

Ιόνιο Πανεπιστήμιο

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ



Πτυχιακή εργασία

**Υλοποίηση ενός σοβαρού παιχνιδιού για τη διδασκαλία της Βυζαντινής
ιστορίας**

Χρήστος Ταφαρλής

Π2016138

Επιβλέπουσα : Άννα Σωτηροπούλου

Σεπτέμβριος 2022

Επιβλέπουσα

Άννα Σωτηροπούλου, Εργαστηριακό Διδακτικό Προσωπικό, Ιόνιο Πανεπιστήμιο

Τριμελής Επιτροπή

Άννα Σωτηροπούλου, Εργαστηριακό Διδακτικό Προσωπικό, Ιόνιο Πανεπιστήμιο

Φοίβος Μυλωνάς, Αναπληρωτής Καθηγητής, Ιόνιο Πανεπιστήμιο

Σπυρίδων Δουκάκης, Επίκουρος Καθηγητής, Ιόνιο Πανεπιστήμιο

Αφιερώσεις

Το παιχνίδι αφιερώνεται στην μνήμη του Κωνσταντίνου Δραγάση Παλαιολόγου, βασιλιά των Ελλήνων, που επέλεξε να μείνει στην αιωνιότητα πέφτοντας μαζί με την Πόλη και έδωσε το έναυσμα στους αγωνιστές του 21' να παλέψουν για την ελευθερία μας τέσσερις αιώνες μετά.

Περίληψη

Σκοπός της παρούσας πτυχιακής εργασίας ήταν η δημιουργία ενός σοβαρού παιχνιδιού για χρήση στη διδασκαλία της Βυζαντινής ιστορίας. Επιλέχθηκε η δημιουργία ενός μικρού δισδιάστατου παιχνιδιού πάνω στη μηχανή Unity 3D, που σαν σκοπό είχε να προβάλει δύο μάχες κατά την περίοδο του Ιουστινιανού. Στην αρχή γίνεται μία αναδρομή στα ηλεκτρονικά παιχνίδια και στα σοβαρά παιχνίδια, μιας και αυτό ανήκει σε αυτήν την κατηγορία. Συνεχίζοντας γίνεται λεπτομερής αναφορά στις δύο ιστορικές μάχες, τη μάχη της Δάρας στην Ανατολία και τη μάχη των Ταγινών ή Βουσταγαλλώρων στην Ιταλική χερσόνησο. Ακολουθεί ανάλυση των εργαλείων που χρησιμοποιήθηκαν για τη δημιουργία του παιχνιδιού, η περιγραφή του παιχνιδιού, τα τεχνικά χαρακτηριστικά του και τέλος η ανάλυση του κώδικα. Ο παίκτης έχει τη δυνατότητα να περιηγηθεί επάνω στον χάρτη και να υπερασπιστεί όλες τις διοικητικές περιφέρειες της Βυζαντινής αυτοκρατορίας από τις εισβολές των βαρβάρων. Μαζεύοντας πόντους εμπειρίας ανεβαίνει επίπεδα, γίνεται ισχυρότερος και αντιμετωπίζει όλο και αυξανόμενης δυσκολίας μάχες. Με κάθε επίπεδο που ανεβαίνει, έχει περισσότερη ζωή και ισχυρότερο τόξο στα χέρια του. Τελικός στόχος του παίκτη είναι να φτάσει αρκετά μεγάλο επίπεδο έτσι ώστε να είναι αρκετά δυνατός και να νικήσει και στις δύο μάχες που προβάλλει το παιχνίδι.

Abstract

The object of this thesis is a serious game that would be used to teach Byzantine history. I opted for a small 2D game on the Unity 3D game engine, which aimed to project two battles during the Justinian period. At the beginning an analysis of electronic games and the subcategory of serious games, of which this game is a part of, is presented. Subsequently, there is a detailed reference to the two historical battles, the Battle of Dara in Anatolia and the Battle of the Tagines or Busta Gallorum in the Italian peninsula. Finally, the tools used to create the game, the game description, the technical characteristics and the code are presented. The player has the ability to navigate on the map and defend all the administrative regions of the Byzantine empire from invasions of the barbarians. By collecting experience points (s)he levels up, becomes stronger and faces increasingly difficult battles. With each level up, (s)he has more life and a stronger bow in his/her hands. The final goal of the player is to reach a high enough level so that (s)he is strong enough to win both battles that are featured the game.

Πρόλογος και ευχαριστίες

Η παρούσα εργασία μας εισάγει σε μια ωφέλιμη πλευρά των ηλεκτρονικών παιχνιδιών καθώς εμπλουτίζει τις γνώσεις μας πάνω στη ιστορία της Βυζαντινής Αυτοκρατορίας μέσω της δημιουργίας ενός σοβαρού παιχνιδιού που εκτυλίσσεται στη διάρκεια του αυτοκράτορα Ιουστινιανού.

Θα ήθελα να ευχαριστήσω όλους τους καθηγητές του τμήματός μου, που κατά την διάρκεια των σπουδών μου, δώσανε τον καλύτερο εαυτό τους για να μας μεταφέρουν τις γνώσεις τους. Ιδιαίτερως την κυρία Σωτηροπούλου που καθ' όλη την διάρκεια εκπόνησης αυτής της εργασίας στάθηκε δίπλα μου και με καθοδηγούσε.

Περιεχόμενα

Περιεχόμενα.....	6
1) Εισαγωγή.....	9
1.1) Ηλεκτρονικά παιχνίδια	9
1.2) Είδη και κατηγορίες ηλεκτρονικών παιχνιδιών	9
1.3) Παιχνίδια και μάθηση	10
1.4) Σοβαρά παιχνίδια	11
1.5) Εργασία	12
2) Ιστορικά στοιχεία των μαχών	15
2.1) Μάχη της Δάρας	15
2.2) Μάχη των Ταγινών ή Βουσταγαλλώρων	17
3) Εργαλεία	19
3.1) Unity 3D	19
3.2) Γραφικά	20
3.2.1) Illustrator	20
3.2.2) Photoshop	20
3.3) Azgaar's Map Generator	21
3.4) Visual Studio Code	21
4) Περιγραφή παιχνιδιού	23
5) Gameplay	24
6) Τεχνικά	30
7) Sprites	33
7.1) Sprites αρχικού χάρτη.....	33
7.2) Sprites εντός μαχών.....	33
8) Κώδικας και σχολιασμός του	36
9) Γλωσσάρι – Επεξηγήσεις.....	66

Κατάλογος Εικόνων

Εικόνα 1 Loading Screen.....	13
Εικόνα 2 Παράταξη δυνάμεων Βυζαντινών και Σασσανιδών.....	16
Εικόνα 3 Παράταξη δυνάμεων Βυζαντινών και Οστρογότθων.....	18
Εικόνα 4 Λογότυπο Unity	19
Εικόνα 5 Λογότυπο Illustrator	20
Εικόνα 6 & Εικόνα 7 Λογότυπο Photoshop & Banner - μάχη της Δάρας.....	21
Εικόνα 8 Λογότυπο Azgaar's Map Generator.....	21
Εικόνα 9 Λογότυπο VS Code.....	22
Εικόνα 10 Ολόκληρος ο χάρτης του παιχνιδιού	23
Εικόνα 11 Αρχή του παιχνιδιού	24
Εικόνα 12 Στατιστικά παίκτη (Ζωή, Επίπεδο, XP, Fire Rate)	25
Εικόνα 13 Μηνύματα περιοχής.....	26
Εικόνα 14 Μηνύματα μαχών.....	26
Εικόνα 15 Σκηνή μάχης	27
Εικόνα 16 Βέλος και Cool Down / Fire Indicator.....	27
Εικόνα 17 Χτύπημα εχθρού.....	28
Εικόνα 18 Μήνυμα νίκης / ήττας.....	28
Εικόνα 19 Επιστροφή στον αρχικό χάρτη	29
Εικόνα 20 Αρχική σκηνή στο Editor του Unity	31
Εικόνα 21 Σκηνή μάχης στο Editor του Unity	32
Εικόνα 22 Sprites αρχικού χάρτη	33
Εικόνα 23 Διαφορετικοί ουρανοί	34
Εικόνα 24 Σύννεφα	35
Εικόνα 25 Έδαφος.....	35

Κατάλογος Πινάκων

Πίνακας 1 Fire Rate και Ζωή ανά επίπεδο παίκτη	25
---	----

1) Εισαγωγή

1.1) Ηλεκτρονικά παιχνίδια

Έχουν περάσει χρόνια πλέον από τις εποχές που οι γονείς μας παίζανε στις γειτονιές των χωριών και τους δρόμους των πόλεων. Έχουμε περάσει πια σε μία εποχή που τα παιχνίδια έχουν αλλάξει τελείως μορφή, έχουν γίνει ηλεκτρονικά και βρίσκονται εντός του σπιτιού μας. Ένας ηλεκτρονικός υπολογιστής, μια κονσόλα παιχνιδιών και ένα κινητό έχουν αντικαταστήσει την μπάλα, το κυνηγητό και τα ποδήλατα. Μιλάμε πλέον για τα ηλεκτρονικά παιχνίδια που σύμφωνα με τον Zyda [1], καθηγητή πρακτικής επιστήμης υπολογιστών, είναι ένας πνευματικός διαγωνισμός με τη βοήθεια ενός υπολογιστή, πάνω σε συγκεκριμένους κανόνες και με σκοπό τη διασκέδαση ή τη νίκη σε ένα στοίχημα. Όλα τα παιδιά στις μέρες μας παίζουν ηλεκτρονικά παιχνίδια που η δημοτικότητά τους αυξάνεται μέρα με την μέρα. Για να παίξει κάποιος αυτά τα παιχνίδια συνήθως χρειάζεται παραπάνω από μια συσκευή. Για το υπολογιστή κυρίως, θα πρέπει να έχει αρκετά περιφερειακά και πολλές εταιρείες κατασκευάζουν περιφερειακά συγκεκριμένα για gaming. Ο σκοπός των ηλεκτρονικών παιχνιδιών είναι συνήθως η διασκέδαση χωρίς να προσφέρεται κάποια ιδιαίτερη ψυχαγωγία. Το Pong θεωρείται η αφετηρία της γέννησης της βιομηχανίας ηλεκτρονικών παιχνιδιών. Ήταν ουσιαστικά το πρώτο παιχνίδι με το οποίο το κοινό ήρθε σε άμεση επαφή. Ακολούθησαν το Pac Man και το Tetris που έγιναν τα σύμβολα των ηλεκτρονικών παιχνιδιών. Οι πρώτες κονσόλες ήρθαν τη δεκαετία του 1960, ενώ τα βιντεοπαιχνίδια μπήκαν στα πρώτα σπίτια στα μέσα της δεκαετίας του 1980.

1.2) Είδη και κατηγορίες ηλεκτρονικών παιχνιδιών

Με μια πρώτη ματιά θα μπορούσαμε να τα χωρίσουμε σε δύο μεγάλες κατηγορίες. Τα ρεαλιστικά και τα φανταστικά παιχνίδια. Στην πρώτη ομάδα εντάσσονται τα παιχνίδια προσομοίωσης και στη δεύτερη τα παιχνίδια φαντασίας. Ωστόσο υπάρχουν αρκετές υποκατηγορίες πέραν από αυτές τις δύο. Έχουμε παιχνίδια δράσης, περιπέτειας, βολών, στρατηγικής, αθλημάτων, γρίφων, γνώσεων και πολλά άλλα. Πρόχειρα παιχνίδια με πολύ απλούς κανόνες και στόχο την ψυχαγωγία. Διαφημιστικά παιχνίδια που από το όνομά τους φαίνεται που αποσκοπούν. Εκπαιδευτικά παιχνίδια για νεαρότερες ηλικίες και E-Sport, που είναι παιχνίδια ανταγωνιστικά και πολλών χρηστών. Τέλος υπάρχουν και τα σοβαρά παιχνίδια με στόχο την εκπαίδευση και την εξάσκηση του παίκτη σε κάποιο αντικείμενο, όπως είναι και το παιχνίδι της συγκεκριμένης εργασίας.

1.3) Παιχνίδια και μάθηση

Η διαδεδομένη χρήση των ηλεκτρονικών υπολογιστών άλλαξε τον τρόπο με το οποίο μαθαίνουμε πράγματα. Τα ηλεκτρονικά παιχνίδια έχουν δημιουργήσει νέα μαθησιακά περιβάλλοντα και νέες κοινωνικο-πολιτισμικές καταστάσεις. Υπάρχει έντονο ερευνητικό ενδιαφέρον για τη μάθηση μέσω των ηλεκτρονικών παιχνιδιών και συνέχεια καινούργιες δημοσιεύσεις γύρω από αυτή. Σύμφωνα με τον Θρασύβουλο-Κωνσταντίνο Τσιάτσο [2]: *“Τα περιβάλλοντα Διαδικτύου που υποστηρίζουν την εκπαιδευτική διαδικασία ή/και τις μαθησιακές δραστηριότητες ονομάζονται Εκπαιδευτικά Περιβάλλοντα Διαδικτύου”*. Μέχρι σήμερα η εκπαιδευτική τεχνολογία έχει χρησιμοποιηθεί για την υποβοήθηση της εκπαίδευσης με πολλούς τρόπους. Οι Παναγιωτακόπουλος, Πιερρακέας & Πιντέλας αναφέρουν ορισμένους από αυτούς, όπως χρήση εφαρμογών (α) ως μέσο πρόσβασης σε μαθησιακές πηγές (ψηφιακές βιβλιοθήκες, βάσεις δεδομένων, Παγκόσμιος Ιστός Πληροφοριών, κ.λπ.) και (β) ως μέσο υποστήριξης της διδασκαλίας και της μάθησης (CAI - Computer Assisted Instruction) με την αξιοποίηση κατάλληλου εκπαιδευτικού λογισμικού. Το εκπαιδευτικό λογισμικό που θα μπορούσε να κατηγοριοποιηθεί ως εξής:

Εκπαιδευτικό λογισμικό εξάσκησης. Τα λογισμικά αυτά δίνουν έμφαση στη συστηματική εξάσκηση του μαθητή και υποστηρίζουν το συμπεριφοριστικό μοντέλο.

Εκπαιδευτικό λογισμικό επίλυσης προβλημάτων. Βρίσκει εφαρμογή κυρίως σε θετικές επιστήμες και για την υποβοήθηση της ανάπτυξης υπολογιστικής σκέψης στους μαθητές.

Εκπαιδευτικό λογισμικό προσομοιώσεων. Χαρακτηριστικό παράδειγμα αποτελούν οι μικρόκοσμοι κι η γλώσσα Logo. Άλλο σημαντικό παράδειγμα είναι οι προσομοιωτές.

Εκπαιδευτικό λογισμικό καθοδήγησης/διδασκαλίας. Πρόκειται για βοηθήματα μάθησης που υποστηρίζουν το διδακτικό μοντέλο. Συνήθως βασίζονται σε οπτικοακουστικά μέσα διδασκαλίας ή σε αφήγηση συνοδευόμενη από την παρουσίαση εκπαιδευτικού υλικού.

Λογισμικό εκπαιδευτικών παιχνιδιών (serious games). Ένα serious game είναι ένα παιχνίδι σχεδιασμένο με πρωταρχικό σκοπό, να εκπαιδευτεί ο χρήστης σε μια συγκεκριμένη έννοια ή δεξιότητα, συνδυάζοντας τη μάθηση με την ψυχαγωγία. Παραδείγματα είναι τα παιχνίδια Volcanic Riddles και OSGame που έχουν αναπτυχθεί στο Εργαστήριο Πολυμέσων του Τμήματος Πληροφορικής του Αριστοτελείου Πανεπιστημίου Θεσσαλονίκης. [2]

1.4) Σοβαρά παιχνίδια

Τα σοβαρά παιχνίδια δεν έχουν ως πρώτο σκοπό τη διασκέδαση αλλά την εκπαιδευτική χρήση. Είναι συνήθως προσομοιώσεις πραγματικών γεγονότων και καταστάσεων, σχεδιασμένες με σκοπό την επίλυση ενός προβλήματος. Μπορούν να συνδυάζουν και άλλα στοιχεία και δίνουν τη δυνατότητα να νικήσεις κάτι, αλλά σε σχέση με τον πραγματικό κόσμο. Η χρήση τους έχει σοβαρά πλεονεκτήματα που τα μετατρέπει σε ισχυρά εργαλεία διδασκαλίας, όπως είναι η ανάπτυξη της στρατηγικής σκέψης, της διαπραγμάτευσης, της επικοινωνίας και της συνεργασίας. Η πρώτη χρήση του όρου «Σοβαρό Παιχνίδι» με νόημα κοντά στη σημερινή του χρήση φαίνεται να είναι το «Serious Games», ένα βιβλίο γραμμένο από τον Κλαρκ Άμπτ το 1970. Ο Άμπτ ήταν ερευνητής που εργάστηκε σε ερευνητικό εργαστήριο των ΗΠΑ κατά τη διάρκεια του ψυχρού πολέμου. Ένας από τους στόχους του ήταν να χρησιμοποιήσει παιχνίδια για εξάσκηση και εκπαίδευση. Στην πραγματικότητα σχεδίασε αρκετά ηλεκτρονικά παιχνίδια όπως το T.E.M.P.E.R. Το παιχνίδι αυτό χρησιμοποιήθηκε από στρατιωτικούς αξιωματικούς για να μελετήσουν τη σύγκρουση του Ψυχρού Πολέμου σε παγκόσμια κλίμακα. Στο βιβλίο του όμως ο Άμπτ, παρέχει κάποια παραδείγματα «μη ψηφιακών» Σοβαρών Παιχνιδιών, όπως παιχνίδια που σχετίζονται με τα μαθηματικά στα σχολεία. Ο Άμπτ δίνει επίσης έναν σαφή ορισμό των "Σοβαρών Παιχνιδιών": *«Τα παιχνίδια μπορούν να παιχτούν σοβαρά ή για διασκέδαση. Ασχολούμαστε με σοβαρά παιχνίδια με την έννοια ότι αυτά τα παιχνίδια έχουν σαφή και προσεκτικά μελετημένο εκπαιδευτικό σκοπό και δεν προορίζονται να παιχτούν κυρίως για διασκέδαση. Αυτό δεν σημαίνει ότι τα σοβαρά παιχνίδια δεν είναι, ή δεν πρέπει να είναι, διασκεδαστικά»*. [3]

Οι ερευνητές των σοβαρών ηλεκτρονικών παιχνιδιών απασχολούνται με την προοπτική εξέλιξής τους. Ο James P. Gee διερωτάται «Πώς στην πραγματικότητα μαθαίνουν οι παίκτες να παίζουν ηλεκτρονικά παιχνίδια; Μήπως η δραστηριότητα του να παίζει κανείς αυτά τα παιχνίδια συνιστά έναν ιδιαίτερα αποτελεσματικό τρόπο για να οργανώνονται οι μαθησιακές δραστηριότητες; » [4]

Η απάντηση σε αυτό το ερώτημα θα μπορούσε κάλλιστα να δοθεί μέσα από τα συμπεράσματα της έρευνας των Α. Γιαννοπούλου και Χ. Παναγιωτακόπουλου με τίτλο «Το ηλεκτρονικό παιχνίδι και το παιδί: Επιδράσεις και προοπτικές»: [5]

«Ο μαγικός εικονικός κόσμος του ηλεκτρονικού παιχνιδιού εξελίσσεται με ολοένα ταχύτερους ρυθμούς. Χαρακτηριστικά των ηλεκτρονικών παιχνιδιών, όπως η διασκέδαση, η φαντασία, η πρόκληση, ο ανταγωνισμός, η διάδραση, η συγκίνηση που προκαλούν, σε σχέση με τις ώρες που διαθέτουν τα παιδιά σε αυτού του είδους τα παιχνίδια, θα πρέπει να ληφθούν πολύ σοβαρά υπόψη, όσο αφορά την ενσωμάτωση αυτών των χαρακτηριστικών στη μάθηση. Άλλωστε, η εμπειρία του ηλεκτρονικού παιχνιδιού δείχνει να επηρεάζει τις

προσδοκίες μάθησης των παιδιών, με την έννοια ότι προτιμούν δραστηριότητες που είναι γρήγορες, ευεργητικές και διερευνητικές με πολλαπλά και παράλληλα κανάλια πληροφόρησης.»

Επιπλέον, η Γιαννοπούλου και ο Παναγιωτακόπουλος προσθέτουν ότι στις ψυχολογικές επιστήμες, στη συζήτηση για την κοινωνική και γνωστική ανάπτυξη του παιδιού, φαίνεται ότι η ενασχόληση με τα ηλεκτρονικά παιχνίδια είναι σημαντικός τρόπος για να αναπτυχθεί η δομική σκέψη, ταυτότητα αξιών και κανόνων του παιδιού.

Τι πιστεύουν όμως οι φοιτητές για τη χρήση τους στη μαθησιακή διαδικασία; Σύμφωνα με το άρθρο των Γ. Κουτρομάνου και Κ. Νικολοπούλου για το 1^ο Εκπαιδευτικό συνέδριο «Έναρξη και Χρήση των ΤΠΕ στην εκπαιδευτική διαδικασία», με τίτλο «Εκπαιδευτικά ψηφιακά παιχνίδια: Τι πιστεύουν οι φοιτητές για τη χρήση τους στη μαθησιακή διαδικασία». [6] Έγινε μια έρευνα για τη στάση των φοιτητών απέναντι στα εκπαιδευτικά ψηφιακά παιχνίδια. Το 61,5% των φοιτητών/τριών χρησιμοποιούν ψηφιακά παιχνίδια για προσωπική χρήση, ενώ μόλις το 38,5% όχι. Πάραυτα όλοι οι φοιτητές είχαν σχετικά θετικές στάσεις και συμπεριφορικές πεποιθήσεις απέναντι στη χρήση των ψηφιακών παιχνιδιών στην διδασκαλία.

“Η δημιουργία ενός σοβαρού παιχνιδιού απαιτεί πολύ περισσότερο χρόνο από την ανάπτυξη ενός απλού ηλεκτρονικού παιχνιδιού.”, σύμφωνα με τον Zyda [1]

Η δημιουργία και χρήση των σοβαρών παιχνιδιών έχει πολλά πλεονεκτήματα καθώς μπορούν να μετατραπούν σε δυνατά εργαλεία διδασκαλίας προάγοντας τη στρατηγική σκέψη, τη συνεργασία, την επικοινωνία, τον σχεδιασμό, τις δεξιότητες διαπραγμάτευσης, τη δημιουργία και προώθηση της γνώσης, αλλά και τη συμμετοχή σε ομάδες. Με τη διάθεση μικρών οικονομικών πόρων μπορούν να δημιουργηθούν εφαρμογές απλές στην χρήση, που θα ωθήσουν τον χρήστη στην απόκτηση γνώσης μέσω της ψυχαγωγίας. Ο στρατός, εταιρικοί οργανισμοί, εκπαιδευτικοί οργανισμοί και οργανισμοί υγείας έχουν ανακαλύψει αυτά τα πλεονεκτήματά και κάνουν χρήση τους όλο και πιο πολύ. Η ανάπτυξή τους στο μέλλον βρίσκεται σε θετική τροχιά αρκεί να υπάρχει το σωστό υλικό που θα διαπραγματευτούν, η επιλογή καλών εκπαιδευτών και η διδαχή αξιόλογων αντικειμένων.

1.5) Εργασία

Η παρούσα πτυχιακή εργασία είχε ως θέμα τη δημιουργία ενός σοβαρού παιχνιδιού γύρω από την ιστορία του Βυζαντίου. Σαν αρχική σκέψη ήταν η δημιουργία ενός παιχνιδιού First Person, όπου ο παίκτης θα περιηγούταν στους δρόμους της Κωνσταντινούπολης, θα έκανε διάφορες δουλειές/tasks για να μαζέψει χρήματα και να αγοράσει πανοπλία και οπλισμό και θα ανέβαινε επίπεδα όταν έπαιρνε μέρος στις μάχες

με τον στρατό του Βυζαντίου. Μία δεύτερη σκέψη διαφορετικού παιχνιδιού που είχα, ήταν ο παίκτης να είχε ένα κομμάτι γης το οποίο θα το εκμεταλλευόταν και θα είχε τον ρόλο του Ακρίτα. Ανά διαστήματα θα τον καλούσε ο αυτοκρατορικός στρατός για να πάρει μέρος σε κάποια μάχη που θα γινόταν κοντά στην περιοχή του, και μέσα από αυτές τις μάχες θα έπαιρνε πόντους εμπειρίας ή αλλιώς ΧΡ. Μετά από περεταίρω σκέψη όμως αποφασίστηκε ότι καλύτερο θα ήταν ένα παιχνίδι χωρίς τόσους περίπλοκους μηχανισμούς, μικρότερης πολυπλοκότητας το οποίο θα ήταν πιο εύκολο να το παίξει μέχρι και ένα μικρό παιδί. Κατέληξα λοιπόν να αλλάξω τελείως την προσέγγισή μου και να επιλεχθεί ένα δυσδιάστατο παιχνίδι τύπου «Tower Defense», δηλαδή ένα παιχνίδι που ο παίκτης πρέπει να υπερασπιστεί τον πύργο του. Στην δικιά μας περίπτωση ο παίκτης δεν υπερασπίζεται κάποιον πύργο, αλλά τις περιοχές του Βυζαντίου. Για να ταιριάζει ο τίτλος του παιχνιδιού με το περιεχόμενό του, αποφασίστηκε να ονομαστεί **«The Byzantine Quest»**



Εικόνα 1 Loading Screen¹

Τελικός στόχος του παιχνιδιού είναι ο παίκτης να νικήσει δύο σημαντικές μάχες κατά την περίοδο του Ιουστινιανού. Επιλέχθηκε ο παίκτης να αρχίζει στην άκρη των τότε συνόρων της αυτοκρατορίας σαν «Level 1», όπου το τόξο του είναι αδύναμο και η ζωή του περιορισμένη. Ο παίκτης μπορεί να εισέλθει σε όλες τις διοικητικές περιφέρειες του Βυζαντίου και να τις υπερασπιστεί από τις επιθέσεις των βαρβάρων που έρχονται να τις καταλάβουν. Με κάθε νίκη του στις γύρω περιφέρειες μαζεύει ΧΡ και έτσι ανεβαίνει σιγά-σιγά επίπεδα. Στόχος είναι να φτάσει σε αρκετά μεγάλο επίπεδο έτσι ώστε να

¹ Γραφικό από τον Κωνσταντίνο Παπά [Instagram.com/p/CeItXLTjbPK/](https://www.instagram.com/p/CeItXLTjbPK/) [Τ.Η.Π 22-9-2022]

μπορέσει να νικήσει στις δύο μάχες, οι οποίες είναι και αρκετά δύσκολες. Οι δύο μάχες που μπήκαν στο παιχνίδι είναι :

1. Η μάχη της Δάρας, όπου το 530 μ.Χ. οι Βυζαντινοί αριθμούσαν τους 25.000 άνδρες με στρατηγό τον Βελισάριο, και οι Πέρσες της αυτοκρατορίας των Σασσανιδών αριθμούσαν περίπου τους 40.000 άνδρες με στρατηγό τον Περόζη. [7]
2. Η μάχη των Ταγινών ή Βουσταγαλλώρων το 552 μ.Χ., με 25.000 Βυζαντινούς υπό τον Νάρση εναντίων 15.000 Οστρογόθων υπό τον βασιλιά Τωτίλα. [7]

2) Ιστορικά στοιχεία των μαχών

Εμβαθύνοντας περισσότερο τα ιστορικά στοιχεία των μαχών, θα δούμε αναλυτικά τι έγινε εντός του πεδίου μάχης σύμφωνα με τον John Haldon στο βιβλίο του «The Byzantine Wars». [7]

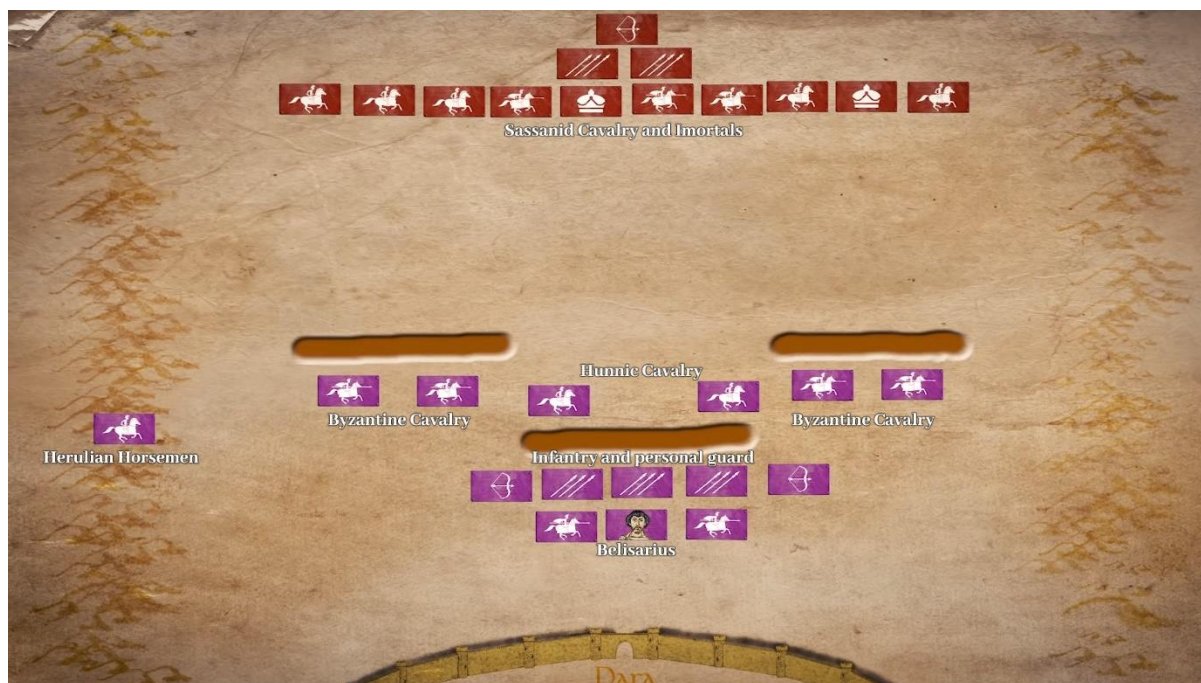
2.1) Μάχη της Δάρας

Συνέβη το 530 μ.Χ. στη Δάρα, στο σημερινό Ογκούζ της Τουρκίας, η οποία χτίστηκε από τον Βυζαντινό αυτοκράτορα Αναστάσιο Α' τον Δίκορο. Κατά το τελευταίο έτος της βασιλείας του αυτοκράτορα Ιουστίνου Α' (518 μ.Χ. – 527 μ.Χ.) είχε ξεσπάσει πόλεμος μεταξύ της αυτοκρατορίας των Σασσανιδών Περσών και της Βυζαντινής αυτοκρατορίας. Στις αρχές του Αυγούστου του 527 μ.Χ. όμως, ο Ιουστίνος ο Α' πέθανε και ο νέος αυτοκράτορας Ιουστινιανός πήρε τη σκυτάλη του θρόνου προτείνοντας μια συνθήκη ειρήνης με τους Πέρσες. Οι Πέρσες, αντί να διαπραγματευτούν, έστειλαν ένα σημαντικό τμήμα του στρατού τους με μέγεθος περίπου 30.000 άνδρες, το οποίο νίκησε τη Βυζαντινή φρουρά στην πόλη της Νίσιβις και κατέστρεψε τα κτίσματα. Το 530 μ.Χ. ο στρατηγός Βελισάριος διορίστηκε ως αρχιστράτηγος του ανατολικού στρατού και στάλθηκε για να αποκαταστήσει τη Βυζαντινή κυριαρχία στην περιοχή. Ο Βελισάριος και ο ανθυπασιπιστής του Ερμογένης είχαν συγκεντρώσει συνολικά περίπου 25.000 άνδρες. Πήραν θέση έξω από την πόλη της Δάρας για να αποκρούσουν την περσική επίθεση. Οι Πέρσες έστειλαν μια δύναμη κάπως μεγαλύτερη, που μάλλον έφτανε τους 40.000 άνδρες. Η νίκη που ακολούθησε αποκατέστησε τη διαπραγματευτική δύναμη των Βυζαντινών.

Η σύγκρουση κράτησε για δύο μέρες. Ο Βελισάριος είχε προετοιμάσει μια θέση μερικώς θωρακισμένη από μια σειρά αμυντικών τάφρων που ήτανε σκαμμένες κατά μήκος του κεντρικού δρόμου από την Δάρα προς τη Νίσιβις. Αποτελούνταν από δύο μακριές πλευρικές τάφρους και από μία μικρότερη στο κέντρο, λίγο πιο πίσω από τις πλαϊνές. Τα μεσαία τμήματα των πλαϊνών ενώνονταν με την μικρή κεντρική τάφρο, και από πάνω υπήρχαν πολυάριθμες θέσεις διέλευσης στρατευμάτων. Οι περσικές δυνάμεις συγκροτήθηκαν σε δύο γραμμές, η μία πίσω από την άλλη.

Η αρχική φάση της περσικής επίθεσης άρχισε με μια κατά μέτωπο έφοδο του ιππικού, προς την δεξιά πτέρυγα του Βυζαντίου. Οι δυνάμεις της δεξιάς πτέρυγας που ήταν υπό τον Βούζη, άρχισαν να υποχωρούν καθώς οι Πέρσες κερδίζανε έδαφος. Τότε ο Βούζης έδωσε εντολή για μια ξαφνική αντεπίθεση. Οι Πέρσες αποσύρθηκαν φοβούμενοι ότι θα χτυπηθούνε από τα πλάγια. Έπειτα, ένας Πέρσης στρατιώτης προκάλεσε τους Βυζαντινούς σε ατομική μονομαχία. Παραδόξως, την πρόκληση δεν την ανέλαβε ένας

απλός στρατιώτης, αλλά κάποιος με το όνομα Ανδρέας, προσωπικός φύλακας στη συνοδεία του Βούζη και παλαιστής στο επάγγελμα. Ο Ανδρέας μπόρεσε γρήγορα να ρίξει τον αντίπαλό του από το άλογο και τον σκότωσε. Αυτό ενίσχυσε πολύ το Βυζαντινό ηθικό. Οι περσικές δυνάμεις φαίνεται να είχαν καταρρακωθεί από αυτή την ατυχία, και καθώς πλησίαζε το σούρουπο, άρχισαν να αποσύρονται στο στρατώνα τους στον Αμμόδιο.



Εικόνα 2 Παράταξη δυνάμεων Βυζαντινών και Σασσανιδών

Την επόμενη μέρα οι Πέρσες έλαβαν περίπου 10.000 ενισχύσεις. Το μεσημέρι και οι δύο πλευρές άρχισαν μια βαριά επίθεση με βέλη και υπέστησαν μερικές απώλειες. Λίγο μετά οι Πέρσες εξαπέλυσαν μια γενική επίθεση σε όλη τη γραμμή. Φαίνεται ότι ακολούθησε μια μικρή παύση κατά την οποία και οι δύο πλευρές ανασυγκροτήθηκαν. Μετά από αυτή την παύση ο Περόζης αποφάσισε να εξαπολύσει την κύρια επίθεσή του εναντίον της δεξιάς Βυζαντινής πτέρυγας. Έστειλε μια μεγάλη δύναμη συνοδευόμενη από τις ειδικές μονάδες προερχόμενες από τους Αθάνατους, τα ελίτ στρατεύματα των Περσών βασιλέων.

Η ισχυρή αυτή επίθεση στη Βυζαντινή δεξιά πτέρυγα, ανάγκασε τις μονάδες ιππικού να υποχωρήσουν και να αποσυρθούν άτακτα. Από τη θέση του στο κέντρο, ωστόσο, ο Βελισάριος συνειδητοποίησε τι συνέβαινε και έστειλε γρήγορα ενισχύσεις πεζικού και ιππικού για να βοηθήσει. Η Βυζαντινή αντιμετώπιση φαίνεται να ήταν πολύ αποτελεσματική γιατί έκοψε την εχθρική γραμμή σε δύο άνισα μέρη. Το μεγαλύτερο από τα δύο μέρη συνέχισε να προελαύνει μετά την υποχώρηση του Βυζαντινού ιππικού, αλλά το μικρότερο τμήμα είχε πλέον αποκοπεί τόσο από την κύρια περσική γραμμή, όσο και από τις προελαύνουσες περσικές μονάδες και βρέθηκε αποκλεισμένο. Η σημαία

του στρατηγού Βαρεσμάνη έπεσε και σε αυτό το σημείο το κύριο σώμα των προωθημένων περσικών μονάδων συνειδητοποίησε τι συνέβαινε. Περίπου πέντε χιλιάδες από τα περσικά στρατεύματα σε αυτή την περιοχή της μάχης έπεσαν, και καθώς οι περσικές μονάδες διασπάστηκαν, ολόκληρη η περσική γραμμή άρχισε να διαλύεται, και τελικά κατέρρευσε.

2.2) Μάχη των Ταγινών ή Βουσταγαλλώρων

Ο στρατός που διοικούσε ο Ναρσής το 552 μ.Χ. αριθμούσε περίπου 20.000-25.000 άνδρες. Αυτό περιλάμβανε σημαντικό αριθμό συμμαχικών στρατευμάτων, καθώς ο Λομβαρδός βασιλιάς Αυδοϊνος είχε στείλει μια δύναμη σχεδόν 6.000 ανδρών για να βοηθήσει την αυτοκρατορία. Υπήρχαν επιπλέον περίπου 4.000 Ερούλοι, οι οποίοι συμπλήρωναν το τακτικό αυτοκρατορικό ιππικό και πεζικό, αλλά και νέες μονάδες που στρατολογήθηκαν από τη Θράκη και το Ιλλυρικό από τον ίδιο τον Ναρσή. Γνωρίζοντας ότι ο Τωτίλας θα αναγκαζόταν να τον αναζητήσει για να τον διώξει από την επικράτειά του, ο Ναρσής αποφάσισε να υιοθετήσει μια αμυντική θέση και να περιμένει τη Γοτθική επίθεση. Η θέση του ήταν σε ελαφρώς ανυψωμένο έδαφος γύρω από την πεδιάδα από όπου θα έπρεπε να περάσει η Γοτθική δύναμη για να του επιτεθεί. Οι σύμμαχοι Λομβαρδοί, Ερούλοι και Γεπίδες πιάσανε αμυντικές θέσεις στις δύο πτέρυγες, τοποθετώντας ιππικό αποτελούμενο από δύο στρατεύματα οπλισμένα με τόξα, λόγχες και επιπλέον 1.500 έφεδρους ιππείς.

Οι δυνάμεις του Τωτίλα στρατοπέδευσαν σε μια τοποθεσία περίπου είκοσι χιλιόμετρα νότια, στη θέση που σήμερα ονομάζεται Fossato (αναμφίβολα από το λατινικό fossatum, το στρατόπεδο). Ο Ναρσής προσπάθησε να βρει μια ειρηνική λύση, ωστόσο ο Τωτίλας την απέρριψε. Την επόμενη ημέρα οι Γοτθικές δυνάμεις εθεάθησαν να πλησιάζουν τις Βυζαντινές θέσεις. Οι Γότθοι έκαναν αρκετές σφοδρές επιθέσεις στη Βυζαντινή θέση. Ο Τωτίλας προσπάθησε να καθυστερήσει την έναρξη της μάχης κάνοντας μια σειρά επιδείξεων υπασίας. Επίσης, προηγήθηκε και μια πρόκληση για ατομική μονομαχία από έναν γότθο στρατιώτη, που κατέληξε στην ήττα και θάνατό του, με αποτέλεσμα οι Γότθοι να αποσυρθούν στο στρατόπεδο τους. Αυτή η κατάσταση ενθάρρυνε τον Ναρσή να εξαπολύσει επίθεση στους Γότθους που αποχωρούσαν, αλλά δεν το έκανε. Σύμφωνα με τον Προκόπιο, από την άλλη πλευρά, ο Τωτίλας ήλπιζε ότι η επέλαση των Βυζαντινών θα του επέτρεπε να κάνει μια αιφνιδιαστική αντεπίθεση, αλλά ο Ναρσής γνώριζε την πιθανότητα και έβαλε τον στρατό σε θέση ετοιμότητας. Ο Τωτίλας είχε αλλάξει τον σχηματισμό του με την ελπίδα ότι το σχέδιό του θα λειτουργούσε συγκεντρώνοντας το ιππικό στο κέντρο και το πεζικό από πίσω τους. Οι Γότθοι προφανώς

σκόπευαν να εξαπολύσουν μια μαζική επίθεση στο κέντρο, έτσι ο Ναρσής έβαλε τους τοξότες στα άκρα κάθε πτέρυγας με σκοπό να τους πλαγιοκοπήσουν.

Η μάχη δεν κράτησε πολύ. Ο Τωτίλας είχε διατάξει όλα τα στρατεύματά του να πολεμήσουν μόνο με τη λόγχη, και με μία μόνο μαζική επίθεση η γοτθική γραμμή πέρασε προς τα εμπρός κατά του Βυζαντινού κέντρου. Οι Γότθοι φαινομενικά δεν είχαν καμία ευκαιρία αν ήθελαν να υποχωρήσουν για να ανασυνταχθούν. Κάτω από την πίεση αυτής της γενικής εφόδου άρχισαν να διαλύονται άτακτα. Αντί να υποχωρήσουν πίσω από την ασφάλεια του πεζικού τους, έπεσαν κατευθείαν επάνω τους, προκαλώντας έτσι περαιτέρω θύματα και πανικό, με αποτέλεσμα ολόκληρος ο στρατός να διαλυθεί σε μια ανεξέλεγκτη μάζα. Σύμφωνα με τον Προκόπιο, σκοτώθηκαν περίπου 6.000 άνδρες, συμπεριλαμβανομένου του ίδιου του Γότθου ηγέτη. Οι Βυζαντινοί είχαν πετύχει μια σημαντική νίκη.

Ένα από τα πιο σημαντικά σημεία που προκύπτουν από την αφήγηση του Προκοπίου είναι η τάξη και η πειθαρχία που επέδειξαν οι Βυζαντινές δυνάμεις. Αυτό παραμένει μια σημαντική πτυχή των Βυζαντινών στρατιωτικών επιχειρήσεων σε όλο τον έκτο αιώνα και μετά, συχνά αντιπαραβαλλόμενη ευνοϊκά όχι μόνο με τους «βάρβαρους» εχθρούς της αυτοκρατορίας αλλά και με εκείνους που είχαν πολύ υψηλότερο βαθμό στρατιωτικής διοίκησης και οργάνωσης, όπως οι Πέρσες.



Εικόνα 3 Παράταξη δυνάμεων Βυζαντινών και Οστρογόθων

3) Εργαλεία

Για τη δημιουργία του παιχνιδιού χρησιμοποιήθηκαν αρκετά εργαλεία, για τον κώδικα, για τις εικόνες, για τους ήχους και για το στήσιμο των σκηνών του παιχνιδιού. Αρχικά το παιχνίδι επέλεξα να φτιαχτεί επάνω σε μία έτοιμη μηχανή παιχνιδιών. Η μηχανή της επιλογής μου είναι το Unity 3D, καθώς είχα προηγούμενη εμπειρία και είχα την άνεση να δουλέψω σε αυτή.

3.1) Unity 3D

Το Unity (γνωστό και ως Unity3D) είναι μια μηχανή παιχνιδιών για τη δημιουργία διαδραστικών μέσων, συνήθως βιντεοπαιχνιδιών. Όπως είπε ο CEO της εταιρίας, David Helgason, το Unity *«είναι ένα σύνολο εργαλείων που χρησιμοποιείται για την κατασκευή παιχνιδιών. Είναι η τεχνολογία που εκτελεί τα γραφικά, τον ήχο, τη φυσική, τις αλληλεπιδράσεις και τη δικτύωση»*. [8] Το Unity φημίζεται για τις γρήγορες δυνατότητες δημιουργίας πρωτοτύπων παιχνιδιών και για τον μεγάλο αριθμό δημοσιευμένων παιχνιδιών.

Η πρώτη έκδοση του Unity (1.0.0) δημιουργήθηκε από τρεις ανθρώπους, τους συνεργάτες David Helgason, Nicholas Francis και Joachim Ante στη Δανία. Το αρχικό προϊόν κυκλοφόρησε στις 6 Ιουνίου 2005. Ο στόχος ήταν να δημιουργηθεί μια οικονομική μηχανή παιχνιδιών με επαγγελματικά εργαλεία για ερασιτέχνες προγραμματιστές παιχνιδιών, ενώ παράλληλα να «εκδημοκρατιστεί η ανάπτυξη παιχνιδιών». Οι τρεις εμπνεύστηκαν από την εύκολη ροή εργασίας, την απλή διεπαφή στοιχείων μέσω Drag and Drop του προϊόντος Final Cut Pro της Apple. Όταν κυκλοφόρησε αρχικά, το Unity ήταν διαθέσιμο μόνο για Mac OS X και οι προγραμματιστές μπορούσαν να αναπτύξουν τις εφαρμογές τους μόνο σε λίγες πλατφόρμες. Η τρέχουσα έκδοση (2022.1.17) υποστηρίζεται τόσο σε Windows, όσο και σε Mac OS και Linux. Δίνει τη δυνατότητα να δημιουργηθούν εκτελέσιμα αρχεία για τουλάχιστον δέκα πλατφόρμες. [8]



Εικόνα 4 Λογότυπο Unity

3.2) Γραφικά

Για τα γραφικά χρησιμοποιήθηκαν δύο κυρίως προγράμματα, το Adobe Photoshop και το Adobe Illustrator. Επιπλέον κάποια από τα γραφικά, όπως οι εχθροί πάρθηκαν από ένα ανοικτό πακέτο sprites για παιχνίδια διαθέσιμο στο ίντερνετ².

3.2.1) Illustrator

Το Illustrator είναι ένα λογισμικό γραφικών από την εταιρία Adobe που δίνει την δυνατότητα δημιουργίας διανυσματικών γραφικών στους χρήστες του. Κυκλοφόρησε για πρώτη φορά το 1987 για συσκευές Apple Macintosh και ήταν ένα από τα καινοτόμα προϊόντα της Adobe. Από τα πιο σημαντικά χαρακτηριστικά του, ήταν το εργαλείο κατασκευής καμπυλών, γνωστό και ως pen tool. Μέσω αυτού του εργαλείου, ο χρήστης μπορούσε εύκολα να κατασκευάσει καμπύλες και να δημιουργήσει σχήματα και εικόνες υψηλής ανάλυσης. Το 2003 έγινε μέρος της Adobe Creative Suite, που είναι ένα πακέτο εργαλείων για γραφίστες και σχεδιαστές. Το 2013 η Adobe ανακοίνωσε πως το Illustrator θα είναι διαθέσιμα μόνο μέσω μια συνδρομητικής υπηρεσίας. [9] Το Illustrator χρησιμοποιήθηκε για την δημιουργία του παίκτη, χρησιμοποιώντας ένα κομμάτι από το sprite των εχθρών και σχεδιάζοντας όλο το υπόλοιπο εντός της εφαρμογής. Έπειτα γίνανε μερικές περαιτέρω τροποποιήσεις στον παίκτη εντός του Photoshop.



Εικόνα 5 Λογότυπο Illustrator

3.2.2) Photoshop

Το Photoshop είναι λογισμικό που χρησιμοποιείται για την επεξεργασία ψηφιδωτών γραφικών. Αναπτύχθηκε το 1987 από δύο Αμερικάνους αδελφούς, τους Tomas και John Knoll. Οι δύο τους πουλήσανε την άδεια διανομής του, στην Adobe το έτος 1988. Το 2003 όπως και με το Illustrator η Adobe έκανε το Photoshop κομμάτι της Adobe Creative Suite και το 2013, έγινε και αυτό διαθέσιμο μέσω μίας συνδρομητικής υπηρεσίας. [10] Με το Photoshop έγινε η περισσότερη δουλειά που

² www.gameart2d.com/the-knight-free-sprites.html [Τ.Η.Π 22-9-2022]

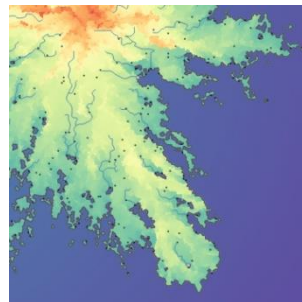
αφορούσε το εμφανισιακό κομμάτι του παιχνιδιού. Εκεί γίνανε οι σημαίες, τα banner για τις μάχες, οι διάφοροι ουρανοί, η μπάρα του XP και όλα τα διαφορετικά εδάφη.



Εικόνα 6 & Εικόνα 7 Λογότυπο Photoshop & Banner - μάχη της Δάρας

3.3) Azgaar's Map Generator

Το Azgaar's Map Generator είναι μια εφαρμογή διαδικαστικής παραγωγής χαρτών. Δηλαδή, παράγει μέσω ενός αλγορίθμου αυτόματα έναν χάρτη, τον χωρίζει σε διάφορες περιοχές-βασίλεια και τους δίνει ένα τυχαίο όνομα. Επίσης μπορούμε να παραμετροποιήσουμε τον χάρτη, δίνοντάς του σχήμα, όνομα στις περιοχές και πόλεις ή επίσης μπορούμε να τον κάνουμε εξ' ολοκλήρου χειροκίνητα. Για τη δημιουργία του χάρτη του παιχνιδιού, χρησιμοποιήθηκε σαν βάση ο χάρτης της Ευρώπης και επάνω του σημειώθηκαν όλες οι διαφορετικές διοικητικές περιφέρειες του Βυζαντίου κατά την περίοδο του Ιουστινιανού. Επίσης σημειώθηκαν πολλές πόλεις της τότε εποχής, με έμφαση στις μεγαλύτερες της κάθε περιφέρειας.³



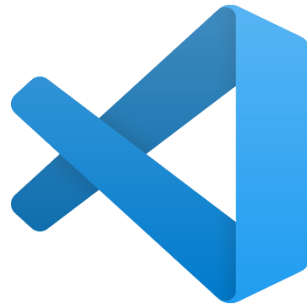
Εικόνα 8 Λογότυπο Azgaar's Map Generator

3.4) Visual Studio Code

Για την δημιουργία του κώδικα χρησιμοποιήθηκε σαν text editor το Visual Studio Code. Το Visual Studio Code είναι ένα ελαφρύ πρόγραμμα δημιουργίας κώδικα φτιαγμένο από την Microsoft και είναι βασισμένο στο πολύ μεγαλύτερο IDE Visual Studio. Το VS Code, όπως συχνά αναφέρεται, είναι σχεδιασμένο να είναι ένας πάρα πολύ γρήγορος επεξεργαστής πηγαίου κώδικα. Έχει την δυνατότητα να υποστηρίξει

³ azgaar.github.io/Fantasy-Map-Generator [Τ.Η.Π 22-9-2022]

εκατοντάδες γλώσσες προγραμματισμού. Έχει εργαλεία που βοηθάνε στον προγραμματισμό όπως, αυτόματη μορφοποίηση, επισήμανση σύνταξης, πολλαπλή επιλογή γραμμών, συντομεύσεις πληκτρολογίου και πολλά άλλα. Προσφέρει επίσης πολλές επιλογές προσαρμογής μέσα από πολλά θέματα, γραμματοσειρές και επεκτάσεις που μπορεί να δημιουργήσει ο καθένας. Τέλος, ο πηγαίος κώδικας του VS Code είναι ανοικτός για να τον δει ο καθένας. Αυτό δίνει την δυνατότητα σε πολλούς προγραμματιστές να συνεισφέρουν στην ανάπτυξη του με διάφορα χαρακτηριστικά.⁴



Εικόνα 9 Λογότυπο VS Code

Για την κατανόηση κάποιων κομματιών του κώδικα χρησιμοποιήθηκε το Stack Overflow, τα Unity Docs και αρκετά tutorials από το YouTube. Οι ήχοι επεξεργάστηκαν για να ταιριάζουν στο παιχνίδι με το Audacity. Οι περισσότεροι εξ' αυτών πάρθηκαν από ελεύθερα προς χρήση audio clips από το YouTube είτε έπειτα από άδεια από τον καλλιτέχνη⁵. Επιπλέον προστέθηκαν και κάποιες έτοιμες γραμματοσειρές, ώστε να δοθεί στο παιχνίδι ένα πιο μεσαιωνικό στίλ. Οι γραμματοσειρές που χρησιμοποιήθηκαν ήταν η Morris Roman⁶ και η JSL Ancient⁷, και τις δύο τις πήρα από το Dafont.com και είναι και οι δύο ελεύθερες προς χρήση.

⁴ code.visualstudio.com/docs/editor/whyvscode [Τ.Η.Π 22-9-2022]

⁵ www.youtube.com/channel/FaryaFaraji [Τ.Η.Π 22-9-2022]

⁶ www.dafont.com/morris-roman.font [Τ.Η.Π 22-9-2022]

⁷ www.dafont.com/jsl-ancient.font [Τ.Η.Π 22-9-2022]

4) Περιγραφή παιχνιδιού

Το παιχνίδι αποτελείται από 13 κύριες περιοχές, συγκεκριμένα : Spaniae, Italia Annonaria, Italia Suburbicaria, Illyricum, Dacia, Macedonia, Quaestura Exercitus, Thracia, Pontica, Asiana, Oriens, Aegyptus και τέλος Africa. Αυτές είναι οι διοικητικές περιοχές κατά την περίοδο που αυτοκράτορας ήταν ο Ιουστινιανός.



Εικόνα 10 Ολόκληρος ο χάρτης του παιχνιδιού

Σκοπός μας είναι να νικήσουμε όλα τα μέτωπα και στις 13 περιοχές έτσι ώστε ο παίκτης μας να ανέβει levels και να γίνει αρκετά δυνατός. Με κάθε νίκη μας σε μία από τις 13 περιοχές μαζεύουμε XP, όταν έχουμε φτάσει ένα μεγάλο level τελικός σκοπός του παιχνιδιού μας είναι να νικήσουμε τις δύο μάχες. Οι δύο μάχες είναι μεγαλύτερης δυσκολίας σε σχέση με τα υπόλοιπα μέτωπα στις 13 διοικητικές περιοχές. Εκ των δύο μαχών, η μάχη της Δάρας είναι η πιο δύσκολη και είναι η μάχη που δίνει τους περισσότερο πόντους εμπειρίας (XP). Μετά από τη νίκη της κάθε μιας από τις δύο μάχες ο παίκτης μας παίρνει σαν bonus την απόσταση την οποία μπορεί να φτάσει να ρίξει το τόξο του. Συγκεκριμένα, αν νικήσει τη μία από τις δύο μάχες τότε η απόσταση πάει +10%, ενώ αν νικήσει και τις δύο πάει +20% της αρχικής απόστασης. Μετά τη νίκη του και στη δεύτερη μάχη, χωρίς να προσδιορίζεται η σειρά, το παιχνίδι έχει τελειώσει και ο παίκτης θεωρείται νικητής! Τώρα θα έχει τη δυνατότητα εάν έχει αφήσει κάποια από τις περιφέρειες ανυπεράσπιστα να μπει να την υπερασπιστεί με ένα παντοδύναμο τόξο.

5) Gameplay

Το παιχνίδι αρχίζει από την Ιβηρική χερσόνησο, στη διοικητική περιοχή Spaniaae στη σημερινή Ισπανία, πάνω από τις Ηράκλειες στήλες.



Εικόνα 11 Αρχή του παιχνιδιού

Εκεί είναι που θα πρέπει να υπερασπιστεί ο παίκτης για πρώτη φορά τα εδάφη της Βυζαντινής αυτοκρατορίας. Οι εχθροί στην Ισπανία είναι οι πιο εύκολοι σε όλο το παιχνίδι, όλες οι περιοχές έχουν στηθεί έτσι ώστε να είναι αυξανόμενης δυσκολίας. Το παιχνίδι έχει σχεδιαστεί επίτηδες έτσι ώστε να κατευθύνει τον παίκτη να παίξει πρώτα στην Ισπανία, να προχωρήσει προς τα ανατολικά μέσω των Βαλκανίων στην Ανατολία και μετέπειτα να παίξει στην Αφρικανική ήπειρο. Αν ο παίκτης ακολουθήσει αυτή την πορεία και αρχίσει από την Ισπανία, μετά παίζει την άνω και κάτω Ιταλία και μετά κατευθυνθεί προς τα Βαλκάνια θα μπορέσει να μαζέψει αρκετό XP χωρίς να χάσει, έτσι ώστε να ανέβει levels και να μπορέσει να νικήσει τις δύο μάχες. Αν παρόλο που το παιχνίδι αρχίζει στην περιοχή της Ισπανίας, ο παίκτης αποφασίσει να περάσει σε άλλες περιοχές πρώτα, τότε κατά πάσα πιθανότητα δεν θα μπορέσει να τις κερδίσει. Οι διοικητικές περιοχές των Βαλκανίων είναι πιο δύσκολες από την Ισπανία και την Ιταλία, οι εχθροί έρχονται με μεγαλύτερο ρυθμό και σε μεγαλύτερα νούμερα. Με το ίδιο μοτίβο, οι περιοχές της Ανατολίας είναι πιο δύσκολες από αυτές των Βαλκανίων και της Αφρικής οι δυσκολότερες.

Στην αρχική οθόνη μπορεί ο παίκτης να δει τα βασικά χαρακτηριστικά του παίκτη του. Ο παίκτης αρχίζει Level 1 με τη δυνατότητα να ρίχνει ένα βέλος το δευτερόλεπτο και 100 πόντους ζωής. Όσο ο παίκτης καταφέρνει και νικάει στις διάφορες

περιοχές επάνω στο χάρτη και μαζεύει XP τόσο ανεβαίνει η ζωή του και το fire rate του τόξου του.

Το Fire Rate και η ζωή του παίκτη διαμορφώνονται ως εξής ανά Level :

Πίνακας 1 Fire Rate και Ζωή ανά επίπεδο παίκτη

Level	1	2	3	4	5	6	7	8	9	10
Life	100	110	120	130	150	170	200	230	270	310
F.R.	1	1.05	1.1	1.2	1.25	1.35	1.5	1.8	2	3.5



Εικόνα 12 Στατιστικά παίκτη (Ζωή, Επίπεδο, XP, Fire Rate)

Ο παίκτης κινείται στον αρχικό χάρτη με τα κλασικά πλήκτρα για κίνηση “WASD”. Μπορεί επίσης να χρησιμοποιήσει και τα βελάκια αλλά δεν ενδείκνυται γιατί είναι μακριά από το αριστερό *Shift* και το *Space*. Για να αυξήσει λίγο την ταχύτητά του και να τρέξει μπορεί να χρησιμοποιήσει το αριστερό *Shift*. Όταν ο παίκτης εισέρχεται εντός των ορίων μιας περιοχής, στο πάνω μέρος της οθόνης, βγαίνει ένα μήνυμα που του λέει να πατήσει *Space* για να μπει να παίξει στην περιοχή. Εάν ο παίκτης έχει ήδη παίξει και έχει νικήσει στην περιοχή αυτή τότε το μήνυμα αλλάζει, του λέει πως έχει ήδη νικήσει και είναι με πράσινα γράμματα. Εάν από την άλλη έχει παίξει αλλά δεν κατάφερε να νικήσει τότε το μήνυμα παραμένει το ίδιο με το αρχικό «Press Space to Join “Name”»



Εικόνα 13 Μηνύματα περιοχής

Όταν ο παίκτης έρχεται εντός των ορίων μιας μάχης βγαίνει ένα παρόμοιο μήνυμα σε κόκκινα γράμματα, αλλά αυτή την φορά βγαίνει και ένα banner που μας λέει με ποιους έγινε η πραγματική μάχη και τη δύναμη της κάθε πλευράς. Επέλεξα να κάνω κόκκινα τα γράμματα των δύο σημαντικών μαχών για να είναι εμφανής η διαφορά τους από τις απλές περιοχές του παιχνιδιού.



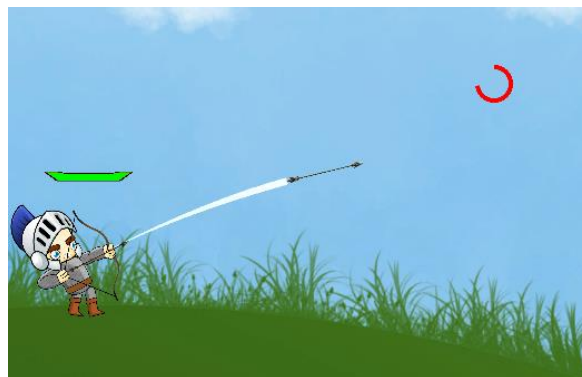
Εικόνα 14 Μηνύματα μαχών

Όταν ο παίκτης μπει για να παίξει σε μία από τις περιοχές το πρώτο που βλέπει είναι ο κάμπος απ' όπου θα έρχονται οι εχθροί, ο ουρανός ο οποίος είναι διαφορετικός ανάλογα με το μέρος που μπήκε να προστατέψει ο παίκτης και τα σύννεφα. Ακόμη ο παίκτης βλέπει έναν πράσινο κύκλο γύρω από το ποντίκι του, ο οποίος λειτουργεί σαν δείκτης που μετράει τον χρόνο που χρειάζεται για να μπορέσει το τόξο να ρίξει ξανά. Επίσης βλέπει τους εχθρούς να έρχονται σιγά – σιγά από τη δεξιά πλευρά της οθόνης και τον παίκτη που χειρίζεται στην αριστερή. Τέλος, βλέπει ένα Wave Counter στο κάτω μέρος την οθόνης όπου δείχνει σε πιο κύμα των εχθρών είμαστε και πόσα είναι συνολικά.



Εικόνα 15 Σκηνή μάχης

Κάθε φορά που κάνει ο παίκτης κλικ, ρίχνει ένα βέλος προς το σημείο που είναι το ποντίκι του. Το βέλος επηρεάζεται από τη βαρύτητα και ακολουθεί μια καμπύλη πορεία. Σαν εφέ υπάρχει μια άσπρη γραμμή να ακολουθεί το βέλος μέχρι να πέσει κάτω ή σε κάποιον από τους εχθρούς. Με το που φύγει το βέλος από το τόξο του παίκτη, αρχίζει το cooldown και ανάλογα το level που έχει ο παίκτης μπορεί να ξαναρίξει όταν τελειώσει.



Εικόνα 16 Βέλος και Cool Down / Fire Indicator

Το κάθε βέλος χτυπάει για 25 πόντους ζωής τους εχθρούς, οι εχθροί έχουν 100 πόντους ζωής, άρα ο κάθε εχθρός πρέπει να χτυπηθεί 4 φορές για να πεθάνει. Με τον τρόπο που έχω φτιάξει τους εχθρούς σε ένα σημείο τους ενώνονται 2 Colliders και συνεπώς εάν ο παίκτης καταφέρει να ρίξει εκεί το βέλος του κάνει ζημιά για 50 πόντους (25 + 25). Αυτή η δυνατότητα ξεκίνησε σαν bug, αλλά αποφάσισα να την κρατήσω.



Εικόνα 17 Χτύπημα εχθρού

Κάθε φορά που ένα βέλος πετυχαίνει έναν εχθρό, αίματα βγαίνουν προς τα πίσω και γεμίζουν το κάμπο. Στο τέλος κάθε κύματος εχθρών ο μετρητής στο κάτω μέρος της οθόνης αυξάνεται κατά ένα και όταν τελειώσει και το τελευταίο κύμα σημαίνει ότι ο παίκτης έχει καταφέρει να υπερασπιστεί με επιτυχία την περιοχή! Αν ο παίκτης δεν μπορέσει να σκοτώσει όλους τους εχθρούς και πεθάνει, τότε σημαίνει ότι έχασε και δεν κατάφερε να υπερασπιστεί την περιοχή. Και στις δύο περιπτώσεις, στο τέλος του παιχνιδιού βγαίνει ένα μήνυμα στο κέντρο της οθόνης για να κάνει ξεκάθαρο στον παίκτη αν έχει νικήσει ή αν έχει χάσει. Στην περίπτωση της νίκης, βγαίνει ένα μήνυμα που με πράσινα γράμματα γράφει Victory και στην περίπτωση της ήττας βγαίνει με κόκκινα γράμματα και γράφει You Died.



Εικόνα 18 Μήνυμα νίκης / ήττας

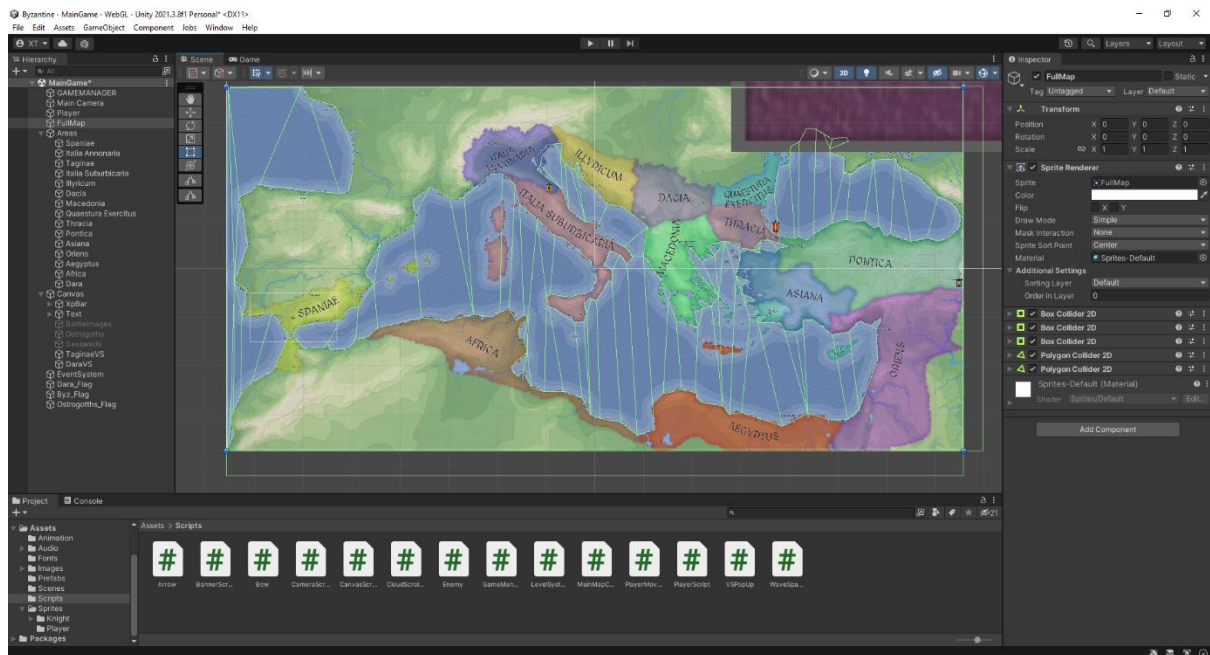
Εάν ο παίκτης καταφέρει να νικήσει στην περιοχή, τότε επιστρέφει στον κεντρικό χάρτη και του προστίθενται XP ώστε να ανέβει level, και η περιοχή πλέον θεωρείται ήδη νικημένη και δεν μπορεί να μπει ξανά. Αν ο παίκτης χάσει και πεθάνει, τότε απλά γυρνάει στον αρχικό χάρτη χωρίς να πάρει καθόλου XP και μπορεί να παίξει ξανά ανά πάσα στιγμή την ίδια περιοχή ώστε να καταφέρει να την κερδίσει.



Εικόνα 19 Επιστροφή στον αρχικό χάρτη

6) Τεχνικά

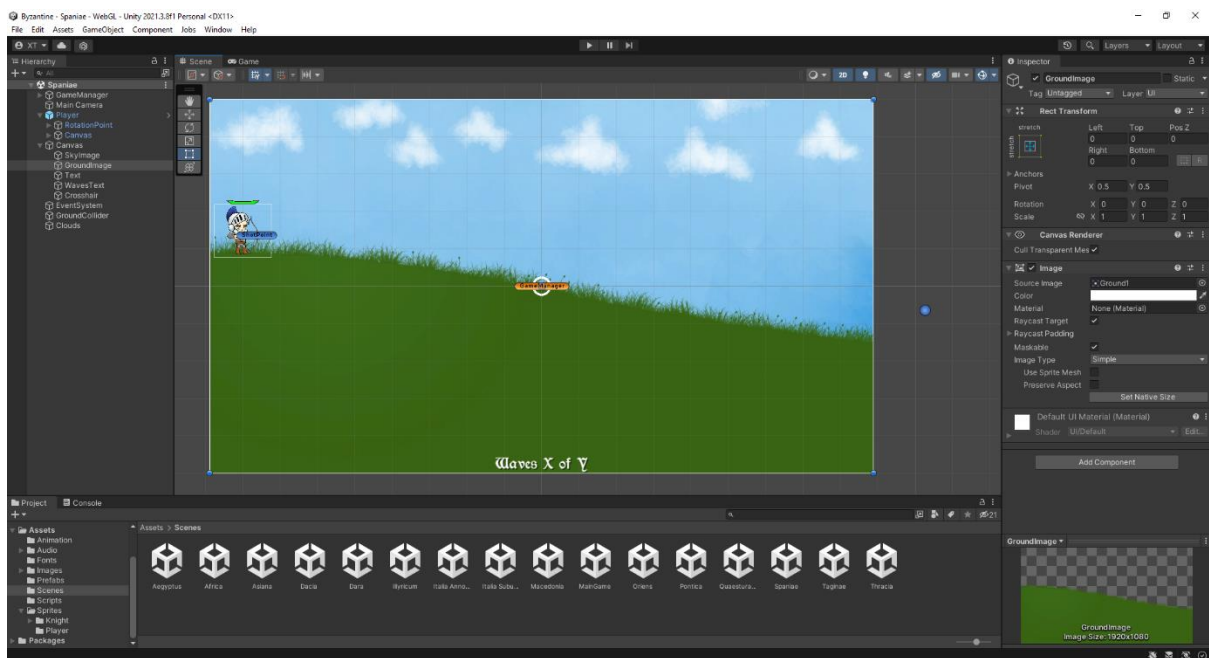
Στο Unity έχουν στηθεί όλα στη δικιά τους σκηνή, δηλαδή ο αρχικός χάρτης είναι μία σκηνή από μόνος του, οι 13 περιοχές όπου παίζει ο παίκτης είναι και αυτές 13 διαφορετικές σκηνές και τέλος οι δύο μάχες είναι και αυτές φτιαγμένες στις δικές τους διαφορετικές σκηνές. Στην αρχική σκηνή του χάρτη, έχουμε 10 Objects που το καθένα παίζει τον ρόλο του. Αρχικά έχει δημιουργηθεί ένας Game Manager, ο οποίος έχει επάνω του το Script για το Level System, δηλαδή χειρίζεται το επίπεδο του παίκτη. Επίσης επάνω στο Game Manager έχει τοποθετηθεί και το Audio Source που παίζει το αρχικό Soundtrack του παιχνιδιού. Μετά έχουμε την κάμερα του παιχνιδιού, πάνω στην οποία υπάρχει το Script που την κάνει να ακολουθεί έναν στόχο. Στην περίπτωση μας ακολουθεί τον παίκτη. Επίσης, το Script έχει παραμέτρους για το πόσο πάνω-κάτω-δεξιά-αριστερά μπορεί να πάει η κάμερα και πόσο ομαλά να κινείται. Μετά έχουμε το Object του παίκτη, το οποίο έχει επάνω του το Script που είναι υπεύθυνο για την κίνησή του. Το Script αυτό κάνει τον παίκτη να περπατάει, να τρέχει και είναι υπεύθυνο για να ελέγχει εάν είναι μέσα σε κάποια περιοχή. Την ακριβή επεξήγηση του πώς δουλεύει θα την δούμε σε επόμενο κεφάλαιο. Μετά, έχουμε τον χάρτη που ουσιαστικά είναι ένα Sprite στο οποίο έχει αφαιρεθεί οποιαδήποτε συμπίεση για να έχει τη μέγιστη δυνατή ποιότητα. Επάνω του ο χάρτης έχει τους colliders που σταματάνε τον παίκτη από το να βγει από τα όρια του χάρτη και να περπατάει επάνω στην θάλασσα. Έπειτα έχουμε μια λίστα αντικειμένων που έχουν όλα επάνω τους έναν collider, αυτόν της κάθε περιοχής και μάχης που μπορεί να παίζει ο παίκτης. Στην συνέχεια υπάρχει το Object του Canvas και του Event System που είναι και τα δύο για να λειτουργεί το HUD. Επάνω του ο καμβάς έχει ένα script το οποίο είναι υπεύθυνο να αλλάζει τα πάντα επάνω στην οθόνη του παίκτη. Συγκεκριμένα αλλάζει το XP, το Life, το Level, το Fire Rate, δείχνει επάνω αριστερά τις δύο μάχες, αν τις έχει κερδίσει ο παίκτης και τέλος είναι υπεύθυνο για να δείχνει επάνω το μήνυμα για να μπει ο παίκτης στις περιοχές. Τελευταίες είναι οι εικόνες που υπάρχουν ως Banners δίπλα από τις δύο μάχες και δίπλα από την Κωνσταντινούπολη. Αυτές οι εικόνες έχουν επάνω τους ένα πολύ μικρό Script το οποίο τις κάνει να κινούνται δεξιά-αριστερά.



Εικόνα 20 Αρχική σκηνή στο Editor του Unity

Οι 13 σκηνές των περιοχών και οι 2 των μαχών είναι σχεδόν ίδιες μεταξύ τους. Στο ψηλότερο σημείο της ιεραρχίας υπάρχει και πάλι ένας Game Manager, ο οποίος έχει επάνω του δύο Scripts. Το ένα είναι το Wave Spawner, το οποίο – όπως λέει και το όνομά του – είναι υπεύθυνο για να βγάξει τα κύματα των εχθρών. Ανάλογα την περιοχή, έχουν ρυθμιστεί οι παράμετροι του Wave Spawner έτσι ώστε να έρχονται όσοι εχθροί επιθυμούμε και όσο κοντά ο ένας στον άλλο χρειάζεται για να επιτευχθεί η επιθυμητή δυσκολία. Το άλλο Script παίζει ένα τυχαίο Soundtrack από μία μικρή λίστα. Ακολουθεί στην ιεραρχία η κάμερα, η οποία παραμένει σταθερή για τις σκηνές των μαχών και δεν έχει κάποιο Script επάνω της. Πιο κάτω στη ιεραρχία, υπάρχει το Prefab του παίκτη μας. Καθώς οι ιδιότητες του παίκτη είναι οι ίδιες σε όλες τις σκηνές που γίνονται οι μάχες, επιλέχθηκε να είναι Prefab. Αυτό σημαίνει ότι ό,τι αλλαγές γίνουν στο Prefab του παίκτη εφαρμόζονται αυτομάτως σε όλες τις σκηνές. Στο Object του παίκτη υπάρχει ένα Script που φροντίζει κάποια στατιστικά του: υπολογίζει πόση ζωή έχει ανάλογα με το Level στο οποίο είναι και πόση ζημιά του έχουν κάνει οι εχθροί, δείχνει την μπάρα επάνω του, για να φαίνεται και γραφικά η ζωή του παίκτη, κοιτάει αν έχουν τελειώσει οι εχθροί για να τελειώσει η μάχη και τέλος ελέγχει εάν η ζωή του παίκτη είναι μηδέν ή λιγότερη και άρα έχει πεθάνει. Κάτω από το Object του παίκτη κρύβονται ορισμένα ακόμη Objects, τα λεγόμενα παιδιά του. Ένα από τα δύο παιδιά του είναι ένας Canvas όπου έχει τα Sprites της μπάρας ζωής του παίκτη και το άλλο είναι το σημείο από το οποίο γυρνάει ο παίκτης για να ρίξει τα βέλη του. Το σημείο που γυρνάει ο παίκτης, το Rotation Point, έχει ένα Script το οποίο είναι κυρίως υπεύθυνο για την λειτουργία του τόξου. Υπολογίζει το Fire Rate ανάλογα με το Level του παίκτη, τη

δύναμη του τόξου, ανάλογα αν έχει κερδίσει κάποια μάχη ο παίκτης, κάνει το τόξο να κοιτάει στον κέρσορα και φυσικά να ρίχνει ένα βέλος όταν ο παίκτης κάνει κλικ. Πιο κάτω στην ιεραρχία της σκηνής μας είναι ο κυρίως Canvas της σκηνής μαζί με το Event System του. Αυτός, έχει σαν παιδιά του τον ουρανό, την πλαγιά, το Text που λέει πόσα κύματα εχθρών έχουν μείνει, το Text που γράφει Victory ή You Died στο τέλος έχει τον κύκλο ή αλλιώς Fire Indicator, που δείχνει και οπτικά αν το τόξο είναι διαθέσιμο για να ρίξει το επόμενο βέλος. Όλα αυτά εκτός των δύο Texts είναι Sprites. Έχει επάνω του ένα Script που είναι υπεύθυνο για το τελικό Text της μάχης, το οποίο δείχνει αν ο παίκτης νίκησε ή έχασε, τη συνάρτηση του όπου κάνει τον κύκλο του Fire Indicator να ακολουθεί τον κέρσορα του ποντικιού και το μέτρημα των κυμάτων των εχθρών στο κάτω μέρος την οθόνης. Πιο κάτω στην ιεραρχία υπάρχει ένα κενό Object, το οποίο έχει το Collider του εδάφους. Τέλος, έχουμε τα σύννεφα τα οποία είναι ένα στατικό Sprite και έχουν επάνω τους ένα Script, το οποίο, στην αρχή της σκηνής, τους δίνει μία τυχαία ταχύτητα είτε προς τα αριστερά είτε προς τα δεξιά.

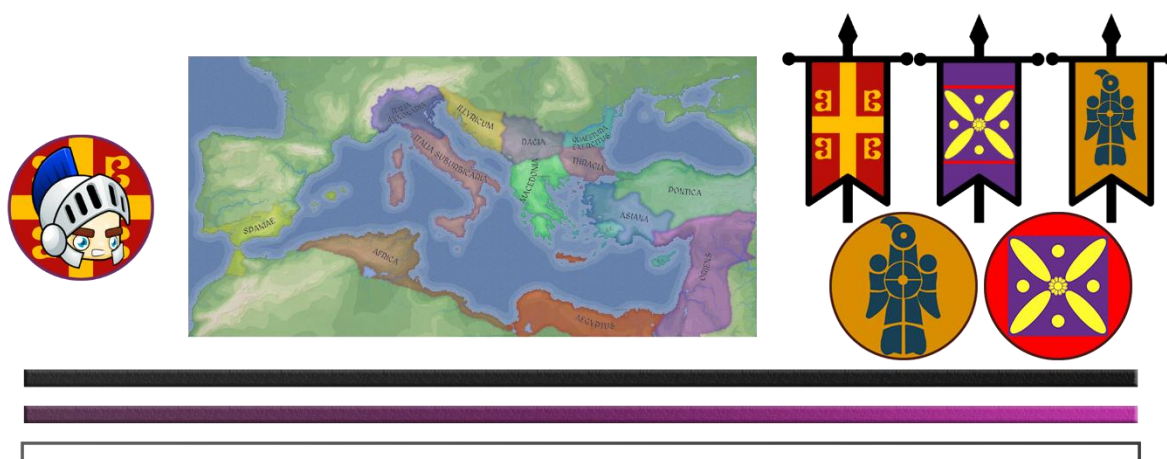


Εικόνα 21 Σκηνή μάχης στο Editor του Unity

7) Sprites

7.1) Sprites αρχικού χάρτη

Αρχικά υπάρχει ο παίκτης, ο οποίος δημιουργήθηκε στο Photoshop παίρνοντας το πρόσωπο που έχει ο παίκτης εντός των μαχών και βάζοντάς το επάνω σε μία σημαία του Βυζαντίου που είχα φτιάξει μόνος μου. Ο χάρτης είναι φτιαγμένος με το Fantasy Map Maker⁸ το οποίο είναι διαθέσιμο online και έχουν σημειωθεί επάνω του μία-μία οι περιοχές και οι πόλεις. Περνώντας τώρα στις σημαίες, η σημαία του Βυζαντίου που υπάρχει δίπλα από την Κωνσταντινούπολη έχει φτιαχτεί στο Photoshop και είναι η ίδια που χρησιμοποιείται ως φόντο στον παίκτη. Η σημαία των Σασσανιδών είναι διαθέσιμη στο internet⁹ όπως και αυτή των Οστρογόθων¹⁰. Αυτές τις δύο σημαίες χρησιμοποιήθηκαν και στο πάνω αριστερό μέρος για να δείξουν ότι ο παίκτης έχει νικήσει τις μάχες. Τέλος τα Sprites που δείχνουν το επίπεδο του παίκτη είναι τρία, το πίσω Sprite που είναι ασπρόμαυρο, το μπροστά που έχει το χρώμα «Μωβ του Βυζαντίου» και από μπροστά από αυτά τα δύο βρίσκεται το περίγραμμα. Και οι τρεις αυτές εικόνες έχουν δημιουργηθεί εξ ολοκλήρου στο Photoshop.



Εικόνα 22 Sprites αρχικού χάρτη

7.2) Sprites εντός μαχών

Για αρχή έχουμε το Sprite του παίκτη. Το Sprite έχει δημιουργηθεί παίρνοντας έτοιμο το κεφάλι από το πακέτο με τους εχθρούς και ζωγραφίζοντας το σώμα στο Illustrator. Για να είναι ο παίκτης λίγο διαφορετικός από τους εχθρούς έχει διαφορετικό χρώμα ματιών και χρώμα περικεφαλαίας. Μετά έχουμε τους εχθρούς οι οποίοι

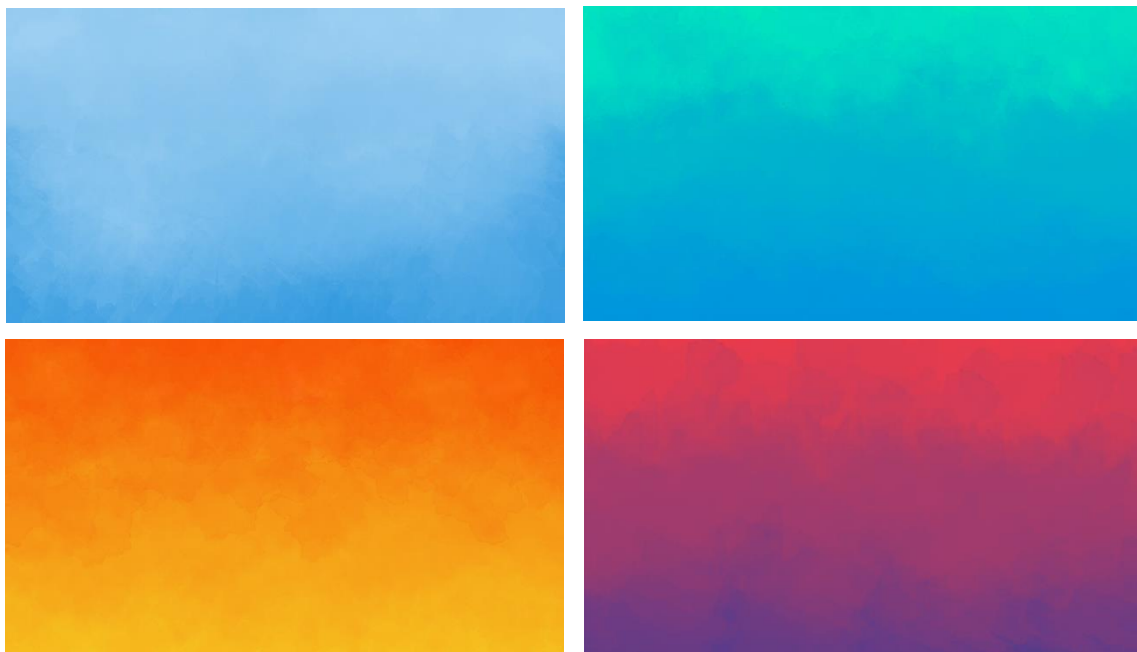
⁸ azgaar.github.io/Fantasy-Map-Generator [Τ.Η.Π 22-9-2022]

⁹ wikipedia.com/Sassanid-Empire-flag.png [Τ.Η.Π 22-9-2022]

¹⁰ Reddit.com/Old-Kingdom-Flags.png [Τ.Η.Π 22-9-2022]

προέρχονται από ένα έτοιμο πακέτο ανοικτών Sprites¹¹ για χρήση σε παιχνίδια. Το Sprite του Cool Down Counter, είναι ένας απλός άσπρος κύκλος ο οποίος δημιουργήθηκε στο Photoshop και χρωματίζεται πράσινος ή κόκκινος ανάλογα με το αν μπορεί να ρίξει ο παίκτης μέσω κώδικα στο Unity.

Δημιουργήθηκαν τέσσερις διαφορετικοί ουρανοί και έχουν χωριστεί με την εξής σειρά: οι περιοχές Spaniae, Italia Annonaria, Italia Suburbicaria, Illyricum και Dacia έχουν τον πρώτο, γαλάζιο ουρανό. Οι περιοχές Macedonia, Quaestura Exercitus, Thracia, Pontica και Asiana έχουν το δεύτερο ουρανό, έναν ουρανό που τείνει προς το γαλαζοπράσινο. Στις τελευταίες τρεις περιοχές επιλέχθηκε ένα διαφορετικό χρώμα και έτσι οι περιοχές Africa, Aegyptus και Oriens έχουν έναν πορτοκαλί ουρανό για να δίνουν την αίσθηση ότι έχουμε φύγει από Ευρώπη και πλέον είμαστε στην εγγύς Ανατολή και την Αφρική. Τέλος οι δύο μάχες να διαδραματίζονται σε έναν σκούρο φόντο, με τον τελευταίο ουρανό να έχει το χρώμα του ανοικτού μωβ και προς τα πάνω να τείνει προς το κόκκινο.



Εικόνα 23 Διαφορετικοί ουρανοί

Τα σύννεφα είναι μία στατική η οποία κάνει loop γύρω-γύρω για να δίνει μία αίσθηση ότι υπάρχει άνεμος. Κάθε φορά που ο παίκτης μπαίνει να παίξει σε ένα από τα μέτωπα υπάρχει μια συνάρτηση που κάνει τα σύννεφα να κινούνται τυχαία προς τα δεξιά ή προς τα αριστερά.

¹¹ gameart2d.com/the-knight-free-sprites.html [Τ.Η.Π 22-9-2022]



Εικόνα 24 Σύννεφα

Οι κάμποι είναι διαφορετικοί σε όλα τα παιδιά των μαχών. Σχεδιάστηκαν εξ αρχής στο Photoshop και με τη βοήθεια κάποιων brushes τοποθετήθηκε και το γρασίδι. Έγινε προσπάθεια να είναι αυξανόμενης δυσκολίας, δηλαδή όσο πιο πέρα προχωράει ο παίκτης στο παιχνίδι τόσο πιο απότομοι να είναι. Έτσι να είναι σταδιακά πιο δύσκολο να πετύχει τους εχθρούς μιας και θα είναι πιο δύσκολο να προβλέψει την πορεία του βέλους συναρτήσει της αλλαγής του εδάφους.



Εικόνα 25 Έδαφος

8) Κώδικας και σχολιασμός του

Σε αυτό το κεφάλαιο θα δούμε αναλυτικά πώς δουλεύει το καθένα από τα Scripts που έχουν δημιουργηθεί. Θα αρχίσουμε από τα Scripts του αρχικού χάρτη και μετέπειτα θα εμβαθύνουμε σε αυτά εντός των μαχών. Αρχίζουμε από το Object Game Manager εντός του αρχικού χάρτη. Μέσα στο Game Manager υπάρχει το Script «LevelSystem».

LevelSystems.cs

```
using UnityEngine;
using UnityEngine.UI;

//Main map script

public class LevelSystem : MonoBehaviour {
    [ Header ( "Variables" ) ]
    [ HideInInspector ] public int currentXp;
    [ HideInInspector ] public int requiredXp;
    public static int level;

    private float lerpTimer;
    private float delayTimer;

    [ Header ( "UI" ) ]
    [ SerializeField ] private Image frontXpBar;
    [ SerializeField ] private Image backXpBar;

    public static int xpOnWin;
```

Βλέπουμε στην αρχική συνάρτηση **Start();** πώς τρέχουμε μερικές συναρτήσεις παρακάτω. Κατά σειρά : η **PlayerPrefChecker();** κοιτάζει στη μνήμη εάν ο παίκτης έχει ξαναπαιξει το παιχνίδι. Αν ο παίκτης έχει παίξει παλαιότερα, τότε αυτή η συνάρτηση είναι υπεύθυνη για να πάρει από τη μνήμη το Level όπως και το XP του παίκτη. Μετά, η **CalculateRequiredXp();** είναι υπεύθυνη για τον υπολογισμό του XP που χρειάζεται ο παίκτης για να πάει στο επόμενο επίπεδο. Χρησιμοποιεί το Level που είναι τώρα ο παίκτης για να μας γυρίσει ένα αποτέλεσμα. Η **SetXpFillAmount();** για αρχή κάνει τις μπάρες του XP να είναι τελείως άδειες. Η **GainExperienceFlatRate (xpOnWin);** είναι υπεύθυνη για να αυξάνει το επίπεδο του παίκτη μετά από μια νίκη του σε κάποιο από τα μέτωπα. Η **xpOnWin** είναι μία καθολική μεταβλητή στην οποία καταχωρείται μία τιμή ανάλογα με το ποια μάχη παίζουμε. Έτσι, όταν ο παίκτης γυρνάει πάλι στον αρχικό χάρτη, αυτή η συνάρτηση του αυξάνει το XP.

```
private void Start ( ) {
    PlayerPrefChecker ( );
    CalculateRequiredXp ( );
    SetXpFillAmount ( );
    GainExperienceFlatRate ( xpOnWin );
}
```

Παρακάτω, η συνάρτηση **Update()**; καλεί τη συνάρτηση **UpdateXpUI()**; Αυτή η συνάρτηση γεμίζει σιγά-σιγά τις μπάρες του XP όταν ο παίκτης κερδίζει. Έχει φτιαχτεί έτσι ώστε να γεμίζει πρώτα την πίσω, ασπρόμαυρη εικόνα κατευθείαν και μετά από μισό δευτερόλεπτο να γεμίζει και την μπροστά εικόνα.

```
private void Update ( ) {
    UpdateXpUI ( );
}

private void SetXpFillAmount ( ) {
    frontXpBar.fillAmount = currentXp / requiredXp;
    backXpBar.fillAmount = currentXp / requiredXp;
}

private void PlayerPrefChecker ( ) {
    if ( PlayerPrefs.HasKey ( "xp" ) == true ) {
        currentXp = PlayerPrefs.GetInt( "xp" );
    }
    else {
        currentXp = 0;
    }
    if ( PlayerPrefs.HasKey( "lvl" ) == true ) {
        level = PlayerPrefs.GetInt( "lvl" );
    }
    else {
        level = 1;
    }
}

public void UpdateXpUI ( ) {
    float xpFraction = ( float ) currentXp / ( float ) requiredXp;
    float FXP = frontXpBar.fillAmount;

    if ( level < 10 ) {
        if ( FXP < xpFraction ) {
            delayTimer = delayTimer + Time.deltaTime;
            backXpBar.fillAmount = xpFraction;
            if ( delayTimer > 0.5 )
            {
                lerpTimer = lerpTimer + Time.deltaTime;
                float percentComplete = lerpTimer / 50;
                frontXpBar.fillAmount = Mathf.Lerp ( FXP, backXpBar.fillAmount, percentComplete );
            }
        }
    }
    else {
        frontXpBar.fillAmount = 1;
    }
}

public void GainExperienceFlatRate ( int xpGained ) {
    currentXp = currentXp + xpGained;
    xpOnWin = 0;
    lerpTimer = 0f;
    delayTimer = 0f;
    PlayerPrefs.SetInt ( "xp", currentXp );
    if ( currentXp >= requiredXp ) {
        LevelUp ( );
    }
}
```

Η παρακάτω συνάρτηση, **LevelUp()**; καλείται μέσα από την **GainExperienceFlatRate()**; όταν ο παίκτης φτάσει ή ξεπεράσει το XP που χρειάζεται για να ανέβει επίπεδο.

```
private void LevelUp ( ) {
    level++;
    frontXpBar.fillAmount = 0f;
}
```

```

        backXpBar.fillAmount = 0f;
        currentXp = Mathf.RoundToInt ( currentXp - requiredXp );
        CalculateRequiredXp ( );

        PlayerPrefs.SetInt ( "lvl", level );
        PlayerPrefs.SetInt ( "xp", currentXp );
    }

    private void CalculateRequiredXp ( ) {
        if ( level == 1 ) {
            requiredXp = 25;
        }
        else if ( level == 2 ) {
            requiredXp = 51;
        }
        else if ( level == 3 ) {
            requiredXp = 73;
        }
        else if ( level == 4 ) {
            requiredXp = 104;
        }
        else if ( level == 5 ) {

            requiredXp = 131;
        }
        else if ( level == 6 ) {
            requiredXp = 160;
        }
        else if ( level == 7 ) {
            requiredXp = 189;
        }
        else if ( level == 8 ) {
            requiredXp = 218;
        }
        else if ( level == 9 ) {
            requiredXp = 249;
        }
        Else {
            requiredXp = 267;
        }
    }
}

```

Το Object Main Camera είναι η κάμερα του παιχνιδιού μας και επάνω της έχει ένα Script με όνομα «CameraScript».

CameraScript.cs

```

using UnityEngine;

public class CameraScript : MonoBehaviour {
    [ Header ( "References" ) ]
    [ SerializeField ] private Transform target;
    [ SerializeField ] private Rigidbody2D rb;

    [ Header ( "Variables" ) ]
    [ Range ( 1 , 50 ) ]
    [ SerializeField ] private float smoothing;
    [ Tooltip ( "Bottom left" ) ]
    [ SerializeField ] private Vector2 minPos;
    [ Space ( 15 ) ]
    [ Tooltip ( "Top right" ) ]
    [ SerializeField ] private Vector2 maxPos;
}

```

Για αρχή το Script ελέγχει αν ο παίκτης έχει κάποια αποθηκευμένη θέση επάνω στον χάρτη. Αν έχει τότε βάζει την κάμερα σε αυτό το σημείο, αλλιώς πάει στο αρχικό σημείο που αρχίζει το παιχνίδι.

```

private void Start ( ) {

```

```

        if ( PlayerPrefs.HasKey ( "X" ) == true || PlayerPrefs.HasKey ( "Y" ) == true ) {
            transform.position = new Vector3 ( PlayerPrefs.GetFloat ( "X" ), PlayerPrefs.GetFloat ( "Y" ), -10 );
        }
        Else {
            transform.position = new Vector3 ( -62f, -10f, -10 );
        }
    }
}

```

Εντός της **FixedUpdate();** είναι ο κώδικας με τον οποίο η κάμερά μας ακολουθεί τον παίκτη. Έχουμε ένα Reference στο Object του παίκτη μας, το οποίο ακολουθείται από την κάμερα, μέχρι κάποια σημεία. Εντός του Editor στο Unity έχουμε δώσει τις μέγιστες συντεταγμένες που μπορεί να πάει η κάμερα. Αυτό έγινε για να μην βγαίνει εκτός χάρτη όσο κοντά και να πάει ο παίκτης στην άκρη του. Τέλος, η κίνηση της κάμερας γίνεται με τη βοήθεια μιας μεταβλητής με όνομα «Smoothing». Αυτή η μεταβλητή χρησιμοποιείται έτσι ώστε η κάμερά μας να κάνει ομαλές κινήσεις και να μην κουνιέται απότομα.

```

private void FixedUpdate ( ) {
    if ( transform.position != target.position ) {
        Vector3 targetPos = new Vector3 ( target.position.x, target.position.y, transform.position.z );

        targetPos.x = Mathf.Clamp ( targetPos.x, minPos.x, maxPos.x );
        targetPos.y = Mathf.Clamp ( targetPos.y, minPos.y, maxPos.y );

        transform.position = Vector3.Lerp ( transform.position, targetPos, smoothing / 100 );
    }
}
}

```

Περνάμε έπειτα στο Object Player που αντιστοιχεί στον παίκτη και έχει επάνω του ένα Script με όνομα «PlayerMovement».

PlayerMovement.cs

```

using UnityEngine;
using UnityEngine.SceneManagement;

//Main scene script

public class PlayerMovement : MonoBehaviour {
    [ Header ( "Variables" ) ]
    [ SerializeField ] private float moveSpeed = 5f;
    [ SerializeField ] private float runSpeed = 8f;

    public static string location = null;

    private Rigidbody2D rigidBody;
    private AudioSource declineSound;
    private Vector2 movement;
    private float finalSpeed;
}

```

Βλέπουμε ότι η αρχική συνάρτηση **Start();** - η οποία καλείται την πρώτη φορά που τρέχει το Script - καλεί με τη σειρά της άλλες δύο συναρτήσεις. Την **LoadPosition();** και την **Initializations();** Η **LoadPosition();** κάνει το ίδιο πράγμα που γίνεται στο Script της κάμερας στην αρχή. Κοιτάει αν υπάρχει αποθηκευμένη θέση του παίκτη από

προηγούμενη φορά που έχει παίξει, και αν υπάρχει τον πηγαίνει εκεί. Ειδικά τον μεταφέρει στην αρχική θέση του παιχνιδιού. Έπειτα η **Initializations();** καλείται για κάποιες αρχικοποιήσεις. Συνδέουμε κάποια Components μέσα στο Unity με κάποιες μεταβλητές έτσι ώστε να τα χρησιμοποιήσουμε στη συνέχεια.

```
private void Start ( ) {
    LoadPosition ( );
    Initializations ( );
}

void Initializations ( ) {
    GetComponent<TrailRenderer>().enabled = true;
    rigidBody = GetComponent<Rigidbody2D>( );
    declineSound = GetComponent<AudioSource>( );
    finalSpeed = moveSpeed;
}
```

Εντός της **Update();** η οποία εκτελείται σε κάθε καρτέ του παιχνιδιού μας, καλούνται τέσσερις συναρτήσεις διαρκώς. Η **MovementRead();** η οποία διαβάζει τα κουμπιά που πατάει ο παίκτης για να μετακινηθεί πάνω στον χάρτη. Η **LocationUpdate();** η οποία ελέγχει αν είμαστε εντός κάποιας περιοχής, αν έχουμε παίξει και ήδη νικήσει στην περιοχή και τέλος είναι υπεύθυνη για να μας πάει στην επόμενη σκηνή, την σκηνή της μάχης. Η **Run();** είναι η συνάρτηση η οποία αν ο παίκτης πατάει το αριστερό *Shift*, αυξάνει την ταχύτητά του. Τέλος, η **Save();** αποθηκεύει διαρκώς τη θέση του παίκτη μας, έτσι ώστε αν κλείσει το παιχνίδι ή μπει σε κάποια μάχη, όταν ξαναφορτώσει η κύρια σκηνή με τον χάρτη να επιστρέφει στο ίδιο σημείο.

```
private void Update ( ) {
    MovementRead ( );
    LocationUpdate ( );
    Run ( );
    Save ( );
}

private void FixedUpdate ( ) {
    rigidBody.velocity = new Vector2 ( movement.x * finalSpeed, movement.y * finalSpeed );
}
```

Τέλος για αυτό το Script, χρησιμοποιήθηκαν οι έτοιμες συναρτήσεις του Unity για τα Colliders, έτσι ώστε να υπάρχει μία καθολική μεταβλητή location, η οποία παίρνει το όνομα της περιοχής που είναι μέσα ο παίκτης. Σε συνδυασμό με την **LocationUpdate();** που παρουσιάστηκε προηγουμένως τοποθετούν τον παίκτη στη σωστή μάχη. Επίσης όταν ο παίκτης εξέρχεται από κάποιο Collider περιοχής, η μεταβλητή location γίνεται null και έτσι ο παίκτης δεν μπαίνει άθελά του σε περιοχές που μπορεί να μην ήθελε.

```
private void OnTriggerEnter2D ( Collider2D collider ) {
    if ( collider.name == "Macedonia" ) {
```

```

        location = "Macedonia";
    }
    else if ( collider.name == "Thracia" ) {
        location = "Thracia";
    }
    else if ( collider.name == "Dacia" ) {
        location = "Dacia";
    }
    else if ( collider.name == "Quaestura Exercitus" ) {
        location = "Quaestura Exercitus";
    }
    else if ( collider.name == "Illyricum" ) {
        location = "Illyricum";
    }
    else if ( collider.name == "Italia Annonaria" ) {
        location = "Italia Annonaria";
    }
    else if ( collider.name == "Italia Suburbicaria" ) {
        location = "Italia Suburbicaria";
    }
    else if ( collider.name == "Pontica" ) {
        location = "Pontica";
    }
    else if ( collider.name == "Asiana" ) {
        location = "Asiana";
    }
    else if ( collider.name == "Oriens" ) {
        location = "Oriens";
    }
    else if ( collider.name == "Aegyptus" ) {
        location = "Aegyptus";
    }
    else if ( collider.name == "Africa" ) {
        location = "Africa";
    }
    else if ( collider.name == "Spaniae" ) {
        location = "Spaniae";
    }
    else if ( collider.name == "Dara" ) {
        location = "Dara";
    }
    else if ( collider.name == "Taginae" ) {
        location = "Taginae";
    }
    }
}

private void OnTriggerExit2D ( Collider2D collider ) {
    location = null;
}

private void MovementRead ( ) {
    movement.x = Input.GetAxis( "Horizontal" );
    movement.y = Input.GetAxis( "Vertical" );
}

private void Run ( ) {
    if ( Input.GetKeyUp ( KeyCode.LeftShift ) ) {
        finalSpeed = moveSpeed;
    }
    if ( Input.GetKeyDown ( KeyCode.LeftShift ) ) {
        finalSpeed = runSpeed;
    }
}

private void LoadPosition ( ) {
    if ( PlayerPrefs.HasKey( "X" ) == true || PlayerPrefs.HasKey( "Y" ) == true ) {
        transform.position = new Vector3 ( PlayerPrefs.GetFloat ( "X" ), PlayerPrefs.GetFloat ( "Y" ), 0 );
    }
    else {
        transform.position = new Vector3 ( -62f, -10f, 0 );
    }
}

private void Save ( ) {
    PlayerPrefs.SetFloat( "X", transform.position.x );
    PlayerPrefs.SetFloat( "Y", transform.position.y );
}

private void LocationUpdate ( ) {
    if ( Input.GetKey ( KeyCode.Space ) ) {
        if ( location == "Spaniae" ) {
            if ( MainMapCanvasScript.hasWonSpaniae ) {
                if ( declineSound.isPlaying == false ) {
                    declineSound.Play ( );
                }
            }
        }
    }
}

```

```

    }
    else {
        SceneManager.LoadScene ( 1 );
    }
}
if ( location == "Italia Annonaria" ) {
    if ( MainMapCanvasScript.hasWonItaliaAnnonaria ) {
        if ( declineSound.isPlaying == false ) {
            declineSound.Play ( );
        }
    }
    else {
        SceneManager.LoadScene ( 2 );
    }
}
if ( location == "Italia Suburbicaria" ) {
    if ( MainMapCanvasScript.hasWonItaliaSuburbicaria ) {
        if ( declineSound.isPlaying == false ) {
            declineSound.Play ( );
        }
    }
    else {
        SceneManager.LoadScene ( 3 );
    }
}
if ( location == "Illyricum" ) {
    if ( MainMapCanvasScript.hasWonIllyricum ) {
        if ( !declineSound.isPlaying ) {
            declineSound.Play ( );
        }
    }
    else {
        SceneManager.LoadScene ( 4 );
    }
}
if ( location == "Dacia" ) {
    if ( MainMapCanvasScript.hasWonDacia ) {
        if ( declineSound.isPlaying == false ) {
            declineSound.Play ( );
        }
    }
    else {
        SceneManager.LoadScene ( 5 );
    }
}
if ( location == "Macedonia" ) {
    if ( MainMapCanvasScript.hasWonMacedonia ) {
        if ( declineSound.isPlaying == false ) {
            declineSound.Play ( );
        }
    }
    else {
        SceneManager.LoadScene ( 6 );
    }
}
if ( location == "Quaestura Exercitus" ) {
    if ( MainMapCanvasScript.hasWonQuaesturaExercitus ) {
        if ( declineSound.isPlaying == false ) {
            declineSound.Play ( );
        }
    }
    else {
        SceneManager.LoadScene ( 7 );
    }
}
if ( location == "Thracia" ) {
    if ( MainMapCanvasScript.hasWonThracia ) {
        if ( declineSound.isPlaying == false ) {
            declineSound.Play ( );
        }
    }
    else {
        SceneManager.LoadScene ( 8 );
    }
}
if ( location == "Pontica" ) {
    if ( MainMapCanvasScript.hasWonPontica ) {
        if ( declineSound.isPlaying == false ) {
            declineSound.Play ( );
        }
    }
    else {
        SceneManager.LoadScene ( 9 );
    }
}
if ( location == "Asiana" ) {

```

```

        if ( MainMapCanvasScript.hasWonAsiana ) {
            if ( declineSound.isPlaying == false ) {
                declineSound.Play ();
            }
        }
        else {
            SceneManager.LoadScene ( 10 );
        }
    }
    if ( location == "Oriens" ) {
        if ( MainMapCanvasScript.hasWonOriens ) {
            if ( declineSound.isPlaying == false ) {
                declineSound.Play ();
            }
        }
        else {
            SceneManager.LoadScene ( 11 );
        }
    }
    if ( location == "Aegyptus" ) {
        if ( MainMapCanvasScript.hasWonAegyptus ) {
            if ( declineSound.isPlaying == false ) {
                declineSound.Play ();
            }
        }
        else {
            SceneManager.LoadScene ( 12 );
        }
    }
    if ( location == "Africa" ) {
        if ( MainMapCanvasScript.hasWonAfrica ) {
            if ( declineSound.isPlaying == false ) {
                declineSound.Play ();
            }
        }
        else {
            SceneManager.LoadScene ( 13 );
        }
    }
    if ( location == "Dara" ) {
        if ( MainMapCanvasScript.hasWonSassanids ) {
            if ( declineSound.isPlaying == false ) {
                declineSound.Play ();
            }
        }
        else {
            SceneManager.LoadScene ( 14 );
        }
    }
    if ( location == "Taginae" ) {
        if ( MainMapCanvasScript.hasWonOstrogoths ) {
            if ( declineSound.isPlaying == false ) {
                declineSound.Play ();
            }
        }
        else {
            SceneManager.LoadScene ( 15 );
        }
    }
}
}
}
}
}

```

Στη συνέχεια παρουσιάζονται τα δύο Object που έχουν επάνω τους τα Colliders για τις δύο μάχες. Και τα δύο αυτά Objects έχουν επάνω τους ένα Script που παρουσιάζει το Banner με τις πληροφορίες των δυνάμεων της πραγματικής μάχης. Το Script λέγεται «VSPopUp» και είναι λιγότερο από είκοσι γραμμές.

VSPopUp.cs

```
using UnityEngine.UI;
using UnityEngine;

public class VSPopUp : MonoBehaviour {
    [SerializeField] private Image image;
```

Όταν ο παίκτης που έχει το Tag “Player” μπαίνει εντός του Collider, το Script εμφανίζει την εικόνα με τις πληροφορίες και όταν ο παίκτης βγαίνει το Script την κρύβει πάλι.

```
private void OnTriggerEnter2D ( Collider2D other ) {  
    if ( other.CompareTag ( "Player" ) ) {  
        image.enabled = true;  
    }  
}  
private void OnTriggerExit2D ( Collider2D other ) {  
    if ( other.CompareTag ( "Player" ) ) {  
        image.enabled = false;  
    }  
}  
}
```

Το Object του καμβά έχει επάνω του ένα Script με όνομα «MainMapCanvasScript» και είναι υπεύθυνο για όλο το text επάνω στην αρχική σκηνή του χάρτη.

MainMapCanvasScript.cs

```
using UnityEngine;  
using UnityEngine.UI;  
  
public class MainMapCanvasScript : MonoBehaviour {
```

Ο κώδικας έχει πολλές μεταβλητές από τις οποίες οι μισές είναι συνδέσεις εντός του Unity με άλλα αντικείμενα και οι υπόλοιπες είναι Booleans που ελέγχουν εάν ο παίκτης έχει νικήσει ένα – ένα όλα τα μέτωπα που έχει παίξει.

```
[ Header ( "References" ) ]  
[ SerializeField ] private LevelSystem GameManagerObj;  
[ Space ( 10 ) ]  
  
[ SerializeField ] private Text xpText;  
[ SerializeField ] private Text levelText;  
[ SerializeField ] private Text lifeText;  
[ SerializeField ] private Text fireRateText;  
[ SerializeField ] private Text victoryText;  
  
[ Space ( 10 ) ]  
  
[ SerializeField ] private Text joinFrontRateText;  
  
[ Header ( "Images" ) ]  
[ SerializeField ] private GameObject sassanids;  
[ SerializeField ] private GameObject ostrogoths;  
[ SerializeField ] private GameObject sassanidsPositions;  
[ SerializeField ] private GameObject ostrogothsPositions;  
  
[ Header ( "Sounds" ) ]  
[ SerializeField ] private AudioClip image;  
  
public static bool hasWonSassanids;  
public static bool hasWonOstrogoths;  
  
public static bool hasWonSpaniae;  
public static bool hasWonItaliaAnnonaria;  
public static bool hasWonItaliaSuburbicaria;  
public static bool hasWonIllyricum;  
public static bool hasWonDacia;  
public static bool hasWonMacedonia;  
public static bool hasWonQuaesturaExercitus;  
public static bool hasWonThracia;
```

```

public static bool hasWonAsiana;
public static bool hasWonPontica;
public static bool hasWonOriens;
public static bool hasWonAegyptus;
public static bool hasWonAfrica;

```

Στην αρχή του Script η **Start();** τρέχει την **SetPlayerPrefs();**. Η **SetPlayerPrefs();**, ελέγχει στη μνήμη εάν ο παίκτης έχει ξαναπαιξει και έχει κερδίσει κάποια από τις περιοχές. Σε περίπτωση που υπάρχει κάποια νικημένη περιοχή, τότε το αντίστοιχο *Bool* που δηλώσαμε πάνω για αυτήν την περιοχή γίνεται *true*.

```

private void Start ( ) {
    SetPlayerPrefs ( );
}

```

Μέσα της η **Update();** καλεί αρκετές συναρτήσεις, τις οποίες θα παρουσιάσουμε στην συνέχεια αναλυτικά.

```

private void Update ( ) {
    LevelText ( );
    XPText ( );
    LifeText ( );
    FireRateText ( );
    JoinFrontRateText ( );
    ShowBattles ( );
    ShowVictoryText ( );
}

private void SetPlayerPrefs ( ) {
    hasWonSpaniae = ( PlayerPrefs.GetInt ( "hasWonSpaniae" ) != 0 );
    hasWonItaliaAnnonaria = ( PlayerPrefs.GetInt ( "hasWonItaliaAnnonaria" ) != 0 );
    hasWonItaliaSuburbicaria = ( PlayerPrefs.GetInt ( "hasWonItaliaSuburbicaria" ) != 0 );
    hasWonIllyricum = ( PlayerPrefs.GetInt ( "hasWonIllyricum" ) != 0 );
    hasWonDacia = ( PlayerPrefs.GetInt ( "hasWonDacia" ) != 0 );
    hasWonMacedonia = ( PlayerPrefs.GetInt ( "hasWonMacedonia" ) != 0 );
    hasWonThracia = ( PlayerPrefs.GetInt ( "hasWonThracia" ) != 0 );
    hasWonQuaesturaExercitus = ( PlayerPrefs.GetInt ( "hasWonQuaesturaExercitus" ) != 0 );
    hasWonPontica = ( PlayerPrefs.GetInt ( "hasWonPontica" ) != 0 );
    hasWonAsiana = ( PlayerPrefs.GetInt ( "hasWonAsiana" ) != 0 );
    hasWonAegyptus = ( PlayerPrefs.GetInt ( "hasWonAegyptus" ) != 0 );
    hasWonAfrica = ( PlayerPrefs.GetInt ( "hasWonAfrica" ) != 0 );
    hasWonOstrogoths = ( PlayerPrefs.GetInt ( "hasWonOstrogoths" ) != 0 );
    hasWonSassanids = ( PlayerPrefs.GetInt ( "hasWonSassanids" ) != 0 );
}

```

Η **LevelText();** είναι μία απλή συνάρτηση υπεύθυνη για να δείχνει το Level του παίκτη, στο κέντρο κάτω χαμηλά στην οθόνη.

```

private void LevelText ( ) {
    if ( LevelSystem.level < 10 ) {
        levelText.text = "Level " + LevelSystem.level;
    }
    else {
        levelText.text = "Level 10 (Max)";
    }
}

```

Η **XPText()**; δείχνει κάτω από το Level το XP που έχει μαζέψει ο παίκτης.

```
private void XPText ( ) {  
    if ( LevelSystem.level < 10 ) {  
        xpText.text = GameManagerObj.currentXp + " of " + GameManagerObj.requiredXp;  
    }  
    else {  
        xpText.text = "Max XP";  
    }  
}
```

Η συνάρτηση **ShowBattles()**; έχει φτιαχτεί για να δείχνει πάνω αριστερά στη γωνία της οθόνης τις εικόνες από τις μάχες που έχει κερδίσει ο παίκτης.

```
private void ShowBattles ( ) {  
    if ( hasWonOstrogoths ) {  
        ostrogoths.SetActive ( true );  
    }  
    if ( hasWonSassanids ) {  
        sassanids.SetActive ( true );  
    }  
}
```

Οι συναρτήσεις **ShowSassanidsPositions()**; και **ShowOstrogothsPositions()**; δείχνουν τις θέσεις των Βυζαντινών και των αντιπάλων τους στις δύο μάχες. Όταν ο παίκτης νικήσει κάποια από τις μάχες και πάνω αριστερά στην οθόνη του εμφανιστεί η σημαία της μάχης που έχει νικήσει, μπορεί βάζοντας το ποντίκι του επάνω της να δει τις διατάξεις της μάχης. Οι συναρτήσεις **HideSassanidsPositions()**; και **HideOstrogothsPositions()**; κρύβουν αυτές τις εικόνες.

```
public void ShowSassanidsPositions ( ) {  
    sassanidsPositions.SetActive( true );  
    if( !GetComponent<AudioSource>().isPlaying ) {  
        GetComponent<AudioSource>().PlayOneShot( image );  
    }  
}  
  
public void HideSassanidsPositions(){  
    sassanidsPositions.SetActive( false );  
}  
  
public void ShowOstrogothsPositions(){  
    ostrogothsPositions.SetActive( true );  
    if( !GetComponent<AudioSource>().isPlaying ) {  
        GetComponent<AudioSource>().PlayOneShot( image );  
    }  
}  
  
public void hideOstrogothsPositions(){  
    ostrogothsPositions.SetActive( false );  
}
```

Παρακάτω έχουμε δύο συναρτήσεις, οι οποίες δείχνουν το τελικό κείμενο στο κέντρο της οθόνης όταν νικάει ο παίκτης. Όταν έχει κερδίσει και τις δύο μάχες ο παίκτης, τότε η συνάρτηση **ShowVictoryText()**; εμφανίζει το μήνυμα «Victory!» στο κέντρο της

αρχικής σκηνής του χάρτη. Μετά από 3 δευτερόλεπτα, καλείτε η **HideVictoryText();** η οποία το αποκρύπτει.

```
public void ShowVictoryText(){
    if(victoryText && hasWonSassanids && hasWonOstrogoths){
        victoryText.enabled = true;
        Invoke("HidedVictoryText", 3);
    }
}

private void HidedVictoryText(){
    Destroy(victoryText);
}
```

Η **LifeText();** είναι υπεύθυνη για να δείχνει πόση ζωή έχει ο παίκτης ανάλογα με το Level του.

```
private void LifeText ( ) {
    if ( LevelSystem.level == 1 ) {
        lifeText.text = "Life : 100";
    }
    else if ( LevelSystem.level == 2 ) {
        lifeText.text = "Life : 110";
    }
    else if ( LevelSystem.level == 3 ) {
        lifeText.text = "Life : 120";
    }
    else if ( LevelSystem.level == 4 ) {
        lifeText.text = "Life : 130";
    }
    else if ( LevelSystem.level == 5 ) {
        lifeText.text = "Life : 150";
    }
    else if ( LevelSystem.level == 6 ) {
        lifeText.text = "Life : 180";
    }
    else if ( LevelSystem.level == 7 ) {
        lifeText.text = "Life : 230";
    }
    else if ( LevelSystem.level == 8 ) {
        lifeText.text = "Life : 310";
    }
    else if ( LevelSystem.level == 9 ) {
        lifeText.text = "Life : 440";
    }
    else if ( LevelSystem.level >= 10 ) {
        lifeText.text = "Life : 650 (Max)";
    }
}
```

Η **FireRateText();** είναι υπεύθυνη για να δείχνει πόσα βέλη μπορεί να ρίξει ο παίκτης ανά δευτερόλεπτο με βάση το Level του.

```
private void FireRateText ( ) {
    if ( LevelSystem.level == 1 ) {
        fireRateText.text = "1 arrow per second";
    }
    else if ( LevelSystem.level == 2 ) {
        fireRateText.text = "1.05 arrows per second";
    }
    else if ( LevelSystem.level == 3 ) {
        fireRateText.text = "1.1 arrows per second";
    }
    else if ( LevelSystem.level == 4 ) {
        fireRateText.text = "1.2 arrows per second";
    }
}
```

```

    }
    else if ( LevelSystem.level == 5 ) {
        fireRateText.text = "1.25 arrows per second";
    }
    else if ( LevelSystem.level == 6 ) {
        fireRateText.text = "1.35 arrows per second";
    }
    else if ( LevelSystem.level == 7 ) {
        fireRateText.text = "1.5 arrows per second";
    }
    else if ( LevelSystem.level == 8 ) {
        fireRateText.text = "1.8 arrows per second";
    }
    else if ( LevelSystem.level == 9 ) {
        fireRateText.text = "2 arrows per second";
    }
    else if ( LevelSystem.level >= 10 ) {
        fireRateText.text = "3.5 arrows per second (Max)";
    }
}

```

Με τη συνάρτηση **JoinFrontRateText()**;παρουσιάζουμε το μήνυμα στο πάνω μέρος της οθόνης όταν ο παίκτης εισέρχεται εντός των ορίων μιας περιοχής. Εάν ο παίκτης μπορεί να μπει να παίξει, τα γράμματα είναι λευκά, ενώ εάν έχει ήδη νικήσει στη συγκεκριμένη περιοχή τα γράμματα είναι πράσινα και το μήνυμα διαφορετικό. Εάν είναι μία από τις δύο μάχες που ο παίκτης είναι εντός των ορίων, τότε τα γράμματα είναι κόκκινα αντί για λευκά στην αρχή.

```

private void JoinFrontRateText ( ) {
    if ( PlayerMovement.location == null ) {
        joinFrontRateText.text = " ";
        joinFrontRateText.color = Color.white;
    }
    else if ( PlayerMovement.location == "Macedonia" ) {
        if ( hasWonMacedonia == true ) {
            joinFrontRateText.text = "Already won Macedonia";
            joinFrontRateText.color = Color.green;
        }
        else {
            joinFrontRateText.text = "Press Space to join Macedonia";
            joinFrontRateText.color = Color.white;
        }
    }
    else if ( PlayerMovement.location == "Thracia" ) {
        if ( hasWonThracia == true ) {
            joinFrontRateText.text = "Already won Thracia";
            joinFrontRateText.color = Color.green;
        }
        else {
            joinFrontRateText.text = "Press Space to join Thracia";
            joinFrontRateText.color = Color.white;
        }
    }
    else if ( PlayerMovement.location == "Dacia" ) {
        if ( hasWonDacia == true ) {
            joinFrontRateText.text = "Already won Dacia";
            joinFrontRateText.color = Color.green;
        }
        else {
            joinFrontRateText.text = "Press Space to join Dacia";
            joinFrontRateText.color = Color.white;
        }
    }
    else if ( PlayerMovement.location == "Quaestura Exercitus" ) {
        if ( hasWonQuaesturaExercitus == true ) {
            joinFrontRateText.text = "Already won Quaestura Exercitus";
            joinFrontRateText.color = Color.green;
        }
        else {
            joinFrontRateText.text = "Press Space to join Quaestura Exercitus";
            joinFrontRateText.color = Color.white;
        }
    }
}

```

```

    }
    else if ( PlayerMovement.location == "Illyricum" ) {
        if ( hasWonIllyricum == true ) {
            joinFrontRateText.text = "Already won Illyricum";
            joinFrontRateText.color = Color.green;
        }
        else {
            joinFrontRateText.text = "Press Space to join Illyricum";
            joinFrontRateText.color = Color.white;
        }
    }
    else if ( PlayerMovement.location == "Italia Annonaria" ) {
        if ( hasWonItaliaAnnonaria == true ) {
            joinFrontRateText.text = "Already won Italia Annonaria";
            joinFrontRateText.color = Color.green;
        }
        else {
            joinFrontRateText.text = "Press Space to join Italia Annonaria";
            joinFrontRateText.color = Color.white;
        }
    }
    else if ( PlayerMovement.location == "Italia Suburbicaria" ) {
        if ( hasWonItaliaSuburbicaria == true ) {
            joinFrontRateText.text = "Already won Italia Suburbicaria";
            joinFrontRateText.color = Color.green;
        }
        else {
            joinFrontRateText.text = "Press Space to join Italia Suburbicaria";
            joinFrontRateText.color = Color.white;
        }
    }
    else if ( PlayerMovement.location == "Pontica" ) {
        if ( hasWonPontica == true ) {
            joinFrontRateText.text = "Already won Pontica";
            joinFrontRateText.color = Color.green;
        }
        else {
            joinFrontRateText.text = "Press Space to join Pontica";
            joinFrontRateText.color = Color.white;
        }
    }
    else if ( PlayerMovement.location == "Asiana" ) {
        if ( hasWonAsiana == true ) {
            joinFrontRateText.text = "Already won Asiana";
            joinFrontRateText.color = Color.green;
        }
        else {
            joinFrontRateText.text = "Press Space to join Asiana";
            joinFrontRateText.color = Color.white;
        }
    }
    else if ( PlayerMovement.location == "Oriens" ) {
        if ( hasWonOriens == true ) {
            joinFrontRateText.text = "Already won Oriens";
            joinFrontRateText.color = Color.green;
        }
        else {
            joinFrontRateText.text = "Press Space to join Oriens";
            joinFrontRateText.color = Color.white;
        }
    }
    else if ( PlayerMovement.location == "Aegyptus" ) {
        if ( hasWonAegyptus == true ) {
            joinFrontRateText.text = "Already won Aegyptus";
            joinFrontRateText.color = Color.green;
        }
        else {
            joinFrontRateText.text = "Press Space to join Aegyptus";
            joinFrontRateText.color = Color.white;
        }
    }
    else if ( PlayerMovement.location == "Africa" ) {
        if ( hasWonAfrica == true ) {
            joinFrontRateText.text = "Already won Africa";
            joinFrontRateText.color = Color.green;
        }
        else {
            joinFrontRateText.text = "Press Space to join Africa";
            joinFrontRateText.color = Color.white;
        }
    }
    else if ( PlayerMovement.location == "Spaniae" ) {
        if ( hasWonSpaniae == true ) {
            joinFrontRateText.text = "Already won Spaniae";
            joinFrontRateText.color = Color.green;
        }
    }

```

```

    }
    else {
        joinFrontRateText.text = "Press Space to join Spaniae";
        joinFrontRateText.color = Color.white;
    }
}
else if ( PlayerMovement.location == "Dara" ) {
    if ( hasWonSassanids == true ) {
        joinFrontRateText.text = "Already won Battle of Dara";
        joinFrontRateText.color = Color.green;
    }
    else {
        joinFrontRateText.text = "Press Space to join the Battle of Dara";
        joinFrontRateText.color = Color.red;
    }
}
else if ( PlayerMovement.location == "Taginae" ) {
    if ( hasWonOstrogoths == true ) {
        joinFrontRateText.text = "Already won Battle of Taginae";
        joinFrontRateText.color = Color.green;
    }
    else {
        joinFrontRateText.text = "Press Space to join the Battle of Taginae";
        joinFrontRateText.color = Color.red;
    }
}
}
}
}
}

```

Τέλος για την αρχική σκηνή με τον χάρτη, έχουμε τρία Objects, τα Objects των σημαίων. Και τα τρία έχουν επάνω τους ένα μικρό Script που τα κάνει να κουνιούνται παλινδρομικά.

BannerScript.cs

```

using UnityEngine;

public class BannerScript : MonoBehaviour {
    private bool flag = false;
    private void Update ( ) {
        if ( transform.rotation.x < 0.05 && flag == false ) {
            transform.Rotate ( 5f * Time.deltaTime, 5f * Time.deltaTime, 5f * Time.deltaTime );
        }
        else {
            flag = true;
            transform.Rotate ( -5f * Time.deltaTime, -5f * Time.deltaTime, -5f * Time.deltaTime );
        }

        if ( transform.rotation.x < -0.05 ) {
            flag = false;
        }
    }
}

```

Ο συγκεκριμένος κώδικας κάνει *Rotate* τη σημαία μέχρι ένα σημείο και πίσω πάλι. Με το Script αυτό ολοκληρώνεται η αρχική σκηνή του χάρτη. Στη συνέχεια θα παρουσιάσουμε τον κώδικα των μαχών. Θα πάρουμε για παράδειγμα τη σκηνή της μάχης της Δάρας. Και εδώ υπάρχει το Object Game Manager με δύο Scripts: το «WaveSpanwer» και το «GameManager». Θα παρουσιάσουμε αρχικά το WaveSpawner.

WaveSpawner.cs

```

using System.Collections;
using UnityEngine;

public class WaveSpawner : MonoBehaviour {

    public enum SpawnState { SPAWNING, WAITING, COUNTING };

```

```

[ System.Serializable ]
[ SerializeField ] private class Wave {
    public string name;
    public Transform enemy;
    public int count;
    public float rate;
}

[ SerializeField ] private Wave [ ] waves;
[ SerializeField ] private Transform spawnPoint;
[ SerializeField ] private float timeBetweenWaves = 2f;
[ HideInInspector ] public int waveCount;
public static bool wavesFinished = false;

private int nextWave = 0;
private float waveCountdown;
private float searchCountdown = 1f;
SpawnState state = SpawnState.COUNTING;

```

Για να είναι επεκτάσιμο το παιχνίδι το κάθε Wave εχθρών αποτελείται από μία κλάση. Για ευκολία και ευκολότερο διάβασμα του κώδικα το κάθε κύμα εχθρών έχει ένα όνομα. Μαζί με το όνομα υπάρχει μία μεταβλητή για το Prefab του εχθρού που θέλουμε, από κάτω το πλήθος των εχθρών που θα έχει το κύμα και τέλος με τι ρυθμό θα κάνουν Spawn. Όλες αυτές οι μεταβλητές ρυθμίζονται ανά Wave εντός του Unity. Τέλος, με το **System.Serializable** μπορούμε πάλι εντός του Unity να ρυθμίσουμε πόσα θα θέλαμε Waves να έχει η κάθε περιοχή και μάχη.

Παρακάτω στην **Start();** γίνονται κάποιες αρχικοποιήσεις, συγκεκριμένα βάζουμε σε μία μεταβλητή πόσα Waves έχουμε και πόση ώρα αφού μπει ο παίκτης στην περιοχή θα αρχίσουν να έρχονται οι εχθροί.

```

void Start ( ) {
    waveCount = waves.Length;
    waveCountdown = timeBetweenWaves;
}

```

Εντός της **Update();** ελέγχουμε σε ποια κατάσταση είναι το παιχνίδι μας. Εάν δεν υπάρχουν ζωντανοί εχθροί, εκτελείται η **WaveCompleted();** και αυτή με την σειρά της ελέγχει εάν έχει τελειώσει το παιχνίδι, δηλαδή αν δεν υπάρχει άλλο Wave να βγει, ή διαφορετικά πάει στο επόμενο Wave. Εάν έχει περάσει ο χρόνος ανάμεσα από τα Waves, τότε αρχίζει το επόμενο κύμα εχθρών να έρχεται.

```

void Update ( ) {
    if ( state == SpawnState.WAITING ) {
        //Check if enemies are still alive
        if ( EnemyIsAlive ( ) == false ) {
            //Begin new round
            waveCount = waveCount - 1;
            WaveCompleted ( );
            return;
        }
    }
    else {
        return;
    }
}

```

```

        if ( waveCountdown <= 0 ) {
            if ( state != SpawnState.SPAWNING ) {
                StartCoroutine ( SpawnWave ( waves [ nextWave ] ) );
            }
        }
        else {
            waveCountdown = waveCountdown - Time.deltaTime;
        }
    }

    void WaveCompleted ( ) {
        state = SpawnState.COUNTING;
        waveCountdown = timeBetweenWaves;

        if ( nextWave + 1 > waves.Length - 1 ) {
            nextWave = 0;
            wavesFinished = true;
            state = SpawnState.COUNTING;
        }
        else {
            nextWave++;
        }
    }
}

```

Εντός της **EnemyIsAlive()**; ελέγχουμε κάθε ένα δευτερόλεπτο εάν υπάρχει ή όχι κάποιος εχθρός ζωντανός. Ο λόγος που εκτελείται κάθε ένα δευτερόλεπτο είναι για να κερδίσουμε επιδόσεις σε σχέση με το να εκτελείται κάθε φορά που εκτελείται η **Update()**;

```

bool EnemyIsAlive ( ) {
    searchCountdown = searchCountdown - Time.deltaTime;
    if ( searchCountdown <= 0 ) {
        searchCountdown = 1f;
        if ( GameObject.FindGameObjectWithTag ( "Enemy" ) == null ) {
            return false;
        }
    }
    return true;
}

IEnumerator SpawnWave( Wave _wave ) {
    state = SpawnState.SPAWNING;
    //Spawning
    for ( int i = 0; i < _wave.count; i++ ) {
        SpawnEnemy ( _wave.enemy );
        yield return new WaitForSeconds( 1f / _wave.rate );
    }

    //Waiting
    state = SpawnState.WAITING;
    yield break;
}

void SpawnEnemy ( Transform _enemy ) {
    Instantiate ( _enemy, spawnPoint.position, spawnPoint.rotation );
}
}

```

Το δεύτερο Script του Game Manager, το «GameManagerScript είναι υπεύθυνο για τη μουσική που παίζει με το που αρχίσει η σκηνή. Έχει ένα Array από μουσικές και παίζει μία τυχαία στην αρχή του παιχνιδιού.

GameManagerScript.cs

```

using UnityEngine;

public class GameManagerScript : MonoBehaviour {
    AudioSource soundtrack;
    [ SerializeField ] private AudioClip [ ] audioClips;
}

```

```

private void Start ( ) {
    soundtrack = GetComponent<AudioSource>( );
    soundtrack.clip = audioClips [ Random.Range ( 0, audioClips.Length ) ];
    soundtrack.Play( );
}
}

```

Δεύτερο στο Hierarchy του Unity είναι το Prefab του παίκτη με το Script με όνομα «PlayerScript». Επίσης, ένα από τα παιδιά του στην ιεραρχία, είναι υπεύθυνο για τη λειτουργία του τόξου με ένα Script με όνομα «Bow». Θα παρουσιάσουμε τα δύο script.

PlayerScript.cs

```

using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

// In-game script

public class PlayerScript : MonoBehaviour {
    [ Header ( "References" ) ]
    [ SerializeField ] private Image healthBar;
    [ SerializeField ] private GameObject rotationPoint;
    [ SerializeField ] private AudioSource audioSource;
    [ SerializeField ] private AudioClip [ ] playerPainSounds;
    [ Space ( 10 ) ]

    public bool wonTheGame; //Used for text when level finishes
    public bool endTheGame;

    [Header("Variables")]
    public int damage = 5;
    public int life;
    private int maxLife;
    private GameObject [ ] enemys;
}

```

Αρχικά στο Script έχουμε την **Start();** η οποία εκτελεί την **SetPlayerLife();**, η οποία είναι υπεύθυνη για να δώσει στον παίκτη όση ζωή πρέπει ανάλογα με το Level που έχει φτάσει.

```

private void Start ( ) {
    SetPlayerLife ( );
    maxLife = life;
}

```

Έπειτα έχουμε την **Update();**, η οποία εκτελεί σε κάθε frame τρεις συναρτήσεις. Η **GameFinished();** ελέγχει μία καθολική μεταβλητή εντός του Game Manager που είδαμε προηγούμενος η οποία ελέγχει εάν έχουν τελειώσει τα Waves. Μόλις αυτή η μεταβλητή γίνει *true*, καλεί με τη σειρά της την **XpWon();** και την **MainMenu();**. Η **XpWon();** δίνει στον παίκτη XP ανάλογα σε ποια περιοχή είναι και μετά η **MainMenu();** τον πηγαίνει στον αρχικό χάρτη. Στη συνέχεια η **HealthRenderer();** είναι υπεύθυνη για το γράφημα επάνω από τον παίκτη που δείχνει την ζωή του. Τέλος, η **Death();** ελέγχει εάν ο παίκτης έχει χάσει όλη του την ζωή και καλεί και αυτή την **MainMenu();**.

Ταυτόχρονα σταματάει όλους τους αντιπάλους και τους βάζει σε Idle animation. Η **MainMenu();** όπως δείχνει και το όνομα, απλά καλεί την σκηνή του αρχικού χάρτη.

```
private void Update ( ) {
    GameFinished ( );
    HealthRender ( );
    Death ( );
}

private void HealthRender ( ) {
    if ( healthBar != null ) {
        healthBar.fillAmount = Mathf.Lerp ( healthBar.fillAmount, ( float ) life / maxLife, 0.05f );
    }
}

private void SetPlayerLife ( ) {
    if ( LevelSystem.level == 1 ) {
        life = 100;
    }
    else if ( LevelSystem.level == 2 ) {
        life = 110;
    }
    else if ( LevelSystem.level == 3 ) {
        life = 120;
    }
    else if ( LevelSystem.level == 4 ) {
        life = 130;
    }
    else if ( LevelSystem.level == 5 ) {
        life = 150;
    }
    else if ( LevelSystem.level == 6 ) {
        life = 180;
    }
    else if ( LevelSystem.level == 7 ) {
        life = 230;
    }
    else if ( LevelSystem.level == 8 ) {
        life = 310;
    }
    else if ( LevelSystem.level == 9 ) {
        life = 440;
    }
    else if ( LevelSystem.level >= 10 ) {
        life = 650;
    }
}
```

Η **TakeDamage(int dmg);** καλείται εντός του Script του εχθρού όταν ο παίκτης χτυπιέται από τον εχθρό και έχει σαν σκοπό να αφαιρέσει ένα κομμάτι της ζωής.

```
public void TakeDamage ( int dmg ) {
    AudioSource.PlayOneShot ( playerPainSounds [ Random.Range ( 0, playerPainSounds.Length ) ] );
    life = life - dmg;
}

private void GameFinished( ) {
    if ( WaveSpawner.wavesFinished == true ) {
        WaveSpawner.wavesFinished = false;
        endTheGame = true;
        wonTheGame = true;
        XpWon ( );
        Invoke ( "MainMenu", 2f );
    }
}

private void XpWon ( ) {
    if ( SceneManager.GetActiveScene( ).name == "Spaniae" ) {
        LevelSystem.xpOnWin = 20;
        MainMapCanvasScript.hasWonSpaniae = true;
        PlayerPrefs.SetInt ( "hasWonSpaniae", ( MainMapCanvasScript.hasWonSpaniae ? 1 : 0 ) );
    }
}
```

```

        else if ( SceneManager.GetActiveScene( ).name == "Italia Annonaria" ) {
            LevelSystem.xpOnWin = 25;
            MainMapCanvasScript.hasWonItaliaAnnonaria = true;
            PlayerPrefs.SetInt ( "hasWonItaliaAnnonaria", ( MainMapCanvasScript.hasWonItaliaAnnonaria ? 1 : 0 ) );
        }
        else if ( SceneManager.GetActiveScene( ).name == "Italia Suburbicaria" ) {
            LevelSystem.xpOnWin = 32;
            MainMapCanvasScript.hasWonItaliaSuburbicaria = true;
            PlayerPrefs.SetInt ( "hasWonItaliaSuburbicaria", ( MainMapCanvasScript.hasWonItaliaSuburbicaria ? 1 : 0 ) );
        }
    };

    }
    else if ( SceneManager.GetActiveScene( ).name == "Illyricum" ) {
        LevelSystem.xpOnWin = 40;
        MainMapCanvasScript.hasWonIllyricum = true;
        PlayerPrefs.SetInt ( "hasWonIllyricum", ( MainMapCanvasScript.hasWonIllyricum ? 1 : 0 ) );
    }
    else if ( SceneManager.GetActiveScene( ).name == "Dacia" ) {
        LevelSystem.xpOnWin = 50;
        MainMapCanvasScript.hasWonDacia = true;
        PlayerPrefs.SetInt ( "hasWonDacia", ( MainMapCanvasScript.hasWonDacia ? 1 : 0 ) );
    }
    else if ( SceneManager.GetActiveScene( ).name == "Macedonia" ) {
        LevelSystem.xpOnWin = 65;
        MainMapCanvasScript.hasWonMacedonia = true;
        PlayerPrefs.SetInt ( "hasWonMacedonia", ( MainMapCanvasScript.hasWonMacedonia ? 1 : 0 ) );
    }
    else if ( SceneManager.GetActiveScene( ).name == "Thracia" ) {
        LevelSystem.xpOnWin = 75;
        MainMapCanvasScript.hasWonThracia = true;
        PlayerPrefs.SetInt ( "hasWonThracia", ( MainMapCanvasScript.hasWonThracia ? 1 : 0 ) );
    }
    else if ( SceneManager.GetActiveScene( ).name == "Quaestura Exercitus" ) {
        LevelSystem.xpOnWin = 80;
        MainMapCanvasScript.hasWonQuaesturaExercitus = true;
        PlayerPrefs.SetInt ( "hasWonQuaesturaExercitus", ( MainMapCanvasScript.hasWonQuaesturaExercitus ? 1 : 0 ) );
    }
};

    }
    else if ( SceneManager.GetActiveScene( ).name == "Pontica" ) {
        LevelSystem.xpOnWin = 83;
        MainMapCanvasScript.hasWonPontica = true;
        PlayerPrefs.SetInt ( "hasWonPontica", ( MainMapCanvasScript.hasWonPontica ? 1 : 0 ) );
    }
    else if ( SceneManager.GetActiveScene( ).name == "Asiana" ) {
        LevelSystem.xpOnWin = 85;
        MainMapCanvasScript.hasWonAsiana = true;
        PlayerPrefs.SetInt ( "hasWonAsiana", ( MainMapCanvasScript.hasWonAsiana ? 1 : 0 ) );
    }
    else if ( SceneManager.GetActiveScene( ).name == "Oriens" ) {
        LevelSystem.xpOnWin = 90;
        MainMapCanvasScript.hasWonOriens = true;
        PlayerPrefs.SetInt ( "hasWonOriens", ( MainMapCanvasScript.hasWonOriens ? 1 : 0 ) );
    }
    else if ( SceneManager.GetActiveScene( ).name == "Aegyptus" ) {
        LevelSystem.xpOnWin = 100;
        MainMapCanvasScript.hasWonAegyptus = true;
        PlayerPrefs.SetInt ( "hasWonAegyptus", ( MainMapCanvasScript.hasWonAegyptus ? 1 : 0 ) );
    }
    else if ( SceneManager.GetActiveScene( ).name == "Africa" ) {
        LevelSystem.xpOnWin = 120;
        MainMapCanvasScript.hasWonAfrica = true;
        PlayerPrefs.SetInt ( "hasWonAfrica", ( MainMapCanvasScript.hasWonAfrica ? 1 : 0 ) );
    }
    else if ( SceneManager.GetActiveScene( ).name == "Taginae" ) {
        MainMapCanvasScript.hasWonOstrogoths = true;
        PlayerPrefs.SetInt ( "hasWonOstrogoths", ( MainMapCanvasScript.hasWonOstrogoths ? 1 : 0 ) );
        LevelSystem.xpOnWin = 110;
    }
    else if ( SceneManager.GetActiveScene( ).name == "Dara" ) {
        MainMapCanvasScript.hasWonSassanids = true;
        PlayerPrefs.SetInt ( "hasWonSassanids", ( MainMapCanvasScript.hasWonSassanids ? 1 : 0 ) );
        LevelSystem.xpOnWin = 150;
    }
    }
}

private void MainMenu ( ) {
    SceneManager.LoadScene ( 0 );
}

private void Death ( ) {
    if ( life <= 0 ) {
        endTheGame = true;
        wonTheGame = false;

        rotationPoint.GetComponent<Bow>( ).enabled = false;
    }
}

```

```

        enemys = GameObject.FindGameObjectsWithTag ( "Enemy" );
        foreach ( GameObject enemy in enemys ) { //Freeze enemies
            enemy.GetComponent<Animator>().SetBool( "EnemyIsStill", true );
            enemy.GetComponent<Animator>().SetBool( "EnemyAttacks", false );
            enemy.GetComponent<Enemy>().enabled = false;
        }
        Invoke ( "MainMenu", 2f );
    }
}

```

Στην συνέχεια παρουσιάζουμε το Script του τόξου.

Bow.cs

```

using UnityEngine;

public class Bow : MonoBehaviour {

    [ Header ( "References" ) ]
    [ SerializeField ] private GameObject arrow;
    [ SerializeField ] private Transform shotPoint;
    [ SerializeField ] private AudioSource shotSource;

    [ Header ( "Variables" ) ]
    [ HideInInspector ] public float fireRate;
    [ HideInInspector ] public bool canFire = false;
    private float launchForce;
    private float nextFire = 0f;
    private float zRotation;
}

```

Για αρχή μέσα στην **Awake();** και στην **Start();** αρχικοποιείται το fire rate του τόξου που είναι ανάλογο με το επίπεδο που έχει ο παίκτης και η δύναμη που ρίχνει τα βέλη, που είναι ανάλογη με το αν έχει κερδίσει κάποια από τις δύο μάχες.

```

private void Awake ( ) {
    SetFireRate ( );
}

private void Start ( ) {
    SetLaunchForce ( );
}

```

Έπειτα, μέσα στην **Update();** έχουμε τρεις συναρτήσεις, την **MouseLook();**, την **ClampRotation();** και την **Fire();**. Η **MouseLook();** είναι υπεύθυνη για να κάνει το τόξο να γυρνάει στον δείκτη του ποντικιού. Η **ClampRotation();** επιτρέπει στον παίκτη να μπορεί να γυρίσει μέχρι μερικές μοίρες πάνω και κάτω. Χωρίς αυτή τη συνάρτηση ο παίκτης θα μπορούσε να γυρίσει το σώμα του 180 μοίρες και να ρίξει προς τα πίσω. Και τέλος η **Fire();** είναι υπεύθυνη για να ρίχνει ένα βέλος κάθε φορά που ο παίκτης κάνει αριστερό κλικ και έχει περάσει το Cool Down.

```

private void Update ( ) {
    MouseLook ( );
    ClampRotation ( );
    Fire ( );
}

```

```

private void SetLaunchForce ( ) {
    if ( MainMapCanvasScript.hasWonOstrogoths && MainMapCanvasScript.hasWonSassanids ) {
        launchForce = 13f;
    }
    else if ( MainMapCanvasScript.hasWonOstrogoths || MainMapCanvasScript.hasWonSassanids ) {
        launchForce = 11.5f;
    }
    else {
        launchForce = 10f;
    }
}

private void SetFireRate ( ) {
    if ( LevelSystem.level == 1 ) {
        fireRate = 1f;
    }
    else if ( LevelSystem.level == 2 ) {
        fireRate = 1.05f;
    }
    else if ( LevelSystem.level == 3 ) {
        fireRate = 1.1f;
    }
    else if ( LevelSystem.level == 4 ) {
        fireRate = 1.2f;
    }
    else if ( LevelSystem.level == 5 ) {
        fireRate = 1.25f;
    }
    else if ( LevelSystem.level == 6 ) {
        fireRate = 1.35f;
    }
    else if ( LevelSystem.level == 7 ) {
        fireRate = 1.5f;
    }
    else if ( LevelSystem.level == 8 ) {
        fireRate = 1.8f;
    }
    else if ( LevelSystem.level == 9 ) {
        fireRate = 2f;
    }
    else if ( LevelSystem.level >= 10 ) {
        fireRate = 3.5f;
    }
}

private void Fire ( ) {
    //Bow follow the mouse and shoots when left click is pressed
    if ( canFire == false && Time.time >= nextFire ) {
        canFire = true;
    }
    if ( canFire == true ) {
        if ( Input.GetMouseButtonDown ( 0 ) ) {
            canFire = false;
            nextFire = Time.time + 1f / fireRate;
            Shoot ( );
        }
    }
}

private void MouseLook ( ) {
    Vector2 bowPosition = transform.position;
    Vector2 mousePosition = Camera.main.ScreenToWorldPoint ( Input.mousePosition );
    Vector2 direction = mousePosition - bowPosition;
    transform.right = direction;
}

private void ClampRotation ( ) {
    zRotation = transform.eulerAngles.z;
    if ( transform.eulerAngles.z > 50f && zRotation < 180f ) {
        zRotation = 49.9f;
    }
    else if ( zRotation < 310f && zRotation > 180f ) {
        zRotation = 310.1f;
    }
    transform.eulerAngles = new Vector3 ( transform.eulerAngles.x, transform.eulerAngles.y, zRotation );
}

private void Shoot ( ) {
    GetComponentInChildren<Animator>().Play ( "Player_Shoot", 0, 0f );
    shotSource.pitch = Random.Range( 0.9f, 1.1f );
    shotSource.Play();
    GameObject newArrow = Instantiate( arrow, shotPoint.position, shotPoint.rotation );
    newArrow.GetComponent<Rigidbody2D>().velocity = transform.right * launchForce;
}
}

```

Μετά έχουμε το Script του καμβά. Με αυτό το Script φαίνεται με Text στο κάτω μέρος σε ποιο Wave είμαστε, το τελικό μήνυμα Victory ή You Died και επίσης με αυτό το Script κάνουμε το Cool Down Indicator γύρω από τον κέρσορα να δουλεύει.

CanvasScript.cs

```
using UnityEngine;
using UnityEngine.UI;

public class CanvasScript : MonoBehaviour {
    [ Header ( "References" ) ]
    [ SerializeField ] private GameObject player;
    [ SerializeField ] private GameObject rotationPoint;
    [ SerializeField ] private GameObject GameManager;
    [ Space ( 10 ) ]
    [ SerializeField ] private Text text;
    [ SerializeField ] private Text wavesText;
    [ SerializeField ] private Image crosshair;
    [ Space ( 10 ) ]

    private float fireRate;
    private float count;
    private Canvas myCanvas;

    private void Start( ) {
        myCanvas = GetComponent<Canvas>( );
        fireRate = rotationPoint.GetComponent<Bow>( ).fireRate;
    }
}
```

Μέσα στην **Update()**; καλούμε την **TextUpdater()**; η οποία εμφανίζει το τελικό μήνυμα στο κέντρο της οθόνης όταν τελειώσει το παιχνίδι και νικήσει ή χάσει ο παίκτης. Μετά καλούμε την **CrosshairFollowMouse()**; η οποία είναι υπεύθυνη για να κάνει το Fire Rate Indicator, δηλαδή τον κύκλο γύρω από το ποντίκι να ακολουθεί το ποντίκι. Έπειτα η **FireIndicator()**; είναι υπεύθυνη για τους μαθηματικούς υπολογισμούς που κάνουν το Cool Down Indicator να δουλεύει. Τέλος, έχουμε την **WaveCountDisplay()**; η οποία δείχνει στο κάτω μέρος της οθόνης σε ποιο από Wave είμαστε.

```
private void Update ( ) {
    TextUpdater ( );
    CrosshairFollowMouse ( );
    FireIndicator ( );
    WaveCountDisplay ( );
}

private void TextUpdater ( ) {
    if ( player.GetComponent<PlayerScript>( ).endTheGame == true ) {
        if ( player.GetComponent<PlayerScript>( ).wonTheGame == true ) {
            text.text = "Victory!";
            text.color = new Color ( 0.36f, 0.88f, 0.42f, 1f );
        }
        else {
            text.color = new Color ( 0.88f, 0.36f, 0.42f, 1f );
        }
    }
}

private void CrosshairFollowMouse ( ) {
    Vector2 pos;
    RectTransformUtility.ScreenPointToLocalPointInRectangle ( myCanvas.transform as RectTransform,
Input.mousePosition, myCanvas.worldCamera, out pos );
    crosshair.transform.position = myCanvas.transform.TransformPoint ( pos );
}

private void FireIndicator ( ) {
```

```

        if ( rotationPoint.GetComponent<Bow>( ).canFire == true || crosshair.fillAmount <= 0f ) {
            count = 1f / fireRate;
            crosshair.color = Color.green;
            crosshair.fillAmount = 1f;
        }
        if ( rotationPoint.GetComponent<Bow>( ).canFire == false ) {
            count = count - Time.deltaTime;
            crosshair.color = Color.red;
            crosshair.fillAmount = count / ( 1f / fireRate );
        }
    }

    private void WaveCountDisplay ( ) {
        if ( GameManager.GetComponent<WaveSpawner>( ).waveCount == 1 ) {
            wavesText.text = "Last wave";
        }
        else if ( GameManager.GetComponent<WaveSpawner>( ).waveCount == 2 ) {
            wavesText.text = "Wave 2 of 3";
        }
        else if ( GameManager.GetComponent<WaveSpawner>( ).waveCount == 3 ) {
            wavesText.text = "Wave 1 of 3";
        }
    }
}

```

Μετά έχουμε τις δύο σημαίες, του Βυζαντίου και της απέναντι πλευράς. Και οι δύο αυτές σημαίες έχουν επάνω τους το **BannerScript()**; που παρουσιάστηκε προηγουμένως και στον αρχικό χάρτη για να τους δίνει μία παλινδρομική κίνηση δεξιά – αριστερά.

```

using UnityEngine;

public class BannerScript : MonoBehaviour {
    private bool flag = false;
    private void Update ( ) {
        if ( transform.rotation.x < 0.05 && flag == false ) {
            transform.Rotate ( 5f * Time.deltaTime, 5f * Time.deltaTime, 5f * Time.deltaTime );
        }
        else {
            flag = true;
            transform.Rotate ( -5f * Time.deltaTime, -5f * Time.deltaTime, -5f * Time.deltaTime );
        }

        if ( transform.rotation.x < -0.05 ) {
            flag = false;
        }
    }
}

```

Τέλος, το Object με το Sprite από τα σύννεφα έχει επάνω του το Script με όνομα **CloudScroller()**; . Ο συγκεκριμένος κώδικας παίρνει την εικόνα και να της δίνει μία τυχαία κατεύθυνση κατά την αρχή του παιχνιδιού. Έχοντας ρυθμίσει την εικόνα μέσα από το Unity να κάνει loop αυτή συνεχίζει να κάνει κύκλους και δίνει την αίσθηση στον παίκτη ότι περνάνε σύννεφα από πάνω.

```

using UnityEngine;

public class CloudScroller : MonoBehaviour {
    private float scrollSpeed;
    private float offset;
    private Material mat;

    private void Start ( ) {
        scrollSpeed = Random.Range( 0.3f, 0.7f ) * ( Random.Range ( 0, 2 ) * 2 - 1 );
        mat = GetComponent<Renderer>().material;
    }
}

```

```

    }

    private void Update ( ) {
        offset = offset + ( Time.deltaTime * scrollSpeed ) / 10f;
        mat.SetTextureOffset ( "_MainTex", new Vector2 ( offset, 0 ) );
    }
}

```

Πάμε τέλος σε δύο Scripts που δεν συνδέονται με κάποιο Object εξαρχής αλλά με τα Prefabs που χρησιμοποιούμε κατά την διάρκεια του παιχνιδιού. Το Prefab του εχθρού έχει ένα εκτενές Script που παρέχει τις λειτουργίες του. Θα παρουσιάσουμε πρώτα το Script με όνομα «Enemy».

Enemy.cs

```

using UnityEngine;
using UnityEngine.UI;

public class Enemy : MonoBehaviour {

```

Αρχικά περιέχει τα Reference που χρησιμοποιούμε στη συνέχεια, με πρώτα τα references σε άλλα Objects και Components εντός της σκηνής και στη συνέχεια τα references σε components του ίδιου του εχθρού, όπως και σε layers που χρησιμοποιούνται στα Colliders στη συνέχεια.

```

[ Header ( "References" ) ]
[ SerializeField ] private Transform groundCheck;
[ SerializeField ] private LayerMask groundLayer;
[ SerializeField ] private LayerMask enemyLayer;
[ SerializeField ] private Image healthBar;
[ SerializeField ] private GameObject bloodEffect;
Rigidbody2D rb;
Animator anim;

```

Στη συνέχεια παρουσιάζονται τα references των ήχων και των Components που παίζουν αυτούς τους ήχους.

```

[ Header ( "Audio References" ) ]
[ SerializeField ] private AudioSource deathAudioSource;
[ SerializeField ] private AudioClip [ ] painSounds;
[ Space ( 5 ) ]
[ SerializeField ] private AudioSource swordHitAudioSource;
[ SerializeField ] private AudioClip [ ] swordHitSounds;

```

Στην συνέχεια γίνεται η δήλωση κάποιων μεταβλητών με τα βασικά στατιστικά του εχθρού. Με το Attribute [**SerializedField**] μπορούμε να τους δίνουμε τιμές εντός του Editor χωρίς να τα κάνουμε καθολικές μεταβλητές.

```

[ Header ( "Variables" ) ]
[ SerializeField ] private float speed; //Enemy speed
[ SerializeField ] private int damage; //Enemy damage

```

```
[ SerializeField ] private int life; //Enemy life
private int maxLife;
```

Στο πεδίο με τίτλο «Circle» υπάρχουν References τα οποία χρησιμοποιήθηκαν για ένα πρόβλημα που αντιμετωπίσαμε. Όταν οι εχθροί έφταναν μπροστά στον παίκτη δεν σταματούσαν. Αυτές οι μεταβλητές και τα References είναι για τον κώδικα που αναπτύχθηκε για να ξεπεραστεί το συγκεκριμένο πρόβλημα.

```
[ Header ( "Circle" ) ]
[ SerializeField ] private Transform stopCheckUpper; //Circle start
[ SerializeField ] private Transform stopCheckMiddle; //Circle start
[ SerializeField ] private float length; //Circle radius
[ HideInInspector ] public bool isStoped; //Bool if player is stopped

private Collider2D enemyColliderA;
private Collider2D enemyColliderB;
private GameObject target; //Attack reference
private bool isGrounded; //Checking if enemy is grounded
private bool isFighting; //Checking if enemy is fighting
```

Στην αρχική συνάρτηση **Start();**, αρχικοποιούμε τα References για τα Components του Animator που χρησιμοποιούμε για τα animations, και του Rigidbody2D που χρησιμοποιούμε για να κινήσουμε τον εχθρό. Επίσης αρχίζουμε το πρώτο Animation, όπου ο εχθρός περπατάει προς την αριστερή πλευρά της οθόνης μας από ένα τυχαίο σημείο.

```
private void Start ( ) {
    maxLife = life;
    anim = GetComponent<Animator>( );
    rb = GetComponent<Rigidbody2D>( );
    anim.Play( "Enemy_Walking", -1, Random.Range ( 0.0f, 1.0f ) );
}
```

Εντός της **Update();** τρέχουν σε κάθε καρτέ πέντε διαφορετικές συναρτήσεις τις οποίες θα παρουσιάσουν παρακάτω αναλυτικότερα.

```
private void Update ( ) {
    GroundCheck ( );
    Walk ( );
    CheckForEnemy ( );
    Death ( );
    HealthRender ( );
}
```

Μέσα στην **OnTriggerEnter2D();** ελέγχουμε διαρκώς εάν κάποιο βέλος έχει πετύχει τον εχθρό. Αυτό γίνεται όταν ένα βέλος με tag *Arrow* μπει εντός του Collider του εχθρού. Με το που χτυπήσει το βέλος τον εχθρό, βρίσκουμε στην σκηνή μας στο Unity τον παίκτη και αφαιρούμε από τον εχθρό όσο Damage κάνει.

```
private void OnTriggerEnter2D ( Collider2D other ) {
    //Arrow physics
    if ( other.gameObject.tag.Equals ( "Arrow" ) ) {
        //Getting hit by an arrow
        life = life - GameObject.FindGameObjectWithTag( "Player" ).GetComponent<PlayerScript>().damage;
        Instantiate ( bloodEffect, new Vector3 ( transform.position.x, other.transform.position.y ),
            other.transform.rotation );
        deathAudioSource.PlayOneShot ( painSounds [ Random.Range ( 0, painSounds.Length ) ] );
    }
}
```

Εντός της **OnCollisionEnter2D()**; ελέγχουμε πότε ο εχθρός θα ακουμπήσει επάνω στον παίκτη. Μόλις γίνει αυτό, αρχίζει να παίζει το Animation της επίθεσης και περνάμε κάποιες μεταβλητές.

```
private void OnCollisionEnter2D ( Collision2D other ) {
    if ( other.gameObject.tag.Equals ( "Player" ) ) {
        anim.SetBool ( "EnemyAttacks", true );
        target = other.gameObject;
        isFighting = true;
        isStopped = true;
    }
}
```

Μέσα στην **OnCollisionExit2D()**; κάνουμε τον παίκτη να βγει από το Idle animation σε περίπτωση που ήταν σταματημένος.

```
private void OnCollisionExit2D ( Collision2D other ) {
    if ( other.gameObject.tag.Equals ( "Enemy" ) ) {
        anim.SetBool ( "EnemyIsStill", false );
    }
}
```

Η **HealthRenderer()**; γεμίζει την μπάρα επάνω από το κεφάλι του εχθρού με το ποσοστό ζωής που το απομένει.

```
private void HealthRender ( ) {
    if ( healthBar != null ) {
        healthBar.fillAmount = Mathf.Lerp ( healthBar.fillAmount, ( float ) life / maxLife, 0.05f );
    }
}
```

Στη συνέχεια η **GroundCheck()**; είναι υπεύθυνη για να ρυθμίζει το βάρος του εχθρού. Υπήρξε ένα μικρό πρόβλημα όταν ο εχθρός ανέβαινε απότομες ανηφόρες ή κατέβαινε απότομες κατηφόρες, είτε δεν μπορούσε να ανέβει είτε πετούσε αντίστοιχα. Το πρόβλημα λύθηκε αυξάνοντας ή μειώνοντας τη βαρύτητα ανάλογα με το εάν ο εχθρός ακουμπούσε στο έδαφος. Όσο ο εχθρός ακουμπάει είναι «ελαφρύς», μόλις πάει να «πετάξει» αυξάνεται η βαρύτητα και τον κολλάει και πάλι στον έδαφος.

```

private void GroundCheck ( ) {
    if ( groundCheck != null ) {
        isGrounded = Physics2D.OverlapCircle ( groundCheck.position, 0.1f, groundLayer );
        if ( isGrounded ) {
            rb.gravityScale = 1f;
        }
        else {
            rb.gravityScale = 8f;
        }
    }
}

```

Εντός της **Walk();** ελέγχουμε εάν ο εχθρός είναι σταματημένος πίσω από κάποιον άλλο εχθρό. Αν όχι τότε αρχίζει να κινείται προς τα αριστερά. Αν ναι, τότε τον αφήνουμε ακίνητο.

```

private void Walk ( ) {
    if ( isStopped == false ) {
        rb.velocity = new Vector2 ( -speed, 0 ); //Making enemy move
    }
    else {
        rb.velocity = Vector2.zero;
        rb.gravityScale = 0f;
    }
}

```

Η παρακάτω συνάρτηση **CheckForEnemy();** δημιουργήθηκε για να διορθωθεί το πρόβλημα ότι ο εχθρός δεν σταματούσε όταν ήταν πίσω από άλλο εχθρό. Δουλεύει κάνοντας έναν απλό έλεγχο σε μία μικρή απόσταση μπροστά από τον εχθρό για το εάν υπάρχει άλλο Object με tag *Enemy*. Εάν υπάρχει τότε εκτελείται το Idle animation και θέτει το Boolean IsStopped σε true. Με το που φύγει ο εχθρός από μπροστά, σταματάει το Idle animation και θέτει και πάλι το Boolean IsStopped σε false.

```

private void CheckForEnemy ( ) {
    if ( life > 0 ) {
        //Casting the 2 circles
        enemyColliderA = Physics2D.OverlapCircle ( stopCheckUpper.position, length, enemyLayer );
        enemyColliderB = Physics2D.OverlapCircle ( stopCheckMiddle.position, length, enemyLayer );

        //Upper circle
        if ( enemyColliderA && enemyColliderA.tag.Equals ( "Enemy" ) ) { //Check if circle is overlapping an enemy
            if ( enemyColliderA.GetComponent<Enemy>().isStopped == true ) { //Check if enemy in front is stopped
                anim.SetBool ( "EnemyIsStill", true ); //Playing idle animation
                isStopped = true; //Stopping
            }
        }
        else if ( isFighting == false ) { //If this enemy is not fighting
            anim.SetBool ( "EnemyIsStill", false ); //Play walking animation again and...
            isStopped = false; //...Start moving
        }

        //Middle circle
        if ( enemyColliderB && enemyColliderB.tag.Equals ( "Enemy" ) ) { //Same as above
            if ( enemyColliderB.GetComponent<Enemy>().isStopped == true ) {
                anim.SetBool ( "EnemyIsStill", true );
                isStopped = true;
            }
        }
        else if ( isFighting == false ) {
            anim.SetBool ( "EnemyIsStill", false );
        }
    }
}

```

```

        isStoped = false;
    }
}

```

Η **AttackEnable();** καλείται μέσω του Animator του Unity. κάθε φορά που ο εχθρός φτάνει σε σημείο του animation του, όπου το σπαθί είναι επάνω στον παίκτη μας. Είναι υπεύθυνη για το αίμα που βγάζει ο παίκτης για να δείξει ότι έχασε ζωή και για να μειώσει το αίμα του παίκτη. Επίσης παίζει και το ηχητικό εφέ του σπαθιού που χτυπάει τον παίκτη.

```

private void AttackEnable ( ) { //Called in animator
    Instantiate ( bloodEffect, new Vector3 ( target.transform.position.x + 0.4f, target.transform.position.y ),
Quaternion.Euler ( 0f, 0f, 180f ) );
    target.GetComponent<PlayerScript>().TakeDamage ( damage );
    swordHitAudioSource.PlayOneShot ( swordHitSounds [ Random.Range ( 0, swordHitSounds.Length ) ] );
}

```

Τέλος, εντός της **Death();** ελέγχουμε εάν ο εχθρός έχει πεθάνει. Εάν η ζωή του έχει φτάσει ή έχει πέσει κάτω από το μηδέν, τότε σταματάει, απενεργοποιούνται τα Colliders του για να μπορούν οι επόμενοι να περάσουν και παίζει το Death animation. Μετά από λιγότερο ένα δευτερόλεπτο τον διαγράφουμε από τον χάρτη για να μην ενοχλεί τον παίκτη.

```

private void Death ( ) {
    if ( life <= 0 ) {
        rb.velocity = new Vector2 ( 0f, 0f ); //Enemy death physics
        GetComponent<CapsuleCollider2D>().enabled = false; //Stopping enemy
        GetComponent<CircleCollider2D>().enabled = false; //Disabling colliders
        anim.SetBool( "EnemyIsDead", true ); //Disabling colliders
        rb.gravityScale = 0; //Playing death animation
        //Removing gravity

        foreach ( Transform child in transform ) {
            if ( child.name != "Canvas" ) { //Removing arrows
                Destroy ( child.gameObject );
            }
            else {
                Destroy ( child.gameObject, 0.09f );
            }
        }
        Destroy ( gameObject, 1f ); //Removing enemy's sprite
    }
}

private void OnDrawGizmosSelected ( ) {
    Gizmos.DrawWireSphere ( stopCheckUpper.position, length ); //Debug gizmos
    Gizmos.DrawWireSphere ( stopCheckMiddle.position, length );
    Gizmos.DrawWireSphere ( groundCheck.position, 0.1f );
}
}

```

Το τελευταίο Script του παιχνιδιού που θα παρουσιάσουμε είναι το Script του βέλους, το οποίο είναι υπεύθυνο για να αλλάζει το Sprite του βέλους όταν πέφτει κάτω ή χτυπάει κάποιον εχθρό, να κάνει την περιστροφή όπως πέφτει από τον ουρανό και τέλος να σβήνει από τον χάρτη το βέλος για να μην ενοχλεί.

Arrow.cs

```
using UnityEngine;

public class Arrow : MonoBehaviour {
    [ Header ( "References" ) ]
    [ SerializeField ] private Sprite HittSprite;
    [ SerializeField ] private GameObject trail;
    private Rigidbody2D rb;
    private bool hasHit;
```

Για αρχή, παραπάνω υπάρχουν κάποια References και μερικές μεταβλητές. Στη συνέχεια στην **Start()**, αρχικοποιούμε το Rigidbody2D που το χρησιμοποιούμε για να κουνήσουμε το βέλος μας.

```
private void Start ( ) {
    rb = GetComponent<Rigidbody2D>();
}
```

Εντός της **Update()**, εάν το βέλος μας είναι ακόμα στον αέρα, τότε υπολογίζουμε τη γωνία που πρέπει να έχει σε σχέση με το έδαφος.

```
private void Update ( ) {
    //Arrow rotation mathematics
    if ( hasHit == false ) {
        float angle = Mathf.Atan2( rb.velocity.y, rb.velocity.x ) * Mathf.Rad2Deg;
        transform.rotation = Quaternion.AngleAxis( angle, Vector3.forward );
    }
}
```

Μετά μέσα στην **OnTriggerEnter2D()**, η οποία καλείται όταν το βέλος έχει χτυπήσει κάπου γίνονται τα εξής: αλλάζει το Sprite σε αυτό στο οποίο είναι κρυμμένο το μπροστά μέρος του βέλους, σβήνει το άσπρο γραφικό που βγαίνει πίσω από το βέλος, παίζει ο ήχος για το έδαφος αν το έχει χτυπήσει και τέλος του απενεργοποιούνται τα Colliders και το βέλος κολλάει επάνω σε ό,τι χτύπησε. Μετά από είκοσι δευτερόλεπτα, σβήνει και αυτό από τον χάρτη.

```
private void OnTriggerEnter2D ( Collider2D other ) {
    GetComponent<SpriteRenderer>().sprite = HittSprite;    /*[Arrow on hit physics]*/
    Destroy ( trail, 0.1f );                                //Remove tip of arrow
    if ( other.gameObject.tag.Equals ( "Ground" ) ) {
        GetComponent<AudioSource>().pitch = Random.Range( 0.85f, 1.15f );
        GetComponent<AudioSource>().Play();
    }
    transform.parent = other.transform;                      //Child the arrow to the target
    GetComponent<CircleCollider2D>().enabled = false;       //Disabling the colliders of thrown arrows
    hasHit = true;                                           //Disabling rotation of arrow
    rb.velocity = Vector2.zero;                             //Stopping arrow
    rb.isKinematic = true;                                  //Stopping gravity for thrown arrow
    Destroy( gameObject, 20.0f );                            //Destroying arrow after 20 seconds
}
}
```

9) Γλωσσάρι – Επεξηγήσεις

Audio Clips	Κομμάτια ήχου
Banner	Πανό
Brusher	Βούρτσες
Canvas	Καμβάς, το μέρος της οθόνης όπου εμφανίζονται στοιχεία
CEO	Διευθύνων Σύμβουλος
Colliders	Εξάρτημα του Unity που προσφέρει ανίχνευση σύγκρουσης
Drag and drop	Σύρε και εναπόθεση
E-Sport	Ηλεκτρονικά σπορ
Event System	Εξάρτημα του Unity που είναι υπεύθυνο για τον χειρισμό συμβάντων του καμβά
Fire Rate	Ρυθμός εκτόξευσης βέλους
Game Manager	Αντικείμενο που έχει πάνω του κώδικα υπεύθυνο για την κατάσταση του παιχνιδιού
Gameplay	Τρόπος παιχνιδιού
HUD	Μέθοδος προβολής πληροφοριών οπτικά προς τον παίκτη
IDE	Ολοκληρωμένο περιβάλλον ανάπτυξης
Level	Επίπεδο
Level System	Σύστημα που ρυθμίζει το επίπεδο του παίκτη
Life	Ζωή
Loop	Βρόγχος
Mac OS X / Windows / Linux	Λειτουργικά συστήματα υπολογιστή
Objects	Αντικείμενα
Prefab	Επαναχρησιμοποιούμενα αντικείμενα
Rotation Point	Σημείο περιστροφής
Script	Αρχείο κώδικα
Soundtrack	Μουσική κομμάτι για υπόκρουση
Sprite	Εικόνα / Γραφικό
Text	Κείμενο

Tutorials	Σεμινάρια / βοηθήματα
Wave counter	Μετρητής κυμάτων (εχθρών)
XP	Experience Points / Πόντοι εμπειρίας

10) Βιβλιογραφία

- [1] M. Zyda, «From visual simulation to virtual reality to games.,» *Computer*, τόμ. 38, αρ. 9, p. 25–32, 2005.
- [2] Θ. Κ. Τσιάτσος, Εκπαιδευτικά Περιβάλλοντα Διαδικτύου: Χαρακτηριστικά & Είδη., Κάλλιπος, Ανοικτές Ακαδημαϊκές Εκδόσεις, 2016.
- [3] D. Djaouti, J. Alvarez, J. P. Jessel και O. Rampnoux, *Origins of serious games.*, London: Springer, 2011.
- [4] J. P. Gee, *What video games have to teach us about learning and literacy.*, New York: Association for Computing Machinery, 2003.
- [5] Α. Γιαννόπουλος και Χ. Παναγιωτακόπουλος, «Το ηλεκτρονικό παιχνίδι και το παιδί: Επιδράσεις και προοπτικές,» σε *Ένταξη και χρήση των ΤΠΕ στην Εκπαιδευτική Διαδικασία*, Πάτρα, 2011.
- [6] Γ. Κουτρομάνος και Κ. Νικολοπούλου, «Εκπαιδευτικά ψηφιακά παιχνίδια: Τι πιστεύουν οι φοιτητές για τη χρήση τους στη μαθησιακή διαδικασία,» σε *Ένταξη και χρήση των ΤΠΕ στην εκπαιδευτική διαδικασία*, Βόλος, 2009.
- [7] J. Haldon, *The Byzantine Wars*, Gloucestershire: The History Press, 2008.
- [8] J. K. Haas, *A history of the unity game engine*, Worcester: Worcester Polytechnic Institute, 2014.
- [9] T. Editors of Encyclopedia, «Adobe Illustrator,» *Encyclopedia Britannica*, 2022. [Ηλεκτρονικό]. Available: www.britannica.com/technology/Adobe-Illustrator. [Πρόσβαση 22 September 2022].
- [10] T. Editors of Encyclopedia, «Adobe Photoshop,» *Encyclopedia Britannica*, 2022. [Ηλεκτρονικό]. Available: www.britannica.com/technology/Adobe-Photoshop. [Πρόσβαση 22 September 2022].