Vampires vs Werewolves

Project Report

Created by:

- Δημήτριος Αναγνωστόπουλος (sdi2100002), sdi2100002@di.uoa.gr, 2ο έτος
- Γεώργιος-Αλέξανδρος Κώστας (sdi2100080), k.alexandros299@gmail.com, 2ο έτος

Github Link: https://github.com/AlexKostas/VampiresVsWerewolves
Youtube Video Link: https://www.youtube.com/watch?v=IXnz3B6AptE

Καταμερισμός Εργασιών:

Η σχεδίαση/συγγραφή του κώδικα, των διάφορων reports καθώς και η λήψη του βίντεο έγινε ταυτόχρονα και από τους δύο μας. Τα commits στο github έγιναν και από τους 2 λογαριασμούς μας, ανάλογα με το μηχάνημα που χρησιμοποιούσαμε εκείνη την στιγμή.

IDE/Compiler version/Compilation Instructions:

Για την συγγραφή της εργασίας χρησιμοποιήσαμε το Microsoft Visual Studio 2022 και η μεταγλώττιση έγινε με τον ενσωματωμένο compiler που προσφέρει το περιβάλλον. Η εργασία χρησιμοποιεί Windows specific library calls, οπότε κατά πάσα πιθανότητα δεν θα δουλεύει σωστά σε άλλα λειτουργικά συστήματα. Το πρόγραμμα δοκιμάστηκε σε διαφορετικές συσκευές οι οποίες τρέχουν όλες Windows 10.

Μαζί με τον κώδικα της εργασίας, περιλαμβάνεται και το solution (.sln) file του project, το οποίο και μπορείτε να ανοίξετε μέσω του Visual Studio προκειμένου να μεταγλωττίσετε το πρόγραμμα με τις αντίστοιχες επιλογές που προσφέρει το IDE.

Παραδοχές:

Σε γενικές γραμμές η υλοποίηση του παιχνιδιού μας βασίστηκε ακριβώς στις οδηγίες που έχουν δωθεί. Συγκεκριμένα το παιχνίδι είναι real-time, που σημαίνει ότι η δράση θα εξελίσσεται ανεξάρτητα από το αν και πότε ο παίκτης θα δώσει κάποιο input. Επίσης, η ζωή των vampires/werewolves έχει οριστεί στους 5 πόντους, ώστε να τελειώνει το παιχνίδι σχετικά γρήγορα. Τέλος, προσθέσαμε το requirement το εύρος τιμών του attack ενός werewolf/vampire να είναι πάντα μεγαλύτερο από το εύρος τιμών του defense του. Διαφορετικά, οι περισσότερες επιθέσεις από ένα entity σε ένα άλλο δεν κάνουν καθόλου damage και το παιχνίδι δεν τελειώνει ποτέ.

How to play:

Έχουν υλοποιηθεί **όλα** τα ζητούμενα τις εργασίας, οπότε ο χρήστης έχει τη δυνατότητα ανά πάσα στιγμή να κάνει τις εξής ενέργειες:

- Κίνηση του avatar (A) πάνω/κάτω/δεξιά/αριστερά με τα βελάκια του πληκρολογίου. Ο avatar δεν μπορεί να κουνηθεί πέρα από τα όρια του χάρτη ή να περάσει πάνω από δέντρα (T) ή νερό (~).
- Συλλογή potion όταν ο παίκτης περάσει πάνω απο τετράγωνο με το γράμμα P.

- Pause/Unpause με το πλήκτρο Tab. Όσο το παιχνίδι είναι paused ο χρήστης μπορεί να δει επιπλέον πληροφορίες όπως η ζωή του κάθε entity του παιχνιδιού.
- Heal teammates με το αγγλικό lower case πλήκτρο k αν διαθέτει αρκετά potions και είναι μέρα για werewolves ή νύχτα για vampires. Ένα μήνυμα θα εμφανιστεί για λιγό στην οθόνη που θα δείχνει αν η χρήση του potion ήταν επιτυχημένη ή οχι.
- Έξοδο του παιχνιδιού με το ESC. Όταν το παιχνίδι τελειώσει, θα εμφανιστεί ο νικητής και ο χρήστης θα πρέπει να δώσει οποιαδήποτε τυχαία είσοδο στο terminal προκειμένου να κλείσει το παράθυρο.

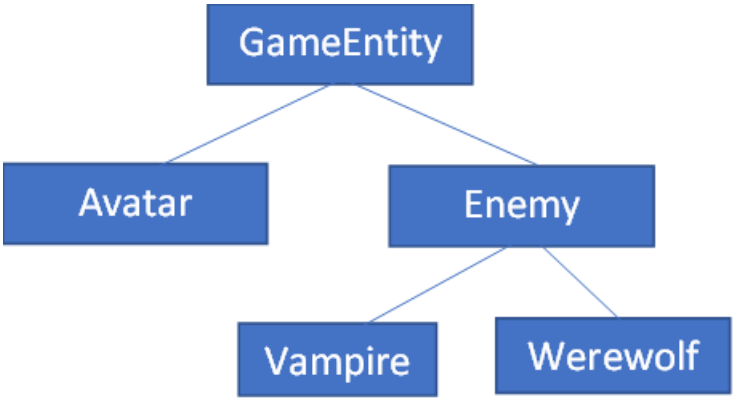
Ανεξάρτητα από τα παραπάνω, σε κάθε γύρο το κάθε Vampire και Werewolf κινείται ανεξάρτητα στο map και επιλέγει αν θα επιτεθεί ή θα κάνει heal κάποιον teammate του. Να σημειωθεί ότι, σύμφωνα με την εκφώνηση, τα vampires κινούνται ή επιτίθενται και διαγώνια.

Βαθμός Δυσκολίας:

Από πλευράς των requirements που έπρεπε να υλοποιηθούν, δεν βρήκαμε την εργασία ιδιαίτερα απαιτητική. Ωστόσο, θεωρούμε πως η προτίμιση που εκφράστηκε προφορικά από τον διδάσκοντα να είναι το παιχνίδι real-time προσθέτει αχρείαστες δυσκολίες που δεν συνάδουν με τους διδακτικούς σκοπούς ενός μαθήματος πάνω στον αντικειμενοστραφή προγραμματισμό. Επίσης, θα βρίσκαμε χρήσιμη μια περαιτέρω καθοδήγηση πάνω στο κομμάτι της αντικειμενοστραφούς ανάλυσης, καθώς η εκφώνηση ήταν υπερβολικά γενική για να καταλάβουμε ποιο είναι το standard της σχεδίασης του κώδικα που μπορεί να πάρει το άριστα.

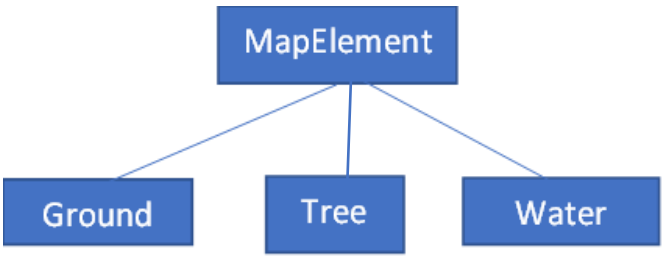
Ιεραρχία Κλάσεων:

Η abstract κλάση GameEntity είναι η parent κλάση για όλες τις οντότητες που μπορούν να υπάρξουν στο παιχνίδι. Περιέχει τις pure virtual συναρτήσεις Print(), Update() και GetTeam() που πρέπει υποχρεωτικά να υλοποιεί το κάθε επιμέρους Entity. Επειδή τα Werewolves και τα Vampires μοιράζονται αρκετές πληροφορίες και λειτουργίες, τις οποίες, όμως, ο άβαταρ δεν χρειάζεται, έχουμε ορίσει την abstract κλάση Enemy, από την οποία κληρονομούν τα vampires και τα werewolves και στην οποία φιλοξενούνται οι κοινές λειτουργίες και τα κοινά δεδομένα των οντοτήτων αυτών. Ένα παράδειγμα τέτοιας λειτουργίας είναι η συνάρτηση Update(), η οποία αποφασίζει και εκτελεί την κίνηση/επίθεση της οντότητας σε κάθε γύρο. Η συνάρτηση αυτή είναι κοινή και για τις δύο κατηγορίες εχθρών και για αυτόν τον λόγο βρίσκεται στο Enemy class. Το μόνο που διαφέρει είναι το πού μπορεί να πάει, ποιοι είναι οι εχθροί και ποιοι οι σύμμαχοι της κάθε οντότητας, κάτι που αναλαμβάνει η ίδια η οντότητα να αποφασίσει υλοποιώντας τις αντίστοιχες pure virtual συναρτήσεις getEnemies(), getAllies(), getPossibleMovementCells() της Enemy. Τέλος, ο ίδιος ο Avatar κληρονομεί απο την GameEntity, καθώς δεν μοιράζεται άλλα κοινά χαρακτηριστικά με την Enemy.



Η κάθε κλάση ενθυλακώνει πλήρως τις λειτουργίες που αντιστοιχούν σε κάθε οντότητα, παρέχοντας ένα όσο το δυνατόν πιο καθαρό public interface, το οποίο περιλαμβάνει τις ίδιες τις λειτουργίες πάνω στο αντικείμενο και όχι τις πληροφορίες που χρειάζεται κάποιος τρίτος για να πραγματοποιήσει αυτές τις λειτουργίες. Με άλλα λόγια το κάθε Entity ξέρει το ίδιο πώς να επιτελέσει την εκάστοτε λειτουργία για την οποία είναι υπεύθυνο.

Η abstract κλάση MapElement χρησιμοποιείται για να περιγράψει τις κοινές λειτουργίες και δεδομένα που θα έχουν όλα τα κελιά του χάρτη. Όλα τα κελιά για παράδειγμα μπορούν δυνητικά να έχουν κάποιο GameEntity (η παραπάνω ιεραρχία μας επιτρέπει να αναφερθούμε στον Avatar, τα Werewolves και τα Vampires με το ίδιο όνομα της parent κλάσης τους) μέσα τους (occupant). Έτσι προσφέρονται οι συναρτήσεις Get και SetOccupant(). Όμως, σύμφωνα με την εκφώνηση, δεν επιτρέπεται σε όλα τα κελιά να πατήσει κάποιος GameEntity. Επομένως η MapElement προσφέρει ένα pure virtual function CanBeOccupied(), έτσι ώστε ο κάθε τύπος κελιού (Ground, Tree, Water) που κληρονομεί από αυτή να μπορεί να αποφασίσει ο ίδιος αν θα δεχτεί κάποια οντότητα μέσα του. Για παράδειγμα ένα δέντρο δεν θα δεχτεί ποτέ παίκτη πάνω του, ενώ για το έδαφος αυτό εξαρτάται από το αν έχει ήδη κάποιον άλλον παίκτη πάνω του ή οχι. Αντίστοιχα προσφέρεται και η pure virtual function Print() η οποία αναλαμβάνει να εκτυπώσει κατάλληλα το κάθε κελί στην οθόνη, ανάλογα με τον τύπο κελιού στο οποίο αναφέρεται.



Η κλάση Map έχει υλοποιηθεί για να κρύψει όλες τις σχετικές πληροφορίες σχετικά με τον κόσμο πίσω από ενα καθαρό interface. Ο constructor της καλείται με τις ζητούμενες διαστάσεις και δημιουργεί έναν κόσμο με τυχαία τοποθέτηση νερού και δέντρων. Ένας 2D πίνακας από δείκτες σε MapElement αξιοποιείται για να αποθηκευτεί ο συγκεκριμένος κόσμος στη μνήμη. Μια βασική λειτουργία που προσφέρεται είναι η GetNeighboringCells() που δίνει μια λίστα με τα γειτονικά κελιά από το κελί με τις δοθείσες συντεταγμένες. Αντίστοιχα, υπάρχει και η συνάρτηση GetDiagonalNeighboringCells(), καθώς και η Show που εκτυπώνει όλο το Map.

Όλες οι πληροφορίες και οι λειτουργίες του παιχνιδιού, όπως το game loop, το framerate, το αν είναι μέρα ή νύκτα και το ποια Game Entities υπάρχουν αυτή τη στιγμή στο παιχνίδι, κρύβονται πίσω από την κλάση Game η οποία προσφέρει τη βασική λειτουργία Run(), που είναι το μόνο που χρειάζεται να κάνει η main για να τρέξει ένα παιχνίδι. Σε κάθε iteration του game loop το πρόγραμμα διαβάζει την είσοδο, εκτελεί τις λειτουργίες που ζήτησε ο χρήστης, διατρέχει όλα τα entities και τους ζητάει να κουνηθούν (Update()) και στο τέλος ζητάει από το Map να εκτυπωθεί στην οθόνη. Τα παραπάνω επαναλαμβάνονται μέχρι το παιχνίδι να τελειώσει ή ο χρήστης να ζητήσει να σταματήσει.

Η κλάση Utils προσφέρει μερικές static λειτουργίες χρήσιμες σε διάφορα σημεία του προγράμματος, όπως η παραγωγή τυχαίων αριθμών και η ανάγνωση αριθμού από την είσοδο.

Class Dependencies:

Η κλάση Game κρατάει έναν δείκτη στο αντικείμενο του Map προκείμενου να μπορεί να του ζητήσει πληροφορίες. Επίσης, κρατάει μια λίστα με όλα τα ζωντανά Game Entities ανά πάσα στιγμή, για να μπορεί να τους ζητήσει να ανανεωθούν μέσω της Update() σε κάθε γύρο του παιχνιδιού.

Το κάθε Game Entity κρατάει ένα δείκτη στο παιχνίδι στο οποίο ανήκει για να μπορεί να το ρωτήσει ποιοι είναι οι γείτονές του (το Game μετά ρωτάει το Map και μεταβιβάζει το αποτέλεσμα) έτσι ώστε να αποφασίσει τι θα κάνει σε κάθε κίνηση ανάλογα με το τι γίνεται γύρω του, καθώς και για να το ενημερώσει όταν πεθάνει (και το game να ενημερώσει την λίστα με τις ενεργές οντότητες κατάλληλα). Επίσης, ανά πάσα στιγμή κρατάει μια αναφορά στο κελί στο οποίο βρίσκεται για να μπορεί να σβήσει τον εαυτό του από εκεί όταν μετακινηθεί αλλού.

Να σημειωθεί, τέλος, ότι επειδή η κίνηση του avatar είναι event based (ανάλογα με το πότε ο χρήστης θα πατήσει ένα κουμπί) η κίνηση του δεν πραγματοποιείται στην συνάρτηση Update() (η οποία παραμένει κενή), αλλά με ξεχωριστές συναρτήσεις για κίνηση πάνω/κάτω/δεξιά/αριστερά. Αν ο avatar κινούταν αυτόνομα, τότε προφανώς η λογική της κίνησής του θα έμπαινει στην Update() του.