ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«ΚΑΤΑΣΚΕΥΗ ΒΙΝΤΕΟΠΑΙΧΝΙΔΙΟΥ “SLAVE” 2D ΧΑΡΤΟΠΑΙΧΝΙΔΙ ΜΕ UNITY ΜΗΧΑΝΗ ΚΑΙ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ ΣΕ C#»

Του φοιτητή
Οδυσσέα Καραστεργίου
Αρ. Μητρώου: 144322

Επιβλέπων
Γεώργιος Κοκκώνης
Επίκουρος Καθηγητής

Ημερομηνία 18-01-2024

Τίτλος Δ.Ε.: Κατασκευή βιντεοπαιχνιδιού “Slave” 2D χαρτοπαιχνίδι με Unity μηχανή και προγραμματισμό σε C#
Κωδικός Δ.Ε.: 21195
Ονοματεπώνυμο φοιτητή: Καραστεργίου Οδυσσέας
Ονοματεπώνυμο εισηγητή: Γεώργιος Κοκκώνης
Ημερομηνία ανάληψης Δ.Ε.: 18-03-2022
Ημερομηνία περάτωσης Δ.Ε.: 10-12-2023

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Πρόλογος

Η παρούσα διπλωματική εργασία επιλέχθηκε στο πλαίσιο της ενασχόλησής μου τόσο με τα βιντεοπαιχνίδια όσο και με την κατασκευή τους. Πηγή έμπνευσης υπήρξε η επιθυμία αναβίωσης της αίσθησης των κλασικών παιδικών/εφηβικών παιχνιδιών με κάρτες, όπως και η πρώιμη μεταφορά τους στον ψηφιακό κόσμο. Βασικός στόχος είναι η ανάπτυξη ενός προσιτού για το χρήστη περιβάλλοντος ψυχαγωγίας ψηφιοποιώντας ένα κλασικό παιχνίδι καρτών, αξιοποιώντας ταυτόχρονα τις νέες δυνατότητες που έχουν παρουσιαστεί τα τελευταία χρόνια στην κατασκευή βιντεοπαιχνιδιών. Για την υλοποίηση αυτής της ιδέας, και μετά από σύγκριση διάφορων επιλογών, επιλέχθηκε το περιβάλλον Unity. Κύριο λόγο αποτέλεσε η ταχύτερη ανάπτυξη του παιχνιδιού στο συγκεκριμένο περιβάλλον, καθώς οι δυνατότητες της Unity ταιριάζουν απρόσκοπτα με τις προγραμματιστικές απαιτήσεις για τη μεταφορά του συγκεκριμένου παιχνιδιού στον ψηφιακό κόσμο.

Περίληψη

Το θέμα της πτυχιακής εργασίας αφορά τη διαδικασία δημιουργίας ενός δισδιάστατου (2D) παιχνιδιού, ονομαζόμενο “Slave”, με την αξιοποίηση μιας συγκεκριμένης μηχανής ανάπτυξης παιχνιδιού, της Unity. Στην παρούσα εργασία παρουσιάζονται αναλυτικά τα στάδια και πρωτόκολλα που ακολουθήθηκαν για τη δημιουργία του παιχνιδιού, τα προβλήματα που εμφανίστηκαν και αντιμετωπίστηκαν καθώς και μελλοντικές δυνατότητες περαιτέρω αναβάθμισης του παιχνιδιού.

Η εργασία αποτελείται από τρία βασικά μέρη: i) την έρευνα και επιλογή μεταξύ των μηχανών ανάπτυξης παιχνιδιών, ii) την κατανόηση του επιλεγμένου παιχνιδιού και τέλος iii) την ανάπτυξη του στο επιλεγμένο περιβάλλον. Αναλυτικότερα, αρχικά γίνεται μία εισαγωγή και σύγκριση μεταξύ των διαθέσιμων τρόπων και μηχανών ανάπτυξης παιχνιδιών, ενώ δίνεται ιδιαίτερη έμφαση στη Unity και στους λόγους επιλογής της. Έχοντας ως στόχο την κατανόηση του περιβάλλοντος της Unity, γίνεται μια λεπτομερής περιγραφή του (Unity Integrated development environment - IDE), των βασικών χαρακτηριστικών και λειτουργιών του “interface” της, ενώ ιδιαίτερη προσοχή δίνεται σε βασικές έννοιες που θα εφαρμοστούν κατά την ανάπτυξη του επιλεγμένου παιχνιδιού. Σημαντικό στάδιο στην ανάπτυξη του κώδικα και συνεπώς στη δημιουργία του παιχνιδιού, αποτελεί η κατανόηση της φύσης, των κανόνων και του σκοπού του “Slave” οι οποίοι αναλύονται πριν το κυριότερο στάδιο της εργασίας. Επόμενο στάδιο αποτελεί η δημιουργία και επεξεργασία του βασικού ‘project’ (board, τμήματα και τρόποι με τα οποία ο χρήστης μπορεί να αλληλοεπιδράσει) και όλων των επιμέρους τμημάτων του (κανόνες παιχνιδιού, interface, μενού). Τελικό στάδιο αποτελεί η δοκιμή από την πλευρά του παίκτη και η διόρθωση πιθανών προβλημάτων και λεπτομερειών καταλήγοντας στην ολοκλήρωση του παιχνιδιού.

Τα κύρια αποτελέσματα της διπλωματικής συνοψίζονται στη δημιουργία ενός λειτουργικού και εύχρηστου ψηφιακού παιχνιδιού καρτών και στην επιτυχημένη χρήση της Unity, το οποίο μετά από δοκιμή, προτείνεται για την αντιμετώπιση παρόμοιου είδους προκλήσεων.

«DEVELOPMENT OF “SLAVE” VIDEO-GAME 2D CARD GAME WITH UNITY ENGINE AND C#»

«Odysseas Karastergiou»

Abstract

The present thesis deals with the developing process of a 2D card game, called “Slave”, using the Unity2D game engine. Here are presented in detail the stages and protocols followed during the game development, the problems faced as well as possible future upgrades of the game.

The thesis is comprised of three main parts: i) the research, comparison and choosing between the different game development engines, ii) the understanding of the chosen game and finally iii) its development in the chosen environment.

In more detail there is an introduction comparing the different available game engines, while specific emphasis is given on Unity as well as to the reasons for choosing to work with it. The Unity Integrated Development Environment (IDE), its basic features and interface are thoroughly explained. Fundamental Unity concepts are explored, especially those necessary to establish a foundation for constructing our chosen card game, “Slave”. Another main step for the developing of the code and consequently the creation of the game, is the understanding of the concept, rules, card gameplay and overall purpose of “Slave”, which are explored before the final and main part of this work. Next comes the creation and processing of the main project (board game, parts of the game with which the user can interact) and all the secondary parts (game rules, interface, menu). The final step is to test the game and deal with possible problems and details resulting in the completion of the game.

The main results of the thesis are concluded in the creation of a working and user-friendly digital card game as well as in the successful application of the Unity game engine, which after trial is recommended for dealing with similar tasks and challenges.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον κ.Δεληγιάννη που δέχτηκε να εκτελέσω το θέμα που είχα σκεφτεί
Και φυσικά μετά τον κ.Κοκκώνη που ανέλαβε να συνεχίσει την πτυχιακή μου ώστε να φέρω εις πέρα το έργο μου.

Περιεχόμενα

Κατάλογος Σχημάτων

Κεφάλαιο 1ο: Game engines and Unity 2D Engine

1.1 Εισαγωγή

Συγκρίνοντας τα πλεονεκτήματα και τα μειονεκτήματα των υπόλοιπων μηχανών ανάπτυξης παιχνιδιών (Game Engines) επιλέξαμε να χρησιμοποιήσουμε τη Unity2D η οποία αυτήν την στιγμή είναι από τις πιο δημοφιλής μηχανές ανάπτυξης παιχνιδιών στον κόσμο. Έγινε χρήση της δωρεάν έκδοσης η οποία διανέμεται ελεύθερα. Επιπλέον, η Unity παρέχει και μια premium έκδοση με έξτρα περιεχόμενο, assets και λειτουργίες, τα οποία όμως στη συγκεκριμένη περίπτωση δεν θεωρήθηκαν απαραίτητα.

Η χρήση της Unity διαθέτει πληθώρα πλεονεκτημάτων αρχικά λόγω της ευελιξίας της, της ευκολίας χρήσης και της ισχυρής κοινότητας που την υποστηρίζει. Επιτρέπει στους προγραμματιστές να δημιουργήσουν εντυπωσιακά παιχνίδια με ελάχιστη προηγούμενη εμπειρία, παρέχοντας παράλληλα πολλές δυνατότητες για εξελιγμένη ανάπτυξη. Με εργαλεία όπως το Unity Editor, το οποίο επιτρέπει τον οπτικό σχεδιασμό και τη γρήγορη προεπισκόπηση, η Unity αποτελεί ιδανική επιλογή για όσους θέλουν να δημιουργήσουν εναρκτήρια, ποιοτικά παιχνίδια 2D με ελάχιστο χρόνο και προσπάθεια.

Ως διαφορετική επιλογή για την ανάπτυξη της πτυχιακής μου, εξέτασα το ενδεχόμενο δημιουργίας του βιντεοπαιχνιδιού με την μηχανή Unreal Engine, αλλά μετά από ανάλυση κατέληξα στην Unity καθώς είναι η μόνη που προσφέρει την δυνατότητα εναλλαγής ‘Game State’ κατά την διάρκεια ενός παιχνιδιού το οποίο ήταν κρίσιμο κομμάτι δημιουργίας του. Γι’αυτό τον λόγο άλλα διάσημα παιχνίδια όπως τα ‘Hearthstone’, ‘Uno’ έχουν αναπτυχθεί σε Unity.

1.2 Unity IDE

Το Unity IDE (Integrated Development Environment) είναι το περιβάλλον ανάπτυξης που παρέχεται από την Unity Technologies για τη δημιουργία παιχνιδιών και εφαρμογών. Είναι ένα ολοκληρωμένο πακέτο που συμπεριλαμβάνει διάφορα εργαλεία και λειτουργίες για την ανάπτυξη παιχνιδιών σε διάφορες πλατφόρμες με τις πιο δημοφιλείς να περιλαμβανομένων τα desktop, κινητά, κονσόλες τ των και VR.

Τα βασικά χαρακτηριστικά του Unity IDE περιλαμβάνουν:

1. Unity Editor: Ένα οπτικό περιβάλλον που επιτρέπει το σχεδιασμό, την επεξεργασία και την προεπισκόπηση των παιχνιδιών.

2. Σύστημα Ανάπτυξης (Development System): Περιλαμβάνει γλώσσες προγραμματισμού όπως τη C# και τη JavaScript για τη δημιουργία λειτουργικών στοιχείων του παιχνιδιού.

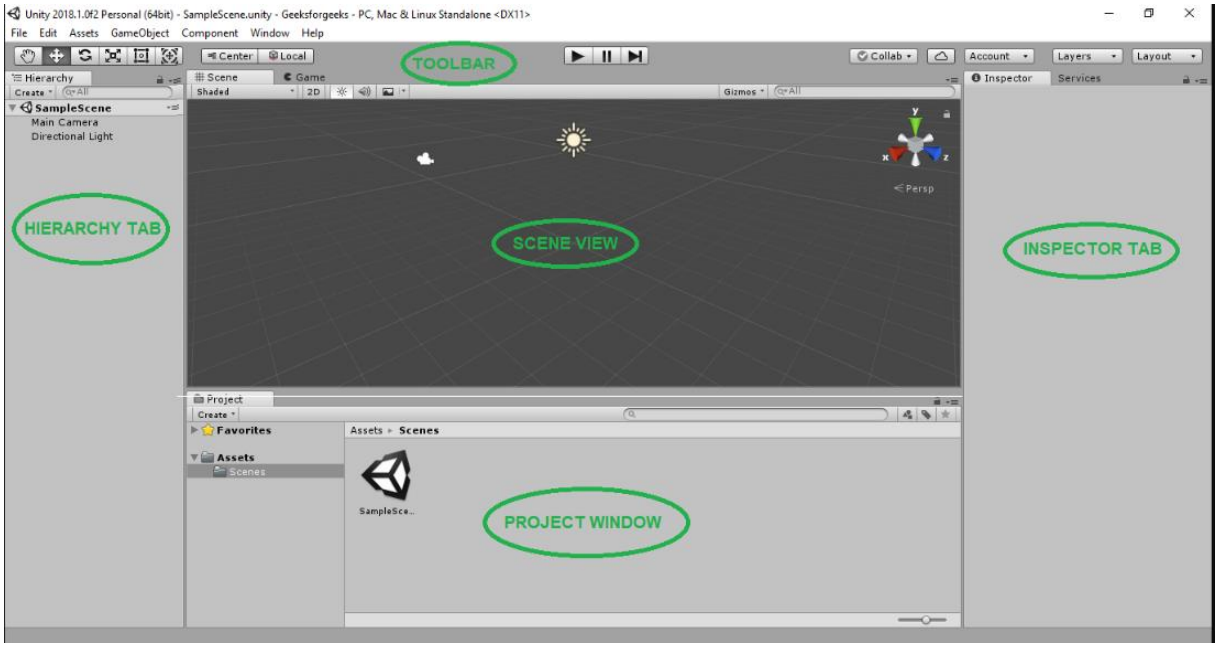
3. Κινηματογραφικά Εργαλεία (Graphics Tools): Παρέχει εργαλεία για τη δημιουργία και τη διαχείριση γραφικών, όπως 2D και 3D αντικείμενα, εφέ, φωτισμό και πολλά άλλα.

4. Ήχος και Μουσική: Υποστηρίζει την ενσωμάτωση ήχου και μουσικής στα παιχνίδια.

5. Φυσική και Κινηματικά: Περιλαμβάνει εργαλεία για τη διαχείριση φυσικής μηχανής, κινηματικά αντικείμενα και άλλα.

1.3 Unity Interface

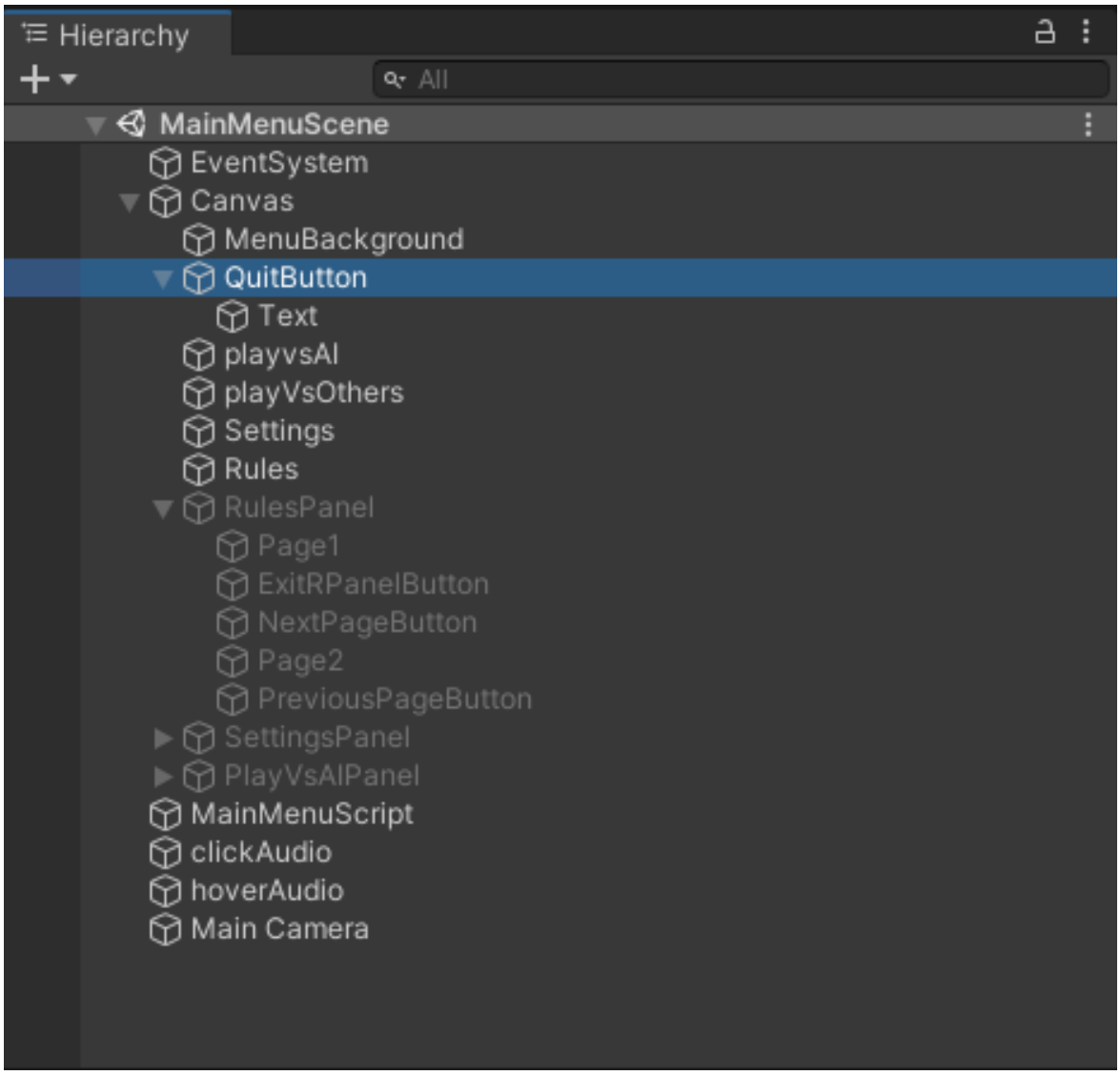
Η πρώτη εικόνα με την οποία βρίσκεται κάποιος αντιμέτωπος μόλις θέσει σε λειτουργία τη Unity παρουσιάζεται παρακάτω (Σχήμα 1.1). Αυτή θα αποτελέσει και τη βασική επιφάνεια εργασίας μας η οποία διαχωρίζεται σε πέντε σημεία (πράσινα τμήματα) που αναλύονται στη συνέχεια.



Σχήμα 1.1: Περιβάλλον της Unity

1.3.1 Hierarchy Tab

Το Hierarchy tab (Σχήμα 1.2) αντιπροσωπεύει ένα από τα βασικά παράθυρα του Unity Editor. Απεικονίζει ιεραρχικά όλα τα αντικείμενα που ανήκουν στη σκηνή του παιχνιδιού ή της εφαρμογής που αναπτύσσουμε. Το Hierarchy παρέχει επισκόπηση του δέντρου αντικειμένων, όπως γραφικά, φωτισμός, κάμερες και άλλα, επιτρέποντας στους προγραμματιστές και σχεδιαστές να διαχειρίζονται και να οργανώνουν τα στοιχεία του παιχνιδιού τους με ευκολία. Από το Hierarchy tab, μπορούμε να επεξεργαστούμε, να ομαδοποιήσουμε και να διαχειριστούμε τα αντικείμενα, επιτρέποντας την ομαλή ανάπτυξη και συντήρηση του παιχνιδιού ή της εφαρμογής μας.



Σχήμα 1.2: Hierarchy Tab

Στο παραπάνω σχήμα μπορούμε να δούμε ότι το GameObject (βλ. 1.2) RulesPanel είναι πατέρας των Page1, Page2, ExitRPanelButton, NextPageButton και PreviousPageButton, όπως επίσης ότι κάποια από αυτά τα GameObject είναι με γκρίζο φόντο το οποίο υποδηλώνει ότι είναι σε κατάσταση την οποία δεν μπορούμε να επεξεργαστούμε (disabled state).

1.3.2 Scene View

Το Scene View (Σχήμα 1.3) αντιπροσωπεύει ένα από τα κύρια παράθυρα του Unity Editor. Παρέχει μια διεπαφή οπτικοποίησης του περιβάλλοντος ανάπτυξης, επιτρέποντας στους χρήστες να οπτικοποιούν και να επεξεργάζονται τον τρέχοντα κόσμο του παιχνιδιού ή της εφαρμογής τους. Μέσα από το Scene View, οι προγραμματιστές και σχεδιαστές μπορούν να τοποθετούν, να ρυθμίζουν και να αλληλεπιδρούν με 3D ή 2D αντικείμενα, φωτισμό, κάμερες και άλλα στοιχεία του περιβάλλοντος. Επιτρέπει επίσης τον προσδιορισμό και την επεξεργασία της θέασης της σκηνής, ενισχύοντας την οπτική ανάλυση και τον έλεγχο κατά τη διάρκεια της διαδικασίας ανάπτυξης του παιχνιδιού.

Σχήμα 1.3: Scene View

1.3.3 Inspector Tab

Το Inspector tab (Σχήμα 1.4) αποτελεί ένα κεντρικό παράθυρο στο Unity Editor. Παρέχει λεπτομερείς πληροφορίες και ρυθμίσεις για το επιλεγμένο αντικείμενο στη σκηνή ή στο Hierarchy tab. Μέσω του Inspector, οι χρήστες μπορούν να προσαρμόσουν τις ιδιότητες των αντικειμένων, όπως μετατόπιση, κλίμακα, περιστροφή, χρώμα, υλικά, συμπεριφορές, και άλλες παράμετροι. Το Inspector αποτελεί απαραίτητο εργαλείο για τον προγραμματιστή ή το σχεδιαστή, καθώς επιτρέπει τη γρήγορη ρύθμιση και προσαρμογή των χαρακτηριστικών των αντικειμένων, ενισχύοντας τη διαδικασία ανάπτυξης και επιτρέποντας την άμεση οπτικοποίηση των αλλαγών που γίνονται στο παιχνίδι ή την εφαρμογή.

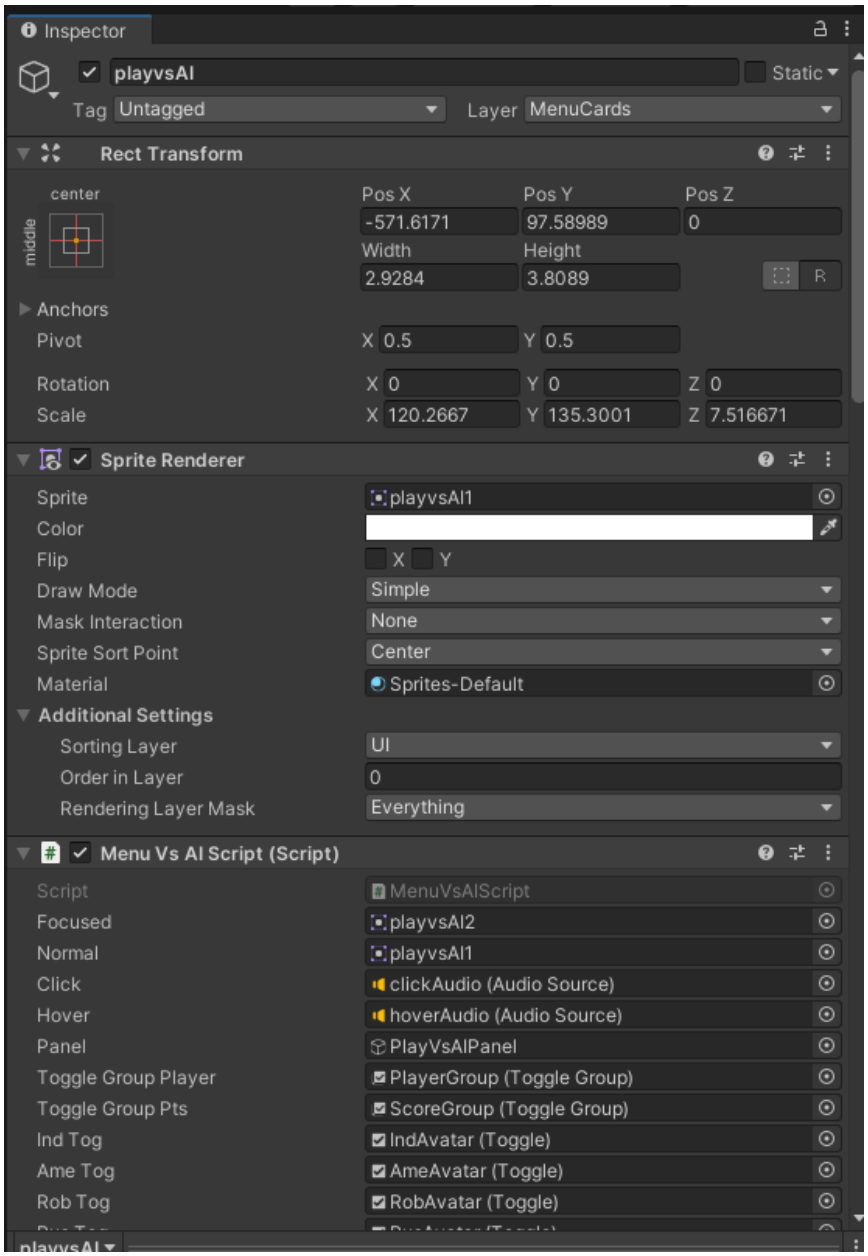
Στο παρακάνω σχήμα μπορούμε να δούμε το inspector tab ενός GameObject που έχουμε επιλέξει, στην παρούσα φάση το ‘playVsAi’. Μέσα βλέπουμε ότι περιέχει το ‘Rect Transform’ που χρησιμοποιείται για τον έλεγχο και διαχείριση της διάταξης, θέσης, μεγέθους και κλίμακας ενός Game Object. Αμέσως επόμενο, το ‘Sprite Renderer’ που χρησιμοποιείται για την απεικόνιση των γραφικών μας. Κάποια χαρακτηριστικά του περιλαμβάνουν :

1. Sprite: Ορίζει την εικόνα που θα απεικονίζεται από το Sprite Renderer.

2. Color: Καθορίζει το χρώμα του Sprite.

3. Sorting Layer: Ορίζει τη στρώση ταξινόμησης του Sprite για τον έλεγχο της σειράς απεικόνισης.

4. Order in Layer: Καθορίζει τη θέση του Sprite εντός της επιλεγμένης στρώσης ταξινόμησης.

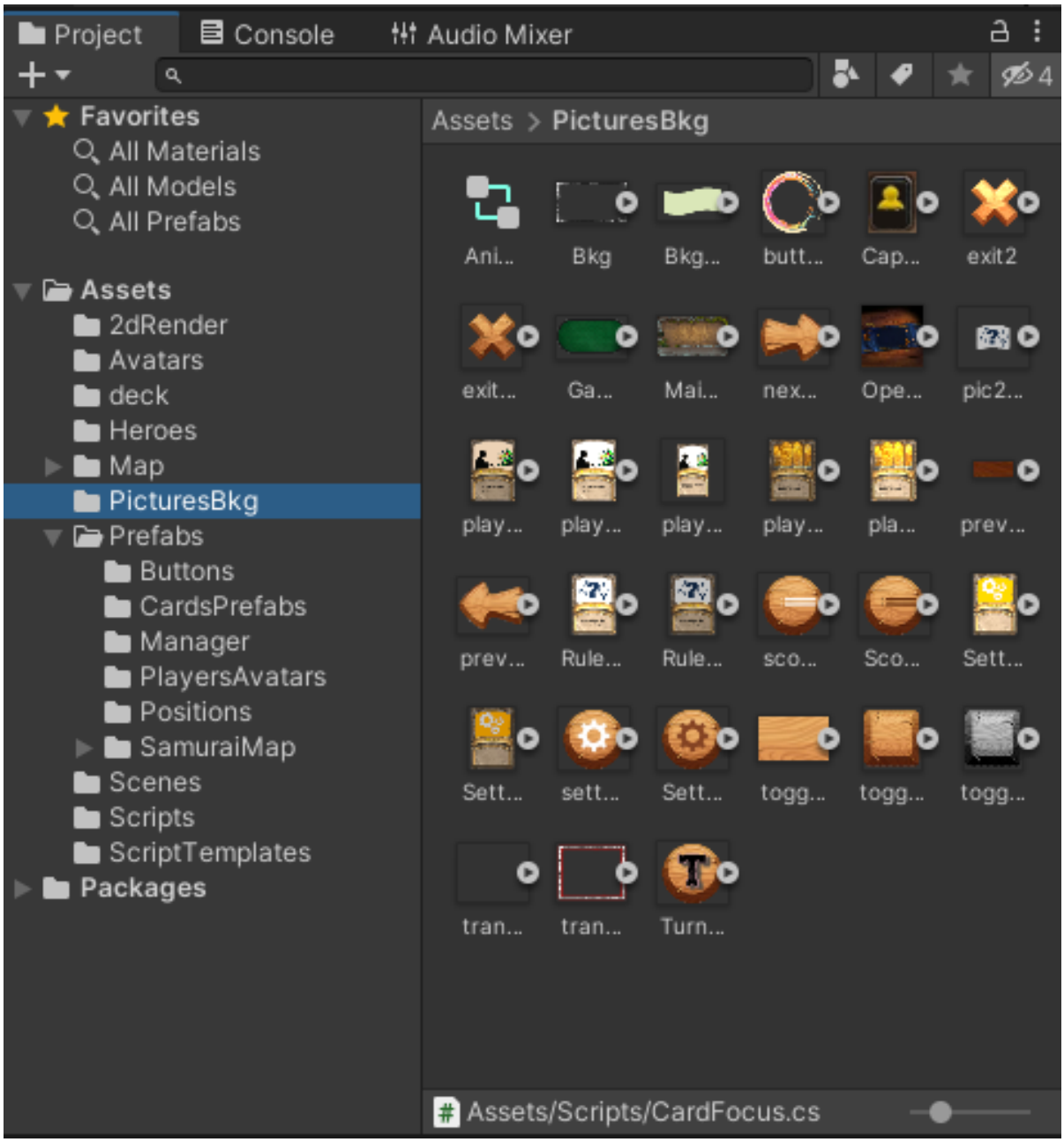


Σχήμα 1.4: Inspector Tab

1.3.4 Project window

Το Project Window (Σχήμα 1.5) στη Unity αντιπροσωπεύει ένα βασικό παράθυρο του Unity Editor, παρέχοντας μια οργανωμένη θεώρηση των αρχείων και των πόρων που σχετίζονται με ένα έργο ανάπτυξης. Μέσω αυτού του παραθύρου, οι χρήστες μπορούν να διαχειριστούν τα assets του προγράμματος, τους εικόνες, ήχοι, κώδικας, σκηνές και άλλα.

Το Project Window παρέχει δυνατότητες φιλτραρίσματος, αναζήτησης, και οργάνωσης των αρχείων σε φακέλους, επιτρέποντας τους προγραμματιστές και σχεδιαστές να εντοπίζουν γρήγορα και αποτελεσματικά τους πόρους που χρειάζονται. Επιπλέον, επιτρέπει τη διαχείριση των συμπεριλαμβανομένων πακέτων, των εξωτερικών πόρων και την εύκολη προβολή και επεξεργασία των αρχείων του έργου.



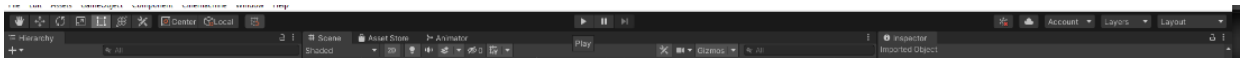
Σχήμα 1.5: Project Window

Στο παραπάνω σχήμα (Σχήμα 1.5) έχουμε επιλέξει στο Project window, μέσα στα assets του project μας, τον φάκελο 'PicturesBkg', στον οποίο έχουμε προσθέσει συγκεκριμένες εικόνες που θα χρησιμοποιήσουμε στο project μας.

1.3.5: Toolbar

Το Toolbar (Σχήμα 1.6) αναφέρεται σε μια σειρά εργαλείων και επιλογών που βρίσκονται στο πάνω μέρος του Unity Editor. Αυτό το παράθυρο παρέχει γρήγορη πρόσβαση σε κοινά εργαλεία και λειτουργίες που χρησιμοποιούνται κατά τη διάρκεια της διαδικασίας ανάπτυξης ενός παιχνιδιού ή μιας εφαρμογής στο Unity.

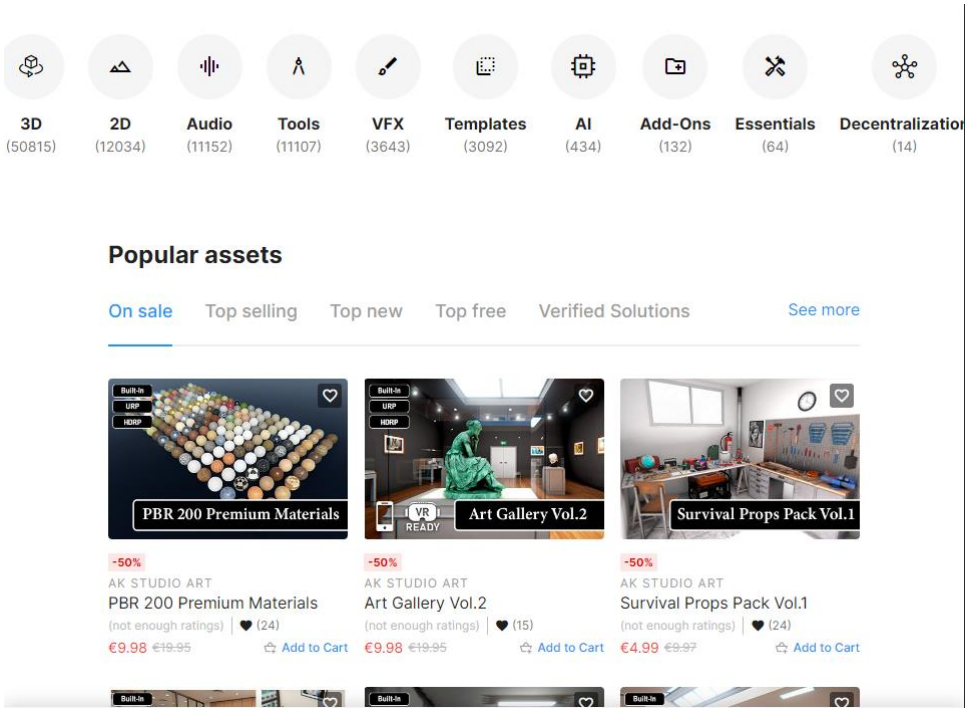
Το Toolbar περιλαμβάνει συνήθως κουμπιά για την εκτέλεση, τον διακόπτη ανάπτυξης (Play/Stop), την αποθήκευση των αλλαγών, καθώς και εργαλεία για την επιλογή, τη μετακίνηση, την περιστροφή και άλλες κοινές λειτουργίες. Το Toolbar εξυπηρετεί τη γρήγορη πρόσβαση και την απλούστευση των βασικών εργαλείων κατά τη διάρκεια της ανάπτυξης και του debugging ενός έργου στο Unity Editor.



Σχήμα 1.6: Toolbar

1.3.6: Unity Asset Store

Το Asset Store (Σχήμα 1.7) αποτελεί μια διαδικτυακή πλατφόρμα όπου οι χρήστες μπορούν να αγοράζουν, να πωλούν και να μοιράζονται πόρους και πακέτα που βελτιώνουν τη διαδικασία ανάπτυξης παιχνιδιών και εφαρμογών στην Unity. Το Asset Store περιλαμβάνει πληθώρα assets, όπως 3D μοντέλα, textures, animations, scripts, εφέ, ήχοι, και άλλα, που μπορούν να ενσωματωθούν στα έργα των χρηστών για την επίτευξη εντυπωσιακών και επαγγελματικών αποτελεσμάτων. Οι χρήστες μπορούν επίσης να δωρεάν λήψη πόρων και να αξιολογήσουν, κριτικάρουν και αναφέρουν προβλήματα σχετικά με τα assets. Με τη διευκόλυνση της ανταλλαγής και της συνεργασίας, το Asset Store συμβάλλει στη δημιουργία πιο εύκολης και αποτελεσματικής ανάπτυξης στο πλαίσιο της Unity κοινότητας.



Σχήμα 1.7: Asset Store

Η παραπάνω εικόνα εμφανίζει ένα κομμάτι της σελίδας assetstore.unity.com. Το asset store σε προηγούμενες εκδόσεις πρόσφερε αυτή την δυνατότητα μέσα από την μηχανή της Unity.

1.4 Έννοιες της Unity

Για να μπορέσει κάποιος να χρησιμοποιήσει την μηχανή της Unity για δημιουργία κάποιου παιχνιδιού, θα πρέπει να έχει μία καλή εικόνα από τις βασικές έννοιες της. Πολλές από αυτές τις έννοιες θα τις χρησιμοποιήσουμε και εμείς στην δημιουργία του δικού μας παιχνιδιού. Παρακάτω θα αναλύσουμε μερικές από αυτές λεπτομερώς ώστε να καταλάβουμε καλύτερα τι και πως θα χρησιμοποιούμε κατά την διάρκεια του project.

1.4.1 Game Object

Τα Game Objects [1] στη Unity αντιπροσωπεύουν τα βασικά στοιχεία στη σκηνή ενός παιχνιδιού ή μιας εφαρμογής. Αποτελούν τα οικοδομικά στοιχεία που αποτελούν τα υποκείμενα της ανάπτυξης, και περιλαμβάνουν 3D ή 2D αντικείμενα, φωτισμό, κάμερες, και άλλα. Κάθε Game Object έχει ένα Transform που καθορίζει τη θέση, την κλίμακα και την περιστροφή του στο χώρο 3D.

Τα Game Objects μπορούν να έχουν επιπλέον συστατικά, όπως scripts, colliders, και renderers, τα οποία καθορίζουν τη συμπεριφορά, τη φυσική αλληλεπίδραση, και την εμφάνισή τους. Το Unity χρησιμοποιεί ένα σύστημα συστατικών (Component-based architecture), επιτρέποντας την ευελιξία και τη δημιουργία πολύπλοκων σκηνών μέσω της σύνθεσης διαφορετικών συστατικών σε κάθε Game Object.

1.4.2 Components

Στη Unity, τα Components [2] είναι οι βασικές μονάδες που προσδίδουν συγκεκριμένες λειτουργίες και χαρακτηριστικά σε ένα GameObject. Τα Components προστίθενται στα GameObjects για να τα καθιστούν δραστικά και να ορίζουν τη συμπεριφορά τους. Κάθε Component εκτελεί συγκεκριμένες λειτουργίες και συμβάλλει στη συνολική δυναμική του παιχνιδιού ή της εφαρμογής.

Κάποια κοινά Components περιλαμβάνουν:

1. Transform: Καθορίζει τη θέση, την περιστροφή και την κλίμακα του GameObject.

2. Collider: Προστίθεται για την ανίχνευση συγκρούσεων με άλλα αντικείμενα στη σκηνή.

3. Rigidbody: Προσδίδει φυσική συμπεριφορά σε ένα GameObject, επιτρέποντας του να αντιδρά σε δυνάμεις όπως η βαρύτητα και η κίνηση.

4. Script (MonoBehaviour): Προγραμματικός κώδικας που προστίθεται για να επεκτείνει τη λειτουργικότητα του GameObject, προσθέτοντας προσαρμοσμένη λογική και αλληλεπίδραση.

5. Sprite Renderer: Επιτρέπει την απεικόνιση 2D γραφικών (sprites) σε ένα GameObject.

6. Audio Source: Προσδίδει ήχο σε ένα GameObject.

Η συνδυασμένη χρήση διαφορετικών Components επιτρέπει τη δημιουργία πολύπλοκων και ποικιλόμορφων συμπεριφορών για τα GameObjects σε ένα παιχνίδι ή μια εφαρμογή.

1.4.3 2D Colliders

Οι 2D Colliders [3] στη Unity είναι στοιχεία που προστίθενται σε 2D Game Objects και χρησιμοποιούνται για την ανίχνευση συγκρούσεων σε ένα περιβάλλον 2D παιχνιδιού. Αυτά τα στοιχεία επιτρέπουν στο παιχνίδι να ανταποκρίνεται στη φυσική αλληλεπίδραση μεταξύ των διάφορων 2D αντικειμένων.

Ορισμένοι από τους κύριους 2D Colliders που μπορούμε να χρησιμοποιήσουμε περιλαμβάνουν:

1. Box Collider 2D: Παρέχει έναν αόρατο παραλληλεπίπεδο περίγραμμα γύρω από ένα αντικείμενο με το οποίο ανιχνεύεται η συγκρούση.

2. Circle Collider 2D: Αντιπροσωπεύει έναν αόρατο κύκλο γύρω από το αντικείμενο.
3. Polygon Collider 2D: Μας επιτρέπει να καθορίσουμε ένα περίπλοκο πολύγωνο σχήμα γύρω από το αντικείμενο.

4. Edge Collider 2D: Καθορίζει ένα αόρατο τετράγωνο ή οριζόντιο/κατακόρυφο γραμμικό collider.

Αυτοί οι 2D Colliders επιτρέπουν στον κώδικα του παιχνιδιού να ανταποκρίνεται σε συγκρούσεις, να εκτελεί διάφορες δράσεις όταν ανιχνεύεται σύγκρουση με άλλα αντικείμενα, όπως είναι η αλλαγή του σκορ, η μείωση της ζωής και άλλα.

1.4.4 Prefabs (Prefabricated Object)

Τα Prefabs [4] (συντομογραφία για τον όρο "prefabricated") στη Unity είναι προ-δημιουργημένα αντικείμενα ή σύνολα στοιχείων που μπορούν να χρησιμοποιηθούν ξανά σε διάφορα σημεία του έργου μας. Τα Prefabs είναι χρήσιμα για την αποφυγή διπλού κώδικα και τη διευκόλυνση της ενημέρωσης των αλλαγών σε πολλαπλά στοιχεία του παιχνιδιού μας.
Ορισμένα βασικά σημεία σχετικά με τα Prefabs:

1. Επαναχρησιμοποίηση: Μπορούμε να δημιουργήσουμε ένα Prefab για ένα στοιχείο ή ομάδα στοιχείων και να το χρησιμοποιήσουμε σε διάφορα μέρη του παιχνιδιού μας.

2. Ευκολία Ενημέρωσης: Αν τροποποιήσουμε ένα Prefab, οι αλλαγές αυτές θα εφαρμοστούν αυτόματα σε όλα τα αντικείμενα που χρησιμοποιούν το ίδιο Prefab.

3. Αποθήκευση στην Ιεραρχία: Τα Prefabs εμφανίζονται στο Project Window και μπορούν να τοποθετηθούν στην Ιεραρχία (Hierarchy) όπως οποιοδήποτε άλλο αντικείμενο.

4. Περιέχουν Συστατικά (Components): Ένα Prefab μπορεί να περιλαμβάνει τυπικά διάφορα συστατικά όπως Colliders, Scripts, Renderers κ.ά.

Η χρήση Prefabs είναι σημαντική για την οργάνωση του κώδικα και των στοιχείων στο Unity και συμβάλλει στη διευκόλυνση της διαχείρισης και της ανάπτυξης του παιχνιδιού ή της εφαρμογής μας.

1.4.5 Scenes

Στη Unity, τα Scenes [5] (σκηνές) αντιπροσωπεύουν διαφορετικά περιβάλλοντα ή επίπεδα στα οποία εξελίσσεται το παιχνίδι ή η εφαρμογή μας. Κάθε σκηνή μπορεί να περιέχει διάφορα Game Objects με τις αντίστοιχες συνιστώσες και λογική που προγραμματίζουμε. Οι σκηνές επιτρέπουν τη διαχωρισμένη διαχείριση των διαφορετικών μερών του παιχνιδιού ή της εφαρμογής μας.

Ορισμένες βασικές πληροφορίες σχετικά με τις σκηνές:

1. Δημιουργία Σκηνής: Μπορούμε να δημιουργήσουμε νέα σκηνή πηγαίνοντας στο "File" > "New Scene" στο Unity Editor.

2. Εναλλαγή μεταξύ Σκηνών: Μπορούμε να εναλλάσσουμε μεταξύ σκηνών κατά τη διάρκεια της εκτέλεσης του παιχνιδιού ή της εφαρμογής. Αυτό γίνεται με τη χρήση κώδικα, συνήθως μέσω του συστατικού SceneManager.

Με τη χρήση σκηνών, μπορούμε να οργανώσουμε το παιχνίδι ή την εφαρμογή μας σε διάφορα περιβάλλοντα, όπως επίπεδα, μενού, και άλλα.

1.4.6 Tags και Layers

Τα Tags [6] και τα Layers στη Unity είναι δύο συστήματα που χρησιμοποιούνται για τον έλεγχο και τη διαχείριση των Game Objects στο περιβάλλον της σκηνής. Και τα δύο παρέχουν ευελιξία και οργάνωση κατά τη διάρκεια της ανάπτυξης του παιχνιδιού ή της εφαρμογής.

Tags:

1. Ορισμός Ομάδων: Τα Tags επιτρέπουν τον ορισμό ομάδων Game Objects με κοινές ιδιότητες ή χρήσεις. Για παράδειγμα, μπορούμε να ορίσουμε ένα Tag "Player" για όλα τα αντικείμενα που αντιπροσωπεύουν τον παίκτη στο παιχνίδι μας.

2. Ευκολία Επιλογής: Τα Tags καθιστούν ευκολότερη την επιλογή και την αλληλεπίδραση με ομάδες Game Objects μέσω κώδικα ή στο Unity Editor.

3. Layers: Τα Layers χρησιμοποιούνται για να ορίσουν τη θέση και τη στοίβαση των Game Objects σε διαφορετικά επίπεδα σε ένα 3D περιβάλλον.

4. Φυσική Αλληλεπίδραση: Τα Layers είναι χρήσιμα για τον έλεγχο της φυσικής αλληλεπίδρασης, όπως η ανίχνευση συγκρούσεων μεταξύ συγκεκριμένων ομάδων Game Objects.

Και τα δύο συστήματα συμβάλλουν στην οργάνωση του παιχνιδιού, την ευκολότερη διαχείριση και τον προγραμματισμό, καθώς και στον προσδιορισμό των αλληλεπιδράσεων μεταξύ διάφορων στοιχείων της σκηνής.

1.4.7 GameState

Το GameState [7] αναφέρεται στην κατάσταση ή την τρέχουσα κατάσταση ενός παιχνιδιού. Αυτό το στοιχείο αντιπροσωπεύει την ενεργή φάση του παιχνιδιού και την κατάσταση των διαφόρων στοιχείων που συμμετέχουν. Οι διαφορετικές καταστάσεις μπορεί να συμπεριλαμβάνουν τον τρέχοντα επίπεδο, το σκορ, τον αριθμό των ζωών, τον χρόνο και άλλες πληροφορίες που ορίζουν την εξέλιξη του παιχνιδιού.

Ο προγραμματιστής συχνά χρησιμοποιεί το GameState για να παρακολουθεί και να ελέγχει την πορεία του παιχνιδιού, καθώς και για να προσαρμόζει τη συμπεριφορά του παιχνιδιού ανάλογα με τις συνθήκες και τις επιλογές του παίκτη. Η διαχείριση του GameState συμβάλλει στην καλύτερη οργάνωση και ευελιξία του κώδικα, επιτρέποντας την εύκολη προσαρμογή στις απαιτήσεις και τις αλλαγές του παιχνιδιού κατά τη διάρκεια της ανάπτυξης και της συντήρησης.

1.4.8 Console

Η ‘console’ [8] της Unity, είναι ένα παράθυρο στο Unity Editor που παρέχει πληροφορίες, ειδοποιήσεις, και αναφορές λαθών κατά τη διάρκεια της ανάπτυξης και του τεστ του παιχνιδιού ή της εφαρμογής μας. Είναι ένα σημαντικό εργαλείο για τον προγραμματιστή και τον game developer. Εκτός από το Unity Console, υπάρχει επίσης το Console στη γλώσσα προγραμματισμού C#, το οποίο χρησιμοποιείται για να εμφανιστούν μηνύματα εξόδου κατά τη διάρκεια της εκτέλεσης του κώδικα. Το Unity Console έχει τις εξής δυνατότητες:

1. Εμφανίζει Μηνύματα Συστήματος: Το Unity Console εμφανίζει πληροφορίες, προειδοποιήσεις και σφάλματα που σχετίζονται με την ανάπτυξη και την εκτέλεση του παιχνιδιού.

2. Ανίχνευση Λαθών: Το Console βοηθά στην ανίχνευση και την αποκατάσταση σφαλμάτων κατά τη διάρκεια της ανάπτυξης του παιχνιδιού.

3. Καταγραφή Μηνυμάτων Debugging: Οι προγραμματιστές χρησιμοποιούν το Console για την εμφάνιση μηνυμάτων debugging κατά την ανάπτυξη του κώδικα.

1.4.9 Transform

Το Transform[9**]** είναι ένα component που αντιπροσωπεύει τη θέση, την περιστροφή και την κλίση ενός Game Object. Κάθε αντικείμενο σε μια σκηνή έχει ένα Transform, και αυτό χρησιμοποιείται για να ορίσει τη θέση και τον προσανατολισμό του στον 3D χώρο. Το Transform χρησιμοποιείται και στην Unity2D, αλλά ορισμένες λειτουργίες του εφαρμόζονται σε ένα επίπεδο.

Ορισμένες βασικές έννοιες για το Transform:

1. Θέση (Position): Οι συντεταγμένες XYZ που καθορίζουν τη θέση του αντικειμένου στον χώρο.
2. Περιστροφή (Rotation): Οι γωνίες Euler που καθορίζουν τον προσανατολισμό του αντικειμένου.

3. Κλίση (Scale): Οι συντεταγμένες XYZ που καθορίζουν το μέγεθος του αντικειμένου σε κάθε άξονα.

Ορισμένες χρήσιμες μεθόδοι και ιδιότητες του Transform:

1. Translate(Vector3 translation): Μετακινεί το αντικείμενο κατά τη διάρκεια της εκτέλεσης.
2. Rotate(Vector3 eulerAngles): Περιστρέφει το αντικείμενο κατά δεδομένες γωνίες Euler.
3. localPosition, localRotation, localScale: Οι τοπικές τιμές των ιδιοτήτων θέσης, περιστροφής και κλίσης, αναφερόμενες στον τοπικό χώρο του αντικειμένου.
4. parent: Ο γονέας του αντικειμένου στον οποίο ανήκει, εφόσον υπάρχει.

Η χρήση του Transform είναι κρίσιμη για την καθοριστική διαχείριση της θέσης και του προσανατολισμού των Game Objects στο περιβάλλον της Unity.

1.4.10 Rigidbody

Το Rigidbody [10] είναι ένα component που προστίθεται σε ένα Game Object για να του παρέχει φυσική συμπεριφορά. Εφαρμόζει τους νόμους της φυσικής στο αντικείμενο, καθιστώντας το αληθοφανές και αλληλεπιδραστικό στον χώρο του παιχνιδιού.

Ορισμένα βασικά χαρακτηριστικά του Rigidbody:

1. Φυσική Κίνηση (Physics Motion): Το Rigidbody επιτρέπει στο αντικείμενο να αντιδρά στις δυνάμεις και τις συγκρούσεις με άλλα αντικείμενα στο περιβάλλον.
2. Σύγκρουση και Ανίχνευση: Ένα αντικείμενο με Rigidbody αντιδρά στη σύγκρουση με άλλα αντικείμενα και μπορεί να ανιχνεύει αν τα Game Objects βρίσκονται σε επαφή.
3. Επιτάχυνση, Ταχύτητα, Δύναμη: Το Rigidbody διαχειρίζεται την επιτάχυνση, την ταχύτητα και τις δυνάμεις που επηρεάζουν την κίνηση του αντικειμένου.
4. Βαρύτητα (Gravity): Το Rigidbody υποστηρίζει την εφαρμογή της βαρύτητας, καθιστώντας το αντικείμενο να πέφτει στον χώρο μέχρι να βρει κάποιο άλλο αντικείμενο.

Για να προσθέσουμε ένα Rigidbody σε ένα αντικείμενο στη Unity, επιλέγουμε το αντικείμενο, πηγαίνουμε στο παράθυρο "Inspector", και πατάμε το κουμπί "Add Component". Στη συνέχεια, επιλέγουμε "Physics" και προσθέτουμε το "Rigidbody".
Η χρήση του Rigidbody είναι σημαντική για τη δημιουργία ρεαλιστικής κίνησης και αλληλεπίδρασης στον εικονικό χώρο της Unity.

1.4.11 Animations

Τα animations [11] (κινήσεις ή κινούμενα σχέδια) είναι σημαντικό στοιχείο για τη δημιουργία δυναμικών και εντυπωσιακών παιχνιδιών ή εφαρμογών. Τα animations επιτρέπουν την κίνηση και τη μεταμόρφωση των Game Objects κατά τη διάρκεια του χρόνου. Οι κινήσεις αυτές μπορούν να αφορούν τη θέση, την περιστροφή, την κλίση, την κλίμακα ή άλλες ιδιότητες.

Βασικά στοιχεία σχετικά με τα animations στη Unity:

1. Animator Component: Το Animator είναι ένα component που προστίθεται σε ένα Game Object και διαχειρίζεται τα animations του αντικειμένου. Περιέχει καταστάσεις (states) και μεταβάσεις (transitions) μεταξύ αυτών των καταστάσεων.

2. Animation Clip: Ένα Animation Clip είναι μια προκαθορισμένη κινηματογραφική σκηνή που καταγράφει την κίνηση ή τη μεταμόρφωση ενός αντικειμένου. Κάθε Animation Clip μπορεί να παίζεται και να διακόπτεται ανάλογα με τις συνθήκες που ορίζει ο προγραμματιστής.
3. Animator Controller: Ο Animator Controller είναι ένας γραφικός πίνακας που περιέχει τις διάφορες καταστάσεις και μεταβάσεις μεταξύ τους. Συνδέεται με το Animator και διαχειρίζεται πώς θα εκτελούνται τα animations.
4. Κλειδιά Κινήσεων (Keyframes): Τα animations ορίζονται μέσω κλειδιών κινήσεων, τα οποία καθορίζουν την κατάσταση του αντικειμένου σε συγκεκριμένα χρονικά σημεία.

Η Unity παρέχει ένα ευέλικτο σύστημα animations που επιτρέπει στους προγραμματιστές να δημιουργήσουν πολύπλοκες κινήσεις και εφέ με ελάχιστη προγραμματιστική προσπάθεια.

1.4.12 Audio

Το AudioSource [12] είναι ένα component που χρησιμοποιείται για την αναπαραγωγή ήχου. Προστίθεται σε ένα Game Object και δίνει τη δυνατότητα στο αντικείμενο να παίζει ήχους κατά τη διάρκεια της εκτέλεσης του παιχνιδιού.

Ορισμένα σημαντικά στοιχεία του AudioSource:

1. Audio Clip: Αναπαράγει έναν ήχο που καθορίζεται από ένα AudioClip. Το AudioClip περιέχει την πραγματική ηχητική πληροφορία που θα ακουστεί.
2. Volume: Καθορίζει την ένταση του ήχου που θα αναπαραχθεί.
3. Pitch: Αλλάζει το ύψος του ήχου, επιτρέποντας προσαρμογή της ταχύτητας αναπαραγωγής.
4. Loop: Επιτρέπει στον ήχο να επαναλαμβάνεται αυτόματα.
5. Spatial Blend: Καθορίζει πόσο ο ήχος επηρεάζεται από τη θέση και την κατεύθυνση του ακροατή στον εικονικό χώρο 3D.
6. Play, Pause, Stop: Προσφέρει μεθόδους για τον έλεγχο της αναπαραγωγής του ήχου.

Ο προγραμματιστής μπορεί να χρησιμοποιήσει το AudioSource για να ενσωματώσει ήχο στο παιχνίδι του, όπως μουσική, εφέ ήχου, φωνητικά και άλλα. Επιπλέον, η Unity παρέχει API για τον προγραμματιστή, προκειμένου να ελέγχει δυναμικά τα χαρακτηριστικά του AudioSource κατά τη διάρκεια του παιχνιδιού.

1.4.13 Camera

Το component Camera [13**],** χρησιμοποιείται για την απεικόνιση του περιβάλλοντος του παιχνιδιού ή της εφαρμογής μας. Είναι υπεύθυνο για τον τρόπο με τον οποίο ο παίκτης βλέπει το παιχνίδι ή τη σκηνή μας κατά τη διάρκεια της εκτέλεσης.

Ορισμένα σημαντικά χαρακτηριστικά του Camera:

1. Προβολή (Projection): Οι κύριοι τύποι προβολής είναι η προοπτική (perspective) και η ορθογώνια (orthographic). Η προοπτική παρέχει βάθος και ρεαλιστική αναπαράσταση, ενώ η ορθογώνια παρέχει μια επίπεδη αναπαράσταση.
2. Φακός (Field of View): Καθορίζει πόσο ευρύ είναι το οπτικό πεδίο της κάμερας. Έχει εφαρμογή μόνο σε προοπτικές κάμερες.
3. Θέση και Προσανατολισμός: Καθορίζει πού βρίσκεται η κάμερα και προς πού κοιτάει. Συνήθως, οι πληροφορίες αυτές διαχειρίζονται αυτόματα ανάλογα με τη σκηνή μας.

4. Κόσμος vs. Κάμερα (World vs. Camera Space): Ορίζει εάν οι συντεταγμένες αντικειμένων μετριούνται σε σχέση με τον κόσμο ή την κάμερα.
5. Culling Masks: Χρησιμοποιείται για να ελέγξουμε ποια αντικείμενα θα είναι ορατά από την κάμερα με βάση τα επίπεδα (Layers) τους.
6. Κλιμάκωση (Clipping Planes): Καθορίζει την ελάχιστη και μέγιστη απόσταση από την κάμερα στην οποία αντικείμενα θα είναι ορατά.

1.4.14 Scripting

Το scripting [14] στη Unity αναφέρεται στη χρήση γλωσσών προγραμματισμού, όπως ο C# ή ο JavaScript, για τη δημιουργία λογικής και λειτουργικότητας σε παιχνίδια ή εφαρμογές που αναπτύσσονται στο περιβάλλον της Unity.

Τα scripts χρησιμοποιούνται για να διαχειριστούν τη συμπεριφορά των αντικειμένων, την αλληλεπίδραση του παίκτη, την ανίχνευση συγκρούσεων, τη δημιουργία γραφικών εφέ, και πολλά άλλα.

Η Unity παρέχει ένα πλούσιο σύστημα API που επιτρέπει την πρόσβαση και τον έλεγχο διάφορων στοιχείων του παιχνιδιού, ενώ οι προγραμματιστές μπορούν να χρησιμοποιήσουν το scripting για να προσαρμόσουν τη λειτουργία της εφαρμογής σύμφωνα με τις ανάγκες τους.

1.4.15 Αλληλεπίδραση C# με Unity

Η σύνδεση της C# με την Unity γίνεται με τη δημιουργία και την επεξεργασία scripts στο περιβάλλον ανάπτυξης της Unity. Βασικά βήματα για την δημιουργία και χρήση ενός script είναι:

Δημιουργία Script:

- Στο Project panel της Unity, επιλέγουμε το φάκελο όπου θέλουμε να δημιουργήσουμε το script.
- Δεξί κλικ -> Create -> C# Script.
- Ονομάζουμε το script και πατάμε Enter.
- Επεξεργασία Script:
- Διπλό κλικ στο script που δημιουργήσαμε για να το ανοίξουμε στον επεξεργαστή κώδικα.

Προσθήκη Κώδικα:

- Στο script, μπορούμε να προσθέσουμε κώδικα C# για τον έλεγχο του παιχνιδιού.

Σύνδεση με Αντικείμενα:

- Μπορούμε να συνδέσουμε το script με αντικείμενα στο παιχνίδι (π.χ., χαρακτήρες, αντικείμενα) σύροντας το script πάνω σε ένα αντικείμενο στο Hierarchy panel.

Εκτέλεση Κώδικα:

- Ο κώδικας μας θα εκτελείται αυτόματα κατά τη διάρκεια του παιχνιδιού όταν συμβούν συγκεκριμένα γεγονότα.

Αυτά τα βήματα αποτελούν ένα βασικό σενάριο σύνδεσης της C# με την Unity. Μπορούμε να χρησιμοποιήσουμε το script για να ελέγξουμε τη συμπεριφορά των αντικειμένων, να αντιδράει σε συγκεκριμένα γεγονότα, και να δημιουργήσουμε προσαρμοσμένη λειτουργικότητα στο παιχνίδι μας.

1.4.16 User Interface

Το User Interface (UI) στη Unity αναφέρεται στον τρόπο με τον οποίο οι χρήστες αλληλεπιδρούν με το παιχνίδι ή την εφαρμογή.

Στην Unity, η δημιουργία UI γίνεται με τον σχεδιασμό και την τοποθέτηση διάφορων στοιχείων, όπως κουμπιά, κείμενα και πάνελ, χρησιμοποιώντας το σύστημα Canvas.

Το UI μπορεί να προγραμματιστεί με scripts σε γλώσσες όπως ο C#, προσθέτοντας λειτουργίες όπως αλλαγή κατάστασης, ανταλλαγή δεδομένων με τον παίκτη, και διαχείριση γεγονότων. Με τη χρήση του Unity Editor, οι προγραμματιστές μπορούν να προσαρμόσουν την εμφάνιση και τη συμπεριφορά του UI, δημιουργώντας έναν χρήσιμο και ελκυστικό χρήστη.

Στο πλαίσιο της Unity, το "Screen" αναφέρεται συνήθως στην οθόνη του υπολογιστή ή της συσκευής στην οποία εκτελείται η εφαρμογή. Οι πληροφορίες σχετικά με την οθόνη μπορούν να είναι χρήσιμες για τον κώδικα που αφορά τη διαχείριση του γραφικού περιβάλλοντος, το μέγεθος του παραθύρου και άλλες γραφικές ρυθμίσεις.

1.4.17 Screen Resolution

Σχετικά με την ανάλυση οθόνης (screen resolution), μπορούμε να αποκτήσουμε πληροφορίες για το μέγεθος της οθόνης στην Unity χρησιμοποιώντας κώδικα. Για παράδειγμα, μπορούμε να χρησιμοποιήσουμε την κλάση Screen για να πάρουμε πληροφορίες όπως το πλάτος, το ύψος και το refresh rate της οθόνης.

1.4.18 Time

Στο πλαίσιο της Unity, ο όρος "time" [16] χρησιμοποιείται συχνά για να αναφερθεί στον χρόνο που σχετίζεται με τον χρόνο εκτέλεσης της εφαρμογής ή του παιχνιδιού. Η Unity παρέχει μερικές χρήσιμες κλάσεις και μεθόδους που σχετίζονται με το χρόνο.

1. Time.deltaTime:

Η μέθοδος Time.deltaTime [15] επιστρέφει τον χρόνο που πέρασε από τον τελευταίο ’’tick’ του παιχνιδιού ή της εφαρμογής. Χρησιμοποιείται συχνά για τον υπολογισμό της προόδου των κινούμενων αντικειμένων ή άλλων διαδικασιών που εξαρτώνται από τον χρόνο.

2. Time.time:

Η μεταβλητή Time.time αντιπροσωπεύει τον συνολικό χρόνο που έχει περάσει από την έναρξη της εφαρμογής. Χρησιμοποιείται για την καταγραφή της συνολικής διάρκειας του παιχνιδιού ή για την εκτέλεση εργασιών που σχετίζονται με τον συνολικό χρόνο.

3. Time.timeScale:

Η μεταβλητή Time.timeScale ελέγχει τον ρυθμό αναπαραγωγής του χρόνου στο παιχνίδι. Μπορεί να χρησιμοποιηθεί για την επιτάχυνση ή την επιβράδυνση της εκτέλεσης του παιχνιδιού.

Κεφάλαιο 2ο: Slave - Κανόνες και κάρτες

2.1 Εισαγωγή

Το “Slave” είναι ένα επιτραπέζιο παιχνίδι με κάρτες που αποτελείται από 5 παίκτες, όπου ο καθένας στο ξεκίνημα κάθε παιχνιδιού ή γύρου, έχει από 6 κάρτες στο χέρι του. Οι κάρτες αυτές είναι οι κλασσικές κάρτες που έχουν όλες οι τράπουλες όμως για το συγκεκριμένο παιχνίδι επιλέγονται αυτές από το νούμερο 7 και πάνω (Σχήμα 2.1).

Σχήμα 2.1: Game Cards

Στο κεντρικό μενού επίσης για την καλύτερη εμπειρία του χρήστη επιλέξαμε να προσθέσουμε μερικά εικονίδια (avatars) για να υπάρχει διαφοροποίηση μεταξύ των παικτών.

Κάθε φωτογραφία στο μενού αντιστοιχεί σε μία διαφορετική μέσα στο παιχνίδι.

Σχήμα 2.2: Avatar παίκτη 'Ινδίας'

Σχήμα 2.3: Avatar παίκτη 'Αμερικής'

Σχήμα 2.4: Avatar παίκτη 'Robot'

Σχήμα 2.5: Avatar παίκτη 'Ρωσίας'

Σχήμα 2.6: Avatar παίκτη 'Samurai'

Σχήμα 2.7: Avatar παίκτη 'Zombie '

Σχήμα 2.8: Avatar παίκτη 'Zulu'

2.2 Κανόνες παιχνιδιού

Στην αρχή του παιχνιδιού η τράπουλα αυτών των φύλλων που αποτελείται από 32 κάρτες, μοιράζεται στους παίκτες έως ότου ο καθένας έχει από 6 κάρτες. Από αυτές τις 6 οι 2 παραμένουν κλειστές και χρησιμοποιούνται στον επόμενο γύρο (Σχήμα 2.9).

Ο κάθε παίκτης με κυκλική σειρά ρολογιού ρίχνει τις κάρτες που έχει με τους εξής κανόνες:

1. Εάν ο προηγούμενος παίκτης έχει ρίξει φύλλο με μονό αριθμό, τότε οι επόμενοι μπορούν να ρίξουν αποκλειστικά μονά φύλλα μεγαλύτερης αξίας.

2. Η παραπάνω επιλογή παιχνιδιού μπορεί να γίνει και για την ρίψη δύο ή και τριών ίδιας αξίας φύλλων.

3. Εάν περάσει μια γύρα παικτών και δεν έχει απαντήσει κανένας παίκτης στο φύλλο του πρώτου που έριξε, τότε έρχεται ξανά η σειρά του πρώτου παίκτη ο οποίος έχει τη δυνατότητα να ρίξει ό,τι θέλει.

4. Αν κάποιος έχει στην κατοχή του 4 φύλλα ίδιας αξίας (π.χ. τέσσερα 8αρια) τότε μπορεί να τα ρίξει στην σειρά του και με αυτό τον τρόπο να δημιουργήσει τη λεγόμενη ‘ανταρσία’ με την οποία μετά από αυτό το σημείο, για το υπόλοιπο του γύρου, οι κάρτες έχουν αντίστροφη αξία.

Ο σκοπός του παιχνιδιού είναι να είσαι ο πρώτος που θα μείνει χωρίς φύλλα, αλλά και να ‘βγεις’ όσο πιο γρήγορα γίνεται καθώς οι θέσεις μετράνε στον επόμενο γύρο, όπως και οι πόντοι που μαζεύεις.

Σχήμα 2.9: Το ταμπλό ενός εν διάρκεια παιχνιδιού

2.3 Ιδιότητες παιχνιδιού

Αφού τελειώσει ο πρώτος γύρος και μοιράσουμε πάλι τις κάρτες, πριν ξεκινήσουμε να παίζουμε, αρχίζει η φάση του ‘taxing’, κατά την οποία οι παίκτες πρέπει να ανταλλάξουν κάρτες ανάλογα με την θέση τους στον προηγούμενο γύρο. Η διαδικασία έχει ως εξής:

1. Ο παίκτης που στον προηγούμενο γύρο τερμάτισε πρώτος με αυτόν που τερμάτισε τελευταίος: Ο πρώτος πρέπει να δώσει τις 2 μικρότερες σε αξία κάρτες του στον τελευταίο και ταυτόχρονα ο τελευταίος να δώσει τις 2 μεγαλύτερες σε αξία κάρτες του στον πρώτο.

2. Την ίδια διαδικασία ακολουθούν ο 2ος σε θέση με τον προτελευταίο, αλλά με την ανταλλαγή μίας κάρτας.

3. Ο 3ος έχει την επιλογή να ανταλλάξει 2 από τις κάρτες που έχει στην κατοχή του με τις δύο εναπομείνασες κάρτες που αφήσαμε κρυμμένες στο τέλος του μοιράσματος.

Μετά από αυτήν την διαδικασία συνεχίζει το παιχνίδι με τον τελευταίο του προηγούμενου γύρου να παίζει πρώτος.

2.4 Συμπεράσματα

Το “Slave” είναι ένα παιχνίδι όπου οι παίκτες δεν έχουν πληροφορίες για τις αντίπαλες κάρτες αλλά μπορούν να μαζεύουν τις πληροφορίες από τις κάρτες που παίζονται καθ’ όλη την διάρκεια του παιχνιδιού. Έτσι το παιχνίδι απαιτεί από τον παίκτη όχι μόνο να υπολογίζει τις επόμενες κινήσεις του άλλα και των αντιπάλων του, αλλά και να κρατάει με μνήμη τις εναπομείναντες κάρτες ώστε να παίξει τις υπόλοιπες του αντίστοιχα στον δρόμο για την κατάκτηση της καλύτερης δυνατής θέσης.

Κεφάλαιο 3ο: Ανάπτυξη του παιχνιδιού με Unity

3.1 Δημιουργία των σκηνών

Σε αυτό το κεφάλαιο θα δούμε τον τρόπο με τον οποίο υλοποιήσαμε το παιχνίδι μέσα από την μηχανή της Unity. Το παιχνίδι δημιουργήθηκε σε περιβάλλον 2D, δηλαδή σε δισδιάστατο κόσμο. Ο παίκτης μας θα βρίσκεται στο μπροστινό μέρος της οθόνης, έχοντας τα φύλλα του ανοιχτά. Αντίπαλοί του θα είναι ο υπολογιστής για καθένα από τους 4 άλλους παίκτες (Σχήμα 2.2).

3.1.1 Opening Scene

Αρχικά, με το που ανοίξει ο χρήστης το παιχνίδι μεταφερόμαστε στην πρώτη εισαγωγική σκηνή πριν το menu (Σχήμα 3.1).

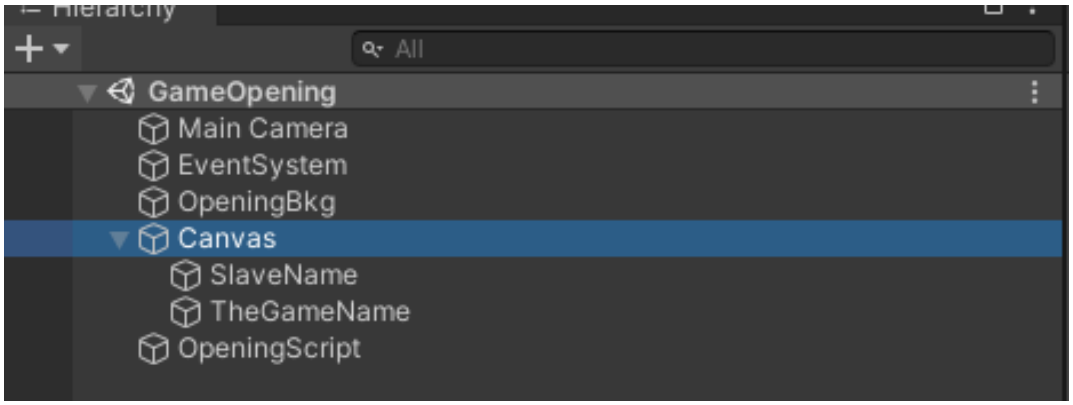
Σχήμα 3.1: Εισαγωγική εικόνα του παιχνιδιού

Με την εκκίνηση του συγκεκριμένου scene ‘Game Opening’ τρέχει το ‘OpeningScript’ script, όπου με την μέθοδο Invoke, η οποία χρησιμεύει στην εκκίνηση μίας διαδικασίας μετά από το χρονικό περιθώριο που θα δώσουμε, μετά από 3 δευτερόλεπτα μεταφέρουμε τον χρήστη στην επόμενη σκηνή του κεντρικού menu του παιχνιδιού (MainMenuScene) (Σχήμα 3.2).

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

0 references | 0 changes | 0 authors, 0 changes
public class OpeningScript : MonoBehaviour
{
    // Start is called before the first frame update
    0 references | 0 changes | 0 authors, 0 changes
    void Start()
    {
        Invoke("MainMenuGo", 3.0f);
    }

    // Update is called once per frame
    0 references | 0 changes | 0 authors, 0 changes
    void Update()
    {

    }

    0 references | 0 changes | 0 authors, 0 changes
    void MainMenuGo()
    {
        SceneManager.LoadScene("MainMenuScene");
    }
}
```

Σχήμα 3.2: Κώδικας OpeningScript

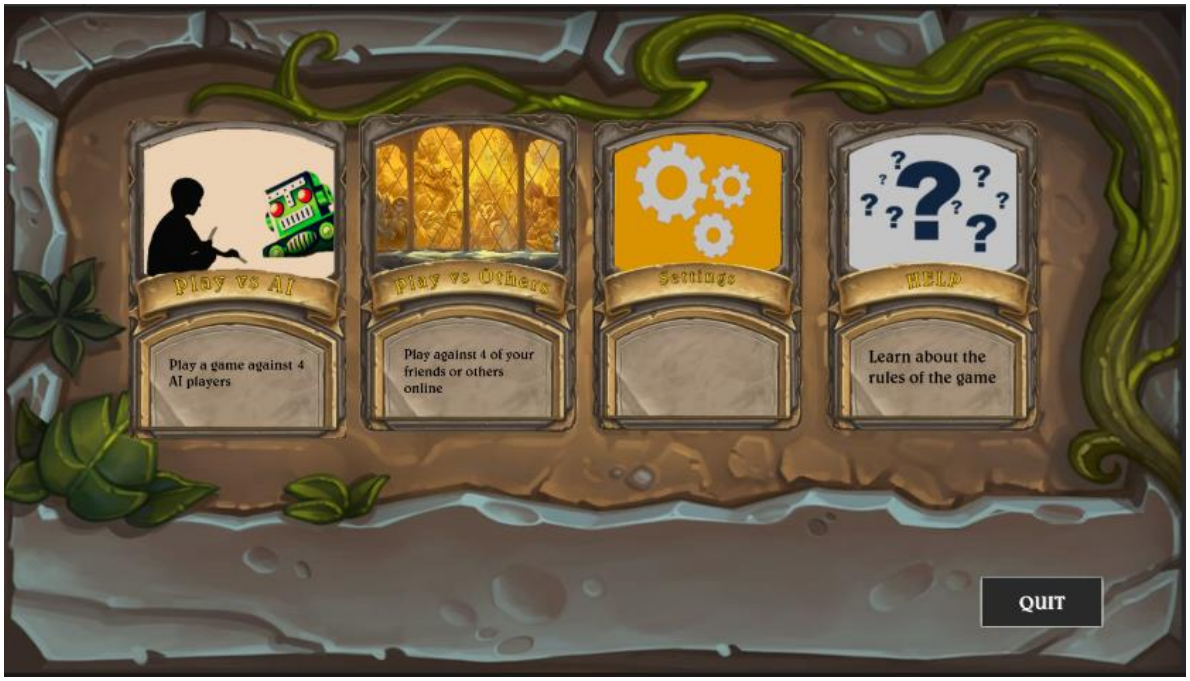
Μέσα στο Unity Editor για την συγκεκριμένη σκηνή έχουμε GameObject την εικόνα που θα χρησιμοποιήσουμε για την σκηνή και ένα canvas, το οποίο περιέχει τα 2 text πάνω στην εικόνα (Σχήμα 3.3).



Σχήμα 3.3: Game Object Tree

3.1.2 Main Menu scene

Αφού περάσουν τα 3 δευτερόλεπτα της Invoke μεθόδου το παιχνίδι θα μας μεταφέρει στην επόμενη σκηνή (MainMenuScene) όπου θα συναντήσουμε το παρακάτω περιβάλλον.

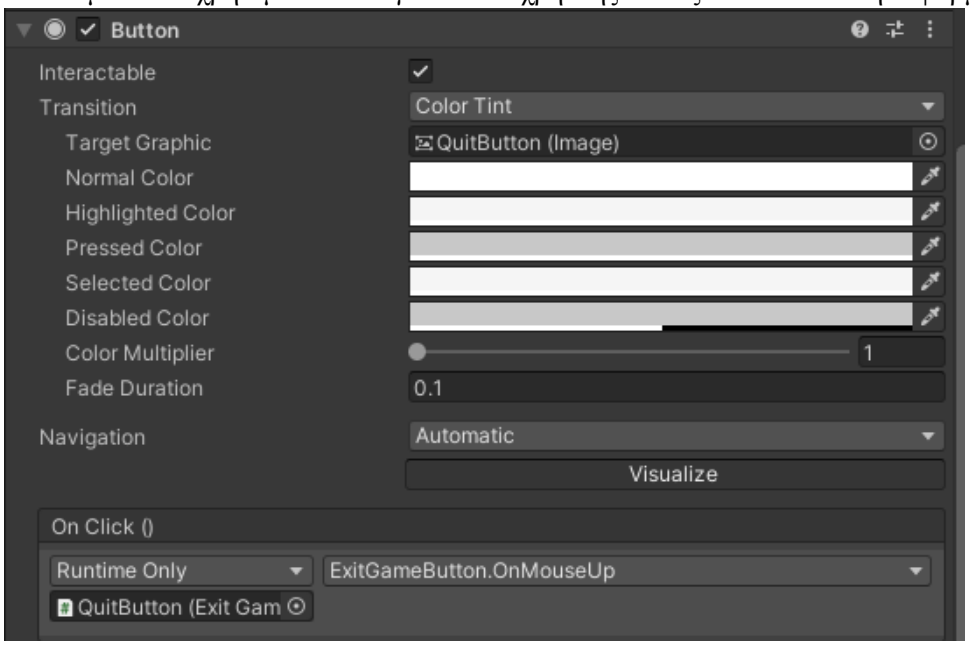


Σχήμα 3.4: Κεντρικό μενού

Πάνω σε αυτήν την σκηνή (Σχήμα 3.4) βλέπουμε μία εικόνα στην οποία έχουμε 5 Game Object .
Οι 4 κάρτες στην μέση και το κουμπί "Quit" κάτω δεξιά. Αναλυτικά :

- Quit Button :

Το κουμπί αυτό χρησιμοποιείται για όταν ο χρήστης επιλέξει να κλείσει την εφαρμογή.



Σχήμα 3.5: Inspector Quit Button

Στην παραπάνω εικόνα (Σχήμα 3.5) βλέπουμε ότι έχουμε δηλώσει το ‘QuitButton’ Game Object σαν UI αντικείμενο ‘Button’ στο οποίο του έχουμε δώσει ενα script “QuitButton” να λειτουργεί με την επιλογή του κουμπιού (OnClick μέθοδος).

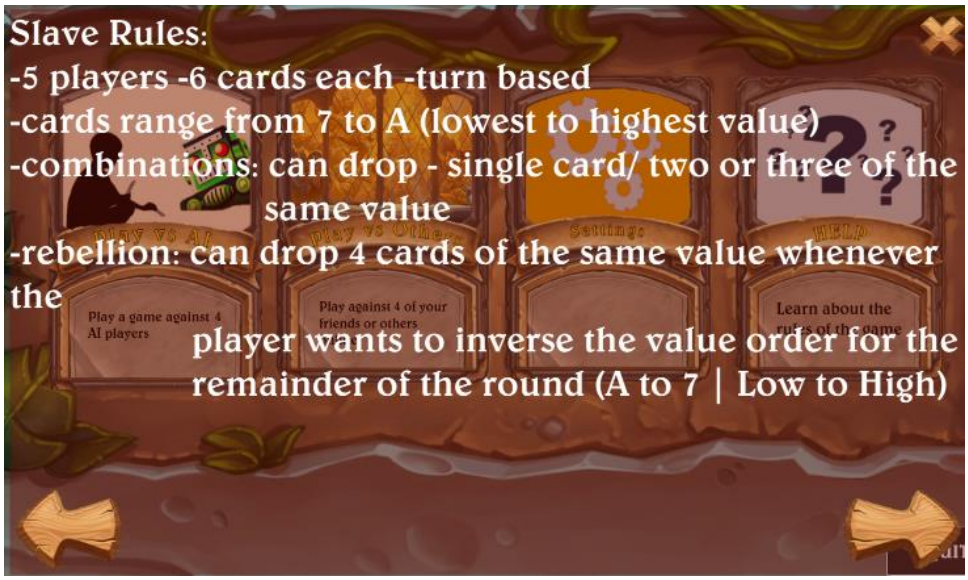
```
0 references | - changes | -authors, -changes
public void OnMouseUp()
{
    Application.Quit();
}
```

Σχήμα 3.6: Quit script

Με την OnClick μέθοδο μπορούν να τρέξουν μερικά event τα οποία τα δέχεται η Unity Engine. Αυτό που χρησιμοποιούμε εδώ είναι το OnMouseUp() (Σχήμα 3.6), αφού ο χρήστης κλικάρει το κουμπί ‘Quit’ και αφήσει το ποντίκι τότε τρέχει η μέθοδος Application.Quit() με την οποία κλείνει η εφαρμογή.

- Rules Game Object :

Με αυτό το Game Object στην επιλογή του ανοίγουμε ένα panel (Σχήμα 3.7), στο οποίο εξηγούμε τους κανόνες του παιχνιδιού στον παίκτη.



Σχήμα 3.7: Μετά την επιλογή του 'Help'

```
0 references | 0 changes | 0 authors, 0 changes
public void OnMouseOver()
{

    gameObject.GetComponent<Image>().sprite = focused;
    hover.Play();
}

0 references | 0 changes | 0 authors, 0 changes
public void OnMouseExit()
{
    gameObject.GetComponent<Image>().sprite = normal;
}

0 references | 0 changes | 0 authors, 0 changes
public void OnMouseDown()
{
    gameObject.GetComponent<Image>().sprite = normal;
    click.Play();
    rulesPanel.SetActive(true);


}

1 reference | 0 changes | 0 authors, 0 changes
public void ClosePanel()
{
    rulesPanel.SetActive(false);
    page2.enabled = false;
    page1.enabled = true;
}

1 reference | 0 changes | 0 authors, 0 changes
public void NextPage()
{
    page1.enabled = false;
    page2.enabled = true;
    previousPageButton.enabled = true;


}
1 reference | 0 changes | 0 authors, 0 changes
public void PreviousPage()
{
    page2.enabled = false;
    page1.enabled = true;
}
```

Σχήμα 3.8: Κώδικας 'Help'

Στον κώδικα βλέπουμε ότι χρησιμοποιούμε διαφορετικά Sprite ανάλογα με το state του ποντικιού μας (Σχήμα 3.8).(OnMouseOver() - OnMouseExit() - OnMouseDown())

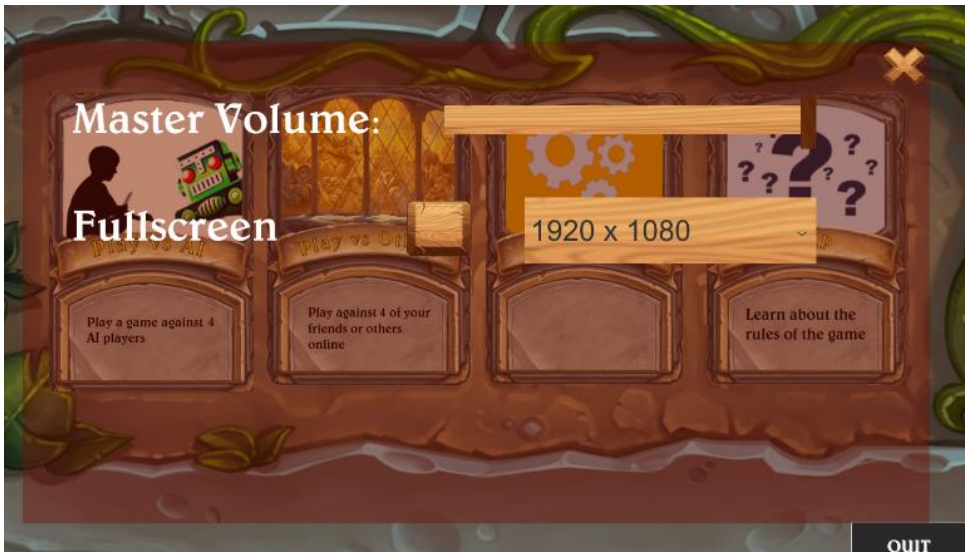
Στο event OnMouseOver επίσης έχουμε προσθέσει το hover.Play(); Το πεδίο hover ειναι Public AudioSource, ένα Game Object στο οποίο έχουμε βάλει ένα ηχητικό κλιπ το οποίο αναπαράγεται κάθε φορά που τριγγάρουμε το event.

Παρακάτω βλέπουμε τις μεθόδους ClosePanel, NextPage , PreviousPage.

- Close Panel: Με το 'X' κουμπί πάνω στο panel τρέχει η συγκεκριμένη μέθοδος όπου αλλάζουμε την τιμή του GameObject rulesPanel σε false, ώστε να μην είναι ορατό.

- Next/Previous Page: με τα 'βελάκια' στο panel μπορούμε να μεταφερθούμε στην επόμενη ή προηγούμενη σελίδα.

- Settings Game Object: Ανάλογα κάνουμε ορατό ένα διαφορετικό panel (SettingsPanel GameObject)



Σχήμα 3.9: Μετά την επιλογή του 'Settings'

Σε αυτό το panel (Σχήμα 3.9) μπορούμε να επεξεργαστούμε την ένταση οποιουδήποτε ήχου αναπαράγεται στο παιχνίδι, καθώς επίσης να ανοίξουμε το παιχνίδι σε fullscreen, ή σε μικρότερο παράθυρο και να αλλάξουμε την ανάλυση ανάλογα με την οθόνη του υπολογιστή μας.

```
void Start()
{
    Button btn = buttonExit.GetComponent<Button>();
    btn.onClick.AddListener(ClosePanel);

    resolutions = Screen.resolutions;
    resolutionDropdown.ClearOptions();

    List<string> options = new List<string>();

    int defaultWidth = 1920;  // Set your desired default width
    int defaultHeight = 1080; // Set your desired default height
    int curResIndex = 0;

    for (int i = 0; i < resolutions.Length; i++)
    {
        string option = resolutions[i].width + " x " + resolutions[i].height;
        options.Add(option);

        if (resolutions[i].width == defaultWidth && resolutions[i].height == defaultHeight)
        {
            curResIndex = i;
        }
    }

    resolutionDropdown.AddOptions(options);
    resolutionDropdown.value = curResIndex;
    resolutionDropdown.RefreshShownValue();
}
```

```
public void SetVolume(float volume)
{
    audioMixer.SetFloat("volume",volume);
}

public void SetFullScreen(bool isFullscreen)
{
    Screen.fullScreen = isFullscreen;
}

public void SetResolution( int resolutionIndex)
{
    Resolution resolution = resolutions[resolutionIndex];
    Screen.SetResolution(resolution.width, resolution.height,Screen.fullScreen);
}
```

Σχήμα 3.10: Κώδικας Settings Script

Εδώ (Σχήμα 3.10) βλέπουμε ότι με την εκκίνηση του ‘SettingsScript’ αρχικοποιούμε την βασική ανάλυση της εφαρμογής και βάζουμε στο resolutionDropdown GameObject άλλες αναλύσεις που μπορεί να υποστηρίξει ο υπολογιστής που τρέχει την εφαρμογή.

Στην συνέχεια του κώδικα έχουμε τις μεθόδους που μπορούμε να επιλέξουμε τις τιμές.

- PlayVsOthers Game Object: Σε αυτήν την κάρτα θα υποστηριχτεί στο μέλλον η επιλογή ο χρήστης να μπορεί να παίξει με άλλα άτομα στο ίντερνετ.

- PlayVsAi GameObject: Σε αυτό το panel ο χρήστης μπορεί να επιλέξει τον χαρακτήρα με τον οποίο θα παίξει(είναι διακοσμητικό και δεν έχει επίδραση στο παιχνίδι), ο χάρτης(ταμπλό) στον οποίο θα γίνει το παιχνίδι(για την ώρα είναι ένας), αλλά και το τελικό σκορ που χρειάζεται κάποιος για να νικήσει. Στην συνέχεια αφού κάνει αυτές τις επιλογές, το κουμπί ‘Play’ τον μεταφέρει στο επόμενο Scene για να ξεκινήσει το παιχνίδι (Σχήμα 3.11).

Σχήμα 3.11: Μετά την επιλογή του 'PlayVsAi'

```
ferences | 0 changes | 0 authors, 0 changes
lic class MenuVsAIScript : MonoBehaviour


  public Sprite focused;
  public Sprite normal;
  public AudioSource click;
  public AudioSource hover;
  public GameObject panel;
  public ToggleGroup toggleGroupPlayer,toggleGroupPts;
  public Toggle IndTog, AmeTog, RobTog, RusTog, SamTog, ZomTog, ZulTog;
  public GameObject IndPic, AmePic, RobPic, RusPic, SamPic, ZomPic, ZulPic;
  public Sprite IndFoc, AmeFoc, RobFoc, RusFoc, SamFoc, ZomFoc, ZulFoc;
  public Sprite IndNor, AmeNor, RobNor, RusNor, SamNor, ZomNor, ZulNor;

  public Toggle pts50Tog, pts100Tog, pts150Tog, pts200Tog;
  public Sprite ptsBkg, ptsNor;

  public Button btnExit;

  public static string player1, map1, pts1;
```

Σχήμα 3.12: Κώδικας 'MenuVsAI' script

Εδώ γίνεται η αρχικοποίηση των μεταβλητών μέσω τον οποίων με το Unity Editor θα αντιστοιχίσουμε τις εικόνες ή μεταβλητές στα ανάλογα Game Object (Σχήμα 3.12).

```
0 references | 0 changes | 0 authors, 0 changes
public void PlayGame()
{

    player1 = toggleGroupPlayer.ActiveToggles().LastOrDefault().name;
    pts1 = toggleGroupPts.ActiveToggles().LastOrDefault().name;
    SceneManager.LoadScene("SamuraiMap");
}
```

Σχήμα 3.13: Κώδικας 'MenuVsAI'

Τα GameObject αυτά βρίσκονται σε toggle groups, και ανάλογα με την τελευταία μας επιλογή όταν πατήσουμε το κουμπί 'Play' κρατάμε σε μεταβλήτες τα στοιχεία που επέλεξε ο χρήστης. Στην συνέχεια, φορτώνουμε την επόμενη σκηνή (SamuraiMap scene) η οποία έχει το παιχνίδι μας (Σχήμα 3.13).

3.1.3 Δημιουργία του board

Σε αυτό το σημείο έχουμε φορτώσει το βασικό κομμάτι του παιχνιδιού μας, την Main Game Scene. Στην οθόνη μας εμφανίζεται το board του παιχνιδιού, ο παίκτης που δηλώσαμε στην θέση μας, τυχαίοι παίκτες στις άλλες θέσεις και μετά από ένα mini animation μέσω της Unity αρχίζουν να εμφανίζονται οι κάρτες πάνω στο board.

Σχήμα 3.14: Το board του παιχνιδιού

Στην επιφάνεια του board (Σχήμα 3.14) ξεχωρίζουμε μερικά άσπρα τετραγωνάκια (μέσω του Scene View της Unity) τα οποία είναι οι θέσεις στις οποίες έχουμε επιλέξει εμείς να εμφανιστούν οι κάρτες καθώς μοιράζονται στους παίκτες ή και να παιχτούν μετά.

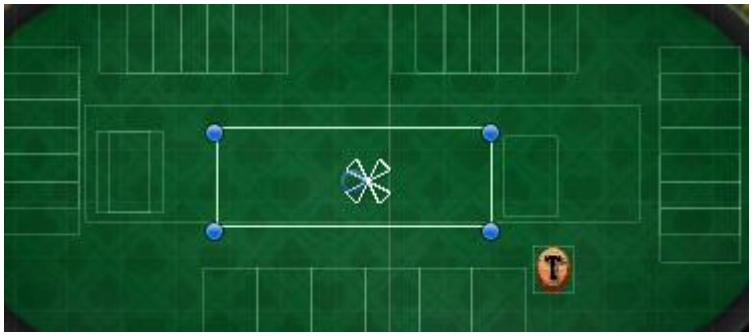
- Στη μέση του board έχουμε το deckArea (Σχήμα 3.15) GameObject στο οποίο δημιουργούμε τις κάρτες του παιχνιδιού στην αρχή σε τυχαία σειρά

Σχήμα 3.15: Board Game Objects

- Ο κάθε παίκτης μπροστά του έχει 5 GameObjects για την κάθε κάρτα(Σχήμα 3.16)

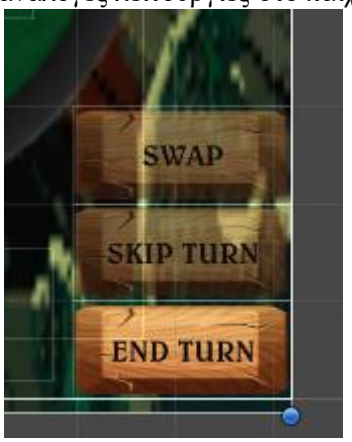
Σχήμα 3.16: Card Area Objects

- Στην μέση του board βρίσκεται το playDeckArea GameObject στο οποίο θα πηγαίνουν οι κάρτες που έχουν παιχτεί.



Σχήμα 3.17: Play Area Game Object

- Το 'T' είναι ενα GameObject (Σχήμα 3.17) μπροστά από κάθε παίκτη, το οποίο λειτουργεί σαν indicator με σκοπό την καλύτερη κατανόηση του ποιος παίκτης έχει σειρά παιχνιδιού.
- Τα κουμπιά 'Swap' 'Skip Turn' 'End Turn' στα δεξιά με τα οποία ο χρήστης κάνει τις ανάλογες λειτουργίες στο παιχνίδι (Σχήμα 3.18).



Σχήμα 3.18: Playable κουμπιά

- Τα κουμπιά του 'scoreboard' 'settings' στην πάνω δεξιά θέση για να ανατρέξει ο χρήστης στο σκορ του παιχνιδιού ή να αλλάξει κάποια ρυθμιση ήχου αντίστοιχα (Σχήμα 3.19).

Σχήμα 3.19: Settings, 'Ρυθμίσεις' Κουμπιά

3.1.4 Animation εκκίνησης παιχνιδιού

Σχήμα 3.20: Animation στην αρχή του παιχνιδιού

Για τα οπτικά δημιουργίας του deck των καρτών με των οποίων θα παίξουμε, δημιουργήσαμε ένα animation (Σχήμα 3.20).

```
public Transform panda;
public float speed;
public GameObject target;
private SamuraiMapScript samScript;
public float xOffset = -300;

[SerializeField] public Animator myAnimCon;

0 references | 0 changes | 0 authors, 0 changes
void Start()
{

    samScript = FindObjectOfType<SamuraiMapScript>();
    target = samScript.deckArea;
    Invoke("PandaAnimation", 1.5f);
    Invoke("DestroyObject", 2.5f);
```

Σχήμα 3.21: Κώδικας 'pandaAnimation'

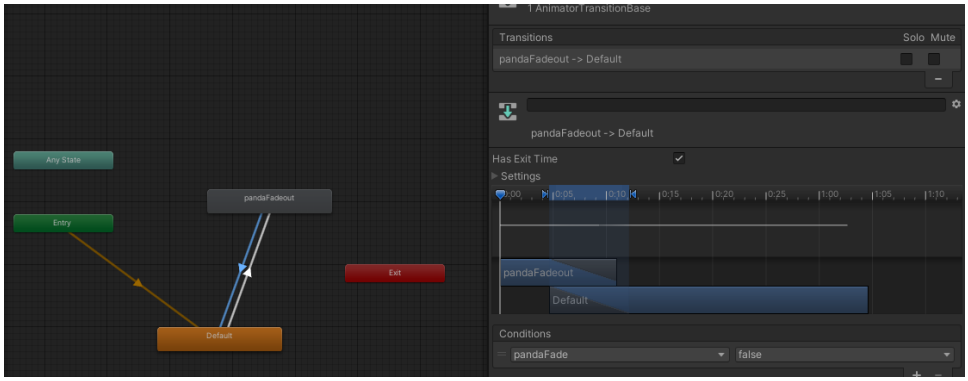
Με την εκκίνηση του Game Scene τρέχει το 'pandaAnimationStart' script (Σχήμα 3.21) στο οποίο αρχικοποιούμε το Sprite, ταχύτητα με την οποία θα κινείται και τα σημεία εκκίνησης και έναρξης του animation, καθώς και με την Invoke μέθοδο λέμε ότι θα ξεκινήσει 1.5 δευτερόλεπτο μετά την δημιουργία του scene, και το εν λόγω Game Object θα καταστραφεί 2.5 δευτερόλεπτα μετά.

```
0 references | 0 changes | 0 authors, 0 changes
void Update()
{

    transform.position = Vector2.MoveTowards(transform.position, new Vector3(target.transform.position.x + xOffset,target.transform.position.y,0), speed * Time.deltaTime);


}


0 references | 0 changes | 0 authors, 0 changes
void PandaAnimation()
{
    myAnimCon.SetBool("pandaFade", true);
}

0 references | 0 changes | 0 authors, 0 changes
void DestroyObject()
{
    Destroy(panda.gameObject);
}
```

Σχήμα 3.22: Κώδικας 'pandaAnimation'

Με την Vector2.MoveTowards μέθοδο (Σχήμα 3.22) μπορούμε να δηλώσουμε την θέση που έχει το αντικείμενο μας, την θέση στην οποία θέλουμε να κινηθεί, αλλά και την ταχύτητα με την οποία θα πάει (αυτό γίνεται μέσω του Time.deltaTime).

Για το animation χρησιμοποιούμε το Animator Tab που βρίσκουμε δίπλα από το game scene στον editor αφού επιλέξουμε το Game Object στο οποίο θέλουμε να δημιουργήσουμε το animation. Στην συγκεκριμένη περίπτωση το Game Object με το Panda Sprite.



Σχήμα 3.23: Animator Tab

Με το animator (Σχήμα 3.23) μπορούμε να επεξεργαστούμε τα χρονικά σημεία στα οποία θέλουμε να γίνουν αλλαγές στο animation μας και να αλλάξουμε συγκεκριμένες μεταβλητές όπως την 'pandaFade' η οποία σβήνει το Sprite έως ότου εξαφανιστεί τελείως και καταστραφεί το Game Object.

3.1.5 Map Script

Σε αυτό το κομμάτι θα αναλύσουμε το 'SamuraiMapScript' στο οποίο ξεκινάμε το animation, παίρνουμε τα δεδομένα που μας ήρθανε από το menu, αρχικοποιούμε τις θέσεις στις οποίες θα εμφανίσουμε τις κάρτες αλλά και τα animation με τα οποία θα τις μοιράσουμε.

```
//GameObjects
public GameObject C7, C8, C9, C10, CJ, CQ, CK, CA;
public GameObject D7, D8, D9, D10, DJ, DQ, DK, DA;
public GameObject H7, H8, H9, H10, HJ, HQ, HK, HA;
public GameObject S7, S8, S9, S10, SJ, SQ, SK, SA;

public GameObject myAreaPos1;
public GameObject myAreaPos2;
public GameObject myAreaPos3;
public GameObject myAreaPos4;
public GameObject myAreaPos5;
public GameObject myAreaPos6;

public GameObject enemyAreaPos1;
public GameObject enemyAreaPos2;
public GameObject enemyAreaPos3;
public GameObject enemyAreaPos4;
public GameObject enemyAreaPos5;
public GameObject enemyAreaPos6;

public GameObject enemy2AreaPos1;
public GameObject enemy2AreaPos2;
public GameObject enemy2AreaPos3;
public GameObject enemy2AreaPos4;
public GameObject enemy2AreaPos5;
public GameObject enemy2AreaPos6;

public GameObject enemy3AreaPos1;
public GameObject enemy3AreaPos2;
public GameObject enemy3AreaPos3;
public GameObject enemy3AreaPos4;
public GameObject enemy3AreaPos5;
public GameObject enemy3AreaPos6;

public GameObject enemy4AreaPos1;
public GameObject enemy4AreaPos2;
public GameObject enemy4AreaPos3;
public GameObject enemy4AreaPos4;
public GameObject enemy4AreaPos5;
public GameObject enemy4AreaPos6;

public GameObject extraCardPos1;
public GameObject extraCardPos2;

public GameObject playDeckArea;
public GameObject playerAvatar, enemy1Avatar, enemy2Avatar, enemy3Avatar, enemy4Avatar;
public GameObject cardBack;
public GameObject deckArea;
public GameObject canvas;

//Sprites
public Sprite faceC7, faceC8, faceC9, faceC10, faceCJ, faceCQ, faceCK, faceCA;
public Sprite faceD7, faceD8, faceD9, faceD10, faceDJ, faceDQ, faceDK, faceDA;
public Sprite faceH7, faceH8, faceH9, faceH10, faceHJ, faceHQ, faceHK, faceHA;
public Sprite faceS7, faceS8, faceS9, faceS10, faceSJ, faceSQ, faceSK, faceSA;

public Sprite indHero, ameHero, robHero, rusHero, samHero, zomHero, zulHero;

private SpriteRenderer spriteRenderer;
public Sprite[] cardFaces;
public Sprite cardFace;
```

Σχήμα 3.24: Αρχικοποίηση των Game Object

```
//Lists
public List<GameObject> myAreaPos;
public List<GameObject> enemyAreaPos;
public List<GameObject> enemy2AreaPos;
public List<GameObject> enemy3AreaPos;
public List<GameObject> enemy4AreaPos;

public List<GameObject> extraCardPos;
public List<GameObject> extraCard2Pos;
public List<GameObject> playDeckAreaPos;

public List<GameObject> heroesAvatars;
public List<Sprite> heroesSprites;

public List<Animator> myAreaPosCon;
public List<Animator> enemyAreaPosCon;
public List<Animator> enemy2AreaPosCon;
public List<Animator> enemy3AreaPosCon;
public List<Animator> enemy4AreaPosCon;
public List<Animator> extraCardPosCon;

public List<GameObject> deck = new List<GameObject>();
public List<Sprite> deckFaces = new List<Sprite>();

//Animations
[SerializeField] private Animator myArea1Con;
[SerializeField] private Animator myArea1Con2;
[SerializeField] private Animator myArea1Con3;
[SerializeField] private Animator myArea1Con4;
[SerializeField] private Animator myArea1Con5;
[SerializeField] private Animator myArea1Con6;

[SerializeField] private Animator enemy1Con;
[SerializeField] private Animator enemy1Con2;
[SerializeField] private Animator enemy1Con3;
[SerializeField] private Animator enemy1Con4;
[SerializeField] private Animator enemy1Con5;
[SerializeField] private Animator enemy1Con6;

[SerializeField] private Animator enemy2Con;
[SerializeField] private Animator enemy2Con2;
[SerializeField] private Animator enemy2Con3;
[SerializeField] private Animator enemy2Con4;
[SerializeField] private Animator enemy2Con5;
[SerializeField] private Animator enemy2Con6;

[SerializeField] private Animator enemy3Con;
[SerializeField] private Animator enemy3Con2;
[SerializeField] private Animator enemy3Con3;
[SerializeField] private Animator enemy3Con4;
[SerializeField] private Animator enemy3Con5;
[SerializeField] private Animator enemy3Con6;

[SerializeField] private Animator enemy4Con;
[SerializeField] private Animator enemy4Con2;
[SerializeField] private Animator enemy4Con3;
[SerializeField] private Animator enemy4Con4;
[SerializeField] private Animator enemy4Con5;
[SerializeField] private Animator enemy4Con6;

[SerializeField] private Animator extraCardCon;
[SerializeField] private Animator extraCardCon2;
```

Σχήμα 3.25: Βασικές μεταβλητές

Στις παραπάνω εικόνες δείχνουμε τις μεταβλητές που φτιάξαμε ώστε να ταιριάξουμε τα ανάλογα GameObject μέσω του Unity Editor, στα οποία περιλαμβάνουμε, τις κάρτες, θέσεις παικτών, εικόνες αλλά και σε Animator fields τις θέσεις ώστε να χρησιμοποιήσουμε τα animation που κατασκευάσαμε για την δημιουργία των καρτών (Σχήμα 3.24).

```
heroesSprites = new List<Sprite>(new Sprite[] { indHero, ameHero, robHero, rusHero, samHero, zomHero, zulHero });


//generation of the deck and faces
deck = generateDeck();
deckFaces = GenerateSprites();
//shuffling the generated deck
Shuffle(deck, deckFaces);
//putting it on the board
gameDeal(deck);
//starting animations for player's hands, and then instantiating the cards of the deck
SetPlayers();
StartCoroutine(AnimDeal(2.5f));
StartCoroutine(HandDeal(deck,deckFaces, 3.5f));
StartCoroutine(AnimDealStop(4.5f));
StartCoroutine(Anim1Deal(3.5f));
StartCoroutine(Hand1Deal(deck, deckFaces, 4.5f));
StartCoroutine(Anim1DealStop(5.5f));
StartCoroutine(Anim2Deal(4.5f));
StartCoroutine(Hand2Deal(deck, deckFaces, 5.5f));
StartCoroutine(Anim2DealStop(6.5f));
StartCoroutine(Anim3Deal(5.5f));
StartCoroutine(Hand3Deal(deck, deckFaces, 6.5f));
StartCoroutine(Anim3DealStop(7.5f));
StartCoroutine(Anim4Deal(6.5f));
StartCoroutine(Hand4Deal(deck, deckFaces, 7.5f));
StartCoroutine(Anim4DealStop(8.5f));
StartCoroutine(AnimExtraDeal(7.5f));
StartCoroutine(LastCardsDeal(deck, deckFaces, 8.5f));
StartCoroutine(AnimExtraDealStop(9.5f));
```

Σχήμα 3.26: Start method 'Game Script'

Στην start() μέθοδο του script ξεκινάμε με την μέθοδο generateDeck() με την οποία ‘ταίζουμε στην μεταβλητή λίστα ‘deck’ την τράπουλα (Σχήμα 3.25).

```
2 references | 0 changes | 0 authors, 0 changes
public List<Sprite> GenerateSprites()
{
    List<Sprite> newFaces = new List<Sprite>();

    newFaces.Add(faceC7);
    newFaces.Add(faceC8);
    newFaces.Add(faceC9);
    newFaces.Add(faceC10);
    newFaces.Add(faceCJ);
    newFaces.Add(faceCQ);
    newFaces.Add(faceCK);
    newFaces.Add(faceCA);
    newFaces.Add(faceD7);
    newFaces.Add(faceD8);
    newFaces.Add(faceD9);
    newFaces.Add(faceD10);
    newFaces.Add(faceDJ);
    newFaces.Add(faceDQ);
    newFaces.Add(faceDK);
    newFaces.Add(faceDA);
    newFaces.Add(faceH7);
    newFaces.Add(faceH8);
    newFaces.Add(faceH9);
    newFaces.Add(faceH10);
    newFaces.Add(faceHJ);
    newFaces.Add(faceHQ);
    newFaces.Add(faceHK);
    newFaces.Add(faceHA);
    newFaces.Add(faceS7);
    newFaces.Add(faceS8);
    newFaces.Add(faceS9);
    newFaces.Add(faceS10);
    newFaces.Add(faceSJ);
    newFaces.Add(faceSQ);
    newFaces.Add(faceSK);
    newFaces.Add(faceSA);
    return newFaces;
}
```

Σχήμα 3.27: ‘GenerateSprites’ μέθοδος

Με τον ίδιο τρόπο αμέσως μετά αντιστοιχούμε τα σωστά Sprites στις σωστές κάρτες (Σχήμα 3.26).

```
2 references | 0 changes | 0 authors, 0 changes
public void Shuffle<T>(List<T> list, List<Sprite> cardFronts)     //shuffling cards based on the Fisher-Yates Shuffle
{
    System.Random random = new System.Random();
    int n = list.Count;
    while (n > 1)
    {
        int k = random.Next(n);
        n--;
        T temp = list[k];         //shuffle cards
        list[k] = list[n];
        list[n] = temp;
        Sprite tmp = cardFronts[n];       //shuffle sprite list order according to the shuffle of the deck
        cardFronts[n] = cardFronts[k];
        cardFronts[k] = tmp;
    }
}
```

Σχήμα 3.28: 'Shuffle' μέθοδος

Στην συνέχεια, με την μέθοδο Shuffle (Σχήμα 3.27), την οποία δίνουμε ως παραμέτρους, τις κάρτες και τα Sprites τους, με τον αλγόριθμο fisher-yates αναμειγνύουμε τις κάρτες, δίνοντας τες παράλληλα το σωστό εικονίδιο τους.

```
2 references | 0 changes | 0 authors, 0 changes
public void gameDeal(List<GameObject> list) //instantiate cards on the board, and deactivate for animation purpose
{


    int i = 0;

    foreach (GameObject card in list)
    {
        //dealing the deck on the table
        GameObject newCard = Instantiate(list[i], new Vector3(0, 0, 0), Quaternion.identity);
        //hiding the cards after animation
        newCard.SetActive(false);
        newCard.transform.SetParent(deckArea.transform, false);
        newCard.name = card.name;

        i++;


    }


}
```

Σχήμα 3.29: 'GameDeal' μέθοδος

Αφού τελειώσει αυτή η διαδικασία, με την μέθοδο Instantiate στο σημείο που έχουμε δηλώσει δημιουργούμε την τράπουλα και την βάζουμε πάνω στο board μας (Σχήμα 3.28).

Τέλος, με StartCoroutine() κάνουμε πρώτα ένα animation μπροστά από κάθε παίκτη για να εμφανίσουμε τις κάρτες και στην επόμενη μέθοδο τις στέλνουμε.

```
2 references | 0 changes | 0 authors, 0 changes
public IEnumerator AnimDeal(float delayTime)
{
    yield return new WaitForSeconds(delayTime);

    for (var i = 0; i < 6; i++)
    {
        //playing the animation for the hand before it is dealt
        myAreaPosCon[i].SetBool("deckAnimStart", true);
        audioSlice.Play();
    }
}
```

Σχήμα 3.30: 'AnimDeal' μέθοδος

Ακολουθάμε τον ίδιο τρόπο με το προηγούμενο animation, τώρα όμως προσθέτοντας ένα audioClip για την εμπειρία του χρήστη (Σχήμα 3.29).

```
2 references | 0 changes | 0 authors, 0 changes
public IEnumerator HandDeal(List<GameObject> list, List<Sprite> cardFronts, float delayTime)
{
    yield return new WaitForSeconds(delayTime);

    for (var i = 0; i < 6; i++) //myplayer
    {

        //spawning last card of deck on each area position on player's hand
        GameObject lastCard = list.Last();
        GameObject theLast = Instantiate(lastCard, new Vector3(0, 0, 0), Quaternion.identity);
        theLast.transform.SetParent(myAreaPos[i].transform, false);
        theLast.name = lastCard.name;
        //changing card sprite to the face of the card from the cardFaces List
        theLast.GetComponent<Image>().sprite = cardFronts[list.Count - 1];
        //assigning card sprite to the face of the card from the cardFaces List
        theLast.GetComponent<SpriteRenderer>().sprite = cardFronts[list.Count - 1];
        //set card as Hand Tag
        theLast.tag = "myHand" ;
        //removing the card from the top of the deck
        list.RemoveAt(list.Count - 1);


    }


}
```

Σχήμα 3.31: 'HandDeal' μέθοδος

Με την μέθοδο HandDeal (Σχήμα 3.30), παίρνουμε τις τελευταίες 6 κάρτες της αρχικής τράπουλας που φτιάξαμε και τις βάζουμε μια-μια στις θέσεις του παίκτη ενώ παράλληλα τις αφαιρούμε από την αρχική τράπουλα.
Η ίδια διαδικασία animation-dealing ακολουθείται και για τους υπόλοιπους παίκτες.

3.1.6 Game Handler Script

Σε αυτό το σημείο έχουμε μοιράσει τις κάρτες σε όλους τους παίκτες και είμαστε έτοιμοι να παίξουμε το παιχνίδι μας. Πριν προχωρήσουμε όμως εκεί πρέπει να αναλύσουμε τον κώδικα που υπάρχει στο 'GameHandler' script.

Αρχικά έχουμε την κλάση 'Character' (Σχήμα 3.31) ώστε να μπορούμε προγραμματιστικά να κρατάμε τα στοιχεία κάθε παίχτη καθ'όλη την διάρκεια του παιχνιδιού.

```
26 references | 0 changes | 0 authors, 0 changes
public class Character
{
    public string playerName;
    public List<Animator> playerPos;
    public GameObject[] playerHand;
    public string playerHandTag;
    public int roundPoints;
    public int totalPoints;
    public GameState playerTurn;
    public GameObject playerTurnIndicator;
    public string lastRoundPosition;
    public bool playerFinishedRound;

    5 references | 0 changes | 0 authors, 0 changes
    public Character(string name, List<Animator> playerP
    {
        playerName = name;
        playerPos = playerPosCon;
        playerHand = hand;
        playerHandTag = handTag;
        roundPoints = round;
        totalPoints = total;
        playerTurn = turn;
        playerTurnIndicator = turnIndicator;
        lastRoundPosition = lastRoundPos;
        playerFinishedRound = playerFinRound;
    }
}
```

Σχήμα 3.32: 'Character' class

Τα πεδία που περιέχει το αντικείμενο 'Character' είναι τα εξής:

- PlayerName : το όνομα του παίκτη

- PlayerPos: animator που χρησιμοποιείται για το animation στην αρχή του παιχνιδιού
- PlayerHand[]: λίστα που περιέχει τις κάρτες του παίκτη
- PlayerHandTag: Δίνουμε το αντίστοιχο tag στις κάρτες των παικτών για χρήση τους
- RoundPoints: οι πόντοι που κέρδισε τον τελευταίο γύρο ο παίκτης
- TotalPoints: οι συνολικοί πόντοι που έχει ο παίκτης στο παιχνίδι
- PlayerTurn: η κατάσταση παιχνιδιού που έχει ο παίκτης
- PlayerTurnIndicator: Game Object που βοηθάει στο να υποδείξει ποιανού παίκτη είναι η σειρά να παίξει την σειράτου
- LastRoundPosition: κρατάει την θέση που κατέκτησε στον τελευταίο γύρο ο παίκτης
- PlayerFinishedRound: boolean μεταβλητή που ελέγχει αν τελείωσε ο παίκτης τον γύρο

```
public class GameHandler : MonoBehaviour
{
    public Character player = new Character("player", null, null, null, 0, 0, GameState.FirstTurn, null, null, false);  // pass a variable from login screen
    public Character enemy1 = new Character("enemy1", null, null, null, 0, 0, GameState.Enemy1Turn, null, null, false);
    public Character enemy2 = new Character("enemy2", null, null, null, 0, 0, GameState.Enemy2Turn, null, null, false);
    public Character enemy3 = new Character("enemy3", null, null, null, 0, 0, GameState.Enemy3Turn, null, null, false);
    public Character enemy4 = new Character("enemy4", null, null, null, 0, 0, GameState.Enemy4Turn, null, null, false);
    public GameState curState;
```

Σχήμα 3.33: Αρχικοποίηση 'Characters'

Με την παραπάνω κλάση στην αρχή του παιχνιδιού αρχικοποιούμε τους 5 χαρακτήρες που θα παίξουν το παιχνίδι (Σχήμα 3.32).

```
68 references | 0 changes | 0 authors, 0 changes
public enum GameState { FirstTurn, PlayerTurn, Enemy1Turn, Enemy2Turn, Enemy3Turn, Enemy4Turn, RoundEnding, RoundStarting, Taxing, PlayerTaxing, EndGame };
```

Σχήμα 3.34: Game State

Χρησιμοποιούμε τα παραπάνω GameState για να ξεκινήσουμε το παιχνίδι, να δώσουμε σειρά στους παίκτες, για να τελειώσουμε τον γύρο, να ξεκινήσουμε καινούργιο αλλά και να τελειώσουμε ένα παιχνίδι (Σχήμα 3.33).

Αυτές οι λειτουργίες γίνονται στην Update() μέθοδο του ‘GameHandler’ script ώστε να γίνεται συνεχής έλεγχος. Το παιχνίδι ξεκινάει σε ‘First Turn’ state, ώστε να ξεκινάει πάντα ο χρήστης μας πρώτος το παιχνίδι.

Ο χρήστης έχει 3 επιλογές για να προχωρήσει το παιχνίδι.

Το κουμπί ‘Swap’ , ‘Skip Turn’, ‘End Turn’.

- Swap: Το κουμπί αυτό χρησιμοποιείται στο ξεκίνημα κάθε επόμενου γύρου στην περίπτωση που ο χρήστης έχει βγει 3ος στον προηγούμενο γύρο, για να ανταλλάξει 2 από τις κάρτες που έχει στο χέρι του με τις δυο έξτρα στο board.(Taxing)

```
0 references | 0 changes | 0 authors, 0 changes
public void TaxingPlayerThird()
{

    Transform temp, temp2;
    Character starting = roundsTable[0];
    if (playingHand.Length == 2)
    {

        temp = playingHand[0].transform.parent.transform;
        playingHand[0].transform.position = extraAreaCardPos1.transform.position;
        playingHand[0].transform.SetParent(extraAreaCardPos1.transform, true);
        playingHand[0].tag = "extraCards";

        extraAreaCards[0].transform.position = temp.position;
        extraAreaCards[0].transform.SetParent(temp, true);
        extraAreaCards[0].GetComponent<Image>().sprite = extraAreaCards[0].GetComponent<SpriteRenderer>().sprite;
        extraAreaCards[0].tag = player.playerHandTag;

        temp2 = playingHand[1].transform.parent.transform;
        playingHand[1].transform.position = extraAreaCardPos2.transform.position;
        playingHand[1].transform.SetParent(extraAreaCardPos2.transform, true);
        playingHand[1].tag = "extraCards";

        extraAreaCards[1].transform.position = temp2.position;
        extraAreaCards[1].transform.SetParent(temp2, true);
        extraAreaCards[1].GetComponent<Image>().sprite = extraAreaCards[1].GetComponent<SpriteRenderer>().sprite;
        extraAreaCards[1].tag = player.playerHandTag;



    }
    else
    {
        turnText.text = "Pick 2 cards or skip";
        goto WrongFinish;
    }

    endTaxing = true;
    ResetNewRound();
    curState = starting.playerTurn;
    swapCardsButton.interactable = false;

WrongFinish:

    return;

} //Giving the player option to swap 2 of his chosen cards with the 2 extra
```

Σχήμα 3.35: 'TaxingPlayerThird' μέθοδος

Μέσα σε αυτήν την μέθοδο ελέγχουμε αν την ώρα που ο χρήστης πατάει το κουμπί έχει επιλέξει 2 κάρτες και αν είναι σωστό τότε αλλάζουμε θέση τις κάρτες αυτές με τις έξτρα του board και τις αναποδογυρίζουμε ώστε να μην φαίνονται στους άλλους παίκτες (Σχήμα 3.34).

- Skip Turn: Ο χρήστης μπορεί να μην έχει κάποια διαθέσιμη κάρτα να παίξει ή και λόγω στρατηγικής να μην θέλει να παίξει κάποια κάρτα.

```
0 references | 0 changes | 0 authors, 0 changes
public void SkipPlayerTurn()
{
    Character starting = roundsTable[0];


    if (curState == GameState.PlayerTaxing)
    {
        endTaxing = true;
        ResetNewRound();
        curState = starting.playerTurn;
        swapCardsButton.interactable = false;
    }
    else
    {
        hasHand = false;
        turnText.text = "it's your opponent's turn";
        curState = GameState.Enemy1Turn;
    }

}
```

Σχήμα 3.36: 'Skip Turn' μέθοδος

Παραπάνω δείχνουμε την μέθοδο του 'Skip Turn' όπου έχει 2 λειτουργίες (Σχήμα 3.35).

Πρώτα ελέγχουμε αν βρισκόμαστε στην κατάσταση του 'Taxing' όπου ο χρήστης μπορεί να επιλέξει να μην κάνει κάποια ενέργεια.

Στην άλλη περίπτωση όπου είναι η σειρά του παίκτη και δεν παίξει κάτι στέλνουμε το Game State στην σείρα του επόμενου παίκτη.

- End Turn: Με αυτό το κουμπί ο χρήστης επικυρώνει την επιλογή του και παίζει τις κάρτες που διάλεξε για να τελειώσει την σειρά του.

```
public void EndPlayerTurn() //checking how many cards are played, to confirm the playing of the hand
{

    if (curState == GameState.FirstTurn || playAreaCards.Count == sameHandCount || (enemy4Finished && !enemy1HasPlayed && !enemy2HasPlayed && !enemy3HasPlayed) || (enemy4Finished && enemy3Fi
    {
        hasHand = true;
        endTaxing = false;
    }

    if (hasHand)  // if player has Hand then he can play whatever he wants
    {

        if (playingHand.Length == 1) //if 1 card is played
        {
            playAreaCards.Add(playingHand[0]); //if the play is correct, add the playingCards to the playArea
            playingHand[0].transform.position = PushIntoPlayArea(); //change position of the chosen cards to the PlayArea
            playingHand[0].transform.SetParent(playArea.transform);
            playingHand[0].tag = "playArea"; //change tag of the card
            previousPlayerCards = 1;
        }
        else if (playingHand.Length == 2 && playingHand[0].GetComponent<CardHandler>().value == playingHand[1].GetComponent<CardHandler>().value) //if 2 cards are played and are of same value
        {
            for (int j = 0; j < playingHand.Length; j++)
            {
                playAreaCards.Add(playingHand[j]);
                playingHand[j].transform.position = PushIntoPlayArea();
                playingHand[j].transform.SetParent(playArea.transform);
                playingHand[j].tag = "playArea";
                previousPlayerCards = 2;

            }

        }
        else if (playingHand.Length == 3 && playingHand[0].GetComponent<CardHandler>().value == playingHand[1].GetComponent<CardHandler>().value && playingHand[0].GetComponent<CardHandler>()
        {
            for (int j = 0; j < playingHand.Length; j++)
            {
                playAreaCards.Add(playingHand[j]);
                playingHand[j].transform.position = PushIntoPlayArea();
                playingHand[j].transform.SetParent(playArea.transform);
                playingHand[j].tag = "playArea";
                previousPlayerCards = 3;
            }
        }
```

Σχήμα 3.37: 'End Turn' μέθοδος

Για την σωστή διαχείριση αυτής της κατάστασης πρέπει να κάνουμε κάποιους ελέγχους πριν περάσουμε στην επόμενη σειρά (Σχήμα 3.36).

Την ώρα που πατάμε το κουμπί πρέπει να τσεκάρουμε αν έχουμε ‘χέρι’ ή όχι για να ξέρουμε τι μπορεί να παιχτεί. Βλέπουμε αν είναι η πρώτη σειρά, αν κάποιος έχει παίξει άλλο φύλλο πάνω στο δικό μας ή ο προηγούμενος παίκτης έχει ‘βγει’ και δεν υπάρχει κάποιο καινούργιο φύλλο κάτω.

Στην περίπτωση που έχουμε ‘χέρι’ στην παραπάνω εικόνα δείχνουμε τις διαθέσιμες επιλογές του χρήστη.

Μετά την επιλογή ενός, δύο, τριών ή και 4 φύλλων αλλάζουμε το transform position των καρτών από το ‘χέρι’ του παίκτη στο Game Object ‘Play Area’ αλλάζοντας τους ταυτόχρονα και το tag. Επίσης, ανάλογα με την επιλογή αποθηκεύουμε και στην μεταβλητή ‘previousPlayerCards’ τον αριθμό που παίχτηκε ώστε να ξέρουμε τι μπορεί να παιχτεί αυτή την γύρα.

```
if (previousPlayerCards == 1)
{
    if (playingHand.Length == 1 && playingHand[0].GetComponent<CardHandler>().value > playAreaCards.Last().GetComponent<CardHandler>().value)
    {
        playAreaCards.Add(playingHand[0]); //if the play is correct, add the playingCards to the playArea
        playingHand[0].transform.position = PushIntoPlayArea(); //change position of the chosen cards to the PlayArea
        playingHand[0].transform.SetParent(playArea.transform);
        playingHand[0].tag = "playArea"; //change tag of the card
        previousPlayerCards = 1;
    }
    else
    {
        turnText.text = "Have to Play 1 Card\r\nof greater value";
        goto WrongFinish;
    }
}
else if (previousPlayerCards == 2)
{
    if (playingHand.Length == 2 && playingHand[0].GetComponent<CardHandler>().value == playingHand[1].GetComponent<CardHandler>().value) //if 2 cards are played and are of same valu
    {
        for (int j = 0; j < playingHand.Length; j++)
        {
            playAreaCards.Add(playingHand[j]);
            playingHand[j].transform.position = PushIntoPlayArea();
            playingHand[j].transform.SetParent(playArea.transform);
            playingHand[j].tag = "playArea";
            previousPlayerCards = 2;

        }
    }
    else
    {
        turnText.text = "Have to Play 2 Cards\r\nof greater value";
        goto WrongFinish;
    }
}
else if (previousPlayerCards == 3)
{
    if (playingHand.Length == 3 && playingHand[0].GetComponent<CardHandler>().value == playingHand[1].GetComponent<CardHandler>().value && playingHand[0].GetComponent<CardHandler>(
    {
        for (int j = 0; j < playingHand.Length; j++)
        {
            playAreaCards.Add(playingHand[j]);
            playingHand[j].transform.position = PushIntoPlayArea();
            playingHand[j].transform.SetParent(playArea.transform);
            playingHand[j].tag = "playArea";
```

Σχήμα 3.38: Έλεγχος παίξιμου φύλλων

Στην περίπτωση που η μεταβλητή ‘hasHand’ είναι false, που σημαίνει ότι έχει παίξει άλλος παίκτης πριν’, ακολουθούμε την ίδια διαδικασία αυτήν την φορά όμως πριν από κάθε επιλογή ελέγχοντας την μεταβλητή ‘previousPlayerCards’. Εάν ο χρήστης παίξει κάτι το οποίο δεν επιτρέπεται θα εμφανιστεί ανάλογο μήνυμα και θα ακυρώσει το παιχνίδι, καθώς και θα του επιδείξει πως πρέπει να παίξει σωστά (Σχήμα 3.37).

```
                }
            }
        }

        sameHandCount = playAreaCards.Count;
        curState = GameState.Enemy1Turn;
        hasHand = false;
        turnText.text = "it's your opponent's turn";

    WrongFinish:
        return;
    }
```

Σχήμα 3.39: Ετοιμασία για επόμενο state

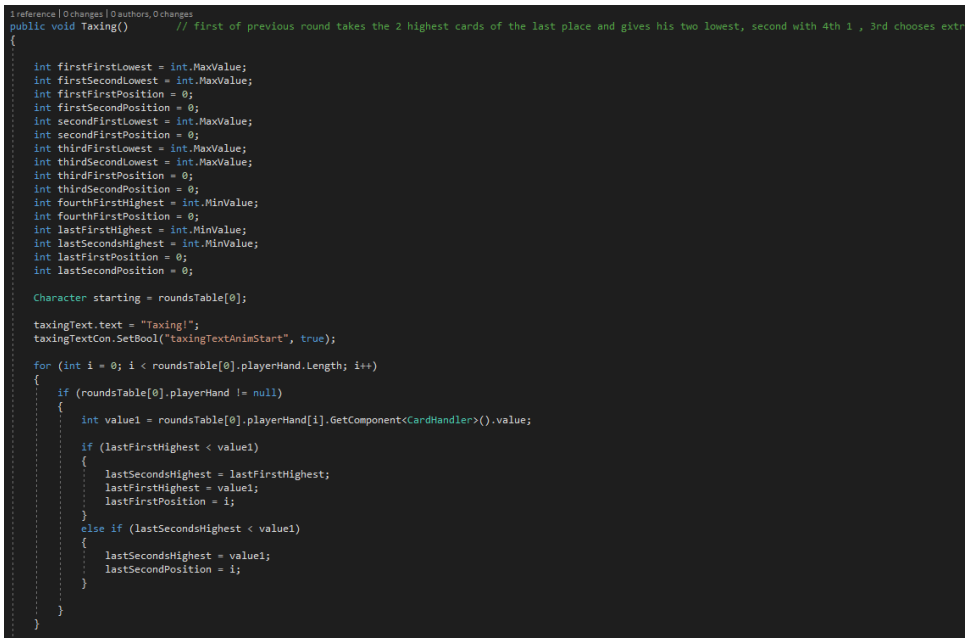
Στο τέλος της μεθόδου σεττάρουμε μερικές μεταβλητές (Σχήμα 3.38). Η ‘sameHandCount’ κρατάει τα φύλλα που παίχτηκαν και σε περίπτωση που τελειώσουν οι υπόλοιποι παίκτες την σειρά τους και είναι πάλι τα ίδια φύλλα, η μεταβλητή ‘hasHand’ θα αλλάξει σε true.
Στην συνέχεια δίνουμε την σειρά στον επόμενο παίκτη.

```
0 references | 0 changes | 0 authors, 0 changes
public void Enemy1Playing()
{

    if (curState == GameState.Enemy1Turn)
    {
        if (previousPlayerCards == 1)
        {
            int lowest = int.MaxValue;
            int position = 0;


            for (int i = 0; i < enemy1.playerHand.Length; i++)
            {


                if (enemy1.playerHand[i] != null) //check if null for error
                {
                    int value = enemy1.playerHand[i].GetComponent<CardHandler>().value;

                    if (value < lowest && value > playAreaCards.Last().GetComponent<CardHandler>().value)
                    {
                        lowest = value;
                        position = i;
                    }
                }


            }


            if (lowest < int.MaxValue) //if lowest card is greater than the last on deck play it
            {
                playAreaCards.Add(enemy1.playerHand[position]);
                enemy1.playerHand[position].transform.position = PushIntoPlayArea();
                enemy1.playerHand[position].transform.SetParent(playArea.transform);
                enemy1.playerHand[position].tag = "playArea";
                enemy1.playerHand[position].GetComponent<Image>().sprite = enemy1.playerHand[position].GetComponent<SpriteRenderer>().sprite;
                previousPlayerCards = 1;
                enemy1HasPlayed = true;

            }
            else
            {
                enemy1HasPlayed = false;
            }

        }
```

Σχήμα 3.40: Κώδικας παιχνιδιού AI

Για τις κινήσεις του υπολογιστή έχουμε δύο διαφορετικές μεθόδους, μία για την περίπτωση που ο παίκτης έχει σειρά και μία που δεν έχει (Σχήμα 3.39).

Με περίπου ανάλογο τρόπο όπως βλέπουμε στην παραπάνω εικόνα, ελέγχουμε τις προηγούμενες κάρτες που παίχτηκαν και από τις διαθέσιμες επιλογές ο υπολογιστής παίζει κάρτα/ες μεγαλύτερης αξίας.

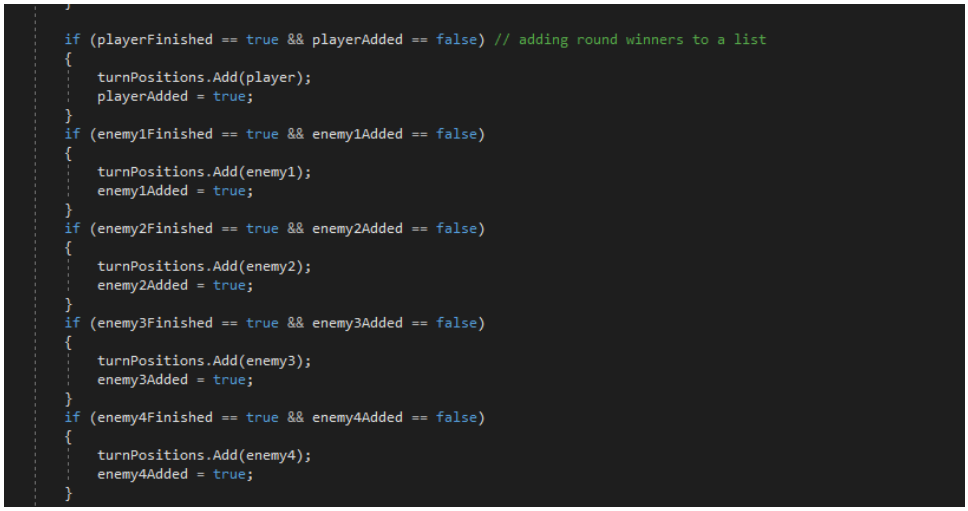


```
1 reference | 0 changes | 0 authors, 0 changes
public void Taxing()        // first of previous round takes the 2 highest cards of the last place and gives his two lowest, second with 4th 1 , 3rd chooses extr
{

    int firstFirstLowest = int.MaxValue;
    int firstSecondLowest = int.MaxValue;
    int firstFirstPosition = 0;
    int firstSecondPosition = 0;
    int secondFirstLowest = int.MaxValue;
    int secondFirstPosition = 0;
    int thirdFirstLowest = int.MaxValue;
    int thirdSecondLowest = int.MaxValue;
    int thirdFirstPosition = 0;
    int thirdSecondPosition = 0;
    int fourthFirstHighest = int.MinValue;
    int fourthFirstPosition = 0;
    int lastFirstHighest = int.MinValue;
    int lastSecondsHighest = int.MinValue;
    int lastFirstPosition = 0;
    int lastSecondPosition = 0;

    Character starting = roundsTable[0];

    taxingText.text = "Taxing!";
    taxingTextCon.SetBool("taxingTextAnimStart", true);

    for (int i = 0; i < roundsTable[0].playerHand.Length; i++)
    {
        if (roundsTable[0].playerHand != null)
        {
            int value1 = roundsTable[0].playerHand[i].GetComponent<CardHandler>().value;

            if (lastFirstHighest < value1)
            {
                lastSecondsHighest = lastFirstHighest;
                lastFirstHighest = value1;
                lastFirstPosition = i;
            }
            else if (lastSecondsHighest < value1)
            {
                lastSecondsHighest = value1;
                lastSecondPosition = i;
            }

        }
    }
```

Σχήμα 3.41: 'Taxing' μέθοδος

Η μέθοδος Taxing είναι η διαδικασία με την οποία εναλλάσσονται κάρτες στην αρχή κάθε καινούργιου γύρου. Μέσω της μεταβλητής πίνακα ‘roundsTable’ την οποία κάθε φορά που κάποιος παίκτης τελειώνει με τις κάρτες του, μπαίνει στον πίνακα με την σωστή σειρά που βγήκε. Έτσι, έχουμε την σωστή σειρά παικτών ώστε να κάνουμε την ανταλλαγή του ‘Taxing’. (Σχήμα 3.40)

Κατά την διάρκεια του γύρου στην Update μέθοδο ελέγχουμε τις κάρτες που έχει ο κάθε παίκτης στο χέρι του ώστε να ξέρουμε πότε έχει τελειώσει.



```
    if (playerFinished == true && playerAdded == false) // adding round winners to a list
    {
        turnPositions.Add(player);
        playerAdded = true;
    }
    if (enemy1Finished == true && enemy1Added == false)
    {
        turnPositions.Add(enemy1);
        enemy1Added = true;
    }
    if (enemy2Finished == true && enemy2Added == false)
    {
        turnPositions.Add(enemy2);
        enemy2Added = true;
    }
    if (enemy3Finished == true && enemy3Added == false)
    {
        turnPositions.Add(enemy3);
        enemy3Added = true;
    }
    if (enemy4Finished == true && enemy4Added == false)
    {
        turnPositions.Add(enemy4);
        enemy4Added = true;
    }
```

Σχήμα 3.42: Έλεγχος γύρου

Αφού κάποιος ‘αδειάσει’, η μεταβλητή του κάθε αντικειμένου παίκτη που ελέγχει αν τελειώσει αλλάζει σε true και τον προσθέτουμε στον πίνακα ελέγχου του γύρου (Σχήμα 3.41).

```
1 reference | 0 changes | 0 authors, 0 changes
public void RoundEnd()
{


    for (int i = 0; i < turnPositions.Count(); i++)
    {
        if (i == 0)
        {
            turnPositions[0].roundPoints = 5;
            turnPositions[0].totalPoints += 5;

        }
        if (i == 1)
        {
            turnPositions[1].roundPoints = 4;
            turnPositions[1].totalPoints += 4;
        }
        if (i == 2)
        {
            turnPositions[2].roundPoints = 3;
            turnPositions[2].totalPoints += 3;
        }
        if (i == 3)
        {
            turnPositions[3].roundPoints = 2;
            turnPositions[3].totalPoints += 2;
        }
        if (i == 4)
        {
            turnPositions[4].roundPoints = 1;
            turnPositions[4].totalPoints = 1;
        }
    }

    roundsTable = roundsTable.OrderByDescending(x => (int)x.roundPoints).ToList();
    playersTable = playersTable.OrderByDescending(x => (int)x.totalPoints).ToList();

    for (int i = 0; i < playersTable.Count(); i++)
    {
        placeName[i].text = playersTable[i].playerName;
        placePoints[i].text = playersTable[i].totalPoints.ToString();
        lastRoundPosition[i].text = roundsTable[i].roundPoints.ToString();
    }
}
```

Σχήμα 3.43: Διάταξη παικτών με πόντους

Όταν τελειώσει ο γύρος ξεκινάει το Game State RoundEnding και η μέθοδος RoundEnd() με την οποία προσθέτουμε τους πόντους που κέρδισε ο κάθε παίκτης ανάλογα με την θέση που τελείωσε. Ταυτόχρονα, γεμίζουμε και τις τιμές που εμφανίζονται στο 'Scoreboard' Panel (Σχήμα 3.42).

```
2 references | 0 changes | 0 authors, 0 changes
public void CleanDeck()
{

    GameObject[] myHandOldCards = player.playerHand;
    GameObject[] myPlayingHandOldCards = playingHand;
    GameObject[] enemy1PlayingHandOldCards = enemy1.playerHand;
    GameObject[] enemy2PlayingHandOldCards = enemy2.playerHand;
    GameObject[] enemy3PlayingHandOldCards = enemy3.playerHand;
    GameObject[] enemy4PlayingHandOldCards = enemy4.playerHand;
    GameObject[] playingArea = playingAreaCards;
    GameObject[] oldExtraCards = extraAreaCards;

    GameObject[] oldCards = myHandOldCards.Concat(myPlayingHandOldCards).Concat(enemy1PlayingHandOldCards).Concat(enemy2PlayingHandOldCards).Concat(ene
ToArray();

    foreach (GameObject card in oldCards)
    {
        Destroy(card);
    }
}
```

Σχήμα 3.44: 'Clean Deck' μέθοδος

Μαζί με το τέλος του γύρου ξεκινάει η μέθοδος CleanDeck() όπου καταστρέφουμε την προηγούμενη τράπουλα για να φτιάξουμε την καινούργια (Σχήμα 3.43). Μετά την επαναφορά των μεταβλητών στις αρχικές τιμές τους ξεκινάμε τις μεθόδους που φτιάχνουν τον γύρο.

```
if(playersTable[0].totalPoints > targetPoints)
{
    curState = GameState.EndGame;
}
else
{
    endTheRound = true;
    curState = GameState.RoundEnding;
}
```

Σχήμα 3.45: Έλεγχος τέλος παιχνιδιού

Πριν τελειώσει κάθε γύρος και προχωρήσουμε στον επόμενο βλέπουμε αν ο παίκτης στην πρώτη θέση (με τους περισσότερους συνολικούς πόντους) έχει φτάσει το target ποσό που διαλέξαμε στην αρχή του παιχνιδιού, αλλιώς προχωράμε στον επόμενο γύρο (Σχήμα 3.44).

3.1.7 Άλλες λειτουργίες για καλύτερη εμπειρία χρήστη

```
Assembly-CSharp                    CardFocus                    canvas
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

0 references | 0 changes | 0 authors, 0 changes
public class CardFocus : MonoBehaviour
{
    public GameObject canvas;
    GameObject focusedCard;
    GameHandler handleChar;
    bool isFocused = false;

    0 references | 0 changes | 0 authors, 0 changes
    public void Awake()
    {
        canvas = GameObject.Find("Canvas1");
        handleChar = FindObjectOfType(typeof(GameHandler)) as GameHandler;
    }

    0 references | 0 changes | 0 authors, 0 changes
    void Start()
    {
    }

    0 references | 0 changes | 0 authors, 0 changes
    void Update()
    {
    }

    0 references | 0 changes | 0 authors, 0 changes
    public void OnMouseOver()
    {
        if (!isFocused && gameObject.tag == handleChar.player.playerHandTag && gameObject.layer != 7)
        {
            isFocused = true;

            focusedCard = Instantiate(gameObject, new Vector2(Input.mousePosition.x, Input.mousePosition.y + 250), Quaternion.identity);
            focusedCard.transform.SetParent(canvas.transform, true);
            focusedCard.layer = LayerMask.NameToLayer("Focus");
            RectTransform rect = focusedCard.GetComponent<RectTransform>();
            rect.sizeDelta = new Vector2(240, 344);
        }
    }

    0 references | 0 changes | 0 authors, 0 changes
    public void OnMouseExit()
    {
        if (isFocused)
        {
            isFocused = false;
            Destroy(focusedCard);
        }
    }
}
```

Σχήμα 3.46: 'Card Focus' Script

Σε κάθε Game Object κάρτας έχουμε προσθέσει το script ‘Card Focus’ (Σχήμα 3.43), στο οποίο αφού τσεκάρουμε ότι είναι κάρτα του χρήστη δημιουργούμε ένα καινούργιο Game Object clone της κάρτας στο ίδιο σημείο αλλά σε μεγαλύτερο μέγεθος για καθαρότερη ανάγνωση.(Σχήμα 3.44)

Σχήμα 3.47: Μια κάρτα με το zoom

'Charge Button' Script: Κάθε κουμπί για καλύτερη και πιο προσεκτική χρήση θέλει λίγο χρόνο κρατημένο ώστε να κάνουμε την επιλογή μας.

```
using UnityEngine.UI;

0 references | 0 changes | 0 authors, 0 changes
public class Chargebutton : MonoBehaviour, IPointerDownHandler, IPointerUpHandler
{
    bool pointerDown;
    float pointerDownTimer;

    [SerializeField] private float requiredHoldTime;
    public UnityEvent onLongClick;
    [SerializeField] private Image fillImage;

    0 references | 0 changes | 0 authors, 0 changes
    public void OnPointerDown(PointerEventData eventData)
    {
        pointerDown = true;
    }

    0 references | 0 changes | 0 authors, 0 changes
    public void OnPointerUp(PointerEventData eventData)
    {
        Reset();
    }

    0 references | 0 changes | 0 authors, 0 changes
    void Update()
    {
        if(gameObject.GetComponent<Button>().interactable)
        {
            if (pointerDown)
            {
                pointerDownTimer += Time.deltaTime;
                if (pointerDownTimer >= requiredHoldTime)
                {
                    if (onLongClick != null)
                        onLongClick.Invoke();

                    Reset();
                }
                fillImage.fillAmount = pointerDownTimer / requiredHoldTime;
            }
        }
    }

    2 references | 0 changes | 0 authors, 0 changes
    private void Reset()
    {
        pointerDown = false;
        pointerDownTimer = 0;
        fillImage.fillAmount = pointerDownTimer / requiredHoldTime;
    }
}
```

Σχήμα 3.48: 'Charge Button' Script

Δίνοντας ένα χρονικό περιθώριο με την Time.deltaTime γεμίζουμε ταυτόχρονα ένα Sprite που έχουμε διαλέξει για να καταλάβει ο χρήστης πόσο πρέπει να κρατήσει το κουμπί μέχρι να επικυρωθεί η επιλογή του. Στην περίπτωση που δεν το κάνει, ακυρώνεται η λειτουργία του κουμπιού.(Σχήμα 3.47)

Σχήμα 3.49: Το 'charge effect' σε ισχύ

Σχήμα 3.50: Τελική οθόνη παιχνιδιού

Αφού κάποιος παίκτης φτάσει το πόσο που έχουμε ως στόχο το παιχνίδι τελειώνει, το ‘scoreboard’ panel (Σχήμα 3.49) ανοίγει με την τελική βαθμολογία, το λεκτικό του νικητή και μετά από κάποιο χρονικό όριο που έχουμε δώσει εμφανίζεται το κουμπί εξόδου, ώστε να γυρίσουμε στο κεντρικό μενού.

Κεφάλαιο 4ο: Συμπεράσματα

Στην παρούσα διπλωματική εργασία είδαμε τον τρόπο ανάπτυξης του επιτραπέζιου παιχνιδιού "Slave" σε περιβάλλον 2D με το πρόγραμμα Unity. Κατά τη διάρκεια της έρευνας, αντιμετωπίστηκαν πολλές προγραμματιστικές προκλήσεις που καλύπτουν ένα ευρύ φάσμα λειτουργικοτήτων στο παιχνίδι "Slave" στο Unity. Τα κύρια προβλήματα που παρουσιάστηκαν αντιμετωπίστηκαν με τους εξής τρόπους:

1, Διασύνδεση UI με C#:

Χρήση walkthroughs στο YouTube και πτυχιακές εργασίες ως υλικό αναφοράς για τη σωστή διασύνδεση του UI στοιχείου της Unity με τον κώδικα C#. Ανάπτυξη μεθόδων για τη διαχείριση γεγονότων χρήστη, όπως το κλικ σε ένα κουμπί, προκειμένου να επιτευχθεί η αναμενόμενη συμπεριφορά.

2, Fisher-Yates Shuffle Αλγόριθμος:

Ενσωμάτωση του αλγόριθμου Fisher-Yates για την ανάμιξη των καρτών στο παιχνίδι, εξασφαλίζοντας ένα τυχαίο και δίκαιο μοίρασμα στους παίκτες κατά την έναρξη του παιχνιδιού και κάθε νέου γύρου.

3, Προγραμματιστικές Προκλήσεις για το UI:

Για τη μεγέθυνση των καρτών με στόχο τη βελτίωση της αναγνωσιμότητας, δημιουργήθηκε ένα κουμπί με φόρτιση (charge) για καλύτερη εμπειρία χρήστη, και εφαρμογή αλγορίθμων για τη διαχείριση των animations στο board, τους παίκτες, και τις μετακινήσεις των καρτών.

4, Διατήρηση Κατάστασης με το GameState:

Χρήση του GameState της C# & Unity για την εντοπισμό της κατάστασης του παιχνιδιού. Εφαρμογή αλγορίθμων για την αλλαγή της σειράς των παικτών, τη διατήρηση στοιχείων προηγούμενου γύρου, και την εναλλαγή καρτών σύμφωνα με τις απαιτήσεις των κανόνων.

Συνοψίζοντας, η κατασκευή του "Slave" στο περιβάλλον Unity αποτέλεσε μια εξαιρετική ευκαιρία για την απόκτηση νέων δεξιοτήτων στον τομέα της προγραμματισμού, καθώς και για τη βαθύτερη κατανόηση των πλατφορμών Unity και της γλώσσας προγραμματισμού C#. Παρά τα πολλές φορές αναπάντεχα προβλήματα που προέκυψαν, επιτεύχθηκε ο αρχικός στόχος της δημιουργίας ενός ολοκληρωμένου και λειτουργικού παιχνιδιού.

Η χρήση της Unity σε παρόμοια προγραμματιστικά προβλήματα θα ήταν μια σοφή επιλογή κυρίως λόγω της εξοικονόμησης χρόνου που μπορεί να προσφέρει στον προγραμματιστή όσο και λόγω της δυνατότητας να χειρίζεται εναλλασσόμενες καταστάσεις μέσω του Game State καθιστώντας ως προτιμότερη μηχανή για ανάπτυξη ενός χαρτοπαιχνιδιού με πολλαπλούς χρήστες.

Πιθανές αλλαγές και βελτιώσεις σε μελλοντικό χρόνο θα μπορούσαν να αποτελέσουν αρχικά η ανάπτυξη του 'multiplayer', ώστε να γίνεται ταυτόχρονη χρήση του παιχνιδιού από περισσότερους παίκτες δημιουργώντας περισσότερες δυνατότητες και προσδίδοντας πιο επιθυμητό χαρακτήρα στο παιχνίδι. Επιπλέον, ποικίλες όψεις του παιχνιδιού θα μπορούσαν να αλλάξουν όπως διάφορες έξτρα διαδράσεις σε μερικές κάρτες με σκοπό το να γίνει η ροή του παιχνιδιού πιο απρόβλεπτη και

διασκεδαστική. Τέλος μπορούν να προκύψουν διάφορες βελτιώσεις ανάλογα πάντα με την ανατροφοδότηση από τους χρήστες.

Κεφάλαιο 5ο: Βιβλιογραφία

[1] Unity Technologies. (2024) Unity Documentation, from https://docs.unity3d.com/ScriptReference/GameObject.html

[2] Unity Technologies. (2024) Unity Documentation, from https://docs.unity3d.com/Manual/Components.html

[3] Unity Technologies. (2024) Unity Documentation, from https://docs.unity3d.com/Manual/Collider2D.html

[4] Unity Technologies. (2024) Unity Documentation, from https://docs.unity3d.com/Manual/CreatingPrefabs.html

[5] Unity Technologies. (2024) Unity Documentation, from https://docs.unity3d.com/Manual/CreatingScenes.html

[6] Unity Technologies. (2024) Unity Documentation, from https://docs.unity3d.com/Manual/class-TagManager.html

[7] Unity Technologies. (2024) Unity Documentation, from https://docs.unity3d.com/Manual/StateMachineBasics.html

[8] Unity Technologies. (2024) Unity Documentation, from https://docs.unity3d.com/Manual/Console.html

[9] Unity Technologies. (2024) Unity Documentation, from https://docs.unity3d.com/ScriptReference/Transform.html

[10] Unity Technologies. (2024) Unity Documentation, from https://docs.unity3d.com/Manual/class-Rigidbody2D.html

[11] Learn Unity(2024) Unity Documentation, from https://learn.unity.com/tutorial/introduction-to-sprite-animations

[12] Unity Technologies. (2024) Unity Documentation, from https://docs.unity3d.com/Manual/class-AudioSource.html

[13] Unity Technologies. (2024) Unity Documentation, from https://docs.unity3d.com/Manual/class-Camera.html

[14] Unity Technologies. (2024) Unity Documentation, from https://docs.unity3d.com/Manual/ScriptingSection.html

[15] Unity Technologies. (2024) Unity Documentation, from https://docs.unity3d.com/ScriptReference/Time-deltaTime.html

[16] Unity Technologies. (2024) Unity Documentation, from https://docs.unity3d.com/ScriptReference/Time.html

Κεφάλαιο 6ο: Application Note

Για την ανάπτυξη της Π.Ε χρησιμοποιήθηκαν:

- Unity (version 2023.3.0f1 Personal)
- Visual Studio 2017