Πανεπιστήμιο Πειραιώς

Τμήμα Πληροφορικής

Τεχνολογίες Ανάπτυξης Ηλεκτρονικών Παιχνιδιών

Ανάπτυξη 2d παιχνιδιού με SFML

-

Creating a 2d game with SFML

ΌΝΟΜΑ: Αριστοτέλης Ματακιάς - Α.Μ: Π19100

Email Επικοινωνίας: mataktelis01@gmail.com

Επιβλέπων καθηγητής: Θεμιστοκλής Παναγιωτόπουλος

Ιανουάριος 2023

Περιεχόμενα

1 Εισαγωγή **2**

2 Περιγραφή του προβλήματος **3**
2.1 Κριτήρια της εκφώνησης 3
2.2 Παρατηρήσεις 4

3 Ανάλυση **4**

4 Σχεδίαση **5**

5 Υλοποίηση **5**
5.1 Προ-επισκόπηση 5
5.2 Αρχείο CMakeLists.txt και Αρχειοθέτηση του project 6
5.2.1 Βασικοί ορισμοί 6
5.3 Διαχείριση εισόδου από τον χρήστη 7
5.4 Καταστάσεις του προγράμματος 7
5.5 Κλάση Map 8
5.6 Δημιουργία Χαρτών 8
5.7 Δημιουργία Εχθρών 8

6 Εκτέλεση **8**

7 Κάλυψη κριτηρίων **8**

8 Συμπεράσματα και παρατηρήσεις **8**

9 Βιβλιογραφία **8**

1 Εισαγωγή

Το παρόν αρχείο αποτελεί την τεκμηρίωση για την εργασία του μαθήματος **Τεχνολογίες Ανάπτυξης Ηλεκτρονικών Παιχνιδιών**. Στόχος της εργασίας είναι η ανάπτυξη ενός βιντεοπαιχνιδιού. Η γλώσσα προγραμματισμού που επιλέχθηκε είναι η **C++**. Περισσότερες λεπτομέριες για την υλοποίηση θα παρουσιαστούν σε επόμενες ενότητες.

Ο λόγος για τον οποίο δεν χρησιμοποιήθηκαν τα εργαλεία που παρουσιάστηκαν στο μάθημα (C# και Unity) είναι ότι παρόλο που επιτρέπουν στον χρήστη να δημιουργήσει εύκολα και γρήγορα ένα παιχνίδι, ο χρήστης δεν γνωρίζει ακριβώς πώς το παιχνίδι λειτουργεί. Μηχανές όπως η Unity παρέχουν ένα έτοιμο περιβάλλον ανάπτυξης πάνω στο οποίο ο χρήστης θα αναπτύξει αυτό που θέλει. Το περιβάλλον αυτό διαχειρίζεται όλες τις λειτουργίες του προγράμματος και απαλλάσσει τον χρήστη από πολλές λεπτομέρειες υλοποίησης, οι οποίες για κάποιους θεωρούνται δύσκολες. Αντίθετα όταν ο ίδιος ο χρήστης δημιουργεί το περιβάλλον, κάτι που μπορεί να επιτευχθεί με γλώσσες χαμηλότερου επιπέδου όπως η C και η C++, μπορεί να το προσαρμόσει ακριβώς στις ανάγκες του. Σαφώς αυτό σημαίνει ότι απαιτείται περισσότερος χρόνος για την ανάπτυξη του παιχνιδιού, καθώς πρώτα πρέπει να φτιαχτεί το περιβάλλον, στο οποίο συνήθως αναφερόμαστε ως μηχανή (engine) του παιχνιδιού. Μπορεί η δημιουργία μίας μηχανής από το μηδέν να ακούγεται ως μία δύσκολη και επίπονη δουλεία αλλά στη πραγματικότητα είναι μία αρκετά δημιουργική διαδικασία για κάθε προγραμματιστή που ασχολείται είτε επαγγελματικά είτε ερασιτεχνικά με την ανάπτυξη βιντεοπαιχνιδιών.

Η εργασία εστιάζει κυρίως στην Υλοποίηση του παιχνιδιού, το οποίο είναι 2D. Επιπλέον οι βιβλιοθήκες και τα περιεχόμενα (Assets) του παιχνιδιού είναι όλα δωρεάν. Ο πηγαίος κώδικας του παιχνιδιού βρίσκεται στα παραδοτέα της εργασίας αλλά και στο Github. Είναι σημαντικό να αναφερθεί ότι ο κώδικας του παιχνιδιού βασίστηκε στο βιβλίο **SFML Game Development By Example** του οποίου ο κώδικας είναι διαθέσιμος και αποτελεί έναν πολύ καλό οδηγό για εισαγωγή στον προγραμματισμό βιντεοπαιχνιδιών. Η εργασία βασίστηκε στο κεφάλαιο 7 του βιβλίου στο οποίο ενώ δεν περιέχονται όλα σχεδιαστικά πρότυπα (design patterns) του βιβλίου, εξακολουθεί να είναι επαρκές για τη δημιουργία ενός ολοκληρωμένου 2D παιχνιδιού.

2 Περιγραφή του προβλήματος

2.1 Κριτήρια της εκφώνησης

Σκοπός της εργασίας είναι η δημιουργία ενός βιντεοπαιχνιδιού και δίνονται τα παρακάτω κριτήρια τα οποία θα πρέπει στην ιδανική περίπτωση να πληρούνται:

- **Αληθοφάνεια** Ο χώρος σας θα πρέπει να "πείθει" τον χρήστη, να μη δημιουργεί την εντύπωση πως παραβιάζει τους φυσικούς νόμους και, γενικά, να παρέχει τη ζητούμενη εμπειρία χωρίς να βασίζεται σε σχεδιαστικές ή αισθητικές ακρότητες.

- **Περιεχόμενο** Θα χρησιμοποιήσετε περιεχόμενο που θα θυμίζει τον φυσικό, "πραγματικό" κόσμο για το σύνολο του οποίου θα αναφέρετε πηγές (μέσα στον εικονικό κόσμο, χρησιμοποιώντας κατάλληλα μέσα απεικόνισης πληροφορίας).

- **Πληρότητα** Ο χώρος θα είναι ολοκληρωμένος σαν συστατικό εφαρμογής εικονικής πραγματικότητας: θα εμφανίζεται στο χρήστη ως ένας τρισδιάστατος κόσμος, θα έχει λειτουργικότητα σε διάφορα σημεία ανάλογα με το τι αναπαριστά και το τι εξυπηρετεί, θα εξελίσσεται με το πέρασμα του χρόνου χάρη σε κινούμενα στοιχεία και στοιχεία με λειτουργικότητα, θα έχει φωτισμό και άλλα διακοσμητικά στοιχεία, και γενικά θα παρέχει μία ολοκληρωμένη, πολυτροπική, δυναμική εμπειρία αναπαράστασης χωρίς αδικαιολόγητα κενά και σημεία ασυνέχειας ή ασυνέπειας.

- **Σχεδιασμός** Ο σχεδιασμός του χώρου σας θα χαρακτηρίζεται από ακρίβεια και λεπτομέρεια. Η αυξημένη σχεδιαστική/δομική πολυπλοκότητα δεν αποτελεί στοιχείο το οποίο θα αξιολογηθεί απαραίτητα θετικά.

- **Αισθητική** Από άποψη αισθητικής, ο χώρος σας θα πρέπει να είναι ευπαρουσίαστος και να μην χαρακτηρίζεται από αισθητικές ακρότητες. Επίσης, θα πρέπει να “προσκαλεί” τον χρήση και να του κεντρίζει την προσοχή.

- **Πρωτοτυπία** Η δομή, ο σχεδιασμός, η αισθητική και η λειτουργικότητα του χώρου σας θα αντανακλούν την έκταση στην οποία διερευνήσατε τις σχεδιαστικές δυνατότητες της πλατφόρμας, τις δυνατότητες της οικείας γλώσσας, καθώς και διάφορα σχεδιαστικά πρότυπα και πρακτικές υλοποίησης.

- **Χρηστικότητα** Ο χώρος θα προορίζεται για χρήση από ανθρώπους και για την κάλυψη υπαρκτών αναγκών τους. (π.χ. δυνατότητα μετακίνησης και επίσκεψης διαφόρων σημείων του τρισδιάστατου κόσμου)

- **Κίνηση (animation)** Η κατασκευή σας θα περιέχει αναπαραστάσεις κινούμενων στοιχείων, ώστε να δημιουργεί στον χρήστη την εντύπωση ότι συμμετέχει σε ένα δυναμικό, εξελισσόμενο με το χρόνο, κόσμο. Τέτοια στοιχεία μπορεί να είναι, για παράδειγμα, "άψυχα" αντικείμενα, αντικείμενα που μετακινούνται σε προκαθορισμένες διαδρομές, ζώα/ρομπότ, κ.ά. Τονίζεται ότι ο όρος κίνηση χρησιμοποιείται εδώ όχι με την έννοια της μετακίνησης, αλλά της οποιασδήποτε μεταβολής της τιμής κάποιου χαρακτηριστικού ενός αντικειμένου. Έτσι, ως κινούμενο συστατικό νοείται και ένα αντικείμενο που αλλάζει χρώμα με την πάροδο του χρόνου (π.χ., ένα φανάρι ρύθμισης κυκλοφορίας, κ.ά.), ένα αντικείμενο που αλλάζει διαστάσεις, κ.ά.

- **Λειτουργικότητα (functionality)** Ανάλογα με το τι αναπαριστά, η κατασκευή σας θα περιέχει συστατικά με λειτουργικότητα, δηλαδή, συστατικά πάνω στα οποία θα μπορεί να επιδράσει ο χρήστης ή, γενικότερα, συστατικά τα οποία αντιδρούν με συγκεκριμένο τρόπο σε συγκεκριμένες ενέργειες του χρήστη. Παραδείγματα αντικειμένων

με λειτουργικότητα είναι μία πόρτα η οποία ανοίγει και κλείνει όταν ο χρήστης επιδρά σε ένα διακόπτη, ένας μηχανισμός που ξεκινά και σταματά όταν ο χρήστης επιδρά σε ένα μοχλό, ένα ρομπότ ή ένας συναγερμός που ενεργοποιείται όταν ο χρήστης πλησιάσει κάποιο σημείο του εικονικού κόσμου ή βρεθεί μέσα σε κάποια περιοχή του, ένα ραδιόφωνο του οποίου την ένταση ο χρήστης προσαρμόζει με κάποιο χειριστήριο, μία συσκευή τηλεόρασης την οποία ο χρήστης ενεργοποιεί και απενεργοποιεί, κ.ά.

- **Ανάπτυξη** Η λειτουργικότητα των διαφόρων αντικειμένων θα υλοποιηθεί στην οικεία γλώσσα (C#/Unity 3D). Ο κώδικας που θα συντάξετε θα είναι καλά σχεδιασμένος, θα ενσωματώνει βέλτιστες τεχνικές υλοποίησης, θα είναι ευπαρουσίαστος και, το σημαντικότερο, θα είναι συνολικά, αναλυτικά και κατανοητά σχολιασμένος.

2.2 Παρατηρήσεις

Το παιχνίδι που θα παρουσιαστεί θα είναι δισδιάστατο (2D) και όχι τρισδιάστατο (3D) όπως αναφέρεται στα κριτήρια, χωρίς να σημαίνει αυτό ότι τα κριτήρια αλλάζουν σημαντικά. Πιο συγκεκριμένα το παιχνίδι θα είναι ένα απλό 2D platformer συγκρίσιμο σε μεγάλο βαθμό με το κλασσικό παιχνίδι **Super Mario Bros**. (Εικόνα 1).

Εικόνα 1: Super Mario Bros. (NES)

Ο παίκτης θα μετακινεί με τα τυπικά πλήκτρα **W**, **A**, **S**, **D** έναν χαρακτήρα σε ένα περιβάλλον φτιαγμένο από **tiles**. Στο περιβάλλον αυτό θα υπάρχουν "εχθροί" τους οποίους θα μπορεί ο παίκτης να προσπερνά ή να πολεμάει. Σκοπός του παίκτη είναι να φτάσει στο τέλους του χάρτη.

3 Ανάλυση

Στα πλαίσια της εργασίας είχαν οριστεί οι παρακάτω στόχοι:

- **Χρήση δωρέαν και Open Source περιβάλλοντος για την ανάπτυξη του παιχνιδιού** Η γλώσσα προγραμματισμού είναι η **C++** και βασικό ρόλο στην ανάπτυξη του παιχνιδιού έπαιξε η βιβλιοθήκη **SFML**. Τα αρχικά σημαίνουν **Simple and Fast Multimedia**

Library και είναι μια cross-platform βιβλιοθήκη που παρέχει απλά API για τη δημιουργία παιχνιδιών και εφαρμογών. Η SFML μπορεί να χρησιμοποιηθεί τόσο για την ανάπτυξη μεγάλης κλίμακας εμπορικών παιχνιδιών, όσο και για μικρότερες ομάδες και χομπίστες που είναι αρχάριοι με την C++.

- **Χρήση από δωρεάν assets**
  Τα assets είναι ένας ευρύς όρος που χρησιμοποιείται για να περιγράψει κάθε μεμονωμένο στοιχείο που υπάρχει σε ένα παιχνίδι. Αυτά μπορεί να περιλαμβάνουν έργα τέχνης, τρισδιάστατα μοντέλα, GUIs, ειδικά, μουσική, ακόμη και scripts για πράγματα όπως η φυσική.

  Η δημιουργία game assets αποτελεί ένα έργο σχεδίασης. Ιδανικά, ο δημιουργός ενός παιχνιδιού φτιάχνει ο ίδιος τα assets για το παιχνίδι του, ή στη περίπτωση μίας μεγαλύτερης ομάδας που ασχολείται με την ανάπτυξη ενός παιχνιδιού, θα υπάρχουν άτομα που θα εξειδικεύονται αποκλειστικά σε τέτοια θέματα σχεδίασης. Τα assets είναι αυτά που φαίνονται άμεσα σε αυτόν που βλέπει το παιχνίδι και για πολλούς θεωρείται πώς είναι στοιχειώδης μέρος του ίδιου του παιχνιδιού.

- **Δημιουργεία εκτελέσιμου αρχείου για πολλές πλατφόρμες (τουλάχιστον GNU/Linux και Windows)**

4 Σχεδίαση

5 Υλοποίηση

5.1 Προ-επισκόπηση

Ο πηγαίος κώδικας του προγράμματος αποτελείται από πολλά ξεχωριστά αρχεία. Όπως έχει αναφερθεί στην εισαγωγή, ο κώδικας βασίζεται στον κεφάλαιο 7 του βιβλίου SFML Game development by example. Ο κώδικας του βιβλίου παρέχει μία καλή αρχή για τη δημιουργία ενός παιχνιδιού, έχοντας ορίσει αρκετές βασικές κλάσεις. Ο κώδικας αυτός αξιοποιήθηκε για την δημιουργία ενός πιο ολοκληρωμένου παιχνιδιού. Πιο συγκεκριμένα έγιναν οι παρακάτω προσθήκες:

- Προσθήκη ηχητικών εφέ και μουσικής
- Υλοποίηση ενός απλού συστήματος pick up αντικειμένων για τον παίκτη.
- Προσθήκη HUD με ενδείξεις healthbar και αντικείμενα που έχουν συλλεχθεί από τον παίκτη
- Δημιουργία Background Layer στους χάρτες
- Ορισμός ενός πρακτικού συστήματος για την εύκολη δημιουργία χαρτών
- Δημιουργία περισσότερων εχθρών
- Εισαγωγή ενός μεγαλύτερου tileset και αλλαγή του spritesheet για τον παίκτη
- Διόρθωση διαφόρων bugs στον πρωτογεννή κώδικα
- Δημιουργία ενός CMakeLists.txt αρχείου (θα εξηγηθεί αναλυτικά σε επόμενη ενότητα)

Είναι σαφές ότι ο τελικός πηγαίος κώδικας είναι αρκετά μεγάλος και δεν θα εξηγηθεί σε όλη του την έκταση σε αυτό το αρχείο. Αντίθετα έχουν επιλεχθεί κατάλληλα μέρη τα οποία αξίζουν μία επεξήγηση και θα παρουσιαστούν σε επόμενες ενότητες.

Πριν όμως περάσουμε σε εξήγηση του κώδικα πρέπει να αναφερθούμε στην αρχειοθέτηση του project. Ενδεχομένως έμπειροι προγραμματιστές της C++ να έχουν ήδη καταλάβει την αρχειοθέτηση, αλλά για λόγους πληρότητας θα δοθεί μία αναλυτική περιγραφή στην ενότητα που ακολουθεί.

5.2 Αρχείο CMakeLists.txt και Αρχειοθέτηση του project

5.2.1 Βασικοί ορισμοί

Ο πηγαίος κώδικας του project αποτελείται από πολλές κλάσεις. Είναι σημαντικό να εξηγηθεί πως κάθε κλάση εμφανίζεται στον κώδικα σε δύο αρχεία:

- Το **header file**, το οποίο έχει κατάληξη **.h** και περιέχει τον ορισμό (definition) της κλάσης και των συναρτήσεών της.
- Το αρχείο που περίεχει την υλοποίηση της κλάσης (implementation) και έχει κατάληξη **.cpp**.

Με αυτόν τον τρόπο, αν η υλοποίηση μίας κλάσης δεν αλλάξει, δεν χρειάζεται να μεταγλωττιστεί ξανά. Συνήθως τα IDEs θα το κάνουν αυτό αυτόματα, δηλαδή θα μεταγλωττίσουν μόνο τις κλάσεις που έχουν αλλάξει στο πρόγραμμα. Αυτό δεν θα ήταν δυνατό αν για μία κλάση αν αποτελούταν μόνο από ένα αρχείο ή αν υπήρχε υλοποίηση στο header file. Παρακάτω δίνεται ένα απλό παράδειγμα:

- Αρχείο **Num.h**

```
class Num {
private:
        int num;

public:
        Num();
        Num(int n);
        int getNum();
};
```

- Αρχείο **Num.cpp**

```
#include "Num.h"

Num::Num() : num(0) { } // constructor definition:
                        // ": num(0)" is the initializer list
                        // "{ }" is the function body

Num::Num(int n) : num(n) { } // similarly ": num(n)" is the initializer list

int Num::getNum() {
        return num;
}
```

- Αρχείο **main.cpp**

```
#include <iostream>
#include "Num.h"

int main()
{
        Num n(35);
        std::cout << n.getNum() << std::endl;
        return 0;
}
```

Για να κάνουμε compile αρκεί να δώσουμε στον μεταγλωττιστή g++ τα αρχεία main.cpp και Num.cpp με την παρακάτω εντολή:

```
$ g++ main.cpp Num.cpp
```

Όπως βλέπουμε στα παραπάνω αρχεία, κάθε .cpp αρχείο μπορεί να χρησιμοποιήσει ένα header αρχείο μέσω ενός include statement (γραμμή 1 στο Num.cpp και γραμμή 2 στο main.cpp). Γενικά τα header files περιέχουν ορισμούς συναρτήσεων και μέσω της δήλωσης #include ο μεταγλωττιστής ενημερώνεται πως πρέπει πρώτα να μεταγλωττιστούν οι συναρτήσεις που είναι καταγεγραμμένες στο header file και στη συνέχεια θα μεταγλωττίσει το τωρινό αρχείο.
Υπάρχουν δύο ειδών header files:

- **Standard library header files** (όπως το iostream στη γραμμή 1 του αρχείου **main.cpp**)
  Τα αρχεία αυτά περιέχονται ήδη στον g++ μεταγλωττιστή. Παρόμοια βιβλιοθήκες που έχουν εγκατασταθεί, όπως και η SFML μπορούν να κληθούν με τον ίδιο τρόπο εφόσον έχουν οριστεί στο PATH του υπολογιστή.

- **User-defined header files** (όπως το Num.h του παραδείγματος)
  Αυτά τα header files έχουν δημιουργηθεί από τον χρήστη. Μπορούμε να τα ξεχωρίσουμε καθώς στο include statement το όνομά τους είναι μέσα σε double quotes και όχι μέσα σε "<" και ">".

Πριν προχωρήσουμε καλό είναι να επισημάνουμε τα **initializer list** που αναφέρονται και στα σχόλια του παραδείγματος, καθώς χρησιμοποιούνται αρκετά στον κώδικα του project. Πιο συγκεκριμένα, το **initializer list** χρησιμοποιείται για να αρχικοποιήσει πεδία μίας κλάσης. Δηλώνεται μετά το όνομα του constructor και των παραμέτρων του.

```
EntityBase::EntityBase(EntityManager* l_entityMgr)
        :m_entityManager(l_entityMgr), m_name("BaseEntity"),
        m_type(EntityType::Base), m_id(0), m_referenceTile(nullptr),
        m_state(EntityState::Idle), m_collidingOnX(false), m_collidingOnY(false){}
```

Ο πηγαίος κώδικας βρίσκεται στον φάκελο **Simple-SFML-2D-Game** και αποτελείται από **.cpp** αρχεία και **.h** αρχεία. Τα **.cpp** αρχεία βρίσκονται στον φάκελο src και τα **.h** αρχεία στον φάκελο **include** αντίστοιχα.

5.3 Διαχείριση εισόδου από τον χρήστη

5.4 Καταστάσεις του προγράμματος

Ένα βιντεοπαιχνίδι δεν αποτελείται μόνο από γραφικά. Είναι πολύ συνηθισμένο σήμερα ένα παιχνίδι να έχει μια εισαγωγή πριν την έναρξή του, η οποία μάλιστα θα συνοδεύεται από κάποιο animation. Επίσης είναι αναγκαίο να ύπαρχει και ένα μενού από το οποίο ο χρήστης θα ξεκινάει το παιχνίδι ή θα επιλέγει ανάμεσα σε ορισμένες ρυθμίσεις ή θα κλείνει την εφαρμογή. Είναι σημαντικό επίσης το παιχνίδι να έχει τη δυνατότητα να διακόπτεται (pause) από τον χρήστη. Όλες αυτές οι λειτουργικότητες πρέπει να ληφθούν υπ'όψη από τα πρώτα στάδια ανάπτυξης ενός παιχνιδιού. Ο τρόπος με τον οποίο θα αντιμετωπιστούν αυτές οι απαιτήσεις είναι μέσω της υλοποίησης των **καταστάσεων**.

Πριν συνεχίσουμε πρέπει να εξηγηθεί ακριβώς η έννοια της **κατάστασης** σε ένα παιχνίδι. Πρόκειται για ένα από τα πολλά επίπεδα (layers) που μπορεί να έχει ένα παιχνίδι, όπως το κεντρικό μενού, το animation που εμφανίζεται πριν το κεντρικό μενού και τέλος το ίδιο το παιχνίδι. Κάθε ένα από αυτά τα επίπεδα έχει τον δικό του τρόπο να ενημερώνεται και να κάνει render τα περιεχόμενά του στην οθόνη.

Το βιβλίο SFML Game Development by example προτείνει μία υλοποίηση αρκετά καλή για τα πλαίσια της εργασίας και έχει ενσωματωθεί στον πηγαίο κώδικα. Βασικό στοιχείο της υλοποίησης είναι η **δημιουργία ξεχωριστών κλάσεων** για τη κάθε κατάσταση. Όλες οι κλάσεις θα έχουν ορισμένες κοινές μεθόδους για να ενημερώνονται και να κάνουν render, κάτι που επιτυγχάνεται με την **κληρονομικότητα**. Παρακάτω ορίζουμε την κλάση BaseState, από την οποία θα κληρωνομούν όλες οι κλάσεις που αντιστοιχούν σε μία κατάσταση του παιχνιδιού.

Επειδή είναι αναγκαίο κάθε κλάση που κληρωνομεί από την BaseState να έχει υλοποιήσει τις μεθόδους της, όλες οι μέθοδοι δηλώνονται ως **purely virtual** (έχουν το keyword 'virtual' και στο τέλος '=0').

5.5 Κλάση Map

5.6 Δημιουργία Χαρτών

5.7 Δημιουργία Εχθρών

6 Εκτέλεση

7 Κάλυψη κριτηρίων

8 Συμπεράσματα και παρατηρήσεις

9 Βιβλιογραφία