ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«Δημιουργία Ηλεκτρονικού Παιχνιδιού RPG »

Του φοιτητή
Χριστόδουλο Κεστελίδη
Αρ. Μητρώου: 154469

Επιβλέπων
Στέφανος Ουγγιάρογλου
Επίκουρος Καθηγητής

Σεπτέμβριος 2025

Πρόλογος

Η δημιουργία ενός παιχνιδιού αποτελεί πάντα μια πρόκληση που συνδυάζει τεχνικές δεξιότητες με δημιουργικότητα. Η παρούσα εργασία αποτέλεσε για μένα μια μοναδική ευκαιρία να εμβαθύνω στις βασικές τεχνολογίες του web και να εφαρμόσω τις γνώσεις μου στη δημιουργία ενός πλήρως λειτουργικού παιχνιδιού. Μέσα από αυτή τη διαδικασία, όχι μόνο κατάφερα να κατανοήσω σε βάθος τον τρόπο λειτουργίας του browser και τη διαχείριση των δεδομένων και των συμβάντων, αλλά απέκτησα και εμπειρία στη σχεδίαση αρχιτεκτονικών προτύπων που διασφαλίζουν τη συντηρησιμότητα και την επεκτασιμότητα ενός έργου.

Θέλω να ευχαριστήσω όλους όσους με στήριξαν σε αυτή την προσπάθεια, και ειδικά τους καθηγητές μου για τις πολύτιμες κατευθύνσεις και τη γνώση που μου προσέφεραν.

Περίληψη

Η παρούσα πτυχιακή εργασία εστιάζει στην ανάλυση και τεκμηρίωση της ανάπτυξης ενός browser-based turn-based παιχνιδιού, υλοποιημένου αποκλειστικά με τη χρήση HTML, CSS και Vanilla JavaScript. Στόχος είναι η παρουσίαση της αρχιτεκτονικής, των βασικών συστατικών και των μηχανισμών που στηρίζουν τη λειτουργία του παιχνιδιού, όπως το game loop, ο χειρισμός συμβάντων και η διαχείριση του game state. Επιπλέον, αναδεικνύονται οι τεχνικές δυσκολίες που προέκυψαν κατά την υλοποίηση, καθώς και οι δυνατότητες επέκτασης και βελτίωσης του έργου. Η εργασία υπογραμμίζει τη σημασία της καθαρής αρχιτεκτονικής και της ορθής οργάνωσης του κώδικα, αποδεικνύοντας ότι η δημιουργία λειτουργικών και επεκτάσιμων παιχνιδιών είναι εφικτή χωρίς τη χρήση εξειδικευμένων εργαλείων ή frameworks.

«Development of an Electronic RPG Game»

«Christodoulos Kestelidis»

Abstract

This thesis focuses on the analysis and documentation of the development of a browser-based turn-based game, implemented solely using HTML, CSS, and Vanilla JavaScript. The main objective is to present the architecture, key components, and mechanisms underlying the game's functionality, including the game loop, event handling, and game state management. Furthermore, the technical challenges encountered during development, as well as the possibilities for future expansion and improvement, are highlighted. This work emphasizes the importance of clean architecture and proper code organization, demonstrating that the creation of functional and scalable games is feasible without the use of specialized tools or frameworks.

Ευχαριστίες

Αφιερώνω αυτή την εργασία στην οικογένειά μου, που με στήριξε αδιάκοπα καθ’ όλη τη διάρκεια των σπουδών μου, καθώς και στους καθηγητές μου που με ενέπνευσαν και με καθοδήγησαν. Επίσης, αφιερώνω αυτό το έργο σε όλους όσους αγαπούν να δημιουργούν και να μαθαίνουν, γιατί η γνώση είναι το μεγαλύτερο ταξίδι.

Περιεχόμενα

Κεφάλαιο 1ο: Εισαγωγή

Η ραγδαία ανάπτυξη της τεχνολογίας τις τελευταίες δεκαετίες έχει επηρεάσει καθοριστικά τον χώρο της ψυχαγωγίας, με τα ηλεκτρονικά παιχνίδια (video games) να αποτελούν έναν από τους σημαντικότερους τομείς της ψηφιακής κουλτούρας. Από απλές μορφές διασκέδασης που ξεκίνησαν τη δεκαετία του 1970, τα video games έχουν εξελιχθεί σε πολύπλοκες, διαδραστικές εμπειρίες που συνδυάζουν στοιχεία αφήγησης, τεχνολογίας, τέχνης και κοινωνικής αλληλεπίδρασης.

Σήμερα, η βιομηχανία των video games δεν περιορίζεται μόνο στη διασκέδαση, αλλά επεκτείνεται σε τομείς όπως η εκπαίδευση, η προσομοίωση, η κοινωνική δικτύωση και ακόμη και η επαγγελματική ενασχόληση μέσω των esports.
Η παρούσα εργασία εντάσσεται σε αυτό το πλαίσιο, εστιάζοντας τόσο στην ανάλυση των video games ως φαινόμενο, όσο και στη δημιουργία ενός δικού μας παιχνιδιού RPG, με στόχο την κατανόηση των τεχνολογιών, των μηχανισμών και των προκλήσεων που εμπεριέχει η ανάπτυξη ενός τέτοιου έργου.

1.1 Video Games

Τα ηλεκτρονικά παιχνίδια (video games) αποτελούν διαδραστικά ψηφιακά μέσα ψυχαγωγίας, στα οποία ο χρήστης αλληλεπιδρά με ένα εικονικό περιβάλλον μέσω υπολογιστικών συστημάτων, κονσολών ή φορητών συσκευών. Η βασική τους διαφορά από άλλες μορφές ψηφιακής ψυχαγωγίας έγκειται στο γεγονός ότι απαιτούν τη συνεχή συμμετοχή και τις αποφάσεις του παίκτη, οι οποίες επηρεάζουν άμεσα την εξέλιξη της εμπειρίας [3].

Η ιστορία των video games ξεκινά τη δεκαετία του 1950, όταν σε πανεπιστημιακά εργαστήρια υπολογιστών δημιουργήθηκαν οι πρώτες απλές προσομοιώσεις, όπως το **Tennis for Two** (1958) του William Higinbotham και το **Spacewar!** (1962) από τους Steve Russell και την ομάδα του στο MIT. Στη δεκαετία του 1970, με την κυκλοφορία του **Pong** (1972) από την Atari, τα video games άρχισαν να αποκτούν εμπορικό χαρακτήρα και να καθίστανται δημοφιλή στο ευρύ κοινό. Η περίοδος αυτή σηματοδότησε τη γέννηση της βιομηχανίας των video games, η οποία γνώρισε ραγδαία ανάπτυξη τα επόμενα χρόνια [2]/[1].

Κατά τη δεκαετία του 1980 παρατηρήθηκε σημαντική πρόοδος τόσο στο σχεδιασμό όσο και στην τεχνολογία των παιχνιδιών, με την εισαγωγή προσωπικών υπολογιστών και φορητών συσκευών, ενώ η δεκαετία του 1990 σηματοδότησε την καθιέρωση των τρισδιάστατων γραφικών και των κονσολών νέας γενιάς, όπως το PlayStation και το Nintendo 64. Στον 21ο αιώνα, η ανάπτυξη του διαδικτύου και των ψηφιακών αγορών μετέβαλε εκ νέου το τοπίο, καθιστώντας δυνατή τη διαδικτυακή συνεργασία και αντιπαράθεση μεταξύ παικτών (multiplayer online gaming), ενώ ταυτόχρονα άνοιξε τον δρόμο για την άνοδο των ανεξάρτητων (indie) δημιουργών και της διανομής παιχνιδιών μέσω πλατφορμών όπως το Steam [2].

Τα video games, πέραν του ρόλου τους ως μορφή ψυχαγωγίας, έχουν πλέον αναγνωριστεί και ως πολιτιστικό και καλλιτεχνικό μέσο. Αποτελούν εργαλείο αφήγησης, προσομοίωσης κοινωνικών καταστάσεων και ανάπτυξης κριτικής σκέψης. Επιπλέον, η βιομηχανία των video games συγκαταλέγεται πλέον ανάμεσα στις πιο κερδοφόρες στον τομέα της ψυχαγωγίας, ξεπερνώντας σε παγκόσμια έσοδα τη μουσική και τον κινηματογράφο.

Η συνεχής εξέλιξή τους συνδέεται με την πρόοδο της τεχνολογίας: από την τεχνητή νοημοσύνη και τη φυσική αλληλεπίδραση (motion sensing) μέχρι την εικονική και επαυξημένη πραγματικότητα, τα video games προσαρμόζονται στις νέες συνθήκες και προωθούν καινοτόμες εμπειρίες. Ως εκ τούτου, η μελέτη τους δεν αφορά μόνο την ψυχαγωγία, αλλά αγγίζει πεδία όπως η εκπαίδευση, η επιστημονική έρευνα και η κοινωνική ανάπτυξη.

Εκτός από την τεχνολογική τους εξέλιξη και τον πολιτιστικό τους αντίκτυπο, τα video games διαφοροποιούνται και σε σχέση με το περιεχόμενο, τον στόχο και τη μορφή αλληλεπίδρασης που προσφέρουν στους παίκτες. Στη βιβλιογραφία συναντάμε διάφορα σχήματα κατηγοριοποίησης, με τις πιο συχνές κατηγορίες να περιλαμβάνουν [9]:

- **Action games**: παιχνίδια που βασίζονται στην ταχύτητα, την αντανακλαστικότητα και τον συντονισμό του παίκτη.
- **Role-Playing Games (RPGs)**: παιχνίδια όπου ο παίκτης αναλαμβάνει τον ρόλο ενός χαρακτήρα και εξελίσσεται μέσα σε έναν φανταστικό κόσμο.
- **Strategy games**: εστιάζουν στον σχεδιασμό και τη λήψη αποφάσεων, συχνά σε πραγματικό χρόνο ή με τη μορφή γύρων.
- **Simulation games**: αναπαράγουν πτυχές της πραγματικής ζωής, όπως η πτήση, η οδήγηση ή η διαχείριση πόρων.
- **Sports & Racing games**: αναπαραστάσεις αθλημάτων ή αγωνισμάτων ταχύτητας.
- **Adventure games**: παιχνίδια που δίνουν έμφαση στην εξερεύνηση και την αφήγηση.

Η παραπάνω ταξινόμηση λειτουργεί ως μια πρώτη ένδειξη της ποικιλομορφίας των ηλεκτρονικών παιχνιδιών. Η αναλυτική παρουσίαση και διερεύνηση των βασικών κατηγοριών θα πραγματοποιηθεί εκτενώς στο **Κεφάλαιο 2 – Κατηγορίες Video Games**, όπου θα επιχειρηθεί και μια σύνδεση με το είδος του RPG που αναπτύχθηκε στο πλαίσιο της παρούσας εργασίας.

1.2 Κίνητρο και Συνεισφορά

Η ανάπτυξη ενός ηλεκτρονικού παιχνιδιού στο πλαίσιο μιας πτυχιακής εργασίας δεν αποτελεί μόνο μια δημιουργική διαδικασία αλλά και μια ουσιαστική ακαδημαϊκή πρόκληση. Το κίνητρο πίσω από την παρούσα εργασία πηγάζει από την αυξανόμενη σημασία που κατέχουν τα video games στη σύγχρονη κοινωνία, τόσο ως μορφή ψυχαγωγίας όσο και ως πεδίο τεχνολογικής καινοτομίας. Τα παιχνίδια έχουν εξελιχθεί σε έναν από τους πιο κερδοφόρους κλάδους της παγκόσμιας βιομηχανίας των μέσων, με έσοδα που ξεπερνούν πλέον τον κινηματογράφο και τη μουσική βιομηχανία.

Σε προσωπικό επίπεδο, το κίνητρο για την επιλογή του θέματος συνδέεται με την επιθυμία εμβάθυνσης στην ανάπτυξη διαδραστικών εφαρμογών που συνδυάζουν προγραμματισμό, γραφικά και αφήγηση. Η υλοποίηση ενός **RPG (Role-Playing Game)** αποτελεί ιδανική πρόκληση, καθώς το συγκεκριμένο είδος απαιτεί συνδυασμό μηχανισμών μάχης, διαχείρισης πόρων και σεναριακής εξέλιξης. Έτσι, η εργασία δεν περιορίζεται στην τεχνική διάσταση, αλλά αγγίζει και το πεδίο της σχεδιαστικής σκέψης και της αφηγηματικής εμπειρίας, στοιχεία που αναγνωρίζονται ως κομβικά στη θεωρία του game design [11].

Η συνεισφορά της εργασίας εντοπίζεται σε δύο κύριες διαστάσεις:

1. **Ακαδημαϊκή συνεισφορά**: Μέσα από την ανάλυση και την υλοποίηση παρουσιάζεται ένα παράδειγμα κατασκευής ηλεκτρονικού παιχνιδιού με χρήση **web technologies (HTML, CSS, JavaScript)** χωρίς την αξιοποίηση εξωτερικών frameworks. Με αυτόν τον τρόπο επιδεικνύεται πώς μπορούν να αξιοποιηθούν οι βασικές τεχνολογίες του ιστού για την ανάπτυξη πλήρως λειτουργικών και διαδραστικών εφαρμογών, προσφέροντας μια διδακτική βάση για φοιτητές και ερευνητές.
2. **Πρακτική συνεισφορά**: Η εργασία παρέχει έναν ολοκληρωμένο οδηγό-πρότυπο για τη δημιουργία ενός RPG παιχνιδιού, παρουσιάζοντας τη δομή αρχείων, την αρχιτεκτονική, τους μηχανισμούς μάχης και τις τεχνικές διαχείρισης καταστάσεων. Επιπλέον, η εμπειρία ανάπτυξης καταγράφει τα εμπόδια και τις λύσεις που υιοθετήθηκαν, γεγονός που μπορεί να φανεί χρήσιμο σε μελλοντικούς δημιουργούς παιχνιδιών.

Τέλος, η ερευνητική αξία της παρούσας εργασίας έγκειται στη σύνδεση θεωρίας και πράξης: από την παρουσίαση του θεωρητικού υπόβαθρου για τα video games και τις κατηγορίες τους, έως την ανάλυση του τρόπου με τον οποίο τα τεχνικά εργαλεία και οι μηχανισμοί παιχνιδιού μπορούν να αποδώσουν μια ολοκληρωμένη εμπειρία χρήστη. Έτσι, η εργασία συνεισφέρει στην ακαδημαϊκή συζήτηση γύρω από την αξιοποίηση των τεχνολογιών ιστού στη δημιουργία διαδραστικών εμπειριών και επιβεβαιώνει τον παιδαγωγικό ρόλο που μπορούν να διαδραματίσουν τα παιχνίδια στην εκπαιδευτική διαδικασία.

1.3 Περιγραφή του παιχνιδιού

Το παιχνίδι που αναλύεται στην παρούσα εργασία αποτελεί έναν **2D browser-based τίτλο στρατηγικής με μηχανισμό μάχης που βασίζεται σε γύρους (turn-based combat system)**. Ο παίκτης αναλαμβάνει τον έλεγχο μιας ομάδας από φανταστικά "πιτσόμορφα" πλάσματα (γνωστά και ως *pizzas*) και καλείται να αντιμετωπίσει αντίπαλες ομάδες σε διαδοχικές μάχες, οι οποίες εκτυλίσσονται σε περιβάλλον HTML canvas και JavaScript DOM rendering.

Οπτικά, το παιχνίδι υιοθετεί μια αισθητική τύπου **pixel-art**, με χαρακτήρες και περιβάλλοντα που έχουν δημιουργηθεί σε μορφή sprites. Η διεπαφή χρήστη είναι απλή και λειτουργική, με τα βασικά στοιχεία του gameplay να παρουσιάζονται καθαρά στον παίκτη: διαθέσιμες ενέργειες, κατάσταση ζωής (HP), ενεργά status effects και turn order.

Κατά τη διάρκεια κάθε μάχης, ο παίκτης και ο αντίπαλος εναλλάσσονται στον ρόλο του επιτιθέμενου. Ο παίκτης επιλέγει μία από τις διαθέσιμες ενέργειες (attacks ή abilities), οι οποίες μπορεί να προκαλέσουν ζημιά στον εχθρό, να επηρεάσουν την κατάστασή του (π.χ. *status effects* όπως "clumsy" ή "saucy") ή να προσφέρουν υποστήριξη στην ομάδα του (π.χ. *healing* ή *status recovery*). Η στρατηγική έγκειται στην έξυπνη χρήση των ενεργειών, στην αλλαγή μελών της ομάδας (π.χ. αντικατάσταση ενός τραυματισμένου pizza), αλλά και στη διαχείριση αντικειμένων (*items*) τα οποία ο παίκτης μπορεί να χρησιμοποιήσει κατά τη διάρκεια της μάχης.

Η κάθε πίτσα (μονάδα μάχης) διαθέτει τα εξής χαρακτηριστικά:

- **HP (Health Points)**: Η ζωή της μονάδας, η οποία μειώνεται όταν δέχεται επίθεση.
- **Level & XP**: Το επίπεδο εμπειρίας και η πρόοδός της προς το επόμενο επίπεδο.
- **Type**: Ο τύπος της πίτσας (π.χ. *spicy*, *veggie*, *fungi*) που θα μπορούσε σε μελλοντική ανάπτυξη να έχει ρόλο σε συστήματα τύπου "rock-paper-scissors".
- **Available Actions**: Οι επιθέσεις/ενέργειες που έχει στη διάθεσή της.

Η ροή του παιχνιδιού καθορίζεται από έναν μηχανισμό **"TurnCycle"**, ο οποίος διαχειρίζεται εναλλάξ τις ενέργειες του παίκτη και του αντιπάλου, παρακολουθεί την κατάσταση κάθε μονάδας και ανιχνεύει πότε μια ομάδα χάνει όλα τα ενεργά μέλη της. Σε αυτή την περίπτωση, η μάχη λήγει και προκύπτει ένας νικητής.

Οι αντίπαλοι (enemies) είναι προεγκατεστημένοι χαρακτήρες με συγκεκριμένα sprites, στατιστικά και συνθέσεις ομάδων. Κάθε μάχη συνδέεται με ένα γεγονός (π.χ. η είσοδος σε έναν χώρο ή η συνάντηση με έναν NPC), κάτι που προσδίδει στο παιχνίδι τη μορφή σεναριακής εξέλιξης.

Η υλοποίηση του παιχνιδιού έχει γίνει με στόχο τη **σαφήνεια, την επεκτασιμότητα και την εκπαιδευτική αξία**. Ο κώδικας είναι modular, και κάθε επιμέρους κομμάτι (π.χ. menus, combat mechanics, player state, enemies, actions) βρίσκεται σε ξεχωριστό αρχείο. Με αυτόν τον τρόπο, διευκολύνεται τόσο η συντήρηση όσο και η μελλοντική επέκταση του παιχνιδιού, είτε με προσθήκη νέων λειτουργιών είτε με αλλαγή της δομής χωρίς να επηρεάζεται το υπόλοιπο σύστημα.

Συνολικά, το παιχνίδι αποτελεί ένα πλήρως λειτουργικό παράδειγμα **μηχανισμού turn-based μάχης**, βασισμένο εξολοκλήρου σε τεχνολογίες ιστού και ενδεικτικό των δυνατοτήτων που προσφέρει η JavaScript στην κατασκευή διαδραστικών εφαρμογών χωρίς εξωτερικές βιβλιοθήκες ή game engines.

1.4 Μεθοδολογία ανάλυσης

Η προσέγγιση που ακολουθήθηκε για την ανάλυση της παρούσας εργασίας είναι **δομική και λειτουργική**, με στόχο την πλήρη κατανόηση του τρόπου με τον οποίο αναπτύχθηκε το παιχνίδι, των τεχνολογιών που χρησιμοποιήθηκαν, καθώς και του τρόπου με τον οποίο συνεργάζονται τα επιμέρους μέρη της εφαρμογής.

Η ανάλυση βασίστηκε στις παρακάτω μεθοδολογικές αρχές:

Ανάγνωση και οργάνωση του πηγαίου κώδικα

Αρχικά, ολόκληρη η δομή του έργου εξετάστηκε ώστε να κατανοηθεί η **αρχιτεκτονική του συστήματος**: ποια είναι τα βασικά αρχεία, πώς αυτά διασυνδέονται, και ποιος είναι ο ρόλος τους στο τελικό αποτέλεσμα. Έγινε συστηματική μελέτη των JavaScript αρχείων, της HTML και των στατικών πόρων (εικόνες, sprites, icons κτλ).

Διάκριση λογικών ενοτήτων

Ο πηγαίος κώδικας διαχωρίστηκε σε **λογικές ενότητες** όπως:

- **Κατάσταση παίκτη (PlayerState)**
- **Διαχείριση αντιπάλων (Enemies)**
- **Μοντέλα πίτσας (Pizzas)**
- **Σύστημα ενεργειών (Actions)**
- **Game loop (TurnCycle & Battle flow)**
- **UI Στοιχεία και Animation**

Για κάθε μία από αυτές τις ενότητες, εντοπίστηκαν τα βασικά σημεία λειτουργίας και αλληλεπίδρασης με τα υπόλοιπα μέρη της εφαρμογής.

Παρακολούθηση της ροής εκτέλεσης (execution flow)

Μελετήθηκε η **ροή των γεγονότων** από τη στιγμή που ξεκινά μια μάχη έως τη λήξη της, με σκοπό να γίνει κατανοητός ο μηχανισμός εναλλαγής γύρων, εκτέλεσης ενεργειών, διαχείρισης καταστάσεων (*status effects*), εφαρμογής ζημιάς και ενημέρωσης του interface.

Ιδιαίτερη προσοχή δόθηκε στον **μηχανισμό ενεργοποίησης των events**, όπως εμφανίζονται στα αντικείμενα τύπου textMessage, stateChange, animation, και στον τρόπο με τον οποίο αυτοί "αλυσιδωτά" δημιουργούν την εμπειρία μάχης.

Ανάλυση της λογικής χωρίς εξωτερικές βιβλιοθήκες

Καθώς το παιχνίδι είναι υλοποιημένο αποκλειστικά με **Vanilla JavaScript**, χωρίς τη χρήση frameworks ή engines, δόθηκε έμφαση στο πώς επιτεύχθηκε αυτή η λειτουργικότητα:

- Δημιουργία και διαχείριση DOM στοιχείων με document.createElement
- Ενημέρωση UI με βάση την κατάσταση των αντικειμένων
- Χρήση κλασικής OOP προσέγγισης (με class) και modular αρχιτεκτονικής
- Σχεδιασμός μικρών helper functions όπως το utils.emitEvent() για ανταλλαγή δεδομένων μεταξύ μονάδων

Καταγραφή τεχνικών προκλήσεων και σχεδιαστικών επιλογών

Τέλος, επισημάνθηκαν τα σημεία του κώδικα όπου εντοπίστηκαν τεχνικές προκλήσεις, όπως:

- Η διαχείριση του χρόνου και των animations (με await και custom event systems)
- Η σύνθεση των αντικειμένων Combatant, Team, Battle, που διαχειρίζονται πολλές ευθύνες ταυτόχρονα
- Ο τρόπος με τον οποίο υλοποιήθηκαν τα custom behaviors μέσω JSON-like αντικειμένων (π.χ. actions, events), αποφεύγοντας επανάληψη κώδικα

Η μεθοδολογία αυτή επιτρέπει όχι μόνο την **κατανόηση του τρόπου λειτουργίας** της εφαρμογής, αλλά και την **τεκμηρίωση του σχεδιασμού**, καθώς και την **αποτύπωση πιθανών βελτιώσεων** ή προεκτάσεων σε μελλοντικές εκδόσεις.

1.5 Δομή του εγγράφου

Η παρούσα πτυχιακή εργασία είναι οργανωμένη με τέτοιον τρόπο ώστε να καθοδηγεί τον αναγνώστη από τη γενική περιγραφή του έργου έως την εις βάθος τεχνική ανάλυση των επιμέρους στοιχείων. Η δομή της εργασίας ακολουθεί λογική και σταδιακή πρόοδο, ώστε κάθε κεφάλαιο να οικοδομεί πάνω στις έννοιες και τα συμπεράσματα των προηγούμενων.

Συγκεκριμένα, η εργασία αποτελείται από τα εξής βασικά κεφάλαια:

Κεφάλαιο 1: Εισαγωγή

Παρουσιάζεται το αντικείμενο, τα κίνητρα και οι στόχοι της πτυχιακής. Επιπλέον, περιγράφεται η δομή του εγγράφου ώστε να διευκολυνθεί η ανάγνωση και κατανόηση της πορείας της ανάλυσης.

Κεφάλαιο 2: Κατηγορίες Video Games

Γίνεται αναφορά στις κυριότερες κατηγορίες βιντεοπαιχνιδιών (RPG, Action, Strategy, Simulation, Adventure κ.ά.) και εξετάζονται τα ιδιαίτερα χαρακτηριστικά τους. Δίνεται ιδιαίτερη έμφαση στα RPGs, τα οποία σχετίζονται με το αντικείμενο της εργασίας.

Κεφάλαιο 3: Τεχνολογίες που Χρησιμοποιούνται για την Ανάπτυξη Video Games

Παρουσιάζονται οι τεχνολογίες που χρησιμοποιούνται για την ανάπτυξη παιχνιδιών, από τα game engines (Unity, Unreal Engine, Godot) έως τις τεχνολογίες διαδικτύου (HTML, CSS, JavaScript). Ειδική αναφορά γίνεται στη χρήση *Vanilla JS* και στην επιλογή προσέγγισης “no-framework”.

Κεφάλαιο 4: Τεχνολογικό Υπόβαθρο

Σε αυτό το κεφάλαιο αναλύονται οι τεχνολογίες που χρησιμοποιήθηκαν για την κατασκευή του παιχνιδιού, με έμφαση στη χρήση **Vanilla JavaScript**, HTML και CSS, καθώς και στη φιλοσοφία του "no-framework" development. Εξετάζεται η σημασία της απουσίας εξωτερικών βιβλιοθηκών και η επίδραση αυτής της επιλογής στην αρχιτεκτονική και τη συντήρηση του έργου

Κεφάλαιο 5: Αρχιτεκτονική του Κώδικα

Παρουσιάζεται η συνολική αρχιτεκτονική της εφαρμογής, με αναφορά στις βασικές κατηγορίες αρχείων (State, Content, Battle, UI κτλ), στον ρόλο κάθε αρχείου και στον τρόπο με τον οποίο αυτά επικοινωνούν. Παρέχονται σχήματα και διαγράμματα για την οπτικοποίηση της δομής.

Κεφάλαιο 6: Μηχανισμός Μάχης & Game Loop

Εμβαθύνουμε στον τρόπο που υλοποιείται η "καρδιά" του παιχνιδιού — το σύστημα μάχης. Περιγράφεται η σειρά ενεργειών, ο χειρισμός των γεγονότων (events), οι εναλλαγές γύρων, η διαχείριση ενεργειών (skills, animations, αλλαγές κατάστασης), και η χρήση της κλάσης TurnCycle.

Κεφάλαιο 7: Χρήση UI & DOM Manipulation

Εξετάζεται πώς υλοποιούνται τα οπτικά στοιχεία του παιχνιδιού, πώς ενημερώνεται το DOM κατά τη διάρκεια των μαχών, και πώς αποτυπώνονται οι καταστάσεις του παιχνιδιού στον χρήστη με τη χρήση κλάσεων όπως Team, Combatant, Battle, κ.ά.

Κεφάλαιο 8: Το Μέλλον των Video Games

Εξετάζονται οι σύγχρονες τάσεις και μελλοντικές προοπτικές της βιομηχανίας των παιχνιδιών, όπως το cloud gaming, οι τεχνολογίες VR/AR, η ενσωμάτωση τεχνητής νοημοσύνης και οι κοινωνικές διαστάσεις του gaming. Το κεφάλαιο αυτό λειτουργεί ως τεχνολογικό υπόβαθρο για την κατανόηση της εργασίας.

Κεφάλαιο 9: Συμπεράσματα και προοπτικές

Το τελευταίο κεφάλαιο περιλαμβάνει τη συνολική αποτίμηση του έργου, καθώς και πιθανές προεκτάσεις και βελτιώσεις στο μέλλον — όπως η προσθήκη ήχου, multiplayer δυνατότητες, χρήση canvas/WebGL ή υιοθέτηση frameworks.

Κεφάλαιο 2ο: Κατηγορίες Video Games

Η ταξινόμηση των ηλεκτρονικών παιχνιδιών αποτελεί θεμελιώδες βήμα για την κατανόηση της εξέλιξης και της ποικιλίας τους. Μέσα από την κατηγοριοποίηση αναδεικνύονται οι διαφορετικές εμπειρίες που προσφέρουν, οι σχεδιαστικές φιλοσοφίες που ακολουθούν οι δημιουργοί και οι τρόποι με τους οποίους οι παίκτες αλληλεπιδρούν με τον ψηφιακό κόσμο.

Στη βιβλιογραφία έχουν προταθεί πολυάριθμες προσεγγίσεις ταξινόμησης, από απλούς διαχωρισμούς σε «action» και «strategy» έως πιο πολύπλοκα σχήματα που λαμβάνουν υπόψη κοινωνικές, τεχνολογικές και αφηγηματικές διαστάσεις (Apperley, 2006· Wolf, 2001). Στο παρόν κεφάλαιο υιοθετείται μία λειτουργική ταξινόμηση βασισμένη κυρίως στο gameplay και τους στόχους του κάθε είδους.

Εξετάζονται έξι βασικές κατηγορίες – **Action Games, Role-Playing Games (RPGs), Strategy Games, Simulation Games, Sports & Racing Games και Adventure Games** – οι οποίες αποτυπώνουν την ευρεία γκάμα εμπειριών που μπορεί να προσφέρει ένα video game. Η ανάλυση αυτή αποτελεί το εννοιολογικό υπόβαθρο της εργασίας, καθώς το υπό μελέτη παιχνίδι ανήκει στη συγκεκριμένη οικογένεια των RPGs [10].

2.1 Action Games

Τα **Action Games** αποτελούν ένα από τα πιο διαχρονικά και δημοφιλή είδη ηλεκτρονικών παιχνιδιών. Η βασική τους αρχή στηρίζεται στην ικανότητα του παίκτη να ανταποκρίνεται άμεσα σε ερεθίσματα, γεγονός που τα καθιστά παιχνίδια έντονου ρυθμού, απαιτητικά σε **αντανακλαστικά**, **συντονισμό κινήσεων** και **χρονισμό**. Η πλοκή, παρότι σε ορισμένες περιπτώσεις έχει σημαντικό ρόλο, συχνά παραμένει δευτερεύουσα σε σχέση με το gameplay, το οποίο επικεντρώνεται σε μάχες, γρήγορες αποφάσεις και μηχανισμούς που ενισχύουν την ένταση και την αδρεναλίνη [9].

Χαρακτηριστικά των action games περιλαμβάνουν:

- **Άμεσο έλεγχο του χαρακτήρα**: Ο παίκτης ελέγχει τις κινήσεις σε πραγματικό χρόνο.
- **Συστήματα μάχης**: Συχνά περιλαμβάνουν melee ή ranged combat, με έμφαση στην ταχύτητα εκτέλεσης.
- **Δοκιμασίες ταχύτητας και αντίδρασης**: Όπως αποφυγή εμποδίων, γρήγορη στόχευση, ή ακριβής χειρισμός.
- **Επίπεδα (levels)** ή **πίστες** με αυξανόμενη δυσκολία, που διατηρούν το ενδιαφέρον και προσφέρουν πρόκληση.

Ιστορικά, τα action games έχουν τις ρίζες τους ήδη από τα πρώτα **arcade games** της δεκαετίας του 1970, όπως το *Space Invaders* και το *Pac-Man*, τα οποία καθόρισαν την αίσθηση της πρόκλησης και της επιβράβευσης μέσω σκορ. Στη δεκαετία του 1980 και 1990, με την ανάπτυξη των κονσολών και των προσωπικών υπολογιστών, είδη όπως τα **platformers** (*Super Mario Bros.*, 1985) και τα **beat 'em ups** (*Streets of Rage*, 1991) καθιέρωσαν τους μηχανισμούς που εξακολουθούν να αποτελούν βάση του είδους [22].

Σύγχρονα action games περιλαμβάνουν πλέον πιο σύνθετους μηχανισμούς, με **τρισδιάστατα γραφικά, εξελιγμένα physics engines** και στοιχεία που δανείζονται από άλλα είδη, όπως RPGs και open-world adventures (*Assassin's Creed, Devil May Cry, Dark Souls*). Η συνεχής εξέλιξη του hardware επιτρέπει την υλοποίηση ρεαλιστικών περιβαλλόντων και ταυτόχρονα την ανάπτυξη πιο περίπλοκων συστημάτων μάχης και τεχνητής νοημοσύνης αντιπάλων [3].

Αξίζει επίσης να σημειωθεί ότι τα action games αποτελούν τη «βάση» για πολλά υβριδικά είδη. Για παράδειγμα, τα **Action RPGs** (*The Witcher 3, Elden Ring*) συνδυάζουν την ταχύτητα και την ένταση των action games με την αφηγηματική και μηχανιστική πολυπλοκότητα των RPGs. Αυτό επιβεβαιώνει τη σημασία των action mechanics ως θεμελιώδες στοιχείο του σχεδιασμού παιχνιδιών.

Συνοψίζοντας, τα action games αντιπροσωπεύουν ένα είδος που συνεχίζει να πρωταγωνιστεί στη βιομηχανία, εξελισσόμενο τεχνολογικά και σχεδιαστικά, παραμένοντας παράλληλα άρρηκτα συνδεδεμένο με την ουσία της διαδραστικής ψυχαγωγίας: την **άμεση δράση και την πρόκληση των δεξιοτήτων του παίκτη**.

2.2 Role-Playing Games (RPGs)

Τα **Role-Playing Games (RPGs)** αποτελούν μία από τις πιο πολυδιάστατες και επιδραστικές κατηγορίες ηλεκτρονικών παιχνιδιών, καθώς συνδυάζουν στοιχεία αφήγησης, στρατηγικής, εξερεύνησης και σταδιακής ανάπτυξης χαρακτήρων. Η βασική ιδέα που τα χαρακτηρίζει είναι ότι ο παίκτης αναλαμβάνει τον ρόλο ενός ή περισσότερων χαρακτήρων σε έναν **φανταστικό κόσμο**, συμμετέχοντας ενεργά στην εξέλιξη της ιστορίας και στην πορεία του παιχνιδιού.

Ένα από τα κύρια γνωρίσματα των RPGs είναι η **εξέλιξη του χαρακτήρα**. Οι παίκτες έχουν τη δυνατότητα να βελτιώνουν τις ικανότητες, τα στατιστικά και τον εξοπλισμό των χαρακτήρων τους μέσω ενός **συστήματος επιπέδων (leveling system)** ή μηχανισμών εμπειρίας. Αυτό δημιουργεί ένα αίσθημα προόδου και επένδυσης χρόνου, που διαφοροποιεί τα RPGs από πιο "στατικά" είδη παιχνιδιών.

Παράλληλα, τα RPGs δίνουν ιδιαίτερη βαρύτητα στην **αφήγηση και την πλοκή**. Συχνά διαθέτουν περίπλοκες ιστορίες, με παρακλάδια και επιλογές που επηρεάζουν την έκβαση της πλοκής, ενισχύοντας έτσι την αίσθηση εμβύθισης (immersion). Η ελευθερία επιλογής, όπως το να αποφασίσει ο παίκτης αν θα δράσει ηθικά ή ανήθικα, αποτελεί θεμελιώδες στοιχείο του είδους [12].

Υποκατηγορίες των RPGs περιλαμβάνουν:

- **Turn-Based RPGs**: Οι μάχες διεξάγονται σε γύρους, με έμφαση στη στρατηγική (π.χ. *Final Fantasy*).
- **Action RPGs**: Συνδυάζουν τα στοιχεία δράσης με RPG μηχανισμούς, με μάχες σε πραγματικό χρόνο (π.χ. *Dark Souls, The Witcher 3*).
- **MMORPGs (Massively Multiplayer Online RPGs)**: Ανοικτά διαδικτυακά περιβάλλοντα με χιλιάδες παίκτες που αλληλεπιδρούν ταυτόχρονα (π.χ. *World of Warcraft*).

Η ιστορία των RPGs ξεκινά από τα **tabletop role-playing games**, όπως το *Dungeons & Dragons* (1974), που εισήγαγαν τους κανόνες για την ανάπτυξη χαρακτήρων και τις αλληλεπιδράσεις μέσα σε φανταστικούς κόσμους (Fine, 2002). Με την ψηφιοποίηση αυτής της

εμπειρίας, τα πρώτα RPGs στον χώρο των video games εμφανίστηκαν στη δεκαετία του 1980, με τίτλους όπως το *Ultima* και το *Dragon Quest*.

Στις μέρες μας, τα RPGs έχουν εξελιχθεί σε υβριδικά συστήματα που συνδυάζουν διαφορετικά είδη, όπως **RPG-shooters** (*Mass Effect*), **RPG-strategy hybrids** (*Fire Emblem*), και **sandbox RPGs** (*Skyrim*), επιβεβαιώνοντας την ευελιξία και την ανθεκτικότητα του είδους. Επιπλέον, η χρήση νέων τεχνολογιών, όπως το **cloud gaming** και η **εικονική πραγματικότητα (VR)**, ανοίγουν νέες δυνατότητες για ακόμα πιο εμβυθιστικές εμπειρίες.

Συνολικά, τα RPGs αποτελούν ένα είδος που δεν περιορίζεται απλώς στη διασκέδαση, αλλά συχνά προσφέρει στον παίκτη την ευκαιρία για **αυτοέκφραση, κοινωνική αλληλεπίδραση** και **διερεύνηση ηθικών διλημμάτων**, καθιστώντας τα από τα πιο ολοκληρωμένα και επιδραστικά είδη video games.

2.3 Strategy Games

Τα **Strategy Games** αποτελούν μία από τις σημαντικότερες κατηγορίες ηλεκτρονικών παιχνιδιών, καθώς βασίζονται κυρίως στην **ικανότητα του παίκτη να σχεδιάζει, να αναλύει και να λαμβάνει αποφάσεις** σε διαφορετικά επίπεδα πολυπλοκότητας. Ο πρωταρχικός τους στόχος είναι η ανάπτυξη στρατηγικής σκέψης, η διαχείριση πόρων και η αξιοποίηση τακτικών για την επίτευξη νίκης έναντι αντιπάλων, είτε αυτοί είναι **υπολογιστικοί αλγόριθμοι (AI)** είτε άλλοι παίκτες [13].

Σε αντίθεση με τα παιχνίδια δράσης, όπου κυριαρχούν η ταχύτητα και τα αντανακλαστικά, στα strategy games η **πνευματική ετοιμότητα και η ικανότητα προγραμματισμού κινήσεων** παίζουν τον σημαντικότερο ρόλο. Αυτή η διαφοροποίηση έχει καταστήσει το είδος ιδιαίτερα δημοφιλές σε παίκτες που απολαμβάνουν τον συνδυασμό λογικής, ανάλυσης και προνοητικότητας [14].

Υποκατηγορίες Strategy Games

1. **Turn-Based Strategy (TBS):** Ο παίκτης και οι αντίπαλοι εκτελούν τις κινήσεις τους σε διαδοχικούς γύρους. Η μορφή αυτή προσφέρει χρόνο για στοχασμό και προσεκτική λήψη αποφάσεων. Κλασικά παραδείγματα αποτελούν τα *Civilization* και *XCOM*.
2. **Real-Time Strategy (RTS):** Οι ενέργειες πραγματοποιούνται σε πραγματικό χρόνο, απαιτώντας ταυτόχρονη λήψη αποφάσεων και τακτική σκέψη σε υψηλή ταχύτητα. Σημαντικοί τίτλοι της κατηγορίας είναι τα *StarCraft* και *Age of Empires*.
3. **4X Games (Explore, Expand, Exploit, Exterminate):** Στρατηγικά παιχνίδια μεγάλης κλίμακας που εστιάζουν στην εξερεύνηση, την επέκταση, την εκμετάλλευση πόρων και την εξόντωση αντιπάλων. Το *Master of Orion* και το *Stellaris* ανήκουν σε αυτή την κατηγορία.
4. **Tower Defense & Hybrid Strategy:** Παραλλαγές που συνδυάζουν στρατηγική με στοιχεία δράσης, όπως το *Plants vs Zombies*, δείχνουν τη συνεχή εξέλιξη του είδους και την προσαρμογή του σε διαφορετικά κοινά.

Εκπαιδευτική και κοινωνική διάσταση

Τα strategy games έχουν μελετηθεί εκτενώς και από παιδαγωγική σκοπιά, καθώς συμβάλλουν στην ανάπτυξη γνωστικών δεξιοτήτων όπως η **επίλυση προβλημάτων, η διαχείριση πόρων και η λήψη αποφάσεων υπό πίεση**. Επιπλέον, τα multiplayer strategy games ενισχύουν την

κοινωνική συνεργασία και τον ανταγωνισμό, αφού οι παίκτες καλούνται να συντονιστούν, να συμμαχήσουν ή να αναμετρηθούν σε δυναμικά περιβάλλοντα.

Ιστορική εξέλιξη και σημασία

Η καταγωγή των strategy games εντοπίζεται σε παραδοσιακά παιχνίδια, όπως το σκάκι και το Go, τα οποία αποτέλεσαν πρότυπα για την ανάπτυξη των πρώτων ψηφιακών στρατηγικών παιχνιδιών κατά τις δεκαετίες του 1970 και 1980. Σήμερα, τα strategy games δεν περιορίζονται μόνο στην ψυχαγωγία, αλλά χρησιμοποιούνται και σε **στρατιωτικές προσομοιώσεις**, **εκπαιδευτικά περιβάλλοντα** και **έρευνα τεχνητής νοημοσύνης**, καθώς αποτελούν ιδανικό πεδίο για την αξιολόγηση αλγορίθμων λήψης αποφάσεων.

Συνοψίζοντας, τα strategy games συνδυάζουν τη λογική με τη δημιουργικότητα, προσφέροντας στους παίκτες ένα πλαίσιο όπου η **σκέψη, η υπομονή και η διορατικότητα** είναι τα βασικά εργαλεία επιτυχίας. Η συνεχής τους εξέλιξη, σε συνδυασμό με την προσαρμογή τους σε νέες τεχνολογίες, τα καθιστά ένα από τα πιο **ανθεκτικά και διαχρονικά είδη** στον χώρο των video games.

2.4 Simulation Games

Τα **Simulation Games** αποτελούν μια ιδιαίτερα σημαντική κατηγορία ηλεκτρονικών παιχνιδιών, καθώς στόχος τους είναι η **αναπαράσταση πραγματικών διαδικασιών, συστημάτων ή εμπειριών** μέσα σε ένα εικονικό περιβάλλον. Σε αντίθεση με άλλα είδη video games που εστιάζουν κυρίως στην αφήγηση, τη δράση ή τη στρατηγική, τα simulation games επιχειρούν να δημιουργήσουν μια **ρεαλιστική προσομοίωση** που δίνει στον παίκτη τη δυνατότητα να αλληλεπιδρά με καταστάσεις παρόμοιες με αυτές του πραγματικού κόσμου [7].

Κατηγορίες Simulation Games

1. **Life Simulation:** Παιχνίδια που εστιάζουν στην προσομοίωση κοινωνικών ή βιολογικών πτυχών της ζωής. Ενδεικτικό παράδειγμα αποτελεί το *The Sims*, το οποίο αναπαριστά την καθημερινότητα των χαρακτήρων, επιτρέποντας στον παίκτη να διαχειρίζεται πτυχές όπως οι σχέσεις, η εργασία και οι δραστηριότητες.
2. **Vehicle Simulation:** Εστιάζουν στην προσομοίωση οδήγησης ή πτήσης, όπως τα *Microsoft Flight Simulator* και *Gran Turismo*. Η ακρίβεια της φυσικής, των μηχανισμών και του περιβάλλοντος αποτελεί κρίσιμο παράγοντα για την επιτυχία τους.
3. **Construction & Management Simulation (CMS):** Παιχνίδια όπου ο παίκτης διαχειρίζεται πόρους, οικονομικά συστήματα και υποδομές. Το *SimCity* και το *RollerCoaster Tycoon* είναι χαρακτηριστικά παραδείγματα.
4. **Medical & Professional Simulation:** Χρησιμοποιούνται όχι μόνο για ψυχαγωγία αλλά και για εκπαίδευση, όπως τα *Surgery Simulator* ή προγράμματα προσομοίωσης στρατιωτικών επιχειρήσεων.

Εκπαιδευτική και επαγγελματική χρήση

Τα simulation games διαδραματίζουν σημαντικό ρόλο και εκτός της ψυχαγωγίας, καθώς έχουν **εκπαιδευτικές και επαγγελματικές εφαρμογές**. Στην αεροπορία, για παράδειγμα, οι πτητικοί προσομοιωτές χρησιμοποιούνται εκτενώς για την εκπαίδευση πιλότων, προσφέροντας ασφαλή και οικονομικά αποδοτική εξάσκηση. Παρομοίως, στον χώρο της ιατρικής, τα simulation

games προσφέρουν την ευκαιρία για εκπαίδευση σε πολύπλοκες διαδικασίες χωρίς να τίθεται σε κίνδυνο η ανθρώπινη ζωή.

Ακαδημαϊκές έρευνες δείχνουν ότι η χρήση προσομοιώσεων στη μάθηση βελτιώνει την **κατανόηση, την κριτική σκέψη και την ικανότητα λήψης αποφάσεων**. Για τον λόγο αυτό, τα simulation games θεωρούνται εργαλεία που συνδυάζουν την **εκπαιδευτική διάσταση με την ψυχαγωγία**, καθιστώντας τα ένα από τα πιο πολυδιάστατα είδη παιχνιδιών [8].

Σχέση με τον ρεαλισμό και την αληθοφάνεια

Κρίσιμο στοιχείο στα simulation games είναι ο βαθμός **ρεαλισμού**. Η αληθοφάνεια των γραφικών, η ακρίβεια της φυσικής και η πιστότητα των μηχανισμών καθορίζουν την εμπειρία του παίκτη. Παράλληλα, τα simulation games δεν περιορίζονται μόνο στην "αντιγραφή" της πραγματικότητας, αλλά συχνά εισάγουν **παραλλαγές και σενάρια** που ενθαρρύνουν τη δημιουργικότητα και τη δοκιμή νέων στρατηγικών από τον παίκτη.

Ιστορική εξέλιξη

Οι πρώτες μορφές simulation games εμφανίστηκαν ήδη από τη δεκαετία του 1960 με απλές αναπαραστάσεις, όπως το *Spacewar!* (1962), ενώ κατά τις δεκαετίες 1980 και 1990 το είδος γνώρισε τεράστια ανάπτυξη με τίτλους όπως το *SimCity* και το *Flight Simulator*. Σήμερα, τα simulation games συνεχίζουν να εξελίσσονται με τη χρήση τεχνολογιών **VR (Virtual Reality)** και **AR (Augmented Reality)**, που ενισχύουν ακόμα περισσότερο την αίσθηση εμβύθισης.

Συνοψίζοντας, τα simulation games αποτελούν ένα είδος που συνδυάζει **ψυχαγωγία, εκπαίδευση και επαγγελματική ανάπτυξη**, με εφαρμογές που ξεπερνούν τα όρια της βιομηχανίας του gaming και εκτείνονται στην πραγματική ζωή. Η εξέλιξή τους αναμένεται να συνεχιστεί δυναμικά, ιδιαίτερα με την πρόοδο της τεχνητής νοημοσύνης και των τεχνολογιών εικονικής πραγματικότητας.

2.5 Sports & Racing Games

Τα **Sports και Racing Games** αποτελούν δύο από τις πιο διαχρονικές και δημοφιλείς κατηγορίες ηλεκτρονικών παιχνιδιών. Βασικός τους στόχος είναι η **αναπαράσταση αθλητικών δραστηριοτήτων ή αγώνων ταχύτητας**, προσφέροντας στον παίκτη την εμπειρία συμμετοχής σε ένα αθλητικό γεγονός ή σε έναν ανταγωνιστικό αγώνα οδήγησης. Σε αντίθεση με τα simulation games που επικεντρώνονται στον ρεαλισμό σε ποικίλα πεδία, τα sports και racing games έχουν πιο εξειδικευμένη εστίαση, δίνοντας έμφαση είτε στην πιστή αναπαράσταση υπαρκτών αθλημάτων είτε στη δημιουργία έντονων εμπειριών ανταγωνισμού και δεξιοτήτων [6].

Sports Games

Τα αθλητικά παιχνίδια καλύπτουν μια ευρεία γκάμα αθλημάτων, όπως ποδόσφαιρο, μπάσκετ, τένις, γκολφ, πυγμαχία κ.ά. Μερικά από τα πλέον χαρακτηριστικά παραδείγματα είναι η σειρά *FIFA* και το *NBA 2K*, που συνδυάζουν ρεαλιστικά γραφικά, προσομοίωση κανόνων και αθλητικού περιβάλλοντος, καθώς και στοιχεία διαχείρισης ομάδων.

Τα sports games δεν περιορίζονται μόνο στην απλή αναπαράσταση του αθλήματος, αλλά ενσωματώνουν και στοιχεία **στρατηγικής και προσομοίωσης**, όπως η διαχείριση μιας ομάδας

(manager mode), η ανάπτυξη παικτών και η λήψη αποφάσεων σε επίπεδο τακτικής. Επιπλέον, η άνοδος του διαδικτυακού gaming έχει ενισχύσει τη **διαδραστικότητα** και τον ανταγωνισμό, καθώς οι παίκτες μπορούν να αγωνίζονται σε παγκόσμιο επίπεδο.

Racing Games

Τα racing games επικεντρώνονται στην οδήγηση οχημάτων, συνήθως σε αγωνιστικές πίστες ή ανοιχτά περιβάλλοντα. Το είδος αυτό περιλαμβάνει διαφορετικές υποκατηγορίες, όπως:

- **Arcade racing games**, τα οποία δίνουν έμφαση στην ταχύτητα και την αδρεναλίνη, με λιγότερο ρεαλιστικούς μηχανισμούς (π.χ. *Need for Speed*).
- **Simulation racing games**, τα οποία προσπαθούν να αναπαραστήσουν με ακρίβεια τις φυσικές δυνάμεις, τον χειρισμό και τη μηχανολογία των οχημάτων (π.χ. *Gran Turismo, Forza Motorsport*).

Η τεχνολογία των racing games έχει εξελιχθεί σημαντικά, φτάνοντας σήμερα σε επίπεδα υψηλού ρεαλισμού μέσω **VR (Virtual Reality)** και **περιφερειακών συσκευών** όπως τιμόνια, πεντάλ και cockpit simulators, που προσφέρουν στον παίκτη μια καθηλωτική εμπειρία.

Κοινωνική και πολιτισμική διάσταση

Η δημοτικότητα των sports και racing games δεν περιορίζεται μόνο στην ψυχαγωγία. Τα παιχνίδια αυτά συμβάλλουν στη διαμόρφωση μιας **ψηφιακής αθλητικής κουλτούρας**, καθώς επιτρέπουν στους παίκτες να συμμετέχουν σε τουρνουά eSports, με κορυφαία παραδείγματα τα **FIFA eWorld Cup** και **Formula 1 Esports Series**. Η ανάπτυξη των eSports έχει αναδείξει τα sports και racing games σε ανταγωνιστικό πεδίο με διεθνή αναγνώριση, οικονομικές απολαβές και παγκόσμια κοινότητα θεατών.

Ιστορική εξέλιξη

Οι ρίζες των sports games μπορούν να ανιχνευθούν ήδη από τη δεκαετία του 1970 με το *Pong* (1972), που προσομοίωνε το τένις, ενώ τα racing games εμφανίστηκαν λίγο αργότερα με το *Gran Trak 10* (1974) και το *Pole Position* (1982). Από τότε, η τεχνολογική πρόοδος οδήγησε σε όλο και πιο **ρεαλιστικά γραφικά, μηχανισμούς φυσικής και τεχνητής νοημοσύνης**, με αποτέλεσμα τη συνεχή άνοδο της ποιότητας και της αλληλεπίδρασης [22].

Συμπέρασμα

Τα **Sports και Racing Games** αποτελούν είδη που συνδυάζουν **δεξιότητες, στρατηγική και ανταγωνισμό**, αντανακλώντας την αθλητική κουλτούρα της κοινωνίας και ενισχύοντας τη δυναμική του eSports. Ο ρόλος τους δεν περιορίζεται απλά στην ψυχαγωγία αλλά επεκτείνεται σε πεδία όπως η **ψηφιακή κοινωνικοποίηση, η εκπαίδευση δεξιοτήτων και η καλλιέργεια πνεύματος ανταγωνισμού και συνεργασίας**. Με την περαιτέρω ανάπτυξη της τεχνολογίας, το μέλλον των sports και racing games αναμένεται να συνδεθεί άρρηκτα με την επαυξημένη και εικονική πραγματικότητα, καθώς και με την εξέλιξη της βιομηχανίας των eSports.

2.6 Adventure Games

Τα **Adventure Games** αποτελούν μια από τις πιο εμβληματικές κατηγορίες ηλεκτρονικών παιχνιδιών, καθώς έθεσαν τις βάσεις για την ανάπτυξη της αφήγησης, της εξερεύνησης και της

αλληλεπίδρασης στον χώρο του ψηφιακού παιχνιδιού. Ο βασικός τους πυρήνας εντοπίζεται στην **εξερεύνηση, την επίλυση γρίφων και την αλληλεπίδραση με τον κόσμο και τους χαρακτήρες** που παρουσιάζονται. Σε αντίθεση με άλλα είδη, τα adventure games δίνουν μεγαλύτερη βαρύτητα στην **ιστορία και την ατμόσφαιρα** παρά στη δράση ή τον ανταγωνισμό.

Χαρακτηριστικά γνωρίσματα

Τα adventure games χαρακτηρίζονται από:

- **Ισχυρή αφήγηση**: η ιστορία αποτελεί τον πυρήνα του παιχνιδιού και καθοδηγεί τον παίκτη μέσα από σενάρια, διάλογους και γεγονότα.
- **Γρίφους και προκλήσεις λογικής**: ο παίκτης καλείται να λύσει προβλήματα που απαιτούν σκέψη και παρατηρητικότητα.
- **Εξερεύνηση**: έμφαση στη διαδραστικότητα με τον κόσμο, μέσω αντικειμένων, χαρακτήρων και περιβαλλόντων.
- **Αλληλεπίδραση με NPCs (Non-Playable Characters)**: διάλογοι και αποφάσεις που επηρεάζουν την πορεία της ιστορίας.

Σε πολλά παιχνίδια, η επιτυχία δεν μετριέται με βάση τη "νίκη" ή την "ήττα", αλλά με το πόσο βαθιά μπορεί ο παίκτης να κατανοήσει και να απολαύσει την αφήγηση.

Ιστορική εξέλιξη

Η ιστορία των adventure games ξεκινά τη δεκαετία του 1970 με τα **text-based adventures**, όπως το *Colossal Cave Adventure* (1976), που βασίζονταν αποκλειστικά σε περιγραφές κειμένου και εντολές του παίκτη. Αργότερα, με την τεχνολογική πρόοδο, εμφανίστηκαν τα **graphic adventures**, με χαρακτηριστικά παραδείγματα τη σειρά *King's Quest* (1984) και τα παιχνίδια της LucasArts όπως *The Secret of Monkey Island* (1990) [22].

Κατά τη δεκαετία του 1990, το είδος έφτασε στο απόγειό του με τίτλους όπως το *Myst* (1993), που συνδύαζε προηγμένα γραφικά με καινοτόμους γρίφους, και το *Gabriel Knight* (1993), που έδωσε ιδιαίτερη έμφαση στην αφήγηση και τη δραματοποίηση.

Στη σύγχρονη εποχή, τα adventure games έχουν ανανεωθεί μέσω **interactive storytelling** και επιλογών που αλλάζουν την εξέλιξη της ιστορίας, με παραδείγματα όπως *Life is Strange* (2015) και *The Walking Dead* (2012) της Telltale Games. Η μετάβαση αυτή ενισχύθηκε από τη διάδοση των κινητών συσκευών και των ψηφιακών πλατφορμών διανομής (Steam, Epic Games Store), που επέτρεψαν σε ανεξάρτητους δημιουργούς να αναπτύξουν νέα αφηγηματικά παιχνίδια.

Κοινωνική και πολιτισμική διάσταση

Τα adventure games έχουν συμβάλει σημαντικά στην ανάπτυξη της **αφηρημένης σκέψης, της δημιουργικότητας και της καλλιέργειας της φαντασίας**. Επιπλέον, συχνά προσεγγίζουν **φιλοσοφικά και ηθικά διλήμματα**, προσφέροντας στον παίκτη τη δυνατότητα να σκεφτεί βαθύτερα για τις πράξεις του και τις συνέπειές τους.

Σε πολιτισμικό επίπεδο, τα adventure games έχουν λειτουργήσει ως **γέφυρα ανάμεσα στη λογοτεχνία, τον κινηματογράφο και το διαδραστικό μέσο**, εισάγοντας έννοιες όπως η "interactive fiction" και επηρεάζοντας την εξέλιξη και άλλων ειδών παιχνιδιών.

Σύγχρονες τάσεις

Σήμερα, τα adventure games τείνουν να συνδυάζονται με άλλα είδη (action-adventure, RPG-adventure), δημιουργώντας υβριδικές εμπειρίες. Παράλληλα, η ανάπτυξη της **VR (Virtual Reality)** τεχνολογίας δίνει νέα διάσταση στην εξερεύνηση και την αφήγηση, καθώς ο παίκτης μπορεί να "βυθιστεί" κυριολεκτικά μέσα στον κόσμο του παιχνιδιού.

Συμπέρασμα

Τα **Adventure Games** αποτελούν θεμελιώδες είδος της βιομηχανίας, καθώς ανέδειξαν τη σημασία της αφήγησης και της εξερεύνησης στον σχεδιασμό παιχνιδιών. Η μακροχρόνια επίδρασή τους είναι εμφανής όχι μόνο στη διαμόρφωση της βιομηχανίας των video games, αλλά και στη σύγκλιση με άλλες μορφές τέχνης και πολιτισμού. Παρά την προσωρινή τους κάμψη κατά τη δεκαετία του 2000, το είδος έχει επανέλθει δυναμικά, με νέες τεχνολογίες και δημιουργικές προσεγγίσεις να εξασφαλίζουν τη συνεχή του εξέλιξη.

Κεφάλαιο 3ο: Τεχνολογίες που Χρησιμοποιούνται για την Ανάπτυξη Video Games

Η ανάπτυξη ηλεκτρονικών παιχνιδιών αποτελεί ένα από τα πιο απαιτητικά και πολυσύνθετα πεδία της πληροφορικής, καθώς συνδυάζει προγραμματισμό, γραφικά, ήχο, δικτύωση και διαδραστικότητα σε ένα ενιαίο σύστημα. Οι τεχνολογίες που αξιοποιούνται για τη δημιουργία ενός παιχνιδιού καθορίζουν τόσο τις τεχνικές του δυνατότητες όσο και την εμπειρία του παίκτη.

Στο κεφάλαιο αυτό παρουσιάζονται συνοπτικά οι βασικοί άξονες που συνθέτουν το τεχνολογικό υπόβαθρο της ανάπτυξης video games: από τις γλώσσες προγραμματισμού που αποτελούν τη βάση του κώδικα, μέχρι τις μηχανές παιχνιδιών που προσφέρουν έτοιμα εργαλεία για τη διαχείριση γραφικών και φυσικής, αλλά και τις σύγχρονες τεχνολογικές τάσεις όπως η εικονική και επαυξημένη πραγματικότητα ή η χρήση τεχνητής νοημοσύνης.

Η παρουσίαση αυτή δεν αποσκοπεί σε εξαντλητική ανάλυση, αλλά σε μια σφαιρική και τεκμηριωμένη εικόνα των βασικών εργαλείων που διαμορφώνουν τη σύγχρονη βιομηχανία παιχνιδιών και παρέχουν το πλαίσιο κατανόησης για την εργασία που ακολουθεί [4].

3.1 Γλώσσες Προγραμματισμού

Η επιλογή γλώσσας προγραμματισμού αποτελεί έναν από τους σημαντικότερους παράγοντες στην ανάπτυξη video games, καθώς επηρεάζει άμεσα την απόδοση, τη φορητότητα, την επεκτασιμότητα και την ευκολία υλοποίησης των μηχανισμών του παιχνιδιού. Οι γλώσσες που χρησιμοποιούνται σε αυτόν τον τομέα καλύπτουν ένα ευρύ φάσμα αναγκών, από την υλοποίηση συστημάτων χαμηλού επιπέδου που σχετίζονται με τη διαχείριση μνήμης και την απόδοση, μέχρι scripting γλώσσες που διευκολύνουν τη δημιουργία σεναρίων και τη γρήγορη ανάπτυξη πρωτοτύπων [16].

Χαμηλού επιπέδου γλώσσες: C και C++

Η **C++** θεωρείται η κυρίαρχη γλώσσα στον χώρο του game development, καθώς χρησιμοποιείται εκτενώς σε εμπορικές μηχανές όπως η **Unreal Engine**. Ο λόγος είναι η δυνατότητα άμεσης πρόσβασης στη μνήμη και η υψηλή απόδοση, που είναι κρίσιμα στοιχεία για τη δημιουργία παιχνιδιών με πολύπλοκα γραφικά και απαιτητικούς μηχανισμούς φυσικής. Η **C**, παλαιότερη αλλά εξίσου σημαντική, χρησιμοποιείται για βασικά συστήματα και game engines που χρειάζονται απόλυτο έλεγχο σε επίπεδο hardware.

Υψηλού επιπέδου γλώσσες: C#, Java και Python

Η **C#** έχει αποκτήσει μεγάλη δημοτικότητα χάρη στη **Unity Engine**, μία από τις πιο διαδεδομένες μηχανές παιχνιδιών παγκοσμίως. Προσφέρει ευκολία στη σύνταξη, υποστηρίζει αντικειμενοστραφή προγραμματισμό και παρέχει έτοιμες βιβλιοθήκες για ανάπτυξη 2D και 3D παιχνιδιών. Η **Java** χρησιμοποιείται κυρίως για mobile game development, ιδιαίτερα σε εφαρμογές Android, καθώς προσφέρει φορητότητα μέσω της Java Virtual Machine (JVM). Η **Python**, παρότι δεν είναι ιδιαίτερα διαδεδομένη σε AAA παραγωγές, αξιοποιείται σε scripting, εργαλεία ανάπτυξης και εκπαιδευτικά παιχνίδια λόγω της απλότητας και της αναγνωσιμότητας του κώδικά της [19]/[18].

Web-based γλώσσες: JavaScript και TypeScript

Η ανάπτυξη browser-based παιχνιδιών βασίζεται κυρίως στη **JavaScript**, συχνά σε συνδυασμό με **HTML5** και **CSS3**, που επιτρέπουν τη δημιουργία διαδραστικών εφαρμογών χωρίς την ανάγκη εγκατάστασης πρόσθετων. Τα τελευταία χρόνια, η **TypeScript**, ως υπερσύνολο της JavaScript με στατικό έλεγχο τύπων, έχει κερδίσει έδαφος στην ανάπτυξη παιχνιδιών μεγάλης κλίμακας, λόγω της καλύτερης διαχείρισης σύνθετων δομών δεδομένων.

Scripting γλώσσες και εξειδικευμένα περιβάλλοντα

Πέραν των κύριων γλωσσών, σημαντικό ρόλο παίζουν και οι **scripting γλώσσες** που ενσωματώνονται σε game engines, όπως η **Lua** (χρησιμοποιείται στη Roblox και σε μηχανές όπως CryEngine) και η **GDScript** της Godot Engine. Οι γλώσσες αυτές διευκολύνουν τον ορισμό συμπεριφορών χαρακτήρων, event triggers και μηχανισμών παιχνιδιού, προσφέροντας ευελιξία στους σχεδιαστές χωρίς να απαιτείται βαθιά γνώση χαμηλού επιπέδου προγραμματισμού.

Συμπέρασμα

Συνολικά, η επιλογή γλώσσας προγραμματισμού εξαρτάται από το είδος του παιχνιδιού, τις τεχνολογικές απαιτήσεις και την πλατφόρμα στόχευσης. Οι **C++ και C#** κυριαρχούν σε επαγγελματικές παραγωγές, η **JavaScript** κυρίως στο web gaming, ενώ οι scripting γλώσσες παρέχουν ευελιξία και ταχύτητα στην ανάπτυξη. Στην πράξη, τα περισσότερα έργα αξιοποιούν συνδυασμό γλωσσών, ώστε να επιτυγχάνεται η βέλτιστη ισορροπία μεταξύ απόδοσης και παραγωγικότητας.

3.2 Game Engines (Μηχανές Παιχνιδιών)

Οι μηχανές παιχνιδιών (game engines) αποτελούν τον πυρήνα της σύγχρονης βιομηχανίας ανάπτυξης video games, καθώς προσφέρουν ένα ολοκληρωμένο περιβάλλον για τη δημιουργία, τη δοκιμή και τη διανομή παιχνιδιών. Μία μηχανή παιχνιδιού παρέχει έτοιμες βιβλιοθήκες και εργαλεία που καλύπτουν κρίσιμες λειτουργίες, όπως η **απεικόνιση γραφικών**, η **διαχείριση φυσικής**, ο **ήχος**, η **τεχνητή νοημοσύνη**, το **multiplayer networking** και η **υποστήριξη πολλαπλών πλατφορμών**. Με αυτόν τον τρόπο, μειώνεται το κόστος και ο χρόνος ανάπτυξης, ενώ οι δημιουργοί μπορούν να επικεντρωθούν στη σχεδίαση του gameplay και της εμπειρίας του χρήστη [23]/[17].

Κύρια χαρακτηριστικά των game engines

Οι σύγχρονες μηχανές παιχνιδιών συνδυάζουν τεχνολογίες που εξυπηρετούν διαφορετικές πτυχές της ανάπτυξης [25]:

- **Graphics Rendering**: Υποστήριξη 2D και 3D γραφικών, συχνά με ενσωμάτωση APIs όπως DirectX, OpenGL και Vulkan.
- **Physics Engines**: Προσομοίωση κίνησης, βαρύτητας και συγκρούσεων με βιβλιοθήκες όπως Havok και PhysX.

- **Scripting Systems**: Ενσωμάτωση scripting γλωσσών (π.χ. Lua, Python, C#) που επιτρέπουν την εύκολη παραμετροποίηση της συμπεριφοράς του παιχνιδιού.
- **Cross-Platform Development**: Δυνατότητα ανάπτυξης παιχνιδιών για διαφορετικές πλατφόρμες (PC, κονσόλες, κινητά).
- **Editor Tools**: Οπτικοί editors που επιτρέπουν σε προγραμματιστές και σχεδιαστές να δημιουργούν assets, επίπεδα και σενάρια χωρίς να γράφουν αποκλειστικά κώδικα.

Δημοφιλείς μηχανές παιχνιδιών

Η βιομηχανία έχει υιοθετήσει συγκεκριμένες μηχανές που καθόρισαν την ανάπτυξη video games τα τελευταία χρόνια [25]:

- **Unreal Engine (Epic Games)**: Μία από τις πιο ισχυρές μηχανές παιχνιδιών, γραμμένη σε C++. Προσφέρει ρεαλιστικά γραφικά μέσω του Unreal Engine 5 και του συστήματος Nanite, καθώς και το Lumen για φωτισμό σε πραγματικό χρόνο. Χρησιμοποιείται τόσο σε AAA τίτλους όσο και σε προσομοιώσεις και κινηματογραφικές παραγωγές.
- **Unity Engine (Unity Technologies)**: Δημοφιλής για την ευκολία χρήσης και την ευρεία κοινότητα υποστήριξης. Βασίζεται κυρίως στη γλώσσα C# και χρησιμοποιείται εκτεταμένα σε indie παραγωγές, mobile games και VR/AR εφαρμογές. Διαθέτει asset store που διευκολύνει την πρόσβαση σε έτοιμα μοντέλα, textures και scripts.
- **Godot Engine**: Ανοιχτού κώδικα μηχανή που έχει αυξηθεί σημαντικά σε δημοτικότητα λόγω της ευελιξίας της. Υποστηρίζει scripting μέσω της GDScript, μιας γλώσσας που μοιάζει με την Python, αλλά και C#. Η φιλοσοφία της Godot δίνει έμφαση στην ελευθερία των δημιουργών χωρίς κόστος αδειοδότησης.
- **CryEngine (Crytek)**: Ισχυρή μηχανή που έγινε γνωστή με τη σειρά *Crysis*, χάρη στη ρεαλιστική απεικόνιση περιβαλλόντων και τον προηγμένο φωτισμό. Παρότι λιγότερο δημοφιλής σήμερα σε σχέση με Unity και Unreal, παραμένει σημαντική για high-end παραγωγές.
- **Custom Engines**: Μεγάλες εταιρείες ανάπτυξης παιχνιδιών συχνά δημιουργούν εσωτερικές μηχανές, ειδικά προσαρμοσμένες στις ανάγκες των έργων τους. Παραδείγματα αποτελούν η **RAGE Engine** της Rockstar Games (*Grand Theft Auto V, Red Dead Redemption 2*) και η **Anvil Engine** της Ubisoft (*Assassin's Creed* series). Αυτές οι μηχανές προσφέρουν εξαιρετική βελτιστοποίηση αλλά απαιτούν υψηλό κόστος συντήρησης [24].

Σημασία των game engines στην εξέλιξη του gaming

Η διάδοση των μηχανών παιχνιδιών συνέβαλε σημαντικά στη **δημοκρατικοποίηση της ανάπτυξης**, δίνοντας τη δυνατότητα ακόμα και σε μικρές ομάδες ή μεμονωμένους δημιουργούς να παράγουν παιχνίδια υψηλής ποιότητας. Παράλληλα, η συνεχής ενσωμάτωση τεχνολογιών όπως **VR/AR**, **machine learning** και **procedural content generation** δείχνει πως οι game engines δεν περιορίζονται πλέον στην κλασική ανάπτυξη παιχνιδιών, αλλά χρησιμοποιούνται σε τομείς όπως η εκπαίδευση, η αρχιτεκτονική και οι κινηματογραφικές παραγωγές.

Συμπέρασμα

Οι μηχανές παιχνιδιών αποτελούν τον θεμέλιο λίθο της σύγχρονης ανάπτυξης video games. Από εμπορικούς τίτλους υψηλών απαιτήσεων έως ανεξάρτητες δημιουργίες, οι game engines καθορίζουν την ποιότητα, την αποδοτικότητα και τις τεχνολογικές δυνατότητες των παιχνιδιών. Η γνώση και η σωστή επιλογή μηχανής αποτελεί κρίσιμο παράγοντα για κάθε project, επηρεάζοντας όχι μόνο την τεχνική υλοποίηση αλλά και τη βιωσιμότητα του έργου στο ανταγωνιστικό περιβάλλον της βιομηχανίας [5].

3.3 Γραφικά και Ήχος

Η εμπειρία του παίκτη σε ένα video game εξαρτάται σε μεγάλο βαθμό από την ποιότητα των γραφικών και του ήχου, οι οποίοι αποτελούν τα βασικά μέσα επικοινωνίας μεταξύ του παιχνιδιού και του χρήστη. Τα γραφικά και ο ήχος δεν περιορίζονται απλώς στην αισθητική, αλλά επηρεάζουν άμεσα τη **ναρατήγηση, την αλληλεπίδραση και τη συναισθηματική εμπλοκή** του παίκτη. Η ανάπτυξή τους απαιτεί συνδυασμό τεχνολογικών γνώσεων και δημιουργικής προσέγγισης, με στόχο την αρμονική ενσωμάτωση στο gameplay [21]/[20].

Γραφικά

Τα γραφικά ενός παιχνιδιού περιλαμβάνουν δύο βασικές κατηγορίες: **2D** και **3D** γραφικά [16].

- **2D γραφικά**: Χρησιμοποιούνται κυρίως σε παιχνίδια πλατφόρμας, puzzle ή κινητών συσκευών. Περιλαμβάνουν sprites, tilesets και vector-based στοιχεία. Η ανάπτυξή τους απαιτεί κατανόηση τεχνικών όπως sprite animation, texture mapping και parallax scrolling.
- **3D γραφικά**: Παρέχουν τρισδιάστατη απεικόνιση κόσμων, χαρακτήρων και αντικειμένων. Η ανάπτυξή τους στηρίζεται σε τεχνολογίες rendering, shaders, lighting και particle systems. Η ρεαλιστική απεικόνιση επιτυγχάνεται με τεχνικές όπως **normal mapping**, **ambient occlusion** και **global illumination**. Οι σύγχρονες μηχανές παιχνιδιών, όπως η Unreal Engine και η Unity, προσφέρουν ενσωματωμένα εργαλεία για τη δημιουργία ρεαλιστικών 3D περιβαλλόντων.

Animation: Η κινούμενη εικόνα αποτελεί κρίσιμο στοιχείο για την αίσθηση ρεαλισμού και την ελκυστικότητα του παιχνιδιού. Χρησιμοποιούνται σκελετικές κινήσεις (rigging), keyframe animation και motion capture για φυσικό αποτέλεσμα.

Εργαλεία γραφικών: Photoshop, Illustrator, Blender, Maya και 3ds Max είναι μερικά από τα πιο διαδεδομένα εργαλεία που επιτρέπουν τη δημιουργία και επεξεργασία assets, textures και animations.

Ήχος

Ο ήχος στο gaming περιλαμβάνει **μουσική, ηχητικά εφέ (SFX)** και **φωνητική καθοδήγηση (voice-over)**, συμβάλλοντας στην ενίσχυση της ατμόσφαιρας, της αλληλεπίδρασης και της ψυχολογικής εμπλοκής του παίκτη .

- **Μουσική**: Προσδιορίζει το συναισθηματικό πλαίσιο και υπογραμμίζει γεγονότα ή σκηνές. Η δυναμική ή adaptive μουσική προσαρμόζεται σε πραγματικό χρόνο στις ενέργειες του παίκτη, αυξάνοντας την εμβύθιση.
- **Ηχητικά εφέ**: Καλύπτουν κάθε γεγονός στο παιχνίδι, από βήματα και όπλα έως φυσικά φαινόμενα. Η σωστή διαχείριση της χωρικής ακουστικής (3D sound positioning) ενισχύει την αίσθηση παρουσίας στον ψηφιακό κόσμο.
- **Φωνητική καθοδήγηση**: Χρησιμοποιείται για διάλογους χαρακτήρων ή ειδοποιήσεις, βελτιώνοντας την αλληλεπίδραση και την αφήγηση. Η επεξεργασία φωνής και η ενσωμάτωση σε μηχανές ήχου όπως FMOD ή Wwise είναι σημαντική για την ομαλή λειτουργία στο gameplay.

Η αποτελεσματική ενσωμάτωση γραφικών και ήχου απαιτεί εργαλεία και τεχνολογίες που συνδέουν assets με τον κώδικα του παιχνιδιού:

- **Rendering Pipelines**: Συστήματα που διαχειρίζονται την απεικόνιση γραφικών σε πραγματικό χρόνο.
- **Audio Middleware**: Λογισμικά όπως FMOD ή Wwise που παρέχουν δυναμική μίξη ήχου, spatial audio και συγχρονισμό με γεγονότα του παιχνιδιού.
- **Optimization Techniques**: Level of Detail (LOD), occlusion culling και compression formats για βελτίωση απόδοσης και μείωση απαιτήσεων μνήμης και επεξεργαστή.

Συμπέρασμα

Η αρμονική συνύπαρξη γραφικών και ήχου αποτελεί καθοριστικό παράγοντα για την επιτυχία ενός video game. Η υψηλή ποιότητα των assets, σε συνδυασμό με την ορθολογική χρήση των τεχνολογιών rendering και audio middleware, επηρεάζει την αισθητική, την αλληλεπίδραση και τη συναισθηματική εμπλοκή του παίκτη. Κατά συνέπεια, η γνώση αυτών των στοιχείων είναι απαραίτητη για την αποτελεσματική ανάπτυξη και τον σχεδιασμό παιχνιδιών.

3.4 Δικτύωση και Multiplayer

Η δικτύωση στα video games αποτελεί κρίσιμο στοιχείο για τη δημιουργία πολυπαικτικών εμπειριών (multiplayer), επιτρέποντας σε παίκτες από διαφορετικές τοποθεσίες να αλληλεπιδρούν σε πραγματικό χρόνο. Η υλοποίηση multiplayer απαιτεί συνδυασμό τεχνικών γνώσεων σε **δικτύωση, συγχρονισμό δεδομένων, latency management** και **ασφάλεια**, προκειμένου να διασφαλιστεί η ομαλή και δίκαιη εμπειρία για όλους τους συμμετέχοντες.

Τύποι Multiplayer

Τα παιχνίδια μπορούν να υλοποιηθούν σε διάφορα σενάρια multiplayer:

- **Local Multiplayer**: Παίκτες χρησιμοποιούν την ίδια συσκευή ή κονσόλα (split-screen ή shared-screen). Η υλοποίηση είναι σχετικά απλή, καθώς δεν απαιτεί σύνθετη δικτύωση, αλλά μπορεί να περιορίζεται από το hardware και την οθόνη.
- **Online Multiplayer**: Επιτρέπει σε παίκτες να συνδεθούν μέσω διαδικτύου, χρησιμοποιώντας servers ή peer-to-peer (P2P) μοντέλα. Οι κύριες αρχιτεκτονικές περιλαμβάνουν:
    - **Client-Server Architecture**: Όλοι οι παίκτες συνδέονται σε έναν κεντρικό server που διαχειρίζεται τη λογική του παιχνιδιού και συγχρονίζει τα δεδομένα. Προσφέρει καλύτερο έλεγχο, μειωμένη απάτη (cheating) και σταθερότητα.
    - **Peer-to-Peer (P2P)**: Οι πελάτες επικοινωνούν άμεσα μεταξύ τους χωρίς κεντρικό server. Μειώνει το κόστος server αλλά αυξάνει την πολυπλοκότητα στη διαχείριση συγχρονισμού και ασφαλείας.

Δικτυακή Αρχιτεκτονική και Συγχρονισμός

Η αποτελεσματική δικτύωση απαιτεί:

- **Συγχρονισμό κατάστασης (State Synchronization)**: Τα δεδομένα του παιχνιδιού πρέπει να παραμένουν συνεπή μεταξύ όλων των πελατών. Αυτό γίνεται μέσω τεχνικών όπως **client-side prediction**, **server reconciliation** και **lag compensation**.
- **Latency Management**: Η καθυστέρηση (latency) και η απώλεια πακέτων (packet loss) μπορούν να επηρεάσουν αρνητικά την εμπειρία. Τεχνικές όπως **interpolation**, **extrapolation** και **network smoothing** χρησιμοποιούνται για την αντιστάθμιση των καθυστερήσεων.
- **Data Serialization & Compression**: Τα δεδομένα που μεταφέρονται πρέπει να είναι βελτιστοποιημένα για μικρότερο bandwidth και γρηγορότερη ανταπόκριση. Τυπικά χρησιμοποιούνται μορφές όπως JSON, Protocol Buffers ή custom binary formats.

Τεχνολογίες και Πλατφόρμες

Η ανάπτυξη multiplayer παιχνιδιών βασίζεται σε διάφορα εργαλεία και τεχνολογίες [5]:

- **Networking Libraries**: Socket.IO, WebRTC, Photon, Mirror (Unity) παρέχουν έτοιμες λύσεις για real-time επικοινωνία.
- **Servers & Cloud**: Amazon GameLift, Microsoft Azure PlayFab και Google Cloud προσφέρουν scalable server hosting για online παιχνίδια μεγάλης κλίμακας.
- **Security Mechanisms**: Αυθεντικοποίηση χρηστών, κρυπτογράφηση πακέτων και anti-cheat συστήματα είναι απαραίτητα για την προστασία των παικτών και τη διατήρηση της ακεραιότητας του παιχνιδιού.

Προκλήσεις στην Ανάπτυξη Multiplayer

Η ανάπτυξη multiplayer παιχνιδιών αντιμετωπίζει αρκετές τεχνικές και σχεδιαστικές προκλήσεις [15]:

- **Διαχείριση καθυστερήσεων και συγχρονισμού** για τη διατήρηση της ομαλής εμπειρίας.
- **Κλίμακα και επιβάρυνση server** όταν πολλοί παίκτες συμμετέχουν ταυτόχρονα.
- **Ασφάλεια και προστασία από απάτες (cheating)**, ιδιαίτερα σε ανταγωνιστικά περιβάλλοντα.
- **Cross-platform συμβατότητα**, που απαιτεί συντονισμό διαφορετικών συσκευών και λειτουργικών συστημάτων.

Συμπέρασμα

Η δικτύωση και το multiplayer αποτελούν κρίσιμο παράγοντα για τη μοντέρνα εμπειρία gaming, επηρεάζοντας τόσο την τεχνική υλοποίηση όσο και την κοινωνική διάσταση των παιχνιδιών. Η σωστή διαχείριση latency, συγχρονισμού και ασφάλειας είναι καθοριστική για την επιτυχία και την αποδοχή ενός online παιχνιδιού. Η χρήση σύγχρονων εργαλείων και πλατφορμών επιτρέπει την ανάπτυξη σταθερών, ασφαλών και επεκτάσιμων multiplayer εμπειριών.

3.5 Σύγχρονες Τάσεις

Η βιομηχανία των video games εξελίσσεται ραγδαία, με τεχνολογικές καινοτομίες που επηρεάζουν τον τρόπο ανάπτυξης, διανομής και εμπειρίας των παιχνιδιών. Οι σύγχρονες τάσεις περιλαμβάνουν τεχνολογίες που αυξάνουν την αλληλεπίδραση των παικτών, βελτιώνουν την οπτική και ακουστική εμπειρία, και διευρύνουν το κοινωνικό και οικονομικό περιβάλλον των παιχνιδιών.

Cloud Gaming

Το cloud gaming επιτρέπει στους χρήστες να παίζουν απαιτητικά παιχνίδια χωρίς ισχυρό hardware, αφού η επεξεργασία και το rendering πραγματοποιούνται σε απομακρυσμένους servers. Οι παίκτες λαμβάνουν το βίντεο του παιχνιδιού μέσω internet, ενώ οι εντολές τους μεταδίδονται σε πραγματικό χρόνο στον server. Οφέλη περιλαμβάνουν:

- Απεξάρτηση από ισχυρές κονσόλες ή υπολογιστές.
- Άμεση πρόσβαση σε νέους τίτλους χωρίς εγκατάσταση.
- Δυνατότητα cross-platform gaming.

Εικονική και Επαυξημένη Πραγματικότητα (VR/AR)

Η VR και η AR εισάγουν νέες διαστάσεις στην εμπειρία παιχνιδιού:

- **VR (Virtual Reality)**: Παίκτες βυθίζονται σε πλήρως ψηφιακά περιβάλλοντα με χρήση headset και αισθητήρων κίνησης. Απαιτεί προηγμένα rendering techniques και βελτιστοποίηση για μειωμένο latency.
- **AR (Augmented Reality)**: Ενσωματώνει ψηφιακά αντικείμενα στο πραγματικό περιβάλλον μέσω smartphones ή AR glasses, όπως στο δημοφιλές *Pokémon Go*.

Τεχνητή Νοημοσύνη (AI) και Μηχανική Μάθηση

Η AI χρησιμοποιείται για:

- Προγραμματισμό ευφυών NPCs (non-player characters) που προσαρμόζουν τη συμπεριφορά τους στους παίκτες.
- Δημιουργία procedural περιεχομένου, όπως επίπεδα ή αποστολές, μειώνοντας τον χρόνο ανάπτυξης.
- Ανάλυση δεδομένων χρηστών για personalization και βελτίωση εμπειρίας.

Κοινωνικές και Πολιτισμικές Τάσεις

Τα σύγχρονα παιχνίδια ενσωματώνουν κοινωνικά χαρακτηριστικά:

- Multiplayer και co-op modes για διαδραστικότητα και συνεργασία.
- Integration με social media για κοινοποίηση επιτευγμάτων και συμμετοχή σε κοινότητες.
- E-sports, που έχουν δημιουργήσει επαγγελματικό και οικονομικό οικοσύστημα γύρω από τα παιχνίδια.

Mobile Gaming και Cross-Platform Εμπειρία

Η αύξηση των smartphones και tablets έχει αλλάξει το τοπίο του gaming:

- Τα mobile games συνήθως βασίζονται σε μικρές συνεδρίες, free-to-play μοντέλα με in-app purchases.
- Η cross-platform ανάπτυξη επιτρέπει στον ίδιο τίτλο να λειτουργεί σε PC, κονσόλα και κινητά, διευρύνοντας το κοινό.

Συμπέρασμα

Οι σύγχρονες τάσεις αναδεικνύουν την πολυπλοκότητα και τη δυναμική του σύγχρονου gaming. Cloud gaming, VR/AR, AI, κοινωνικές διασυνδέσεις και cross-platform ανάπτυξη δημιουργούν μια εμπειρία παιχνιδιού που δεν περιορίζεται σε μία συσκευή ή έναν τύπο παιχνιδιού. Οι τεχνολογικές εξελίξεις επηρεάζουν άμεσα την ανάπτυξη, το design και τη διάδραση των παικτών, καθιστώντας τα παιχνίδια έναν σημαντικό παράγοντα τεχνολογικής και πολιτισμικής καινοτομίας.

Κεφάλαιο 4ο: Τεχνολογικό υπόβαθρο

Η υλοποίηση του παιχνιδιού βασίστηκε σε έναν πυρήνα τεχνολογιών του **web stack** — HTML, CSS και JavaScript — χωρίς τη χρήση εξωτερικών βιβλιοθηκών ή framework. Αυτή η επιλογή ήταν συνειδητή, με σκοπό την πλήρη κατανόηση και έλεγχο όλων των επιπέδων του κώδικα, καθώς και την εστίαση στην εκπαιδευτική αξία της ανάπτυξης ενός παιχνιδιού από την αρχή (from scratch).

Η προσέγγιση που ακολουθήθηκε καθιστά το έργο ένα χαρακτηριστικό παράδειγμα **Vanilla JavaScript εφαρμογής**, στην οποία η διαχείριση του DOM, η μοντελοποίηση της κατάστασης (state), οι μηχανισμοί ελέγχου ροής και οι μηχανισμοί απόκρισης σε ενέργειες του χρήστη υλοποιούνται αποκλειστικά μέσω "καθαρής" JavaScript. Το έργο είναι δομημένο με τρόπο που θυμίζει **Single Page Application (SPA)**, χωρίς όμως να βασίζεται σε βιβλιοθήκες όπως React ή Vue.

Η παρούσα ενότητα αναλύει τις τεχνολογικές επιλογές του έργου και εξηγεί πώς αυτές αξιοποιήθηκαν για να δημιουργηθεί ένα πλήρως λειτουργικό turn-based παιχνίδι περιπέτειας στο περιβάλλον του browser.

4.1 Επιλογή γλωσσών (HTML, CSS, JavaScript)

Η επιλογή των τεχνολογιών HTML, CSS και JavaScript για την ανάπτυξη του παιχνιδιού δεν ήταν τυχαία, αλλά βασίστηκε στην ευρεία αποδοχή, υποστήριξη και καταλληλότητα αυτών των τεχνολογιών για εφαρμογές στο περιβάλλον του φυλλομετρητή (browser). Πρόκειται για τον λεγόμενο **"πυρήνα του ιστού"** (core web stack), ο οποίος επιτρέπει τη δημιουργία δυναμικών, διαδραστικών και οπτικά ελκυστικών εφαρμογών χωρίς την ανάγκη πρόσθετων plugins ή εγκαταστάσεων από τον χρήστη.

4.1.1 HTML (HyperText Markup Language)

Η **HTML (HyperText Markup Language)** αποτελεί τον θεμέλιο λίθο κάθε εφαρμογής που εκτελείται σε περιβάλλον φυλλομετρητή (browser), καθώς είναι υπεύθυνη για την **οργάνωση, δόμηση και περιγραφή του περιεχομένου** που εμφανίζεται στον χρήστη. Στην περίπτωση της συγκεκριμένης εφαρμογής, η HTML χρησιμοποιήθηκε ως ο βασικός σκελετός πάνω στον οποίο χτίστηκε όλη η εμπειρία του παιχνιδιού.

Δόμηση και Ιεραρχία του Περιεχομένου

Η HTML επιτρέπει στον προγραμματιστή να ορίσει με σαφήνεια τη δομή της σελίδας, χωρίζοντας τα επιμέρους στοιχεία σε νοητά και λογικά τμήματα. Για ένα παιχνίδι βασισμένο στον browser, αυτό μεταφράζεται σε ξεκάθαρη διάκριση ανάμεσα σε:

- Περιοχές gameplay (όπου προβάλλεται η δράση του παιχνιδιού, όπως μέσω του <canvas> ή περιοχών με animation),
- Διαδραστικά στοιχεία UI (όπως μενού επιλογών, κουμπιά, μπαρ υγείας, πλαίσια διαλόγου),

- Πληροφοριακές ενότητες (π.χ. τρέχουσα κατάσταση του χαρακτήρα, διαθέσιμα αντικείμενα κ.ά.).

Η χρήση HTML tags όπως:

- για δομικά μπλοκ,
- <img> για απεικόνιση χαρακτήρων και αντικειμένων,
- <button> για επιλογές του χρήστη κατά τη διάρκεια των μαχών,
- <canvas> για animation ή rendering γραφικών (εάν χρησιμοποιηθεί),
  παρείχε όλα τα απαραίτητα εργαλεία για την **οπτική αποτύπωση του παιχνιδιού**, χωρίς να απαιτείται κάποια εξωτερική τεχνολογία ή game engine.

Αναγνωσιμότητα και Συντηρησιμότητα

Η HTML είναι μια **δηλωτική (declarative) γλώσσα**. Αυτό σημαίνει πως ο προγραμματιστής περιγράφει το «τι» θέλει να εμφανίσει και όχι το «πώς» θα υλοποιηθεί αυτό βήμα-βήμα (όπως σε προγραμματιστικές γλώσσες τύπου imperative). Αυτή η φύση της HTML καθιστά τον κώδικα πιο:

- **Αναγνώσιμο**: είναι ευκολότερο να εντοπιστούν τα μέρη της εφαρμογής που αφορούν κάθε λειτουργία.
- **Συντηρήσιμο**: αλλαγές ή βελτιώσεις στη δομή μπορούν να εφαρμοστούν με μικρές, τοπικές παρεμβάσεις χωρίς να επηρεάζεται η συνολική λειτουργικότητα.

Αυτό είναι εξαιρετικά σημαντικό για έργα όπως αυτό, όπου η ευελιξία και η ταχύτητα ανάπτυξης είναι κρίσιμες.

Συμβατότητα και Προσβασιμότητα

Η HTML έχει σχεδιαστεί ώστε να είναι **πλήρως συμβατή με όλους τους σύγχρονους browsers**, και υποστηρίζεται από εργαλεία ανάπτυξης και ανάλυσης, όπως τα Developer Tools των Chrome, Firefox, Safari και Edge. Επιπλέον, η σωστή χρήση semantic tags (όπως <section>, <header>, <main>, <nav> – αν και δεν απαιτήθηκαν σε αυτή τη βασική υλοποίηση) ενισχύει την **προσβασιμότητα** της εφαρμογής και τη δυνατότητα χρήσης της από άτομα με αναπηρίες ή διαφορετικές συσκευές (screen readers, tablets, κινητά).

Όπως αναφέρει και το W3C:

«Η HTML επιτρέπει στους προγραμματιστές να εκφράζουν τη σημασία και τη δομή του περιεχομένου, ενισχύοντας τη δυνατότητα πρόσβασης και επαναχρησιμοποίησης»
– *W3C, 2023*

Χρήση στο Παιχνίδι

Στο πλαίσιο της παρούσας εργασίας, η HTML αποτέλεσε:

- Τη βάση για τον αρχικό φορτωτή του παιχνιδιού (εισαγωγική οθόνη),

- Τη «σκαλωσιά» πάνω στην οποία εμφωλεύουν όλα τα συστήματα διεπαφής με τον χρήστη (UI),
- Το μέσο για την εμφάνιση διαλόγων (με HTML containers),
- Τη βάση πάνω στην οποία συνδέθηκαν δυναμικά τα στοιχεία μέσω DOM manipulation (μέσω JavaScript),
- Την υποδομή για προσαρμογές και responsive σχεδιασμό (αν χρειαστεί μελλοντικά).

Η HTML λειτούργησε, επομένως, ως το θεμέλιο όλων των οπτικών και λειτουργικών στοιχείων της εφαρμογής, επιτρέποντας με **απλό και δομημένο τρόπο** την ανάπτυξη ενός λειτουργικού περιβάλλοντος παιχνιδιού στον φυλλομετρητή.

4.1.2 CSS (Cascading Style Sheets)

Η **CSS (Cascading Style Sheets)** αποτελεί μια από τις θεμελιώδεις τεχνολογίες του Παγκόσμιου Ιστού, υπεύθυνη για τη **μορφοποίηση, αισθητική και διάταξη** του περιεχομένου που περιγράφεται μέσω HTML. Η κύρια λειτουργία της είναι να αποσυνδέσει την **παρουσίαση** από τη **δομή**, δίνοντας στους προγραμματιστές τη δυνατότητα να διαχειριστούν ανεξάρτητα την εμφάνιση μιας εφαρμογής χωρίς να επεμβαίνουν στη λογική ή στη λειτουργικότητα.

Ρόλος της CSS στο παιχνίδι

Στο πλαίσιο του παρόντος browser-based παιχνιδιού, η CSS διαδραμάτισε σημαντικό ρόλο σε πολλαπλά επίπεδα:

- **Διαμόρφωση του περιβάλλοντος παιχνιδιού (game layout):** Ο καθορισμός του κεντρικού καμβά (game container) και η στοίχιση του στο κέντρο της σελίδας έγιναν μέσω CSS, διασφαλίζοντας ότι το παιχνίδι παραμένει ευανάγνωστο και αισθητικά ευχάριστο σε διάφορες αναλύσεις οθονών.
- **Διακόσμηση του UI (User Interface):** Χρησιμοποιήθηκαν κανόνες CSS για την εμφάνιση των **κουμπιών**, **μενού επιλογών**, **παραθύρων διαλόγου**, **μπαρ ζωής (HP bars)**, **εικονιδίων** και άλλων βασικών λειτουργικών στοιχείων. Η αισθητική σχεδίαση των αντικειμένων αυτών είναι κρίσιμη για την εμπειρία χρήστη (UX), ειδικά σε turn-based περιβάλλοντα.
- **Χρώματα και Τυπογραφία:** Ο ορισμός χρωμάτων για την απόδοση τύπων πίτσας (π.χ. spicy, veggie, fungi) και η χρήση τυπογραφικών χαρακτηριστικών (γραμματοσειρές, μεγέθη, σκιές) βοήθησαν στην ενίσχυση της ατμόσφαιρας του παιχνιδιού.
- **Τοποθέτηση και ευθυγράμμιση (layouting):** Μέσω της CSS χρησιμοποιήθηκαν μοντέλα διάταξης όπως **flexbox** και **absolute positioning** ώστε να επιτευχθεί η σωστή τοποθέτηση των αντικειμένων εντός του παιχνιδιού (π.χ. κουμπιά μάχης στο κάτω μέρος, διάλογοι στο πάνω μέρος, κεντρικοί χαρακτήρες στο μέσο της οθόνης κτλ.).

Animations και Keyframes

Αν και το μεγαλύτερο μέρος της κίνησης στο παιχνίδι προγραμματίστηκε μέσω **JavaScript**, η CSS συνέβαλε επίσης στον οπτικό εμπλουτισμό του παιχνιδιού με τη χρήση εφέ όπως:

- Fade-in/Fade-out εναλλαγές παραθύρων ή στοιχείων,
- Μετακινήσεις (transform, translateX/Y) χαρακτήρων ή αντικειμένων,
- Παλλόμενες κινήσεις (π.χ. @keyframes shake) για την υπογράμμιση συγκεκριμένων συμβάντων (όπως ένα χτύπημα ή επιλογή).

Αυτές οι **CSS animations** προσφέρουν απλές αλλά αποτελεσματικές λύσεις για τη βελτίωση της οπτικής ροής, χωρίς την ανάγκη για πολύπλοκο λογισμικό animation.

Responsive Προσαρμογή

Αν και το συγκεκριμένο παιχνίδι έχει πρωτίστως σχεδιαστεί για desktop περιβάλλοντα, η χρήση CSS κανόνων **responsive σχεδιασμού** (όπως media queries) επιτρέπει μελλοντικά την εύκολη προσαρμογή του σε μικρότερες συσκευές, όπως tablets ή smartphones. Η δυνατότητα αυτή είναι ιδιαίτερα σημαντική στην εποχή της πολυσυσκευής και της φορητότητας.

Οφέλη από τη χρήση CSS

Η CSS, ως τεχνολογία, προσφέρει πολυάριθμα πλεονεκτήματα:

- **Επαναχρησιμοποίηση στυλ** μέσω class-based κανόνων,
- **Σαφής διαχωρισμός των αρμοδιοτήτων** μεταξύ προγραμματισμού και αισθητικής (σύμφωνα με την αρχή *Separation of Concerns*),
- **Καλύτερη συντήρηση** και ευκολότερες αλλαγές σε επίπεδο εμφάνισης χωρίς επεμβάσεις στη JavaScript ή στην HTML,
- **Υψηλή απόδοση και συμβατότητα** με όλους τους μοντέρνους browsers.

Η προσέγγιση αυτή είναι σύμφωνη με τις πρακτικές της μοντέρνας ανάπτυξης εφαρμογών, όπου η HTML χρησιμοποιείται για τη δομή, η CSS για τη μορφή και η JavaScript για τη λογική και τη συμπεριφορά.

4.1.3 JavaScript (Vanilla JS)

Η **JavaScript** αποτέλεσε τον ακρογωνιαίο λίθο για τη λειτουργικότητα και τη δυναμική συμπεριφορά του παιχνιδιού. Πρόκειται για μια **δυναμική, αντικειμενοστραφή, event-driven γλώσσα προγραμματισμού** που υποστηρίζεται εγγενώς από όλους τους σύγχρονους browsers. Στην παρούσα εργασία, χρησιμοποιήθηκε **καθαρή JavaScript (Vanilla JS)**, δηλαδή χωρίς τη μεσολάβηση εξωτερικών βιβλιοθηκών ή frameworks, προσφέροντας πλήρη έλεγχο στον προγραμματιστή και ενισχύοντας τη βαθύτερη κατανόηση της λειτουργίας του κώδικα.

Κύριοι ρόλοι της JavaScript στο παιχνίδι

Η JavaScript αξιοποιήθηκε σε πολλαπλά επίπεδα της ανάπτυξης του παιχνιδιού:

- **Διαχείριση αντικειμένων παιχνιδιού (game entities):**
  Μέσω κλάσεων όπως Combatant, Battle, Team, ReplacementMenu κ.ά., ορίζονται τα δυναμικά στοιχεία του παιχνιδιού, όπως οι πίτσες (ήρωες και εχθροί), οι κινήσεις (actions), τα status effects και τα αντικείμενα (items). Αυτή η δομή επέτρεψε τη

δημιουργία ενός μοντέλου βασισμένου σε **αντικειμενοστραφή προγραμματισμό (OOP)**, το οποίο είναι επεκτάσιμο και ευανάγνωστο.

- **Υλοποίηση του game loop:**
  Ένα από τα πιο κρίσιμα σημεία κάθε διαδραστικού παιχνιδιού είναι ο **βρόχος παιχνιδιού (game loop)** — ένας επαναλαμβανόμενος κύκλος που ελέγχει την εναλλαγή φάσεων (παίκτης–αντίπαλος), την κατάσταση των μονάδων, την απόδοση κινήσεων, τις καθυστερήσεις μεταξύ ενεργειών και την εμφάνιση των αποτελεσμάτων στην οθόνη. Στην υλοποίηση αυτή, χρησιμοποιήθηκε μια custom κλάση TurnCycle που ελέγχει την ακολουθία των ενεργειών και την προτεραιότητα (π.χ., ποιος επιτίθεται πρώτος).

- **Ανταπόκριση σε αλληλεπιδράσεις χρήστη (event handling):**
  Ο χειρισμός γεγονότων, όπως το πάτημα πλήκτρων (keydown) και το κλικ κουμπιών (click), είναι βασικό στοιχείο της εμπειρίας χρήστη. Χάρη στη JavaScript, η εφαρμογή μπορεί να ανταποκρίνεται δυναμικά στις επιλογές του παίκτη, να εμφανίζει μενού, να αλλάζει οθόνες και να εκτελεί τις επιλεγμένες ενέργειες σε πραγματικό χρόνο.

- **Custom Events και επικοινωνία μεταξύ modules:**
  Η χρήση προσαρμοσμένων συμβάντων (π.χ. utils.emitEvent("LineupChanged")) διευκόλυνε τη **χαλαρή σύζευξη** μεταξύ των ενοτήτων. Έτσι, για παράδειγμα, μια αλλαγή στο lineup του παίκτη δεν απαιτεί άμεση γνώση του πού και πώς θα αποδοθεί στην οθόνη — το συμβάν ανακοινώνεται, και όποιος έχει ενδιαφέρον για αυτό το γεγονός μπορεί να το "ακούσει" και να δράσει ανάλογα (observer pattern).

Γιατί επιλέχθηκε Vanilla JS;

Η απόφαση να χρησιμοποιηθεί **Vanilla JavaScript**, αντί για ένα framework όπως **React**, **Vue** ή **Svelte**, βασίστηκε σε εκπαιδευτικά και τεχνικά κριτήρια:

- **Ελαχιστοποίηση εξαρτήσεων (dependencies):**
  Δεν απαιτείται κανένα build tool (webpack, vite), κανένα npm package, ούτε runtime μεταγλωττιστές. Το αποτέλεσμα είναι ένα απλό, φορητό και διαφανές project που λειτουργεί με απλό άνοιγμα αρχείου HTML.

- **Ανεξαρτησία από προκαθορισμένες αρχιτεκτονικές:**
  Ενώ τα frameworks παρέχουν σαφείς οδηγίες και δομές, η χρήση Vanilla JS επέτρεψε τον πλήρη **έλεγχο της αρχιτεκτονικής**, κάτι που ενίσχυσε την κατανόηση θεμελιωδών εννοιών όπως:
  - rendering χειροκίνητων UI στοιχείων,
  - state management με αντικείμενα JavaScript,
  - routing και οθόνες χωρίς χρήση router.

- **Εκπαιδευτικός χαρακτήρας:**
  Η υλοποίηση του παιχνιδιού εξ αρχής με Vanilla JS απαιτεί από τον δημιουργό να κατανοήσει και να υλοποιήσει από την αρχή μηχανισμούς όπως:

- DOM Manipulation,
- queue-based event cycles,
- εσωτερική διαχείριση κατάστασης (state),
- συναρτήσεις υψηλότερης τάξης,
- καθυστέρηση εκτέλεσης (async/await, setTimeout).

Αυτός ο τρόπος προσέγγισης δεν είναι μόνο λειτουργικός αλλά και **ιδιαίτερα διδακτικός**, καθιστώντας την εργασία κατάλληλη για παρουσίαση τόσο τεχνικών γνώσεων όσο και αρχιτεκτονικών επιλογών.

4.2 DOM Manipulation και Vanilla JS

Η **διαχείριση του DOM (Document Object Model)** αποτελεί μία από τις σημαντικότερες πτυχές κάθε εφαρμογής που υλοποιείται σε περιβάλλον browser. Το DOM λειτουργεί ως ενδιάμεσος μεταξύ του HTML εγγράφου και της JavaScript, επιτρέποντας την αλληλεπίδραση του κώδικα με τα στοιχεία της σελίδας. Μέσω της **Vanilla JavaScript** — δηλαδή της χρήσης καθαρής JS χωρίς βιβλιοθήκες όπως jQuery ή React — επιτυγχάνεται πλήρης έλεγχος στην τροποποίηση, δημιουργία και διαγραφή στοιχείων DOM με μηδενικό εξωτερικό κόστος.

Τι είναι το DOM;

Το **DOM** είναι μια δομή δεδομένων σε μορφή δέντρου (tree structure), στην οποία κάθε κόμβος (node) αντιστοιχεί σε ένα HTML στοιχείο. Η JavaScript επιτρέπει την πρόσβαση και επεξεργασία αυτών των κόμβων σε πραγματικό χρόνο. Για παράδειγμα:

```
const button = document.querySelector("#startButton");
button.textContent = "Ξεκινάμε!";
```

Στο παραπάνω παράδειγμα, ο προγραμματιστής εντοπίζει ένα κουμπί (<button id="startButton">) και αλλάζει το περιεχόμενο του. Αυτή είναι μία απλή αλλά ισχυρή μορφή DOM manipulation.

Χρήση DOM στην υλοποίηση του παιχνιδιού

Η κατασκευή του παιχνιδιού βασίστηκε έντονα στη χειροκίνητη δημιουργία και διαχείριση στοιχείων DOM. Κάποιες χαρακτηριστικές περιπτώσεις περιλαμβάνουν:

1. Δημιουργία UI components μέσω JavaScript

Αντί να υπάρχουν όλα τα γραφικά στοιχεία προκαθορισμένα στο HTML, δημιουργούνται δυναμικά με JavaScript. Για παράδειγμα, η KeyboardMenu δημιουργεί κουμπιά επιλογών με βάση το context του παιχνιδιού (τις διαθέσιμες κινήσεις):

```
const button = document.createElement("button");
button.innerText = option.label;
```

button.addEventListener("click", option.handler);

container.appendChild(button);

Η παραπάνω προσέγγιση είναι κρίσιμη για την ευελιξία της εφαρμογής, καθώς κάθε μάχη ή αντικείμενο έχει διαφορετικό UI που πρέπει να αποδίδεται δυναμικά.

2. Αντιδραστικότητα με event listeners

Η χρήση addEventListener() επιτρέπει στο παιχνίδι να ανταποκρίνεται σε ενέργειες του χρήστη, όπως κλικ, πάτημα πλήκτρων ή hover. Παράδειγμα από το σύστημα επιλογών:

document.addEventListener("keydown", (e) => {

if (e.key === "ArrowUp") this.navigate(-1);

});

Αυτό επιτρέπει στο σύστημα μενού να υποστηρίζει **πληκτρολογική πλοήγηση**, βασική λειτουργικότητα για turn-based RPG περιβάλλοντα.

3. Ενημέρωση της διεπαφής (UI) με βάση το state

Το παιχνίδι συντηρεί εσωτερική κατάσταση (π.χ. ποιες πίτσες είναι ζωντανές, τι ζωή έχουν, ποια είναι η τρέχουσα φάση) και αποτυπώνει αυτά τα δεδομένα στην οθόνη μέσω DOM manipulation. Παράδειγμα:

this.hpFills.forEach(el => {

el.style.width = `${percent}%`;

});

Αυτό αλλάζει το μήκος των health bars με βάση το υπόλοιπο HP, προσφέροντας στον παίκτη οπτική ανατροφοδότηση σε πραγματικό χρόνο.

Πλεονεκτήματα της χρήσης Vanilla JS για DOM

Η χρήση καθαρής JavaScript για την αλληλεπίδραση με το DOM προσφέρει:

- **Εκπαίδευση και διαφάνεια:** Ο προγραμματιστής κατανοεί πλήρως τον τρόπο λειτουργίας του UI και της ροής δεδομένων, χωρίς "μαύρα κουτιά".
- **Ευελιξία:** Δεν υπάρχουν περιορισμοί από abstractions ή state management libraries. Ο προγραμματιστής αποφασίζει πώς να χειριστεί το rendering και τις μεταβολές του DOM.
- **Απόδοση:** Το DOM ενημερώνεται μόνο όταν και όπως το καθορίζει ο δημιουργός. Δεν υπάρχουν αυτόματες ανανεώσεις ή εικονικοί μηχανισμοί (virtual DOM), κάτι που σε μικρές εφαρμογές προσφέρει καλύτερη επίδοση.

Περιορισμοί της χειροκίνητης DOM διαχείρισης

Ωστόσο, η απευθείας διαχείριση του DOM έχει και μειονεκτήματα:

- **Πολυπλοκότητα:** Καθώς η εφαρμογή μεγαλώνει, ο χειροκίνητος χειρισμός του DOM μπορεί να γίνει δύσκολος και ευάλωτος σε σφάλματα.
- **Απουσία "state-driven UI":** Σε contrast με frameworks όπως React, εδώ δεν υπάρχει αυτόματη αντιστοίχιση του state με την απόδοση (rendering) του UI. Όλα πρέπει να ελέγχονται χειροκίνητα.
- **Δυσκολία testing & debugging:** Λόγω απουσίας δομών όπως component trees και declarative rendering, απαιτείται αυξημένη προσοχή στον έλεγχο αλλαγών του UI.

Συμπεράσματα και σημασία

Η επιλογή να χρησιμοποιηθεί **Vanilla JS για DOM manipulation** ήταν στρατηγική. Παρότι επιβαρύνει τον προγραμματιστή με περισσότερες ευθύνες, προσφέρει ένα **εξαιρετικό πεδίο εκπαίδευσης** στις βασικές αρχές της web ανάπτυξης. Παράλληλα, αποδεικνύει ότι ακόμα και **χωρίς framework**, μπορεί να υλοποιηθεί ένα πλήρες και λειτουργικό browser game, βασισμένο αποκλειστικά στη δύναμη της εγγενούς JavaScript και του DOM API.

4.3 Χειρισμός συμβάντων (Event Handling)

Ο χειρισμός συμβάντων (event handling) αποτελεί μια κρίσιμη συνιστώσα στην ανάπτυξη διαδραστικών εφαρμογών στο περιβάλλον του φυλλομετρητή (browser). Στο πλαίσιο του παιχνιδιού, τα συμβάντα είναι ο μηχανισμός μέσω του οποίου ο χρήστης αλληλεπιδρά με τον κόσμο του παιχνιδιού – είτε μέσω του πληκτρολογίου είτε μέσω κλικ είτε μέσω χρονισμένων αντιδράσεων εντός του παιχνιδιού.

Η κατανόηση και η σωστή εφαρμογή του event handling είναι απαραίτητη για τη ροή του παιχνιδιού, την ενεργοποίηση animations, την ανταπόκριση σε εντολές του παίκτη και την εκκίνηση αλυσιδωτών ενεργειών μεταξύ των επιμέρους υποσυστημάτων.

Βασικές αρχές Event Handling στην JavaScript

Η JavaScript παρέχει μια σειρά από ενσωματωμένες δυνατότητες για τον χειρισμό συμβάντων. Οι πιο συχνές μέθοδοι που χρησιμοποιούνται είναι:

element.addEventListener("click", function(event) {

// κώδικας που εκτελείται όταν γίνει click

});

Η παραπάνω μέθοδος είναι ευέλικτη και ισχυρή, καθώς:

- Επιτρέπει την προσθήκη πολλαπλών ακροατών (listeners) στο ίδιο στοιχείο
- Προσφέρει πρόσβαση σε αντικείμενα τύπου Event με πληροφορίες για το συμβάν (π.χ. ποιο πλήκτρο πατήθηκε, σε ποιο σημείο έγινε το κλικ)
- Διατηρεί την προσέγγιση **separation of concerns** (διαχωρισμός της λογικής από την εμφάνιση)

Χειρισμός πληκτρολογίου (Keyboard Input)

Ένα βασικό μέρος της λειτουργικότητας του παιχνιδιού βασίζεται στη χρήση του πληκτρολογίου, καθώς ο παίκτης επιλέγει κινήσεις ή πλοηγείται στο UI χρησιμοποιώντας πλήκτρα όπως ArrowUp, ArrowDown, Enter κ.ά.

Παράδειγμα από το σύστημα μενού:

```
document.addEventListener("keydown", (e) => {
if (e.key === "ArrowUp") {
this.navigate(-1);
} else if (e.key === "ArrowDown") {
this.navigate(1);
} else if (e.key === "Enter") {
this.confirmSelection();
}
});
```

Αυτό επιτρέπει τη **λειτουργία τύπου console RPG**, όπου η πλοήγηση στο UI δεν γίνεται με ποντίκι, αλλά με πλήκτρα, δίνοντας στον παίκτη άμεση απόκριση και έλεγχο.

Συμβάντα μάχης & custom events

Πέρα από τα εγγενή (native) συμβάντα του DOM, το παιχνίδι αξιοποιεί **custom events** για την επικοινωνία μεταξύ διαφορετικών μονάδων λογικής. Για παράδειγμα, όταν ολοκληρώνεται μία ενέργεια (όπως επίθεση), ένα CustomEvent μπορεί να ενημερώσει άλλες κλάσεις ότι πρέπει να προχωρήσουν στην επόμενη φάση.

Παράδειγμα:

```
const event = new CustomEvent("BattleActionComplete", { detail: { who: "player" } });
document.dispatchEvent(event);
```

Και σε άλλη κλάση:

```
document.addEventListener("BattleActionComplete", (e) => {
this.nextTurn();
});
```

Αυτή η τεχνική διατηρεί τη **χαλαρή σύζευξη (loose coupling)** μεταξύ των modules. Έτσι, η λογική της μάχης δεν εξαρτάται άμεσα από το UI ή από άλλες μονάδες.

Αλυσίδες γεγονότων (event chains)

Ένα σημαντικό και ιδιαίτερο χαρακτηριστικό του παιχνιδιού είναι η ύπαρξη **αλυσίδων γεγονότων**, όπου η εκτέλεση μίας ενέργειας οδηγεί με χρονική καθυστέρηση στην ενεργοποίηση της επόμενης. Αυτό υλοποιείται κυρίως μέσω setTimeout() και Promise-based μηχανισμών.

Παράδειγμα:

await this.showMessage(`${caster.name} used ${action.name}!`);

await action.doAction(this);

await this.wait(500);

utils.emitEvent("BattleActionComplete");

Η παραπάνω ακολουθία είναι αντιπροσωπευτική ενός **game loop**, όπου κάθε φάση είναι διαδοχική και πρέπει να περιμένει την ολοκλήρωση της προηγούμενης.

Σχεδιαστικά οφέλη της προσέγγισης

Ο συνδυασμός native και custom events προσφέρει πολλαπλά οφέλη:

- **Επεκτασιμότητα:** Η προσθήκη νέων λειτουργιών μπορεί να γίνει χωρίς αλλαγές στον βασικό κορμό του παιχνιδιού.
- **Επαναχρησιμοποίηση:** Πολλαπλά συστήματα μπορούν να ανταποκρίνονται στο ίδιο γεγονός (π.χ. ενημέρωση UI, ήχοι, animation).
- **Ασυγχρονία:** Ο event-driven προγραμματισμός διευκολύνει την ασύγχρονη εκτέλεση ενεργειών, ιδανική για turn-based RPG.

Προκλήσεις και επιπλοκές

Παρόλο που η χρήση των γεγονότων είναι ιδιαίτερα αποτελεσματική, ενέχει και δυσκολίες:

- **Δυσκολία στον εντοπισμό σφαλμάτων (debugging):** Όταν πολλά συστήματα ανταποκρίνονται σε γεγονότα, είναι δύσκολο να βρεθεί ποιος "απαντά" σε τι.
- **Κρυφή ροή λογικής:** Η λογική ροή του παιχνιδιού γίνεται πιο δύσκολη στην ανάγνωση, καθώς η σειρά ενεργειών εξαρτάται από τα συμβάντα και όχι από ευθύγραμμο κώδικα.
- **Πολλαπλοί ακροατές (listeners):** Υπάρχει ο κίνδυνος να δημιουργηθούν πολλαπλοί listeners για το ίδιο event, προκαλώντας απρόβλεπτη συμπεριφορά.

Συμπεράσματα

Ο χειρισμός συμβάντων με τη χρήση τόσο **native DOM events** όσο και **custom events** είναι απαραίτητος για τη λειτουργία ενός διαδραστικού παιχνιδιού. Η προσέγγιση που υιοθετήθηκε στην παρούσα υλοποίηση βασίζεται σε τεχνικές **event-driven programming**, οι οποίες είναι ιδιαίτερα αποδοτικές σε turn-based μηχανισμούς και συστήματα real-time αλληλεπίδρασης.

Η προσεκτική οργάνωση των event handlers, η αποσυζευγμένη αρχιτεκτονική και η χρήση custom γεγονότων ενίσχυσαν τη modular λογική και την επεκτασιμότητα της εφαρμογής.

4.4 Προσέγγιση SPA χωρίς framework

Η ανάπτυξη ενός **Single Page Application (SPA)** χωρίς τη χρήση κάποιου framework (όπως React, Vue ή Angular) αποτελεί μια πρόκληση, αλλά ταυτόχρονα μια ευκαιρία βαθύτερης

κατανόησης των θεμελιωδών μηχανισμών του web. Στην παρούσα εφαρμογή, επιλέχθηκε η προσέγγιση "**Vanilla SPA**", δηλαδή η κατασκευή της εφαρμογής εξ ολοκλήρου με JavaScript, HTML και CSS, χωρίς καμία εξωτερική βιβλιοθήκη.

Η απόφαση αυτή ήταν στρατηγική, καθώς εξυπηρετεί τον σκοπό της πτυχιακής: την ανάδειξη του τρόπου με τον οποίο μπορούμε να κατασκευάσουμε λειτουργικές, μοντέρνες και modular web εφαρμογές, βασιζόμενοι αποκλειστικά σε εγγενή χαρακτηριστικά του browser και στην καθαρή JavaScript.

Τι είναι SPA;

Μια **Single Page Application (SPA)** είναι μια web εφαρμογή η οποία φορτώνεται **μία φορά** και αλληλεπιδρά με τον χρήστη **δυναμικά**, χωρίς να απαιτείται ανανέωση της σελίδας. Όλα τα επιμέρους στοιχεία και οι αλλαγές στη διεπαφή (UI) προκύπτουν μέσω JavaScript και DOM manipulation.

Αντίθετα με τις παραδοσιακές web εφαρμογές (Multi-Page Applications – MPA), όπου κάθε σελίδα απαιτεί νέο request στον server και ανανέωση του HTML, το SPA μοντέλο βελτιώνει την εμπειρία χρήστη (UX) μέσω:

- Ταχύτερων αποκρίσεων
- Ομαλής πλοήγησης
- Λιγότερης κυκλοφορίας δεδομένων
- Ενιαίου state κατά τη διάρκεια της χρήσης

Πώς υλοποιήθηκε SPA χωρίς framework

1. Μονοσέλιδη αρχιτεκτονική

Το παιχνίδι στηρίζεται σε μία και μόνο HTML σελίδα (index.html), η οποία περιλαμβάνει ένα βασικό DOM container:

* *

Όλα τα γραφικά, μενού, και μηνύματα του παιχνιδιού εγχέονται δυναμικά μέσα σε αυτό το container μέσω JavaScript. Δεν υπάρχουν links που οδηγούν σε άλλες HTML σελίδες, ούτε επαναφόρτωση του εγγράφου (page reload).

2. Δυναμική φόρτωση περιεχομένου

Η λογική της εφαρμογής είναι σχεδιασμένη ώστε κάθε κατάσταση του παιχνιδιού (battle, overworld, cutscene) να διαχειρίζεται τον DOM μέσω JavaScript:

this.element = document.createElement("div");

this.element.classList.add("Battle");

this.container.appendChild(this.element);

Κάθε module είναι υπεύθυνο για τη δική του απόδοση περιεχομένου, ακολουθώντας αρχές **modular UI rendering**.

3. Αντικατάσταση views μέσω DOM manipulation

Η εναλλαγή "οθονών" ή καταστάσεων (π.χ. από τον κόσμο του παιχνιδιού στη μάχη) γίνεται με την **αφαίρεση/προσθήκη DOM στοιχείων**:

this.element.remove(); // καθαρισμός προηγούμενου view

container.appendChild(newView); // νέα φάση ή μενού

Έτσι, επιτυγχάνεται η εμπειρία SPA — όλο το UI "ζει" σε μία σελίδα.

4. Διαχείριση κατάστασης (state management)

Αντί να χρησιμοποιηθεί κάποιος state management μηχανισμός όπως Redux ή Vuex, το παιχνίδι διατηρεί το state σε JavaScript objects:

- playerState → πληροφορίες παίκτη
- battle → ενεργή μάχη
- overworld → παγκόσμια κίνηση
- eventHandlers → ενεργά συμβάντα

Η λογική του state ενημερώνει απευθείας το UI, μέσω custom events και DOM updates. Παράδειγμα:

utils.emitEvent("LineupChanged");

Και στη συνέχεια το UI ακούει:

document.addEventListener("LineupChanged", () => {

this.updateHud();

});

5. Απλή πλοήγηση μέσω λογικής αντί για routes

Σε ένα framework, οι "routes" χειρίζονται ποιο view προβάλλεται. Εδώ, κάθε φάση (π.χ. cutscene, μάχη, κίνηση στο χάρτη) υλοποιείται μέσω *game state transitions*, χωρίς τη χρήση URLs.

Η ροή καθορίζεται από SceneTransitions, EventChains, και StoryFlags που ελέγχουν τη μετάβαση μεταξύ καταστάσεων.

Πλεονεκτήματα της προσέγγισης

- **Μηδενικές εξαρτήσεις (0 dependencies)**
  Δεν απαιτείται εγκατάσταση npm packages ή μεταγλώττιση. Το παιχνίδι τρέχει εξολοκλήρου από ένα απλό server.
- **Απόλυτος έλεγχος της ροής**
  Ο δημιουργός καθορίζει εξ' ολοκλήρου τη ροή, χωρίς να περιορίζεται από abstraction layers των frameworks.

- **Εκπαιδευτική αξία**

  Η απουσία abstraction αναγκάζει την πλήρη κατανόηση του DOM, του JavaScript event model, και της δομής ενός SPA.

- **Χαμηλό κόστος συντήρησης**

  Λιγότερος κώδικας-βιβλιοθήκες σημαίνει λιγότερες πιθανότητες για προβλήματα συμβατότητας στο μέλλον.

Προκλήσεις και περιορισμοί

- **Χειροκίνητη διαχείριση DOM/UI state:** Σε μεγάλα έργα, αυτή η προσέγγιση γίνεται γρήγορα περίπλοκη και επιρρεπής σε bugs.
- **Έλλειψη abstraction layers:** Χρειάζεται να γράψουμε μόνοι μας συστήματα όπως component rendering, state diffs, virtual DOM, κ.λπ.
- **Δυσκολότερη επεκτασιμότητα:** Αν το παιχνίδι επεκταθεί με περισσότερες λειτουργίες, η ανάγκη για abstraction θα γίνει επιτακτική.

Συμπεράσματα

Η προσέγγιση ενός **SPA χωρίς framework**, αν και λιγότερο συνηθισμένη σήμερα, παραμένει επίκαιρη σε έργα μικρής ή μεσαίας κλίμακας, ειδικά όταν ο στόχος είναι η πλήρης κατανόηση των θεμελιωδών εργαλείων του Web. Στην παρούσα υλοποίηση, απέδειξε πως είναι εφικτό να δημιουργηθεί ένα λειτουργικό και διαδραστικό turn-based RPG παιχνίδι αποκλειστικά με Vanilla JS, διατηρώντας modularity, επεκτασιμότητα και πλήρη έλεγχο στη ροή του περιβάλλοντος.

Η κατανόηση του πώς να δημιουργηθεί ένα SPA χωρίς “crutches” βοηθά τον προγραμματιστή να εκτιμήσει βαθύτερα τις έννοιες πίσω από frameworks και βιβλιοθήκες που χρησιμοποιούνται κατά κόρον στη σύγχρονη ανάπτυξη εφαρμογών.

Κεφάλαιο 5ο: Αρχιτεκτονική του Παιχνιδιού

Η επιτυχής ανάπτυξη μιας web-based εφαρμογής παιχνιδιού, ιδιαίτερα όταν υλοποιείται εξ ολοκλήρου με JavaScript χωρίς frameworks, προϋποθέτει μια καλά σχεδιασμένη και οργανωμένη αρχιτεκτονική. Η αρχιτεκτονική αυτή δεν αφορά μόνο την κατανομή των αρχείων και των φακέλων, αλλά και τον τρόπο με τον οποίο οργανώνεται ο κώδικας, η ροή της εκτέλεσης, η διαχείριση του game state, και η επικοινωνία μεταξύ των επιμέρους ενοτήτων.

Στο συγκεκριμένο έργο, το οποίο βασίζεται σε μια SPA (Single Page Application) προσέγγιση, η αρχιτεκτονική ακολουθεί μια **modular** και **event-driven** φιλοσοφία. Κάθε λειτουργική ενότητα (π.χ. μάχη, παίκτης, UI, μηχανισμός ιστορίας) διατηρεί τη δική της ευθύνη και αυτονομία, ενώ η επικοινωνία μεταξύ τους επιτυγχάνεται μέσω ενός κεντρικού βρόχου παιχνιδιού (game loop) και custom events. Η χρήση **global αντικειμένων μέσω του window** παρέχει ένα απλό αλλά αποτελεσματικό σύστημα διαμοιρασμού πληροφορίας, διατηρώντας παράλληλα το modularity του έργου.

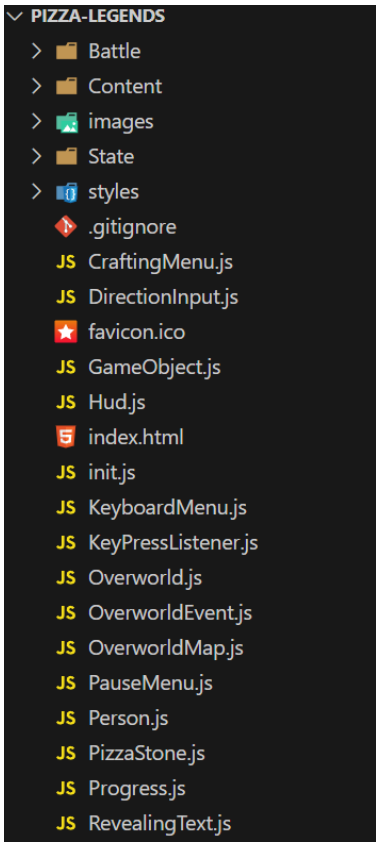
Στο παρόν κεφάλαιο θα εξετάσουμε διεξοδικά:

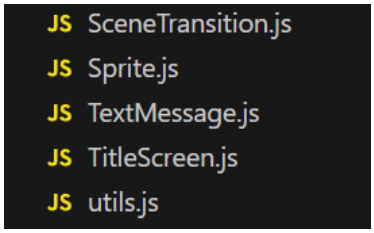
- Τη **δομή των φακέλων και των αρχείων** που συνθέτουν την εφαρμογή
- Τη **λογική ροή εκτέλεσης του παιχνιδιού** μέσα από τον μηχανισμό game loop
- Τον τρόπο που υλοποιούνται και χρησιμοποιούνται **global αντικείμενα** για την αποθήκευση και προσπέλαση των βασικών μονάδων του παιχνιδιού
- Το **μοντέλο μάχης** και το πώς η κεντρική κλάση TurnCycle διαχειρίζεται την εναλλαγή επιθέσεων και ενεργειών ανάμεσα στον παίκτη και τον εχθρό

Η αποτύπωση αυτής της αρχιτεκτονικής στο πλαίσιο της πτυχιακής εργασίας δεν αποσκοπεί μόνο στη θεωρητική της περιγραφή, αλλά και στην **κατανόηση των αποφάσεων σχεδιασμού**, των πλεονεκτημάτων και των πιθανών βελτιώσεων, λειτουργώντας ως οδηγός για παρόμοια εγχειρήματα στο μέλλον.

5.1 Δομή αρχείων και φακέλων

Η σωστή οργάνωση του project structure αποτελεί κρίσιμο παράγοντα για τη συντηρησιμότητα, την επεκτασιμότητα και την κατανόηση ενός έργου, ειδικά όταν πρόκειται για ένα παιχνίδι χωρίς τη χρήση framework ή game engine. Στην περίπτωση της παρούσας υλοποίησης, η δομή των αρχείων έχει οργανωθεί με τρόπο που διαχωρίζει τη λειτουργικότητα σε **σαφείς, θεματικές ενότητες**, διευκολύνοντας τόσο την ανάπτυξη όσο και την αναγνωσιμότητα του κώδικα. Παρακάτω βλέπουμε αναλυτικά την δομή:





Ριζικός κατάλογος (/project-root)

Ο κύριος φάκελος περιέχει τα βασικά entry-point αρχεία:

- **index.html** – Το βασικό HTML έγγραφο που φορτώνει το παιχνίδι και δημιουργεί το περιβάλλον για το canvas, menus, HUD κ.λπ.
- **init.js** – Το αρχείο εκκίνησης του παιχνιδιού: εδώ αρχικοποιείται η Overworld.
- **utils.js** – Βοηθητικές συναρτήσεις, π.χ. για randomness, delays, κ.ά.
- **favicon.ico**, **.gitignore** – Διαχειριστικά αρχεία για branding και version control.
- **Πλήθος αυτόνομων JS modules** – όπως PizzaStone.js, PauseMenu.js, TextMessage.js, που εξυπηρετούν συγκεκριμένες λειτουργίες.

/Battle/

Περιέχει όλη τη λογική που αφορά το **σύστημα μάχης**, δομημένο με modular τρόπο:

- **Battle.js** – Κεντρική οντότητα για τη δημιουργία του battle scene.
- **TurnCycle.js** – Ο μηχανισμός εναλλαγής γύρων μεταξύ μαχητών.

- **Combatant.js** – Κλάση που αναπαριστά έναν χαρακτήρα-μαχητή (παίκτη ή εχθρό).
- **SubmissionMenu.js, ReplacementMenu.js** – Μενού επιλογών μέσα στη μάχη.
- **BattleEvent.js, BattleAnimations.js** – Διαχείριση animation και event sequence για κάθε action.
- **Team.js** – Λογική διαχείρισης ομάδων (player / enemy team).

Η ύπαρξη υποσυστημάτων εντός της μάχης επιτρέπει την ανεξαρτησία και επεκτασιμότητα.

/Content/

Αρχείο περιεχομένου (content-driven architecture):

- **pizzas.js** – Κατάλογος όλων των διαθέσιμων "μονάδων" (π.χ. c001, f001 κ.λπ.), με στατιστικά και abilities.
- **enemies.js** – Ορισμοί αντιπάλων, team setups, boss compositions.
- **actions.js** – Ενέργειες (attacks, buffs, heals), με animation references και επιπτώσεις.

Όλα τα παραπάνω είναι externalized ως JSON-like JS αρχεία, ώστε να μπορούν να επεκταθούν εύκολα χωρίς να αλλάζει η λογική.

/Images/

Ο φάκελος των assets χωρίζεται σε υποφακέλους:

- **characters/** – Sprites για αντικείμενα και NPCs.
- **characters/people/** – Πλήθος NPC sprites (hero, npc1-npc8, bosses).
- **characters/pizzas/** – Εικονίδια για τις μονάδες-pizza (f001, c002 κ.ά.).
- **icons/** – Icons για είδη/ιδιότητες (chill, spicy, fungi κ.λπ.).
- **maps/** – Layers χαρτών: Lower, Upper, Battle, κ.λπ.
- **ui/** – UI assets όπως πλαίσια πληροφορίας (combatant plates).

Η δομή αυτή διευκολύνει την αναζήτηση και ανανέωση οπτικού υλικού.

/State/

- **PlayerState.js** – Αναπαριστά τη **λογική κατάστασης του παίκτη** (lineup, αντικείμενα, story flags, κ.ά.).
- Είναι κεντρικό σημείο για την **εξομοίωση μνήμης** μεταξύ σκηνών, μάχης και overworld.

/Styles/

Κάθε στοιχείο του UI ή υποσύστημα έχει το δικό του CSS module:

- **global.css** – Βασικοί κανόνες για το σώμα, resets, canvas, κ.ά.
- **Battle.css, Hud.css, Menus.css, TextMessage.css, κ.ά.** – Modular styles για συγκεκριμένα components.

Ακολουθείται η πρακτική του **scoped styling per component**, ενισχύοντας την αναγνωσιμότητα και αποφυγή conflicts.

Ανεξάρτητα αρχεία (modules)

Πολλά JS αρχεία υπάρχουν ως **αυτόνομες κλάσεις**:

- **DirectionInput.js**, **KeyPressListener.js** – Χειρίζονται input devices.
- **Hud.js**, **KeyboardMenu.js** – Διαχειρίζονται user interface.
- **Overworld.js, OverworldMap.js, OverworldEvent.js** – Οργανώνουν τη χαρτογράφηση και την πλοήγηση.
- **SceneTransition.js, TitleScreen.js, RevealingText.js** – Στοιχεία εμπειρίας (UX).

Η χρήση module-per-class προωθεί την επαναχρησιμοποίηση και την **αντικειμενοστραφή οργάνωση**.

5.2 Κεντρική ροή παιχνιδιού (Game Loop)

Η **ροή του παιχνιδιού**, σε ένα browser-based RPG χωρίς χρήση game engine, στηρίζεται σε μια θεμελιώδη έννοια: τον **game loop**. Πρόκειται για τον κυκλικό μηχανισμό εκτέλεσης που διατηρεί το παιχνίδι "ζωντανό", ανανεώνοντας την εικόνα, τον έλεγχο και την αλληλεπίδραση του χρήστη με το περιβάλλον σε κάθε frame.

Τι είναι το Game Loop

Ο game loop είναι ένας συνεχής κύκλος που:

1. Διαβάζει είσοδο χρήστη (input),
2. Ενημερώνει την κατάσταση του παιχνιδιού (state),
3. Αποδίδει οπτικά την τρέχουσα κατάσταση (rendering),
4. Επαναλαμβάνεται, συνήθως μέσω requestAnimationFrame().

Αυτή η διαδικασία διασφαλίζει ότι το παιχνίδι ανταποκρίνεται σε κάθε ενέργεια του παίκτη και προβάλλει συνεχώς μια επικαιροποιημένη εικόνα του κόσμου.

Υλοποίηση στο πλαίσιο του έργου

Η κύρια ροή παιχνιδιού ξεκινά μέσω του αρχείου init.js, όπου αρχικοποιείται το βασικό αντικείμενο Overworld. Εκεί, ορίζεται η βασική game loop:

```
startGameLoop() {
  const step = () => {
    this.update(); // Ενημέρωση θέσεων, collision, κ.λπ.
    this.render(); // Ζωγραφική canvas και sprites
    requestAnimationFrame(() => {
      step(); // Αναδρομική κλήση του loop
```

```
    });
  };
  step();
}
```

Η μέθοδος αυτή καλείται όταν φορτωθεί ο κόσμος (Overworld), ξεκινώντας ουσιαστικά την εμπειρία του παιχνιδιού.

Καθήκοντα του Loop ανά κύκλο

Ο κάθε κύκλος του loop επιτελεί μια σειρά από ενέργειες:

1. Ενημέρωση (update)

- Ενημερώνει τη θέση του ήρωα (βάσει DirectionInput)
- Ανιχνεύει collisions με αντικείμενα ή NPCs
- Ελέγχει αν πυροδοτούνται γεγονότα (π.χ. έναρξη μάχης)
- Διαχειρίζεται animations που είναι σε εξέλιξη

2. Απόδοση (render)

- Καθαρίζει και επανασχεδιάζει το canvas
- Ζωγραφίζει το **κάτω layer** (π.χ. έδαφος)
- Τοποθετεί και απεικονίζει sprites του παίκτη και NPCs
- Ζωγραφίζει το **πάνω layer** (π.χ. τοίχους, αντικείμενα)
- Εμφανίζει HUD στοιχεία (όπως menus, text boxes)

Η χρήση του canvas.getContext("2d") είναι κρίσιμη για την απόδοση των frames.

Αλληλεπίδραση με τον Παίκτη

Ο χρήστης δεν "σταματά" το game loop. Οι ενέργειες του (keypress, επιλογές σε menus κ.λπ.) **τροποποιούν την κατάσταση** (state), η οποία αντανακλάται στον επόμενο κύκλο του loop. Αυτό κάνει την εμπειρία διαδραστική και ρευστή.

Επιπλέον, στοιχεία όπως KeyPressListener, DirectionInput και KeyboardMenu ενσωματώνονται λειτουργικά στη ροή, αφού αναλύουν την είσοδο και επηρεάζουν την ενημέρωση του κόσμου ή των menus.

Χρονισμός και Απόδοση

Το requestAnimationFrame() είναι η προτιμώμενη μέθοδος για browser game loops, καθώς:

- Προσαρμόζεται δυναμικά στον ρυθμό ανανέωσης της οθόνης (~60fps)
- Βελτιστοποιείται από τον browser (σε idle tabs δεν εκτελείται)
- Παρέχει καλύτερη εμπειρία σε σχέση με setInterval ή setTimeout

Αυτή η επιλογή ενισχύει την **αποδοτικότητα** και **ρευστότητα** του παιχνιδιού, ιδιαίτερα σε περιβάλλον χωρίς framework ή engine.

Διαχείριση Καταστάσεων – Παρεμβολές στη Ροή

Παρόλο που το game loop εκτελείται συνεχώς, υπάρχουν περιπτώσεις όπου η **ροή "παγώνει"** ή **τροποποιείται προσωρινά**, όπως:

- Εμφάνιση menus (PauseMenu, KeyboardMenu)
- Διαλόγων (TextMessage, RevealingText)
- Εναλλαγή σκηνών (SceneTransition)
- Εκκίνηση μάχης (OverworldEvent → Battle)

Αυτό επιτυγχάνεται μέσω **flags** ή **μηχανισμών promise/await**, ώστε να διακόπτεται η είσοδος ή η ενημέρωση μέχρι την ολοκλήρωση του συμβάντος.

Σχέση με άλλα Υποσυστήματα

Το game loop συνεργάζεται άμεσα με:

- Sprite.js για την απεικόνιση χαρακτήρων και animation frames
- Person.js για την ενημέρωση θέσεων NPCs
- OverworldMap.js για τη λογική των layers και των αντικειμένων
- OverworldEvent.js για τα γεγονότα που ενσωματώνονται στη ροή

Αυτό το επίπεδο συνεργασίας διατηρεί τη λογική "component-based", δίνοντας modular χαρακτήρα στη ροή.

Συμπέρασμα

Ο game loop στο συγκεκριμένο έργο:

- Υλοποιείται με requestAnimationFrame() για απόδοση και ευελιξία
- Διαχειρίζεται ενημέρωση κατάστασης και απόδοση γραφικών σε κάθε frame
- Ενσωματώνει διαδραστικά στοιχεία (input, events, UI) σε πραγματικό χρόνο
- Διατηρεί τον πλήρη έλεγχο της εφαρμογής, χωρίς τη χρήση engine ή framework

Η ικανή και καθαρή υλοποίηση αυτού του loop αποτελεί τον θεμέλιο λίθο για όλη τη δυναμική του παιχνιδιού.

5.3 Οργάνωση σε "window" global αντικείμενα

Η επιλογή οργάνωσης της εφαρμογής μέσω **global μεταβλητών και αντικειμένων στο `window` object** αποτέλεσε έναν συνειδητό σχεδιαστικό συμβιβασμό, με στόχο την ευκολία πρόσβασης και τη διασύνδεση υποσυστημάτων σε ένα πλαίσιο χωρίς χρήση framework ή module system.

Σε ένα περιβάλλον Vanilla JavaScript χωρίς ES Modules ή bundler (όπως Webpack/Vite), η χρήση του `window` ως κοινός χώρος αποθήκευσης **παγκόσμια προσβάσιμων οντοτήτων** ήταν πρακτική λύση για την αποδοτική διαχείριση της κατάστασης και των μηχανισμών του παιχνιδιού.

Γιατί χρησιμοποιήθηκαν window αντικείμενα;

Στόχοι της προσέγγισης:

- **Διαμοιρασμός δεδομένων και αντικειμένων** μεταξύ ανεξάρτητων αρχείων χωρίς import/export
- **Ορατότητα και debugging** σε περιβάλλον browser (όλα διαθέσιμα από το DevTools console)
- **Εκτέλεση logic-driven triggers** από οποιοδήποτε σημείο της εφαρμογής (π.χ. ενεργοποίηση μάχης, πρόσβαση στον παίκτη, αναφορά σε assets)

Παρόλο που αυτή η προσέγγιση **δεν ενδείκνυται για μεγάλης κλίμακας εφαρμογές** (λόγω του κινδύνου namespace pollution και διαρροών), στην παρούσα περίπτωση εξυπηρέτησε εκπαιδευτικό και πειραματικό σκοπό.

Παραδείγματα window αντικειμένων στο παιχνίδι

Ας δούμε κάποια βασικά global αντικείμενα που έχουν δεσμευτεί στο window:

window.PizzaTypes:

Αντικείμενο που περιέχει όλα τα διαθέσιμα **pizzas** του παιχνιδιού, με ιδιότητες όπως name, type, src, actions.

Δημιουργείται βάσει του περιεχομένου του Content/pizzas.js.

window.Actions:

Περιέχει τους ορισμούς όλων των δυνατών **επιθετικών, αμυντικών ή utility ενεργειών** που μπορεί να εκτελέσει μια πίτσα σε μάχη. Οι actions αυτές χρησιμοποιούνται από το Battle System και το SubmissionMenu.

window.Enemies:

Περιλαμβάνει όλους τους **αντιπάλους** (NPCs ή bosses) με πληροφορίες όπως:

- Ποιες πίτσες έχουν στην ομάδα τους
- Ποια τα attributes τους
- Ποιες οι διαθέσιμες κινήσεις

window.playerState:

Το σημαντικότερο ίσως global αντικείμενο, υπεύθυνο για την αποθήκευση:

- Της σύνθεσης της ομάδας του παίκτη
- Του inventory (αντικείμενα, υλικά crafting)

- Της προόδου στο παιχνίδι (π.χ. νικημένοι εχθροί)

Η ύπαρξή του επιτρέπει την **κεντρική διαχείριση της κατάστασης** από οποιοδήποτε σημείο, χωρίς την ανάγκη για context providers ή state management libraries.

window.Utils:

Περιέχει χρήσιμες βοηθητικές συναρτήσεις όπως:

- wait(ms) για καθυστερήσεις
- randomFromArray() για RNG λογική
- emitEvent() για custom DOM events

Σχέση με άλλα συστήματα

Η χρήση αυτών των global αντικειμένων διασυνδέεται άμεσα με τη δομή του παιχνιδιού:

- Το Overworld τραβάει από το window.playerState για να ορίσει ποιος χαρακτήρας θα εμφανιστεί στον χάρτη
- Το Battle.js φορτώνει πληροφορίες από PizzaTypes και Actions για την απόδοση της μάχης
- Τα μενού (KeyboardMenu, CraftingMenu) διαβάζουν ή επηρεάζουν την κατάσταση μέσω playerState

Προβληματισμοί και Περιορισμοί

Παρότι πρακτική, η χρήση των window αντικειμένων συνοδεύεται από κάποια σημαντικά μειονεκτήματα:

Πρόβλημα	Περιγραφή
Ονοματοδοσία/σύγκρουση	Όλα είναι στο ίδιο namespace – πιθανότητα σύγκρουσης με άλλες global μεταβλητές
Δυσκολία συντήρησης	Δε φαίνεται ξεκάθαρα από πού προέρχονται ή ποιος εξαρτάται από τι
Απώλεια modularity	Τα scripts δεν είναι πλήρως ανεξάρτητα
Κίνδυνος διαρροών/τροποποιήσεων	Οποιοδήποτε μέρος της εφαρμογής μπορεί να μεταβάλει την global κατάσταση

Ωστόσο, στα πλαίσια του συγκεκριμένου project, οι αρνητικές συνέπειες ήταν ελεγχόμενες χάρη στη σαφή ονομασία, την οργανωμένη δομή και το μικρό μέγεθος της εφαρμογής.

Συμπέρασμα

Η χρήση των global window αντικειμένων αποτέλεσε μια συνειδητή απόφαση για να διευκολυνθεί:

- Η γρήγορη υλοποίηση χωρίς module loader
- Η άμεση προσβασιμότητα δεδομένων
- Η ενότητα μεταξύ ανεξάρτητων scripts

Παρότι αυτή η προσέγγιση **δεν αποτελεί best practice σε μεγάλης κλίμακας εφαρμογές**, στην εκπαιδευτική φύση του έργου ήταν επαρκής και λειτουργική. Η αυστηρή ονοματοδοσία και η χρήση καθαρών δομών απέτρεψε παγίδες συντήρησης και αύξησε την αναγνωσιμότητα.

5.4 Μοντέλο μάχης και μηχανισμός TurnCycle

Η μάχη αποτελεί τον πιο σύνθετο και δυναμικό μηχανισμό του παιχνιδιού. Το σύστημα είναι δομημένο με τρόπο ώστε να υποστηρίζει:

- turn-based στρατηγική αλληλεπίδραση (τύπου JRPG),
- modular επέκταση νέων ενεργειών και αντιπάλων,
- καθαρό διαχωρισμό μεταξύ **game state**, **UI rendering**, και **battle logic**.

Η καρδιά της λειτουργίας βρίσκεται στην κλάση TurnCycle, η οποία διαχειρίζεται τη ροή των γύρων και των ενεργειών κάθε συμμετέχοντα, με γνώμονα τον **κανόνα εναλλαγής** και την **εκτέλεση ενεργειών**.

Αρχιτεκτονικό μοντέλο της μάχης

Η μάχη υλοποιείται μέσα από το αρχείο Battle/Battle.js και τις υποστηρικτικές του κλάσεις:

Κλάση / Μονάδα	Ρόλος
Battle.js	Αρχικοποιεί το battle context (ομάδες, UI, DOM)
Combatant.js	Ορίζει τις μονάδες μάχης (παίκτης/εχθρός), με stats, status effects κ.ά.
BattleEvent.js	Εκτελεί atomic events (επιθέσεις, status updates, αλλαγές)
SubmissionMenu.js	Υπεύθυνο για την είσοδο του παίκτη (επιλογή ενέργειας)
ReplacementMenu.js	Υλοποιεί μηχανισμό αλλαγής πίτσας
TurnCycle.js	Διαχειρίζεται τη ροή των γύρων, την εναλλαγή και την ακολουθία των ενεργειών

Αυτός ο διαχωρισμός καθιστά εύκολη την κατανόηση, επαναχρησιμοποίηση και επέκταση του συστήματος.

Μηχανισμός TurnCycle – Πώς λειτουργεί;

Ο TurnCycle είναι υπεύθυνος για τη **συντονισμένη εκτέλεση ενεργειών ανά γύρο**, ακολουθώντας την παρακάτω ροή:

1. **Καθορισμός ενεργού combatant (ή παίκτη)**
   Ορίζει ποιος έχει σειρά: παίκτης ή εχθρός.
2. **Προσπάθεια υποβολής ενέργειας**
   - Αν είναι ο παίκτης → ανοίγει SubmissionMenu

- Αν είναι ο εχθρός → αυτόματη απόφαση (AI)

3. **Δημιουργία και εκτέλεση BattleEvent**
   Το event αφορά την επίθεση, heal, status effect κτλ.
4. **Έλεγχος αποτελέσματος και κατάστασης μάχης**
   Αν κάποιος combatant πέσει κάτω, πυροδοτείται defeat logic.
5. **Εναλλαγή σειράς / συνέχεια στον επόμενο γύρο**

Ουσιαστικά, η TurnCycle είναι ένα **loop εντός της μάχης** που περιμένει input ή AI, επεξεργάζεται συνέπειες και ανανεώνει την κατάσταση.

Δομή κλάσης TurnCycle (περίληψη)

```
class TurnCycle {
  constructor({ battle, onNewEvent }) {
    this.battle = battle;
    this.onNewEvent = onNewEvent;
    this.currentTeam = "player"; // εναλλάσσεται
  }

  async turn() {
    // Λήψη ενεργού combatant
    const caster = this.battle.activeCombatants[this.currentTeam];

    // Λήψη ενέργειας (από menu ή AI)
    const submission = await this.getSubmission(caster);

    // Δημιουργία BattleEvent και εκτέλεση
    const event = new BattleEvent({ submission, battle: this.battle });
    await event.init();

    // Εναλλαγή σειράς και επανάληψη
    this.currentTeam = this.currentTeam === "player" ? "enemy" : "player";
    this.turn();
  }
}
```

Αυτό το loop είναι **ασύγχρονο** (χρήση await) ώστε να περιμένει τις υποβολές και τα animation delays.

Το μοντέλο Combatant – Δεδομένα μονάδας μάχης

Κάθε μονάδα μάχης (π.χ. πίτσα του παίκτη ή του εχθρού) εκπροσωπείται από ένα αντικείμενο Combatant με τα εξής δεδομένα:

- hp, maxHp
- xp, level, status (π.χ. burned, saucy)
- actions που μπορεί να χρησιμοποιήσει
- team: player ή enemy
- isActive: αν είναι η τρέχουσα ενεργή μονάδα

Η δομή αυτή επιτρέπει το **stateful rendering** και την απεικόνιση κατάστασης στο Hud.js.

```
class Combatant {
  constructor(config, battle) {
    Object.keys(config).forEach(key => {
      this[key] = config[key];
    })
    this.hp = typeof(this.hp) === "undefined" ? this.maxHp : this.hp;
    this.battle = battle;
  }

  get hpPercent() {
    const percent = this.hp / this.maxHp * 100;
    return percent > 0 ? percent : 0;
  }

  get xpPercent() {
    return this.xp / this.maxXp * 100;
  }

  get isActive() {
    return this.battle?.activeCombatants[this.team] === this.id;
  }

  get givesXp() {
    return this.level * 20;
  }

  createElement() {
    this.hudElement = document.createElement("div");
    this.hudElement.classList.add("Combatant");
    this.hudElement.setAttribute("data-combatant", this.id);
    this.hudElement.setAttribute("data-team", this.team);
    this.hudElement.innerHTML = (`
      <p class="Combatant_name">${this.name}</p>
      <p class="Combatant_level"></p>
      <div class="Combatant_character_crop">
        <img class="Combatant_character" alt="${this.name}" src="${this.src}" />
      </div>
      <img class="Combatant_type" src="${this.icon}" alt="${this.type}" />
      <svg viewBox="0 0 26 3" class="Combatant_life-container">
        <rect x=0 y=0 width="0%" height=1 fill="#82ff71" />
        <rect x=0 y=1 width="0%" height=2 fill="#3ef126" />
      </svg>
      <svg viewBox="0 0 26 2" class="Combatant_xp-container">
        <rect x=0 y=0 width="0%" height=1 fill="#ffd76a" />
        <rect x=0 y=1 width="0%" height=1 fill="#ffc934" />
      </svg>
      <p class="Combatant_status"></p>
    `);

    this.pizzaElement = document.createElement("img");
    this.pizzaElement.classList.add("Pizza");
    this.pizzaElement.setAttribute("src", this.src );
    this.pizzaElement.setAttribute("alt", this.name );
    this.pizzaElement.setAttribute("data-team", this.team );

    this.hpFills = this.hudElement.querySelectorAll(".Combatant_life-container > rect");
    this.xpFills = this.hudElement.querySelectorAll(".Combatant_xp-container > rect");
  }
```

```
update(changes={}) {
    //Update anything incoming
    Object.keys(changes).forEach(key => {
      this[key] = changes[key]
    });

    //Update active flag to show the correct pizza & hud
    this.hudElement.setAttribute("data-active", this.isActive);
    this.pizzaElement.setAttribute("data-active", this.isActive);

    //Update HP & XP percent fills
    this.hpFills.forEach(rect => rect.style.width = `${this.hpPercent}%`)
    this.xpFills.forEach(rect => rect.style.width = `${this.xpPercent}%`)

    //Update level on screen
    this.hudElement.querySelector(".Combatant_level").innerText = this.level;

    //Update status
    const statusElement = this.hudElement.querySelector(".Combatant_status");
    if (this.status) {
      statusElement.innerText = this.status.type;
      statusElement.style.display = "block";
    } else {
      statusElement.innerText = "";
      statusElement.style.display = "none";
    }
  }

  getReplacedEvents(originalEvents) {

    if (this.status?.type === "clumsy" && utils.randomFromArray([true, false, false])) {
      return [
        { type: "textMessage", text: `${this.name} flops over!` },
      ]
    }

    return originalEvents;
  }

  getPostEvents() {
    if (this.status?.type === "saucy") {
      return [
        { type: "textMessage", text: "Feelin' saucy!" },
        { type: "stateChange", recover: 5, onCaster: true }
      ]
    }
    return [];
  }

  decrementStatus() {
    if (this.status?.expiresIn > 0) {
      this.status.expiresIn -= 1;
      if (this.status.expiresIn === 0) {
        this.update({
          status: null
        })
        return {
          type: "textMessage",
          text: "Status expired!"
        }
      }
    }
    return null;
  }

  init(container) {
    this.createElement();
    container.appendChild(this.hudElement);
    container.appendChild(this.pizzaElement);
    this.update();
  }

}
```

AI μοντέλου για τον εχθρό

Η υποβολή ενέργειας για τον εχθρό γίνεται αυτόματα με λογική RNG:

getSubmission(caster) {

if (caster.team === "enemy") {

return {

```
        action: randomFromArray(caster.actions),
        target: playerActiveCombatant
      }
    }
    // Αλλιώς, SubmissionMenu για τον παίκτη
  }
```

Η συμπεριφορά αυτή μπορεί να εξελιχθεί με **prioritization logic** ή **adaptive AI**.

Οπτικοποίηση / Animation συστήματος μάχης

Τα animations (π.χ. επιθέσεις, ζημιές, αλλαγές sprite) γίνονται μέσω BattleAnimations.js που ορίζει χρονικά block βασισμένα σε DOM manipulation και CSS classes.

```
window.BattleAnimations = {
  async spin(event, onComplete) {
    const element = event.caster.pizzaElement;
    const animationClassName = event.caster.team === "player" ? "battle-spin-right" : "battle-spin-left";
    element.classList.add(animationClassName);

    //Remove class when animation is fully complete
    element.addEventListener("animationend", () => {
      element.classList.remove(animationClassName);
    }, { once:true });

    //Continue battle cycle right around when the pizzas collide
    await utils.wait(100);
    onComplete();
  },

  async glob(event, onComplete) {
    const {caster} = event;
    let div = document.createElement("div");
    div.classList.add("glob-orb");
    div.classList.add(caster.team === "player" ? "battle-glob-right" : "battle-glob-left");

    div.innerHTML = (`
      <svg viewBox="0 0 32 32" width="32" height="32">
        <circle cx="16" cy="16" r="16" fill="${event.color}" />
      </svg>
    `);

    //Remove class when animation is fully complete
    div.addEventListener("animationend", () => {
      div.remove();
    });

    //Add to scene
    document.querySelector(".Battle").appendChild(div);

    await utils.wait(820);
    onComplete();
  }
}
```

Πλεονεκτήματα της προσέγγισης

- Καθαρός διαχωρισμός ευθυνών
- Δυνατότητα επεκτασιμότητας (π.χ. νέα abilities, διαφορετικοί κανόνες)
- Καλή υποστήριξη debugging

- Εύκολη προσαρμογή σε διαφορετικά UI

Προκλήσεις / Περιορισμοί

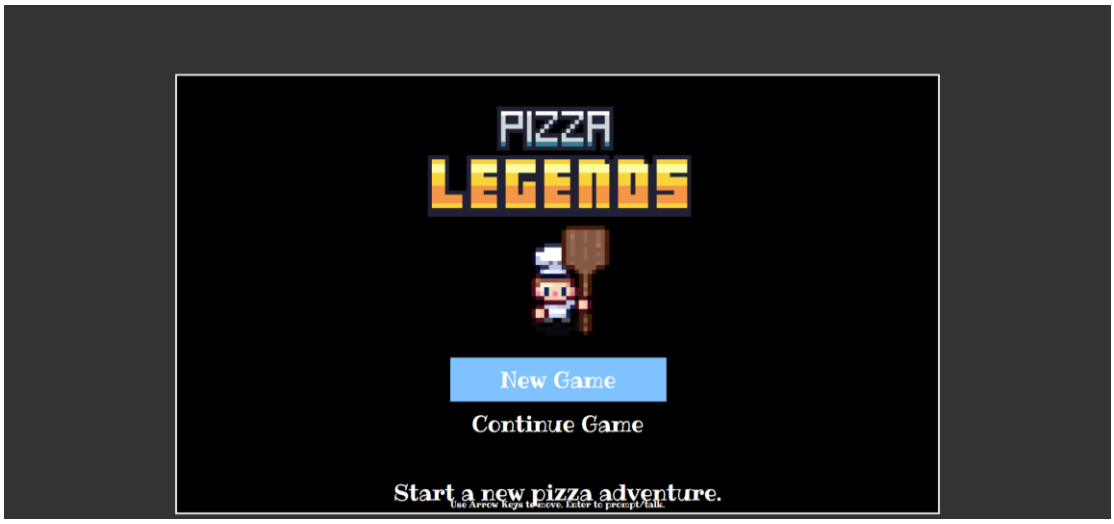
- Δεν υποστηρίζεται ακόμα queue για παράλληλα effects
- Τα status effects χρειάζονται ακόμα workarounds (π.χ. cooldowns, debuffs)
- Δεν υπάρχει αρχιτεκτονική για multiplayer ή πραγματικό χρόνο

Συμπέρασμα

Το σύστημα μάχης βασισμένο σε TurnCycle προσφέρει ένα λειτουργικό, επεκτάσιμο και καθαρά ορισμένο combat model. Παρά την απλότητα, προσομοιώνει με επιτυχία turn-based ροή και καθιστά τον κώδικα αναγνώσιμο και ευέλικτο.

Η απόφαση να χτιστεί χωρίς framework ενίσχυσε την κατανόηση του τρόπου λειτουργίας κάθε επιμέρους στοιχείου και παρείχε σημαντική εμπειρία σε game logic σχεδιασμό.

Κεφάλαιο 6ο: Βασικά Συστατικά & Game Entities

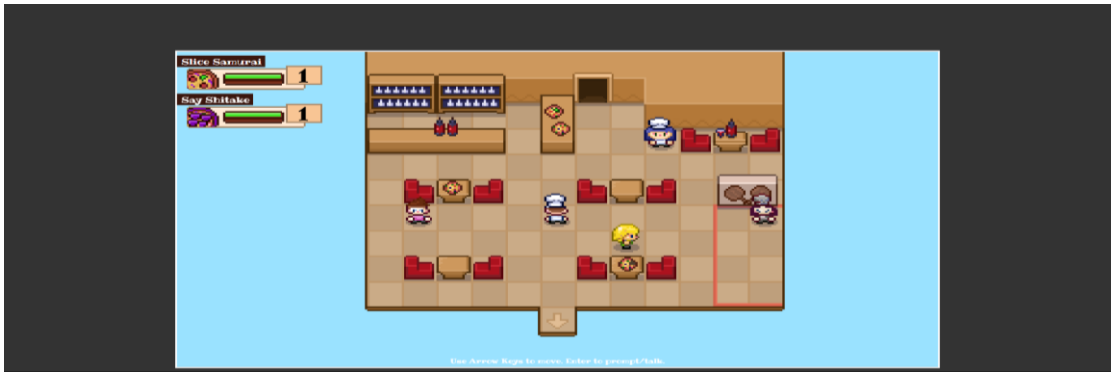


Σε κάθε παιχνίδι ρόλων (RPG) ή adventure-based σύστημα, ο πυρήνας της εμπειρίας βασίζεται στην ύπαρξη και τη συμπεριφορά των βασικών οντοτήτων (entities) που διαμορφώνουν τον κόσμο του παιχνιδιού. Στην παρούσα υλοποίηση, οι "οντότητες" αυτές είναι προσεκτικά σχεδιασμένες ώστε να υποστηρίζουν:

- **μηχανισμούς μάχης** (όπως πίτσες και εχθροί),
- **προσωρινές ή μόνιμες καταστάσεις** (όπως status effects),
- **ενεργές αποφάσεις από τον παίκτη** (μέσω actions ή αντικειμένων),
- και **καταγραφή της εξέλιξης** (PlayerState, XP, αντικείμενα).

Το σύστημα είναι πλήρως **modular** και βασίζεται σε **δεδομένα JSON/JS** που καθορίζουν τις ιδιότητες κάθε entity. Αυτό επιτρέπει την εύκολη προσθήκη νέων τύπων μονάδων, ενεργειών ή καταστάσεων χωρίς την ανάγκη επεμβάσεων στον βασικό κώδικα.

Η υλοποίηση βασίζεται σε αρχές **composition over inheritance**, εστιάζοντας στην εκφραστικότητα και την ευελιξία. Οι οντότητες δεν είναι απλά "εικονίδια" σε μια canvas, αλλά έχουν **δεδομένα κατάστασης, συμπεριφοράς και δράσης** τα οποία ενεργοποιούνται μέσα από το battle logic ή το world exploration.



Η ενότητα αυτή αναλύει τα βασικά στοιχεία του παιχνιδιού:

- **Pizzas**: Οι βασικές "μονάδες μάχης" του παίκτη, με stats, τύπους και ενεργές ικανότητες.
- **Εχθροί (Enemies)**: Προκαθορισμένοι χαρακτήρες με μοναδικά χαρακτηριστικά και στρατηγικές.
- **PlayerState**: Το αντικείμενο που διατηρεί τα δεδομένα του παίκτη (πίτσες, XP, inventory).
- **Καταστάσεις & Items**: Ειδικές επιδράσεις (burned, saucy κ.ά.) και χρήση αντικειμένων (π.χ. healing).

Συνολικά, τα entities του παιχνιδιού λειτουργούν ως θεμέλια του game logic και προσφέρουν ένα **συνεκτικό, επεκτάσιμο και πλήρως παραμετροποιήσιμο πλαίσιο gameplay**.

6.1 Pizzas: Ιδιότητες, Τύποι & Ενέργειες

Στο πλαίσιο του παιχνιδιού, οι **Pizzas** λειτουργούν ως οι βασικές "μονάδες μάχης" του παίκτη. Παρότι σε θεματικό επίπεδο πρόκειται για φανταστικές πίτσες με ονόματα και ιδιαίτερα χαρακτηριστικά, σε επίπεδο λογικής αντιστοιχούν σε **μάχιμους χαρακτήρες (combatants)**, με στατιστικά, ενέργειες, τύπους και δυνατότητες εξέλιξης.

Οι πληροφορίες που αφορούν τις πίτσες ορίζονται στο αρχείο `Content/pizzas.js`, σε μορφή αντικειμένων JavaScript. Κάθε pizza έχει τις δικές της ιδιότητες, που περιγράφονται παρακάτω.

```
window.PizzaTypes = {
  normal: "normal",
  spicy: "spicy",
  veggie: "veggie",
  fungi: "fungi",
  chill: "chill",
}

window.Pizzas = {
  "s001": {
    name: "Slice Samurai",
    description: "Pizza desc here",
    type: PizzaTypes.spicy,
    src: "/images/characters/pizzas/s001.png",
    icon: "/images/icons/spicy.png",
    actions: [ "saucyStatus", "clumsyStatus", "damage1" ],
  },
  "s002": {
    name: "Bacon Brigade",
    description: "A salty warrior who fears nothing",
    type: PizzaTypes.spicy,
    src: "/images/characters/pizzas/s002.png",
    icon: "/images/icons/spicy.png",
    actions: [ "damage1", "saucyStatus", "clumsyStatus" ],
  },
  "v001": {
    name: "Call Me Kale",
    description: "Pizza desc here",
    type: PizzaTypes.veggie,
    src: "/images/characters/pizzas/v001.png",
    icon: "/images/icons/veggie.png",
    actions: [ "damage1" ],
  },
  "v002": {
    name: "Archie Artichoke",
    description: "Pizza desc here",
    type: PizzaTypes.veggie,
    src: "/images/characters/pizzas/v001.png",
    icon: "/images/icons/veggie.png",
    actions: [ "damage1" ],
  },
  "f001": {
    name: "Portobello Express",
    description: "Pizza desc here",
    type: PizzaTypes.fungi,
    src: "/images/characters/pizzas/f001.png",
    icon: "/images/icons/fungi.png",
    actions: [ "damage1" ],
  },
  "f002": {
    name: "Say Shitake",
    description: "Pizza desc here",
    type: PizzaTypes.fungi,
    src: "/images/characters/pizzas/f001.png",
    icon: "/images/icons/fungi.png",
    actions: [ "damage1" ],
  }
}
```

Ιδιότητες Pizza (Attributes)

Κάθε Pizza ορίζεται από τις εξής βασικές ιδιότητες:

- name: Το όνομα της πίτσας (π.χ. "Slice Samurai", "Saucy Dog").

- description: Μικρή περιγραφή (για το UI).
- type: Ο "τύπος" της πίτσας (π.χ. spicy, veggie, chill).
- src: Η εικόνα που εμφανίζεται στο UI και στις μάχες.
- icon: Το εικονίδιο τύπου (π.χ. εικονίδιο chili για τις spicy πίτσες).
- actions: Πίνακας ενεργειών που μπορεί να εκτελέσει.
- maxHp: Το μέγιστο HP της πίτσας.
- xp: Το τρέχον επίπεδο εμπειρίας της πίτσας (για level up).

Αυτή η προσέγγιση επιτρέπει την **εύκολη επέκταση** του συστήματος: για να προστεθεί μια νέα πίτσα, αρκεί να δημιουργηθεί ένα νέο entry με τα ανάλογα χαρακτηριστικά.

Τύποι Pizza (Types)

Οι τύποι πίτσας λειτουργούν παρόμοια με τα **elemental types** σε άλλα RPGs (όπως τα Fire/Water/Grass στα Pokémon). Υπάρχουν τρεις βασικοί τύποι:

- **Spicy**
  Έχουν πιο επιθετικό προφίλ, εστιάζοντας στο damage. Συνήθως χρησιμοποιούν κινήσεις που προκαλούν burn ή debuffs.
- **Veggie**
  Πιο ισορροπημένες πίτσες, με έμφαση στην υποστήριξη και στη χρήση αντικειμένων ή buffs.
- **Chill**
  Αμυντικές πίτσες, με κινήσεις που επιβραδύνουν ή παγώνουν τον αντίπαλο. Ιδανικές για στρατηγικό παιχνίδι.

Ο τύπος πίτσας επηρεάζει τόσο τη **συμβατότητα** με τις κινήσεις, όσο και τη **στρατηγική** της μάχης (π.χ. μία spicy πίτσα μπορεί να είναι πιο ευάλωτη σε chill επιθέσεις).

Ενέργειες (Actions)

Κάθε Pizza διαθέτει ένα σύνολο διαθέσιμων ενεργειών, όπως:

- damage1: Κίνηση που προκαλεί βασική ζημιά.
- saucyStatus: Κίνηση που προσθέτει την κατάσταση "saucy" στον εχθρό.
- clumsyStatus: Κίνηση που μειώνει την ακρίβεια του εχθρού.
- healStatus: Ενέργεια που αφαιρεί αρνητικά effects.
- item_recoverHp: Χρήση αντικειμένου για ανάκτηση HP.

Αυτές οι κινήσεις είναι ορισμένες στο Content/actions.js και ανατίθενται στην κάθε pizza με βάση τις στρατηγικές της. Το κάθε action έχει:

- type: Το είδος της ενέργειας (π.χ. damage, status, item).
- success: Λογική που περιγράφει τι συμβαίνει όταν πετύχει (π.χ. πόση ζημιά γίνεται).

- targetType: Αν στοχεύει εχθρό ή τον εαυτό.
- status: Προσθήκη ειδικών επιπτώσεων (π.χ. burned, saucy, clumsy).

Η **δομή ενεργειών είναι data-driven**, επιτρέποντας τον καθορισμό νέων κινήσεων χωρίς αλλαγή στο engine του παιχνιδιού.

Ενσωμάτωση Pizzas στον Κώδικα

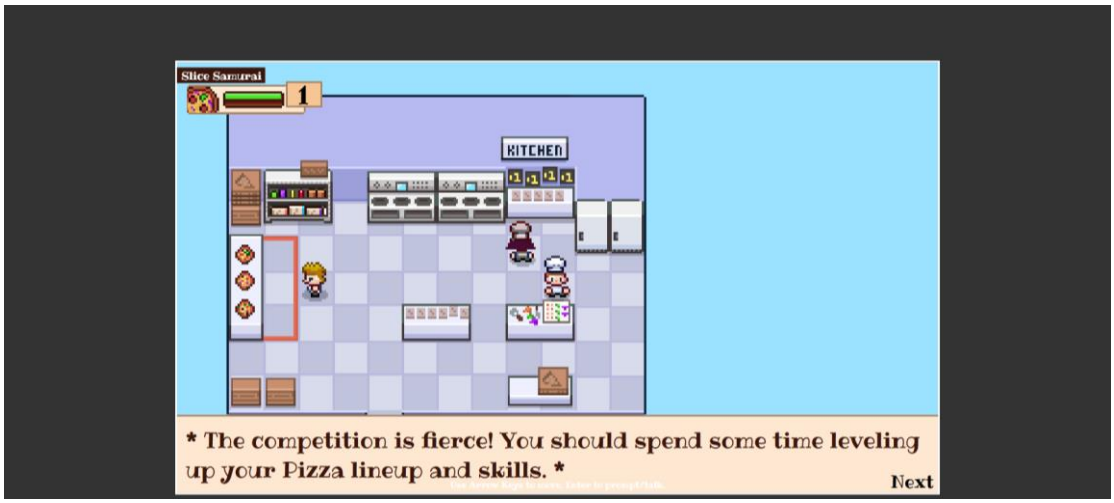
Η φόρτωση των Pizzas στον παίκτη γίνεται μέσω του PlayerState.js, όπου καθορίζονται ποιες πίτσες διαθέτει ο παίκτης, μαζί με τα stats τους (HP, XP, level κ.λπ). Αυτά τα δεδομένα περνούν στο battle σύστημα μέσω των Combatant.js και Team.js, που μετατρέπουν τις pizza-οντότητες σε ενεργούς μαχητές στο πεδίο της μάχης.

Εξέλιξη Pizzas (Level Up & XP)

Κατά τη διάρκεια των μαχών, οι πίτσες κερδίζουν XP. Με την επίτευξη συγκεκριμένων ορίων, οι πίτσες ανεβαίνουν επίπεδο (levelUp) και αυξάνουν:

- το maxHp,
- το strength,
- και ενδεχομένως ξεκλειδώνουν νέες ενέργειες.

Η εξέλιξη γίνεται είτε **αυτόματα**, είτε με βάση triggers που καθορίζονται στο PlayerState.



```
class PlayerState {
  constructor() {
    this.pizzas = {
      "p1": {
        pizzaId: "s001",
        hp: 50,
        maxHp: 50,
        xp: 0,
        maxXp: 100,
        level: 1,
        status: null,
      }
    }
    this.lineup = ["p1"];
    this.items = [
      { actionId: "item_recoverHp", instanceId: "item1" },
      { actionId: "item_recoverHp", instanceId: "item2" },
      { actionId: "item_recoverHp", instanceId: "item3" },
    ]
    this.storyFlags = {
    };
  }

  addPizza(pizzaId) {
    const newId = `p${Date.now()}`+Math.floor(Math.random() * 99999);
    this.pizzas[newId] = {
      pizzaId,
      hp: 50,
      maxHp: 50,
      xp: 0,
      maxXp: 100,
      level: 1,
      status: null,
    }
    if (this.lineup.length < 3) {
      this.lineup.push(newId)
    }
    utils.emitEvent("LineupChanged");
  }

  swapLineup(oldId, incomingId) {
    const oldIndex = this.lineup.indexOf(oldId);
    this.lineup[oldIndex] = incomingId;
    utils.emitEvent("LineupChanged");
  }

  moveToFront(futureFrontId) {
    this.lineup = this.lineup.filter(id => id !== futureFrontId);
    this.lineup.unshift(futureFrontId);
    utils.emitEvent("LineupChanged");
  }

}
window.playerState = new PlayerState();
```

Συμπεράσματα

Το σύστημα Pizzas είναι εξαιρετικά **παραμετροποιήσιμο**, εύκολο στη συντήρηση και κατάλληλο για επέκταση με νέα στοιχεία gameplay. Χάρη στην **data-first λογική** του, διαχωρίζει την περιγραφή των entities από την υλοποίηση, και παρέχει έναν απλό αλλά ισχυρό τρόπο δημιουργίας ποικιλίας στις μάχες.

Μέσω αυτών των μονάδων, το παιχνίδι αποκτά στρατηγικό βάθος, προσωπικότητα, και συνεκτικότητα.

6.2 Εχθροί (Enemies)

Οι **Εχθροί (Enemies)** στο παιχνίδι λειτουργούν ως οι αντίπαλοι των πίτσας του παίκτη σε μάχες τύπου turn-based RPG. Αποτελούν ένα βασικό συστατικό του συστήματος μάχης, με ρόλο παρόμοιο με αυτόν των Pizzas, αλλά με ιδιότητες και προγραμματισμένη συμπεριφορά που εξυπηρετεί τον σχεδιασμό κάθε σεναρίου.

Η περιγραφή και διαχείριση των εχθρών γίνεται μέσω του αρχείου `Content/enemies.js`, το οποίο περιλαμβάνει αντικείμενα JavaScript που περιγράφουν κάθε αντίπαλο, τις κινήσεις του, τις διαθέσιμες πίτσες του, και άλλες πληροφορίες.

Δομή Δεδομένων Εχθρών

Κάθε εχθρός ορίζεται στο enemies.js ως εξής:

```
window.Enemies = {
  "erio": {
    name: "Erio",
    src: "/images/characters/people/erio.png",
    pizzas: {
      "a": {
        pizzaId: "s001",
        maxHp: 50,
        level: 1,
      },
      "b": {
        pizzaId: "s002",
        maxHp: 50,
        level: 1,
      },
    }
  },
  "beth": {
    name: "Beth",
    src: "/images/characters/people/npc1.png",
    pizzas: {
      "a": {
        hp: 1,
        pizzaId: "f001",
        maxHp: 50,
        level: 1,
      },
    }
  },
```

```
  "chefRootie": {
    name: "Rootie",
    src: "/images/characters/people/secondBoss.png",
    pizzas: {
      "a": {
        pizzaId: "f002",
        maxHp: 30,
        level: 2,
      }
    }
  },
  "streetNorthBattle": {
    name: "Pizza Thug",
    src: "/images/characters/people/npc8.png",
    pizzas: {
      "a": {
        pizzaId: "s001",
        maxHp: 20,
        level: 1,
      }
    }
  },
```

```
"diningRoomBattle": {
  name: "Pizza Thug",
  src: "/images/characters/people/npc8.png",
  pizzas: {
    "a": {
      pizzaId: "s001",
      maxHp: 15,
      level: 1,
    },
    "b": {
      pizzaId: "s002",
      maxHp: 15,
      level: 1,
    }
  }
},
"streetBattle": {
  name: "Pizza Thug",
  src: "/images/characters/people/npc8.png",
  pizzas: {
    "a": {
      pizzaId: "f002",
      maxHp: 25,
      level: 1,
    }
  }
}
}
```

Κύρια χαρακτηριστικά:

- **name**: Το όνομα του εχθρού που εμφανίζεται στο UI.
- **description**: Περιγραφή του χαρακτήρα.
- **src**: Εικόνα/εικονίδιο του εχθρού.
- **pizzas**: Αντικείμενο που περιέχει τις πίτσες του εχθρού (όπως και ο παίκτης).
    - Για κάθε πίτσα καθορίζονται stats όπως hp, maxHp, level, status, xp κ.ά.
- **team** (προαιρετικό): Μπορεί να προστεθεί σε σύνθετες μάχες για να δείξει την ταυτότητα της ομάδας του αντιπάλου.

Χρήση στο σύστημα μάχης

Όταν ξεκινά μια μάχη, η σκηνή (Battle.js) φορτώνει τα δεδομένα του εχθρού με βάση το ID που έχει οριστεί στο event της OverworldMap:

```
class Battle {
  constructor({ enemy, onComplete, arena }) {

    this.enemy = enemy;
    this.onComplete = onComplete;
    this.arena = arena;

    this.combatants = {

    }

    this.activeCombatants = {
      player: null, //"player1",
      enemy: null, //"enemy1",
    }

    //Dynamically add the Player team
    window.playerState.lineup.forEach(id => {
      this.addCombatant(id, "player", window.playerState.pizzas[id])
    });
    //Now the enemy team
    Object.keys(this.enemy.pizzas).forEach(key => {
      this.addCombatant("e_"+key, "enemy", this.enemy.pizzas[key])
    })


    //Start empty
    this.items = []

    //Add in player items
    window.playerState.items.forEach(item => {
      this.items.push({
        ...item,
        team: "player"
      })
    })

    this.usedInstanceIds = {};

  }

  addCombatant(id, team, config) {
      this.combatants[id] = new Combatant({
        ...Pizzas[config.pizzaId],
        ...config,
        team,
        isPlayerControlled: team === "player"
      }, this)

      //Populate first active pizza
      this.activeCombatants[team] = this.activeCombatants[team] || id
  }
```

```
createElement() {
  this.element = document.createElement("div");
  this.element.classList.add("Battle");

  // If provided, add a CSS class for setting the arena background
  if (this.arena) {
    this.element.classList.add(this.arena);
  }

  this.element.innerHTML = (`
  <div class="Battle_hero">
    <img src="${'/images/characters/people/hero.png'}" alt="Hero" />
  </div>
  <div class="Battle_enemy">
    <img src=${this.enemy.src} alt=${this.enemy.name} />
  </div>
  `)
}

init(container) {
  this.createElement();
  container.appendChild(this.element);

  this.playerTeam = new Team("player", "Hero");
  this.enemyTeam = new Team("enemy", "Bully");

  Object.keys(this.combatants).forEach(key => {
    let combatant = this.combatants[key];
    combatant.id = key;
    combatant.init(this.element)

    //Add to correct team
    if (combatant.team === "player") {
      this.playerTeam.combatants.push(combatant);
    } else if (combatant.team === "enemy") {
      this.enemyTeam.combatants.push(combatant);
    }
  })

  this.playerTeam.init(this.element);
  this.enemyTeam.init(this.element);

  this.turnCycle = new TurnCycle({
    battle: this,
    onNewEvent: event => {
      return new Promise(resolve => {
        const battleEvent = new BattleEvent(event, this)
        battleEvent.init(resolve);
      })
    },
    onWinner: winner => {
```

```
        if (winner === "player") {
          const playerState = window.playerState;
          Object.keys(playerState.pizzas).forEach(id => {
            const playerStatePizza = playerState.pizzas[id];
            const combatant = this.combatants[id];
            if (combatant) {
              playerStatePizza.hp = combatant.hp;
              playerStatePizza.xp = combatant.xp;
              playerStatePizza.maxXp = combatant.maxXp;
              playerStatePizza.level = combatant.level;
            }
          })

          //Get rid of player used items
          playerState.items = playerState.items.filter(item => {
            return !this.usedInstanceIds[item.instanceId]
          })

          //Send signal to update
          utils.emitEvent("PlayerStateUpdated");
        }

        this.element.remove();
        this.onComplete(winner === "player");
      }
    })
    this.turnCycle.init();


  }

}
```

Από εκεί, η Battle δημιουργεί το Team του εχθρού, αρχικοποιώντας τα Combatant instances (δηλαδή τις πίτσες του εχθρού) με βάση τα δεδομένα του enemies.js.

Η αρχιτεκτονική αυτή επιτρέπει **εύκολη αναπαραγωγή εχθρών** σε διαφορετικά σημεία του παιχνιδιού, χωρίς ανάγκη αλλαγής του κώδικα μάχης.

Συμπεριφορά Εχθρών και Τεχνητή Νοημοσύνη

Οι εχθροί στο συγκεκριμένο παιχνίδι **δεν έχουν περίπλοκη AI**, αλλά ακολουθούν μια **predefined λογική** για την επιλογή ενεργειών. Συγκεκριμένα:

- Επιλέγουν μία από τις διαθέσιμες ενέργειες από το actions.js, τυχαία ή με βάση κάποιες προτεραιότητες.

```
window.Actions = {
  damage1: {
    name: "Whomp!",
    description: "Pillowy punch of dough",
    success: [
      { type: "textMessage", text: "{CASTER} uses {ACTION}!"},
      { type: "animation", animation: "spin"},
      { type: "stateChange", damage: 10}
    ]
  },
  saucyStatus: {
    name: "Tomato Squeeze",
    description: "Applies the Saucy status",
    targetType: "friendly",
    success: [
      { type: "textMessage", text: "{CASTER} uses {ACTION}!"},
      { type: "stateChange", status: { type: "saucy", expiresIn: 3 } }
    ]
  },
  clumsyStatus: {
    name: "Olive Oil",
    description: "Slippery mess of deliciousness",
    success: [
      { type: "textMessage", text: "{CASTER} uses {ACTION}!"},
      { type: "animation", animation: "glob", color: "#dafd2a" },
      { type: "stateChange", status: { type: "clumsy", expiresIn: 3 } },
      { type: "textMessage", text: "{TARGET} is slipping all around!"},
    ]
  },
  //Items
  item_recoverStatus: {
    name: "Heating Lamp",
    description: "Feeling fresh and warm",
    targetType: "friendly",
    success: [
      { type: "textMessage", text: "{CASTER} uses a {ACTION}!"},
      { type: "stateChange", status: null },
      { type: "textMessage", text: "Feeling fresh!", },
    ]
  },
  item_recoverHp: {
    name: "Parmesan",
    targetType: "friendly",
    success: [
      { type:"textMessage", text: "{CASTER} sprinkles on some {ACTION}!", },
      { type:"stateChange", recover: 10, },
      { type:"textMessage", text: "{CASTER} recovers HP!", },
    ]
  },
}
```

- Ανάλογα με τον τύπο της πίτσας (spicy, veggie, chill), η μάχη μπορεί να πάρει διαφορετική τροπή.
- Το TurnCycle.js είναι υπεύθυνο για την ενεργοποίηση της enemySubmission — δηλαδή της επιλογής ενέργειας για τον αντίπαλο.

Αν και απλό, το σύστημα αυτό είναι **επεκτάσιμο**, επιτρέποντας μελλοντικά τη δημιουργία πιο εξελιγμένων συστημάτων AI (π.χ. adaptive strategies, buffs/debuffs prioritization κ.λπ.).

Οπτική Ταυτότητα Εχθρών

Η εμφάνιση των εχθρών βασίζεται σε sprite images που βρίσκονται στον φάκελο:

/images/characters/people/

Αυτός ο οπτικός διαχωρισμός βοηθάει τον παίκτη να ξεχωρίσει κάθε boss ή NPC που εμπλέκεται σε μάχη. Κάποιοι εχθροί έχουν και **ειδικές animated εμφανίσεις** κατά τη μετάβασή τους στη μάχη, ελέγχοντας το immersion και την αφήγηση.

Σύνθεση Εχθρών (Enemy Teams)

Αν και στην πλειοψηφία των μαχών χρησιμοποιείται ένας μόνο εχθρός με μία πίτσα, το σύστημα είναι έτοιμο να υποστηρίξει **πολλαπλές πίτσες ανά εχθρό**, όπως:

```
"secondBoss": {
 name: "Big Tony",
 description: "Ο μεγαλύτερος σεφ της πόλης",
 src: "/images/characters/people/secondBoss.png",
 pizzas: {
  "a": { pizzaId: "f003", hp: 60, maxHp: 60, ... },
  "b": { pizzaId: "v002", hp: 45, maxHp: 45, ... }
 }
}
```

Αυτό προσφέρει **δυνατότητα κλιμάκωσης της δυσκολίας** αλλά και αυξανόμενη ποικιλία.

Συμπεράσματα

Το σύστημα Εχθρών είναι χτισμένο γύρω από την ίδια λογική με αυτή των πίτσας του παίκτη, προσφέροντας:

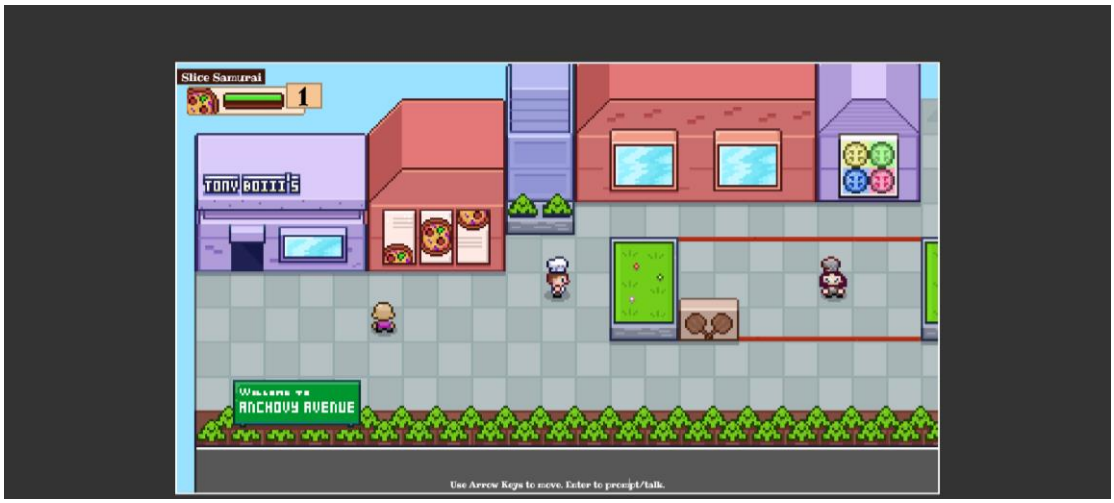
- **Συμμετρία στη μάχη** (όμοιοι μηχανισμοί για παίκτη και εχθρό),
- **Απλότητα στη διαχείριση περιεχομένου** (data-driven),
- **Ευελιξία στην προσθήκη νέων σεναρίων**.

Οι αντίπαλοι, παρόλο που τεχνικά είναι «αντικείμενα JSON», δίνουν χαρακτήρα στο παιχνίδι και υποστηρίζουν τόσο το gameplay όσο και την αφήγηση.

6.3 Παίκτης (PlayerState) & αντικείμενα

Ο παίκτης αποτελεί το κεντρικό υποκείμενο της εμπειρίας παιχνιδιού, τόσο κατά την πλοήγηση στον κόσμο (`Overworld`) όσο και κατά τη διάρκεια των μαχών (`Battle`). Η κατάστασή του (πίτσες, αντικείμενα, πρόοδος) παρακολουθείται και ελέγχεται μέσα από το αντικείμενο `PlayerState`, το οποίο βρίσκεται στον φάκελο:

/State/PlayerState.js



Το σύστημα αυτό αναλαμβάνει τον ρόλο του **state manager** του παίκτη — ελέγχει ποια πίτσα είναι ενεργή, ποια είναι τα στατιστικά τους, ποια αντικείμενα έχει στην κατοχή του, καθώς και την πρόοδο στην ιστορία.

Δομή του PlayerState

Το αρχείο PlayerState.js ορίζει μια JavaScript class που υλοποιεί την εσωτερική κατάσταση του παίκτη, η οποία μπορεί να περιλαμβάνει:

```
class PlayerState {
  constructor() {
    this.pizzas = {
      "p1": { pizzaId: "s001", hp: 50, maxHp: 50, xp: 75, maxXp: 100, level: 2, status: null },
      "p2": { pizzaId: "v001", hp: 30, maxHp: 40, xp: 20, maxXp: 100, level: 1, status: null },
    };

    this.lineup = ["p1", "p2"];

    this.items = [
      { actionId: "item_recoverHp", instanceId: "item1" },
      { actionId: "item_statusCleanse", instanceId: "item2" },
    ];
  }

  ...
```

}

- **pizzas**: Λίστα όλων των πίτσας που ανήκουν στον παίκτη. Η καθεμία έχει στατιστικά μάχης (hp, xp, level, status).
- **lineup**: Ποια πίτσα είναι ενεργή στη μάχη και σε ποια σειρά.
- **items**: Αντικείμενα που μπορεί να χρησιμοποιήσει κατά τη διάρκεια μάχης.
- **instanceId**: Κάθε αντικείμενο έχει μοναδικό ID για να μπορεί να αφαιρεθεί μετά τη χρήση.

Διατήρηση και Ανάκτηση State

Το PlayerState λειτουργεί ανεξάρτητα από την εμφάνιση ή τη μάχη. Αυτό σημαίνει ότι το state:

- Φορτώνεται και ενημερώνεται συνεχώς κατά την πλοήγηση (Overworld)
- Αποστέλλεται στο σύστημα μάχης (Battle) για να δημιουργηθεί η ομάδα του παίκτη
- Ενημερώνεται μετά από κάθε μάχη, με αλλαγές σε:
    - Πόντους ζωής
    - Πόντους εμπειρίας (XP)
    - Αναβάθμιση επιπέδων (level up)
    - Αφαίρεση/κατανάλωση αντικειμένων

Αυτό επιτρέπει **μονιμότητα** στον κόσμο του παιχνιδιού, μεταφέροντας τις συνέπειες κάθε μάχης.

Αντικείμενα (Items)

Τα αντικείμενα είναι σημαντικό μέρος της στρατηγικής, επιτρέποντας στον παίκτη να θεραπεύσει πίτσες ή να καθαρίσει αρνητικά status.

Ορίζονται στο αρχείο:

/Content/actions.js

Ένα παράδειγμα:

```
"item_recoverHp": {
  name: "Healing Sauce",
  description: "Αναπληρώνει 20 HP",
  targetType: "friendly",
  success: [
    { type: "textMessage", text: "{CASTER} έριξε σάλτσα πάνω σε {TARGET}!" },
    { type: "stateChange", recover: 20 },
  ]
}
```

- **actionId** συνδέει το αντικείμενο με την ενέργεια που εκτελεί
- **success** περιγράφει την ακολουθία των αποτελεσμάτων

Κατά τη χρήση:

- Το αντικείμενο επιλέγεται μέσω SubmissionMenu
- Εφαρμόζεται το effect
- Το αντίστοιχο instance αφαιρείται από το PlayerState.items

Αλληλεπίδραση με τη Μάχη

Όταν ξεκινά μάχη:

1. Το PlayerState δημιουργεί μια ομάδα Combatant για τον παίκτη
2. Ορίζεται η τρέχουσα lineup (ποια πίτσα είναι ενεργή)
3. Ο χρήστης μπορεί να:
    - Επιτεθεί με ενέργεια (action)
    - Χρησιμοποιήσει αντικείμενο (item)
    - Αντικαταστήσει πίτσα (replacement)
4. Μετά τη μάχη, το PlayerState ενημερώνεται με τα αποτελέσματα (π.χ. μειωμένο HP)

Επέκταση και Αποθήκευση

Το PlayerState μπορεί εύκολα να επεκταθεί με νέα δεδομένα όπως:

- Συλλογή XP για τον παίκτη (όχι μόνο για τις πίτσες)
- Προσθήκη νομισμάτων (currency)
- Καταγραφή ανακαλυφθέντων NPC ή περιοχών
- Αποθήκευση/φόρτωση από localStorage ή απομακρυσμένο backend

Σύνοψη

Το PlayerState λειτουργεί ως πυρήνας της κατάστασης του παίκτη, διαχωρίζοντας πλήρως τη λογική από την παρουσίαση. Κύρια πλεονεκτήματα:

- **Καθαρή διαχείριση δεδομένων** (data-driven model)
- **Εύκολη συντήρηση** (centralized structure)
- **Συνέπεια στο gameplay** (επιβίωση, level up, item usage)
- **Δυνατότητα επέκτασης** για μελλοντική αποθήκευση ή cloud sync

6.4 Καταστάσεις (Status Effects) και Items

Στο πλαίσιο του συστήματος μάχης, η εισαγωγή **καταστάσεων (status effects)** και **αντικειμένων (items)** εμπλουτίζει τη στρατηγική διάσταση του παιχνιδιού. Παρέχουν μηχανισμούς επιβράδυνσης, ενίσχυσης, θεραπείας ή ελέγχου, μετατρέποντας τις μάχες από απλές ανταλλαγές επιθέσεων σε πολύπλευρες τακτικές αναμετρήσεις.

1. Καταστάσεις (Status Effects)

Οι καταστάσεις αντιπροσωπεύουν προσωρινές επιδράσεις που μπορούν να επηρεάσουν ένα Combatant (είτε φιλικό είτε εχθρικό), επηρεάζοντας τη συμπεριφορά ή τα στατιστικά του.

Ορισμός:

Καταστάσεις ορίζονται ως status properties σε κάθε πίτσα μέσα από το PlayerState ή τους εχθρούς.

Παράδειγμα:

```
"p1": {
 pizzaId: "s001",
 hp: 50,
 maxHp: 50,
 status: {
  type: "saucy",
  expiresIn: 3,
 }
}
```

Λειτουργία:

Η κατάσταση περιγράφεται από δύο βασικές ιδιότητες:

- type: το είδος της κατάστασης (π.χ. saucy, clumsy, spicy)
- expiresIn: πόσοι γύροι απομένουν έως ότου λήξει

Εφαρμογή:

Καταστάσεις επηρεάζουν τη ροή της μάχης με ενέργειες που προστίθενται στην ακολουθία events μέσα στο BattleEvent.js.

Π.χ. για το status clumsy, μπορεί να εισαχθεί λογική τύπου:

- **Πιθανότητα αποτυχίας επίθεσης**
- **Μείωση damage**
- **Καθυστέρηση ενέργειας**

Ενημέρωση σε κάθε γύρο:

Με το πέρας κάθε γύρου, το expiresIn μειώνεται κατά 1. Όταν φτάσει στο 0, η κατάσταση αφαιρείται αυτόματα και εμφανίζεται αντίστοιχο TextMessage τύπου:

{ type: "textMessage", text: "{TARGET} δεν είναι πλέον clumsy!" }

2. Αντικείμενα (Items)

Τα αντικείμενα (items) είναι ενεργοποιούμενα στοιχεία που ο παίκτης μπορεί να χρησιμοποιήσει κατά τη διάρκεια της μάχης, προσφέροντας πλεονεκτήματα, όπως:

- Θεραπεία HP
- Καθαρισμός καταστάσεων
- Βραχυχρόνια ενίσχυση

Ορισμός:

Τα items είναι καταγεγραμμένα στο:

/Content/actions.js

Παράδειγμα:

```
"item_statusCleanse": {
  name: "Bubble Tea",
  description: "Αφαιρεί αρνητικές καταστάσεις",
  targetType: "friendly",
  success: [
    { type: "textMessage", text: "{CASTER} έδωσε Bubble Tea στον {TARGET}!" },
    { type: "stateChange", status: null },
  ]
}
```

Κύρια χαρακτηριστικά:

- **actionId**: μοναδικό id της ενέργειας του item
- **success**: λίστα ενεργειών που εκτελούνται όταν χρησιμοποιείται
- **targetType**: καθορίζει αν εφαρμόζεται σε φίλους ή εχθρούς

Χρήση:

- Ο παίκτης επιλέγει "Χρήση Αντικειμένου" στο SubmissionMenu
- Εμφανίζεται λίστα διαθέσιμων items (βάσει του PlayerState.items)
- Το item εφαρμόζει το success event
- Το instance αφαιρείται από το inventory του παίκτη

Παραδείγματα άλλων Items:

```
"item_recoverHp": {
  name: "Healing Sauce",
  description: "Αναπληρώνει 20 HP",
  success: [
    { type: "textMessage", text: "{CASTER} έριξε σάλτσα πάνω στον {TARGET}!" },
    { type: "stateChange", recover: 20 }
  ]
}
```

3. Συνδυαστική Δύναμη Status + Items

Η δυναμική του παιχνιδιού ενισχύεται ιδιαίτερα όταν τα status effects συνδυάζονται με τα items:

- Ένα item_statusCleanse γίνεται χρήσιμο μόνο αν υπάρχει αρνητικό status.
- Μια πικάντικη επίθεση μπορεί να προκαλεί burning status, οδηγώντας σε ανάγκη για αντικείμενο κατάστασης.
- Το σύστημα αυτό δημιουργεί **στρατηγική ισορροπία** ανάμεσα σε επίθεση και υποστήριξη.

Συντήρηση και Επέκταση

Το μοντέλο status και items είναι **δεδομενοκεντρικό (data-driven)**, επιτρέποντας εύκολη επέκταση:

- Προσθήκη νέων status effects με απλό JSON schema
- Δημιουργία σύνθετων items με ακολουθίες πολλαπλών stateChange και animation events
- Σύνδεση με buffs/debuffs ή ειδικές συνθήκες νίκης

Σύνοψη

Η υλοποίηση των **καταστάσεων (status effects)** και των **αντικειμένων (items)** λειτουργεί ως κρίσιμος πυλώνας βάθους και στρατηγικής στο σύστημα μάχης. Μέσω ενός data-driven και modular συστήματος:

- Παρέχεται ευκολία διαχείρισης και προσθήκης νέου περιεχομένου
- Ο παίκτης ενθαρρύνεται να προσαρμόζει τακτικές ανάλογα με την κατάσταση
- Δημιουργείται ένα πιο πλούσιο, engaging gameplay loop

Κεφάλαιο 7ο: Μηχανισμοί UI & Χρήστη

Η διεπαφή χρήστη (User Interface - UI) αποτελεί κρίσιμο τμήμα της εμπειρίας του παίκτη, καθώς λειτουργεί ως το βασικό σημείο επικοινωνίας μεταξύ του ανθρώπου και του λογισμικού. Στην παρούσα εφαρμογή, η σχεδίαση του UI δίνει ιδιαίτερη έμφαση στη **διαδραστικότητα**, τη **σαφήνεια επιλογών** και την **οπτική ανάδραση (feedback)**, με στόχο την ενίσχυση της κατανόησης των μηχανισμών και της στρατηγικής λήψης αποφάσεων του παίκτη.

Η υλοποίηση των στοιχείων UI γίνεται κυρίως με τη χρήση **Vanilla JavaScript** και **CSS**, χωρίς τη μεσολάβηση κάποιου framework. Οι μηχανισμοί ελέγχου του παίκτη και η απόδοση της πληροφορίας οργανώνονται σε **modular κλάσεις**, οι οποίες παρέχουν αφαιρετικά επίπεδα για κοινά μοτίβα, όπως επιλογές μενού, εμφανίσεις κειμένων και animation feedback.

Οι κύριες περιοχές του UI περιλαμβάνουν:

- **Δομές μενού** για επιλογή ενεργειών, αλλαγή μαχητών ή χρήση αντικειμένων.
- **Κινούμενα μηνύματα (TextMessages)** που καθοδηγούν ή ενημερώνουν τον παίκτη.
- **Σύστημα animation feedback**, το οποίο ενημερώνει οπτικά για την επιτυχία, αποτυχία ή επίδραση μιας ενέργειας.
- **Χειρισμός πληκτρολογίου** για επιλογές χωρίς χρήση ποντικιού, ενισχύοντας τη ρετρό RPG εμπειρία.

Η λειτουργία του UI είναι **στενά συνδεδεμένη με τον μηχανισμό μάχης**, καθώς τα μενού και οι ενέργειες καθορίζουν την είσοδο του παίκτη στο TurnCycle. Επιπλέον, το feedback που επιστρέφεται στον παίκτη (μέσω animations ή μηνυμάτων) ενισχύει την αντιληπτική κατανόηση της μάχης και τη ροή του gameplay.

Σε αυτό το κεφάλαιο, θα εξετάσουμε αναλυτικά τους επιμέρους μηχανισμούς του UI και τη μεταξύ τους διασύνδεση:

- **5.1** Τις βασικές κλάσεις μενού και τις δυνατότητες που προσφέρουν.
- **5.2** Το σύστημα μηνυμάτων και animations για καθοδήγηση του παίκτη.
- **5.3** Τον χειρισμό επιλογών του παίκτη και τη διασύνδεση με το game logic.
- **5.4** Την επικοινωνία του UI με το σύστημα animation και τις μάχες.

Η σχεδιαστική προσέγγιση δίνει έμφαση στην **αμεσότητα** και τη **διαφάνεια** — χαρακτηριστικά απαραίτητα σε παιχνίδια στρατηγικής και τακτικής, όπου η λήψη σωστής απόφασης εξαρτάται από την ποιότητα της πληροφορίας που εμφανίζεται στον χρήστη.

7.1 Κλάσεις για μενού (KeyboardMenu, ReplacementMenu)

Στο πλαίσιο της αλληλεπίδρασης του παίκτη με το σύστημα μάχης και γενικότερα με το παιχνίδι, βασικός ρόλος αποδίδεται στις **δομές μενού**. Οι κλάσεις `KeyboardMenu` και

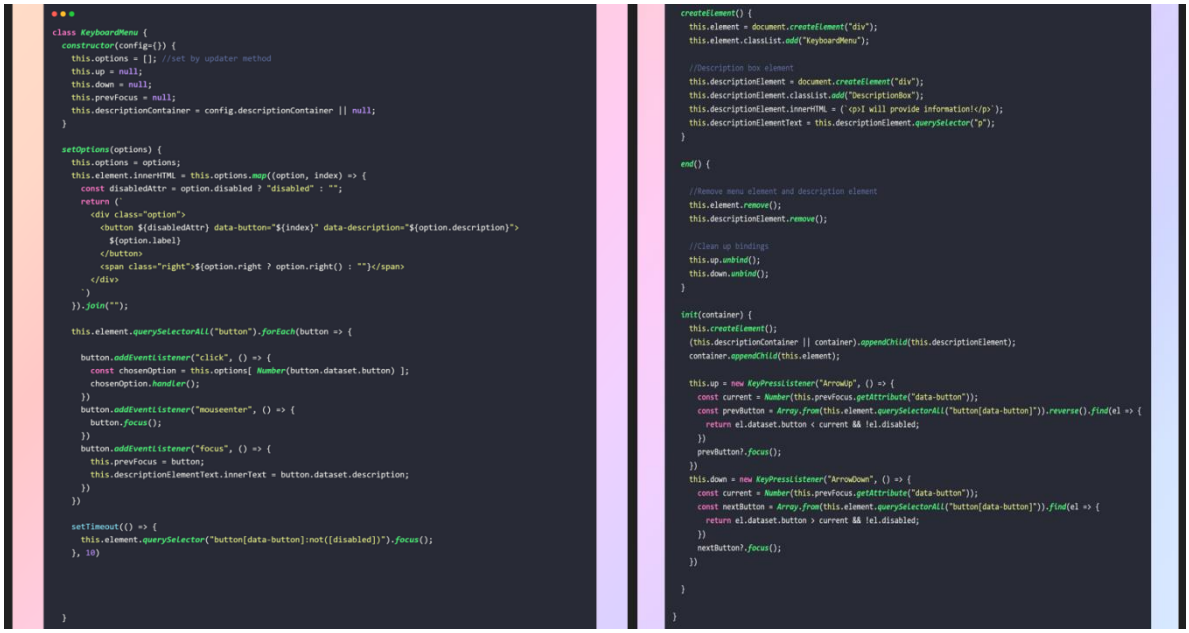
`ReplacementMenu` επιτελούν αυτόν τον σκοπό, επιτρέποντας την **πλοήγηση μέσω πληκτρολογίου**, την **επιλογή ενεργειών**, καθώς και τη **διαχείριση ομάδας και αντικειμένων**.

Ας δούμε κάθε μία ξεχωριστά:

KeyboardMenu

Η KeyboardMenu είναι μια γενικού σκοπού κλάση μενού, η οποία λειτουργεί ως **ευέλικτη βάση για διαφορετικά μενού επιλογών**. Χρησιμοποιείται επανειλημμένα σε διάφορες καταστάσεις, όπως:

- Επιλογή ενέργειας (επίθεση, αντικείμενο, αλλαγή μαχητή)
- Πλοήγηση σε υπομενού (όπως επιλογή αντικειμένου ή στόχου)
- Επιλογή σε context-sensitive περιβάλλοντα (π.χ. κατά τη διάρκεια ή μετά από ένα event)



```
class KeyboardMenu {
  constructor(config={}) {
    this.options = []; //set by updater method
    this.up = null;
    this.down = null;
    this.prevFocus = null;
    this.descriptionContainer = config.descriptionContainer || null;
  }

  setOptions(options) {
    this.options = options;
    this.element.innerHTML = this.options.map((option, index) => {
      const disabledAttr = option.disabled ? "disabled" : "";
      return (`
        <div class="option">
          <button ${disabledAttr} data-button="${index}" data-description="${option.description}">
            ${option.label}
          </button>
          <span class="right">${option.right ? option.right() : ""}</span>
        </div>
      `)
    }).join("");

    this.element.querySelectorAll("button").forEach(button => {

      button.addEventListener("click", () => {
        const chosenOption = this.options[ Number(button.dataset.button) ];
        chosenOption.handler();
      })
      button.addEventListener("mouseenter", () => {
        button.focus();
      })
      button.addEventListener("focus", () => {
        this.prevFocus = button;
        this.descriptionElementText.innerText = button.dataset.description;
      })
    })

    setTimeout(() => {
      this.element.querySelector("button[data-button]:not([disabled])").focus();
    }, 10)



  }
```

```
  createElement() {
    this.element = document.createElement("div");
    this.element.classList.add("KeyboardMenu");

    //Description box element
    this.descriptionElement = document.createElement("div");
    this.descriptionElement.classList.add("DescriptionBox");
    this.descriptionElement.innerHTML = (`<p>I will provide information!</p>`);
    this.descriptionElementText = this.descriptionElement.querySelector("p");
  }

  end() {

    //Remove menu element and description element
    this.element.remove();
    this.descriptionElement.remove();

    //Clean up bindings
    this.up.unbind();
    this.down.unbind();
  }

  init(container) {
    this.createElement();
    (this.descriptionContainer || container).appendChild(this.descriptionElement);
    container.appendChild(this.element);

    this.up = new KeyPressListener("ArrowUp", () => {
      const current = Number(this.prevFocus.getAttribute("data-button"));
      const prevButton = Array.from(this.element.querySelectorAll("button[data-button]")).reverse().find(el => {
        return el.dataset.button < current && !el.disabled;
      })
      prevButton?.focus();
    })
    this.down = new KeyPressListener("ArrowDown", () => {
      const current = Number(this.prevFocus.getAttribute("data-button"));
      const nextButton = Array.from(this.element.querySelectorAll("button[data-button]")).find(el => {
        return el.dataset.button > current && !el.disabled;
      })
      nextButton?.focus();
    })

  }
}
```

Κύρια χαρακτηριστικά της:

- **Δομή HTML**: Δημιουργεί δυναμικά ένα DOM στοιχείο με ul > li, με κάθε στοιχείο να αντιστοιχεί σε μία επιλογή.
- **Υποστήριξη για custom callbacks**: Κάθε επιλογή συνοδεύεται από action handler (handler) που εκτελείται όταν ο παίκτης την επιλέξει.
- **Ανταπόκριση σε πληκτρολόγιο**: Χρησιμοποιεί την κλάση KeyPressListener για να καταγράφει ArrowUp, ArrowDown, Enter και Escape.
- **Οπτική ανάδραση**: Ο ενεργός δείκτης (active index) ενημερώνεται και επισημαίνεται κάθε φορά που αλλάζει επιλογή.
- **Υποστήριξη υπομενού**: Εύκολη ενσωμάτωση υπο-επιλογών για nested λειτουργίες.

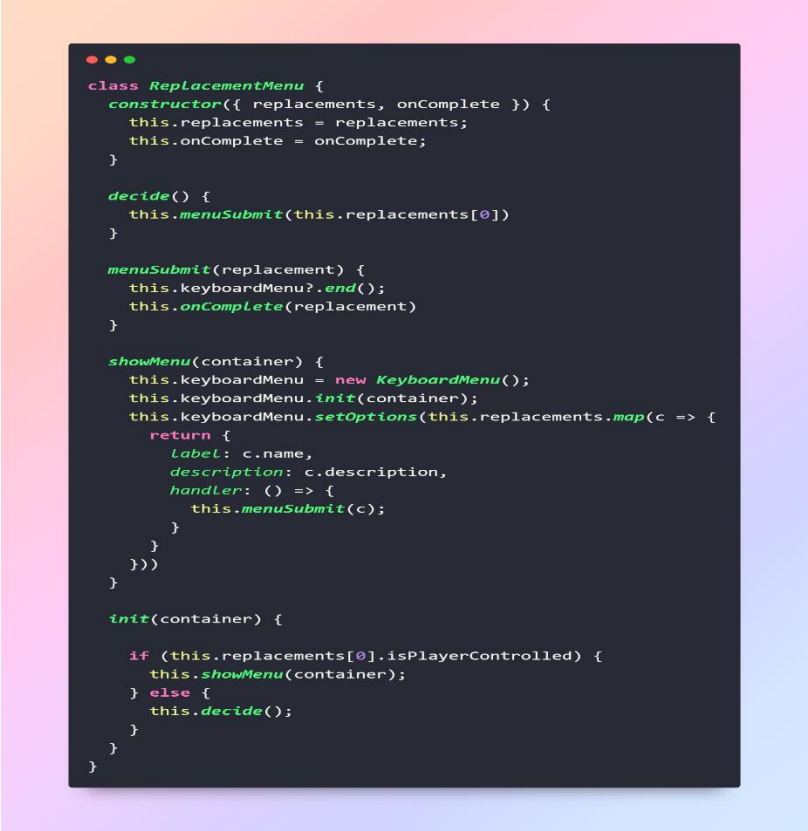
Παράδειγμα χρήσης:

```
this.keyboardMenu.setOptions([
  { label: "Attack", handler: () => { ... } },
  { label: "Use Item", handler: () => { ... } },
  { label: "Switch", handler: () => { ... } },
]);
```

Ο modular σχεδιασμός της επιτρέπει να χρησιμοποιείται τόσο σε μάχες όσο και εκτός αυτών (π.χ. σε crafting μενού ή pause screens).

ReplacementMenu

Η ReplacementMenu είναι μια πιο εξειδικευμένη κλάση, η οποία χρησιμοποιείται **αποκλειστικά στη μάχη**, όταν ο παίκτης καλείται να **αντικαταστήσει έναν μαχητή που ηττήθηκε** ή όταν θέλει να κάνει αλλαγή προληπτικά.



```
class ReplacementMenu {
  constructor({ replacements, onComplete }) {
    this.replacements = replacements;
    this.onComplete = onComplete;
  }

  decide() {
    this.menuSubmit(this.replacements[0])
  }

  menuSubmit(replacement) {
    this.keyboardMenu?.end();
    this.onComplete(replacement)
  }

  showMenu(container) {
    this.keyboardMenu = new KeyboardMenu();
    this.keyboardMenu.init(container);
    this.keyboardMenu.setOptions(this.replacements.map(c => {
      return {
        label: c.name,
        description: c.description,
        handler: () => {
          this.menuSubmit(c);
        }
      }
    }))
  }

  init(container) {

    if (this.replacements[0].isPlayerControlled) {
      this.showMenu(container);
    } else {
      this.decide();
    }
  }
}
```

Χαρακτηριστικά:

- **Φιλτραρισμένες επιλογές**: Εμφανίζει μόνο τις μονάδες του παίκτη που **δεν είναι ήδη ενεργές και δεν είναι K.O.**.
- **Οπτική πληροφόρηση**: Για κάθε επιλογή εμφανίζεται το όνομα της πίτσας, τα HP της, και ενδεχομένως κάποιο status effect.
- **Επαναχρησιμοποιεί την KeyboardMenu** ως βάση**, αλλά εισάγει custom λογική για το filtering και rendering των διαθέσιμων μονάδων.

Παράδειγμα δημιουργίας:

```
new ReplacementMenu({
  team: playerTeam,
  onComplete: resolveReplacement
});
```

Αυτό το μενού προβάλλει τις διαθέσιμες "πίτσες" του παίκτη και επιτρέπει την άμεση αλλαγή χαρακτήρα μέσα στον TurnCycle.

Σχέση με το TurnCycle και τα BattleEvents

Και τα δύο αυτά μενού **συνδέονται άμεσα με τον κύκλο μάχης (TurnCycle)**. Συγκεκριμένα:

- Ο παίκτης καλείται να αλληλεπιδράσει με αυτά **κατά το γύρο του**, ώστε να επιλέξει ενέργεια.
- Τα SubmissionMenus (που είναι composite δομές) συχνά χρησιμοποιούν KeyboardMenu και ReplacementMenu ως μέρη της διαδικασίας συλλογής input.
- Η επιλογή που επιστρέφουν **τροφοδοτεί το event queue**, το οποίο εκτελεί τη σχετική ενέργεια μέσω των BattleEvent instances.

Σημασία στην εμπειρία του χρήστη

Η υλοποίηση αυτών των κλάσεων:

- Διαχωρίζει πλήρως το **UI logic** από το **game logic**.
- Επιτρέπει την **εύκολη επέκταση** με νέες επιλογές ή παραλλαγές μενού.
- Διατηρεί την εμπειρία **πληκτρολογιοκεντρική**, ενισχύοντας τον ρετρό χαρακτήρα του παιχνιδιού.

7.2 TextMessage & animation feedback

Η ενότητα **TextMessage & animation feedback** αφορά τη διαχείριση των **μηχανισμών οπτικής και λεκτικής ανατροφοδότησης** προς τον παίκτη, κυρίως μέσα από διαλόγους και κινούμενες ενδείξεις (animations). Οι μηχανισμοί αυτοί επιτελούν διπλό ρόλο: αφενός **ενημερώνουν** τον παίκτη για όσα συμβαίνουν στο παιχνίδι (σε μάχες, events, εξερεύνηση), και αφετέρου **εμπλουτίζουν την αισθητική εμπειρία**, προσδίδοντας χαρακτήρα και βάθος στο gameplay.

Κλάση: TextMessage

```
class TextMessage {
  constructor({ text, onComplete }) {
    this.text = text;
    this.onComplete = onComplete;
    this.element = null;
  }

  createElement() {
    //Create the element
    this.element = document.createElement("div");
    this.element.classList.add("TextMessage");

    this.element.innerHTML = (`
      <p class="TextMessage_p"></p>
      <button class="TextMessage_button">Next</button>
    `)

    //Init the typewriter effect
    this.revealingText = new RevealingText({
      element: this.element.querySelector(".TextMessage_p"),
      text: this.text
    })

    this.element.querySelector("button").addEventListener("click", () => {
      //Close the text message
      this.done();
    });

    this.actionListener = new KeyPressListener("Enter", () => {
      this.done();
    })

  }

  done() {

    if (this.revealingText.isDone) {
      this.element.remove();
      this.actionListener.unbind();
      this.onComplete();
    } else {
      this.revealingText.warpToDone();
    }
  }

  init(container) {
    this.createElement();
    container.appendChild(this.element);
    this.revealingText.init();
  }

}
```

Η TextMessage είναι μία αυτόνομη κλάση που χρησιμοποιείται για την εμφάνιση **μηνυμάτων κειμένου στην οθόνη**, κυρίως κατά τη διάρκεια:

- **Μαχών** (π.χ. "Ο Erio εξαπέλυσε τη Spicy Blast!")
- **Cutscenes** (π.χ. "Ένας άγνωστος εμφανίζεται μπροστά σου...")
- **Αλληλεπιδράσεων με NPCs**
- **Κατάστασης παιχνιδιού** (π.χ. “Το παιχνίδι αποθηκεύτηκε”)

Χαρακτηριστικά:

- Δημιουργεί δυναμικά ένα DOM στοιχείο με HTML markup, με styling που ορίζεται μέσω CSS.
- Εμφανίζει το κείμενο με **εφέ γραφομηχανής (typewriter effect)**, το οποίο υλοποιείται μέσω της κλάσης RevealingText.
- Παρέχει οπτική παρότρυνση στον παίκτη να προχωρήσει (π.χ. “Press Enter to continue...”).
- Υποστηρίζει **callback** όταν ολοκληρωθεί η προβολή, για συνέχιση της ροής του παιχνιδιού.

Παράδειγμα χρήσης:

```
new TextMessage({
  text: "Ο αντίπαλος pizza ninja εμφανίζεται!",
  onComplete: () => {
    // προχώρα στο επόμενο event
  }
}).init(container);
```

RevealingText: Γραφομηχανή εφέ

```
class RevealingText {
  constructor(config) {
    this.element = config.element;
    this.text = config.text;
    this.speed = config.speed || 60;

    this.timeout = null;
    this.isDone = false;
  }

  revealOneCharacter(list) {
    const next = list.splice(0,1)[0];
    next.span.classList.add("revealed");

    if (list.length > 0) {
      this.timeout = setTimeout(() => {
        this.revealOneCharacter(list)
      }, next.delayAfter)
    } else {
      this.isDone = true;
    }
  }

  warpToDone() {
    clearTimeout(this.timeout);
    this.isDone = true;
    this.element.querySelectorAll("span").forEach(s => {
      s.classList.add("revealed");
    })
  }

  init() {
    let characters = [];
    this.text.split("").forEach(character => {

      //Create each span, add to element in DOM
      let span = document.createElement("span");
      span.textContent = character;
      this.element.appendChild(span);

      //Add this span to our internal state Array
      characters.push({
        span,
        delayAfter: character === " " ? 0 : this.speed
      })
    })

    this.revealOneCharacter(characters);

  }

}
```

Η κλάση RevealingText χειρίζεται τη **βήμα-βήμα αποκάλυψη χαρακτήρων** (typing effect). Κάθε χαρακτήρας εμφανίζεται με καθυστέρηση μερικών milliseconds, δίνοντας μια πιο ζωντανή αίσθηση αφήγησης.

Παράμετροι:

- text: Το πλήρες μήνυμα προς εμφάνιση.
- element: Το HTML στοιχείο στο οποίο θα τοποθετηθεί το κείμενο.
- speed: Καθορίζει τον ρυθμό αποκάλυψης (χαμηλότερο = πιο αργό).

Αυτή η υλοποίηση εμπλουτίζει την εμπειρία, δημιουργώντας προσδοκία και ρυθμό στην παρουσίαση.

Οπτικό animation feedback (πέρα από το κείμενο)

Εκτός από τα μηνύματα κειμένου, το παιχνίδι αξιοποιεί ποικίλες **οπτικές ενδείξεις** ώστε να υποστηρίξει τις δράσεις των παικτών ή των εχθρών:

Παραδείγματα:

- **Δόνηση μαχητή** όταν δέχεται επίθεση (shake effect).
- **Αναλαμπή χρώματος** κατά τη λήψη ζημιάς (γρήγορο flash σε λευκό ή κόκκινο).
- **Fade in/out** για transitions ή αλλαγές σκηνών (SceneTransition).
- **Opacity και scaling transitions** για menus ή overlays.
- **Αλλαγή sprites ή frames** όταν ο μαχητής βρίσκεται υπό status effect (π.χ. “spicy” animation).

Αυτές οι κινήσεις **υλοποιούνται κυρίως με JavaScript** (σε canvas ή DOM) και **συμπληρωματικά με CSS** για απλά transitions (π.χ. fade-in/out σε κείμενο ή overlays).

Σημασία του feedback στην εμπειρία χρήστη

Η ανατροφοδότηση (feedback), είτε με κείμενο είτε με κινούμενα γραφικά, είναι κρίσιμη για:

- **Την κατανόηση των γεγονότων** από τον παίκτη (τι έγινε, από ποιον, με ποιο αποτέλεσμα).
- **Την συναισθηματική εμπλοκή** (π.χ. αγωνία, έκπληξη).
- **Την οπτική ικανοποίηση** που κάνει το παιχνίδι πιο ευχάριστο και "ζωντανό".

Η συνδυασμένη χρήση TextMessage, RevealingText, και animation effects πετυχαίνει ένα ικανοποιητικό επίπεδο εμπειρίας, ακόμη και χωρίς τη χρήση framework ή graphic libraries.

7.3 Χειρισμός επιλογών παίκτη

Ο χειρισμός των επιλογών του παίκτη είναι ένα από τα πιο κρίσιμα στοιχεία διαδραστικότητας σε κάθε παιχνίδι. Στο συγκεκριμένο έργο, η προσέγγιση βασίζεται σε **DOM-based UI menus**, **input listeners** και ένα σύστημα **callback-driven χειρισμού επιλογών**, που επιτρέπει στον παίκτη να λαμβάνει αποφάσεις στις μάχες, στα μενού και στις διαδραστικές σκηνές.

Κεντρικός στόχος

Να δοθεί στον παίκτη **έλεγχος** πάνω σε κάθε ενέργεια, μέσα από:

- **Απλά και καθαρά μενού επιλογών**
- **Κουμπιά πληκτρολογίου (keyboard input)**
- **Προσαρμόσιμες ενέργειες σε πραγματικό χρόνο (turn-based λογική)**

Input Handling με KeyPressListener & DirectionInput

Ο παίκτης χειρίζεται τις επιλογές του μέσω του πληκτρολογίου (βελάκια, Enter, Esc).

DirectionInput.js:

- Χρησιμοποιείται κυρίως στην **εξερεύνηση** του overworld.
- Εντοπίζει πατήματα των βελών (ArrowUp, ArrowDown, κ.λπ.)
- Παρέχει κατεύθυνση στον χαρακτήρα.

KeyPressListener.js:

- Ειδικεύεται στο **ανίχνευση εντολών** τύπου "Επιβεβαίωση", "Ακύρωση".
- Συνήθως περιμένει ένα Enter, Escape, ή custom πλήκτρο και πυροδοτεί **callback**.

Επιλογή μέσω μενού – Συστήματα UI

Η διεπαφή χρήστη είναι βασισμένη σε **menu components**, όπως:

- KeyboardMenu
- SubmissionMenu
- ReplacementMenu

Αυτά τα μενού:

- Εμφανίζουν επιλογές σε λίστα (actions, αντικείμενα, allies).
- Επιτρέπουν την **πλοήγηση με βελάκια** και επιβεβαίωση με Enter.
- Επιστρέφουν callback με την επιλογή του χρήστη, ώστε το παιχνίδι να προχωρήσει.

Παράδειγμα χρήσης:

```
const menu = new KeyboardMenu();
menu.init({
  options: [
    { label: "Attack", description: "Κάνε επίθεση", handler: () => { ... } },
    { label: "Item", description: "Χρησιμοποίησε αντικείμενο", handler: () => { ... } },
  ]
});
```

Η σχεδίαση ακολουθεί **μοτίβο παρατηρητή** (observer pattern): η επιλογή πυροδοτεί γεγονός και το σύστημα παρακολουθεί το callback για να συνεχίσει.

Στην πράξη: Battle Turn επιλογές

Κατά τη διάρκεια των μαχών:

1. Ο TurnCycle ενημερώνει ότι είναι η σειρά του παίκτη.
2. Το SubmissionMenu ανοίγει και περιμένει input.
3. Ο παίκτης επιλέγει π.χ. "Spicy Kick" και στόχο.
4. Το αποτέλεσμα αποστέλλεται ως αντικείμενο:

{

action: selectedAction,

target: selectedTarget,

}

5. Το BattleEvent ερμηνεύει την επιλογή και εκτελεί το αποτέλεσμα.

Αυτό το σχήμα διατηρεί τον **έλεγχο στα χέρια του παίκτη** σε κάθε γύρο, ενώ το σύστημα παραμένει modular και επεκτάσιμο.

Πλεονεκτήματα της αρχιτεκτονικής

- **Αποσυσχέτιση UI και game logic**: Τα μενού δεν γνωρίζουν τι θα γίνει μετά — στέλνουν μόνο την επιλογή.
- **Ευκολία προσθήκης νέων ενεργειών**: Κάθε handler είναι ξεχωριστή συνάρτηση.
- **Υποστήριξη πολλαπλών ειδών input (μελλοντικά gamepad, touch)**.

UX παρατηρήσεις

- Οι επιλογές εμφανίζονται με **οπτική υπογράμμιση** (highlight).
- Χρησιμοποιείται **animation όταν ο χρήστης κάνει focus** σε επιλογή.
- Το Enter επιβεβαιώνει, ενώ το Escape ακυρώνει και επιστρέφει στο προηγούμενο μενού (π.χ. από επιλογή item σε επιλογή action).

Συνοψίζοντας

Ο χειρισμός των επιλογών του παίκτη στο παιχνίδι στηρίζεται σε:

- Καθαρό keyboard input system (με custom listeners)
- Modular menu components με λογική callbacks
- Σαφή αποτύπωση των διαθέσιμων ενεργειών

Αυτό εξασφαλίζει ευκολία χρήσης, καθαρή αρχιτεκτονική, και ένα διαδραστικό και προβλέψιμο gameplay loop.

7.4 Επικοινωνία με BattleAnimation & Animations

Η αισθητική ανατροφοδότηση (visual feedback) αποτελεί θεμελιώδη συνιστώσα της εμπειρίας ενός παίκτη. Στο πλαίσιο της παρούσας εφαρμογής, η **αλληλεπίδραση μεταξύ των UI μηχανισμών και των animation συστημάτων** διαδραματίζει κρίσιμο ρόλο στην ενίσχυση της αντιληπτής επίδρασης των ενεργειών και στην κατανόηση της κατάστασης της μάχης.

Αυτό επιτυγχάνεται μέσω της επικοινωνίας μεταξύ των στοιχείων **BattleEvent**, **BattleAnimation**, και DOM-based **CSS animations**.

Σκοπός των BattleAnimations

Η κλάση BattleAnimations.js περιέχει **animation προτύπων** (presets), που εκτελούνται κάθε φορά που πραγματοποιείται μία ενέργεια, π.χ. επίθεση, λήψη ζημιάς, εξαφάνιση, κ.ά.

Ο βασικός της στόχος είναι να:

- **Ενισχύσει την οπτική κατανόηση** του τι συνέβη στη μάχη
- **Συνδέσει χρονικά την ενέργεια με το αποτέλεσμά της**
- Δώσει **αίσθηση ροής και δράσης**

Επικοινωνία BattleEvent → BattleAnimation

Η σύνδεση ξεκινά από την BattleEvent, η οποία:

1. Λαμβάνει την ενέργεια του παίκτη (π.χ. "Spicy Kick").
2. Εκτελεί τη λογική εφαρμογής (π.χ. μείωση HP του στόχου).
3. Ενεργοποιεί το κατάλληλο animation, π.χ.:

await BattleAnimations.spin(event);

Οι animation functions είναι **async**, ώστε να μπορέσουν να συνδεθούν με την ακολουθία των BattleEvents. Έτσι, διασφαλίζεται ότι η εκτέλεση της επόμενης φάσης δεν ξεκινά πριν ολοκληρωθεί το animation.

Είδη Animations

1. Attack Animation

Συνήθως περιλαμβάνει:

- Κίνηση sprite προς τον αντίπαλο
- Αναπήδηση του στόχου (δείχνει «χτύπημα»)
- Flash ή shake effect

```
static async spin(event, onComplete) {
  const element = event.caster.pizzaElement;
  element.classList.add("battle-spin");
  await utils.wait(1000);
  element.classList.remove("battle-spin");
  onComplete();
}
```

2. Damage Feedback

Προστίθεται CSS class (π.χ. damage-blink) στον combatant:

- Αναβοσβήνει κόκκινο
- Τρέμει στιγμιαία

3. Status Effects

- Μπορεί να εμφανίζεται ένα εικονίδιο ή χρώμα
- Π.χ. "spicy" μπορεί να δίνει μια πορτοκαλί παλμική κίνηση

4. Fade-out / KO Animation

Όταν ο χαρακτήρας πέφτει:

- Σταδιακή εξαφάνιση

Διπλή ροή: Animation → Game Logic

Αν και συχνά η λογική προηγείται του animation (damage → visual), σε ορισμένες περιπτώσεις υπάρχει το αντίθετο:

- Το animation **προετοιμάζει** μια ενέργεια, και η λογική **ακολουθεί** (π.χ. δράση φόρτισης)
- Ένα animation μπορεί να καθυστερήσει ή να ακυρώσει μια ενέργεια (π.χ. stun με animation που σταματά την κίνηση)

Σύνδεση με DOM & CSS

Τα animations υλοποιούνται με:

- **CSS classes** (π.χ. .battle-spin, .battle-blink)
- **@keyframes** (για κινούμενα οπτικά εφέ)
- **JavaScript timers** με setTimeout/await utils.wait() για συγχρονισμό

Αυτό επιτρέπει:

- Ελαφριά υλοποίηση χωρίς Canvas
- Ευελιξία και επέκταση με νέα εφέ
- Επαναχρησιμοποίηση animation patterns (DRY)

Debugging και Testing

Κατά την ανάπτυξη των animations:

- Προστέθηκαν console.log() εντός των animation functions για έλεγχο ροής
- Οι CSS τάξεις ελέγχθηκαν μέσω dev tools
- Τα await εξασφάλισαν σειριακή εκτέλεση — σημαντικό για σταθερότητα στη ροή μάχης

Συνοψίζοντας

Η επικοινωνία μεταξύ των BattleAnimations και των μηχανισμών UI:

- Επιτυγχάνει **συγχρονισμό μεταξύ δράσης και αποτελέσματος**
- Προσφέρει **οπτική ανατροφοδότηση** υψηλής ποιότητας
- Ενισχύει την **αίσθηση επιρροής** του παίκτη στο παιχνίδι
- Χρησιμοποιεί **ασύγχρονες ροές (async/await)** και **DOM animation patterns** για αποτελεσματικότητα

Αποτελεί ένα από τα πλέον κρίσιμα σημεία όπου η λογική (logic) και η παρουσίαση (presentation) συνεργάζονται στενά για μια ενιαία εμπειρία χρήστη.

Κεφάλαιο 8ο: Το Μέλλον Των Video Games

Η εξέλιξη των video games δεν περιορίζεται μόνο στην τεχνολογία και το gameplay· επεκτείνεται και στις κοινωνικές, ηθικές και πολιτισμικές πτυχές που συνδέονται με την αλληλεπίδραση των παικτών και το περιεχόμενο των παιχνιδιών. Αυτό το κεφάλαιο εξετάζει πώς τα παιχνίδια επηρεάζουν και επηρεάζονται από την κοινωνία, αναλύοντας την κοινωνική διάσταση, την έννοια του multiplayer και τις ηθικές και πολιτισμικές προεκτάσεις της δημιουργίας και κατανάλωσης παιχνιδιών. Η προσέγγιση αυτή βοηθά στην κατανόηση της ευρύτερης επίδρασης των video games πέρα από την απλή ψυχαγωγία, επισημαίνοντας ταυτόχρονα την ευθύνη των δημιουργών και των κοινοτήτων παικτών.

8.1 Τεχνολογικές Τάσεις

Η σύγχρονη βιομηχανία των βιντεοπαιχνιδιών εξελίσσεται ραγδαία, με την τεχνολογία να διαμορφώνει συνεχώς τον τρόπο που δημιουργούνται, διανέμονται και βιώνονται τα παιχνίδια. Μία από τις πλέον εμφανείς τάσεις είναι η υιοθέτηση της **εικονικής και επαυξημένης πραγματικότητας (VR/AR)**, η οποία προσφέρει στους παίκτες μια καθηλωτική εμπειρία. Στα VR παιχνίδια, οι παίκτες εισέρχονται σε πλήρως τρισδιάστατους κόσμους όπου η κίνηση και οι φυσικές αλληλεπιδράσεις είναι κρίσιμες, ενώ τα AR παιχνίδια ενσωματώνουν στοιχεία του πραγματικού περιβάλλοντος, δημιουργώντας μια σύνθεση ψηφιακής και πραγματικής πραγματικότητας.

Μια δεύτερη σημαντική τάση είναι το **cloud gaming**, που επιτρέπει την εκτέλεση παιχνιδιών σε απομακρυσμένους servers αντί σε τοπικές κονσόλες ή υπολογιστές. Αυτό αφαιρεί περιορισμούς σχετικά με τον υπολογιστικό εξοπλισμό του χρήστη, ενώ επιτρέπει άμεση πρόσβαση σε τίτλους χωρίς εγκατάσταση. Παράλληλα, το cloud gaming ενισχύει την αλληλεπίδραση και την ομαδική εμπειρία, καθώς η διαχείριση multiplayer και η αποθήκευση προόδου μπορούν να γίνονται κεντρικά και συγχρονισμένα.

Η **τεχνητή νοημοσύνη (AI)** είναι άλλη μια τεχνολογική τάση που επηρεάζει το σχεδιασμό παιχνιδιών. Η AI χρησιμοποιείται για τη δημιουργία πιο ρεαλιστικών και δυναμικών χαρακτήρων, την προσαρμογή της δυσκολίας στο επίπεδο του παίκτη και την αυτόματη δημιουργία περιεχομένου, όπως νέοι χάρτες ή αποστολές. Η ενσωμάτωση της AI επεκτείνει τη διάρκεια ζωής του παιχνιδιού και προσφέρει εξατομικευμένες εμπειρίες, κρατώντας το ενδιαφέρον των παικτών σε υψηλά επίπεδα.

Τέλος, η τάση προς **διαδραστική αφήγηση και εξελιγμένα συστήματα επιλογών** δίνει στους παίκτες μεγαλύτερη αίσθηση ελευθερίας και συμμετοχής στην πλοκή του παιχνιδιού. Μέσω μηχανισμών branching και adaptive storytelling, οι αποφάσεις του χρήστη διαμορφώνουν την εξέλιξη της ιστορίας, δημιουργώντας μοναδικές εμπειρίες σε κάθε παιχνίδι.

Συνολικά, οι τεχνολογικές τάσεις στα video games υπογραμμίζουν μια σαφή μετάβαση από στατικά παιχνίδια σε δυναμικά, εξατομικευμένα και κοινωνικά διασυνδεδεμένα περιβάλλοντα, θέτοντας τα θεμέλια για τις επόμενες δεκαετίες της βιομηχανίας.

8.2 Κοινωνική Διάσταση και Multiplayer

Τα multiplayer παιχνίδια έχουν μεταμορφώσει ριζικά την εμπειρία των βιντεοπαιχνιδιών, καθιστώντας την έννοια της κοινωνικής αλληλεπίδρασης κεντρική για τον σύγχρονο gamer. Στα παιχνίδια αυτά, οι παίκτες μπορούν να συμμετέχουν σε κοινούς κόσμους, να συνεργάζονται ή να ανταγωνίζονται μεταξύ τους σε πραγματικό χρόνο, δημιουργώντας δεσμούς και αίσθηση κοινότητας. Η κοινωνική διάσταση δεν περιορίζεται μόνο στην ψυχαγωγία, αλλά περιλαμβάνει και την ανάπτυξη δεξιοτήτων συνεργασίας, στρατηγικής σκέψης και επικοινωνίας.

Η υιοθέτηση πλατφορμών όπως τα MMORPGs (Massively Multiplayer Online Role-Playing Games) και τα battle royale παιχνίδια επιτρέπει στους παίκτες να συνδεθούν ανεξάρτητα από γεωγραφικούς περιορισμούς. Σε αυτά τα περιβάλλοντα, η αλληλεπίδραση με άλλους παίκτες δεν περιορίζεται στην εκτέλεση παιχνιδιού, αλλά εκτείνεται στη δημιουργία ομάδων, τη συμμετοχή σε κοινότητες, την οργάνωση γεγονότων και την ανταλλαγή στρατηγικών. Έτσι, τα παιχνίδια λειτουργούν παράλληλα ως κοινωνικά δίκτυα, ενισχύοντας τη συνεργασία και την αίσθηση του ανήκειν.

Η ανάπτυξη multiplayer εμπειριών απαιτεί τεχνική υποδομή για συγχρονισμό δεδομένων, χειρισμό latency και διαχείριση server, διασφαλίζοντας ότι οι παίκτες βιώνουν μια ομαλή και συνεπή εμπειρία. Επιπλέον, η κοινωνική διάσταση επεκτείνεται μέσω της ενσωμάτωσης chat, voice communication και leaderboard συστημάτων, που επιτρέπουν την αλληλεπίδραση και τον ανταγωνισμό σε πολλαπλά επίπεδα.

Τα multiplayer παιχνίδια έχουν επίσης επιπτώσεις στην ψυχολογία των παικτών. Προάγουν την ανάπτυξη κοινωνικών δεξιοτήτων, την ομαδική συνεργασία και την επίλυση προβλημάτων σε πραγματικό χρόνο. Παράλληλα, δημιουργούν περιβάλλοντα όπου η επικοινωνία, η στρατηγική και η προσαρμοστικότητα είναι κρίσιμα στοιχεία επιτυχίας, καθιστώντας τα όχι μόνο μέσα ψυχαγωγίας, αλλά και πλατφόρμες μάθησης και κοινωνικής αλληλεπίδρασης.

Συνολικά, η κοινωνική διάσταση των multiplayer παιχνιδιών καταδεικνύει ότι τα video games υπερβαίνουν την ατομική ψυχαγωγία, διαμορφώνοντας κοινοτικές εμπειρίες και ενισχύοντας δεξιότητες που έχουν εφαρμογή πέρα από το παιχνίδι, επηρεάζοντας τόσο την προσωπική ανάπτυξη όσο και τη συλλογική δυναμική των παικτών.

8.3 Ηθικά και Πολιτισμικά Ζητήματα

Τα βιντεοπαιχνίδια δεν αποτελούν μόνο μέσο ψυχαγωγίας, αλλά και έναν καθρέφτη των κοινωνικών, ηθικών και πολιτισμικών αξιών. Η ανάπτυξη περιεχομένου σε αυτά τα περιβάλλοντα απαιτεί προσεκτική προσέγγιση, καθώς οι επιλογές των δημιουργών επηρεάζουν την αντίληψη των παικτών για ηθικά διλήμματα, κοινωνικές συμπεριφορές και πολιτισμικά στερεότυπα. Παιχνίδια με βία, σεξουαλικό περιεχόμενο ή αναπαραστάσεις συγκεκριμένων κοινωνικών ομάδων μπορεί να δημιουργήσουν συζητήσεις για την υπευθυνότητα της ψυχαγωγίας και τις επιπτώσεις της στην ψυχολογία των παικτών.

Η ηθική διάσταση επεκτείνεται και στη διαχείριση δεδομένων και προσωπικών πληροφοριών, ιδιαίτερα στα online και multiplayer παιχνίδια. Η προστασία της ιδιωτικότητας, η αποφυγή εκμετάλλευσης παικτών μέσω μικροπληρωμών (microtransactions) ή loot boxes, και η

πρόληψη της εξάρτησης αποτελούν κρίσιμα ζητήματα που απαιτούν υπεύθυνη σχεδίαση και εποπτεία.

Από πολιτισμική σκοπιά, τα παιχνίδια αντανακλούν ή διαμορφώνουν αξίες και ταυτότητες. Οι δημιουργοί συχνά εμπνέονται από μυθολογίες, ιστορικά γεγονότα ή κοινωνικά φαινόμενα, αλλά είναι σημαντικό να παρουσιάζονται με σεβασμό και ακρίβεια. Η παγκόσμια διάδοση των video games σημαίνει ότι τα πολιτισμικά μηνύματα που μεταδίδονται έχουν απήχηση σε ποικίλα ακροατήρια, καθιστώντας την πολιτισμική ευαισθησία κρίσιμη για την αποφυγή παρεξηγήσεων ή αρνητικών στερεοτύπων.

Τα ηθικά και πολιτισμικά ζητήματα δεν περιορίζονται μόνο στο περιεχόμενο, αλλά αφορούν και τις σχέσεις μεταξύ παικτών. Η συμπεριφορά σε online κοινότητες μπορεί να αντικατοπτρίζει ή να ενισχύει κοινωνικά πρότυπα, ενώ η ενθάρρυνση της θετικής αλληλεπίδρασης και της συνεργασίας συμβάλλει στη δημιουργία ασφαλών και φιλικών χώρων.

Συνοψίζοντας, η κατανόηση των ηθικών και πολιτισμικών πτυχών των βιντεοπαιχνιδιών είναι απαραίτητη τόσο για τους δημιουργούς όσο και για τους παίκτες. Η υπεύθυνη ανάπτυξη και η ευαισθητοποίηση σχετικά με το περιεχόμενο, τη συμπεριφορά και την πολιτισμική αναπαράσταση διασφαλίζουν ότι τα παιχνίδια μπορούν να αποτελέσουν ένα θετικό, εκπαιδευτικό και κοινωνικά υπεύθυνο μέσο ψυχαγωγίας.

Κεφάλαιο 9ο: Συμπεράσματα και Προτάσεις

Το παρόν κεφάλαιο επιχειρεί να συνοψίσει τα βασικά συμπεράσματα από την ανάπτυξη του παιχνιδιού, να εντοπίσει τις τεχνικές δυσκολίες που προέκυψαν και να προτείνει πιθανές κατευθύνσεις για μελλοντική επέκταση ή βελτίωση του έργου. Μέσα από την εμπειρία της υλοποίησης ενός πλήρως λειτουργικού browser-based turn-based παιχνιδιού χωρίς τη χρήση framework ή game engine, αναδείχθηκαν τόσο οι δυνατότητες της Vanilla JavaScript όσο και τα όριά της. Επιπλέον, καταγράφηκαν μαθήματα που μπορούν να αξιοποιηθούν σε παρόμοιες αναπτύξεις στο μέλλον.

Η ενότητα που ακολουθεί καταγράφει με συστηματικό τρόπο τις τεχνικές προκλήσεις, τις πιθανές βελτιώσεις και την προσωπική αξιολόγηση της διαδικασίας.

9.1 Τεχνικές δυσκολίες και αντιμετώπισή τους

Κατά την ανάπτυξη του παιχνιδιού, αναδείχθηκαν αρκετές τεχνικές προκλήσεις που αφορούσαν τόσο την αρχιτεκτονική του κώδικα όσο και τον τρόπο με τον οποίο διαχειρίζονται η λογική του παιχνιδιού και η εμπειρία του χρήστη. Παρακάτω παρουσιάζονται οι σημαντικότερες από αυτές, μαζί με τις στρατηγικές που χρησιμοποιήθηκαν για την αντιμετώπισή τους:

1. Απουσία Framework και ανάγκη για αρχιτεκτονική οργάνωση

Η απόφαση να χρησιμοποιηθεί αποκλειστικά Vanilla JavaScript χωρίς βιβλιοθήκες ή frameworks, αν και ενίσχυσε τη διαφάνεια του έργου, κατέστησε αναγκαία την υλοποίηση μηχανισμών που συνήθως παρέχονται «έτοιμοι». Για παράδειγμα:

- Δεν υπήρχε μηχανισμός state management, οπότε δημιουργήθηκαν custom λύσεις (π.χ. PlayerState, TurnCycle, custom event buses).
- Το routing και η αλλαγή «σελίδων» (μενού, σκηνές, μάχη) έγινε μέσω SPA-οργάνωσης με custom handlers.

Αντιμετώπιση: Σχεδιάστηκαν global namespaces μέσω του window object ώστε να επιτευχθεί σαφής διαχωρισμός ευθυνών. Η modular προσέγγιση των αρχείων και η χρήση κλάσεων βοήθησαν σημαντικά στην οργάνωση.

2. Συγχρονισμός animations και game logic

Ο συντονισμός μεταξύ της λογικής της μάχης (π.χ. ζημιές, αλλαγή χαρακτήρων) και των animations (όπως εφέ επίθεσης ή μετάβασης) ήταν κρίσιμος. Χωρίς σωστό συγχρονισμό, το παιχνίδι έδινε την εντύπωση ασυνέπειας ή καθυστέρησης.

Αντιμετώπιση: Κατασκευάστηκε ένα μηχανισμός await/Promise ώστε κάθε animation να ολοκληρώνεται πριν προχωρήσει η λογική του παιχνιδιού. Οι κλάσεις BattleAnimation και TextMessage αξιοποιήθηκαν ως μονάδες που επιστρέφουν υποσχέσεις (Promises), επιτρέποντας την αλληλουχία ενεργειών.

3. Ανάγκη για διαχείριση πολλών μενού και επιλογών χρήστη

Το παιχνίδι διαθέτει πληθώρα διαδραστικών μενού (επιλογή ενέργειας, αλλαγή χαρακτήρα, μηνύματα, κ.ά.). Η διαχείριση τους χωρίς framework αποτέλεσε πρόκληση, κυρίως όσον αφορά τη συντήρηση και την επεκτασιμότητα.

Αντιμετώπιση: Σχεδιάστηκαν γενικές κλάσεις μενού (KeyboardMenu, ReplacementMenu, SubmissionMenu) που μπορούν να επαναχρησιμοποιηθούν με παραμετροποίηση. Αυτό επέτρεψε τον διαχωρισμό μεταξύ UI και business logic.

4. Οργάνωση assets και δομών περιεχομένου

Η ενσωμάτωση των sprites, maps, audio και δεδομένων εχθρών/ενεργειών απαιτούσε ξεκάθαρη δομή αρχείων ώστε να είναι ευανάγνωστη και εύκολα επεκτάσιμη.

Αντιμετώπιση: Χρησιμοποιήθηκαν φάκελοι όπως Content/, images/, Battle/, και τα δεδομένα ορίστηκαν σε εξωτερικά .js αρχεία (actions.js, enemies.js, pizzas.js) για καλύτερη συντήρηση.

5. Debugging χωρίς εργαλεία υψηλού επιπέδου

Η απουσία framework σήμαινε και την απουσία developer tooling (π.χ. hot reload, error boundaries, component trees). Αυτό δυσκόλεψε τον εντοπισμό σφαλμάτων, ειδικά σε σύνθετες αλληλουχίες (όπως στον μηχανισμό της μάχης).

Αντιμετώπιση: Χρησιμοποιήθηκαν απλοί μηχανισμοί logging (π.χ. console.log, custom debug flags) και μικρές, επαναχρησιμοποιήσιμες μονάδες ώστε να είναι πιο ελεγχόμενη η ροή.

Συνοπτικά, αν και οι τεχνικές προκλήσεις ήταν αρκετές, η αντιμετώπισή τους συνέβαλε όχι μόνο στην επιτυχή ολοκλήρωση του έργου, αλλά και στην εμβάθυνση της κατανόησης βασικών αρχών ανάπτυξης λογισμικού, όπως ο διαχωρισμός ευθυνών, η διαχείριση κατάστασης και η συγχρονισμένη εκτέλεση.

9.2 Δυνατότητες επέκτασης παιχνιδιού

Το παιχνίδι, όπως υλοποιήθηκε, παρέχει μια σταθερή βάση για σημαντική επέκταση τόσο σε επίπεδο περιεχομένου όσο και σε επίπεδο μηχανισμών. Από τη σκοπιά του gameplay, μπορούν εύκολα να προστεθούν νέοι τύποι «pizzas», ειδικές ικανότητες, περισσότερα status effects και μοναδικοί εχθροί με ξεχωριστές τακτικές, εμπλουτίζοντας τη στρατηγική της μάχης. Η υπάρχουσα δομή επιτρέπει την ενσωμάτωση διαλόγων με επιλογές (branching dialogue), quest systems και RPG-like progression (όπως leveling, experience points), χωρίς να χρειάζεται ριζική ανακατασκευή. Επιπλέον, σε επίπεδο UI, η υποστήριξη για mobile περιβάλλον ή χρήση gamepad controllers μπορεί να βελτιώσει σημαντικά τη διαδραστικότητα. Από αρχιτεκτονικής πλευράς, η modular διάρθρωση του κώδικα και η οργάνωση σε ανεξάρτητες κλάσεις και φακέλους διευκολύνουν την επεκτασιμότητα και την ενσωμάτωση νέων σκηνών, χαρτών και animations. Τέλος, η μελλοντική προσθήκη multiplayer δυνατοτήτων (π.χ. μέσω WebSockets) ή αποθήκευσης προόδου στο cloud θα μπορούσε να μετατρέψει το έργο σε μια πλήρως λειτουργική browser-based πλατφόρμα RPG.

9.3 Τελικές σκέψεις

Η υλοποίηση ενός turn-based browser παιχνιδιού αποκλειστικά με χρήση Vanilla JavaScript, HTML και CSS απέδειξε ότι είναι εφικτό να δημιουργηθεί ένα λειτουργικό και επεκτάσιμο

παιχνίδι χωρίς τη βοήθεια εξωτερικών βιβλιοθηκών ή game engines. Μέσα από τη διαδικασία ανάπτυξης αναδείχθηκε η σημασία της καλής αρχιτεκτονικής, της καθαρής οργάνωσης αρχείων και της σωστής διαχείρισης του game state και των γεγονότων. Παρά τις τεχνικές προκλήσεις, η εμπειρία ήταν ιδιαίτερα εκπαιδευτική και ενίσχυσε την κατανόηση θεμελιωδών εννοιών όπως το DOM manipulation, το event handling και η λογική του game loop.

Επιπλέον, το έργο ανέδειξε τη δυνατότητα χρήσης των «καθαρών» τεχνολογιών του ιστού για τη δημιουργία σύνθετων εφαρμογών, ενώ ταυτόχρονα προώθησε την προσέγγιση της απλότητας και του απόλυτου ελέγχου στον κώδικα. Το τελικό αποτέλεσμα δεν αποτελεί μόνο μια απόδειξη τεχνικής ικανότητας, αλλά και μια βάση πάνω στην οποία μπορούν να χτιστούν πιο σύνθετες ιδέες και λειτουργίες στο μέλλον. Το έργο επιβεβαιώνει ότι η δημιουργικότητα και η μεθοδικότητα μπορούν να οδηγήσουν σε ποιοτικές υλοποιήσεις ακόμη και με απλά εργαλεία.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Brodie, Ian. “The Medium of the Video Game. By Mark J. P. Wolf, Ed. Foreword by Ralph H. Baer. (Austin: University of Texas Press, 2001. Xx + 203 P., ISBN 0-292-79150-X).” Ethnologies, vol. 24, no. 2, 2002, p. 264, https://doi.org/10.7202

[2] Praise for the Ultimate History of Video Games.

[3] Juul, J. (2005). Half-Real: Video Games between Real Rules and Fictional Worlds. MIT Press.

[4] Kerr, Aphra. Global Games. New York : Routledge, 2017., Routledge, 27 Mar. 2017, www.taylorfrancis.com/

[5] Statista (2023). Video Game Industry – Revenue Worldwide.

[6] Newzoo (2022). Global Games Market Report 2022. Newzoo, https://kibbutzpsicologia.com/

[7] Salen, K., & Zimmerman, E. (2004). Rules of Play: Game Design Fundamentals. MIT Press, https://gamifique.wordpress.com/

[8] Gee, James Paul. “What Video Games Have to Teach Us about Learning and Literacy.” Computers in Entertainment, vol. 1, no. 1, 2003, p. 20, https://doi.org/10.1145/

[9] Apperley, Thomas H. “Genre and Game Studies: Toward a Critical Approach to Video Game Genres.” Simulation & Gaming, vol. 37, no. 1, Mar. 2006, pp. 6–23, https://doi.org/

[10] Aarseth, Espen. “Playing Research: Methodological Approaches to Game Analysis.” Artnodes, vol. 0, no. 7, 1 Dec. 2007, https://doi.org

[11] Campbell, Arne. “Gary Alan Fine - Shared Fantasy_ Role Playing Games as Social Worlds-Univ of Chicago Pr (1983).” *Scribd*, 2025, www.scribd.com/. Accessed 20 Aug. 2025

[12] Cover, Jennifer Grouling. *IN TABLETOP ROLE-PLAYING GAMES*. 92818656887.pdf

[13] Adams, Ernest. “Fundamentals of Game Design.” Choice Reviews Online, 1 Jan. 2010, www.academia.edu/

[14] Loh, Christian Sebastian, et al. “Serious Games Analytics: Theoretical Framework.” Serious Games Analytics, 2015, pp. 3–29, https://doi.org. Accessed 15 June 2022

[15] Montfort, Nick. Twisty Little Passages: An Approach to Interactive Fiction. Direct.mit.edu, The MIT Press, 7 Nov. 2003, direct.mit.edu/

[16] “Game Programming Patterns - Free Computer, Programming, Mathematics, Technical Books, Lecture Notes and Tutorials.” FreeComputerBooks, 2025, freecomputerbooks.com. Accessed 20 Aug. 2025

[17] Gregory, Jason. Game Engine Architecture. www.latexstudio.net/

[18] “Learning Python, 5th Edition [Book]. www.oreilly.com/

[19] Penton, Ron. Beginning C#. theswissbay.ch

[20] Juul, J. (n.d.). A CASUAL REVOLUTION Reinventing Video Games and Their Players. https://jesperjuul.net/casualrevolution/casual_revolution_chapter1.pdf

[21] Schell, J. (2008). The Art of Game Design. https://www.inventoridigiochi.it/wp-content/uploads/2020/07/art-of-game-design.pdf

[22] Brodie, I. (2002). The Medium of the Video Game. By Mark J. P. Wolf, ed. Foreword by Ralph H. Baer. (Austin: University of Texas Press, 2001. xx + 203 p., ISBN 0-292-79150-X). Ethnologies, 24(2), 264 https://www.researchgate.net/

[23] Rabin, S. (n.d.). Introduction to Game Development. https://files.lyberry.com/books/How%20To/Programming/Introduction_to_Game_Development__Steve_Rabin_2e.pdf

[24] Millington, L. (n.d.). Praise for Game Physics Engine Development. https://www.r-5.org/files/books/computers/algo-list/realtime-3d/Ian_Millington-Game_Physics_Engine_Development-EN.pdf

[25] Godot Documentation. (2023). About Godot Engine. https://docs.godotengine.org