ακολουθώντας την δική του κρίση επιλέγει ποια πακέτα θα εγκαταστήσει αντί να ζητηθεί από τον χρήστη χειροκίνητη παραμετροποίηση. Τέλος, με τον όρο *αυτόνομες εφαρμογές* εννοείται ότι το Spring Boot δημιουργεί εφαρμογές που δεν απαιτούν κάποιον εξωτερικό web server καθώς υπάρχει ήδη ενσωματωμένος web server (Tomcat [15] ή Netty [16]).

2.8 Γραμμικός Προγραμματισμός

Ο γραμμικός προγραμματισμός είναι μια τεχνική μαθηματικής μοντελοποίησης για επίλυση προβλημάτων. Στον γραμμικό προγραμματισμό, υπάρχει μία αντικειμενική συνάρτηση, η οποία αντικατοπτρίζει την ποιότητα της λύσης και που εκφράζεται ως γραμμική παράσταση,

[5] Το GraphQL είναι μια γλώσσα ερωτημάτων που είναι προσβάσιμη μέσω API, καθώς και και ένα λογισμικό για την αποτίμηση αυτών των ερωτημάτων έναντι συλλογών δεδομένων.

βελτιστοποιείται (ελαχιστοποιείται ή μεγιστοποιείται), ενώ παράλληλα τηρούνται διάφοροι περιορισμοί, που περιλαμβάνονται στη διατύπωση του προβλήματος. Ουσιαστικά η αντικειμενική συνάρτηση καθορίζει την ποσότητα που πρέπει να βελτιστοποιηθεί και ο στόχος του γραμμικού προγραμματισμού είναι να βρει τις τιμές των μεταβλητών που μεγιστοποιούν ή ελαχιστοποιούν την αντικειμενική συνάρτηση.

Τα προβλήματα γραμμικού προγραμματισμού είναι προβλήματα που μπορούν να εκφραστούν σε κανονική μορφή ως [1]:

Υπολογισμός ενός διανύσματος y
Που μεγιστοποιεί το $k^T y$
Υπό τον περιορισμό $Ay \leq b$
Και $y \geq 0$

Τα στοιχεία του διανύσματος y είναι οι μεταβλητές των οποίων οι τιμές πρέπει να προσδιοριστούν, ενώ τα k και b είναι δεδομένα διανύσματα (η γραφή k^T υποδεικνύει ότι οι συντελεστές του διανύσματος k χρησιμοποιούνται ως μονοδιάστατος πίνακας για τον υπολογισμό του γινομένου πινάκων), και το A είναι δεδομένος πίνακας. Η συνάρτηση της οποίας η τιμή πρέπει να μεγιστοποιηθεί ή να ελαχιστοποιηθεί ($y \mapsto k^T$ σε αυτήν την περίπτωση) ονομάζεται *αντικειμενική συνάρτηση*. Οι ανισότητες $Ay \leq b$ και $y \geq 0$ είναι οι περιορισμοί που καθορίζουν έναν χώρο πάνω στον οποίο πρέπει να βελτιστοποιηθεί η αντικειμενική συνάρτηση. Σε αυτό το πλαίσιο, δύο διανύσματα είναι συγκρίσιμα όταν έχουν τις ίδιες διαστάσεις. Εάν κάθε στοιχείο στο πρώτο είναι μικρότερο από ή ίσο με το αντίστοιχο στοιχείο στο δεύτερο, τότε μπορούμε να πούμε ότι το πρώτο διάνυσμα είναι μικρότερο ή ίσο με το δεύτερο διάνυσμα.

Ο γραμμικός προγραμματισμός μπορεί να εφαρμοστεί σε διάφορους τομείς. Χρησιμοποιείται ευρέως στα μαθηματικά, σε μικρότερο βαθμό στις επιχειρήσεις και τα οικονομικά. Οι βιομηχανίες και επιχειρήσεις που χρησιμοποιούν μοντέλα γραμμικού προγραμματισμού περιλαμβάνουν τους τομείς των μεταφορών, την παραγωγή ενέργειας και τις τηλεπικοινωνίες. Ο γραμμικός προγραμματισμός έχει αποδειχθεί χρήσιμο εργαλείο στη μοντελοποίηση διαφόρων τύπων προβλημάτων στο σχεδιασμό, τη δρομολόγηση, την ανάθεση καθώς και τον προγραμματισμό.

2.9 Εργαλεία Προγραμματισμού & Βιβλιοθήκες

Για τη μοντελοποίηση και επίλυση του προβλήματος γραμμικού προγραμματισμού, καθώς και για άλλες βοηθητικές λειτουργίες, έγινε χρήση εξωτερικών βιβλιοθηκών και πακέτων. Συγκεκριμένα για το πρόβλημα του γραμμικού προγραμματισμού έγινε χρήση 3 βιβλιοθηκών:

- SCPSolver [17]
- lpsolve [18]
- GLPK [19]

Το SCPSolver είναι μία προγραμματιστική διεπαφή (API) υψηλού επιπέδου που έχει ως στόχο να κάνει ευκολότερη την ανάπτυξη και επίλυση προβλημάτων γραμμικού προγραμματισμού και να ανεξαρτητοποιήσει την μοντελοποίηση του προβλήματος από τον μηχανισμό επίλυσης (solver) του προβλήματος. Το SCPSolver δεν σκοπεύει να δημιουργήσει τη δική του μηχανή επίλυσης στη Java, αλλά να κάνει εύκολη την ενσωμάτωση ήδη υπαρχουσών μηχανών επίλυσης όπως:

- GLPK [19]
- lpsolve [18]

Για αυτή την εργασία επιλέχθηκαν αυτές οι βιβλιοθήκες λόγω του ότι είναι ανοιχτού κώδικα έναντι άλλων γρηγορότερων αλλά εμπορικών βιβλιοθηκών όπως Gurobi [20], GAMS [21], Mosek [22] κ.ά. Η προγραμματιστική διεπαφή του SCPSolver επιτρέπει τη χρήση σε δύο «επίπεδα»: Το πρώτο επίπεδο επιτρέπει τη μοντελοποίηση του προβλήματος μέσω μίας διεπαφής υψηλού επιπέδου (high level interface). Το δεύτερο επίπεδο, δημιουργεί το πρόβλημα με την χρήση διεπαφής χαμηλού επιπέδου (low level interface), στο οποίο για την μοντελοποίηση της αντικειμενικής συνάρτησης (objective function) και των περιορισμών γίνεται με χρήση πινάκων, επιτρέποντας έτσι καλύτερο έλεγχο στις αποδόσεις τιμών. Ακολουθεί ενδεικτικό παράδειγμα, ενός προβλήματος ελαχιστοποίησης, με χρήση και των δύο διεπαφών:

```
Min 6x + 5y

Subject to
3 x + 1 y >= 8
4y >= 3
2x <= 2
```

Με χρήση της χαμηλού επιπέδου διεπαφής το παραπάνω πρόβλημα μοντελοποιείται ως ακολούθως:

Στην πρώτη γραμμή γίνεται αρχικοποίηση του προβλήματος με 2 μεταβλητές, όπως αυτό προκύπτει από το πλήθος των στοιχείων που αρχικοποιούν τον πίνακα. Στις γραμμές 2 έως 4 προστίθενται οι αντίστοιχοι περιορισμοί του προβλήματος. Στην γραμμή 5 το πρόβλημα ορίζεται ως πρόβλημα ελαχιστοποίησης. Στην γραμμή 6 και 7 επιλύεται και αποθηκεύεται σε έναν πίνακα το αποτέλεσμα του προβλήματος

```
1  LinearProgram lp = new LinearProgram(new double[]{6.0,5.0});
2  lp.addConstraint(new LinearBiggerThanEqualsConstraint(new double[]{3.0,1.0}, 8.0, "c1"));
3  lp.addConstraint(new LinearBiggerThanEqualsConstraint(new double[]{0.0,4.0}, 3.0, "c2"));
4  lp.addConstraint(new LinearSmallerThanEqualsConstraint(new double[]{2.0,0.0}, 2.0, "c3"));
5  lp.setMinProblem(true);
6  LinearProgramSolver solver  = SolverFactory.newDefault();
7  double[] sol = solver.solve(lp);
```

Εικόνα 1: Παράδειγμα μοντελοποίηση προβλήματος με χρήση της διεπαφής χαμηλού επιπέδου

Με χρήση της υψηλού επιπέδου διεπαφής το ίδιο πρόβλημα μοντελοποιείται ως εξής:

Στην πρώτη γραμμή και δεύτερη ορίζεται ένα αντικείμενο καθώς και οι 2 μεταβλητές του προβλήματος. Στις γραμμές 3 έως 5 υλοποιούνται οι περιορισμοί του προβλήματος. Στην γραμμή 6 γίνεται επίλυση του προβλήματος. Με χρήση της διεπαφής ψηλού επιπέδου το πρόβλημα αρχικοποιείται ως πρόβλημα ελαχιστοποίησης.

```
1  LPWizard lpw = new LPWizard();
2  lpw.plus("x1",5.0).plus("x2",5.0);
3  lpw.addConstraint("c1",8,"<=").plus("x1",3.0).plus("x2",1.0);
4  lpw.addConstraint("c2",1,"<=").plus("x2",4.0);
5  lpw.addConstraint("c3", 2, ">=").plus("x1",2.0);
6  lwp.solve()
```

Εικόνα 2: Παράδειγμα μοντελοποίηση προβλήματος με χρήση της διεπαφής υψηλού επιπέδου

Παρατηρούμε ότι η χρήση της διεπαφής υψηλού επιπέδου επιτρέπει τη χρήση ονομάτων μεταβλητών και πιο ευέλικτο προσδιορισμό των περιορισμών.

Επιλέχθηκε η χρήση της διεπαφής χαμηλού επιπέδου διότι ήταν πιο ευχερής η απεικόνιση της αναπαράστασης των δεδομένων που λαμβανόταν στις αιτήσεις, στις δομές μοντελοποίησης του προβλήματος. Αυτό γίνεται διότι οι παράμετροι αυτοί εισάγονται σε έναν πίνακα και όπου δεν υπάρχει παράμετρος στον πίνακα εισάγεται η τιμή «0» όπως φαίνεται στην εικόνα 1 στις γραμμές 3 & 4. Σε αντίθετη περίπτωση με την χρήση του high level interface αν κάποια παράμετρος ήταν «0» θα παραλειπόταν τελείως όπως φαίνεται στην εικόνα 2 στις γραμμές 4 & 5.

2.10 Apache Benchmark

Το Apache benchmark (ab) είναι ένα εργαλείο για τη συγκριτική αξιολόγηση (benchmarking) του server. Έχει σχεδιαστεί για να δίνει μια εικόνα για το πώς λειτουργεί μία υπηρεσία κάτω από συνθήκες υψηλού φορτίου και κίνησης. Αυτό δείχνει ιδιαίτερα πόσα αιτήματα ανά δευτερόλεπτο μπορεί να εξυπηρετήσει ο server [23]. Το ab είναι ένα μονο-νηματικό (single-threaded) πρόγραμμα που εκτελείται από την γραμμή εντολών [24].

Ενδεικτικό παράδειγμα εκτέλεσης του:

```
ab -n 2000 -c 4 -H "Accept: application/json" -p test.json -T application/json http://192.168.1.6:8080/matchmaking/
```

Εικόνα 3: Παράδειγμα εκτέλεσης του apache benchmark

Η ένδειξη -n καθορίζει τον συνολικό αριθμό των αιτημάτων που θα σταλούν. Η ένδειξη -c δίνει τον αριθμό των αιτημάτων που θα αποστέλλονται παράλληλα. Η ένδειξη--H ορίζει επικεφαλίδες που θα σταλούν μαζί με το αίτημα. Η ένδειξη -p ορίζει το αρχείο που περιέχει τα δεδομένα που θα σταλούν με αίτηση Post. Η ένδειξη -T ορίζει τον τύπο των δεδομένων που θα σταλούν. Το τελευταίο όρισμα είναι η διεύθυνση στην οποία βρίσκεται ο server.

Πέρα από τις βιβλιοθήκες για το πρόβλημα της μεγιστοποίησης, έγινε χρήση και βιβλιοθηκών για την υλοποίηση βοηθητικών δομών κατά τον διαχωρισμό των δεδομένων από τη μορφή JSON που λαμβάνεται στην αίτηση. Τέτοια βιβλιοθήκη ήταν η Apache Commons Collections η οποία προσθέτει νέες δομές δεδομένων (data structures) επιταχύνοντας και διευκολύνοντας την ανάπτυξη εφαρμογών.

3 ΣΧΕΔΙΑΣΜΟΣ ΣΥΣΤΗΜΑΤΟΣ

Όπως έχει αναφερθεί, στο πλαίσιο της παρούσας πτυχιακής εργασίας δημιουργείται μία υπηρεσία για ταίριασμα παικτών, η οποία μπορεί να αφορά ένα παιχνίδι ή οποιαδήποτε άλλη συνεργατική δραστηριότητα. Πριν δημιουργηθεί η υπηρεσία, στο στάδιο σχεδιασμού έχουν οριστεί τα μοντέλα, δηλ. οι προδιαγραφές κλήσης της υπηρεσίας, οι τύποι δεδομένων που δίδονται ως παράμετροι και επιστρέφονται ως αποτελέσματα, καθώς και τα είδη των απαντήσεων που δίδονται: σύμφωνα με το μοντέλο REST, οι τύπου των απαντήσεων αντιστοιχούν στους κωδικούς του πρωτοκόλλου HTTP, π.χ. 200 για επιτυχή εκτέλεση, 422 (Unprocessable Entity) για μη παραδεκτές παραμέτρους κ.λπ. Η μοντελοποίηση αυτή καταγράφηκε σε YAML στο εργαλείο Swagger, και το εργαλείο, μέσω του γεννήτορα κώδικα, δημιούργησε κλάσεις Java για τις παραμέτρους και τα αποτελέσματα, καθώς και τους απαραίτητους ελεγκτές (controllers) για υποδοχή των αιτημάτων, επεξεργασία και επαλήθευση των ορισμάτων, καθώς και διαμόρφωση και επιστροφή των αποτελεσμάτων. Ο προγραμματιστής σε αυτό το σημείο έχει να υλοποιήσει την επιχειρησιακή λογική υλοποίησης των λειτουργιών.

Από την αυτόματη δημιουργία κώδικα και αρχείων του Swagger δημιουργούνται τέσσερα πακέτα:

1. io.swagger
2. io.swagger.api
3. io.swagger.configuration
4. io.swagger.model

Τα πακέτα στα οποία γίνονται παρεμβάσεις από τον προγραμματιστή είναι το io.swagger.api και το io.swagger.model. Αρχικά στο io.swagger.api δημιουργείται η κλάση MatchmakingAlgorithmImplementation στην οποία γίνονται όλοι οι υπολογισμοί του server που μας ενδιαφέρουν. Επιπλέον, στο πακέτο io.swagger.model δημιουργήθηκε η βοηθητική κλάση UtilityUser, με χρήση της οποίας αποθηκεύονται προσωρινά τα ζευγάρια χρηστών με τα αντίστοιχα βάρη τους.

3.1 Αιτήματα

Ο server λαμβάνει μέσω του API, από τον client τα δεδομένα σε συγκεκριμένη μορφή. Λαμβάνει ένα έγγραφο JSON το οποίο αποτελείται από τρεις πίνακες:

- userGlobalScores[ ]
- userPairwiseScore[ ]
- userCollaborationIntentions[ ]

Ο πρώτος πίνακας αποτελείται από αντικείμενα, όπου καθένα από αυτά είναι η συνολική βαθμολογία κάθε χρήστη. Η προδιαγραφή του αντικειμένου-στοιχείου του πίνακα σε Swagger είναι όπως φαίνεται στο σχήμα που ακολουθεί:

```
UserScore:
  type: object
  properties:
    userId:
      type: string
    score:
      $ref: '#/components/schemas/Score'
  example:
    - {"userId": "user1", "score": {"quality": 3.8, "collaboration": 4.0}}
      - {"userId": "user2", "score": {"quality": 4.9, "collaboration": 2.0}}
```

Εικόνα 4: Προδιαγραφή αντικειμένου-στοιχείου του πίνακα userGlobalScores

ενώ η προδιαγραφή του τύπου στοιχείου `#/components/schemas/Score` είναι:

```
Score:
  type: object
  properties:
    quality:
      type: number
      format: float
    colaboration:
      type: number
        format: float
```

Εικόνα 5: Προδιαγραφή του τύπου στοιχείου #/components/schemas/Score

Ο δεύτερος πίνακας userPairwiseScore αποτελείται από αντικείμενα από τα οποία το κάθε ένα περιέχει το όνομα ενός χρήστη και έναν ακόμα πίνακα με τις βαθμολογίες που έχει αποδώσει σε έναν άλλους χρήστες. Η προδιαγραφή του αντικειμένου-στοιχείου του πίνακα userPairwiseScore σε Swagger είναι όπως φαίνεται στο σχήμα που ακολουθεί:

```
    UserPairwiseScore:
      type: object
      properties:
        gradingUser:
          type: string
        scoresGiven:
          type: array
          items:
            $ref: '#/components/schemas/UserScore'
      example:
        - {"gradingUser": "user1", "scoresGiven": [{"userId": "user2", "score":
{"quality": 3.8, "collaboration": 4}}]}
        - {"gradingUser": "user2", "scoresGiven": [{"userId": "user1", "score":
{"quality": 4.9, "collaboration": 2.0}}]}
```

Εικόνα 6: Προδιαγραφή αντικειμένου-στοιχείου του πίνακα userPairwiseScore

Ομοίως, ο τρίτος πίνακας userCollaborationIntentions αποτελείται από αντικείμενα από τα οποία το κάθε ένα περιέχει το όνομα ενός χρήστη και έναν ακόμα πίνακα με την προτίμηση που έχει για έναν άλλο χρήστη. Η προδιαγραφή του αντικειμένου-στοιχείου του πίνακα userCollaborationIntentions σε Swagger είναι όπως φαίνεται στο σχήμα που ακολουθεί:

```
      type: object
      properties:
        gradingUser:
          type: string
        intentions:
          type: array
          items:
            $ref: '#/components/schemas/UserCollaborationSpec'
      example:
        - {"gradingUser": "user1", "intentions": [{"userId": "user2", "intention":
"idc"}]}
        - {"gradingUser": "user2", "intentions": [{"userId": "user1", "intention":
"want"}]}
```

Εικόνα 7: Προδιαγραφή αντικειμένου-στοιχείου του πίνακα userCollaborationIntentions

Όπου ο τύπος δεδομένων `#/components/schemas/UserCollaborationSpec` ορίζεται ως εξής:

```
    UserCollaborationSpec:
      type: object
      properties:
        userId:
          type: string
        intention:
          type: string
          enum:
            - want
            - dwant
```

```
            - idc
      example:
        - {"userId": "user1", "intention": "want"} #  the expressing user states
that s/he wants to collaborate with user1
        - {"userId": "user2", "intention": "idc"} # the expressing user states
that s/he neither wants, nor opposes to collaborate with user2
```

Εικόνα 8: Προδιαγραφή τύπου δεδομένων `#/components/schemas/UserCollaborationSpec`

Κάθε αντικείμενο του δεύτερου και του τρίτου πίνακα αποτελούν έναν από τους n*(n-1) [6] δυνατούς συνδυασμούς παιχτών. Ο λόγος που δεν είναι n*n είναι διότι δεν λαμβάνουμε υπόψιν τους συνδυασμούς όπου κάθε παίχτης αντιστοιχίζεται με τον εαυτό του. Δεν είναι αναγκαίο ότι θα υπάρχουν όλοι οι συνδυασμοί σε κάθε αίτημα που λαμβάνει ο server καθώς δεν είναι απαραίτητο ότι όλοι οι παίχτες θα έχουν παίξει μεταξύ τους. Είναι αναγκαίο όμως για κάθε αντικείμενο στους δύο αυτούς πίνακες, να υπάρχει και το αντίστροφο του (π.χ. αν ο *χρήστης1* βαθμολογεί και αξιολογεί τον *χρήστης2,* τότε έχουν συνεργαστεί και άρα θα πρέπει να υπάρχει στους πίνακες και ότι ο *χρήστης2* βαθμολογεί και αξιολογεί τον *χρηστης1*). Ακολουθεί ενδεικτικό παράδειγμα αιτήματος για την πιο απλή περίπτωση, την περίπτωση δύο χρηστών:

[6] Όπου n είναι ο αριθμός παιχτών.

```
{
  "userGlobalScores": [
    {
      "userId": "test1",
      "score": {
        "quality": 3.8,
        "collaboration": 4.1
      }
    },
    {
      "userId": "test2",
      "score": {
        "quality": 4.9,
        "collaboration": 2.1
      }
    }
  ],
  "userPairwiseScore": [
    {
      "gradingUser": "test1",
      "scoresGiven": [
        {
          "userId": "test2",
          "score": {
            "quality": 3.8,
            "collaboration": 2.1
          }
        }
      ]
    },
    {
      "gradingUser": "test2",
      "scoresGiven": [
        {
          "userId": "test1",
          "score": {
            "quality": 2.9,
            "collaboration": 2.1
          }
        }
      ]
    }
  ],
  "userCollaborationIntentions": [
    {
      "gradingUser": "test1",
      "intentions": [
        {
          "userId": "test2",
          "intention": "dwant"
        }
      ]
    },
    {
      "gradingUser": "test2",
      "intentions": [
        {
          "userId": "test1",
          "intention": "dwant"
        }
      ]
    }
  ]
}
```


Εικόνα 9: Αίτημα που περιλαμβάνει δύο χρήστες

Κατόπιν της διαμόρφωσης των αναθέσεων ζευγαριών, ο εξυπηρετητής επιστρέφει ένα σύνολο από ζεύγη που αποτελούν τις αναθέσεις. Η προδιαγραφή του στοιχείου αυτού του πίνακα σε Swagger φαίνεται στο ακόλουθο σχήμα:

```
    UserPairAssignment:
      title: User pair assignments
      type: object
      properties:
        user1:
          type: string
        user2:
          type: string
      example:
           - {"user1": "user1", "user2": "user2"}
```

Εικόνα 10: Προδιαγραφή του στοιχείου UserPairAssignment

Έχοντας παραθέσει τους τύπους δεδομένων που χρησιμοποιούνται στην αίτηση και την απάντηση, μπορούμε πλέον να παραθέσουμε την πλήρη προδιαγραφή της διαδικασίας διαδικτύου, όπως μοντελοποιήθηκε στο εργαλείο Swagger:

Έχοντας παραθέσει τους τύπους δεδομένων που χρησιμοποιούνται στην αίτηση και την απάντηση, μπορούμε πλέον να παραθέσουμε την πλήρη προδιαγραφή της διαδικασίας διαδικτύου, όπως μοντελοποιήθηκε στο εργαλείο Swagger:

```
paths:
  /matchmaking:
    post:
      tags:
      - "matchmaking"

      summary: "Compute the pairs for the next game round"
      description: ""
      requestBody:
        description: The body is a JSON structure having the following parts (a)
global user score (b) pairwise user scores and (c) user-to-user collaboration
intentions. The output of the computation is a user pair assignment matrix.
        required: true
        content:
          application/json:
            schema:
              type: object
              properties:
                userGlobalScores:
                  type: array
                  items:
                    $ref: '#/components/schemas/UserScore'
                userPairwiseScore:
```

```
            type: array
            items:
              $ref: '#/components/schemas/UserPairwiseScore'
          userCollaborationIntentions:
            type: array
            items:
              $ref: '#/components/schemas/UserCollaborationIntentions'

responses:
  '200':
    description: Successful response
    content:
      application/json:
        schema:
          type: array
          items:
            $ref: '#/components/schemas/UserPairAssignment'
  405:
      description: "Invalid input"
```

Εικόνα 11: Πλήρης προδιαγραφή της διαδικασίας διαδικτύου «matchmaking»

3.2 Κλάσεις

Το εργαλείο Swagger με τη χρήση του γεννήτορα κώδικα δημιούργησε όλο το πλαίσιο παραλαβής αιτήσεων, ελέγχου των ορισμάτων και επιστροφής απαντήσεων. Ο προγραμματιστής πλέον πρέπει να υλοποιήσει τις διαδικασίες που επεξεργάζονται τα ορίσματα των αιτήσεων και διαμορφώνουν τις απαντήσεις. Δεδομένου ότι χρησιμοποιούμε τη γλώσσα Java, ο κώδικας αυτός διαρθρώνεται σε κλάσεις. Στη συνέχεια περιγράφονται οι κύριες κλάσεις που δημιουργήθηκαν.

Η πρώτη κλάση που υλοποιήθηκε μετά τη δημιουργία κώδικα από το Swagger είναι η UtilityUser, η οποία αποτελείται από έξι μεθόδους, που θα αναλυθούν παρακάτω. Η κλάση αυτή δημιουργήθηκε με σκοπό να δημιουργούνται αντικείμενα και ArrayList αυτού του τύπου έτσι ώστε να αποθηκεύονται προσωρινά τα δεδομένα του προβλήματος.

Η δεύτερη και κυριότερη κλάση που υλοποιήθηκε μετά την δημιουργία κώδικα από το Swagger είναι η MatchmakingAlgorithmImplementation. Σε αυτή την κλάση γίνονται όλοι οι υπολογισμοί για τον διαχωρισμό τον παικτών, τον υπολογισμό των βαρών, τις προτιμήσεις για τα υποψήφια ζευγάρια καθώς και το πρόβλημα γραμμικού προγραμματισμού.

3.3 Μέθοδοι

Στην υποενότητα αυτή θα αναλυθούν οι μέθοδοι που υλοποιήθηκαν καθώς και ενδιάμεσα βήματα μεταξύ των κλήσεων των συναρτήσεων αυτών. Ακολουθούν με σειρά εμφάνισης στον κώδικα οι μέθοδοι που καλούνται πρώτα στην κλάση UtilityUser και κατόπιν στην MatchmakingAlgorithmImplementation:

Οι μέθοδοι που έχουν γραφτεί στην κλάση UtilityUser είναι set & get για τις μεταβλητές weight, user_i, user_j. Οι συναρτήσεις set & get είναι για να υπάρχει πρόσβαση στις private μεταβλητές έξω από την κλάση. Η set χρησιμοποιείται για να τεθεί τιμή στις μεταβλητές και η get επιστρέφει την τιμή της μεταβλητής. Λόγω της τετριμμένης λειτουργίας αυτών των συναρτήσεων δεν γίνεται παρακάτω λεπτομερής αναφορά για αυτές:

- setWeight()
- getWeight()
- setUser_i()
- getUser_j()
- setUser_j()
- getUser_j()

Οι συναρτήσεις που έχουν γραφτεί στην κλάση MatchmakingAlgorithmImplementation είναι οι ακόλουθες:

3.3.1 final_pair()

Η συνάρτηση final_pair() λαμβάνει ως ορίσματα τους τρεις πίνακες που περιλαμβάνονται στην αίτηση, υπό την μορφή λίστας (List). Σε αυτή την συνάρτηση καλούνται όλες οι συναρτήσεις που έχουν υλοποιηθεί στην κλάση και αναλύονται παρακάτω.

Αρχικά δημιουργείται ένα λεξικό που περιέχει όλους τους δυνατούς συνδυασμούς n*n[7] για τους παίχτες. Έπειτα διατρέχεται η λίστα UserPairwiseScore με δύο εμφωλευμένες επαναλήψεις κατά τις οποίες καλούνται οι συναρτήσεις `get_intentions()` &

[7] n ο αριθμός των παιχτών

`weight()`. Έπειτα καλείται η συνάρτηση `utility_per_user_calculator()`, μια αλφαβητική ταξινόμηση με βάση και τους δύο παίχτες, η `global_utilityFunc2()` καθώς και μία ακόμα αλφαβητική ταξινόμηση με βάση και τους δύο παίχτες.

Στη συνέχεια καλείται η συνάρτηση `maximize_lp()` που δημιουργεί και επιλύει το πρόβλημα μεγιστοποίησης και επιστρέφει τα τελικά ζευγάρια χρηστών ως ArrayList τύπου `UserPairAssignment` στην συνάρτηση `final_pair()`.

Τέλος η συνάρτηση επιστρέφει αυτό το ArrayList τύπου `UserPairAssignment` στην συνάρτηση `matchmakingPost()` της κλάσης `MatchmakingApiController`.

3.3.2 weight()

Η συνάρτηση weight() λαμβάνει ως ορίσματα δύο αντικείμενα `UserPairwiseScore` για τον πρώτο και δεύτερο χρήστη αντίστοιχα, την πρόθεση από την get_intentions() σχετικά με την πρόθεση τους να παίξουν ξανά μαζί καθώς και την συνολική βαθμολογία του χρήστη "quality" και "collaboration" που λαμβάνει από τον πρώτο πίνακα της αίτησης. Στη συνέχεια με βάση την ακόλουθη συνάρτηση υπολογίζεται το βάρος για το κάθε ζευγάρι:

$$w_{i,j} = \begin{cases} 1 + \dfrac{avg(collaborationStars_{i,j}) + avg(qualityStars_{i,j})}{10} & \text{αν } o\ i \text{ έχει συνεργαστεί με τον } j \\ & \text{και } o\ i \text{ θέλει να ξανασυνεργαστεί} \\ & \text{με τον } j\ (intention[i,j] = want) \\ -2 + \dfrac{avg(collaborationStars_{i,j}) + avg(qualityStars_{i,j})}{10} & \text{αν } o\ i \text{ έχει συνεργαστεί με τον } j \\ & \text{και } o\ i \text{ δεν θέλει να ξανασυνεργαστεί} \\ & \text{με τον } j\ (intention[i,j] = doNotWant) \\ \dfrac{avg(collaborationStars_{i,j}) + avg(qualityStars_{i,j})}{10} - 0.5 & \text{αν } o\ i \text{ έχει συνεργαστεί με τον } j \text{ και είναι} \\ & \text{αδιάφορος ως προς το αν θέλει} \\ & \text{να ξανασυνεργαστεί } (intention[i,j] = idc) \\ 1 + \dfrac{userGlobalScores_j.quality + userGlobalScores_j.collaboration}{10} & \text{αν οι } i,j \text{ δεν έχουν συνεργαστεί} \\ & \text{ξανά και } o\ i \text{ θέλει να συνεργαστεί} \\ & \text{με τον } j\ (intention[i,j] = want) \\ -2 + \dfrac{userGlobalScores_j.quality + userGlobalScores_j.collaboration}{10} & \text{αν οι } i,j \text{ δεν έχουν συνεργαστεί} \\ & \text{ξανά και } o\ i \text{ δεν θέλει να συνεργαστεί} \\ & \text{με τον } j\ (intention[i,j] = doNotWant) \\ \dfrac{userGlobalScores_j.quality + userGlobalScores_j.collaboration}{10} - 0.5 & \text{αν οι } i,j \text{ δεν έχουν συνεργαστεί} \\ & \text{ξανά και } o\ i \text{ είναι αδιάφορος} \\ & \text{ως προς το αν θέλει να} \\ & \text{ξανασυνεργαστεί } (intention[i,j] = idc) \end{cases}$$

Εικόνα 12: Συνάρτηση υπολογισμού βαρών

Το σκεπτικό διαμόρφωσης των βαρών έχει ως ακολούθως:

- Αν ο χρήστης i επιθυμεί να συνεργαστεί με έναν άλλο χρήστη j, τότε το βάρος παίρνει τιμές στο διάστημα [1, 2]. Η μία μονάδα («βασικό βάρος») προκύπτει από την πρόθεση συνεργασίας, ενώ στο βασικό βάρος προστίθεται και μία ποσότητα που προκύπτει ως εξής:
    - Αν οι χρήστες i και j έχουν συνεργαστεί ξανά, από τα σκορ που ο χρήστης *i* έχει δώσει στον χρήστη *j* ως μέσος όρος όλων των βαθμολογιών, κανονικοποιημένος στο διάστημα [0, 1]
    - Αν οι χρήστες i και j έχουν συνεργαστεί ξανά, από τη συνολική βαθμολογία που έχει λάβει ο χρήστης *j* από άλλους χρήστες, κανονικοποιημένη στο διάστημα [0, 1]
- Αν ο χρήστης i είναι αρνητικός ως προς το ενδεχόμενο να συνεργαστεί με έναν άλλο χρήστη j, τότε το βάρος παίρνει τιμές στο διάστημα [-2, -1]. Οι -2 μονάδες («βασικό βάρος») προκύπτουν από την αρνητική πρόθεση συνεργασίας, ενώ στο βασικό βάρος προστίθεται και μία ποσότητα που προκύπτει όπως περιγράφεται και για την πρώτη περίπτωση.
- Αν ο χρήστης i είναι αδιάφορος ως προς το ενδεχόμενο να συνεργαστεί με έναν άλλο χρήστη j, τότε το βάρος παίρνει τιμές στο διάστημα [-0.5, -0.5]. Η -0.5 μονάδα («βασικό βάρος») προκύπτει από την αδιάφορη στάση συνεργασίας, ενώ στο βασικό βάρος προστίθεται και μία ποσότητα που προκύπτει όπως περιγράφεται και για την πρώτη περίπτωση.

Η συνάρτηση επιστρέφει το βάρος. π.χ. από την Εικόνα 8: Θα επιστρέψει “-1.410 ” στην πρώτη κλήση της, το βάρος για το ζευγάρι χρηστών “test1” ,“test2”.

Η συνάρτηση αυτή καλείται για κάθε δυνατό ζευγάρι χρηστών.

3.3.3 utility_per_user_calculator()

Η συνάρτηση `utility_per_user_calculator()` λαμβάνει ως όρισμα ένα ArrayList τύπου `UtilityUser`. Σκοπός αυτής της συνάρτησης είναι να ξεχωρίσει τις διπλότυπες καταχωρίσεις στο ArrayList. Αυτό επιτυγχάνεται με την χρήση δύο εμφωλευμένων επαναλήψεων κατά τις οποίες διατρέχεται δύο φορές το ArrayList και όταν ό χρήστης της πρώτης επανάληψης είναι ίδιος με τον χρήστη στην δεύτερη επανάληψη (π.χ.

`uu.getUser_i().equals(uu_j.getUser_j())` όπου `uu` & `uu_j` είναι μεταβλητές `UtilityUser`).

Η συνάρτηση επιστρέφει ένα νέο ArrayList τύπου `UtilityUser` και καλείται μία φορά μετά από όλες τις κλίσεις της `weight()`.

3.3.4 global_utilityFunc2()

Η συνάρτηση `global_utilityFunc2()` λαμβάνει ως όρισμα το ArrayList που επιστρέφει η `utility_per_user_calculator()`. Με τον ίδιο τρόπο όπως και η `utility_per_user_calculator()` κάνει τον ίδιο διαχωρισμό προσθέτοντας μία επιπλέον συνθήκη όπου σε κάθε επανάληψη όπου αν το βάρος του ζευγαριού είναι 0, τότε μια μεταβλητή x γίνεται 0. Όταν ισχύει η ίδια συνθήκη και για την δεύτερη επανάληψη, τότε το ζευγάρι εισάγεται σε ένα νέο ArrayList τύπου `UtilityUser`.

Η συνάρτηση επιστρέφει το τελικό ArrayList τύπου `UtilityUser` που θα εισαχθεί στην συνάρτηση μεγιστοποίησης και καλείται μία φορά μετά την `utility_per_user_calculator()` και μία ταξινόμηση.

3.3.5 maximize_lp()

Η συνάρτηση `maximize_lp()` είναι η συνάρτηση στην οποία ορίζεται το πρόβλημα μεγιστοποίησης, καλείται η συνάρτηση που δημιουργεί τους περιορισμούς του προβλήματος και επιλύεται το πρόβλημα που έχει δημιουργηθεί. Λαμβάνει ως όρισμα το ArrayList που επιστρέφεται από την `global_utilityFunc2()` αφού πρώτα έχει υποστεί μία ταξινόμηση ακόμα.

Η δημιουργία του προβλήματος βελτιστοποίησης ακολουθεί τα εξής βήματα:

1. Για κάθε ζεύγος χρηστών (i,j) δημιουργείται η μεταβλητή $x_{i,j}$ η οποία μπορεί να λάβει τιμές 0 ή 1. Η τιμή 1 σημαίνει ότι ο χρήστης *i* τοποθετείται στο ίδιο ζεύγος με τον χρήστη *j*, ενώ η τιμή 0 το αντίθετο.
2. Δεδομένου ότι κάθε χρήστης μπορεί να συμμετέχει μόνο σε ένα ζεύγος, εισάγεται ο περιορισμός

$$\sum_j x_{i,j} = 1$$

για κάθε *i*.

3. Δεδομένου ότι κάθε χρήστης *πρέπει* να συμμετέχει σε ένα ζεύγος, εισάγεται ο περιορισμός

$$\sum_i x_{i,j} = 1$$

για κάθε *j*. Προφανώς αν το πλήθος των παικτών είναι περιττό, ένας χρήστης δεν θα μπορέσει να μετέχει σε ζεύγος και θα θεωρηθεί ζεύγος με τον εαυτό του.

4. Για κάθε χρήστη, η μερική αντικειμενική συνάρτηση διαμορφώνεται ως

$$partialObjFunction_i = \sum_j x_{i,j} * weight_{i,j}$$

5. Η ολική αντικειμενική συνάρτηση διαμορφώνεται ως

$$globalObjFunction = \sum_i partialObjFunction_i$$

6. Ως στόχος του προβλήματος ορίζεται η μεγιστοποίηση της ολικής αντικειμενικής συνάρτησης.

Σε επίπεδο κώδικα, δημιουργείται ένας μονοδιάστατος πίνακας που θα περιέχει την αντικειμενική συνάρτηση, με συνολικό μέγεθος n*n[8]. Διατρέχεται το ArrayList και όταν βρεθεί ζευγάρι χρηστών (δηλαδή στους παίχτες που υπάρχουν μέσα στο ArrayList ως δυνατό ζευγάρι) εισάγει την τιμή του βάρους στην αντικειμενική συνάρτηση, σε αντίθετη περίπτωση εισάγεται ως τιμή το μηδέν.

Μόλις ολοκληρωθεί η διαδικασία απόδοσης τιμών στην αντικειμενική συνάρτηση, δημιουργείται το πρόβλημα γραμμικού προγραμματισμού και τίθεται ως πρόβλημα

[8] n ο αριθμός των παικτών

μεγιστοποίησης. Έπειτα ορίζονται όλες οι θέσεις του πίνακα της αντικειμενικής συνάρτησης ως δυαδικές, επιτρέποντας έτσι στους περιορισμούς τιμές μηδέν και ένα. Μόλις ολοκληρωθούν αυτές οι διαδικασίες, καλείται η συνάρτηση `rowConst()` που θέτει όλους τους περιορισμούς για το πρόβλημα της μεγιστοποίησης.

Μετά υπολογίζεται το πρόβλημα μεγιστοποίησης, το πρόβλημα εμφανίζεται σε μορφή CPLEX[9] και εμφανίζονται τα ζευγάρια που θα δημιουργηθούν σε μορφή δισδιάστατου πίνακα:

```
GLPK Simplex Optimizer, v4.65
6 rows, 4 columns, 10 non-zeros
      0: obj = -0.000000000e+000 inf =  2.000e+000 (2)
      3: obj = -2.910000086e+000 inf =  0.000e+000 (0)
*     4: obj = -2.910000086e+000 inf =  0.000e+000 (0)
OPTIMAL LP SOLUTION FOUND
GLPK Integer Optimizer, v4.65
6 rows, 4 columns, 10 non-zeros
4 integer variables, all of which are binary
Integer optimization begins...
Long-step dual simplex will be used
+     4: mip =     not found yet <=              +inf        (1; 0)
+     4: >>>>> -2.910000086e+000 <= -2.910000086e+000   0.0% (1; 0)
+     4: mip = -2.910000086e+000 <=     tree is empty   0.0% (0; 1)
INTEGER OPTIMAL SOLUTION FOUND
\ This file is autogenerated by the SCPSolver framework
\ Modify at your own risk

MAXIMIZE
 obj: 0 x0 - x1 - x2 + 0 x3
SUBJECT TO
 r0: x0 + x1 = 1.0
 r1: x2 + x3 = 1.0
 vIc0: x0 = 0.0
 vIc1: x1 -1.0 x2 = 0.0
 vIc2: -1.0 x1 + x2 = 0.0
 vIc3: x3 = 0.0
GENERAL
 x0
 x1
 x2
 x3

END
```

Εικόνα 13: Το πρόβλημα σε μορφή CPLEX

```
x1,1=0 x2,1=1
x1,2=1 x2,2=0
```

Εικόνα 14: Τελικός πίνακας για 2 παίχτες

[9] Η μορφή CPLEX είναι μία τυπική μορφή αναπαράστασης ενός προβλήματος γραμμικού προγραμματισμού με υψηλή αναγνωσιμότητα

Τέλος λαμβάνονται τα τελικά ζευγάρια και η συνάρτηση επιστρέφει ένα ArrayList τύπου `UserPairAssignment`.

3.3.6 rowConst()

Η συνάρτηση δέχεται ως ορίσματα το πλήθος των παιχτών, και το πρόβλημα γραμμικού προγραμματισμού. Αρχικά δημιουργούνται δύο δισδιάστατοι πίνακες, ένας για τους περιορισμούς στις γραμμές του προβλήματος και ένας για τις στήλες.

Για τις γραμμές εισάγουμε τον περιορισμό ότι το άθροισμα των υποψήφιων ζευγαριών πρέπει να είναι ένα. Έπειτα γίνεται έλεγχος για το αν το πλήθος των παιχτών είναι άρτιος ή περιττός. Αν είναι ζυγός, το άθροισμα των υποψήφιων ζευγαριών για κάθε στήλη πρέπει να είναι ίσο με ένα και το άθροισμα της διαγώνιου ίσο με μηδέν. Αν είναι περιττός, το άθροισμα των υποψήφιων ζευγαριών για κάθε στήλη πρέπει να είναι ίσο με ένα και το άθροισμα της διαγώνιου ίσο με ένα. Αυτό γίνεται διότι αν υπάρχουν τρεις παίχτες, υποχρεωτικά κάποιος παίχτης δεν θα μπορέσει να αντιστοιχηθεί με έναν άλλο:

```
1   x1,1=0 x2,1=0 x3,1=1
2   x1,2=0 x2,2=1 x3,2=0
3   x1,3=1 x2,3=0 x3,3=0
```

Εικόνα 15: Παράδειγμα 3 παιχτών, όπου ο παίκτης 2 δεν έχει αντιστοιχηθεί με άλλον

Η συνάρτηση δεν επιστρέφει τίποτα.

3.3.7 printConstraints()

Η συνάρτηση δέχεται ως όρισμα έναν δυσδιάστατο πίνακα και εκτυπώνει κάθε γραμμή και στήλη του. Η συνάρτηση δεν επιστρέφει κανένα αποτέλεσμα.

3.4 Απαντήσεις

Ο server αφού πραγματοποιήσει τους υπολογισμούς για το έγγραφο JSON της αίτησης, όπως αυτό περιγράφεται στο 3.1, επιστρέφει μέσω ενός Post request τα αποτελέσματα.

Τα αποτελέσματα είναι ένα JSON request που αποτελείται από έναν πίνακα:

- UserPairAssignment[ ]

Ο πίνακας αυτός αποτελείται από αντικείμενα όπου το κάθε ένα από αυτά αντιπροσωπεύει ένα ζευγάρι παιχτών. Ανάλογα το πλήθος των παιχτών θα υπάρχουν ένα ή περισσότερα αντικείμενα στον πίνακα αυτόν:

Εικόνα 16: Απάντηση του server για δύο παίχτες

Εικόνα 17: Απάντηση του εξυπηρετητή για πέντε παίχτες

3.5 Πρόγραμμα δημιουργίας δοκιμαστικών δεδομένων

Το πρόγραμμα δημιουργίας δοκιμαστικών δεδομένων αποτελείται από:

- Τρεις κλάσεις αντίστοιχες με τα αντικείμενα που υπάρχουν στο JSON που λαμβάνει ο server:

```
class UserX:
    def __init__(self, user_i, quality,colaboration):
        self.user_i = user_i
        self.quality = quality
        self.colaboration = colaboration

class UserPairScore:
    def __init__(self, user_i,user_j, quality,colaboration):
        self.user_i = user_i
        self.user_j = user_j
        self.quality = quality
        self.colaboration = colaboration
class UserPairInt:
    def __init__(self, user_i,user_j,intention):
        self.user_i = user_i
        self.user_j = user_j
        self.intention = intention
```

Εικόνα 18: Κλάσεις για την δημιουργία του αρχείου

- Συναρτήσεις που δημιουργούν τα αντικείμενα σε μορφή JSON:

```
################userGlobalScores#####################################
def gen_user(user_i,quality,colaboration):
    return {"userId":user_i, "score":{"quality":quality, "collaboration":colaboration}}
#####################################################################

################userPairwiseScore####################################
def gen_grading_user(user_i,user_j,quality,colaboration):
    return {"gradingUser":user_i,"scoresGiven":[{"userId":user_j,"score":{"quality":quality, "collaboration":colaboration}}]}
#####################################################################
################userCollaborationIntentions##########################
def gen_grading_user_int(user_i,user_j,intentions):
    return {"gradingUser":user_i,"intentions":[{"userId":user_j,"intention":intentions}]}
#####################################################################
```

Εικόνα 19: Συναρτήσεις δημιουργίας παιχτών, βαθμολογιών και αξιολογήσεων.

Στην αρχή ανάλογα με το πλήθος των χρηστών που έχει εισαχθεί από την γραμμή εντολών δημιουργούνται τόσοι χρήστες, με αντίστοιχα τυχαίες βαθμολογίες της κλάσης UserX. Στη συνέχεια δημιουργούνται όλες οι δυνατές μεταθέσεις ανά δύο και εισάγονται με τυχαία σειρά σε μία λίστα.

Στην συνέχεια ένα προς ένα αφαιρούνται από την λίστα και δημιουργούνται και αποθηκεύονται σε λίστα αντικείμενα της κλάσης UserPairScore. Το ίδιο γίνεται και για το UserPairInt.

Τέλος καλούνται οι συναρτήσεις gen_user(),gen_grading_user(),gen_grading_user_int() με ορίσματα τα αντικείμενα που έχουν οριστεί παραπάνω, εισάγονται ως αντικείμενα JSON και αποθηκεύονται στο αρχείο.

```
"""
Create the json object
"""
userdata = {"userGlobalScores": [gen_user(i.user_i,i.quality,i.colaboration) for i in user_list],
            "userPairwiseScore": [gen_grading_user(i.user_i,i.user_j,i.quality,i.colaboration) for i in pair_score_list],
            "userCollaborationIntentions": [gen_grading_user_int(i.user_i,i.user_j,i.intention) for i in pair_int_list]
            }
```

Εικόνα 20: Τελικό αντικείμενο JSON

4 ΑΞΙΟΛΟΓΗΣΗ

Στο κεφάλαιο αυτό θα παρουσιαστούν και θα αναλυθούν ενδεικτικά παραδείγματα εκτέλεσης του αλγορίθμου (για διαφορετικούς αριθμούς παιχτών) με σκοπό τον υπολογισμό της καταλληλόλητας της λύσης καθώς κα του χρόνου εκτέλεσης του προγράμματος για κάθε περίπτωση.

Για τον ακριβέστερο υπολογισμό του χρόνου εκτέλεσης του αλγορίθμου, έχουν τοποθετηθεί δύο ξεχωριστοί μετρητές:

- Ο πρώτος μετρητής τοποθετήθηκε από την αρχή του προγράμματος μέχρι και μετά τους υπολογισμούς των βαρών για τον κάθε παίχτη, τους υπολογισμούς των βαρών για τα υποψήφια ζευγάρια καθώς και τις δύο αλφαβητικές ταξινομήσεις των λιστών με τα υποψήφια ζευγάρια.
- Ο δεύτερος μετρητής τοποθετήθηκε πριν την κλήση της συνάρτησης μεγιστοποίησης , την συνάρτηση υλοποίησης των περιορισμών και τον υπολογισμό του προβλήματος μεγιστοποίησης μέχρι και την επιστροφή των τελικών ζευγαριών παιχτών και την λήξη του προγράμματος.

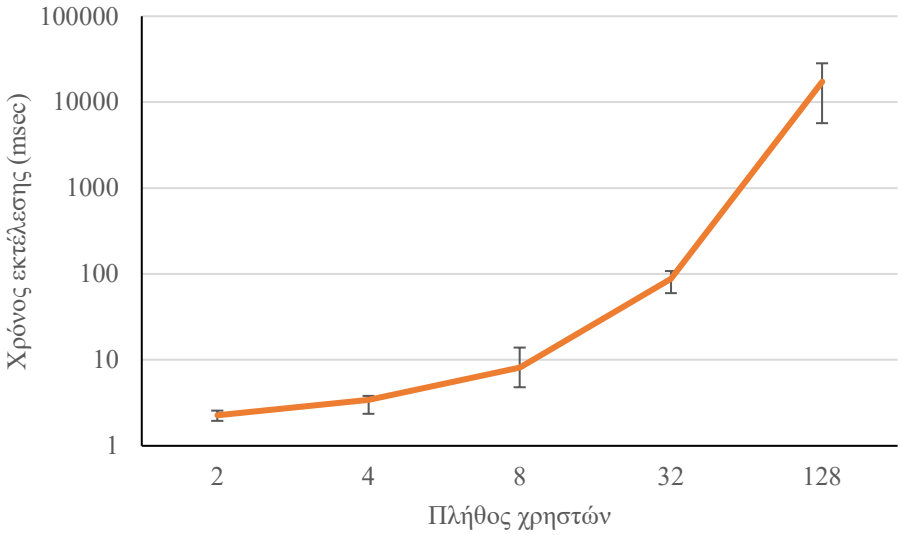
Παρακάτω αναλύονται οι ακόλουθες περιπτώσεις αριθμού χρηστών:

- 2 παίχτες
- 4 παίχτες
- 8 παίχτες
- 32 παίχτες
- 128 παίχτες
- 512 παίχτες

Σε κάθε περίπτωση παιχτών πραγματοποιήθηκαν 8 δοκιμές, με διαφορετικές βαθμολογίες, προτιμήσεις και ζευγάρια που έχουν παίξει μεταξύ τους, προκειμένου να υπολογιστούν οι μέσοι όροι των χρόνων εκτέλεσης.

Στην Εικόνα 21 απεικονίζεται ο χρόνος εκτέλεσης του αλγόριθμου για διαφορετικά πλήθη χρηστών (σημειώστε ότι ο κατακόρυφος άξονας είναι λογαριθμικός). Παρατηρούμε ότι για

δύο χρήστες ο χρόνος κυμαίνεται από 2 χιλιοστά του δευτερολέπτου έως 2.5 χιλιοστά, ενώ για 128 παίχτες ο συνολικός χρόνος εκτέλεσης κυμαίνεται από 5,7 δευτερόλεπτα έως 28,3 δευτερόλεπτα. Εφαρμόζοντας τεχνικές προσαρμογής καμπύλης[10], προκύπτει ότι ο χρόνος αυξάνεται υπερ-γραμμικά (≈ $O(v^4)$, η ακριβής συνάρτηση που υπολογίζεται είναι $y = 0.0001573911 * x^{3.816834}$).



Εικόνα 21. Χρόνος εκτέλεσης της διαδικασίας δημιουργίας ζευγαριών

Στις δοκιμαστικές αιτήσεις, υπήρχαν περιπτώσεις στις οποίες οι συνδυασμοί των παιχτών που είχαν παίξει μεταξύ τους ήταν διαφορετικές: για παράδειγμα, υπήρχαν δοκιμές στις οποίες κάποια από τα ζευγάρια παιχτών που είχαν παίξει μεταξύ τους ήταν 4 ,6,8,10 και 12 σε κάθε περίπτωση το πλήθος των ζευγαριών ήταν μικρότερο ή ίσο $n*(n-1)$[11]. Αυτό σημαίνει ότι υπήρχαν διαφορετικές βαθμολογίες, προθέσεις καθώς και διαφορετικός αριθμός ζευγαριών για κάθε δοκιμή.

Για 512 χρήστες δεν κατέστη δυνατή η εκτέλεση, λόγω εξάντλησης της μνήμης του υπολογιστή.

[10] https://mycurvefit.com/
[11] n ο αριθμός των παιχτών

Στη συνέχεια με χρήση του εργαλείου δοκιμών Apache Benchmark (ab) δοκιμάζεται η ταχύτητα απόκρισης του server και η αντοχή του σε πολλαπλά και ταυτόχρονα αιτήματα. Όπως και στην προηγούμενη περίπτωση, εστάλησαν 2.000 αιτήματα, 4 ταυτόχρονα σε 2 ενδεικτικές περιπτώσεις:

- Για 8 χρήστες
- Για 128 χρήστες

Για 8 χρήστες, όπως φαίνεται παρακάτω ο συνολικός χρόνος των αιτημάτων είναι 4.6 δευτερόλεπτα. Πιο συγκεκριμένα ο μέσος χρόνος κάθε αιτήματος είναι 9.2 χιλιοστά του δευτερολέπτου, ενώ ο μέσος χρόνος των παράλληλων αιτημάτων είναι 2.3 χιλιοστά του δευτερολέπτου. Παρατηρήθηκε επίσης ότι από τα 2000 αιτήματα που στάλθηκαν, 4 από αυτά απέτυχαν. Ο συνολικός χρόνος εκτέλεσης είναι σημαντικά μειωμένος σε σχέση με την περίπτωση να γίνονται οι αιτήσεις σειριακά, καθώς ο υπολογιστής διαθέτει επαρκείς πόρους για την παράλληλη εκτέλεση και των τεσσάρων αιτημάτων.

```
This is ApacheBench, Version 2.3 <$Revision: 1879490 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.1.6 (be patient)


Server Software:
Server Hostname:        192.168.1.6
Server Port:            8080

Document Path:          /matchmaking/
Document Length:        206 bytes

Concurrency Level:      4
Time taken for tests:   4.631 seconds
Complete requests:      2000
Failed requests:        4
   (Connect: 0, Receive: 0, Length: 4, Exceptions: 0)
Total transferred:      649184 bytes
Total body sent:        50422000
HTML transferred:       411184 bytes
Requests per second:    431.88 [#/sec] (mean)
Time per request:       9.262 [ms] (mean)
Time per request:       2.315 [ms] (mean, across all concurrent requests)
Transfer rate:          136.90 [Kbytes/sec] received
                        10632.97 kb/s sent
                        10769.87 kb/s total

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.4      0       1
Processing:     6    9   2.0      9      33
Waiting:        5    8   1.9      8      31
Total:          6    9   2.1      9      34

Percentage of the requests served within a certain time (ms)
  50%      9
  66%     10
  75%     10
  80%     10
  90%     11
  95%     12
  98%     13
  99%     18
 100%     34 (longest request)
```

Εικόνα 22: Αποτελέσματα benchmark για 8 χρήστες, 2000 αιτήματα, 4 ταυτόχρονα

Για 128 χρήστες, όπως φαίνεται παρακάτω ο συνολικός χρόνος των αιτημάτων είναι 1348 δευτερόλεπτα (22 λεπτά). Πιο συγκεκριμένα ο μέσος χρόνος κάθε αιτήματος είναι 26 δευτερόλεπτα, ενώ ο μέσος χρόνος των παράλληλων αιτημάτων είναι 6 δευτερόλεπτα.

Παρατηρήθηκε επίσης ότι από τα 2000 αιτήματα που στάλθηκαν, 6 από αυτά απέτυχαν. Και εδώ παρατηρείται βελτίωση του συνολικού χρόνου εκτέλεσης σε σχέση με το ενδεχόμενο να έχουμε πλήρως σειριακές εκτελέσεις.

```
This is ApacheBench, Version 2.3 <$Revision: 1879490 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.1.6 (be patient)


Server Software:
Server Hostname:        192.168.1.6
Server Port:            8080

Document Path:          /matchmaking/
Document Length:        3274 bytes

Concurrency Level:      4
Time taken for tests:   1348.673 seconds
Complete requests:      201
Failed requests:        6
   (Connect: 0, Receive: 0, Length: 6, Exceptions: 0)
Non-2xx responses:      6
Total transferred:      663411 bytes
Total body sent:        897292944
HTML transferred:       639492 bytes
Requests per second:    0.15 [#/sec] (mean)
Time per request:       26839.268 [ms] (mean)
Time per request:       6709.817 [ms] (mean, across all concurrent requests)
Transfer rate:          0.48 [Kbytes/sec] received
                        649.72 kb/s sent
                        650.20 kb/s total

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.5      0       1
Processing: 15712 26461 1618.6  26134   31022
Waiting:    15706 26459 1618.7  26132   31020
Total:      15713 26461 1618.5  26134   31022

Percentage of the requests served within a certain time (ms)
  50%  26133
  66%  26691
  75%  27134
  80%  27783
  90%  28470
  95%  29063
  98%  30534
  99%  30787
 100%  31022 (longest request)
```

Εικόνα 23: Αποτελέσματα benchmark για 128 χρήστες, 2000 αιτήματα, 4 ταυτόχρονα

Όλες οι παραπάνω δοκιμές πραγματοποιήθηκαν σε υπολογιστή με τις ακόλουθες προδιαγραφές:

Επεξεργαστής: Intel Core i7-6700[12] , 4 πυρήνες (8 threads) στα 3.4 GHz

Μνήμη: HyperX Savage DDR4[13], 2x8 GB στα 2460 MHz

[12] https://ark.intel.com/content/www/us/en/ark/products/88196/intel-core-i76700-processor-8m-cache-up-to-4-00-ghz.html

[13] https://www.hyperxgaming.com/us/company/press/article/40004

5 ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΕΣ ΕΠΕΚΤΑΣΕΙΣ

Σε αυτήν την πτυχιακή εργασία αναπτύχθηκε ένας αλγόριθμος, σκοπός του οποίου είναι να δημιουργεί βέλτιστα ζευγάρια μεταξύ παιχτών, με. χρήση τεχνικών γραμμικού προγραμματισμού. Παρουσιάσθηκαν αρχικά οι τεχνολογίες και τα εργαλεία που χρησιμοποιήθηκαν για την υλοποίηση, καθώς και οι λόγοι που προτιμήθηκαν αυτές έναντι άλλων τεχνολογιών. Στην συνέχεια αναλύεται η δομή του προγράμματος, οι συναρτήσεις που καλούνται, οι απαντήσεις που επιστρέφει το πρόγραμμα μετά την εκτέλεση του, καθώς και η δομή και η λειτουργία ενός προγράμματος δημιουργίας δοκιμαστικών δεδομένων. Τέλος παρουσιάζεται και αξιολογείται η απόδοση του προγράμματος με βάση δοκιμές επιδόσεων που πραγματοποιήθηκαν. Παρατηρήθηκε ότι ο χρόνος εκτέλεσης καθώς και η χρήση μνήμης αυξάνεται ανάλογα με το πλήθος των παιχτών.

Αν και το πρόγραμμα είναι λειτουργικό, υπάρχουν περιθώρια βελτίωσης και εξέλιξης του. Θα μπορούσε να διερευνηθεί η διατύπωση του προβλήματος με τρόπο ώστε να διευκολύνεται η ταχύτερη επίλυσή του. Επίσης ενώ το πρόγραμμα εκτελείται κανονικά, σε μεγάλους αριθμούς χρηστών π.χ. 500, με τον τρόπο που είναι υλοποιημένο το πρόγραμμα και με τις βιβλιοθήκες που χρησιμοποιεί, οι απαιτήσεις μνήμης είναι πολύ μεγάλες, και το πρόγραμμα δεν μπορεί να εκτελεστεί. Για την επίλυση αυτού του ζητήματος θα μπορούσε να γίνει χρήση άλλων βιβλιοθηκών γραμμικού προγραμματισμού π.χ. CPLEX.

Πέρα από βελτιώσεις σε χρόνους επεξεργασίας, θα μπορούσαν να προστεθούν δύο ακόμα λειτουργίες. Η πρώτη λειτουργία είναι η χρήση ενός ακόμα όρου για τον υπολογισμό των βαρών, η «ικανοποίηση» των παιχτών από τα ζευγάρια που δημιουργήθηκαν σε προηγούμενο γύρο, έτσι ώστε τα επόμενα ζευγάρια που θα προκύψουν να έχουν «μάθει» από προηγούμενους γύρους. Η δεύτερη λειτουργία που θα μπορούσε να προστεθεί αφορά τη δημιουργία ομάδων. Αντί να δημιουργούνται πλέον μόνο ζευγάρια, θα μπορούσε να επεκταθεί η λειτουργία έτσι ώστε να υπολογίζει βέλτιστες ομάδες μεγαλύτερου μεγέθους π.χ. τριάδες, πεντάδες κ.λπ. Πάνω σε αυτό θα μπορούσε επίσης να δημιουργηθεί ένα σύστημα εξισορρόπησης, έτσι ώστε να αποφευχθεί η δημιουργία ομάδων στις οποίες όλοι οι παίχτες έχουν την καλύτερη συνεργασία

και βαθμολογία και να έρχονται αντιμέτωποι με ομάδες των οποίων οι παίκτες έχουν τις χειρότερες δυνατές αξιολογήσεις.

Τέλος, θα ήταν ενδιαφέρον να αξιολογηθεί ο βαθμός ικανοποίησης των χρηστών από τα αποτελέσματα της δημιουργίας ζευγών, τόσο εκ των προτέρων (δηλ. με την ανακοίνωση των αποτελεσμάτων της δημιουργίας ζευγών) όσο και εκ των υστέρων (δηλ. μετά τη συνεργασία).

6 ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Wikipedia contributors, «Linear programming,» 16 September 2021. [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Linear_programming. [Πρόσβαση 19 September 2021].

[2] Wikipedia contributors, «Just-in-time compilation,» 21 August 2021. [Ηλεκτρονικό]. Available: https://en.wikipedia.org/w/index.php?title=Just-in-time_compilation&oldid=1039867749. [Πρόσβαση 19 September 2021].

[3] Wikipedia contributors, «HotSpot (virtual machine),» 18 September 2021 . [Ηλεκτρονικό]. Available: https://en.wikipedia.org/w/index.php?title=HotSpot_(virtual_machine)&oldid=1045085695. [Πρόσβαση 19 September 2021].

[4] Wikipedia contributors, «Python (programming language),» 20 September 2021. [Ηλεκτρονικό]. Available: https://en.wikipedia.org/w/index.php?title=Python_(programming_language)&oldid=1045421303. [Πρόσβαση 23 September 2021].

[5] Wikipedia contributors, «Eclipse (software),» 30 August 2021. [Ηλεκτρονικό]. Available: https://en.wikipedia.org/w/index.php?title=Eclipse_(software)&oldid=1041452332. [Πρόσβαση 19 September 2021].

[6] «GitLab,» [Ηλεκτρονικό]. Available: https://about.gitlab.com/. [Πρόσβαση 19 September 2021].

[7] «GitHub,» [Ηλεκτρονικό]. Available: https://github.com/about. [Πρόσβαση 19 September 2021].

[8] «Subversion (SVN),» Apache, [Ηλεκτρονικό]. Available: https://subversion.apache.org/. [Πρόσβαση 19 September 2021].

[9] «SonarLint,» [Ηλεκτρονικό]. Available: https://www.sonarlint.org/. [Πρόσβαση 19 September 2021].

[10 «Maven,» Apache, [Ηλεκτρονικό]. Available: https://maven.apache.org/. [Πρόσβαση 19 September 2021].

[11 Wikipedia contributors, « Representational state transfer,» Wikipedia , 18 September 2021. [Ηλεκτρονικό]. Available: https://en.wikipedia.org/w/index.php?title=Representational_state_transfer&oldid=1045078330. [Πρόσβαση 19 September 2021].

[12 Wikipedia contributors, «JSON,» Wikipedia, 15 September 2021. [Ηλεκτρονικό]. Available: https://en.wikipedia.org/w/index.php?title=JSON&oldid=1044476333. [Πρόσβαση 19 September 2021].

[13 «Swagger,» SmartBear Software, [Ηλεκτρονικό]. Available: https://swagger.io/docs/specification/2-0/what-is-swagger/. [Πρόσβαση 19 September 2021].

[14 Wikipedia contributors, «YAML,» 3 September 2021. [Ηλεκτρονικό]. Available: https://en.wikipedia.org/w/index.php?title=YAML&oldid=1042224518. [Πρόσβαση 19 September 2021].

[15 «Tomcat,» Apache, [Ηλεκτρονικό]. Available: http://tomcat.apache.org/. [Πρόσβαση 19 September 2021].

[16 «Netty project,» The Netty project, [Ηλεκτρονικό]. Available: https://netty.io/. [Πρόσβαση 19 September 2021].

[17 «SCPSolver,» [Ηλεκτρονικό]. Available: http://scpsolver.org/.

[18 «LPSolve,» [Ηλεκτρονικό]. Available: https://sourceforge.net/projects/lpsolve/. [Πρόσβαση 19 September 2021].

[19 «GLPK,» GNU, [Ηλεκτρονικό]. Available: https://www.gnu.org/software/glpk/. [Πρόσβαση 19 September 2021].

[20 «Gurobi,» Gurobi Optimization, LLC, [Ηλεκτρονικό]. Available: https://www.gurobi.com/. [Πρόσβαση 19 September 2021].

[21 «GAMS,» GAMS Software GmbH, [Ηλεκτρονικό]. Available: https://www.gams.com/. [Πρόσβαση 19 September 2021].

[22 «Mosek,» Mosek, [Ηλεκτρονικό]. Available: https://www.mosek.com/. [Πρόσβαση 19 September 2021].

[23 «Apache Benchmark,» Apache, [Ηλεκτρονικό]. Available: https://httpd.apache.org/docs/2.4/programs/ab.html#.

[24 Wikipedia contributors, «ApacheBench,» 19 May 2020. [Ηλεκτρονικό]. Available: https://en.wikipedia.org/w/index.php?title=ApacheBench&oldid=957645671. [Πρόσβαση 23 September 2021].