ΠΜΣ Ανάπτυξη Ψηφιακών Παιχνιδιών και Πολυμεσικών Εφαρμογών 2025-2026. Τμήμα Επικοινωνίας και Ψηφιακών Μέσων Πανεπιστήμιο Δυτικής Μακεδονίας

Μάθημα: Αλγοριθμική Σκέψη

ΔΗΜΙΟΥΡΓΙΑ VIDEO GAME ΜΕ ΧΡΗΣΗ PYTHON ΚΑΙ PYGAME

5 Ιανουαρίου 2026

Χαράλαμπος Στεργιόπουλος mpw00089
Κωνσταντίνος Γαϊτάνης mpw00087

Περιεχόμενα

1. Περίληψη

Η παρούσα τεχνική αναφορά τεκμηριώνει την υλοποίηση ενός δισδιάστατου arcade παιχνιδιού τύπου Digger σε περιβάλλον Python με χρήση της βιβλιοθήκης Pygame. Το παιχνίδι βασίζεται σε grid-based λογική και υλοποιεί βασικούς μηχανισμούς εξερεύνησης λαβυρίνθου, συλλογής αντικειμένων (emeralds), αλληλεπίδρασης με δυναμικά αντικείμενα (gold bags), και αντιμετώπισης εχθρών μέσω τεχνητής νοημοσύνης και συστήματος πυροβόλου όπλου. Η ροή του παιχνιδιού οργανώνεται μέσω αρχιτεκτονικής “scenes”, με διακριτές οθόνες (main menu, game, settings, highscores, how-to-play, game over), ενώ η λειτουργικότητα του gameplay αποσυντίθεται σε επιμέρους συστήματα (movement, tile interaction, AI, difficulty scaling, bullets, scoring, lives, persistence).

Η αναφορά περιγράφει την αρχιτεκτονική του κώδικα, τα κρίσιμα σημεία σχεδίασης, τις επιλογές που σχετίζονται με επεκτασιμότητα (π.χ. υποστήριξη δύο παικτών μέσω split screen), καθώς και τον τρόπο αποθήκευσης και προβολής υψηλών σκορ σε SQLite. Επιπλέον καταγράφεται η προέλευση των assets, οι άδειες χρήσης όπου απαιτείται, και οι περιπτώσεις στις οποίες χρησιμοποιήθηκαν LLMs υποστηρικτικά (π.χ. τεκμηρίωση και αναδιάρθρωση κώδικα), μαζί με τον βαθμό επίδρασής τους στο τελικό αποτέλεσμα.

2. Εισαγωγή

2.1 Στόχος και αντικείμενο της εργασίας

Η εργασία αφορά την ανάπτυξη ενός ολοκληρωμένου παιχνιδιού τύπου arcade με έμφαση στη σαφήνεια υλοποίησης, στη δομημένη αρχιτεκτονική και στην τεκμηριωμένη παρουσίαση του τελικού προϊόντος. Η επιλογή ενός παιχνιδιού τύπου Digger επιτρέπει την ταυτόχρονη διερεύνηση πολλών θεμελιωδών εννοιών ανάπτυξης παιχνιδιών: διαχείριση κατάστασης (game state), κίνηση σε πλέγμα (grid), αλληλεπίδραση με περιβάλλον (tiles), απλές μορφές τεχνητής νοημοσύνης (enemy pursuit/flee), σύστημα συγκρούσεων (collisions), καθώς και βασικά στοιχεία οπτικοποίησης (tilemap, camera, sprites) και ήχου (effects, music, mute/unmute).

Στο πλαίσιο αυτό, η υλοποίηση οργανώθηκε ώστε να προσεγγίζει πρακτικές που συναντώνται σε μεγαλύτερα έργα: διαχωρισμός αρμοδιοτήτων, αποσύνθεση της

λογικής σε “systems”, χρήση αντικειμενοστραφών μοντέλων για τα βασικά entities, και καθορισμός σταθερού API μεταξύ υποσυστημάτων (π.χ. scenes και input).

2.2 Βασική λειτουργικότητα του παιχνιδιού

Ο παίκτης ελέγχει έναν χαρακτήρα (“Digger”) που κινείται σε δισδιάστατο χάρτη με πλακίδια. Το περιβάλλον περιλαμβάνει χώμα, τούνελ, αντικείμενα προς συλλογή (emeralds), καθώς και gold bags που μπορούν να πέσουν/μετατραπούν σε gold piles και να αλληλεπιδράσουν με παίκτες και εχθρούς. Στόχος κάθε επιπέδου είναι η συλλογή όλων των emeralds· όταν αυτό επιτευχθεί, το παιχνίδι μεταβαίνει στο επόμενο level μετά από σύντομη καθυστέρηση. Η δυσκολία κλιμακώνεται ανά επίπεδο (π.χ. αριθμός εχθρών, ρυθμός κίνησης εχθρών).

Οι εχθροί υλοποιούνται με δύο μορφές (Nobbin/Hobbin), οι οποίες διαφέρουν ως προς τους κανόνες μετακίνησης και αλληλεπίδρασης με το terrain. Η συμπεριφορά τους καθορίζεται από AI λογική που υπολογίζει κατεύθυνση κίνησης με βάση τον στόχο (πλησιέστερος ζωντανός παίκτης) και την τρέχουσα κατάσταση παιχνιδιού (π.χ. bonus mode όπου οι εχθροί απομακρύνονται). Επιπλέον, ο παίκτης διαθέτει όπλο με cooldown και βολίδες που συγκρούονται με solid tiles ή εχθρούς.

2.3 Αρχιτεκτονική προσέγγιση και κριτήρια ποιότητας

Η υλοποίηση σχεδιάστηκε γύρω από δύο βασικούς άξονες: (α) σαφήνεια/συντηρησιμότητα κώδικα και (β) πληρότητα λειτουργίας ως “πακέτο” παιχνιδιού. Για τον λόγο αυτό, υιοθετήθηκε scene manager με σαφή σύμβαση (enter/exit/update/render), ώστε κάθε οθόνη να αποτελεί ανεξάρτητη ενότητα. Παράλληλα, η λογική gameplay αποσυντίθεται σε systems, διευκολύνοντας την επέκταση ή αντικατάσταση επιμέρους συμπεριφορών χωρίς διάχυση αλλαγών σε άσχετα αρχεία.

Ως κριτήρια ποιότητας τέθηκαν: σταθερότητα runtime (π.χ. ασφαλείς έλεγχοι bounds), προβλέψιμη διαχείριση χρόνου (dt-based update), επαναχρησιμοποίηση κώδικα (π.χ. κοινό render path για split screen), και αναπαραγωγιμότητα συμπεριφοράς ανά επίπεδο μέσω κεντρικού difficulty scaler.

2.4 Δομή της αναφοράς

Στα επόμενα κεφάλαια παρουσιάζονται οι τεχνολογίες που χρησιμοποιήθηκαν, η σχεδίαση του παιχνιδιού, η αρχιτεκτονική και τα βασικά συστήματα. Ακολουθεί ειδική ενότητα για τα assets και την αδειοδότησή τους, καθώς και τεκμηριωμένη αναφορά στη χρήση LLM όπου αξιοποιήθηκε. Η αναφορά ολοκληρώνεται με χαρτογράφηση της κάλυψης προδιαγραφών και προτεινόμενες μελλοντικές επεκτάσεις για σημεία που δεν υλοποιήθηκαν πλήρως.

3. Τεχνολογίες και Εργαλεία Υλοποίησης

3.1 Γλώσσα προγραμματισμού και περιβάλλον εκτέλεσης

Η υλοποίηση του παιχνιδιού πραγματοποιήθηκε στη γλώσσα Python, η οποία επιλέχθηκε λόγω της ευχρηστίας, της εκφραστικότητάς της και της ευρείας υποστήριξης για ανάπτυξη παιχνιδιών μικρής και μεσαίας κλίμακας. Η Python επιτρέπει ταχεία ανάπτυξη (rapid prototyping), ενώ παράλληλα υποστηρίζει αντικειμενοστραφή σχεδίαση, στοιχείο κρίσιμο για τη δομή του παρόντος έργου.

Το παιχνίδι εκτελείται σε περιβάλλον desktop (Windows/Linux), με χρήση του event loop της Pygame και χρονισμό βασισμένο σε μεταβλητό dt (delta time), ώστε η λογική του παιχνιδιού να παραμένει ανεξάρτητη από το frame rate.

3.2 Βιβλιοθήκη Pygame

Η βασική βιβλιοθήκη που χρησιμοποιήθηκε είναι η Pygame, η οποία παρέχει:

- διαχείριση παραθύρου και rendering 2D γραφικών,
- σύστημα συμβάντων (keyboard, quit events),
- βασική υποστήριξη ήχου,
- primitives για collision και sprites.

Η Pygame χρησιμοποιήθηκε αποκλειστικά ως low-level framework, χωρίς να επιβάλει αρχιτεκτονική. Η ευθύνη για τη δομή του παιχνιδιού (scene management, game loop, διαχωρισμός λογικής/απεικόνισης) παραμένει εξ ολοκλήρου στον σχεδιασμό του έργου, γεγονός που επέτρεψε την εφαρμογή καθαρών αρχιτεκτονικών προτύπων.

3.3 Αρχιτεκτονικό πρότυπο MVC

Η συνολική δομή του παιχνιδιού ακολουθεί χαλαρή αλλά σαφή εφαρμογή του προτύπου Model–View–Controller (MVC):

Model:
Περιλαμβάνει τις κλάσεις που αναπαριστούν την κατάσταση του παιχνιδιού και τους κανόνες του κόσμου, όπως Player, Enemy, Level, GoldBag, Score, Lives, καθώς και enums τύπων (TileType, Direction, GameMode). Οι κλάσεις αυτές δεν γνωρίζουν τίποτα για γραφικά ή input.

View:
Υλοποιείται μέσω των classes του client.views, όπως TilemapView, PlayerSprite, EnemySprite, Camera2D. Οι κλάσεις αυτές είναι υπεύθυνες μόνο για την απεικόνιση της κατάστασης του Model, μετατρέποντας world coordinates σε screen coordinates.

Controller:
Ο ρόλος του controller διαμοιράζεται:

- σε επίπεδο input (InputManager, InputController),
- και σε επίπεδο scene (GameScene, MainMenuScene, κ.λπ.), όπου λαμβάνονται αποφάσεις βάσει input και μεταβάλλεται το model.

Η επιλογή αυτή αποτρέπει τη σύζευξη λογικής και rendering και επιτρέπει μελλοντική αντικατάσταση της απεικόνισης (π.χ. άλλο rendering backend) χωρίς αλλαγές στο gameplay.

3.4 Εφαρμογή DAO για επίμονη αποθήκευση

Για την αποθήκευση και ανάκτηση υψηλών σκορ εφαρμόστηκε το πρότυπο DAO (Data Access Object) μέσω της κλάσης ScoreRepository.

Ο ρόλος του DAO είναι:

- να απομονώνει πλήρως τον κώδικα πρόσβασης στη βάση δεδομένων (SQLite),
- να παρέχει καθαρό API (top_10(), add_score()),
- και να αποτρέπει την άμεση χρήση SQL σε scenes ή gameplay logic.

Με αυτόν τον τρόπο, το παιχνίδι παραμένει ανεξάρτητο από την τεχνολογία persistence, ενώ είναι εφικτή η αντικατάσταση της SQLite με άλλο σύστημα (π.χ. remote database) χωρίς αλλαγές στον υπόλοιπο κώδικα.

3.5 Αρχές SOLID στην υλοποίηση

Κατά τον σχεδιασμό των βασικών συστημάτων εφαρμόστηκαν, στον βαθμό που είναι ρεαλιστικός για ένα παιχνίδι αυτού του μεγέθους, οι αρχές SOLID:

Single Responsibility Principle (SRP):
Κάθε σύστημα έχει σαφή ρόλο (π.χ. EnemyAISystem, BulletSystem, GoldBagSystem).

Open/Closed Principle (OCP):
Η συμπεριφορά των εχθρών επεκτείνεται μέσω νέων systems (π.χ. EnemyFormSystem) χωρίς τροποποίηση του βασικού loop.

Dependency Inversion (DIP):
Τα scenes εξαρτώνται από αφαιρέσεις (interfaces/συμβόλαια συστημάτων) και όχι από συγκεκριμένες υλοποιήσεις rendering ή persistence.

Η εφαρμογή των αρχών αυτών δεν είναι τυπική, αλλά προσαρμοσμένη σε game development περιβάλλον, όπου η αναγνωσιμότητα και η ροή είναι εξίσου κρίσιμες με την καθαρότητα της σχεδίασης.

4. Σχεδίαση Παιχνιδιού (Game Design) και Χαρτογράφηση στην Υλοποίηση

4.1 Στόχος και βασική φιλοσοφία σχεδίασης

Ο σχεδιασμός του παιχνιδιού βασίστηκε στη λογική των κλασικών arcade παιχνιδιών τύπου Digger, με έμφαση στη σαφήνεια των μηχανισμών, στη σταδιακή αύξηση της δυσκολίας και στη δυνατότητα γρήγορης κατανόησης από τον παίκτη, χωρίς εκτενή tutorials. Η πρόθεση δεν ήταν η πιστή αναπαραγωγή ενός υπάρχοντος τίτλου, αλλά η επανασχεδίαση των βασικών ιδεών του μέσα από σύγχρονες αρχές λογισμικού.

Κεντρικός στόχος του παίκτη είναι η συλλογή όλων των emeralds κάθε επιπέδου, αποφεύγοντας ή εξουδετερώνοντας τους εχθρούς. Η ολοκλήρωση όλων των

συλλεκτικών αντικειμένων αποτελεί τη μοναδική συνθήκη μετάβασης στο επόμενο επίπεδο, γεγονός που διατηρεί σαφή και μετρήσιμη πρόοδο.

4.2 Δομή επιπέδων και κόσμου

Ο κόσμος του παιχνιδιού αναπαρίσταται ως πλέγμα (grid) σταθερών διαστάσεων (Level(60, 40)), όπου κάθε κελί αντιστοιχεί σε tile συγκεκριμένου τύπου (TileType). Η επιλογή grid-based κόσμου επιτρέπει:

- απλοποιημένο collision detection,
- καθαρό enemy AI βασισμένο σε Manhattan distance,
- ευκολότερη διαχείριση digging και μετατροπής tiles.

Η σχεδίαση των επιπέδων είναι δυναμική: τα emeralds και τα gold bags τοποθετούνται με ψευδοτυχαίο τρόπο σε έγκυρα tiles κατά την έναρξη κάθε επιπέδου. Αυτό αυξάνει την επαναληψιμότητα του παιχνιδιού χωρίς ανάγκη χειροκίνητου level design.

4.3 Κεντρικοί μηχανισμοί gameplay

4.3.1 Κίνηση και digging

Η κίνηση των παικτών είναι διακριτή (ανά tile) και υλοποιείται μέσω του GridMovementSystem. Κάθε επιτυχής μετακίνηση:

- ελέγχει αν το tile-στόχος είναι προσβάσιμο,
- ενημερώνει τη θέση του παίκτη στο model,
- ενεργοποιεί το TileInteractionSystem, το οποίο μετατρέπει το χώμα σε tunnel και ενεργοποιεί συλλογές αντικειμένων.

Με αυτόν τον τρόπο, το digging δεν είναι ξεχωριστή ενέργεια αλλά φυσικό αποτέλεσμα της κίνησης, όπως στα κλασικά arcade παιχνίδια.

4.3.2 Συλλογή αντικειμένων και scoring

Τα emeralds και τα gold piles αποτελούν τις κύριες πηγές πόντων. Η συλλογή τους ανιχνεύεται αποκλειστικά σε επίπεδο model και όχι view, γεγονός που διασφαλίζει ανεξαρτησία από την απεικόνιση.

- Το σύστημα βαθμολογίας (ScoreSystem) αναλαμβάνει:
- την προσθήκη πόντων,
- τη διαφοροποίηση πόντων ανά αντικείμενο,
- τη σύνδεση με το HUD και τα high scores.

Στο co-op mode, κάθε παίκτης διατηρεί ανεξάρτητο score, ενισχύοντας την ανταγωνιστικότητα χωρίς να διαταράσσεται η συνεργατική φύση του παιχνιδιού.

4.3.3 Gold bags ως μηχανισμός ρίσκου

Τα gold bags σχεδιάστηκαν ως στοιχείο στρατηγικού ρίσκου. Μέσω του GoldBagSystem και του GoldBagPushSystem, ένα bag μπορεί:

- να πέσει κατακόρυφα,
- να μετατραπεί σε gold pile,
- να σκοτώσει εχθρούς κατά την πτώση.

Ο μηχανισμός αυτός εισάγει έμμεση επίθεση, διαφοροποιώντας το gameplay από απλή αποφυγή εχθρών.

4.4 Εχθροί και τεχνητή νοημοσύνη

Οι εχθροί (Nobbin / Hobbin) ακολουθούν σχεδιασμό βασισμένο σε καταστάσεις (forms). Η λογική τους διαχωρίζεται σε πολλαπλά συστήματα:

- EnemyAISystem: λήψη αποφάσεων κίνησης,
- EnemyFormSystem: μετάβαση μεταξύ Nobbin και Hobbin,
- HobbinDiggingSystem: digging μόνο για Hobbin.

Η απόφαση στόχευσης βασίζεται στον κοντινότερο ζωντανό παίκτη, επιτρέποντας φυσική συμπεριφορά τόσο σε single όσο και σε co-op mode. Η υλοποίηση αυτή επιτρέπει εύκολη προσθήκη νέων τύπων εχθρών χωρίς αλλαγές στον βασικό βρόχο του παιχνιδιού.

4.5 Σύστημα όπλων και bullets

Το όπλο του παίκτη σχεδιάστηκε ως περιορισμένος πόρος, με cooldown (Weapon) και ξεχωριστό σύστημα διαχείρισης (WeaponSystem, BulletSystem). Η επιλογή αυτή:

- αποτρέπει το spam πυροβολισμών,
- ενισχύει τη στρατηγική χρήση,
- διατηρεί το arcade ύφος.

Τα bullets φέρουν πληροφορία ιδιοκτησίας (player 1 ή player 2), ώστε η απόδοση πόντων να γίνεται σωστά στο co-op mode.

4.6 Καταστάσεις παιχνιδιού (Game Modes)

Το παιχνίδι υποστηρίζει πολλαπλές καταστάσεις (GameMode):

- NORMAL
- BONUS
- LEVEL_COMPLETE

Η χρήση καταστάσεων επιτρέπει καθαρό έλεγχο της ροής χωρίς πολύπλοκες συνθήκες. Κάθε mode επηρεάζει άμεσα τη συμπεριφορά του update loop, διατηρώντας τον κώδικα αναγνώσιμο.

4.7 Single Player και Co-op σχεδίαση

Η υποστήριξη δύο παικτών σχεδιάστηκε από την αρχή ως επέκταση και όχι ως ξεχωριστό mode. Η ύπαρξη λίστας players και ανεξάρτητων cameras επιτρέπει:

- split-screen απεικόνιση,
- κοινό κόσμο,
- ανεξάρτητες ζωές και scores.

Αν ένας παίκτης χάσει όλες τις ζωές του, μετατρέπεται σε spectator, χωρίς να διακόπτεται άμεσα το παιχνίδι, επιλογή που βελτιώνει την εμπειρία συνεργασίας.

5. Αρχιτεκτονική Λογισμικού και Εσωτερική Δομή Κώδικα

5.1 Γενική αρχιτεκτονική προσέγγιση

Η αρχιτεκτονική του παιχνιδιού βασίζεται σε έναν σαφή διαχωρισμό ευθυνών, με στόχο:

- την αναγνωσιμότητα του κώδικα,
- τη δυνατότητα επέκτασης,
- και τη συντήρηση χωρίς παρενέργειες.

Παρότι το παιχνίδι υλοποιείται σε περιβάλλον pygame (το οποίο δεν επιβάλλει συγκεκριμένο αρχιτεκτονικό πρότυπο), η σχεδίαση ακολουθεί συνειδητά αρχές MVC, DAO και SOLID, προσαρμοσμένες στη φύση ενός real-time παιχνιδιού.

Η συνολική δομή χωρίζεται σε δύο βασικά επίπεδα:

- shared/ – καθαρό game logic, ανεξάρτητο από το framework
- client/ – παρουσίαση, είσοδος χρήστη και οπτικοακουστική απεικόνιση

Αυτή η διάκριση είναι κρίσιμη, καθώς επιτρέπει θεωρητικά τη μεταφορά του παιχνιδιού σε άλλο frontend (π.χ. web ή άλλο engine) χωρίς αλλαγές στο core logic.

5.2 Εφαρμογή του MVC προτύπου

5.2.1 Model

Το Model υλοποιείται εξ ολοκλήρου στο πακέτο shared.model. Εκεί βρίσκονται όλες οι κλάσεις που αναπαριστούν την κατάσταση του παιχνιδιού:

- Player, Enemy, GoldBag, Bullet
- Level, Score, Lives, Weapon
- enums όπως TileType, Direction, GameMode, MonsterForm

Οι κλάσεις αυτές:

- δεν έχουν καμία γνώση για rendering,
- δεν εξαρτώνται από pygame,
- αποθηκεύουν μόνο κατάσταση και απλή συμπεριφορά.

Παράδειγμα χαρακτηριστικής επιλογής σχεδίασης είναι το γεγονός ότι το Player γνωρίζει μόνο τις tile συντεταγμένες του και όχι pixel θέσεις ή sprites. Αυτό καθιστά το model πλήρως testable και ανεξάρτητο.

5.2.2 View

Το View υλοποιείται στο client.views και διαχωρίζεται περαιτέρω σε:

- TilemapView για το επίπεδο
- sprites (PlayerSprite, EnemySprite, GoldBagSprite, EmeraldSprite)
- Camera2D
- εφέ (ExplosionSystem)

Οι κλάσεις view:

- λαμβάνουν δεδομένα από το model,
- δεν τροποποιούν ποτέ την κατάσταση του παιχνιδιού,
- περιορίζονται αποκλειστικά σε drawing.

Χαρακτηριστικό παράδειγμα σωστού MVC είναι ότι η συλλογή emerald:

- δεν γίνεται στο view,
- αλλά ανιχνεύεται στο update loop του GameScene και στα systems.

5.2.3 Controller (Scene-based Controller)

Σε ένα παιχνίδι, ο ρόλος του controller δεν ταυτίζεται πλήρως με τον κλασικό MVC controller. Εδώ, τον ρόλο αυτό αναλαμβάνουν:

- το SceneManager
- οι επιμέρους Scene κλάσεις (π.χ. GameScene, MainMenuScene)

Το GameScene λειτουργεί ως orchestrator:

- συλλέγει input,
- καλεί τα systems,
- ελέγχει transitions (game over, level complete),

- και συγχρονίζει model–view.

Η επιλογή αυτή είναι σύμφωνη με game architecture πρακτικές και όχι παραβίαση του MVC.

5.3 Systems-based σχεδίαση (Service Layer)

Ένα από τα ισχυρότερα σημεία της υλοποίησης είναι η χρήση μικρών, ανεξάρτητων systems, όλα στο shared.services.

Ενδεικτικά:

- GridMovementSystem
- TileInteractionSystem
- EnemyAISystem
- GoldBagSystem
- WeaponSystem
- BulletSystem
- ScoreSystem

Κάθε system:

- έχει μία σαφή ευθύνη,
- λειτουργεί πάνω σε model objects,
- δεν κρατά εσωτερική κατάσταση (stateless ή minimal state).

Αυτός ο σχεδιασμός:

- μειώνει την πολυπλοκότητα του GameScene,
- επιτρέπει unit testing,
- διευκολύνει την αντικατάσταση ή επέκταση logic.

5.4 Εφαρμογή αρχών SOLID

Single Responsibility Principle (SRP)

Κάθε κλάση επιτελεί έναν και μόνο ρόλο.
Παράδειγμα:

- EnemyFormSystem δεν κινεί εχθρούς,
- EnemyAISystem δεν αλλάζει μορφές,
- HobbinDiggingSystem ασχολείται αποκλειστικά με digging.

Η επιλογή αυτή είναι εμφανής στον κώδικα και μειώνει δραστικά τη σύζευξη.

Open–Closed Principle (OCP)

Το σύστημα είναι ανοιχτό σε επέκταση αλλά κλειστό σε τροποποίηση.
Παράδειγμα:

- νέος τύπος enemy μπορεί να προστεθεί με νέο system,
- χωρίς αλλαγές στον κεντρικό βρόχο του παιχνιδιού.
- Liskov Substitution Principle (LSP)

Παρότι δεν γίνεται εκτεταμένη χρήση inheritance, όπου αυτή υπάρχει (π.χ. Scene), η αντικατάσταση είναι ασφαλής. Κάθε scene ακολουθεί αυστηρά το συμβόλαιο της abstract κλάσης.

Interface Segregation Principle (ISP)

Δεν υπάρχουν “φορτωμένα” interfaces.
Κάθε σύστημα εκθέτει μόνο τις απολύτως απαραίτητες μεθόδους, π.χ. update(), on_enter().

Dependency Inversion Principle (DIP)

Το GameScene δεν υλοποιεί λογική χαμηλού επιπέδου. Αντίθετα, εξαρτάται από abstractions (systems), τα οποία μπορούν να αντικατασταθούν. Η AI, για παράδειγμα, βασίζεται στο EnemyBrain, το οποίο θα μπορούσε να αντικατασταθεί από διαφορετική στρατηγική χωρίς αλλαγή στον υπόλοιπο κώδικα.

5.5 DAO και Persistence

Η αποθήκευση high scores υλοποιείται μέσω του ScoreRepository, το οποίο λειτουργεί ως DAO (Data Access Object). Η κλάση αυτή:

- απομονώνει την πρόσβαση σε δεδομένα,
- επιτρέπει αλλαγή αποθηκευτικού μέσου (π.χ. αρχείο → βάση δεδομένων),
- δεν επηρεάζει το gameplay logic.

Αυτή η επιλογή αποτρέπει τη διάχυση persistence κώδικα μέσα στα scenes.

5.6 SceneManager και ροή εφαρμογής

Ο SceneManager αποτελεί το μοναδικό σημείο ελέγχου αλλαγής σκηνών. Αυτό διασφαλίζει:

- καθαρή μετάβαση μεταξύ menu, game, game over,
- απουσία side effects,
- σταθερό lifecycle (enter, exit, update, render).

6. Ροή Εκτέλεσης Παιχνιδιού και Λειτουργικότητα Ανά Frame

6.1 Ο κεντρικός βρόχος του παιχνιδιού (Game Loop)

Η εκτέλεση του παιχνιδιού βασίζεται στον κλασικό game loop, όπως αυτός υλοποιείται στην κλάση GameApp. Ο βρόχος αυτός επαναλαμβάνεται σε κάθε frame και αποτελεί τη ραχοκοκαλιά της λειτουργίας του παιχνιδιού.

- Σε υψηλό επίπεδο, κάθε επανάληψη του βρόχου ακολουθεί την εξής ακολουθία:
- Ανάγνωση γεγονότων (events) από το pygame
- Δημιουργία στιγμιότυπου εισόδου (InputSnapshot)
- Κλήση handle_input() της ενεργής σκηνής
- Κλήση update(dt) της σκηνής
- Κλήση render(surface) της σκηνής
- Αναμονή ώστε να διατηρηθεί σταθερό FPS

Η ύπαρξη του dt (delta time) επιτρέπει στο παιχνίδι να είναι ανεξάρτητο από την ταχύτητα του υπολογιστή, κάτι που θεωρείται βασική αρχή στον real-time προγραμματισμό.

6.2 Scene-driven εκτέλεση

Ολόκληρη η λογική gameplay εκτελείται μέσα στη σκηνή GameScene, η οποία αποτελεί την ενεργή σκηνή κατά τη διάρκεια του παιχνιδιού.

Η σκηνή αυτή:

- διατηρεί την κατάσταση του παιχνιδιού (level, παίκτες, εχθρούς),
- διαχειρίζεται τα transitions (game over, αλλαγή level),
- συντονίζει τα επιμέρους systems.

Η μέθοδος update(dt) της GameScene καλείται μία φορά ανά frame και αποτελεί το σημείο στο οποίο εξελίσσεται η λογική του παιχνιδιού.

6.3 Διαχείριση χρονισμών και καταστάσεων

Στην αρχή κάθε update γίνεται ενημέρωση όλων των χρονικών μετρητών:

- _move_cooldown: περιορίζει τη συχνότητα κίνησης
- _invuln_timer: χρονικό διάστημα αθανασίας μετά από hit
- weapon.update(dt): cooldown όπλου
- timers σχετικοί με bonus mode και ολοκλήρωση επιπέδου

Η χρήση timers αντί για blocking λογική επιτρέπει:

- ομαλή ροή,
- προβλέψιμη συμπεριφορά,
- και καθαρό έλεγχο κατάστασης.

6.4 Διαχείριση καταστάσεων παιχνιδιού (GameMode)

Το παιχνίδι λειτουργεί ως κατάσταση πεπερασμένου αυτόματου (FSM), με βασικές καταστάσεις:

- NORMAL
- BONUS
- LEVEL_COMPLETE

Σε κάθε frame, ελέγχεται πρώτα αν το παιχνίδι βρίσκεται σε ειδική κατάσταση. Για παράδειγμα:

- Αν το επίπεδο έχει ολοκληρωθεί, η λογική κίνησης παγώνει.
- Αν ο χρόνος bonus λήξει, επιστρέφει σε normal mode.

Αυτός ο διαχωρισμός μειώνει δραστικά τα edge cases και αποτρέπει ασυνέπειες (π.χ. κίνηση κατά τη μετάβαση επιπέδου).

6.5 Έλεγχος ολοκλήρωσης επιπέδου

Η ολοκλήρωση επιπέδου ανιχνεύεται με έναν καθαρό και αποδοτικό έλεγχο:

```
emerald_exists = any(TileType.EMERALD in row for row in self.level.tiles)
```

Αν δεν υπάρχουν άλλα emeralds:

- το παιχνίδι εισέρχεται σε LEVEL_COMPLETE,
- ενεργοποιείται χρονική καθυστέρηση,
- και στη συνέχεια φορτώνεται το επόμενο επίπεδο.

Η επιλογή αυτή διατηρεί τον έλεγχο σε ένα μόνο σημείο και αποφεύγει κατακερματισμένη λογική.

6.6 Κίνηση και αλληλεπίδραση παικτών

Η κίνηση των παικτών γίνεται σε grid-based περιβάλλον, μέσω του GridMovementSystem.

Για κάθε παίκτη:

- Ανιχνεύεται direction από το input
- Ελέγχεται αν επιτρέπεται η κίνηση
- Ενημερώνεται η θέση

Καλούνται:

- TileInteractionSystem
- ScoreSystem

Η αλληλεπίδραση με emeralds, tunnels και gold bags γίνεται μετά την επιτυχή κίνηση, διασφαλίζοντας σωστή σειρά γεγονότων.

6.7 Διαχείριση δεύτερου παίκτη (co-op)

Ο δεύτερος παίκτης λειτουργεί πλήρως ανεξάρτητα:

- ξεχωριστό input (WASD + Shift),
- ξεχωριστό σκορ και ζωές,
- ξεχωριστή κάμερα σε split screen mode.

Η μέθοδος _handle_player2_input() ενσωματώνεται ομαλά στον ίδιο update κύκλο, χωρίς ειδικές εξαιρέσεις στον υπόλοιπο κώδικα.

Σε περίπτωση απώλειας όλων των ζωών:

- ο παίκτης 2 μετατρέπεται σε spectator,
- το παιχνίδι συνεχίζεται αν ο παίκτης 1 είναι ζωντανός.

6.8 Εμφάνιση και ενημέρωση εχθρών

Η δημιουργία εχθρών γίνεται ελεγχόμενα μέσω του EnemySpawner και χρονικού περιορισμού (spawn_delay).

Για κάθε ενεργό εχθρό:

- ενημερώνεται η μορφή (EnemyFormSystem),
- εφαρμόζεται AI (EnemyAISystem),

- εκτελείται digging αν είναι Hobbin,
- ελέγχεται σύγκρουση με παίκτες.

Η AI βασίζεται σε pathfinding BFS και προσαρμόζεται δυναμικά στον πλησιέστερο ζωντανό παίκτη.

6.9 Όπλα, σφαίρες και συγκρούσεις

Η εκτόξευση σφαιρών γίνεται μέσω του WeaponSystem, το οποίο:

- ελέγχει cooldown,
- δημιουργεί νέα Bullet objects,
- αποδίδει ownership (p1 ή p2).

Το BulletSystem:

- μετακινεί σφαίρες,
- ελέγχει συγκρούσεις με εχθρούς και tiles,
- επιστρέφει λίστα kills.

Τα kills μεταφράζονται σε πόντους μέσω του scoring logic.

6.10 Gold bags και φυσική αλληλεπίδραση

Τα gold bags αποτελούν σύνθετο μηχανισμό:

- πέφτουν,
- μετατρέπονται σε gold pile,
- σκοτώνουν εχθρούς,
- συλλέγονται από παίκτες.

Η λογική αυτή ελέγχεται από:

- GoldBagSystem
- GoldBagPushSystem

Η επιλογή αυτή απομονώνει μία από τις πιο πολύπλοκες συμπεριφορές του παιχνιδιού σε ξεχωριστό subsystem.

6.11 Οπτικά εφέ και καθαρισμός κατάστασης

Οι εκρήξεις υλοποιούνται με particle system (ExplosionSystem) και ενεργοποιούνται:

- όταν πεθαίνει εχθρός,
- όταν χτυπιέται από falling gold bag.

Στο τέλος κάθε frame:

- αφαιρούνται νεκροί εχθροί,
- καθαρίζονται collected αντικείμενα,
- ενημερώνονται οι κάμερες.

Αυτό διασφαλίζει ότι η κατάσταση παραμένει συνεπής.

7. Τεχνητή Νοημοσύνη Εχθρών και Προσαρμογή Δυσκολίας

7.1 Εισαγωγή

Η τεχνητή νοημοσύνη (AI) των εχθρών αποτελεί έναν από τους πιο κρίσιμους μηχανισμούς του παιχνιδιού, καθώς επηρεάζει άμεσα τόσο το επίπεδο δυσκολίας όσο και την εμπειρία του παίκτη. Στην παρούσα υλοποίηση, η AI δεν περιορίζεται σε τυχαία κίνηση, αλλά βασίζεται σε στοχευμένη πλοήγηση, καταστάσεις συμπεριφοράς και δυναμική προσαρμογή στις συνθήκες του παιχνιδιού.

Η σχεδίαση της AI ακολουθεί αρχές καθαρής αρχιτεκτονικής, με σαφή διαχωρισμό:

- λήψης απόφασης,
- εκτέλεσης κίνησης,
- αλλαγής μορφής,
- και κλιμάκωσης δυσκολίας.

7.2 Αρχιτεκτονική της AI (Separation of Concerns)

Η τεχνητή νοημοσύνη των εχθρών δεν υλοποιείται σε μία ενιαία κλάση, αλλά διαμοιράζεται σε επιμέρους συστήματα:

- EnemyBrain: απόφαση κατεύθυνσης
- EnemyAISystem: εκτέλεση AI ανά enemy

- EnemyFormSystem: αλλαγή μορφής (Nobbin ↔ Hobbin)
- HobbinDiggingSystem: digging logic
- DifficultyScaler: προσαρμογή παραμέτρων δυσκολίας

Η επιλογή αυτή συνάδει πλήρως με την αρχή Single Responsibility Principle (SOLID) και επιτρέπει εύκολη συντήρηση και επέκταση.

7.3 EnemyBrain και λήψη αποφάσεων

Η κλάση EnemyBrain αποτελεί τον «εγκέφαλο» της AI και είναι υπεύθυνη αποκλειστικά για τον υπολογισμό της επόμενης κατεύθυνσης ενός εχθρού.

Η βασική μέθοδος decide(enemy, player, level, game_mode):

- λαμβάνει ως είσοδο την τρέχουσα κατάσταση,
- επιστρέφει μία κατεύθυνση (Direction) ή None.

Η απόφαση βασίζεται σε pathfinding BFS (Breadth-First Search) πάνω στο grid του επιπέδου.

7.4 Pathfinding με BFS σε tile-based περιβάλλον

Το BFS επιλέχθηκε για τους εξής λόγους:

- το περιβάλλον είναι grid-based,
- το κόστος κάθε κίνησης είναι ομοιόμορφο,
- απαιτείται η συντομότερη διαδρομή σε αριθμό βημάτων.

Η υλοποίηση:

- ξεκινά από τη θέση του εχθρού,
- εξερευνά γειτονικά tiles,
- σέβεται τους περιορισμούς ανάλογα με τη μορφή του enemy.

Η BFS δεν υπολογίζει πλήρη διαδρομή, αλλά μόνο το επόμενο βήμα, κάτι που:

- μειώνει υπολογιστικό κόστος,
- δημιουργεί πιο «ανθρώπινη» συμπεριφορά,

- αποφεύγει προβλέψιμες κινήσεις.

7.5 Μορφές εχθρών: Nobbin και Hobbin

Οι εχθροί διαθέτουν δύο διακριτές μορφές:

Nobbin

- κινείται μόνο σε TUNNEL,
- δεν μπορεί να σκάψει,
- έχει πιο προβλέψιμη συμπεριφορά.

Hobbin

- μπορεί να κινείται σε DIRT, TUNNEL, EMERALD,
- σκάβει ενεργά,
- αποτελεί σαφώς πιο επικίνδυνο αντίπαλο.

Η μορφή αποθηκεύεται στο enemy.form και επηρεάζει άμεσα:

- τις επιτρεπτές κινήσεις στο BFS,
- τη λογική digging,
- τη συνολική δυσκολία.

7.6 EnemyFormSystem και δυναμική μεταμόρφωση

Η κλάση EnemyFormSystem είναι υπεύθυνη για τη μετατροπή Nobbin → Hobbin μετά από συγκεκριμένο χρόνο.
Η επιλογή αυτή:

- εισάγει χρονική πίεση στον παίκτη,
- αποτρέπει παθητικό gameplay,
- αυξάνει σταδιακά την ένταση.

Η υλοποίηση βασίζεται σε timer (form_timer) και όχι σε random triggers, εξασφαλίζοντας ελεγχόμενη και προβλέψιμη συμπεριφορά.

7.7 AI σε Bonus Mode

Κατά τη διάρκεια του GameMode.BONUS, η συμπεριφορά των εχθρών αντιστρέφεται.

Αντί να κινούνται προς τον παίκτη:

- υπολογίζουν στόχο μακριά από αυτόν,
- χρησιμοποιούν την ίδια BFS λογική,
- αλλά με διαφορετικό goal.

Η προσέγγιση αυτή:

- επαναχρησιμοποιεί τον ίδιο αλγόριθμο,
- αποφεύγει duplication,
- προσφέρει σαφή διαφοροποίηση gameplay.

7.8 Επιλογή στόχου σε co-op

Σε παιχνίδι δύο παικτών, οι εχθροί δεν στοχεύουν σταθερά έναν παίκτη.

Η μέθοδος _get_closest_player(enemy):

- φιλτράρει ζωντανούς παίκτες,
- υπολογίζει Manhattan distance,
- επιλέγει τον πλησιέστερο.

Αυτό οδηγεί σε:

- πιο ρεαλιστική συμπεριφορά,
- δυναμική κατανομή πίεσης,
- αυξημένο co-op ενδιαφέρον.

7.9 Χρονισμός κίνησης εχθρών

Οι εχθροί δεν κινούνται κάθε frame. Αντίθετα:

- διαθέτουν move_timer,
- κινούνται μόνο όταν ξεπεραστεί enemy_delay.

Το enemy_delay:

- μειώνεται με την αύξηση επιπέδου,
- καθορίζεται από το DifficultyScaler.

Η επιλογή αυτή:

- αποτρέπει χαοτική κίνηση,
- επιτρέπει έλεγχο δυσκολίας,
- βελτιώνει την αναγνωσιμότητα της AI.

7.10 Κλιμάκωση δυσκολίας (DifficultyScaler)

Η κλάση DifficultyScaler αποτελεί τον κεντρικό μηχανισμό προσαρμογής:

- αριθμός εχθρών ανά επίπεδο,
- μέγιστοι ενεργοί εχθροί,
- καθυστέρηση κίνησης,
- ύπαρξη επόμενου επιπέδου.

Η προσέγγιση αυτή:

- διαχωρίζει πλήρως gameplay logic από balancing,
- επιτρέπει εύκολη τροποποίηση,
- διευκολύνει δοκιμές (playtesting).

8. Αρχιτεκτονική Λογισμικού: MVC, DAO και Αρχές SOLID

Η ανάπτυξη του παιχνιδιού βασίστηκε εξαρχής στην αντίληψη ότι ακόμη και ένα arcade-style παιχνίδι πραγματικού χρόνου αποτελεί ένα πλήρες λογισμικό σύστημα, το οποίο οφείλει να σχεδιαστεί με αρχιτεκτονική συνέπεια και όχι ως μια συλλογή αποσπασματικών μηχανισμών. Για τον λόγο αυτό, η υλοποίηση δεν περιορίστηκε στην επίτευξη λειτουργικότητας, αλλά οργανώθηκε γύρω από καθιερωμένα αρχιτεκτονικά πρότυπα, με στόχο τη διαφάνεια, τη συντηρησιμότητα και τη μελλοντική επεκτασιμότητα του κώδικα. Στο πλαίσιο αυτό, υιοθετήθηκε ένας συνδυασμός του προτύπου Model–View–Controller (MVC), του προτύπου Data Access Object (DAO) και των βασικών αρχών SOLID, προσαρμοσμένων στις ιδιαιτερότητες ενός παιχνιδιού που εκτελείται σε συνεχή βρόχο ενημέρωσης (game loop).

Η λογική του MVC εφαρμόστηκε με σαφή και συνειδητό τρόπο. Το Model περιλαμβάνει όλα τα αντικείμενα που αναπαριστούν την κατάσταση του παιχνιδιού και τους κανόνες

που τη διέπουν. Κλάσεις όπως Player, Enemy, Level, GoldBag, Score, Lives, Weapon και Bullet συγκροτούν τον πυρήνα του μοντέλου δεδομένων. Τα αντικείμενα αυτά είναι πλήρως αποσυνδεδεμένα από τη βιβλιοθήκη pygame και δεν περιέχουν καμία γνώση σχετικά με τον τρόπο απεικόνισής τους ή τον τρόπο λήψης εισόδου από τον χρήστη. Ο ρόλος τους περιορίζεται αποκλειστικά στη διατήρηση κατάστασης και στην υλοποίηση βασικής συμπεριφοράς που απορρέει από τους κανόνες του παιχνιδιού. Με τον τρόπο αυτό, το Model παραμένει ανεξάρτητο, επαναχρησιμοποιήσιμο και εύκολα κατανοητό, ενώ ταυτόχρονα καθίσταται δυνατή η μελλοντική αντικατάσταση ή επέκτασή του χωρίς επιπτώσεις στη διεπαφή χρήστη.

Το View αναλαμβάνει αποκλειστικά την οπτικοποίηση της κατάστασης του παιχνιδιού. Η σχεδίαση του συστήματος απεικόνισης βασίζεται στη σαφή διάκριση μεταξύ δεδομένων και παρουσίασης. Κλάσεις όπως TilemapView, Camera2D και τα επιμέρους sprite classes για παίκτες, εχθρούς, αντικείμενα και εφέ λειτουργούν αποκλειστικά ως αναγνώστες της κατάστασης του Model. Δεν τροποποιούν δεδομένα, δεν λαμβάνουν αποφάσεις gameplay και δεν επηρεάζουν τη ροή του παιχνιδιού. Απλώς μεταφράζουν τη λογική κατάσταση του κόσμου σε οπτική αναπαράσταση στην οθόνη. Αυτή η προσέγγιση επιτρέπει την ανεξάρτητη εξέλιξη της αισθητικής και της λογικής του παιχνιδιού, ενώ μειώνει δραστικά τον κίνδυνο εισαγωγής σφαλμάτων κατά την τροποποίηση του γραφικού μέρους.

Ο ρόλος του Controller κατανέμεται σε περισσότερα του ενός υποσυστήματα, γεγονός που αντανακλά τη φύση των real-time παιχνιδιών. Η είσοδος του χρήστη συλλέγεται αρχικά μέσω ενός εξειδικευμένου μηχανισμού και μετατρέπεται σε ένα ουδέτερο αντικείμενο στιγμιότυπου εισόδου (InputSnapshot). Το αντικείμενο αυτό μεταφέρεται στα scenes και καταναλώνεται από τη λογική του παιχνιδιού χωρίς άμεση εξάρτηση από γεγονότα της pygame. Παράλληλα, ο SceneManager λειτουργεί ως κεντρικός ελεγκτής καταστάσεων, διαχειριζόμενος τις μεταβάσεις μεταξύ menu, gameplay, ρυθμίσεων και οθονών τερματισμού. Η προσέγγιση αυτή εξασφαλίζει καθαρό κύκλο ζωής για κάθε scene και αποτρέπει τη σύγχυση λογικής που συχνά παρατηρείται σε μικρά παιχνίδια χωρίς σαφή δομή.

Ιδιαίτερη σημασία έχει η υιοθέτηση scene-based αρχιτεκτονικής, η οποία μπορεί να θεωρηθεί ως εξειδίκευση του MVC για εφαρμογές ψυχαγωγίας. Κάθε κατάσταση του παιχνιδιού υλοποιείται ως ανεξάρτητη κλάση που ακολουθεί ένα κοινό συμβόλαιο, το οποίο περιλαμβάνει μεθόδους για είσοδο, έξοδο, επεξεργασία εισόδου, ενημέρωση και απόδοση γραφικών. Με τον τρόπο αυτό, οι σκηνές είναι πλήρως απομονωμένες μεταξύ τους και η μετάβαση από τη μία στην άλλη πραγματοποιείται με ελεγχόμενο και προβλέψιμο τρόπο. Η σχεδίαση αυτή ευθυγραμμίζεται με πρακτικές που συναντώνται σε σύγχρονες game engines και καθιστά τον κώδικα ευανάγνωστο ακόμη και σε τρίτους προγραμματιστές.

Παράλληλα με το MVC, εφαρμόστηκε το πρότυπο Data Access Object για τη διαχείριση των δεδομένων υψηλών σκορ. Η αποθήκευση και ανάκτηση δεδομένων από τη βάση SQLite υλοποιείται αποκλειστικά μέσω της κλάσης ScoreRepository. Η κλάση αυτή αποτελεί το μοναδικό σημείο του συστήματος που γνωρίζει την ύπαρξη SQL, τον τρόπο δημιουργίας πινάκων και τη σύνταξη ερωτημάτων. Όλα τα υπόλοιπα τμήματα του παιχνιδιού αλληλεπιδρούν με τα δεδομένα υψηλών σκορ μέσω αφηρημένων μεθόδων υψηλού επιπέδου. Η επιλογή αυτή προστατεύει τη λογική του παιχνιδιού από αλλαγές στο σύστημα αποθήκευσης και συνάδει με πρακτικές που συναντώνται σε μεγαλύτερα πληροφοριακά συστήματα.

Οι αρχές SOLID διατρέχουν οριζόντια ολόκληρη την υλοποίηση. Κάθε κλάση έχει σαφή και περιορισμένη ευθύνη, γεγονός που καθιστά τον κώδικα ευανάγνωστο και εύκολα επεκτάσιμο. Τα συστήματα τεχνητής νοημοσύνης των εχθρών, η μεταβολή μορφής τους, η κίνηση στο grid, η αλληλεπίδραση με τα tiles και η διαχείριση των αντικειμένων χρυσού υλοποιούνται ως ανεξάρτητα services, τα οποία συνεργάζονται χωρίς να αλληλοεξαρτώνται. Η προσέγγιση αυτή ευθυγραμμίζεται με την αρχή Single Responsibility και αποτρέπει τη δημιουργία υπερφορτωμένων κλάσεων που συγκεντρώνουν ετερόκλητη λογική.

Η αρχή Open–Closed εφαρμόζεται μέσω της δυνατότητας επέκτασης της συμπεριφοράς χωρίς τροποποίηση του υφιστάμενου κώδικα. Για παράδειγμα, η προσθήκη νέων τύπων εχθρών, νέων επιπέδων δυσκολίας ή εναλλακτικών στρατηγικών τεχνητής νοημοσύνης μπορεί να πραγματοποιηθεί με την εισαγωγή νέων κλάσεων ή

παραμέτρων, χωρίς την ανάγκη αναδόμησης του βασικού game loop. Αντίστοιχα, η αρχή Dependency Inversion εφαρμόζεται μέσω της έγχυσης εξαρτήσεων, όπως στην περίπτωση του EnemyAISystem, το οποίο λαμβάνει τον EnemyBrain ως εξωτερική συνιστώσα και όχι ως εσωτερική υλοποίηση.

9. Οπτικά και Ηχητικά Assets – Παραγωγή, Επεξεργασία και Άδειες Χρήσης

Όλα τα γραφικά assets του παιχνιδιού, συμπεριλαμβανομένων των sprites των παικτών, των εχθρών, των αντικειμένων (emeralds, gold bags, gold piles), καθώς και των tiles του επιπέδου, δημιουργήθηκαν από την ομάδα με τη χρήση μεγάλων γλωσσικών μοντέλων παραγωγής εικόνας (LLM-based image generation), σε συνδυασμό με εκτεταμένη χειροκίνητη επεξεργασία. Τα παραγόμενα γραφικά χρησιμοποιήθηκαν ως αρχικό υλικό, το οποίο στη συνέχεια τροποποιήθηκε και προσαρμόστηκε ώστε να πληροί τις αισθητικές και τεχνικές απαιτήσεις του παιχνιδιού.

Η τελική επεξεργασία των εικόνων πραγματοποιήθηκε στο λογισμικό Photopea, το οποίο χρησιμοποιήθηκε για τη ρύθμιση διαστάσεων, τη μετατροπή των εικόνων σε pixel-art αισθητική, την ενοποίηση χρωματικής παλέτας και τη διασφάλιση ομοιομορφίας μεταξύ των διαφορετικών sprites. Ιδιαίτερη έμφαση δόθηκε στη σωστή κλίμακα των γραφικών (32×32 pixels), ώστε να είναι συμβατά με το grid-based σύστημα κίνησης και απεικόνισης που υλοποιήθηκε στον πυρήνα του παιχνιδιού. Η διαδικασία αυτή δεν ήταν αυτοματοποιημένη, αλλά απαιτούσε επαναληπτικές παρεμβάσεις και αισθητικές αποφάσεις από την ομάδα, γεγονός που καθιστά τα γραφικά προϊόν ανθρώπινης δημιουργικής εργασίας και όχι απλή αυτόματη παραγωγή.

Αντίστοιχα, το σύνολο των ηχητικών εφέ του παιχνιδιού δημιουργήθηκε από την ομάδα με χρήση του λογισμικού Audacity. Οι ήχοι που συνοδεύουν βασικές ενέργειες, όπως η κίνηση στο μενού, η συλλογή αντικειμένων, ο πυροβολισμός και οι εκρήξεις, σχεδιάστηκαν είτε μέσω σύνθεσης απλών κυματομορφών είτε μέσω επεξεργασίας αρχικών ηχητικών δειγμάτων που παρήχθησαν ειδικά για τον σκοπό αυτό. Κατά τη διαδικασία επεξεργασίας εφαρμόστηκαν φίλτρα, μεταβολές έντασης, χρονικές

περικοπές και εξομάλυνση, ώστε τα ηχητικά εφέ να είναι σύντομα, καθαρά και κατάλληλα για real-time αναπαραγωγή χωρίς να κουράζουν τον χρήστη.

Η βασική μελωδία του παιχνιδιού συντέθηκε από την ομάδα, με στόχο να αποδίδει τον ρυθμικό, επαναληπτικό και ελαφρώς αγωνιώδη χαρακτήρα που είναι χαρακτηριστικός των arcade παιχνιδιών. Η σύνθεση πραγματοποιήθηκε με τη χρήση ψηφιακών εργαλείων μουσικής παραγωγής και στη συνέχεια προσαρμόστηκε τεχνικά ώστε να αναπαράγεται σε βρόχο (loop) χωρίς ακουστές ασυνέχειες. Η ένταση της μουσικής ρυθμίστηκε προσεκτικά σε χαμηλά επίπεδα, ώστε να λειτουργεί υποστηρικτικά και να μην επικαλύπτει τα ηχητικά εφέ ή να αποσπά την προσοχή του παίκτη από το gameplay.

Όσον αφορά τα θέματα αδειοδότησης, το σύνολο των οπτικών και ηχητικών assets του παιχνιδιού είναι ελεύθερο από περιορισμούς τρίτων, καθώς δημιουργήθηκε αποκλειστικά από την ομάδα. Δεν χρησιμοποιήθηκαν έτοιμα εμπορικά sprites, βιβλιοθήκες ή ηχητικά δείγματα που να απαιτούν αναφορά σε εξωτερικούς δημιουργούς ή συγκεκριμένες άδειες χρήσης. Η χρήση εργαλείων παραγωγής περιεχομένου βασισμένων σε μεγάλα γλωσσικά μοντέλα περιορίστηκε στη φάση της αρχικής δημιουργίας υλικού και δεν παραβιάζει πνευματικά δικαιώματα.

10. Χρήση Μεγάλων Γλωσσικών Μοντέλων (LLMs) κατά την Ανάπτυξη του Παιχνιδιού

Κατά την υλοποίηση του παιχνιδιού αξιοποιήθηκαν εργαλεία βασισμένα σε Μεγάλα Γλωσσικά Μοντέλα (Large Language Models – LLMs) με υποστηρικτικό και επικουρικό ρόλο, χωρίς να υποκαθιστούν τον σχεδιασμό, την αρχιτεκτονική απόφαση ή τη βασική συγγραφική εργασία του κώδικα από την ομάδα. Η χρήση τους εντάχθηκε σε συγκεκριμένα στάδια της ανάπτυξης, με σαφώς οριοθετημένο σκοπό, και λειτούργησε ως εργαλείο επιτάχυνσης, επαλήθευσης και αναλυτικής υποστήριξης, ιδιαίτερα σε τμήματα αυξημένης πολυπλοκότητας.

Ένας από τους βασικούς τομείς όπου χρησιμοποιήθηκαν LLMs ήταν ο υπολογισμός και η επαλήθευση συντεταγμένων και μετασχηματισμών στο πλαίσιο του grid-based

συστήματος του παιχνιδιού. Δεδομένου ότι το gameplay βασίζεται σε διακριτά tiles και σε μετατροπές από συντεταγμένες κόσμου (world coordinates) σε συντεταγμένες οθόνης (screen coordinates), τα LLMs αξιοποιήθηκαν για τον έλεγχο μαθηματικών σχέσεων, την επιβεβαίωση ορθότητας τύπων και την αποσαφήνιση οριακών περιπτώσεων (off-by-one errors, όρια κάμερας, split-screen προβολή). Στα σημεία αυτά, η τελική υλοποίηση και ενσωμάτωση έγινε πάντοτε χειροκίνητα, με βάση τις αρχιτεκτονικές αποφάσεις της ομάδας.

Επιπλέον, τα LLMs χρησιμοποιήθηκαν για τη συγγραφή βοηθητικού κώδικα μικρής κλίμακας, κυρίως σε επαναλαμβανόμενα ή μη κρίσιμα τμήματα, όπως αρχικοί σκελετοί μεθόδων, πρότυπα (boilerplate) για scenes, ή απλές βοηθητικές συναρτήσεις. Σε όλες τις περιπτώσεις, ο παραγόμενος κώδικας δεν ενσωματώθηκε αυτούσιος, αλλά αποτέλεσε αφετηρία για περαιτέρω τροποποίηση, αναδιάρθρωση και προσαρμογή στο υπάρχον σύστημα κλάσεων, ώστε να τηρείται η συνοχή με την αρχιτεκτονική MVC, τη λογική των services και τις αρχές SOLID που εφαρμόστηκαν στο έργο.

Ιδιαίτερα σημαντική ήταν η συνεισφορά των LLMs στη φάση ανάλυσης και κατανόησης αλγοριθμικών ροών, ειδικά σε ό,τι αφορά τη συμπεριφορά των εχθρών, τη διαχείριση πολλαπλών παικτών και τις αλληλεπιδράσεις μεταξύ αντικειμένων (gold bags, enemies, bullets). Τα μοντέλα χρησιμοποιήθηκαν ως εργαλείο διαλόγου και αναλυτικής σκέψης, βοηθώντας στην αποσύνθεση πολύπλοκων σεναρίων, στον έλεγχο λογικών ακολουθιών και στην επισήμανση πιθανών edge cases. Ωστόσο, η τελική μορφή των αλγορίθμων, καθώς και η απόφαση για το πού ανήκει κάθε ευθύνη (scene, model, service), προέκυψε από συνειδητές σχεδιαστικές επιλογές της ομάδας.

Ένα ακόμη πεδίο όπου τα LLMs αποδείχθηκαν ιδιαίτερα χρήσιμα ήταν το debugging. Κατά την ανάπτυξη του παιχνιδιού προέκυψαν σύνθετα σφάλματα που σχετίζονταν με τη χρονική αλληλεπίδραση συστημάτων (timers, cooldowns, invulnerability windows), με τη διαχείριση κατάστασης παικτών σε co-op λειτουργία και με μεταβάσεις μεταξύ scenes. Σε αυτές τις περιπτώσεις, τα LLMs χρησιμοποιήθηκαν για την ανάλυση logs, την ερμηνεία αποσπασμάτων κώδικα και τη διατύπωση εναλλακτικών υποθέσεων για την αιτία ενός σφάλματος. Η διαδικασία αυτή δεν αντικατέστησε τον κλασικό έλεγχο και

την πειραματική επιβεβαίωση, αλλά λειτούργησε ως εργαλείο ενίσχυσης της αναλυτικής διαδικασίας.

Αξίζει να τονιστεί ότι η χρήση LLMs δεν επηρέασε αρνητικά την πρωτοτυπία ή την ακεραιότητα του έργου. Αντιθέτως, χρησιμοποιήθηκαν με διαφάνεια και με τρόπο αντίστοιχο με τη χρήση τεχνικών εγχειριδίων, τεκμηρίωσης ή εργαλείων αυτόματης συμπλήρωσης κώδικα (IDE assistance). Δεν χρησιμοποιήθηκαν για την αυτόματη παραγωγή πλήρων αρχείων ή για την αντιγραφή έτοιμων λύσεων, αλλά ως μέσο υποστήριξης της σκέψης, της δομής και της τεχνικής τεκμηρίωσης.

11. Κάλυψη και Αξιολόγηση των Προδιαγραφών της Εκφώνησης

Στο παρόν κεφάλαιο αναλύεται συστηματικά ο βαθμός στον οποίο το αναπτυγμένο παιχνίδι καλύπτει τις ελάχιστες και τις επιθυμητές λειτουργικές προδιαγραφές που τέθηκαν στην εκφώνηση της εργασίας. Για κάθε προδιαγραφή εξετάζεται ο τρόπος υλοποίησης, η σχεδιαστική προσέγγιση που ακολουθήθηκε, καθώς και οι δυνατότητες βελτίωσης ή επέκτασης, όπου αυτό κρίνεται σκόπιμο.

Προδιαγραφή 1: Να έχει τουλάχιστον έναν παίκτη

Η συγκεκριμένη προδιαγραφή καλύπτεται πλήρως, καθώς το παιχνίδι έχει σχεδιαστεί εξαρχής γύρω από έναν βασικό ανθρώπινο παίκτη (Player 1). Ο παίκτης αυτός αποτελεί κεντρική οντότητα του συστήματος και διαθέτει σαφώς ορισμένες ιδιότητες, όπως θέση στο επίπεδο, κατεύθυνση κίνησης, αριθμό ζωών και σκορ. Η αλληλεπίδρασή του με το περιβάλλον, τα αντικείμενα και τους εχθρούς υλοποιείται σε πραγματικό χρόνο και αποτελεί τον πυρήνα της λογικής του παιχνιδιού. Η ύπαρξη του παίκτη ενσωματώνεται τόσο στο επίπεδο του μοντέλου δεδομένων όσο και στο επίπεδο των scenes και των συστημάτων (services).

Προδιαγραφή 2: Να υπάρχει κάποιου είδους ταμπλό ή σκηνή

Η απαίτηση για ύπαρξη ταμπλό ή σκηνής ικανοποιείται μέσω της υλοποίησης ενός grid-based επιπέδου, το οποίο λειτουργεί ως το βασικό περιβάλλον παιχνιδιού. Το επίπεδο αποτελείται από διακριτά tiles που αναπαριστούν διαφορετικούς τύπους εδάφους και αντικειμένων, όπως χώμα, τούνελ, σμαράγδια και σακιά χρυσού. Το ταμπλό δεν είναι

στατικό, αλλά μεταβάλλεται δυναμικά κατά τη διάρκεια του παιχνιδιού, καθώς ο παίκτης σκάβει, συλλέγει αντικείμενα και αλληλεπιδρά με εχθρούς. Παράλληλα, η έννοια της σκηνής επεκτείνεται και σε μη παιχνιδικές σκηνές, όπως το κεντρικό μενού, η οθόνη οδηγιών, η οθόνη υψηλών σκορ και η οθόνη game over.

Προδιαγραφή 3: Να υπάρχουν επίπεδα δυσκολίας

Τα επίπεδα δυσκολίας υλοποιούνται μέσω ενός μηχανισμού κλιμακούμενης δυσκολίας, ο οποίος βασίζεται στην πρόοδο του παίκτη στα επίπεδα του παιχνιδιού. Με την αύξηση του επιπέδου μεταβάλλονται κρίσιμες παράμετροι, όπως ο συνολικός αριθμός εχθρών, ο μέγιστος αριθμός ταυτόχρονα ενεργών εχθρών και ο ρυθμός κίνησής τους. Η προσέγγιση αυτή ενσωματώνει τη δυσκολία οργανικά στη ροή του παιχνιδιού, ακολουθώντας τη φιλοσοφία των κλασικών arcade παιχνιδιών. Παρότι δεν παρέχεται ρητή επιλογή δυσκολίας από τον χρήστη, η λειτουργική απαίτηση καλύπτεται επαρκώς και μπορεί να επεκταθεί εύκολα στο μέλλον.

Προδιαγραφή 4: Να υπάρχει κάποιο είδος σκορ για τον κάθε παίκτη

Το παιχνίδι περιλαμβάνει πλήρως υλοποιημένο σύστημα σκορ, το οποίο ενημερώνεται δυναμικά με βάση τις ενέργειες του παίκτη. Η συλλογή σμαραγδιών, η εξόντωση εχθρών και η συλλογή χρυσού αποδίδουν συγκεκριμένο αριθμό πόντων. Στην περίπτωση του multiplayer mode, το σκορ διατηρείται ξεχωριστά για κάθε παίκτη, επιτρέποντας ανεξάρτητη αξιολόγηση της απόδοσής τους. Το σύστημα σκορ υλοποιείται μέσω ανεξάρτητης κλάσης και αντίστοιχου service, γεγονός που ενισχύει την καθαρότητα και τη συντηρησιμότητα του κώδικα.

Προδιαγραφή 5: Να αποθηκεύονται τα δεδομένα σε κάποια δομή δεδομένων

Η αποθήκευση δεδομένων καλύπτεται τόσο σε επίπεδο μνήμης όσο και σε επίπεδο μόνιμης αποθήκευσης. Κατά την εκτέλεση του παιχνιδιού, δεδομένα όπως σκορ, ζωές και κατάσταση επιπέδου αποθηκεύονται σε κατάλληλες δομές και αντικείμενα. Παράλληλα, τα υψηλά σκορ αποθηκεύονται σε μόνιμη μορφή μέσω ειδικού μηχανισμού αποθήκευσης, επιτρέποντας την ανάκτησή τους μεταξύ διαφορετικών εκτελέσεων. Αν

και δεν χρησιμοποιείται εξωτερική βάση δεδομένων, η απαίτηση της εκφώνησης ικανοποιείται πλήρως.

Προδιαγραφή 6: Να υπάρχουν γραφικά

Η απαίτηση για ύπαρξη γραφικών καλύπτεται εκτενώς, καθώς το παιχνίδι περιλαμβάνει δισδιάστατα sprites για παίκτες, εχθρούς, αντικείμενα και περιβάλλον. Επιπλέον, υλοποιείται σύστημα κάμερας που υποστηρίζει scrolling καθώς και split-screen προβολή για το multiplayer mode. Τα γραφικά δεν είναι απλώς αισθητικό στοιχείο, αλλά αποτελούν λειτουργικό μέρος του παιχνιδιού, καθώς αντικατοπτρίζουν την κατάσταση των οντοτήτων και του περιβάλλοντος σε πραγματικό χρόνο.

Προδιαγραφή 7: Να υπάρχει έστω ένας ήχος

Το παιχνίδι περιλαμβάνει ηχητικά εφέ για βασικές ενέργειες, όπως πυροβολισμούς, συλλογή αντικειμένων και επιλογές μενού. Οι ήχοι αυτοί προσφέρουν άμεση ανατροφοδότηση στον παίκτη και ενισχύουν τη συνολική εμπειρία χρήστη. Η ύπαρξη τουλάχιστον ενός ήχου, όπως απαιτείται από την εκφώνηση, καλύπτεται πλήρως.

Προδιαγραφή 8: Περισσότεροι του ενός ανθρώπινοι παίκτες (multi-player)

Η υποστήριξη περισσότερων του ενός ανθρώπινων παικτών υλοποιείται μέσω τοπικού multiplayer (co-op) mode με split screen. Οι παίκτες μοιράζονται τον ίδιο κόσμο, αλλά διαθέτουν ανεξάρτητες κάμερες, σκορ και ζωές. Η λειτουργικότητα αυτή αυξάνει σημαντικά την πολυπλοκότητα της υλοποίησης, τόσο σε επίπεδο λογικής όσο και σε επίπεδο απόδοσης, και καλύπτει την επιθυμητή προδιαγραφή σε υψηλό βαθμό.

Προδιαγραφή 9: Εχθροί ή σύμμαχοι ελεγχόμενοι από τον υπολογιστή με ευφυΐα

Οι εχθροί του παιχνιδιού ελέγχονται από σύστημα τεχνητής νοημοσύνης, το οποίο λαμβάνει αποφάσεις κίνησης και στόχευσης με βάση τη θέση των παικτών και την κατάσταση του παιχνιδιού. Παρότι η τεχνητή νοημοσύνη δεν είναι σύνθετη, βασίζεται σε μη τυχαία λογική και απλές ευρετικές, προσφέροντας λειτουργική συμπεριφορά.

Προδιαγραφή 10: Σύνδεση με βάση δεδομένων

Η σύνδεση με βάση δεδομένων υλοποιείται στην παρούσα έκδοση me sqlite. Ωστόσο, η αρχιτεκτονική του συστήματος, και ειδικότερα ο διαχωρισμός της αποθήκευσης δεδομένων σε ανεξάρτητα repositories, επιτρέπει τη μελλοντική ενσωμάτωση βάσης δεδομένων χωρίς εκτεταμένες αλλαγές στον υπόλοιπο κώδικα.

Προδιαγραφή 11: Πραγματικού χρόνου λειτουργία (όχι turn-based)

Το παιχνίδι λειτουργεί σε πραγματικό χρόνο, βασιζόμενο σε συνεχή βρόχο ενημέρωσης (update loop). Όλες οι κινήσεις, συγκρούσεις και αλληλεπιδράσεις υπολογίζονται δυναμικά με βάση τον χρόνο, καλύπτοντας πλήρως τη συγκεκριμένη προδιαγραφή.

Προδιαγραφή 12: IPv4 σύνδεση για multiplayer mode

Η δυνατότητα multiplayer μέσω **IPv4 σύνδεσης δεν υλοποιείται,** καθώς το παιχνίδι περιορίζεται σε τοπικό multiplayer. Η σχεδίαση του λογισμικού επιτρέπει με μερικές αλλαγές την ενσωμάτωση διαδικτυακού παιχνιδιού.

Προδιαγραφή 13: Έλεγχοι εξαιρέσεων και προβληματικών καταστάσεων

Το παιχνίδι έχει σχεδιαστεί έτσι ώστε να αποφεύγονται κρίσιμες καταστάσεις που θα οδηγούσαν σε tracebacks ή stack dumps. Οι μεταβάσεις μεταξύ σκηνών, οι απώλειες ζωών και οι καταστάσεις game over διαχειρίζονται ομαλά. Αν και δεν γίνεται εκτεταμένη χρήση μηχανισμών exception handling, η συνολική σχεδίαση διασφαλίζει σταθερή λειτουργία κατά την κανονική χρήση.

12. Συμπεράσματα & Μελλοντική Εργασία

Η παρούσα εργασία είχε ως στόχο τον σχεδιασμό και την υλοποίηση ενός ολοκληρωμένου δισδιάστατου παιχνιδιού πραγματικού χρόνου, βασισμένου σε αρχές σύγχρονης αντικειμενοστρεφούς σχεδίασης και καθαρής αρχιτεκτονικής λογισμικού. Το τελικό αποτέλεσμα συνδυάζει λειτουργικότητα, επεκτασιμότητα και παικτική εμπειρία, καλύπτοντας πλήρως τις ελάχιστες απαιτήσεις της εκφώνησης και ένα σημαντικό μέρος των επιθυμητών προδιαγραφών.

Σε επίπεδο υλοποίησης, το παιχνίδι ενσωματώνει έναν πλήρη κύκλο ζωής, από το κεντρικό μενού και τις βοηθητικές σκηνές έως το κύριο gameplay, τη διαχείριση επιπέδων και την ομαλή μετάβαση σε κατάσταση game over. Η ύπαρξη συστήματος σκορ, ζωών, κλιμακούμενης δυσκολίας και τεχνητής νοημοσύνης για τους εχθρούς προσδίδει βάθος και συνέπεια στη συνολική εμπειρία. Ιδιαίτερη σημασία έχει η υποστήριξη τοπικού multiplayer, η οποία αυξάνει σημαντικά την πολυπλοκότητα της σχεδίασης αλλά ταυτόχρονα αναδεικνύει την ευελιξία της αρχιτεκτονικής που επιλέχθηκε.

Από αρχιτεκτονικής πλευράς, η εφαρμογή ακολουθεί με σαφήνεια το πρότυπο MVC, με διακριτό διαχωρισμό μεταξύ μοντέλου δεδομένων, λογικής ελέγχου και παρουσίασης. Παράλληλα, η χρήση DAO και repository-like δομών για την αποθήκευση σκορ και δεδομένων παιχνιδιού επιτρέπει την απομόνωση της επίμονης πληροφορίας από τον υπόλοιπο κώδικα. Οι αρχές SOLID εφαρμόζονται σε μεγάλο βαθμό, κυρίως μέσω της διάσπασης της λογικής σε εξειδικευμένα συστήματα (services), όπως κίνηση, τεχνητή νοημοσύνη, όπλα, συγκρούσεις και διαχείριση αντικειμένων. Η προσέγγιση αυτή οδηγεί σε κώδικα πιο κατανοητό, ευκολότερο στη συντήρηση και σαφώς επεκτάσιμο.

Η χρήση εργαλείων LLM κατά την ανάπτυξη της εφαρμογής λειτούργησε υποστηρικτικά και όχι καθοριστικά. Τα μοντέλα αξιοποιήθηκαν κυρίως για τη συγγραφή βοηθητικού κώδικα, τον υπολογισμό γεωμετρικών ή χρονικών παραμέτρων, την αρχική διατύπωση κειμένων για τις σκηνές και τη διαδικασία debugging. Η τελική αρχιτεκτονική, η λογική ροή του παιχνιδιού και οι κρίσιμες σχεδιαστικές αποφάσεις ελήφθησαν από την ομάδα ανάπτυξης, με ανθρώπινη κρίση και εμπειρία στον προγραμματισμό παιχνιδιών.

Παρά τον υψηλό βαθμό ολοκλήρωσης, υπάρχουν σαφείς δυνατότητες περαιτέρω βελτίωσης και επέκτασης. Μία προφανής κατεύθυνση μελλοντικής εργασίας είναι η προσθήκη δικτυακού multiplayer μέσω IPv4, το οποίο θα απαιτούσε συγχρονισμό κατάστασης, διαχείριση καθυστερήσεων και ανασχεδίαση μέρους της λογικής ενημέρωσης. Επιπλέον, η ενσωμάτωση πραγματικής βάσης δεδομένων για την

αποθήκευση σκορ και στατιστικών παικτών θα αναβάθμιζε το σύστημα persistence και θα ενίσχυε τη διαχρονικότητα του παιχνιδιού.

Άλλες πιθανές επεκτάσεις περιλαμβάνουν την εισαγωγή περισσότερων τύπων εχθρών με πιο σύνθετη τεχνητή νοημοσύνη, την προσθήκη επιλογών δυσκολίας από το μενού, τη βελτίωση των οπτικών εφέ και των animations, καθώς και την υποστήριξη παραμετροποιήσιμων χειρισμών. Σε επίπεδο λογισμικού, θα μπορούσε να ενισχυθεί η χρήση μηχανισμών exception handling και logging, ώστε να βελτιωθεί περαιτέρω η ανθεκτικότητα του συστήματος σε απρόβλεπτες καταστάσεις.

Συνοψίζοντας, το παιχνίδι που αναπτύχθηκε αποτελεί ένα πλήρες και τεχνικά τεκμηριωμένο παράδειγμα εφαρμογής αρχών μηχανικής λογισμικού στον χώρο της ανάπτυξης παιχνιδιών. Η εργασία αυτή δεν εξαντλεί τις δυνατότητες του έργου, αλλά θέτει ένα σταθερό και επεκτάσιμο υπόβαθρο, πάνω στο οποίο μπορούν να χτιστούν μελλοντικές βελτιώσεις τόσο σε λειτουργικό όσο και σε ερευνητικό επίπεδο.

13. Βιβλιογραφία

Atari, Inc. (1983). *Digger* [Video game]. Atari.

Björk, S., & Holopainen, J. (2005). *Patterns in game design*. Charles River Media.

Brooks, F. P. (1995). *The mythical man-month: Essays on software engineering* (Anniversary ed.). Addison-Wesley.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley.

Hunicke, R., LeBlanc, M., & Zubek, R. (2004). MDA: A formal approach to game design and game research. In *Proceedings of the AAAI Workshop on Challenges in Game AI* (pp. 1–5).

Larman, C. (2004). *Applying UML and patterns: An introduction to object-oriented analysis and design and iterative development* (3rd ed.). Prentice Hall.

Martin, R. C. (2003). *Agile software development: Principles, patterns, and practices*. Prentice Hall.

Martin, R. C. (2018). *Clean architecture: A craftsman's guide to software structure and design*. Prentice Hall.

McConnell, S. (2004). *Code complete* (2nd ed.). Microsoft Press.

Millington, I., & Funge, J. (2016). *Artificial intelligence for games* (2nd ed.). CRC Press.

Parker, J. R. (2019). *Game design fundamentals*. Mercury Learning & Information.

Perez, D., & Lucas, S. M. (2014). Rolling horizon evolution versus tree search for navigation in single-player real-time games. In *Proceedings of the 2014 Genetic and Evolutionary Computation Conference* (pp. 351–358). ACM.

Pygame Community. (2024). *Pygame documentation*. https://www.pygame.org/docs/

Russell, S., & Norvig, P. (2021). *Artificial intelligence: A modern approach* (4th ed.). Pearson.

Schell, J. (2015). *The art of game design: A book of lenses* (2nd ed.). CRC Press.

Shore, J., & Warden, S. (2007). *The art of agile development*. O'Reilly Media.

Sommerville, I. (2016). *Software engineering* (10th ed.). Pearson.

Van Rossum, G., & Drake, F. L. (2009). *Python 3 reference manual*. CreateSpace.

Yannakakis, G. N., & Togelius, J. (2018). *Artificial intelligence and games*. Springer.

Zhang, Y., Chen, T., & Yu, Z. (2023). Large language models for software engineering: A systematic literature review. *IEEE Transactions on Software Engineering*, 49(10), 1–20.