ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

Τμήμα Πληροφορικής



Εργασία Μαθήματος Τεχνολογίες Ανάπτυξης Ηλεκτρονικών Παιχνιδιών

|  |  |
| --- | --- |
| ***Τίτλος εργασίας*** | ***Αναφορά Εργασίας Συστήματα Διαχείρισης Βάσεων Δεδομένων*** |
| Όνομα φοιτητή - ΑΜ | Ευστάθιος Αρμαγανίδης - Π16008  Κωνσταντίνος-Ζώης Σύριος - Π16139  Νικόλαος Τόλης - Π16146 |
| Ημερομηνία παράδοσης | 30/03/2020 |

Περιεχόμενα

[Γενική Περιγραφή 4](#__RefHeading___Toc1461_586586232)

[Περιβάλλον Ανάπτυξης 4](#__RefHeading___Toc1463_586586232)

[Βάση Δεδομένων 4](#__RefHeading___Toc1465_586586232)

[Εγκατάσταση 4](#__RefHeading___Toc1467_586586232)

[Διαχείριση Βάσης Δεδομένων 5](#__RefHeading___Toc1469_586586232)

[Άνοιγμα βάσης δεδομένων 5](#__RefHeading___Toc1471_586586232)

[Εκτέλεση Εντολής 5](#__RefHeading___Toc1473_586586232)

[Δημιουργία Νέου Πίνακα 5](#__RefHeading___Toc1475_586586232)

[Εισαγωγή Δεδομένων 6](#__RefHeading___Toc1477_586586232)

[Επεξεργασία Δεδομένων 6](#__RefHeading___Toc1479_586586232)

[DatabaseHelper.cs 6](#__RefHeading___Toc1481_586586232)

[Μέθοδοι 6](#__RefHeading___Toc1483_586586232)

[Κλάσεις του namespace Assets.Combat 7](#__RefHeading___Toc1485_586586232)

[Die.cs 7](#__RefHeading___Toc1487_586586232)

[Constructor 7](#__RefHeading___Toc1489_586586232)

[int RollDie() 7](#__RefHeading___Toc1491_586586232)

[CombatLog.cs 7](#__RefHeading___Toc1493_586586232)

[Character.cs 7](#__RefHeading___Toc1495_586586232)

[bool Move(double) 7](#__RefHeading___Toc1497_586586232)

[void Attack(Weapon, Character) 7](#__RefHeading___Toc1499_586586232)

[HitPoints.cs 8](#__RefHeading___Toc1501_586586232)

[void Damage(int) 8](#__RefHeading___Toc1503_586586232)

[void Heal(int) 8](#__RefHeading___Toc1505_586586232)

[void Rest() 8](#__RefHeading___Toc1507_586586232)

[AbilityScores.cs 8](#__RefHeading___Toc1509_586586232)

[void Increase(string) 8](#__RefHeading___Toc1511_586586232)

[int GetAbilityModifier(string) 8](#__RefHeading___Toc1513_586586232)

[Class.cs 9](#__RefHeading___Toc1515_586586232)

[Armor.cs 9](#__RefHeading___Toc1517_586586232)

[Weapons.cs 9](#__RefHeading___Toc1519_586586232)

[CombatManager.cs 9](#__RefHeading___Toc1521_586586232)

[void Awake() 9](#__RefHeading___Toc1523_586586232)

[void Start() 9](#__RefHeading___Toc1525_586586232)

[void NextActiveCharacter() 10](#__RefHeading___Toc1527_586586232)

[void CheckForCombatEnd() 10](#__RefHeading___Toc1529_586586232)

[void EndCombat(bool) 10](#__RefHeading___Toc1531_586586232)

[void StartCombat() 10](#__RefHeading___Toc1533_586586232)

[void EndActiveCharacterTurn() 10](#__RefHeading___Toc1535_586586232)

[bool ActiveCharacterMoves(double) 10](#__RefHeading___Toc1537_586586232)

[void ActiveCharacterAttacks(Weapon, Character) 10](#__RefHeading___Toc1539_586586232)

[void OtherSourceDamage(Character, object) 10](#__RefHeading___Toc1541_586586232)

[Παράδειγμα Εκτέλεσης 11](#__RefHeading___Toc1543_586586232)

[Start() 11](#__RefHeading___Toc1545_586586232)

[Update() 12](#__RefHeading___Toc1547_586586232)

[Αποτελέσματα 13](#__RefHeading___Toc1549_586586232)

# Γενική Περιγραφή

*Το παρόν έγγραφο είναι ακριβές για την έκδοση 0.1 του Combat System.*

Σκοπός της εργασίας μας ήταν η ανάπτυξη ενός συστήματος μάχης, βασισμένο στους κανόνες Dungeons & Dragons™ (5th edition) και με επιρροές από το αρχικό Baldur’s Gate.

Τα βασικά συστατικά που υλοποιήθηκαν είναι τα εξής:

* Σύστημα για την διαχείριση της μάχης (Combat Manager)
* Εμφάνιση κατάλληλων μηνυμάτων (Combat Log)
* Πλήρως ή μερικώς υλοποιημένες κλάσεις για τα στατιστικά και τις ενέργειες των χαρακτήρων.
* Δημιουργία βάσης δεδομένων η οποία περιλαμβάνει τα δεδομένα των όπλων, των πανοπλιών, των φυλών και των γλωσσών.

# Περιβάλλον Ανάπτυξης

Για την ανάπτυξη της εργασίας χρησιμοποιήθηκαν τα εξής εργαλεία:

* Unity Game Engine (<https://unity.com/>), έκδοση 2019.2.14f1
* Visual Studio 2019 (<https://visualstudio.microsoft.com/>)
* Ως βάση δεδομένων χρησιμοποιήθηκε η SQLite, χάρη στο plugin για την Unity από <https://github.com/rizasif/sqlite-unity-plugin>
* Για την διαχείριση της βάσης δεδομένων, το SQLite Administrator (<http://sqliteadmin.orbmu2k.de/>)
* Για τη δημιουργία διαγραμμάτων, το draw.io (<http://draw.io/>)

# Βάση Δεδομένων

## Εγκατάσταση

Για την εγκατάσταση του SQLite plugin της Unity η διαδικασία έχει ως εξής:

* Λήψη του plugin από το GitHub
* Συμπερίληψη του στο project, μεταφέροντας όλα τα αρχεία του στον φάκελο “Assets\Dataset\Plugins”
* Για την χρήση σε κάποιο C# αρχείο, χρειάζονται τα ακόλουθα using directives:
  + using UnityEngine;
  + using System.Data;
  + using Mono.Data.Sqlite;
  + using System.IO;

Για την εγκατάσταση του SQLite Administrator η διαδικασία έχει ως εξής:

* Λήψη του αρχείου από την ιστοσελίδα (<http://sqliteadmin.orbmu2k.de/>)
* Αποσυμπίεση του αρχείου σε κάποιο φάκελο (δεν είναι απαραίτητο να βρίσκεται μέσα στο project)

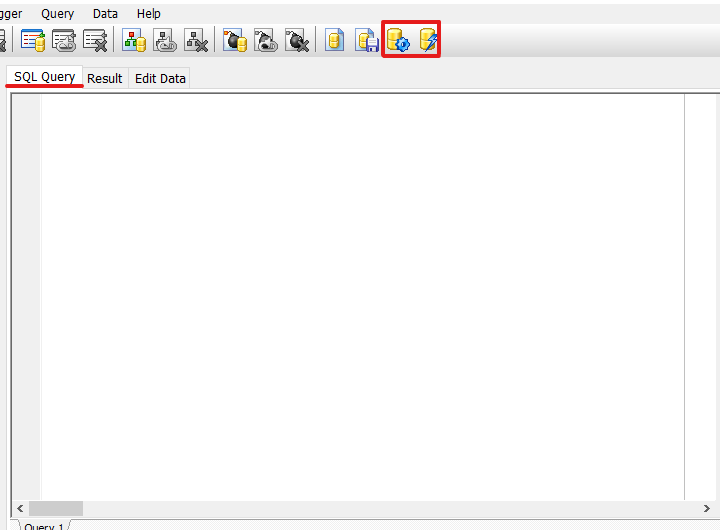
## Διαχείριση Βάσης Δεδομένων

### Άνοιγμα βάσης δεδομένων

1. Εκτέλεση του SQLite Administrator
2. Άνοιγμα της βάσης
3. Εύρεση του SQLite αρχείου στο file system
4. Άνοιγμα του αρχείου

### Εκτέλεση Εντολής

Η εκτέλεση της εκάστοτε εντολής, γίνεται πληκτρολογώντας την εντολή στην καρτέλα “SQL Query” και πατώντας το κουμπί “Execute Query”



### Δημιουργία Νέου Πίνακα

Η δημιουργία νέου πίνακα πραγματοποιείται με την εντολή:

CREATE TABLE table\_name (

column1 datatype,

column2 datatype,

column3 datatype,

....

);

### Εισαγωγή Δεδομένων

Για την εισαγωγή δεδομένων σε υπάρχων πίνακα (π.χ. ένα νέο όπλο στον πίνακα Weapons) απαιτείται εντολή της παρακάτω μορφής:

INSERT INTO Weapons (name, dieDamage, numberOfDice, isMartial, isFinesse, range)

VALUES (‘Dagger’, ‘D4’, 1, 0, 1, 0);

Στη γενική μορφή:

INSERT INTO table\_name (column1, column2, column3, ...)

VALUES (value1, value2, value3, ...);

### Επεξεργασία Δεδομένων

Για την τροποποίηση δεδομένων σε υπάρχων πίνακα απαιτείται εντολή της παρακάτω γενικής μορφής:

UPDATE table\_name

SET column1 = value1, column2 = value2, ...

WHERE condition;

## DatabaseHelper.cs

Η DatabaseHelper είναι η κλάση με την οποία το πρόγραμμα μας θα επικοινωνεί με την βάση δεδομένων και είναι υλοποιημένη ως singleton. Τα παρακάτω using directives είναι απαραίτητα για την λειτουργία του προγράμματος:

* using UnityEngine;
* using System.Data;
* using Mono.Data.Sqlite;
* using System.IO;

Επίσης απαραίτητο είναι το connection string:



*Σημείωση: Μπορεί να χρειαστεί αλλαγή εφόσον η εγκατάσταση της βάσης γίνει σε διαφορετικό path.*

### Μέθοδοι

Οι μέθοδοι που περιλαμβάνονται στο DatabaseHelper δέχονται ως όρισμα το όνομα του επιθυμητού αντικειμένου και επιστρέφουν ένα αντικείμενο της αντίστοιχης κλάσης (π.χ. όρισμα: “Dagger”, αποτέλεσμα: αντικείμενο Weapon).

# Κλάσεις του namespace Assets.Combat

## Die.cs

Μια από τις βασικές κλάσεις είναι αυτή που υλοποιεί τα ζάρια. Αντικείμενα της κλάσης αυτής μπορούν να έχουν την μορφή όλων των απαραίτητων ν-εδρων ζαριών.

### Constructor

Απαραίτητο όρισμα για τη δημιουργία ενός αντικειμένου ζαριού είναι ο τύπος του, τον οποίο λαμβάνουμε από το struct DieType για αποφυγή των λαθών. Με την χρήση ενός στατικού λεξικού γίνεται ανάθεση και της κατάλληλης εμβέλειας του ζαριού.

### int RollDie()

Με αυτή τη μέθοδο γίνεται ρίψη του ζαριού λαμβάνοντας υπόψιν την εμβέλεια του και επιστρέφεται το αποτέλεσμα.

## CombatLog.cs

Η CombatLog είναι μια κλάση η οποία κληρονομεί από το MonoBehaviour της Unity.

Αποτελείται από μια ουρά μηνυμάτων, η οποία είναι στατική ώστε να μπορούν να προστεθούν μηνύματα από οπουδήποτε στον κώδικα.

Τα στιγμιότυπα της κλάσης έχουν τη δυνατότητα σε κάθε frame να εμφανίζουν τα μηνύματα όπου αυτά επιθυμούν.

## Character.cs

Η κλάση Character περιλαμβάνει τα βασικά χαρακτηριστικά ενός DnD χαρακτήρα. Ενδεικτικά αναφέρονται η κλάση, η φυλή, τα hitpoints και τα ability scores. Επίσης, περιλαμβάνει ως μεθόδους τις διαθέσιμες στον χαρακτήρα ενέργειες (π.χ. attack, move). Παρακάτω θα αναλυθούν ορισμένες από τις πιο σύνθετες μεθόδους.

### bool Move(double)

Η μέθοδος αυτή δέχεται ως όρισμα την απόσταση σε units της Unity (μέτρα) και έπειτα από μετατροπή αυτών σε πόδια, γίνεται σύγκριση με την υπολειπόμενη απόσταση που ο χαρακτήρας δύναται να μετακινηθεί. Εάν ο χαρακτήρας μπορεί να μετακινηθεί αφαιρείται η απόσταση από το υπόλοιπό του. Τέλος, επιστρέφεται το αποτέλεσμα της σύγκρισης.

### void Attack(Weapon, Character)

Η μέθοδος Attack δέχεται ως ορίσματα το ενεργό όπλο του χαρακτήρα και τον στόχο της επίθεσης ο οποίος πρέπει να είναι τύπου Character. Στη συνέχεια, λαμβάνοντας υπόψιν το Ability Modifiers και την ιδιότητα του όπλου, υπολογίζεται η επίθεση ως ρίψη ενός D20 + τα αντίστοιχα Ability Modifiers. Το αποτέλεσμα αυτού του αθροίσματος συγκρίνεται με το Armor Class του στόχου και προστίθεται στο message queue το κατάλληλο μήνυμα για το αν η επίθεση ήταν επιτυχής ή ανεπιτυχής.

## HitPoints.cs

Η κλάση HitPoints περιλαμβάνει όλα τα pools των hitpoints (HP), δηλαδή max HP, current HP και temporary HP ενός χαρακτήρα. Παρακάτω ακολουθεί παρουσίαση των βασικών μεθόδων της κλάσης αυτής.

### void Damage(int)

Η μέθοδος αυτή δέχεται ως όρισμα την ποσότητα των πόντων ζημιάς που προκλήθηκαν και την αφαιρεί από το current HP. Σε περίπτωση ύπαρξης temporary HP, εξαντλούνται πρώτα αυτά και στην συνέχεια το υπόλοιπο της ζημιάς (αν υπάρχει) αφαιρείται από το current HP.

### void Heal(int)

Η μέθοδος αυτή δέχεται ως όρισμα την ποσότητα του Heal το οποίο προστίθεται στο current HP, με άνω όριο το max HP.

### void Rest()

Για τις ανάγκες του παιχνιδιού είναι υλοποιημένο μόνο το long rest. Επομένως, η μέθοδος αυτή θέτει το current HP ίσο με το max HP και το temporary HP ίσο με μηδέν (0).

## AbilityScores.cs

Η κλάση αυτή περιλαμβάνει τα βασικά στατιστικά των χαρακτήρων:

* Strength
* Dexterity
* Constitution
* Intelligence
* Wisdom
* Charisma

### void Increase(string)

Η μέθοδος αυτή αυξάνει κατά ένα (1) το Ability Score που δεχόμαστε ως όρισμα. Για αποφυγή των λαθών προτείνεται η χρήση του AbilityType struct.

### int GetAbilityModifier(string)

Σε αυτή τη μέθοδο, υπολογίζεται το Ability Modifier με τον εξής αλγόριθμο:

AbilityModifier = (AbilityScore – 10) / 2

Ο τύπος του επιθυμητού Modifier περνιέται ως όρισμα και για την αποφυγή λαθών προτείνεται η χρήση του AbilityType struct.

## Class.cs

Η κλάση Class περιλαμβάνει πληροφορίες απαραίτητες για την κλάση του χαρακτήρα, όπως επίπεδο, hitDie, proficiencies, speed και saving throws.

Επίσης, στο ίδιο αρχείο περιλαμβάνεται και το struct Alignment το οποίο περιλαμβάνει όλα τα alignments.

## Armor.cs

Η κλάση Armor περιλαμβάνει τις πληροφορίες που είναι απαραίτητες για την λειτουργικότητα των πανοπλιών. Χρησιμοποιείται κυρίως όταν φορτώνουμε δεδομένα πανοπλιών από την βάση δεδομένων και στην συνέχεια γίνεται casting σε κάποια από τις πιο ειδικές κλάσεις (π.χ. Breastplate, ChainShirt, ScaleMail).

## Weapons.cs

Η κλάση Weapons περιλαμβάνει τις πληροφορίες που είναι απαραίτητες για την λειτουργικότητα των όπλων. Χρησιμοποιείται κυρίως όταν φορτώνουμε δεδομένα όπλων από την βάση δεδομένων και στην συνέχεια γίνεται casting σε κάποια από τις πιο ειδικές κλάσεις (π.χ. Dagger, ShortSword, BattleAxe, Morningstar).

## CombatManager.cs

Η CombatManager είναι η κεντρική κλάση για το σύστημα μάχης του παιχνιδιού και κληρονομεί από το MonoBehaviour της Unity. Αποτελείται από ένα combat queue, τον active character και το index του. Η λειτουργικότητα του CombatManager υλοποιείται στις διάφορες μεθόδους που περιλαμβάνει η κλάση.

Στην έναρξη της κάθε μάχης είναι απαραίτητη η δημιουργία ενός καινούριου αντικειμένου CombatManager και η αλλαγή του tag των συμμετεχόντων σε “Combatant”.

*Παρακάτω ακολουθούν οι περιγραφές των μεθόδων που καλούνται εσωτερικά για τη λειτουργία της κλάσης.*

### void Awake()

Στην μέθοδο Awake του CombatManager δημιουργείται στην μνήμη το combat queue, συλλέγουμε όλα τα GameObjects με το tag “Combatant” και προσθέτουμε στο queue τα Character components τους.

### void Start()

Στην μέθοδο Start, η οποία καλείται στο πρώτο frame του αντικειμένου, γίνεται ρίψη για το initiative του κάθε χαρακτήρα που συμμετέχει στη μάχη και ταξινομούνται ανάλογα στο combat queue.

### void NextActiveCharacter()

Η μέθοδος αυτή καλείται όταν πρέπει η μάχη να προχωρήσει στον επόμενο χαρακτήρα και διασχίζει το combat queue μέχρις ότου βρει τον επόμενο μη νεκρό χαρακτήρα.

### void CheckForCombatEnd()

Η μέθοδος αυτή καλείται μετά την εφαρμογή πόντων ζημιάς με σκοπό τον έλεγχο για νέους θανάτους και για το αν έχει λήξη η μάχη. Σε περίπτωση που η μάχη τελείωσε καλείται η συνάρτηση EndCombat.

### void EndCombat(bool)

Η μέθοδος αυτή δεν έχει υλοποιηθεί αλλά έχει δύο μονοπάτια εκτέλεσης:

1. bool == true

Θα εμφανίζεται ανάλογο μήνυμα, θα επανέρχονται τα tags των “Combatants” και θα καταστρέφεται το CombatManager.

1. bool == false

Θα εμφανίζεται παράθυρο διαλόγου για την επαναφόρτωση κάποιου save file.

*Παρακάτω ακολουθούν οι περιγραφές των μεθόδων που καλούνται εξωτερικά λόγω αλληλεπίδρασης του παίκτη με το UI ή απόφαση / ολοκλήρωση κινήσεων του AI.*

### void StartCombat()

Η μέθοδος αυτή καλείται στην αρχή του Combat ώστε να τεθεί ως active character αυτός με το μεγαλύτερο initiative. Επιπρόσθετα, θέτει το speed του χαρακτήρα στην τιμή που πρέπει να έχει για την αρχή του γύρου του.

### void EndActiveCharacterTurn()

Η μέθοδος αυτή καλείται όταν κάποιος χαρακτήρας τελειώνει τον γύρο του ώστε να τεθεί ως active character ο επόμενος μη νεκρός χαρακτήρας στο combat queue. Επιπρόσθετα, θέτει το speed του χαρακτήρα στην τιμή που πρέπει να έχει για την αρχή του γύρου του.

### bool ActiveCharacterMoves(double)

Η μέθοδος αυτή δέχεται ως όρισμα την απόσταση που προσπαθεί να διανύσει ο χαρακτήρας και επιστρέφει το αν μπορεί.

### void ActiveCharacterAttacks(Weapon, Character)

Η μέθοδος αυτή δέχεται ως ορίσματα το ενεργό όπλο και τον στόχο της επίθεσης. Καλεί την ανάλογη συνάρτηση και ελέγχει για το τέλος της μάχης.

### void OtherSourceDamage(Character, object)

Η μέθοδος αυτή δεν έχει υλοποιηθεί αλλά έχει σκοπό την αναπαράσταση της εφαρμογής πόντων ζημιάς με αίτιο κάποιον παράγοντα του περιβάλλοντος όπως παγίδες ή ρούνους.

# Παράδειγμα Εκτέλεσης

Παρακάτω παρουσιάζεται η εκτέλεση ενός παραδείγματος μάχης. Σε μία σκηνή της Unity κατασκευάζεται ένα GameObject που τρέχει το RuntimeExample.cs script.

Πολλές αρχικοποιήσεις αντικειμένων που χρειάζονται για να τρέξει η μάχη, γίνονται hard coded για το συγκεκριμένο παράδειγμα και θα πρέπει να αντικατασταθούν είτε μέσω character creation, save file κλπ.

## Start()

Η Start() περιλαμβάνει τον κώδικα που χρειάζεται για να ξεκινήσει η μάχη. Κατασκευάζεται ένα αντικείμενο CombatLog για τα μηνύματα της μάχης, ορισμένα τυχαία όπλα και χαρακτήρες για να πάρουν μέρος στην μάχη.

Οι χαρακτήρες στη συνέχεια λαμβάνουν το tag “Combatant” ώστε να αναγνωριστούν από το combat manager.

Τέλος, κατασκευάζεται το προαναφερθέν combat manager.



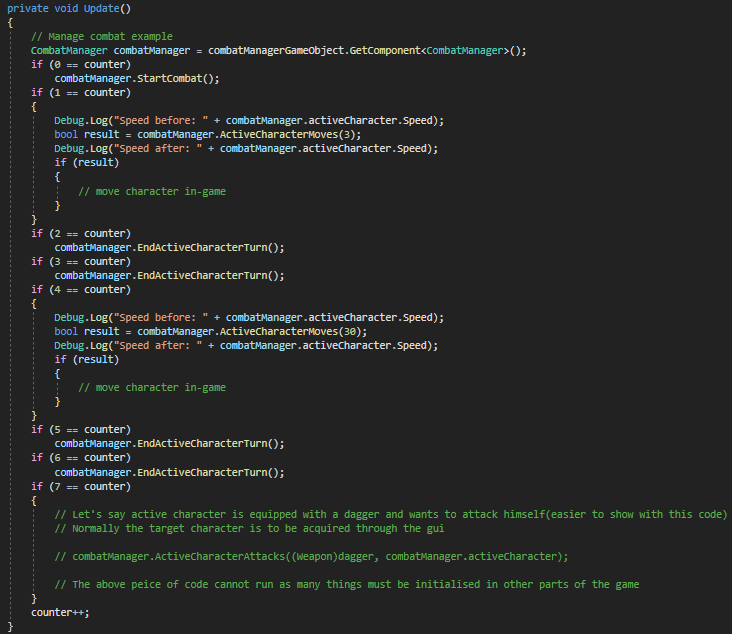
## Update()

Για να γίνει εξομοίωση των frames με εύκολο τρόπο χρησιμοποιείται ένας counter.

Σε κάθε “frame” γίνεται και από ένα βήμα του combat. Θεωρητικά οι παρακάτω συναρτήσεις-βήματα θα πρέπει να καλούνται μετά από αλληλεπίδραση - π.χ του χρήστη μέσω του GUI.

Τα βήματα που παρουσιάζονται είναι:

* Η έναρξη της μάχης
* Προσπάθεια μετακίνησης του χαρακτήρας
* Τέλος γύρου
* Επίθεση χαρακτήρα (το τελευταίο δεν δουλεύει στο παράδειγμα καθώς η αρχικοποίηση που χρειαζόταν ήταν αρκετά σύνθετη – π.χ proficiencies του χαρακτήρα)



## Αποτελέσματα

Τα αποτελέσματα εμφανίζονται στην Debug κονσόλα της unity μέσω του αντικειμένου CombatLog.

