University of Patras - Computer Engineering & Informatics Department

Οντοκεντρικός Προγραμματισμός ΙΙ - C++

Εργαστηριακή Άσκηση

Παυλόπουλος Φίλιππος
A.M.: 5055 • 9ο Εξάμηνο
pavlopou@ceid.upatras.gr

Παπαδομανωλάκης Αριστείδης
A.M.: 5041 • 9ο Εξάμηνο
papadomano@ceid.upatras.gr

Παπαϊωάννου Κωνσταντίνος
A.M.: 5052 • 9ο Εξάμηνο
papaioann@ceid.upatras.gr

January 17, 2015

⚠ **Προσοχή!** Το πρόγραμμα τρέχει αποκλειστικά σε Linux.

Υπόδειξη. *Για την καλύτερη εμπειρία χρήστη προτείνεται η εκτέλεση του προγράμματος να γίνει σε πλήρη οθόνη.*

⚠ Οδηγίες εκτέλεσης: Τοποθετήστε το φάκελο `5041_5052_5055` στον υπολογιστή σας. Πηγαίντε στη διαδρομή `/path/to/5041_5052_5055/Source` και εκτελέστε την εντολή `make`. Θα δημιουργηθεί το εκτελέσιμο `navy`, το οποίο μπορείτε να τρέξετε με την εντολή `./navy`.

Περίληψη προσομοίωσης

Αρχικοποίηση

Η προσομοίωση ξεκινάει με τη δημιουργία ενός δισδιάστατου τετραγωνικού χάρτη σταθερού μεγέθους[1]. Πάνω σε αυτόν τοποθετούνται σε τυχαίες θέσεις αποθέματα θησαυρού, και καθορίζεται η ένταση του καιρού συνολικά. Μετά δημιουργούνται τα πλοία με σταθερό αριθμό ανά κατηγορία (πειρατικά, εμπορικά, εξερευνητικά, επισκευαστικά, εθελοντικά και "jack sparrow"). Τα πλοία αποθηκεύονται σε μια δομή vector και εισέρχονται στο χάρτη έτσι ώστε να μην υπάρχουν συγκρούσεις (2 πλοία σε μια θέση).

Εκτέλεση

Έπειτα ξεκινάει η εκτέλεση της προσομοίωσης. Εμφανίζεται στο χρήστη το σύνολο των πλοίων με τις θέσεις τους στο χάρτη και τις υπόλοιπες πληροφορίες τους, δίνοντας του

[1]Οι μεταβλητές για το μέγεθος του χάρτη καθώς και τον αριθμός των προς κατασκευή πλοίων ανά είδος βρίσκονται στο αρχείο header.h με σχετικά ονόματα.

την ευκαιρία να ξεκινήσει την προσομοίωση, να ζητήσει πληροφορίες σχετικές με τα πλοία ή με το χάρτη, αλλά και να επέμβει σε αυτά προσθέτοντας/αφαιρώντας πλοία, λιμάνια ή θησαυρό. Όταν ο χρήστης επιλέξει να ξεκινήσει την προσομοίωση, ελέγχονται όλες οι θέσεις του χάρτη για λιμάνια, θησαυρό ή κακοκαιρία, και αλλάζουν οι καταστάσεις των πλοίων που επηρεάζονται από αυτές τις συνθήκες (βλ. § Επεξήγηση Αρχείων και Κλάσεων). Στη συνέχεια, όλα τα πλοία κάνουν μια κίνηση (όσο τους επιτρέπει η ταχύτητά τους) και μετά κάνουν τη λειτουργία που ορίζει ο τύπος τους (βλ. § Επεξήγηση Αρχείων και Κλάσεων). Προτού τελειώσει η μέρα όπου τα πλοία προχωράνε και κάνουν τις λειτουργίες τους, ανανεώνεται ο καιρός (αυξομοιώνεται σε κάθε θέση του χάρτη) και έτσι ολοκληρώνεται μια μέρα (ένας γύρος) προσομοίωσης.
Μετά το τέλος κάθε μέρας, ο χρήστης μπορεί τα αποτελέσματα της και ανά πάσα στιγμή να επέμβει όπως προαναφέρθηκε.
Η προσομοίωση ολοκληρώνεται επιτυχώς σε τρεις περιπτώσεις:

(i) Ο χρήστης τερματίζει την προσομοίωση με σχετική εντολή

(ii) Κάποιο απ' όλα τα ενεργά πλοία φτάνει το στόχο θησαυρού

(iii) Όλα τα πλοία έχουν καταστραφεί και δεν υπάρχει κανένα ενεργό

Λίστα Αρχείων Κώδικα

`main.cpp`	*Αρχικοποίηση και προσομοίωση του προγράμματος*
`header.h`	*Βασική βιβλιοθήκη όλων των κλάσεων*
`kbhit.h`	*Συνάρτηση για άμεση είσοδο πληκτρολογίου*
`map.h`	*Κλάσεις και μέθοδοι για το χάρτη*
`ship.h`	*Συναρτήσεις και μέθοδοι για την υπερκλάση Ship*
`pirate.h`	*Συναρτήσεις και μέθοδοι για τα πειρατικά πλοία*
`explorer.h`	*Συναρτήσεις και μέθοδοι για τα εξερευνητικά πλοία*
`repairer.h`	*Συναρτήσεις και μέθοδοι για τα επισκευαστικά πλοία*
`merchant.h`	*Συναρτήσεις και μέθοδοι για τα εμπορικά πλοία*
`new_day.h`	*Συνάρτηση εκτέλεσης προσομοίωσης*
`menu.h`	*Συνάρτηση διεπαφής του χρήστη*

main.cpp

Περιέχει την κύρια συνάρτηση main(). Αρχικά, αρχικοποιούνται όλες οι παράμετροι. Δημιουργείται ο χάρτης, δημιουργούνται τα πλοία στο vector $ships$ και εισάγονται στο χάρτη. Μετά, ξεκινάει ο βρόγχος επανάληψης της προσομοίωσης και καλούνται διαδοχικά οι συναρτήσεις που εκτυπώνουν το μενού (βλ. § menu.h), τις πληροφορίες της προσομοίωσηςβλ. § informations(int) και οι συναρτήσεις που εξελίσσουν την προσομοίωση (βλ. § new_day.h). Μετά και έξω από το βρόγχο της προσομοίωσης καλείται η end() (βλ. § end()), όπου εκτυπώνει το τελικό μήνυμα και τέλος τερματίζεται το πρόγραμμα.

Επεξήγηση Αρχείων και Κλάσεων

header.h

Περιέχει τις δηλώσεις όλων των κλάσεων που χρησιμοποιούμε. Συγκεκριμένα, όπως φαίνεται και στο διάγραμμα των κλάσεων (βλ. § Διάγραμμα Κλάσεων) έχουμε μια κύρια κλάση για τον γενικό όρο πλοίο (Ship) με τέσσερις υποκλάσεις, τα πειρατικά (Pirate), τα εμπορικά (Merchant), μια εξερευνητικά (Explorer) και τα επισκευαστικά (Repairer). Επίσης υπάρχουν δυο υποκλάσεις για εξειδικευμένα πλοία που ορίσαμε, η κλάση Jack_sparrow (πιο γρήγορα πειρατικα πλοίο που προκαλούν μεγαλύτερη ζημιά) και η Volunteer (επισκευαστικά πλοία που δεν παίρνουν χρυσό από τα πλοία που επισκευάζουν). Επίσης υπάρχει η κλάση Point_map, η οποία ορίζει κάθε θέση του χάρτη με τις μεταβλητές και της μεθόδους που χρειάζεται (βλ. § map.h). Ο χάρτης, δηλαδή, είναι ένας δισδιάστατος πίνακας από αντικείμενα της κλάσης Point_map. Όλες οι μεταβλητές των κλάσεων έχουν δηλωθεί private και έτσι μπορούμε να τις χειριστούμε με μεθόδους τύπου get και set που έχουμε υλοποιήσει. Όλες οι μέθοδοι των κλάσεων έχουν δηλωθεί public. Τέλος, ορίζεται μια βοηθητική κλάση για την υλοποίηση της συνάρτησης kbhit() (βλ. § kbhit.h).

ship.h

Το αρχείο αυτό περιέχει τις υλοποιήσεις των μεθόδων της κλάσης $Ship$.
Περιέχει τον constructor και τον destructor της $Ship$ όπως και τις απαραίτητες μεθόδους get και set οι οποίες επιτρέπουν στη main όπως και σε υποκλάσεις της κλάσης $Ship$ να μεταβάλουν τις τιμές των μεταβλητών της που είναι private.
Επίσης, περιέχει την κενή δήλωση της μεθόδου λειτουργίας κάθε πλοίου . Συγκεκριμένα, γίνεται υπερφόρτωση της μεθόδου $operation()$ με κενό όρισμα (για τη λειτουργία των πειρατικών, των εμπορικών και των επισκευαστικών πλοίων) ή με όρισμα τις συντεταγμένες (για τη λειτουργία των εξερευνητικών).

move(int)

Η μέθοδος αυτή δέχεται σαν όρισμα την κατεύθυνση στην οποία θα μετακινηθεί το πλοίο, είτε από τη $main()$, ορίζοντας σε κάθε επανάληψη μια τυχαία τιμή για να μετακινηθεί, ή όπου ένα πλοίο χρειάζεται να μετακινηθεί προς μια κατεύθυνση όπως του το ορίζει η λειτουργία του. Τα ορίσματα που δέχεται είναι ακέραιοι 0-3 (δεξιά, αριστερά, πάνω, κάτω αντίστοιχα). Σε κάθε περίπτωση η συνάρτηση ελέγχει αν με βάση τη θέση και την ταχύτητα τού πλοίου πρόκειται να πέσει έξω από τα όρια του χάρτη ή πάνω σε άλλο πλοίο. Αν όχι μετακινείται με βάση την ταχύτητά του και την κατεύθυνση που έχει επιλεγεί. Αν πρόκειται να πέσει έξω από τα όρια του χάρτη μετακινείται στο ακραίο σημείο του χάρτη αν εκεί δεν υπάρχει άλλο πλοίο.

informations(int)

Η μέθοδος $informations()$ εμφανίζει πληροφορίες για κάθε πλοίο σε κάθε γύρο της προσομοίωσης. Παράλληλα, υπάρχει ένας signal handler για να τυπώνεται μήνυμα όταν το πρόγραμμα τερματίζει με Ctrl-C. Τέλος η συνάρτηση

end()

εμφανίζει πληροφορίες για την τελική έκβαση της προσομοίωσης όταν το πρόγραμμα τερματίζει κανονικά.

repairer.h

Στο repairer.h έχουμε τη δημιουργία ενός επισκευαστικού πλοίου. Η δημιουργία του γίνεται στη συνάρτηση create_repairer(vector<ship*>)[2]. Πέρα απο το κλασικό επισκευαστικό πλοίο δημιουργείται και το επισκευαστικό πλοίο-εθελοντής όπου επισκευάζει γειτονικά πλοία χωρίς αντάλλαγμα χρυσού. Η δημιουργία αυτού του πλοίου γίνεται στη συνάρτηση create_repairer(vector<ship*>).

Repairer :: operation()

Στη μέθοδο αυτή ένα επισκευαστικό πλοίο (είτε repairer είτε volunteer) ελέγχει αν υπάρχει κάποιο γειτονικό πλοίο και καλεί τη μέθοδο repair() που περιγράφεται παρακάτω για να το επισκευάσει.

Repairer :: repair(int x, int y)

Η μέθοδος αυτή καλείται όταν ένα επισκευαστικό πλοίο έχει βρει κάποιο γειτονικό πλοίο. Δέχεται σαν όρισμα τις συντεταγμένες του γειτονικού πλοίου και αυξάνει κατά ένα ποσοστό την αντοχή αυτού. Στη συνέχεια, αν το πλοίο που έκανε την επισκευή δεν είναι της κλάσης Volunteer, τότε παίρνει ως αντάλλαγμα ένα ποσοστό θυσαυρού από το πλοίο που επισκεύασε. Αν το πλοίο είναι volunteer τότε δεν κάνει τίποτα άλλο.

Σημείωση: Τη μέθοδο λειτουργίας του πλοίου Repairer αλλά και τη μέθοδο repair() την εκτελούν και τα αντικείμενα της υποκλάσης Volunteer.

explorer.h

Σ' αυτό το αρχείο περιλαμβάνεται η δημιουργία εξερευνητικών πλοίων και η λειτουργία τους. Κάθε πλοίο εξερεύνησης καλεί τη μέθοδο λειτουργίας του και ελέγχει αν σε κάποια γειτονική θέση υπάρχει πειρατικό πλοίο ή κακοκαιρία. Αν ισχύει κάτι από τα παραπάνω τότε το πλοίο απομακρύνεται προς την αντίθετη κατεύθυνση. Αν δεν ισχύει τίποτα από τα παραπάνω, το πλοίο κινείται τυχαία. Αρχικά γίνεται έλεγχος για την ύπαρξη πειρατικού πλοίου και έπειτα για την συνθήκες κακοκαιρίας.

merchant.h

Εδώ δημιουργούνται εμπορικά πλοία. Στο αρχείο αυτό υπάρχει και η λειτουργία του εμπορικού πλοίο. Κάθε εμπορικό πλοίο καλεί τη μέθοδο λειτουργίας του και ελέγχει αν υπάρχει γειτονικό λιμάνι. Αν ναι, αυξάνει το απόθεμα θυσαυρού του, διαφορετικά συνεχίζει χωρίς να αυξήσει το θυσαυρό του.

[2]Σε κάθε αρχείο πλοίων .h όπου υπάρχει ο constructor του κάθε είδος πλοίου υπάρχει και η συναρτηση create_<ShipType>(vector<ship*>) όπου <ShipType> το είδος κάθε πλοίου (volunteer, jack_sparrow, repairer, merchant, explorer, pirate). Στην συνάρτηση αυτή δημιουργείται ένα πλοίο όπου αρχικά εισάγεται στο vector *ships* για διευκόλυνση χειρισμού του. Έπειτα, αρχικοποιείται τυχαία η θέση του στο χάρτη παίρνοντας συντεταγμένες.

pirate.h

Τέλος στο αρχείο αυτό γίνεται η δημιουργία των πειρατικών πλοίων. Εκτός από τα κλασικά πειρατικά, δημιουργούνται και πειρατικά με κάποιες διαφορετικές ιδιότητες. Τα ειδικά πειρατικά πλοία "Jack_sparrow" προκαλούν μεγαλύτερη ζημιά από τα κανονικά (15 έναντι 10 των κλασικών). Οι δημιουργία των πλοίων αυτών γίνεται στις συναρτήσεις create_pirate(vector<Ship*>) και create_jack(vector<Ship*>) αντίστοιχα.

Pirate :: operation()

Η μέθοδος αυτή καλείται απ' όλα τα πειρατικά πλοία (κλασικά και Jack_sparrow) και αποτελεί τη μέθοδο λειτουργίας τους. Η διεργασία που λαμβάνει χώρα εδώ είναι ο έλεγχος για γειτονικά πλοία. Αν κάποιο πειρατικό συναντήσει ένα γειτονικό πλοίο καλείται η μέθοδος attack(int x, int y) στην οποία γίνεται επίθεση.

Pirate :: attack(int x, int y)

Η μέθοδος αυτή δέχεται ως όρισμα τις συντεταγμένες του γειτονικού πλοίου και επιτίθεται σε αυτό. Αρχικά μειώνει την αντοχή του γειτονικού πλοίου και έπειτα αυξάνει το απόθεμα χρυσού στο πειρατικό. Τέλος, το πειρατικό που έκανε την επίθεση αυξάνει και κατά λίγο την αντοχή του.

Σημείωση: Τις μεθόδους του συγκεκριμένου αρχείου, τις εκτελούν τόσο τα κλασικά πειρατικά (pirate) όσο και τα "Jack_sparrow" που αποτελούν υποκλάση των πειρατικών.

map.h

Στο αρχείο αυτό περιέχεται ο ορισμός της κλάσης Point_map, αντικείμενο της οποίας είναι ο χάρτης στον οποίο κινούνται τα πλοία. Περιέχονται επίσης ο constructor της κλάσης Point_map καθώς και απαραίτητες get και set μέθοδοι που χρειάζονται για να χειριζόμαστε τις μεταβλητές της κλάσης, οι οποίες είναι private.

check_weather()

Η μέθοδος check_weather(), ελέγχει αν υπάρχει πλοίο στη ζητούμενη θέση και αν ναι, ελέγχει την ένταση του καιρού στη συγκεκριμένη θέση του χάρτη. Αν αυτή είναι πάνω από 7 μειώνει την αντοχή του πλοίου.

check_treasure()

Η μέθοδος check_treasure() ελέγχει αν υπάρχει πλοίο στη ζητούμενη θέση και αν ναι, ελέγχει αν υπάρχει θησαυρός. Αν διαπιστώσει οτι υπάρχει θησαυρός αυξάνει το απόθεμα θησαυρού του συγκεκριμένου πλοίου και αφαιρεί τον θησαυρό απ'ό αυτή τη θέση.

check_port()

Η μέθοδος check_port() ελέγχει αν υπάρχει λιμάνι στη ζητούμενη θέση του χάρτη και αν ναι, τότε ελέγχει αν υπάρχουν πλοία στις γειτονικές θέσεις του λιμανιού. Για κάθε πλοίο που υπάρχει σε κάθε γειτονική θέση καλείται η μέθοδος $do\_ship()$, η οποία ελέγχει την

ταυτότητα του πλοίου και αν αυτό είναι πειρατικό του προκαλεί ζημιά (μειώνει την αντοχή του), ενώ αν είναι από τους υπόλοιπους τύπους πλοίων το επισκευάζει (αυξάνει την αντοχή του).

weather_renewal()

Τέλος, υπάρχει η μέθοδος weather_renewal() η οποία σε κάθε γύρο τής προσομοίωσης καλείται για κάθε θέση του χάρτη, και με τυχαίο τρόπο αυξάνει ή μειώνει την ένταση του καιρού στο σημείο αυτό.

kbhit.h

Επειδή η συνάρτηση kbhit() της βιβλιοθήκης conio.h την οποία χρειαζόμαστε για να σταματάει ο χρήστης ασύγχρονα τον βρόγχο στον οποίο γίνεται η προσομοίωση στη main με το πάτημα ενός πλήκτρου δουλεύει μόνο σε Windows και εμείς υλοποιήσαμε εξολοκλήρου την άσκηση σε linux πήραμε έτοιμη την υλοποίηση της kbhit() για linux από τον παρακάτω σύνδεσμο:
`http://www.linux-sxs.org/programming/kbhit.html`

new_day.h

Καλεί τις συναρτήσεις και μεθόδους υπεύθυνες για την εξέλιξη της προσομοίωσης. Αρχικά, ελέγχει όλα τα σημεία του χάρτη για κακοκαιρία, θησαυρό και λιμάνια, έπειτα αφαιρεί από το χάρτη και το vector $ships$ όλα τα πλοία που δεν έχουν αντοχή και στη συνέχεια καλεί τις μεθόδους κίνησης και λειτουργίας για κάθε πλοίο ξεχωριστά. Τέλος, ανανεώνει τον καιρό (αυξομειώνει τυχαία) και επιστρέφει το vector $ships$, που πήρε σαν όρισμα, στη main().

menu.h

Είναι υπεύθυνο για όλες τις ενέργειες του χρήστη απέναντι στην προσομοίωση. Αρχικά, εμφανίζει το μενού επιλογών και αλλάζει την κατάσταση της προσομοίωσης (σε εξέλιξη/σε παύση) ανάλογα με τις επιλογές του χρήστη. Του δίνει τις επιλογές:

- εκκίνηση/παύση προσομοίωσης
- προβολή πληροφοριών πλοίων
- προβολή πληροφοριών χάρτη
- εισαγωγή πλοίου
- αφαίρεση πλοίου
- επεξεργασία χάρτη
- τερματισμός προσομοίωσης

Με βάση τις επιλογές του χρήστη το πρόγραμμα απαντά κατάλληλα, είτε προβάλλοντας μηνύματα είτε τροποποιώντας τον χάρτη και το vector $ships$ των πλοίων και επιστρέφει το vector $ships$ για να συνεχίσει η προσομοίωση.
Αξίζει να σημειωθεί ότι το μενού είναι στη διάθεση του χρήστη μόνιμα ανεξαρτήτως κατάστασης της προσομοίωσης και δίνει στο χρήστη τη δυνατότητα παύσης ή συνέχισης αυτής.

Screenshots

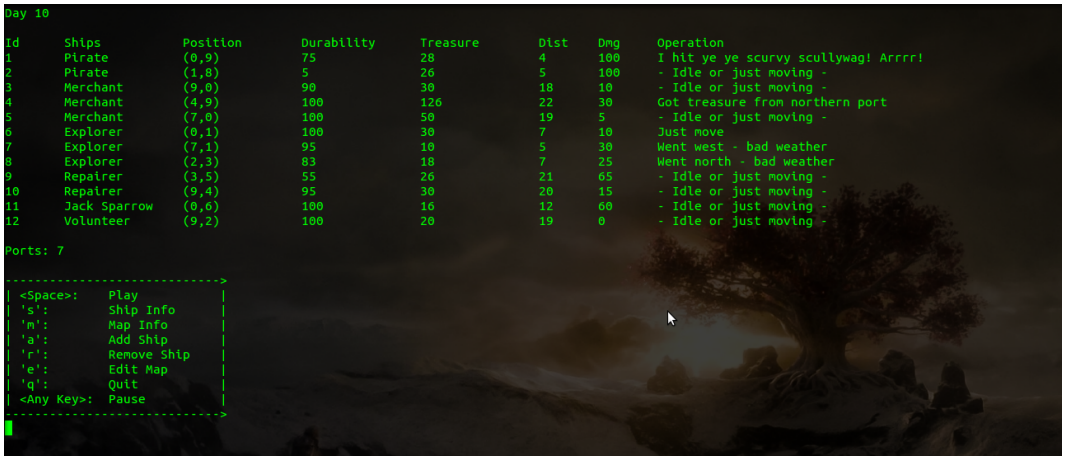

Figure 1: *Κατά τη διάρκεια εκτέλεσης της προσομοίωσης*

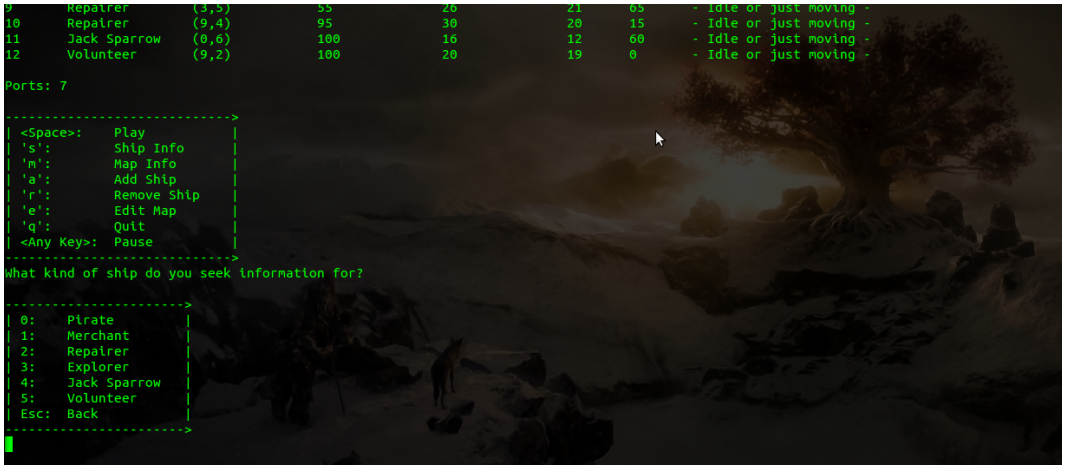

Figure 2: *Παράδειγμα του μενού - πληροφορίες πλοίων*

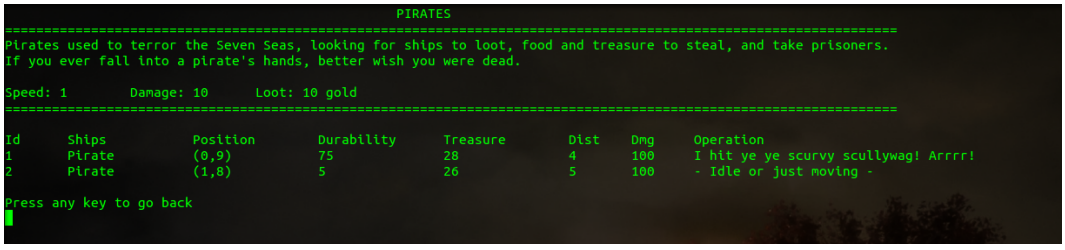

Figure 3: *Υπόδειγμα - Πληροφορίες πειρατικών πλοίων*

Figure 4: *Παράδειγμα του μενού - πληροφορίες χάρτη*

Figure 5: *Τέλος προσομοίωσης με συγκέντρωση απαιτούμενου θησαυρού*

Διάγραμμα Κλάσεων

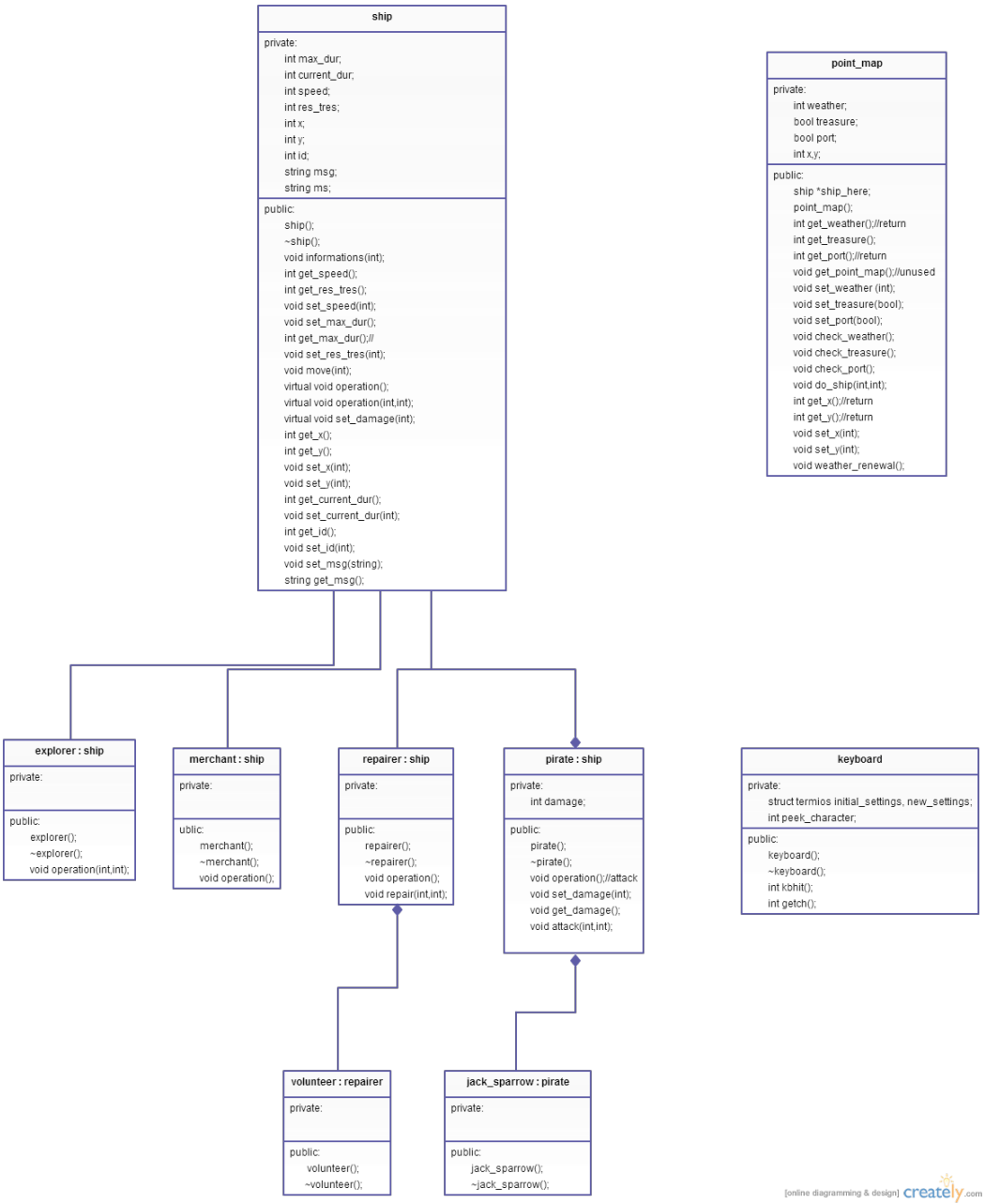

Figure 6: *Διάγραμμα κλάσεων*