Τελική αναφορά εργασίας

Βαγγέλης Κόκκινος – Ιωσήφ Χατζηπαυλής

Στοιχεία μελών ομάδας

- Βαγγέλης Κόκκινος, ΑΜ: 1115202000084, sdi2000084@di.uoa.gr, 3ο έτος
- Ιωσήφ Χατζηπαυλής, ΑΜ: 1115201800299, sdi1800299@di.uoa.gr, 5ο έτος

Πολύ σημαντική σημείωση

Η συγγραφή του κώδικα έγινε κατά κύριο λόγο συγχρονισμένα αντί για ασύγχρονα. Αυτό σημαίνει ότι κανονίζαμε πολύωρες συναντήσεις, είτε δια ζώσεις είτε διαδικτυακές, προκειμένου να κάνουμε ταυτόχρονα το λεγόμενο brainstorming, δίνοντας έτσι άμεσο feedback ο ένας στον άλλο για τις ιδέες μας. Συνεπώς, και οι δύο ασχοληθήκαμε σε διαφορετικό βαθμό με όλα τα ζητούμενα της εργασίας.

Ο Βαγγέλης Κόκκινος ασχολήθηκε κυρίως με:

- Την αρχική δημιουργία της κλάσης και τις συναρτήσεις του χάρτη
- Collisions μεταξύ χαρακτήρων στον πίνακα (έλεγχοι στη συνάρτηση `InsertAt`)
- Εμφάνιση και απόκτηση Magic Potion από τον Avatar
- Εμπόδια του χάρτη (Δέντρα, λίμνες με νερό)
- Διαχείριση ορίων πίνακα (segmentation fault)
- Κινήσεις χαρακτήρων (I/O)
- Pause menu
- Δυναμική δημιουργία τεράτων
- Documentation
- …

Ο Ιωσήφ Χατζηπαυλής ασχολήθηκε κυρίως με:

- Την αρχική δημιουργία των τεσσάρων κλάσεων των χαρακτήρων
- Combat logic and conditions
- Αλλαγή κατάστασης μέρας (day/night cycle)
- Collisions μεταξύ χαρακτήρων στον πίνακα (έλεγχοι στη συνάρτηση `InsertAt`)
- Επανέλεγχος δεδομένων I/O
- Εντοπισμός και διόρθωση σφαλμάτων
- Διαχείριση ορίων πίνακα (segmentation fault)
- Documentation
- …

Περιγραφή του κώδικα

Το πρόγραμμα αποτελείται από 3 αρχεία implementation (`.cpp`) και 2 interface (`.h`).

Ξεκινώντας, το ντουέτο των αρχείων `Characters.cpp` και `Characters.h` υλοποιεί τέσσερις κλάσεις, τη `God`, `Werewolves`, `Vampires` και `Avatar`, με τις 3 τελευταίες να είναι υποκλάσεις και να επεκτείνουν την βασική κλάση `God` μέσω κληρονομικότητας.

Πολλές συναρτήσεις, όπως για παράδειγμα οι τέσσερις Move (`MoveUp`, `MoveDown`, `MoveLeft`, `MoveRight`) χρησιμοποιούνται από όλους (πλην φυσικά του God), οπότε με αυτόν τον τρόπο γλυτώνουμε την αντιγραφή του ίδιου κώδικα για κάθε κλάση ξεχωριστά.

Ακολουθεί η λίστα των συναρτήσεων κάθε κλάσης, μαζί με σύντομη περιγραφή της λειτουργίας κάθε μίας.

`God` class

- Ο constructor αρχικοποιεί τις `private` μεταβλητές `Meds`, `Attack` και `Defense` με τυχαίες τιμές, που ανήκουν βέβαια σε συγκεκριμένο διάστημα ακεραίων.
- Οι «Getter» functions (`getHealth`, `getAttack`, `getDefense`, `getX`, `getY`) επιστρέφουν τις τιμές των αντίστοιχων private μεταβλητών της κλάσης.
- Η συνάρτηση `HealthDecreaseBy` παίρνει σαν όρισμα έναν ακέραιο και με βάση αυτόν μειώνει ανάλογα το επίπεδο ζωής του αντικειμένου.
- Κινήσεις σε κατεύθυνση (`MoveUp`, `MoveDown`, `MoveLeft`, `MoveRight`): Πρόκειται για αρκετά εξειδικευμένες συναρτήσεις που έχουν ως στόχο τη μετακίνηση της οντότητας σε μία θέση πάνω στον πίνακα (τον οποίο θα δούμε στη συνέχεια). Δέχονται σαν όρισμα έναν χαρακτήρα και καλούν δύο συναρτήσεις του χάρτη που αναλαμβάνουν να τον τοποθετήσουν στην επιθυμητή θέση. Ελέγχουν εάν η τοποθέτηση ήταν ανεπιτυχής (γιατί πχ σε εκείνη τη θέση υπάρχει ήδη μία άλλη οντότητα/αντικείμενο) και σε εκείνη την περίπτωση, επιστρέφουν το χαρακτήρα στο σημείο που βρισκόταν προηγουμένως.

`Werewolves`/`Vampires` classes

- Κάθε κλάση έχει τον constructor της, που αναλαμβάνει να τοποθετήσει τα αντικείμενα σε τυχαίες θέσης στον χάρτη. Αυτό επιτυγχάνεται περνώντας έναν δείκτη σε χάρτη σαν όρισμα.
- Η στατική συνάρτηση `HealthRestore` επαναφέρει το επίπεδο ζωής του αντικειμένου στο 10 όταν καλείται.
- Για τις κινήσεις, και οι δύο κλάσεις έχουν τις τέσσερεις βασικές (κληρονομημένες) συναρτήσεις. Οι `vampires` όμως έχουν **άλλες τέσσερις συναρτήσεις επιπλέον** που τους επιτρέπουν να **κινηθούν διαγωνίως** στο χάρτη. Παρ' όλα αυτά, η φιλοσοφία τους είναι ίδια με τις υπόλοιπες τέσσερις βασικές.
- Αυτές τις συναρτήσεις κατευθύνσεων τις επιλέγει τυχαία μία άλλη συνάρτηση `Move`, που κάνει χρήση της `rand`. Είναι ξεχωριστά υλοποιημένη σε κάθε κλάση, λόγω διαφορετικού πλήθους επιλογών κατεύθυνσης.

`Avatar` class

- Ο constructor του `Avatar`, πέρα από το δείκτη σε χάρτη, δέχεται και έναν χαρακτήρα σαν όρισμα. Ο χαρακτήρας αυτός χρησιμοποιείται για τη δήλωση ομάδας. Γίνεται επίσης η αρχικοποίηση των public μεταβλητών.
- Όπως και οι προηγούμενες κλάσεις, έτσι και εδώ υπάρχει και η `Move`, με τη διαφορά ότι οι κατευθύνσεις δεν επιλέγονται τυχαία, αλλά τις εισάγει ο χρήστης μέσω του πληκτρολογίου. Πέρα από εκείνες, η `Move` αναγνωρίζει πότε ζητήθηκε η παύση ή ο τερματισμός του παιχνιδιού, καθώς και η χρήση του μαγικού φίλτρου. Χρησιμοποιείται η βιβλιοθήκη `conio.h`.
- Η `PauseGame`, κάνει αυτό που λέει το όνομά της. Πέρα από αυτό, εμφανίζει το πλήθος των ζωντανών werewolf/vampire και τα διαθέσιμα μαγικά φίλτρα που έχει ο χρήστης. Προσθέσαμε τη λειτουργία να ρωτάει εάν ο χρήστης θέλει να επιστρέψει στο παιχνίδι ή να τερματίσει από εκεί.

- Δύο άλλες συναρτήσεις, η `PrintCurrentDayTime` και η `PrintCurrentTeam`, εμφανίζουν την κατάσταση της μέρας και την ομάδα που επέλεξε ο χρήστης, αντίστοιχα. Αυτό βοηθάει στη χρήση του μαγικού φίλτρου, μιας και οι `Werewolves` μπορούν να κάνουν heal μόνο όταν είναι νύχτα, και οι `Vampires` όταν είναι μέρα.
- Όπως αναφέρθηκε πριν, το μαγικό φίλτρο εφαρμόζεται στην ομάδα που επέλεξε ο χρήστης, ανάλογα με τη μέρα και τη διαθεσιμότητα. Η `UseMagicPot` αναλαμβάνει να ελέγξει την κατάσταση του παιχνιδιού και να καλέσει τη στατική συνάρτηση `HealthRestore` της ανάλογης οντότητας, προκειμένου να επαναφέρει το επίπεδο υγείας των μελών της ομάδας. Εμφανίζει τα κατάλληλα μηνύματα.
- Μία ακόμη στατική συνάρτηση είναι η `IncreaseMagicPotCount`, η οποία όταν καλείται αυξάνει το μετρητή διαθέσιμων μαγικών φίλτρων.

Ας πάμε τώρα στις συναρτήσεις της κλάσης `Map`, που βρίσκονται στα αρχεία `Map2.cpp/Map2.h`.

Map class

- Ο constructor της κλάσης δέχεται δύο ακεραίους σαν ορίσματα, τα οποία της τα δίνει η main όταν κατασκευάζεται ο χάρτης για πρώτη φορά. Πρόκειται για τις διαστάσεις του πίνακα που θα δημιουργηθεί με δυναμική δέσμευση μνήμης. Αφού αυτή η διαδικασία ολοκληρωθεί, ορίζουμε το έδαφος (αστερίσκοι σε κάθε κελί) και στη συνέχεια τοποθετούμε σε τυχαίες θέσεις τα δέντρα, τις λίμνες με νερό, καθώς και το μαγικό φίλτρο.
- Η πολύ σημαντική συνάρτηση `InsertAt` αναλαμβάνει να τοποθετήσει τον χαρακτήρα που της δόθηκε στις θέσεις που επίσης της δόθηκαν. Επιστρέφει μία μεταβλητή τύπου bool για να δηλώσει στη συνάρτηση που την κάλεσε εάν πέτυχε ή απέτυχε η τοποθέτηση. Αφού γίνει ο έλεγχος των συντεταγμένων ώστε να είμαστε μόνο μέσα στα όρια του πίνακα (και να αποτύχουμε τα πολυαγαπημένα segmentation fault) ξεκινάει να χωρίζει περιπτώσεις, με βάση το χαρακτήρα που της δόθηκε και τον χαρακτήρα που ήδη βρίσκεται στο σημείο τοποθέτησης. Με λίγα λόγια, η συνάρτηση αυτή είναι υπεύθυνη για την αποφυγή collisions μεταξύ οντοτήτων και επιτυγχάνεται μέσω χαρακτήρων.
- Η `RemoveFrom` αφαιρεί έναν χαρακτήρα από την προηγμένη θέση του, χάρη στους ακεραίους που δέχεται σαν ορίσματα. Αποφεύγονται οι λανθασμένες αντικαταστάσεις δέντρων και λιμνών.
- Η `PrintMap`, όπως λέει και το όνομά της, τυπώνει ολόκληρο το χάρτη και τα περιεχόμενά του. Γίνεται χρήση ειδικών χαρακτήρων για εμφάνιση χρωμάτων στο terminal, ώστε οι οντότητες να διακρίνονται πιο εύκολα από το χρήστη.
- Οι getter functions getX και getY επιστρέφουν σε όποιον τις καλεί τις διαστάσεις του πίνακα.
- Ο destructor της Map υλοποιήθηκε για να αποδεσμεύει δυναμικά τη μνήμη που χρειάστηκε ο πίνακας του χάρτη.

Client

Το `client.cpp` περιέχει τη συνάρτηση `main`, η οποία αναλαμβάνει να ενορχηστρώσει τις εκτελέσεις των παραπάνω συναρτήσεων που βρίσκονται στις κλάσεις κι έτσι να υλοποιηθεί το παιχνίδι. Περισσότερα θα βρείτε στην παρουσίαση της εργασίας στο βίντεο.

Παραδοχές

- Ο `Avatar` αναπαρίσταται με τον χαρακτήρα «`A`» στον χάρτη, αντί για το αρχικό γράμμα της ομάδας που υποστηρίζει.
- Κάθε `Werewolf` αναπαρίσταται με ένα μωβ “`W`” και κάθε `Vampire` με ένα κόκκινο “`V`”

- Παρόν είναι κάθε στιγμή του παιχνιδιού ο τύπος της μέρας (DAY/NIGHT), καθώς και η ομάδα που επέλεξε να υποστηρίζει ο χρήστης.

Απαιτήσεις της εργασίας που δεν υλοποιήθηκαν

- Η επιλογή ενός τέρατος να υποχωρήσει όταν βρεθεί αντιμέτωπο με τέρας της αντίπαλης ομάδας. Αντί αυτού, υποχρεώνεται να επιτεθεί ο ισχυρότερος.

Software που χρησιμοποιήθηκε για τη συγγραφή κώδικα

Αρχικά, χρησιμοποιήθηκε το Visual Studio Code σε περιβάλλον Ubuntu Linux, αλλά λόγω της βιβλιοθήκης `conio.h` που είναι windows-only, χρειάστηκε να μεταβούμε σε περιβάλλον Windows και να γράψουμε σχεδόν ολόκληρο τον κώδικα με το Visual Studio Community (καμία σχέση με το Visual Studio Code, παρά την ομοιότητα στο όνομα).

Το πρότυπο της γλώσσας C++ που χρησιμοποιήσαμε είναι το ISO C++20 Standard `(/std:c++20)`.

Το debugging έγινε με τη ρύθμιση "`Debug x64`" του Visual Studio.

Δυσκολία εργασίας

Κατά τη δική μας άποψη, η εργασία ήταν μέτριας δυσκολίας. Μερικά από τα ζητούμενα ήταν απαιτητικά, ενώ κάποια άλλα όχι.

Για παράδειγμα, η δημιουργία ενός real-time παιχνιδιού απαιτεί γνώσεις που δεν σχετίζονται με τον Αντικειμενοστραφή Προγραμματισμό (processes, threads, κτλ.).

Η εκφώνηση της εργασίας ήταν πλημμελώς δομημένη και θα μπορούσε να βελτιωθεί.

Από την άλλη μεριά, πιστεύουμε ότι η εργασία αυτή παρουσιάζει πολύ έντονα τα πλεονεκτήματα της αντικειμενοστρέφειας, τα οποία σε πολλές περιπτώσεις, κυριολεκτικά λύνουν χέρια. Ένα προφανές παράδειγμα είναι η κληρονομικότητα.

Χωρίς καμία αμφισβήτηση, οι γνώσεις που αποκτήσαμε στα προηγούμενα μαθήματα «Εισαγωγή στον Προγραμματισμό» και «Δομές Δεδομένων και Τεχνικές Προγραμματισμού» αποδείχθηκαν ιδιαίτερα χρήσιμες στο να υλοποιήσουμε τη συγκεκριμένη εργασία, ακολουθώντας πάντα τις νέες φιλοσοφίες του Αντικειμενοστραφούς Προγραμματισμού και τις επιπρόσθετες, βελτιωμένες δυνατότητες της C++ σε σχέση με τη C.

GitHub (code)

https://github.com/sdi2000084/oop-project-kokkinos-chatzipavlis

YouTube (video)

https://youtu.be/YPXF7AMleag Σημείωση: Τμήματα κώδικα που εμφανίζονται στο βίντεο μπορεί να έχουν αλλάξει .