



Τεχνολογικό
Πανεπιστήμιο
Κύπρου

Σχολή Μηχανικής
και Τεχνολογίας

Πτυχιακή εργασία

ΟΝΤΟΛΟΓΙΚΗ ΜΟΝΤΕΛΟΠΟΙΗΣΗ ΓΝΩΣΗΣ ΣΕ ΕΚΠΑΙΔΕΥΤΙΚΟ ΠΑΙΧΝΙΔΙ 2D

Ανάπτυξη Serious Game με Ενσωμάτωση OWL/RDF Οντολογίας στο Unity

Του

ΧΡΙΣΤΟΥ ΚΩΝΣΤΑΝΤΙΝΟΥ

Αριθμός Φοιτητικής Ταυτότητας: 25327

Λεμεσός, 05-2026

ΤΕΧΝΟΛΟΓΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

**ΤΜΗΜΑ ΗΛ. ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΗΛ. ΥΠΟΛΟΓΙΣΤΩΝ &
ΠΛΗΡΟΦΟΡΙΚΗΣ**

Πτυχιακή εργασία

**ΟΝΤΟΛΟΓΙΚΗ ΜΟΝΤΕΛΟΠΟΙΗΣΗ ΓΝΩΣΗΣ ΣΕ
ΕΚΠΑΙΔΕΥΤΙΚΟ ΠΑΙΧΝΙΔΙ 2D**

Ανάπτυξη Serious Game με Ενσωμάτωση OWL/RDF Οντολογίας στο Unity

του

ΧΡΙΣΤΟΥ ΚΩΝΣΤΑΝΤΙΝΟΥ

Αριθμός Φοιτητικής Ταυτότητας: 25327

Επιβλέπων Καθηγητής:

Δρ. Ανδρέου Ανδρέας

Λεμεσός, 05-2026

Πνευματικά δικαιώματα

Copyright © Χρίστου Κωνσταντίνος 2026

Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Η έγκριση της πτυχιακής εργασίας από το Τμήμα Ηλ. Μηχανικών & Μηχανικών Ηλ. Υπολογιστών & Πληροφορικής του Τεχνολογικού Πανεπιστημίου Κύπρου δεν υποδηλώνει κατ' ανάγκη και αποδοχή των απόψεων του/της συγγραφέα.

Περίληψη

Η παρούσα πτυχιακή εργασία παρουσιάζει την ανάπτυξη ενός εκπαιδευτικού παιχνιδιού 2D (serious game) με ενσωμάτωση οντολογικής μοντελοποίησης γνώσης. Το κεντρικό ζητούμενο είναι η δομημένη και επεκτάσιμη διαχείριση του περιεχομένου του παιχνιδιού (NPCs, εργασίες, διάλογοι, ανταμοιβές) μέσω τεχνολογιών OWL/RDF/Turtle, αποφεύγοντας την αποκλειστική χρήση hardcoded λογικής.

Η αρχιτεκτονική βασίζεται σε τρία επίπεδα: το Ontology Layer που ορίζει τις έννοιες και σχέσεις, το Unity Layer που υλοποιεί τη game logic σε C#, και το Content Mapping Layer που γεφυρώνει οντολογία και Unity μέσω URI-based TaskData. Η οντολογία σχεδιάστηκε με το εργαλείο Protégé και αποθηκεύτηκε σε δύο αρχεία Turtle — ένα για το σχήμα και ένα για τα instances.

Το σύστημα υποστηρίζει τρεις τύπους εργασιών (Word-to-Image, Image-to-Text, Image-to-Phrase), σύστημα προόδου με κανόνες σειράς NPCs και proximity constraints, narrator guidance, score-based rewards (Bronze/Silver/Gold/Diamond) και ηχητικό feedback. Τα αποτελέσματα επιβεβαιώνουν ότι η οντολογική μοντελοποίηση μπορεί να ενσωματωθεί πρακτικά σε σύγχρονη game engine, προσφέροντας επεκτάσιμη και συντηρήσιμη βάση για εκπαιδευτικές εφαρμογές.

Λέξεις κλειδιά: serious game, οντολογία, OWL/RDF, Unity, εκπαιδευτικό παιχνίδι, semantic web, gamification

Abstract

This thesis presents the development of a 2D educational game (serious game) incorporating ontological knowledge modelling. The central challenge addressed is the structured and extensible management of game content (NPCs, tasks, dialogues, rewards) through OWL/RDF/Turtle technologies, avoiding exclusive reliance on hardcoded logic.

The architecture is based on three layers: the Ontology Layer defining concepts and relations, the Unity Layer implementing game logic in C#, and the Content Mapping Layer bridging ontology and Unity via URI-based TaskData. The ontology was designed using Protégé and stored in two Turtle files — one for the schema and one for instances.

The system supports three task types (Word-to-Image, Image-to-Text, Image-to-Phrase), a progression system with NPC ordering rules and proximity constraints, narrator guidance, score-based rewards (Bronze/Silver/Gold/Diamond) and audio feedback. Results confirm that ontological modelling can be practically integrated into a modern game engine, providing an extensible and maintainable foundation for educational applications.

Keywords: serious game, ontology, OWL/RDF, Unity, educational game, semantic web, gamification

Πίνακας Περιεχομένων

Contents

Περίληψη	iii
Abstract	iv
Πίνακας Περιεχομένων	v
Πίνακας διαγραμμάτων	ix
Κατάλογος Όρων	x
Κατάλογος Συντομογραφιών	xi
1 Κεφάλαιο 1: Εισαγωγή	1
1.1 Το Πρόβλημα και η Σημασία του	1
1.2 Στόχοι της Πτυχιακής	2
Προσέγγιση και Βασικά Αποτελέσματα	3
1.3 Πεδίο Εφαρμογής – Candy City	3
1.4 Οργάνωση του Εγγράφου	4
2 Κεφάλαιο 2: Βιβλιογραφική Ανασκόπηση	5
2.1 Εισαγωγή	5
2.2 Serious Games και Gamification	5
2.3 Κοινωνική Μάθηση και Αλληλεπίδραση	5
2.4 Εξατομίκευση και Προσαρμοστικά Συστήματα Μάθησης	6
2.5 Data Lakes και Σημασιολογικός Εμπλουτισμός	6
2.6 Κριτική Σύγκριση και Ερευνητικό Κενό	6
2.7 Σύνοψη	7
3 Κεφάλαιο 3: Τεχνολογικό Υπόβαθρο	8
3.1 Θεωρητικό Υπόβαθρο	8
3.1.1 Εμβάθυνση στο RDF και την αναπαράσταση σεναρίου	13
3.1.2 Η πρόκληση του «flat» μοντέλου και η ιεραρχική λύση	13

3.1.3	Σύγκριση με υπάρχουσα βιβλιογραφία (extend / continuation)	13
3.1.4	Pipeline OWL → Unity — θεωρητικά στάδια	14
3.2	Πρακτικό Υπόβαθρο	17
3.2.1	Unity 2D και C#.....	17
3.2.2	Οντολογίες: OWL, RDF και Turtle	18
3.2.3	Protégé	19
3.2.4	JSON για Runtime Δεδομένα	19
3.2.5	Αρχιτεκτονική Συστήματος	20
3.2.6	Task Rendering ανά Modality	21
3.2.7	Ήχος και Εμπειρία Χρήστη	21
4	Κεφάλαιο 4: Μεθοδολογία και Ανάπτυξη.....	22
4.1	Μεθολογική Προσέγγιση.....	23
	Πίνακας λειτουργικών και μη λειτουργικών απαιτήσεων	25
4.2	Σχεδίαση Αρχιτεκτονικής	26
4.2.1	Πίνακας αντιστοίχισης Unity component με Αρχείο.....	26
4.3	Σχεδίαση και Συντήρηση Οντολογίας	27
4.3.1	Αρχεία οντολογίας: core.ttl και data.ttl.....	27
4.3.2	Ορισμός κλάσεων και «boxes».....	27
4.3.3	Object properties → Domain και Range.....	42
4.3.4	Datatype properties και τιμές policies	46
4.3.5	Περιορισμοί OWL (restrictions) -- παραδείγματα από την οντολογία... 47	
4.3.6	Individuals στο data.ttl.....	49
4.3.7	Reasoner και έλεγχος συνέπειας.....	50
4.3.8	Export Turtle και τοποθέτηση σε StreamingAssets.....	52
4.3.9	Policies – ερμηνεία τιμών στο runtime.....	53
4.3.10	Τα τρία NPCs και σειρά περιεχομένου.....	54

4.4	Ενσωμάτωση Οντολογίας στο Unity	54
4.4.1	Ρυθμισή Unity: OntologyLoader και TaskManager στο Scene.....	55
4.4.2	Parse και MapTriplesToDomainObjects	57
4.4.3	TaskManager – runtime queries	58
4.4.4	Σύνδεση URI: InteractiveNPC και TaskData	60
4.4.5	TaskData: Δομή και σχέση με την οντολογία.....	61
4.4.6	Δημιουργία σκηνής (Περιβάλλον).....	63
4.4.7	Πρόσθεση GameObjects, Panels, Text, Images και components στο Unity (Πρακτικό).....	65
4.5	Σύστημα Εισόδου και Αποθήκευση Προφίλ	71
4.5.1	Login – πλήρης ανάλυση και κώδικας.....	71
4.5.2	PlayerProfileManager και JSON	76
4.6	Κίνηση Παίκτη (PlayerMovement)	77
4.7	NPC, Πρόοδος και Διάλογοι.....	78
4.7.2	Διαλόγοι NPC (DialogueManager, NpcDialogue)	80
4.8	Tasks, Task UI και Hints	83
4.8.1	Επιλογή panel ανα modality	87
4.8.2	Hint System.....	88
4.9	Sequence, Rewards και Performance Tracking	89
4.9.1	Sequence και σύνδεση Dialog & Reward.....	89
4.9.2	Rewards – Υπολογισμός και εμφάνιση	90
4.9.3	PerformanceTracker.....	91
4.10	Σύστημα Narrator	93
4.10.1	Narrator -- milestones	93
4.11	Audio System.....	94
5	Κεφάλαιο 5: Επίδειξη Συστήματος.....	96

5.1	Σκοπός Κεφαλαίου	96
5.2	Σενάριο Επίδειξης.....	96
5.3	Παρουσίαση Οθονών	97
5.4	Μετρικές Αποτίμησης.....	103
5.5	Συζήτηση Αποτελεσμάτων	103
6	Κεφάλαιο 6: Συμπεράσματα και Μελλοντική Εργασία	104
6.1	Συμπεράσματα	104
6.2	Επίτευξη Στόχων.....	104
6.3	Περιορισμοί	104
6.4	Μελλοντική Εργασία	105
6.5	Τελική Αποτίμηση	105
7	Βιβλιογραφία	106

Πίνακας διαγραμμάτων

Διάγραμμα 1 – Παράδειγμα απο την οντολογία του έργου	9
Διάγραμμα 2 - Μεθοδολογική ροή	22
Διάγραμμα 3 - ttl files	27
Διάγραμμα 4 – Δομή κλάσεων.....	28
Διάγραμμα 5 – Hierarchy in Protege	28
Διάγραμμα 6 - Analytical Models Box.....	30
Διάγραμμα 7 - Performance summary Demo	30
Διάγραμμα 7 - Performance summary Demo	30

Κατάλογος Όρων

Ο πίνακας συμπληρώνει τις συντομογραφίες και ορίζει κάθε βασική έννοια του έργου με ελληνική απόδοση και ρόλο στο σύστημα.

Serious Game: Ψηφιακό παιχνίδι με εκπαιδευτικό στόχο. στο project: Candy City scene με NPCs και tasks.

Οντολογία: Τυπική αναπαράσταση γνώσης (κλάσεις, properties, individuals) σε OWL/RDF/Turtle.

URI: Μοναδικό κλειδί. πρέπει να ταιριάζει μεταξύ .ttl, InteractiveNPC και TaskData.

NPCArchetype: Οντολογικό αρχέτυπο NPC. Unity: npcArchetypeUri.

TaskTemplate: Πρότυπο εργασίας με modality, initialDifficultyProfile.

Modality: Word-to-Image, Image-to-Text, Image-to-Phrase — καθορίζει UI panel.

DifficultyProfile: Beginner (1), Intermediate (2), Advanced (3) — επίπεδο παίκτη.

DifficultyPolicy: Κανόνας προσαρμογής μετά από sequence (accuracy + hints → delta).

TaskData: ScriptableObject: taskUri → sprites, phrases, hints.

Progression Order: Σειρά NPCs στο GameManager.

PerformanceTracker: Καταγραφή accuracy, score, hints, latency ανά task.

Reward Tier: Bronze/Silver/Gold/Diamond από μέσο score 0-100.

Narrator marker: narrator://seen/... στο CompletedTaskUris — one-time μηνύματα.

StreamingAssets: Φάκελος runtime για .ttl αρχεία.

PBKDF2: Αλγόριθμος hash κωδικού στο PasswordHelper.

Κατάλογος Συντομογραφιών

NPC	Non-Player Character (εικονικός χαρακτήρας)
OWL	Web Ontology Language
RDF	Resource Description Framework
URI	Uniform Resource Identifier
UI	User Interface (διεπαφή χρήστη)
SFX	Sound Effects (ηχητικά εφέ)
JSON	JavaScript Object Notation
2D	Δισδιάστατος (two-dimensional)

1 Κεφάλαιο 1: Εισαγωγή

1.1 Το Πρόβλημα και η Σημασία του

Η παρούσα πτυχιακή εργασία ασχολείται με την ανάπτυξη ενός εκπαιδευτικού παιχνιδιού 2D (serious game), στο οποίο η οργάνωση του περιεχομένου δεν βασίζεται αποκλειστικά σε hardcoded λογική, αλλά συνδυάζεται με οντολογική μοντελοποίηση γνώσης. Η ανάπτυξη εκπαιδευτικών εφαρμογών με δυνατότητες προσαρμογής και επεκτασιμότητας αποτελεί ένα από τα πιο ενεργά ερευνητικά πεδία στη διασταύρωση τεχνολογίας και παιδαγωγικής επιστήμης.

Το βασικό πρόβλημα που αντιμετωπίζουν οι σύγχρονες εκπαιδευτικές εφαρμογές παιχνιδιού είναι η ανάγκη για δομημένη, επεκτάσιμη και συντηρήσιμη διαχείριση του περιεχομένου τους. Στοιχεία όπως NPCs, εργασίες, επίπεδα δυσκολίας, διάλογοι, υποδείξεις (hints) και ανταμοιβές (rewards) συχνά ορίζονται με σκληρά κωδικοποιημένη λογική, που καθιστά δύσκολη τη συντήρηση και επέκταση. Αυτή η προσέγγιση, παρότι επιτρέπει γρήγορη αρχική υλοποίηση, δημιουργεί σημαντικά προβλήματα στη μεσομακροπρόθεσμη ανάπτυξη: κάθε αλλαγή περιεχομένου απαιτεί τροποποίηση πολλαπλών scripts, αυξάνει τον κίνδυνο ασυνεπειών (π.χ. λανθασμένα ή διπλά URIs) και δυσκολεύει την επέκταση σε νέους χαρακτήρες, νέους τύπους ασκήσεων ή νέες σκηνές.

Η χρήση οντολογιών (OWL/RDF) για την αναπαράσταση αυτής της γνώσης προσφέρει μια δομημένη εναλλακτική: επιτρέπει τη σαφή αναπαράσταση εννοιών και σχέσεων, την επαλήθευση συνέπειας με reasoner κατά το σχεδιασμό και τη σύνδεση με runtime συστήματα μέσω σταθερών αναγνωριστικών (URIs). Στο πλαίσιο της εργασίας, η οντολογία λειτουργεί ως «πηγή αλήθειας» για τη δομή του παιχνιδιού (ποιος NPC έχει ποιες εργασίες, ποια modality, ποιες πολιτικές προσαρμογής), ενώ το οπτικό και λεκτικό περιεχόμενο (εικόνες, φράσεις, hints) διαχωρίζεται και αντιστοιχίζεται μέσω TaskData assets στο Unity

Η σημασία του θέματος είναι υψηλή, καθώς συνδυάζει δύο ευρύτερα επιστημονικά πεδία:

- Παιδαγωγικό σχεδιασμό μέσω serious games: χρήση μηχανισμών παιχνιδιού για την ενίσχυση της μάθησης, της συμμετοχής και του κινήτρου των χρηστών.

- Τεχνολογική οργάνωση γνώσης μέσω οντολογιών και semantic δεδομένων: αξιοποίηση τυποποιημένων τεχνολογιών (OWL, RDF, Turtle) για τη μοντελοποίηση και διαχείριση του περιεχομένου.

Η ενοποίηση αυτών των δύο πεδίων σε ένα λειτουργικό πρωτότυπο αποτελεί το κεντρικό ζητούμενο της εργασίας.

1.2 Στόχοι της Πτυχιακής

Οι κύριοι στόχοι που τέθηκαν για την εκπόνηση της παρούσας εργασίας είναι:

1. Σχεδίαση οντολογίας (SGOntology) για την αναπαράσταση των βασικών εννοιών του παιχνιδιού: NPCArchetype, TaskTemplate, Modality, DifficultyProfile, DifficultyPolicy, RewardScheme και σχετικές σχέσεις (hasTaskTemplate, usesModality, initialDifficultyProfile κ.ά.).
2. Σύνδεση της οντολογίας με το Unity runtime μέσω αντιστοίχισης URIs και mapping σε C# domain objects (OntologyLoader, TaskManager), ώστε η δομή να προέρχεται από τα .ttl αρχεία και όχι από hardcoded λίστες
3. Υλοποίηση πολλαπλών τύπων εργασιών (Word-to-Image, Image-to-Text, Image-to-Phrase) με δυναμική επιλογή UI panel ανά modality
4. Ανάπτυξη μηχανισμού προόδου με καθορισμένη σειρά NPCs, proximity constraints, intro/outro dialogues και one-time μηνύματα.
5. Καταγραφή επίδοσης και προσαρμογή δυσκολίας μετά από ολοκληρωμένη ακολουθία tasks (accuracy, hints, latency → εφαρμογή DifficultyPolicy)
6. Ενσωμάτωση συστημάτων διαλόγων, narrator, rewards και ηχητικού feedback για καθοδήγηση, κίνητρο και αίσθηση ολοκληρωμένου παιχνιδιού
7. Δημιουργία λειτουργικού πρωτοτύπου με δυνατότητα επέκτασης (νέοι NPC, tasks, σκηνές) χωρίς ανασχεδιασμό ολόκληρης της λογικής progression.

Προσέγγιση και Βασικά Αποτελέσματα

Η προσέγγιση στηρίχθηκε σε σαφή διαχωρισμό τριών επιπέδων:

- **Ontology Layer:** ορίζει τι υπάρχει στο παιχνίδι και πώς τα στοιχεία συνδέονται (NPCs, tasks, modalities, difficulty, policies, rewards). Το schema οργανώνεται σε ομαδοποιητικές κλάσεις («boxes») εμπνευσμένες από το πλαίσιο των Loizou και Andreou (2025)
- **Unity Layer:** υλοποιεί τη συμπεριφορά του παιχνιδιού κατά το runtime μέσω C# scripts και managers (GameManager, TaskManager, TaskUI, NPCInteractionHandler, PerformanceTracker, RewardManager κ.ά.).
- **Content Mapping Layer (TaskData):** παρέχει το οπτικό και λεκτικό περιεχόμενο ανά task URI, γεφυρώνοντας οντολογία και Unity.

Η επικοινωνία μεταξύ επιπέδων γίνεται αποκλειστικά με **URIs**: κάθε NPC, task και modality έχει μοναδικό αναγνωριστικό που πρέπει να ταιριάζει μεταξύ .ttl, Inspector (InteractiveNPC) και TaskData entries. Αυτός ο κανόνας μειώνει coupling και επιτρέπει επαλήθευση πριν το runtime.

Τα βασικά αποτελέσματα περιλαμβάνουν: επιτυχή οντολογική μοντελοποίηση με αντιστοίχιση URIs, λειτουργία τριών task modalities με ξεχωριστά UI Panels, progression flow με κανόνες σειράς και constraints, narrator guidance σε milestones, score-based rewards και audio feedback (μουσική σκηνής, SFX γεγονότων) και αποθήκευση προόδου του παίκτη σε JSON αρχείο, χωρίς να μπερδεύονται runtime δεδομένα με το ontology schema.

1.3 Πεδίο Εφαρμογής – Candy City

Το πρωτότυπο εστιάζει στη σκηνή **Candy City** (SugarScene) όπου ο παίκτης ξεκινά από Main Menu (login/δημιουργία προφίλ), εισέρχεται στη σκηνή και αλληλεπιδά με NPCs (π.χ. Candy Fox, Vending Machine, Candy Princess) με καθορισμένη σειρά. Κάθε NPC προσφέρει tasks που φιλτράρονται με το τρέχον DifficultyProfile του παίκτη (Beginner / Intermediate / Advanced). Ο narrator παρέχει καθοδήγηση σε milestones (welcome message ή μετά από σημαντικά σημεία progression).

Η σκηνή λειτουργεί ως **proof-of-concept**: δεν υλοποιείται data lake και digital twin, αλλά το schema περιλαμβάνει έννοιες (DataLake και DigitalTwins) για μελλοντική επέκταση

1.4 Οργάνωση του Εγγράφου

Το υπόλοιπο έγγραφο οργανώνεται ως εξής:

- Κεφάλαιο 2: Παρουσιάζεται η βιβλιογραφική ανασκόπηση, με εστίαση σε serious games, gamification, προσαρμοστικά συστήματα μάθησης και semantic enrichment.
- Κεφάλαιο 3: Περιγράφεται το θεωρητικό και πρακτικό υπόβαθρο — Unity, OWL/RDF/Turtle, Protégé, JSON και η αρχιτεκτονική του συστήματος.
- Κεφάλαιο 4: Αναλύεται η μεθοδολογία ανάπτυξης από τις απαιτήσεις έως την υλοποίηση.
- Κεφάλαιο 5: Παρουσιάζεται επίδειξη του συστήματος με σενάριο χρήσης και μετρικές αποτίμησης.
- Κεφάλαιο 6: Παρατίθενται τα συμπεράσματα, αξιολογείται η επίτευξη των στόχων και προτείνεται μελλοντική εργασία.

2 Κεφάλαιο 2: Βιβλιογραφική Ανασκόπηση

2.1 Εισαγωγή

Η βιβλιογραφική ανασκόπηση εξετάζει εργασίες που σχετίζονται με τους κύριους πυλώνες της παρούσας εργασίας: serious games και gamification, κοινωνική μάθηση και αλληλεπίδραση, εξατομίκευση και προσαρμοστικά συστήματα, καθώς και data lakes και σημασιολογικό εμπλουτισμό δεδομένων. Σκοπός είναι η θεωρητική τεκμηρίωση της επιλεγμένης προσέγγισης, ο εντοπισμός εφαρμόσιμων πρακτικών και η διατύπωση του ερευνητικού κενού.

2.2 Serious Games και Gamification

Τα serious games αποτελούν ψηφιακές εφαρμογές που συνδυάζουν μηχανισμούς παιχνιδιού με εκπαιδευτικούς ή επαγγελματικούς στόχους. Οι Bonnechère et al. (2021) σε εκτενή μετα-ανασκόπηση καταλήγουν ότι τα serious games ενισχύουν τη συμμετοχή, το κίνητρο και, υπό κατάλληλες συνθήκες σχεδιασμού, τη μαθησιακή απόδοση.

Οι Lamb et al. (2018) σε συστηματική ανασκόπηση για τη χρήση serious games στις φυσικές επιστήμες τονίζουν ότι η αποτελεσματικότητά τους εξαρτάται σε μεγάλο βαθμό από την ευθυγράμμιση μεταξύ παιδαγωγικών στόχων και game mechanics. Η εργασία Challenge-based Learning (Nichols et al., 2020) αναδεικνύει ότι οι μηχανισμοί πρόκλησης, στόχου και ανατροφοδότησης υποστηρίζουν ενεργή μάθηση — εύρημα που αξιοποιήθηκε στον σχεδιασμό του progression flow.

2.3 Κοινωνική Μάθηση και Αλληλεπίδραση

Η αλληλεπίδραση μεταξύ χρήστη και εικονικών χαρακτήρων αποτελεί κρίσιμο στοιχείο εκπαιδευτικών παιχνιδιών. Οι Taib et al. (2019) τονίζουν ότι η σχεδίαση αλληλεπίδρασης επηρεάζει ουσιαστικά την εμπλοκή των χρηστών, ενώ μηχανισμοί proximity-based interaction και σειριακής προόδου ενισχύουν την αφηγηματική συνοχή.

Οι Makransky & Lilleholt (2018) εξετάζουν τον ρόλο αλληλεπίδρασης και αφηγηματικής δομής σε serious games εικονικής πραγματικότητας, δείχνοντας ότι η

δομημένη αφήγηση επηρεάζει ουσιαστικά την εκπαιδευτική εμπειρία — εύρημα που υποστηρίζει την επιλογή narrator guidance system.

2.4 Εξατομίκευση και Προσαρμοστικά Συστήματα Μάθησης

Οι Kickmeier-Rust & Albert (2010) υποστηρίζουν ότι η προσαρμογή περιεχομένου βάσει προφίλ χρήστη και επίδοσης αυξάνει αποτελεσματικότητα και engagement. Η εργασία των Spruit & Jagesar (2016) για ενσωμάτωση data lakes με self-adaptive serious games θέτει τις βάσεις για αρχιτεκτονικές που χρησιμοποιούν εξωτερικά αποθετήρια ως πηγή γνώσης — προσέγγιση που συγγενεύει άμεσα με τη χρήση οντολογίας ως εξωτερικής πηγής στην παρούσα εργασία.

2.5 Data Lakes και Σημασιολογικός Εμπλουτισμός

Οι Hai et al. (2016) αναδεικνύουν τη σημασία της οντολογικής οργάνωσης για καλύτερη αναζήτηση, κατανόηση και αναλυτική αξιοποίηση δεδομένων. Οι Sawadogo & Darmont (2021) εξετάζουν βελτίωση αλληλεπίδρασης με data lakes μέσω ψηφιακών διδύμων, ενώ οι Diamantini et al. (2019) αναλύουν τεχνικές semantic enrichment για data lake metadata. Και οι δύο εργασίες επιβεβαιώνουν ότι η σημασιολογική οργάνωση αποτελεί κρίσιμη υποδομή για επεκτάσιμα συστήματα.

2.6 Κριτική Σύγκριση και Ερευνητικό Κενό

Η βιβλιογραφία αποκαλύπτει ισχυρή θεωρητική βάση σε κάθε επιμέρους πεδίο. Ωστόσο, εντοπίζεται ένα σαφές ερευνητικό κενό: η έλλειψη πρακτικής ενοποίησης όλων των συνιστωσών (οντολογία + game runtime + adaptation + rewards) σε ένα ενιαίο λειτουργικό πρωτότυπο αναπτυγμένο σε εμπορική game engine. Οι υπάρχουσες εργασίες τείνουν να εστιάζουν είτε στη θεωρητική μοντελοποίηση είτε στην game development χωρίς semantic υποδομή. Η παρούσα εργασία στοχεύει ακριβώς σε αυτή την ενοποίηση.

2.7 Σύνοψη

Το θεωρητικό υπόβαθρο επιβεβαιώνει ότι η επιλεγμένη προσέγγιση είναι επιστημονικά τεκμηριωμένη και τεχνολογικά ώριμη για υλοποίηση. Η βιβλιογραφία υποστηρίζει τόσο τη χρήση serious games για εκπαιδευτικούς σκοπούς όσο και την αξιοποίηση semantic τεχνολογιών για την οργάνωση και προσαρμογή του περιεχομένου.

3 Κεφάλαιο 3: Τεχνολογικό Υπόβαθρο

3.1 Θεωρητικό Υπόβαθρο

Η παρούσα εργασία βασίζεται στο Semantic Web stack: RDF για αναπαράσταση γεγονότων, OWL για δομημένες κλάσεις και περιορισμούς, Turtle (.ttl) για αποθήκευση, και Protégé για σχεδίαση/επαλήθευση. Το κεντρικό επιχείρημα είναι ότι η «λογική του παιχνιδιού» μπορεί να εκφραστεί ως γράφημα γνώσης (knowledge graph) και να συνδεθεί με runtime engine (Unity) μέσω σταθερών URIs.

Στο RDF, κάθε δήλωση είναι τριάδα (Υποκείμενο, Κατηγορημα, Αντικείμενο). Για παράδειγμα, το ότι ο NPC_Candy_Fox έχει μια εργασία Beginner και χρησιμοποιεί modality Word-to-Image μοντελοποιείται ως:

```
:NPC_Candy_Fox rdf:type sgo:NPCArchetype .  
:NPC_Candy_Fox sgo:hasTaskTemplate :CandyFox_Task1_Beginner .  
:CandyFox_Task1_Beginner rdf:type sgo:TaskTemplate .  
:CandyFox_Task1_Beginner sgo:usesModality :Modality_WordToImage .  
:CandyFox_Task1_Beginner sgo:initialDifficultyProfile :Difficulty_Beginner .
```

Παράδειγμα RDF/Turtle από την οντολογία του έργου

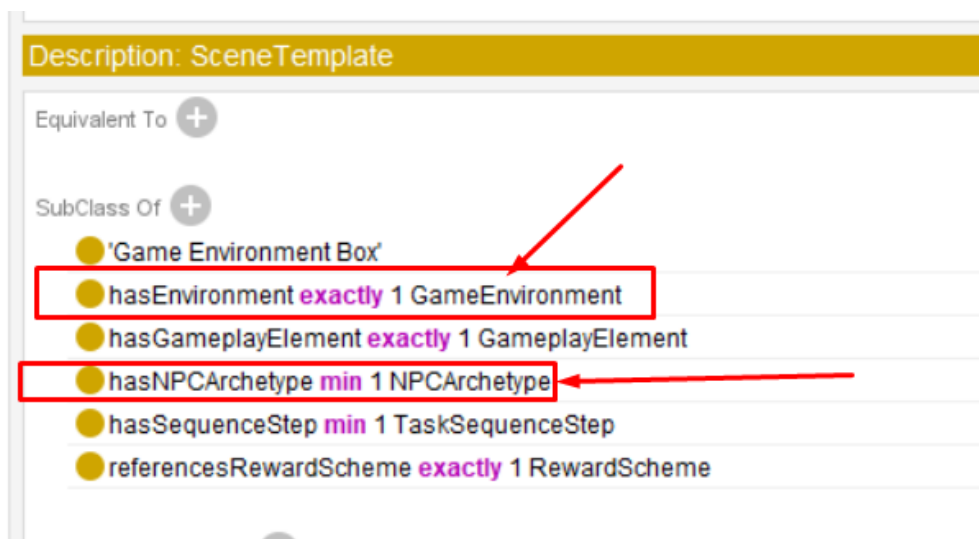
Η «λογική σεναρίου» (scenario logic) δεν είναι if-else στον κώδικα, αλλά σχέσεις: Game → SceneTemplate → NPCArchetype → TaskTemplate → Modality / DifficultyProfile / DifficultyPolicy. Έτσι ορίζεται τι επιτρέπεται να συμβεί. το Unity υλοποιεί πώς εμφανίζεται (UI, κίνηση, ήχος).

Η OWL επεκτείνει το RDF με rdfs:subClassOf (ιεραρχία), domain/range σε properties, και (προαιρετικά) restrictions. Στο SGOntology_core_v2.ttl ορίστηκαν «κουτιά»

(boxes) που ομαδοποιούν έννοιες εμπνευσμένα από το έγγραφο «**A Framework for Standardizing the Development of Serious Games with Real-Time Self-Adaptation Capabilities Using Digital Twins**» του Σπύρου Λοίζου & Ανδρέα Σ. Ανδρέου:

- GameEnvironmentBox — SceneTemplate, NPCArchetype, TaskTemplate, Modality, InteractionMode.
- SelfAdaptiveSimulationsBox — DifficultyProfile, DifficultyPolicy, AdaptationDecision.
- AnalyticalModelsBox — PerformanceSummary, RewardScheme, RewardTier.
- UsersProfilingBox — Player, DataLake.

Πρόκληση σχεδίασης: στις πρώτες εκδόσεις το μοντέλο ήταν «flat» (λιγότερες υποκλάσεις, ασαφής διάκριση template vs instance). Αυτό δυσκόλευε τόσο την επέκταση όσο και στο να κρατηθεί μία σχετικά «αυστηρή» ροή στη δημιουργία/πρόσθεση καινούριων οντότητων στην οντολογία, όπως σκηνές, NPCs, Tasks, Rewards κτλ. Η λύση ήταν επανασχεδιασμός της οντολογίας: προσθήκη μίας πιο αυστηρής ιεραρχίας όπου θα έπρεπε να ακολουθούνται κάποιοι κανόνες πριν την προσθήκη/αφαίρεση οντοτήτων από την οντολογία (π.χ. Μια σκηνή πρέπει να έχει **τουλάχιστον** ένα NPCArchetype --- SceneTemplate → hasNPCArchetype **min** 1 NPCArchetype ή Μια σκηνή πρέπει να έχει **ακριβώς** ένα GameEnvironment --- SceneTemplate → hasEnvironment **exactly** 1 GameEnvironment), (Βλέπε Διάγραμμα 1), περισσότερων κλάσεων/υποκλάσεων, διαχωρισμός σε δύο διαφορετικά .ttl αρχεία. Ένα core.ttl (schema), όπου βρίσκεται μόνο η «βάση,core» της οντολογίας, και data.ttl (instances), όπου υπάρχουν οι οντότητες όπως NPC Candy Fox, Narrator Dialog, SceneTemplate_CandyCity, όπως και οι προτάσεις/λέξεις που χρησιμοποιούνται στα tasks. (π.χ A boy in a boat going down a chocolate river ή Julia and her brother are building a candy castle with marshmallow walls.)



Διάγραμμα 1 – Παράδειγμα από την οντολογία του έργου

Τι υπήρχε (βιβλιογραφία): Απο τη βιβλιογραφία δεν έλειπε η ιδέα οτι τα serious games μπορούν να εκπαιδευτούν, οτι η προσαρμογή και η σωστή χρήση οντοτήτων, δυσκολίας, χρωμάτων, ήχων κ.α βελτιώνουν το engagement ή οτι τα δεδομένα μπορούν να οργανωθούν και η γνώση μπορεί να μοντελοποιηθεί σαν μία οντολογία. Αυτό που έλειπε ήταν ενα συγκεκριμένο, ενιαίο μοντέλο του παιχνιδιού βασισμένο σε μία οντολογία όπου αναπαριστά τη δομή του παιχνιδιού.

Τι επεκτάθηκε (Extend: Απαντά σε «τι μοντέλο φτιάχτηκε πέρα απο τη βιβλιογραφία): Φτιάχτηκε μία domain-specific serious game ontology (Candy City, 3 NPCs, 3 modalities, 3 policies, 4 είδη rewards), η οποία περιγράφει **ακριβώς** τον κόσμο του serious game μου και όχι οποιουδήποτε παιχνιδιού, παρόλο που μπορεί να χρησιμοποιηθεί η «βάση» της οντολογίας για τη δημιουργία νέου παιχνιδιού μετά απο μικροαλλαγές, ακολουθώντας όμως την ιεραρχία της παρών οντολογίας. Η Οντολογία φτιάχτηκε όχι σαν ένα αφηρημένο διάγραμμα αλλά ως ένα συμβόλαιο για το τι επιτρέπεται να υπάρχει στο παιχνίδι και πώς συνδέεται.

Πώς συνεχίστηκε (Continuation: Απαντά σε «Πως αυτό το μοντέλο ζεί στο παιχνίδι):

Πίνακας 1: Extend and Continuation

Extend (Οντολογία)	Continuation (Unity/runtime)
NPCArchetype + tasks	OntologyLoader → TaskManager.GetTasksForNPCAtDifficulty
Modality	TaskUI.ShowTask + TaskData sprites/phrases
Policies	PerformanceTracker → DifficultyAdjuster → update profile JSON
Scene Candy City	NPCInteractionHandler.GetCurrentSceneUri
Progression σειράς NPC	GameManager.npcProgressionOrderUris

Η οντολογία είναι **single source of truth** για δομή και κανόνες (Δες Table 2). (Δηλαδή το σύστημα ξέρει τι ισχύει απο μόνο μία πηγή ωστε να μην υπάρχει η ίδια πληροφορία σε πολλά διαφορετικά μέρη που μπορεί να διαφέρουν.) Η continuation είναι Pipeline (φόρτωση TTL → Dictionaries → URI lookup → UI → Ηχος / Narrator / Rewards) και

δυναμικά δεδομένα (ποιος παίκτης, τι έχει ολοκληρώσει). Η σύνδεση γίνεται με URI. Αν ένα URI αλλάζει ή είναι διαφορετικό σε ένα μέρος, τότε δεν θα φορτώνονται σωστά ή καθόλου.

Πίνακας 2: Single Source of truth

Τι ορίζει	Source of truth
Δομή παιχνιδιού (ποιοι NPCs, ποια tasks, ποια modality, ποια difficulty, policies)	Ontology (.ttl)
Οπτικό/λεκτικό περιεχόμενο (εικόνες, φράσεις, hints)	TaskData (ScriptableObject)
Πρόοδος συγκεκριμένου παίκτη (τι ολοκλήρωσε, score, τρέχουσα δυσκολία)	JSON Profiles

Το **OntologyLoader** φορτώνει τα αρχεία `SGOntology_core_v2.ttl` (schema) και `SGOntology_data_v2.ttl` (instances) από τα `StreamingAssets` και τα μετατρέπει σε C# αντικείμενα που καταλαμβάνει το παιχνίδι. Διαβάζει τα .ttl αρχεία, τα μετατρέπει σε λίστες τριάδων (Υποκείμενο – κατηγορημα – αντικείμενο) και μέσω του `MapTriplesToDomainObjects()` γεμίζει dictionaries όπως `NPCArchetypes`, `TaskTemplates`, `DifficultyProfiles`, `Scene templates` κτλ.

Όταν ζητηθεί π.χ ένα NPC, κάνει lookup με URI: $\rightarrow$ `GetNPCArchetype(uri)`.

Η οντολογία είναι σε Turtle/RDF όπου το Unity δεν μπορεί να την «δει». Ο `OntologyLoader` είναι κάτι σαν «μεταφραστής». «Μεταφράζει» βασικά στο Unity τι υπάρχει και πώς συνδέεται.

Το `TaskData` (Asset στο Unity) είναι ένας πίνακας από entries, όπου κάθε entry έχει:

- `taskUri` $\rightarrow$ Ιδιο με το URI στην οντολογία
- `images` $\rightarrow$ Προσθήκη των φωτογραφιών που θα χρησιμοποιηθούν στα tasks.

correctedImageIndex → Χρησιμοποιείται σε tasks με modality Phrase-To-Image και Word-To-Image. Προσθήκη αριθμού 0-3 ή 0-4 (εξαρτάται απο το tasks) όπου δείχνει στη ορθή φωτογραφία

choicePhrases → Χρησιμοποιείται σε tasks με modality Image-To-Phrase. Προσθήκη τών φράσεων όπου ο παίκτης θα πρέπει να επιλέξει τη σωστή φράση για την ολοκλήρωση του task.

stimulusImage → Χρησιμοποιείται σε tasks με modality Image-To-Phrase. Προσθήκη της εικόνας που θα εμφανίζεται στον παίκτη, μαζί με τα choicePhrases.

Hints → Χρησιμοποιείται σε όλα τα tasks. Προσθήκη των hints που θα δίνονται στον παίκτη όταν ζητάει hint ή όταν επιλέξει λανθασμένα. Υπάρχουν διάφορων ειδών hints όπως (highlighting key words , Text hints όπου δίνεται μία πρόταση που μπορεί να βοηθήσει τον παίκτη στην απόφαση του , wrong image removal όπου αφαιρεί μή ορθές εικόνες που δεν έχουν επιλεγθεί , highlight the correct image). Γίνεται επιλογή είδους hint και προστίθενται με την σειρά που πρέπει να εμφανιστούν.

Τα JSON profiles αποθηκεύουν δυναμικά, ανα παίκτη δεδομένα που αλλάζουν κατα το παιχνίδι. (Οχι στην οντολογία)

Αρχείο: PlayerProfiles.json

Κάθε player profile κρατάει:

- Name, PasswordHash, Id
- DifficultyProfileUri → Τρέχουσα δυσκολία που ενημερώνεται απο policies μετά την ολοκλήρωση ενός sequence of tasks (Ολοκλήρωση ενός NPC).
- CompletedTaskUris → Ποιά tasks έχουν ολοκληρωθεί + One Time markers που εμφανίστηκαν (Narrator, Dialogues, Rewards).
- PerformanceHistory → Accuracy, score, hints, χρόνος που είχε ο παίκτης ανά task
- LastSceneName → Χρήση για να συνεχίσει ο παίκτης απο το ίδιο σημείο όταν ξανα κάνει login.

3.1.1 Εμβάθυνση στο RDF και την αναπαράσταση σεναρίου

Η επιλογή RDF/Turtle αντί για αποκλειστικά hardcoded C# structures επιτρέπει (α) επαναχρησιμοποίηση εργαλείων (Protégé, SPARQL), (β) σαφή τεκμηρίωση του domain model, (γ) μελλοντική σύνδεση με εξωτερικά συστήματα (LMS, analytics, digital twin) χωρίς rewrite του παιχνιδιού. Κάθε NPC, κάθε task και κάθε modality είναι πόρος με URI. Οι σχέσεις είναι properties. Αυτό σημαίνει ότι το τι συμβαίνει στο Candy City μπορεί να διαβαστεί ως γράφημα.

Στην πράξη, το σενάριο «ο παίκτης Beginner πηγαίνει στον Candy Fox και κάνει Word-to-Image task» εκφράζεται ως: (Player profile έχει Difficulty_Beginner) AND (NPC_Candy_Fox hasTaskTemplate CandyFox_TaskX_Beginner) AND (task usesModality Modality_WordToImage). Το Unity δεν «ξέρει» ότι πρόκειται για Beginner μέχρι να διαβάσει το DifficultyProfileUri από το JSON profile και να φιλτράρει tasks. Έτσι διαχωρίζονται: τι υπάρχει στον κόσμο (ontology) και τι έχει κάνει ο συγκεκριμένος παίκτης (JSON).

3.1.2 Η πρόκληση του «flat» μοντέλου και η ιεραρχική λύση

Στην αρχή οι κλάσεις Task και NPC ήταν επίπεδες· δεν υπήρχε σαφής ομαδοποίηση σε boxes (GameEnvironment, SelfAdaptive, Analytical, UsersProfiling). Αυτό δημιούργουσε τρία πρακτικά προβλήματα: Δυσκολία επέκτασης όταν προσθέτονται νέοι τύποι περιεχομένου. Αδύναμη οπτικοποίηση στο Protégé και σύγχυση μεταξύ «τύπου» (class) και «στιγμιότυπου» (individual). Η λύση ήταν εισαγωγή ιεραρχίας rdfls:subClassOf, διαχωρισμός core.ttl (λέξιλόγιο + κλάσεις + property definitions) και data.ttl (μόνο individuals και assertions), και σταθεροποίηση URIs με namespace <http://www.example.org/SGOntology/data#>.

3.1.3 Σύγκριση με υπάρχουσα βιβλιογραφία (extend / continuation)

Το extend από τη βιβλιογραφία (Κεφάλαιο 2) είναι συγκεκριμένο: αντί για γενικό adaptive serious game framework, υλοποιείται πλήρες domain model με τρεις modalities, τρεις NPCs, policies, rewards, narrator κτλπ. Το continuation είναι το Unity project όπου κάθε abstract έννοια έχει αντίστοιχο manager. Χωρίς continuation, η

οντολογία θα ήταν μόνο διάγραμμα, χωρίς extend, το παιχνίδι θα ήταν μόνο hardcoded levels.

3.1.4 Pipeline OWL → Unity — θεωρητικά στάδια

Στάδιο 1 → Σχεδίαση: Protégé, classes, individuals, consistency.

Η σχεδίαση της οντολογίας γίνεται στο εργαλείο «Protégé» όπου μας δίνει τη δυνατότητα να δημιουργήσουμε classes , individuals και properties για να δημιουργήσουμε μία ολοκληρωμένη οντολογία.

Κλάσεις (Classes): Τυποι πραγμάτων – π.χ «NPC» , «Modality» , «TaskTemplate»

Individuals (instances): συγκεκριμένα στοιχεία – π.χ «NPC_Candy_Fox» όχι απλα «ενα NPC»

Properties (ιδιότητες): σχέσεις – π.χ « NPC_Candy_Fox έχει task X», «To task X χρησιμοποιεί modality Image-to-Phrase».

Στάδιο 2 → Serialization: export .ttl.

Αποθήκευση του μοντέλου σε αρχεία Turtle (.ttl) όπου είναι απλο κείμενο με γραμμές τύπου:

:NPC_Candy_Fox rdf:type sgo:NPCArchetype .

:NPC_Candy_Fox sgo:hasTaskTemplate :CandyFox_Task1_Beginner .

Μπορούμε καθαρά να καταλάβουμε πως το NPC_Candy_Fox είναι τύπου NPCArchetype και έχει σαν Task Template το CandyFox_Task1_Beginner.

Το .ttl είναι ανάγνωστο δεν είναι ακριβώς κώδικας, αλλά δεδομένα γνώσης. Γίνεται η «επισημη» λίστα NPCs, tasks, modalities κτλπ.

Στάδιο 3 → Deployment: αρχεία στο StreamingAssets .

Τα .ttl αρχεία μπαίνουν στο φάκελο StreamingAssets του Unity Project, όπου το παιχνίδι τα διαβάζει κατα το runtime απο το δίσκο.

Γιατι Streaming Assets: Είναι ο standard τρόπος Unity για αρχεία που θέλουμε να διαβάσουμε ως αρχεία και όχι ως embedded assets με άλλο τύπο. Ο **OntologyLoader** ψάχνει εκεί τα SGOntology_core_v2.ttl και SGOntology_data_v2.ttl που αντιπροσωπεύουν την οντολογία.

Στάδιο 4 → Parse: OntologyLoader/TTLParser → λίστα Triple.

Όταν ξεκινά η σκηνή, ο OntologyLoader ανοίγει τα .ttl αρχεία και ο TTLParser τα σπάει σε τριάδες (triples) (Βλέπε Πίνακα 3). Κάθε γραμμή του turtle γίνεται μία τέτοια τριάδα στη μνήμη.

Πίνακας 3: Triples

Υποκείμενο	Κατηγορία	Αντικείμενο
NPC_Candy_Fox	hasTaskTemplate	CandyFox_Task1_Beginner
CandyFox_Task1_Beginner	usesModality	Modality_WordToImage

Το Unity δέν καταλαβαίνει OWL. Καταλαβαίνει λίστες αντικειμένων C#. Γι' αυτό το parse είναι το πρώτο βήμα μετάφρασης: Κειμενο → δομημένα δεδομένα.

Στάδιο 5 → Object mapping: MapTriplesToDomainObjects → dictionaries.

Τα triples ομαδοποιούνται σε αντικείμενα που χρησιμοποιεί το παιχνίδι: π.χ

- Dictionary<string, NPCArchetype> -- κλειδί = URI του NPC
- Dictionary<string, TaskTemplate> -- κλειδί = URI του task

Όλες οι τριάδες προστίθενται σε λίστες ετσι ωστε στο στάδιο 6 να μην ψάχνουμε ξανά σε υπερβολικά πολλές τριάδες αλλά να έχουμε έτοιμες λίστες για κάθε κατηγορία (π.χ tasks , policies, NPCs κ.α.)

Παραδειγμα λίστας: HasTaskTemplatesUris ή HasNPCArchetypeUris

Ετσι μπορούμε πολύ απλα να κάνουμε `GetNPCArchetype(uri)` και να το βρούμε απευθείας απο την έτοιμη λίστα που υπάρχει, χωρίς να ψάχνουμε να το βρούμε σε παρα πολλά triples.

Στάδιο 6 → Runtime query: `GetNPCArchetype`, `GetTaskTemplate`, `GetTasksForNPCAtDifficulty`.

Όταν ο παίκτης κάνει κάτι, π.χ κλικ σε ενα NPC, τα scripts ρωτούν την οντολογία μέσω των dictionaries:

1. `NPCInteractionHandler` παίρνει `npcArchetypeUri` απο το `InteractiveNPC` (ίδιο URI με το .ttl)
2. `TaskManager.GetTasksForNPCAtDifficulty(npcUri, playerDifficultyUri)`:
 - Βρίσκει το NPC → Λίστα task URIs,
 - Για κάθε task φορτώνει `TaskTemplate`,
 - Φιλτράρει όσα εχουν `initialDifficultyProfile = current player difficulty` απο το JSON profile του παίκτη.
 - Αγνοεί tasks τα οποία ειναι ήδη στο `CompletedTaskUris`
3. Επιλέγει το πρώτο διαθέσιμο task → `StartTask`

Το runtime query γίνεται λόγο του ότι δεν φορτώνουμε όλη την οντολογία στο UI. Απλά ρωτάμε εκείνη τη στιγμή «Τι πρέπει να δείξω τώρα». Κάτι σαν SQL query, αλλα πάνω στα dictionaries που φτιάχτηκαν στο στάδιο 5.

Γίνεται μια «συνδεση» του JSON με την οντολογία αλλα το κάθε ένα εχει τη δική του λειτουργία. Η οντολογία λεει «τι υπάρχει», π,χ τη λίστα με τα URIs, τα templates κτλπ. Ενώ το profile (JSON αρχείο) λέει τι εχει κάνει ο παιχτης (`CompletedTaskUris`) και σε ποιο difficulty βρίσκεται τώρα (`Current player difficulty`). Χρειάζονται και τα δύο για να δουλέψει.

Στάδιο 7 → Presentation: TaskUI + TaskData.

Το TaskUI:

- διαβάζει το modality του task απο task.UsesModalityUri και επιλέγει το σωστό panel που πρέπει να δείξει για το κάθε task (Word-to-Image / Image-to-Text / Image-to-Phrase),
- Καλεί την TaskData.TryGetEntry(task.Uri) → οπου βρίσκει και εμφανίζει τα σωστά sprites, φράσεις και hints για το κάθε task,
- Σχεδιάζει κουμπιά, hints, score και ήχο/sound effects.

Το στάδιο 7 είναι το μόνο που αγγίζει οπτικό υλικό· όλα τα προηγούμενα είναι δομικά.

Η οντολογία δεν ξέρει ποιες είναι οι σωστές εικόνες / sprites κ.α που πρέπει να εμφανιστούν. Αυτά φαίνονται στο TaskData.

3.2 Πρακτικό Υπόβαθρο

3.2.1 Unity 2D και C#

Η Unity είναι μια από τις πλέον διαδεδομένες game engines παγκοσμίως, με υποστήριξη τόσο για 2D όσο και για 3D ανάπτυξη. Χρησιμοποιεί C# ως κύρια γλώσσα προγραμματισμού και παρέχει πλούσιο οικοσύστημα εργαλείων όπως physics engine, UI framework και asset pipeline. Σε αντίθεση με 3D περιβάλλοντα, η 2D ανάπτυξη επιτρέπει μια πολύ πιο απλή οπτική οργάνωση με (sprites, 2D Colliders, orthographic camera), κάτι που ταιριάζει περισσότερο σε serious games οπου ο κύριος στόχος είναι η αλληλεπίδραση, το κείμενο, η εκπαίδευση και όχι τόσο η ρεαλιστική απεικόνιση.

Στην παρούσα εργασία το Unity αναλαμβάνει τον ρόλο του runtime layer, δηλαδή υλοποιεί τη συμπεριφορά του παιχνιδιού, (Πως κινείται ο παίκτης, NPC Clicks, εμφάνιση των task, διαλόγοι, narrator, rewards, sounds/sound effects), ενώ η δομή του περιεχομένου (Ποια NPCs υπάρχουν, ποιά tasks κτλπ.) ορίζονται απο την οντολογία. Έτσι δεν «κλειδώνουμε» τη γνώση του παιχνιδιού μέσα σε hardcoded λίστες αλλά χρησιμοποιούμε δεδομένα που προέρχονται απο εξωτερικές πηγές (Turtle αρχεία, TaskData, JSON Files).

Στην παρούσα εργασία αξιοποιήθηκαν:

- MonoBehaviour lifecycle (Awake, Start, Update): κάθε script που συνδέεται με κάποιο GameObject του παιχνιδιού ακολουθεί ένα κύκλο ζωής. Γίνεται η αρχικοποίηση του όταν φορτώνεται η σκηνή ή το αντικείμενο (Awake/Start) και η ενημέρωση (Update) ανά frame. Π.χ Φόρτωναται η οντολογία στην έναρξη της σκηνής και μετά γίνεται έλεγχος κλικ κάθε frame και ενεργοποίηση των UI Panels όταν χρειάζεται.
- UI system και TextMeshPro: Το σύστημα UI του Unity (Canvas, panels, buttons, images) χρησιμοποιείται για όλες τις οθόνες αλληλεπίδρασης του παιχνιδιού (login, task panels, dialogues, narrator, rewards). Το TextMeshPro προσφέρει πολύ καλή απόδοση και μορφοποίηση κειμένου και χρησιμοποιείται για (Hints, οδηγίες, επιλογή φράσης κ.α).
- ScriptableObjects: επιτρέπουν αποθήκευση παραμέτρων και περιεχομένου ως assets (TaskData) ανεξάρτητα από το lifecycle μιας σκηνής. Έτσι πολλά tasks μπορούν να μοιράζονται κοινή λογική UI, ενώ το οπτικο/λεκτικό υλικό αλλάζει ανα task μέσω URI Mapping.
- Event-driven αρχιτεκτονική: Η επικοινωνία μεταξύ υποσυστημάτων όπως η ολοκλήρωση κάποιου task sequence → Narrator ή rewards , γίνεται με C# events, έτσι ώστε τα components να μην εξαρτώνται άμεσα μεταξύ τους με αποτέλεσμα να κάνει πιο εύκολη τη συντήρηση και την επέκταση όταν προστεθούν νέα milestones στο παιχνίδι.

3.2.2 Οντολογίες: OWL, RDF και Turtle

Οι οντολογίες αποτελούν τυπικές αναπαραστάσεις ενός τομέα γνώσης, ορίζοντας έννοιες (classes), σχέσεις (properties) και αξιώματα (axioms). Στο πλαίσιο του Semantic Web, οι τεχνολογίες OWL (Web Ontology Language) και RDF (Resource Description Framework) παρέχουν τυποποιημένα πρότυπα για αναπαράσταση και ανταλλαγή γνώσης.

Το RDF χρησιμοποιεί τριάδες υποκειμένου-κατηγορήματος-αντικειμένου για να αναπαριστήσει γεγονότα (π.χ «το NPC_Candy_Fox έχει task το CandyFox_Task1_Beginner»). Η OWL επεκτείνει το RDF με πιο πλούσια εκφραστικότητα, δηλαδή προσθέτει πάνω απο αυτά κλάσεις, ιεραρχίες και περιορισμούς (π.χ TaskTemplate, NPCArchetype). Η σύνταξη Turtle (.ttl) αποτελεί συμπαγή και ευανάγνωστη μορφή αναπαράστασης RDF/OWL. Η turtle γίνεται έτσι

ώστε τα αρχεία να είναι αναγνώσιμα και για την αποθήκευση της οντολογίας.

Αναπτύχθηκαν δύο αρχεία όπου μας βοηθάνε να ξεχωρίζουμε «τι υπάρχει στο μοντέλο» από το «τι συγκεκριμένα υπάρχει στο scene του παιχνιδιού»:

- `SGOntology_core_v2.ttl`: περιέχει το σχήμα και λεξιλόγιο της οντολογίας όπως κλάσεις, properties και αξιώματα.
- `SGOntology_data_v2.ttl`: περιέχει τα instances της οντολογίας όπως συγκεκριμένα NPCs, tasks, συγκεκριμένες προτάσεις όπου χρησιμοποιούνται σε συγκεκριμένα tasks κ.α.

Στην παρούσα εργασία, αυτές οι τεχνολογίες χρησιμοποιήθηκαν ως ένας πρακτικός τρόπος να οργανωθεί η γνώση του παιχνιδιού, αντί να γράφονται όλα σε hardcoded λίστες μέσα στο Unity. Ορίζεται το τι υπάρχει μέσα στον κόσμο του παιχνιδιού, ποια NPCs, ποιά tasks, ποιά modalities, επίπεδα δυσκολίας και κανόνες που αλλάζουν τη δυσκολία κάποιου παίκτη.

3.2.3 Protégé

Το Protégé είναι ελεύθερο, ανοιχτού κώδικα περιβάλλον ανάπτυξης οντολογιών του Stanford University. Παρέχει γραφικό περιβάλλον για OWL/RDF, ενσωματωμένους reasoners (HermiT, Pellet) για έλεγχο συνέπειας και SPARQL queries. Το Protégé χρησιμοποιήθηκε για την δημιουργία της οντολογίας. Πρακτικά για τρεις βασικούς σκοπούς. (1) Σχεδίαση → Δημιουργία κλάσεων, properties και instances και η ένωση τους για το «χτίσιμο» της οντολογίας. (2) Έλεγχος → Για έλεγχο μέσω reasoner και SPARQL queries π.χ («ποια tasks έχει το NPC_Candy_Fox») έτσι ώστε να εντοπιστούν λάθη ή duplicates για να μην εμφανιστούν μετά στο παιχνίδι. (3) Export → Export σε αρχεία turtle (`SGOntology_core_v2.ttl`, `SGOntology_data_v2.ttl`) που τοποθετήθηκαν στο StreamingAssets για να τα φορτώσει ο `OntologyLoader`.

Το Protégé δεν είναι μέρος του runtime του παιχνιδιού. Χρησιμοποιήθηκε ως εργαλείο μοντελοποίησης πριν μεταφερθεί η «γνώση» στο Unity.

3.2.4 JSON για Runtime Δεδομένα

Το JSON (JavaScript Object Notation) χρησιμοποιείται για αποθήκευση των runtime δεδομένων του χρήστη, δηλαδή αποθηκεύει δυναμικά τα δεδομένα που αλλάζουν κατά τη διάρκεια παιχνιδιού όπως την πρόοδο του παίκτη, τα scores του και ποιές εργασίες

έχει ολοκληρώσει όπως tasks/narrator dialogues/NPC dialogues. Ενώ η οντολογία αναπαριστά τη δομική γνώση (τι υπάρχει και πώς συνδέεται). Αυτός ο διαχωρισμός εξασφαλίζει καθαρό separation of concerns και διευκολύνει μελλοντική ανάλυση δεδομένων.

3.2.5 Αρχιτεκτονική Συστήματος

Η αρχιτεκτονική ακολουθεί μια αρχή διαχωρισμού ευθυνών, με κάθε manager να είναι υπεύθυνος για ένα υποσύστημα:

- GameManager: κεντρικός συντονιστής της ροής παιχνιδιού.
- TaskManager: φορτώνει και διαχειρίζεται tasks βάσει URIs.
- TaskUI: χειρίζεται την απόδοση UI ανά task modality.
- DialogueManager: διαχειρίζεται ροή διαλόγων και follow-up μηνυμάτων.
- NPCInteractionHandler: ελέγχει αλληλεπίδραση με NPCs (proximity, order constraints).
- PerformanceTracker: καταγράφει scores, hints και χρόνο ανά task.
- RewardManager: υπολογίζει reward tier (Bronze/Silver/Gold/Diamond) και εκτελεί popup.
- NarratorGuideUI: εμφανίζει οδηγίες με milestones και one-time triggers.
- MusicManager / SfxManager: διαχειρίζονται μουσική υπόκρουση και ηχητικά εφέ.

Το σύστημα οργανώνεται σε 3 επίπεδα: (α) την Οντολογία όπου ορίζει τη δομή, (β) τους Managers στο Unity όπου εκτελούν τη λογική gameplay, (γ) το TaskData που «δίνει» το οπτικό και λεκτικό περιεχόμενο για κάθε task. Η επικοινωνία μεταξύ της οντολογίας και των managers γίνεται μέσω URIs και η σύνδεση μεταξύ των managers και του TaskData μέσω των taskUri που βρίσκονται στο TaskData. Έτσι αν γίνει κάποια αλλαγή στο περιεχόμενο (π.χ νέο Task ή νέο NPC), μπορεί να γίνει ενημέρωση της οντολογίας και του URI Mapping χωρίς να επηρεάζεται όλη η εφαρμογή.

Οι managers συνεργάζονται σε ροές. π.χ για ένα κλικ σε κάποιο NPC, περνάει από το NPCInteractionHandler → Στο DialogueManager → Στο TaskManager → γίνεται καταγραφή του performance του παίκτη → Στο RewardManager.

Ο GameManager είναι αυτός που «κρατά» το ενεργό profile και τους κανόνες του progression του παιχνιδιού (τη σειρά που πρέπει να επισκευτούν οι χρήστες τα NPCs).

Ο MusicManager και SFXManager είναι υπεύθυνοι για μουσική και sound effects του παιχνιδιού, χωρίς να επηρεάζουν τη λογική της δομής.

3.2.6 Task Rendering ανά Modality

Το σύστημα υποστηρίζει τρεις τύπους task modalities:

- Word-to-Image: ο χρήστης βλέπει μια λέξη και επιλέγει την αντίστοιχη εικόνα που την αντιπροσωπεύει από πολλαπλές επιλογές.
- Image-to-Text: ο χρήστης βλέπει μία εικόνα και επιλέγει τη σωστή λέξη/κείμενο που αντιπροσωπεύει καλύτερα την εικόνα.
- Image-to-Phrase: ο χρήστης βλέπει μια πρόταση/φράση και επιλέγει αντίστοιχη εικόνα που αντιπροσωπεύει καλύτερα την πρόταση από πολλαπλές επιλογές.

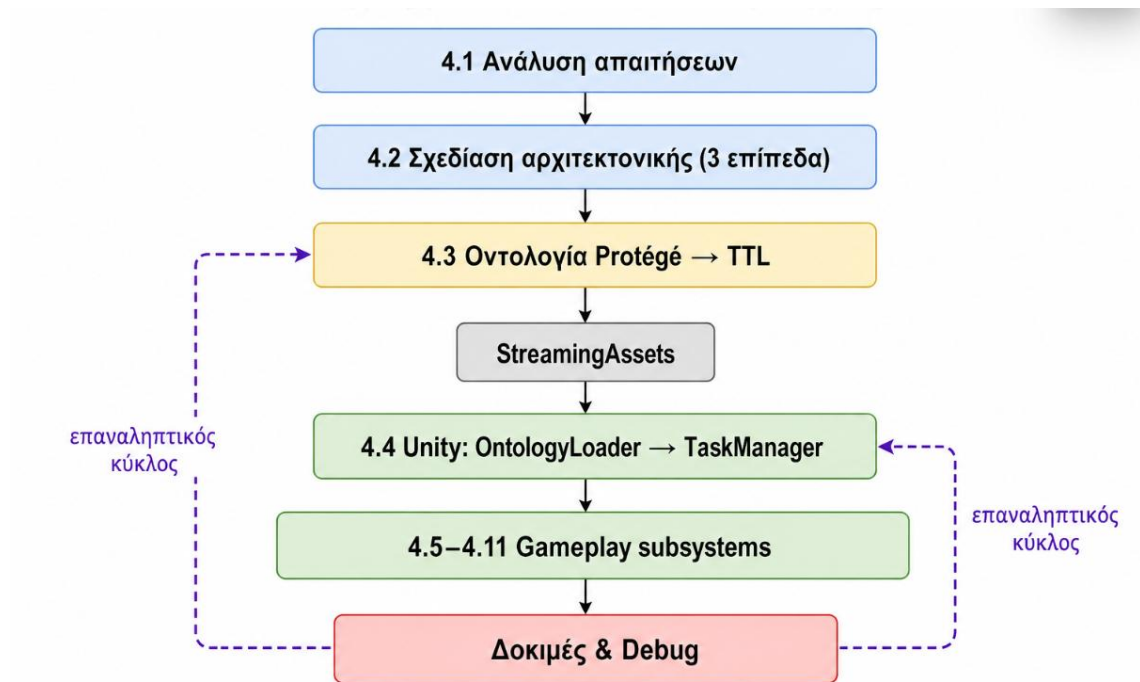
Το modality καθορίζεται από την οντολογία και το TaskManager δημιουργεί δυναμικά το κατάλληλο panel layout κατά το runtime.

3.2.7 Ήχος και Εμπειρία Χρήστη

Υλοποιήθηκαν διαφορετικά μουσικά tracks ανά σκηνή με fade μεταβάσεις και SFX για βασικά γεγονότα: NPC click, σωστή/λάθος απάντηση, reward popup. Ο διαχωρισμός MusicManager/SfxManager επιτρέπει ανεξάρτητο έλεγχο volume ανά κατηγορία ήχου.

Ο ήχος χρησιμοποιήθηκε ως μια άμεση ανατροφοδότηση και engagement για τον χρήστη και όχι ως διακοσμητικό στοιχείο. Τα sound effects (SFX) συνδέονται με συγκεκριμένα γεγονότα όπως κλικ σε NPC, Αν ο παίκτης απάντησε σωστά κατά τη διάρκεια των tasks του και όταν παίρνει κάποιο reward. Η μουσική χρησιμοποιήθηκε για την ενίσχυση του engagement του χρήστη ώστε να προσφέρει μια συνεχή ροή του παιχνιδιού μειώνοντας την αίσθηση αδράνειας ή αλληλεπίδρασης κατά τη διάρκεια του παιχνιδιού. Ένα παιχνίδι χωρίς μουσική και sound effects, μπορεί να προκαλέσει στον χρήστη απουσία ενδιαφέροντος και να μειώσει το κίνητρο για τη συνέχιση του παιχνιδιού.

4 Κεφάλαιο 4: Μεθοδολογία και Ανάπτυξη



Διάγραμμα 2 - Μεθοδολογική ροή

Το Διάγραμμα 4.2 συνοψίζει τη μεθοδολογική ροή της εργασίας. Η ανάπτυξη ξεκίνησε από την ανάλυση απαιτήσεων και τη σχεδίαση αρχιτεκτονικής τριών επιπέδων (οντολογία, Unity, TaskData). Στη συνέχεια σχεδιάστηκε η οντολογία στο Protégé, επαληθεύτηκε με reasoner/SPARQL και αποθηκεύτηκε σε αρχεία Turtle, τα οποία τοποθετήθηκαν στο StreamingAssets του Unity.

Στο επόμενο στάδιο, ο OntologyLoader μετατρέπει τα .ttl σε C# αντικείμενα και το TaskManager τα αξιοποιεί κατά το runtime μέσω URIs. Τα υποσυστήματα gameplay (login, NPC, tasks, rewards, narrator, ήχος) υλοποιήθηκαν πάνω σε αυτό το πλαίσιο, ενώ η πρόοδος παίκτη αποθηκεύεται σε JSON profiles.

Κάθε φάση επαληθεύτηκε με δοκιμές Play Mode. σε περίπτωση ασυνεπειών (URIs, UI, input) ακολουθήθηκε επαναληπτικός κύκλος διόρθωσης, όπως φαίνεται με τα διακεκομμένα βέλη. Τα επόμενα υποκεφάλαια (4.1–§.11) αναλύουν αναλυτικά κάθε βήμα της ροής.

4.1 Μεθοδολογική Προσέγγιση

Η ανάπτυξη δεν ακολούθησε μια γραμμική σειρά, δηλαδή «πρώτα όλη η οντολογία και μετά το παιχνίδι» Ακολουθήθηκε ένας επαναληπτικός κύκλος, όπου κάθε στάδιο επαναξιολογείται. Παρ'όλο που η βασική οντολογία έγινε πρώτη, κατα την δημιουργία του παιχνιδιού γίνονταν πολλές αλλαγές και updates στην οντολογία έτσι ώστε να συμφωνεί η οντολογία με το υπόλοιπο project. Π.χ Κατα την προσθήκη NPC, Tasks, φράσεων κ.α, γίνονταν αναδρομές πίσω στην οντολογία για να προστεθούν και να γίνουν match τα URI.

Βήματα:

1. Ανάλυση απαιτήσεων

Ορίστηκε τι πρέπει να κάνει το σύστημα: όπως (1) login και profiles , (2) τα 3 modalities (Word To Image, Image to Phrase και Image to Text), (3) η σειρά των NPCs που πρέπει να ακολουθήσει ο παίκτης για την ολοκλήρωση των task του και proximity checks μεταξύ NPC και παίκτη (ο παίκτης πρέπει να είναι κοντά στο npc για να μπορεί να κάνει interact μαζί του), (4) τι hints χρειάζομαστε στο σύστημα ανα task και πότε να δίνονται, (5) τι rewards χρειάζονται στο σύστημα, πότε να δίνονται και ποιά κατηγορία reward να παίρνει ο παίκτης ανάλογα με το performance του (Bronze,Silver,Gold,Diamond), (6) Narrator – Πότε να εμφανίζεται , μετα απο ποιά milestones, πού να εμφανίζεται για να δίνει οδηγίες στον παίκτη στο πώς να προχωρήσει στο παιχνίδι και (7) ηχος (μουσική, sound effects) για να «τραβάει» τον χρήστη στο παιχνίδι και να το κάνει πιο «διασκεδαστικό» αλλά παράλληλα εκπαιδευτικό.

Οι απαιτήσεις διαχωρίστηκαν σε λειτουργικές, δηλαδή τι βλέπει/κάνει ο παίκτης και σε μή λειτουργικές (επεκτασιμότητα, σταθερότητα, συντηρησιμότητα).

2. Σχεδίαση οντολογίας στο Protégé

Πρίν γραφτεί κώδικας gameplay, μοντελοποιήθηκε πίο γενικά το τι επιτρέπεται στον κόσμο του παιχνιδιού (Ποια NPCs υπάρχουν, ποια tasks, modalities, difficulties και policies) με αναδρομές πίσω στην οντολογία για τυχών αλλαγές/ενημερώσεις όσο προχωρούσε η υλοποίηση. Η οντολογία έγινε βασικά το «formal specification» και όχι ένα απλό διάγραμμα.

3. Export σε Turtle

Το μοντέλο έγινε export σε .ttl αρχεία (core = schema και data = instances) ώστε να είναι ανεξάρτητο απο το Unity και εύκολο στην επέκταση. Έτσι δεν χρειάζεται να γίνει rewrite του κώδικα, κάθε φορά που θέλουμε να προσθέσουμε κάτι.

4. OntologyLoader και domain models

Υλοποιήθηκε μια «γέφυρα» Unity ↔ RDF: όπου τα triples γίνονται C# objects (NPCArchetype, TaskTemplate, DifficultyPolicy κ.α). Χωρίς αυτο το βήμα τα .ttl θα ήταν απλά αρχεία. Με αυτό τον τρόπο γίνονται runtime δεδομένα.

5. Managers (Game, Task, Dialogue, NPC, Performance, Reward, Narrator)

Κάθε manager υλοποιεί ένα δικό του κομμάτι, διαβάζοντας από την οντολογία ο,τι αφορά τη δομή και κανόνες αλλά επίσης και απο το JSON profile, ο,τι αφορά την πρόοδο του παίκτη. Έτσι αποφεύγεται το να υπάρχει μία κλάση με όλη τη λογική μέσα (god-class), όπου θα έκανε τη συντήρηση, το testing και το debugging πολύ δύσκολο.

6. TaskData mapping

Η οντολογία δεν κρατάει sprites, φράσεις, hints αλλά κρατάει μόνο URIs και σχέσεις. Το TaskData συμπληρώνει μόνο το οπτικο και λεκτικό επίπεδο. Η σύνδεση γίνεται με ίδιο taskUri απο το individual στην οντολογία και το TaskData entry.

7. Runtime tests

Στα runtime tests εμφανίζονται τα πραγματικά προβλήματα. Η στιγμή που περνάμε στο Play Mode σε πραγματικές ροές: login → NPC → Task → sequence → reward → Narrator.

Σε αυτό το σημείο μπορούμε να δούμε καθαρά αν υπάρχουν τυχόν προβλήματα στη ροή, (1) αν τα URI κάνουν match σωστά (αν όχι τότε θα εμφανίζονται λάθος πράγματα ή δεν θα εμφανίζονται καθόλου), (2) αν υπάρχουν λάθη στα UI , (3) στα panels, (4) στη σειρά που γίνονται τα events, (5) αν οι διαλόγοι εμφανίζονται με τη σωστή σειρά, (6) αν έπρεπε ή όχι να εμφανίζεται ο συγκεκριμένος διάλογος ή ο narrator, (7) αν τα rewards δίνονται σωστά και την ώρα που πρέπει και (8) αν η μουσική και τα sound effects δουλεύουν , ρύθμιση έντασης των ηχων και αν ακούγονται εκεί που πρέπει.

8. Διόρθωση URI mismatches και UI

URI mismatches και UI ήταν το πιο συχνό σφάλμα που αντιμετωπίζαμε.

Ένα μικρό λάθος μας γυρνούσε συνεχώς πίσω στη διόρθωση των TTL. Ένα URI mismatch σήμαινε λάθος οθόνες, tasks, panels ή και καμία αντίδραση σε κλικ του παίκτη. Αυτό οδηγούσε σε εμφάνιση λάθος UI Panels, null references και επηρέαζε τη σειρά των events (π.χ να δίνεται reward πριν τον outro διάλογο του NPC).

Πίνακας λειτουργικών και μη λειτουργικών απαιτήσεων

Λειτουργικές απαιτήσεις:

- Είσοδος με όνομα/κωδικό, δημιουργία ή σύνδεση προφίλ, resume scene.
- Φόρτωση οντολογίας από StreamingAssets κατά την έναρξη gameplay scene.
- Τρία modalities με δυναμικό panel και TaskData περιεχόμενο.
- Progression σειράς NPCs + proximity checks πριν την αλληλεπίδραση.
- Intro και outro διαλόγοι μεταξύ NPC και παίκτη.
- Hints (4 τύποι) – (1) Highlighting key words, (2) Hint Phrase, (3) Wrong image removal, (4) Highlighting correct image.
- Καταγραφή επίδοσης (Accuracy, score, hints used, time) και προσαρμογή difficulty με βάση επίδοσης, μετά απο την ολοκλήρωση ενός sequence απο task (Μετά απο την ολοκλήρωση NPC).
- Reward tiers από score (Bronze/Silver/Gold/Diamond)
- One-time Narrator guidance after specific milestones.
- Μουσική και Ηχητικό feedback (SFX)

Μη λειτουργικές απαιτήσεις:

- Σταθερότητα runtime χωρίς null reference exceptions (Null checks).
- Επεκτασιμότητα: εύκολη προσθήκη νέων NPCs, tasks και scenes.
- Συντηρησιμότητα: Διαχωρισμός οντολογίας από game logic και content.
- Ασφάλεια: Ασφάλεια κωδικών (hash only).

4.2 Σχεδίαση Αρχιτεκτονικής

Υιοθετήθηκε αρχιτεκτονική τριών επιπέδων. Το Ontology Layer ορίζει τη δομή σε OWL/RDF. Το Unity Layer υλοποιεί τη game logic σε C# και το Content Mapping Layer παρέχει πολυμεσικό περιεχόμενο μέσω TaskData ScriptableObjects.

Η επικοινωνία μεταξύ επιπέδων γίνεται αποκλειστικά μέσω URIs: κάθε NPC, task και modality αναγνωρίζεται από ένα μοναδικό URI που αντιστοιχεί σε μία εγγραφή στην οντολογία και σε TaskData asset στο Unity. Αυτός ο κανόνας εξασφαλίζει ότι η οντολογία αποτελεί τη μοναδική πηγή αλήθειας (single source of truth) για τη δομή του παιχνιδιού.

4.2.1 Πίνακας αντιστοίχισης Unity component με Αρχείο

MainMenuController.cs → Login

PlayerProfileManager.cs / PlayerProfile.cs → JSON profiles

GameManager.cs → CurrentPlayer, progression order

OntologyLoader.cs → TTL parse

TaskManager.cs → task selection

TaskUI.cs → modalities, hints, scoring

NPCInteractionHandler.cs → proximity, order, start tasks

InteractiveNPC.cs + InteractionClickDetector.cs → clicks

DialogueManager.cs / NpcDialogue.cs → dialogues

PerformanceTracker.cs → metrics

DifficultyAdjuster.cs → policies

RewardManager.cs → trophies

NarratorGuideUI.cs → milestones

MoveToMouse.cs → Player movement

TaskData.asset → multimedia

4.3 Σχεδίαση και Συντήρηση Οντολογίας

Η σχεδίαση περιέλαβε τον ορισμό των κύριων κλάσεων και υποκλάσεων, object properties με domain και range, Datatype properties και τιμές policies, περιορισμοί OWL (restrictions) αλλά και individuals στο data.ttl.

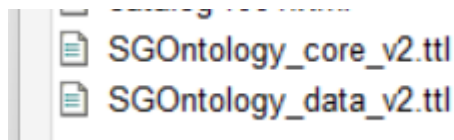
Κατά την ανάπτυξη πραγματοποιήθηκαν επαναλαμβανόμενες ενημερώσεις των URIs για τη διατήρηση συνέπειας, όπου αποδείχθηκε το κρίσιμότερο σημείο της αρχιτεκτονικής.

4.3.1 Αρχεία οντολογίας: core.ttl και data.ttl

Η οντολογία χωρίζεται σε δύο Turtle αρχεία (Διάγραμμα 3):

- SGOntology_core_v2.ttl — schema (κλάσεις, properties, restrictions, boxes) και επεξεργάζεται κυρίως στο Protégé.
- SGOntology_data_v2.ttl — individuals (NPC_Candy_Fox, SceneTemplate_CandyCity, TaskTemplates, Policies, Modalities).

Αυτός ο διαχωρισμός επιτρέπει την επέκταση του περιεχομένου (νέα tasks/NPCs) χωρίς να αλλάζει το λεξιλόγιο και χωρίς να «σπάσουν» όλα τα instances αν τηρηθούν οι URIs.



Διάγραμμα 3 - ttl files

4.3.2 Ορισμός κλάσεων και «boxes»

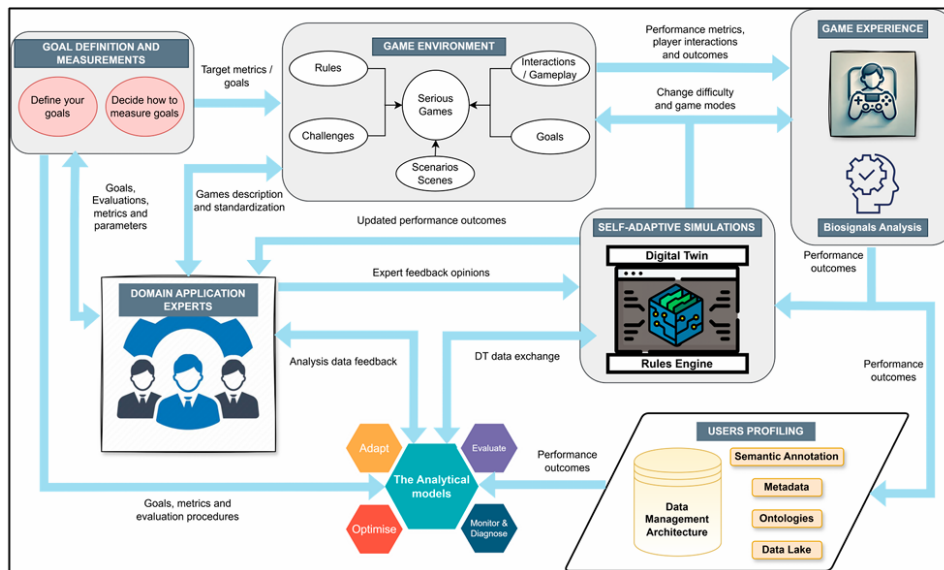
Αρχικά για την το χτίσιμο της οντολογίας έγινε ο ορισμός των βασικών κλάσεων και υποκλάσεων: Στο Protégé → tab «Classes» → Add subclass. Δημιουργήθηκαν ομαδοποιητικές κλάσεις («boxes») για οπτική οργάνωση με τις κύριες υποκλάσεις τους:

- GameEnvironmentBox → Game, GameEnvironment, GameplayElement, SceneTemplate, NPCArchetype, TaskTemplate, Modality, InteractionMode, TaskSequenceStep.
- SelfAdaptiveSimulationsBox → DifficultyProfile, DifficultyPolicy, AdaptationDecision.

- AnalyticalModelsBox → PerformanceSummary, RewardScheme, RewardTier.
- UsersProfilingBox → Player, DataLake.

Κάθε domain κλάση (π.χ. NPCArchetype) ορίζεται ως `rdfs:subClassOf` του αντίστοιχου box.

Η δομή των κλάσεων είναι εμπνευσμένη από την αρχιτεκτονική οργάνωσης που προτείνουν οι Loizou και Andreou (2025) για *serious games* με *self-adaptation* και *digital twins*. (Βλέπε Διάγραμμα 4):



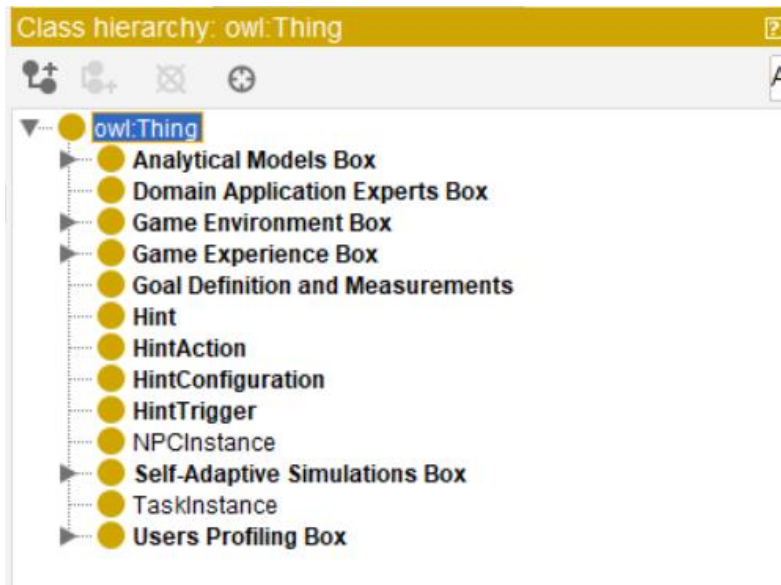
Διάγραμμα 4 – Δομή κλάσεων

4.3.2.1 Κλάσεις οντολογίας και Ιεραρχία

Παρακάτω φαίνονται τα «boxes» που δημιουργήθηκαν για την οργάνωση της οντολογίας. Με αρχική κλάση την `owl:Thing` όπου το protege δημιουργεί αυτόματα σαν την αρχική. Όλες οι υπόλοιπες κλάσεις μπαίνουν αμέσως μετά από αυτή.

(Βλέπε διάγραμμα 5)

Σημείωση: Στις παρακάτω figures υπάρχουν κλάσεις οι οποίες δεν χρησιμοποιούνται στο παρόν σύστημα. Δημιουργήθηκαν για δοκιμαστικούς σκοπούς ή/και για μελλοντική επέκταση της οντολογίας.

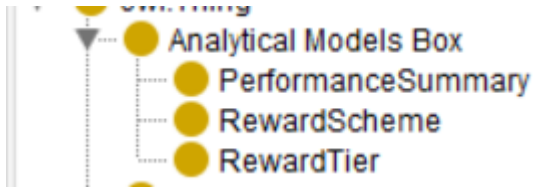


Διάγραμμα 5 – Hierarchy in Protege

Analytical Models Box:

Το Analytical Models Box ομαδοποιεί ό,τι αφορά μέτρηση απόδοσης και ανταμοιβές. Υπάρχουν ως subclasses οι κλάσεις: PerformanceSummary, όπου δέν χρησιμοποιείται, βρίσκεται απλά σαν ένα template του πώς πρέπει να γίνεται η αποθήκευση του performance ενός παίκτη (Η πραγματική ευρεση και αποθήκευση του performance γίνεται στο Unity). (Βλέπε Διάγραμμα 6).

Οι κλάσεις RewardScheme και RewardTier ορίζουν στο semantic layer ποιες κατηγορίες rewards ισχύουν και με ποια κατώφλια score (π.χ. RewardScheme_Default με hasRewardTier → Bronze–Diamond). Η SceneTemplate_CandyCity συνδέεται με αυτό το scheme μέσω του referencesRewardScheme και ο OntologyLoader φορτώνει τα αντίστοιχα individuals στο runtime. Στην τρέχουσα υλοποίηση όμως, η ανάθεση tier μετά από sequence βασίζεται στον RewardManager με σταθερά όρια score (60/80/100). Το reward model της οντολογίας τεκμηριώνει τη σχεδίαση και επιτρέπει μελλοντική πλήρη ανάγνωση thresholds από TTL αλλά χωρίς να αντικαθιστά ακόμα το TaskPerformance-based scoring.» (Βλέπε Διάγραμμα 7)



Διάγραμμα 6 - Analytical Models Box

Property assertions: PerfSummary_Demo_Task1	
Object property assertions +	
■	appliedHint 'Hint given to Player_Demo during WordToImage task (Need to be deleted)'
■	generatedForPlayer Player_Demo
■	producesAdaptation Adaptation_Demo_NextTask
■	recordedInDataLake DataLake_Main
■	subjectToPolicy Policy_Increase
■	summaryInScene SceneTemplate_CandyCity
■	summaryOfTask 'Match the word to the correct image.'
Data property assertions +	
■	accuracyValue 0.88
■	eventTimestamp "2025-11-12T09:30:00Z"^^xsd:dateTime
■	hintCount 1
■	latencyMs 1400
■	recommendedDifficultyShift 1
■	scoreValue 0.90

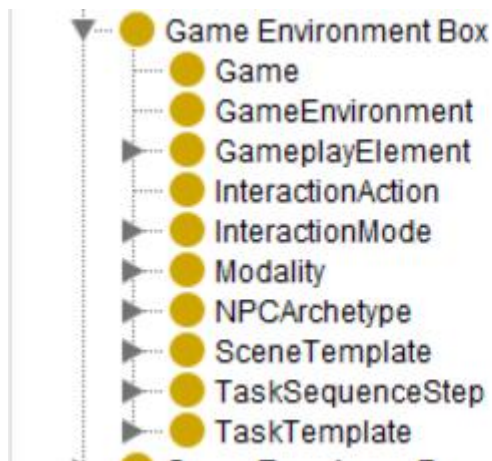
Διάγραμμα 7 - Performance summary Demo

Game Environment Box

Το Game Environment Box, είναι το σημαντικότερο μέρος της οντολογίας με κύριες κλάσεις του demo να είναι κυρίως (1) NPCArchetype, (2) TaskTemplate και (3) Modality όπου ορίζονται τα NPCs, τα tasks τους και τι είδους tasks είναι και σε ποιά difficulty πρέπει αυτά τα tasks να εμφανιστούν μέσω (hasTaskTemplate, usesModality, initialDifficultyProfile) σε συνδιασμό με URI matching στο Unity inspector. Η SceneTemplate είναι επίσης πολύ σημαντική γιατί λειτουργεί στον εντοπισμό της τρέχουσας σκηνής και στη σύνδεση NPC με policies.

Οι κλάσεις Game, GameEnvironment, GameplayElement, InteractionMode και InteractionAction φορτώνονται απο τον OntologyLoader χωρίς όμως να

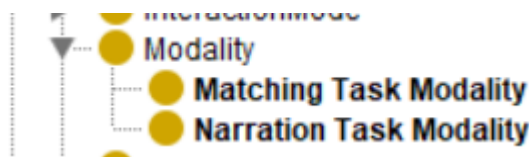
χρησιμοποιούνται ακόμη έμμεσα στο σύστημα. Η taskSequenceStep δείχνει την τυπική σειρά των βημάτων στη σκηνή αλλά η τρέχουσα ροή βασίζεται κυρίως σε hasTaskTemplate ανα NPC και σε σειρά NPCs στον GameManager, όχι απο αυτές τις κλάσεις. Οι κλάσεις αυτές δημιουργήθηκαν ως μία βάση για μελλοντική δουλειά και επεκτασιμότητα. (Βλέπε Διάγραμμα 8)



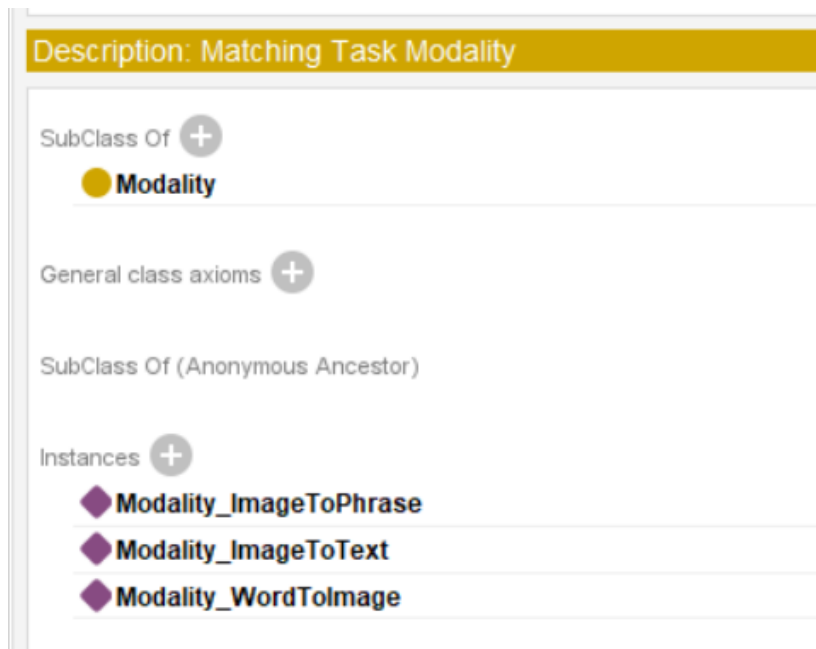
Διάγραμμα 8 - Game Environment Box

Παρακάτω βλέπουμε πιο αναλυτικά την κάθε υπο-κλάση

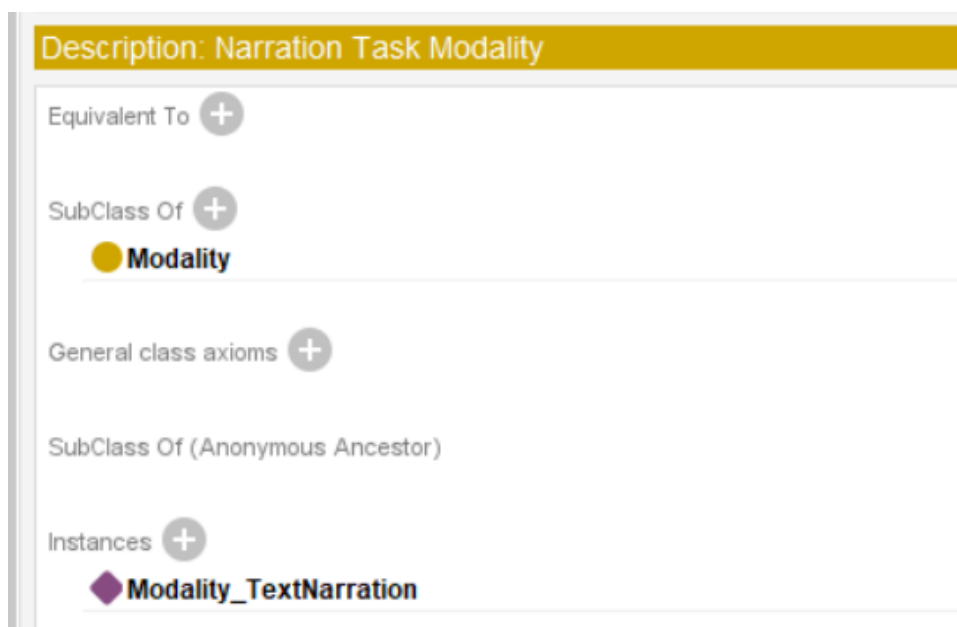
Modality → Matching Task Modality με τα τρία είδη modality που μπορούν να έχουν τα tasks του μοντέλου. (Image to Phrase, Image to Text, Word to Image) και Narration Task Modality όπου χρησιμοποιείται για τη χρήση του narrator στο σύστημα. (Βλέπε Διάγραμμα 9,10 και 11)



Διάγραμμα 9 – Modality



Διάγραμμα 10 - Είδη Task Modality

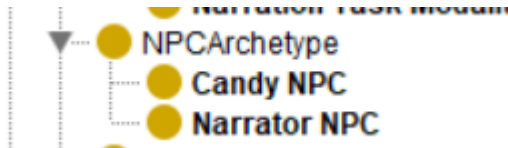


Διάγραμμα 11 - Narration Modality

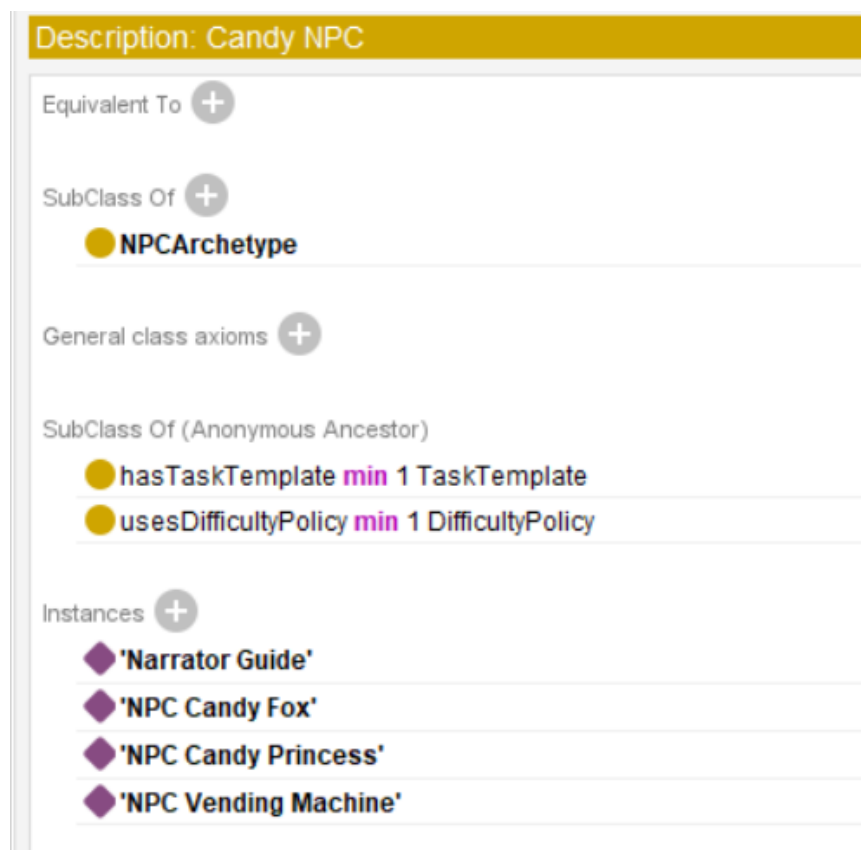
NPCArchetype → Εδώ βλέπουμε τα NPCs που υπάρχουν στη κάθε σκηνή (Candy Scene → NPC Candy Fox , Vending Machine , Candy Princess) και τον Narrator που θεωρούμε σαν ένα «φανταστικό βοηθητικό NPC» που ο σκοπός του δεν είναι να είναι άλλο ένα gameplay NPC με task modalities αλλά να έχει ρόλο narration tasks (Modality_TextNarration). Υπάρχουν επίσης τα 3 Instances του narrator (Guide, dialog

και completion). Προς το παρόν όμως στο μοντέλο, ο narrator δουλεύει μέσω του Unity με το NarratorGuideUI και κείμενα στο Inspector και γίνεται trigger απο τα URIs των 3 gameplay NPCs. (Βλέπε Διάγραμμα 12,13 και 14)

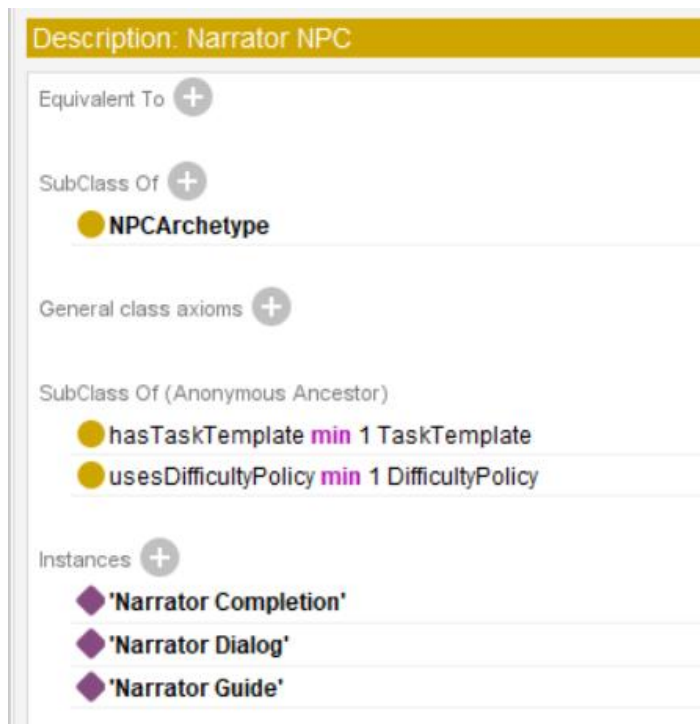
Μελλοντική σύνδεση του NarratorGuideUI με την οντολογία, έτσι ώστε να διαβάζει τα τις φράσεις του απο την οντολογία αντί απο το inspector.



Διάγραμμα 12 - NPCArchetype



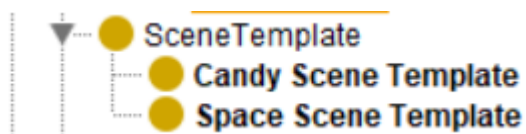
Διάγραμμα 13 - Candy NPC



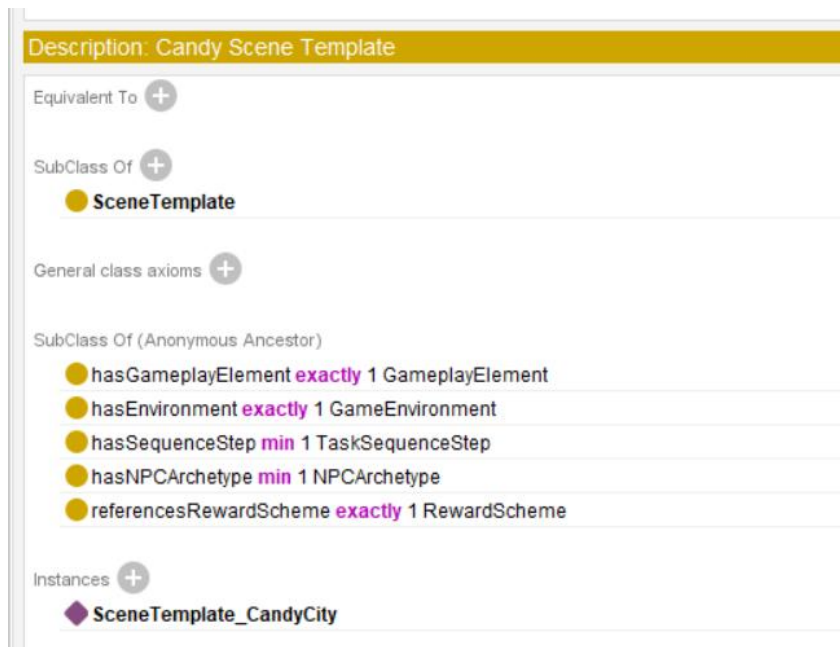
Διάγραμμα 14 - Narrator NPC

Scene Template → Εδώ φαίνονται όλες οι σκηνές που βρίσκονται στο μοντέλο

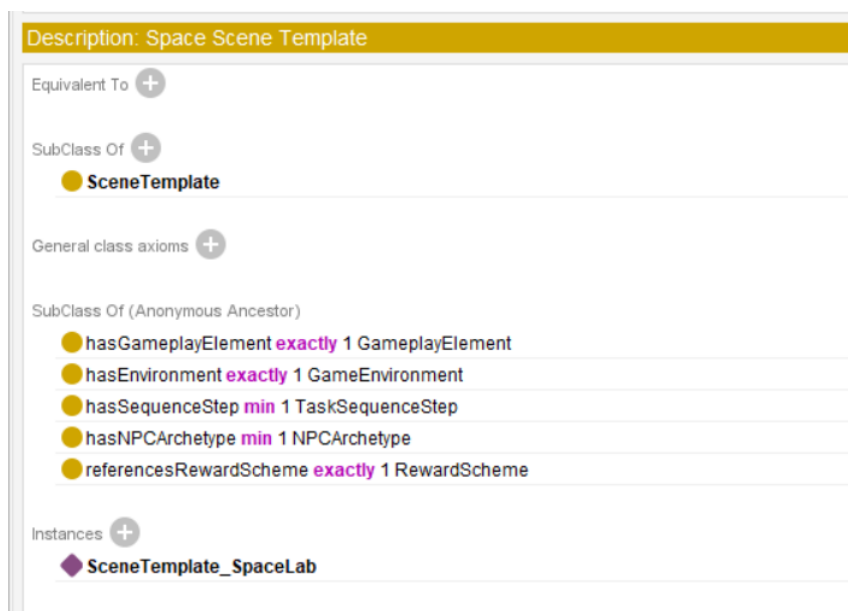
Candy Scene Template → Candy City (Active) & Space Scene Template → Space Lab (Future Work) (Βλέπε Διάγραμμα 15,17 και 17)



Διάγραμμα 15 - Scene templates



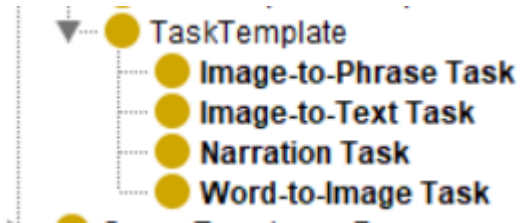
Διάγραμμα 16 - Candy City



Διάγραμμα 17 - Space Lab

TaskTemplate → Εδώ βλέπουμε τα 3 είδη tasks που υπάρχουν (Image to Phrase, Image to Text, Word to Image) αλλά και τα instances (προτάσεις και λέξεις που χρησιμοποιούνται στα tasks). Δεν τα ψάχνουμε ακριβώς με το ονομά τους (π.χ Image-to-Phrase Task), αλλά μέσω του Unity ο OntologyLoader δημιουργεί αντικείμενα όταν το rdf:type είναι TaskTemplate και μετά το TaskUI αποφασίζει τι panel να δώσει απο

το task.UsesModalityUri, όχι απευθείας από το TaskTemplate. (Βλέπε Διάγραμμα 18,19 και 20)



Διάγραμμα 18 - Task templates

Description: Image-to-Phrase Task

Equivalent To

SubClass Of

- TaskTemplate

General class axioms

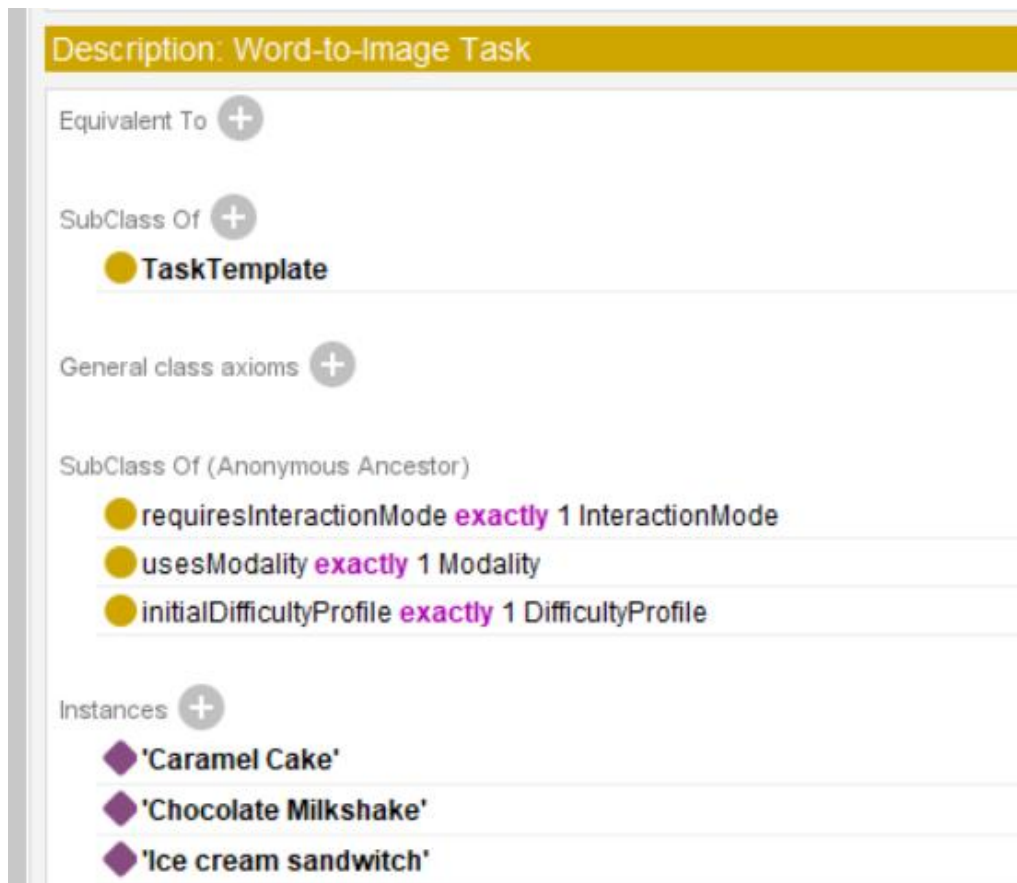
SubClass Of (Anonymous Ancestor)

- requiresInteractionMode **exactly** 1 InteractionMode
- usesModality **exactly** 1 Modality
- initialDifficultyProfile **exactly** 1 DifficultyProfile

Instances

- 'A boy in a boat going down a chocolate river'
- 'A boy licking a big strawberry lollipop.'
- 'A girl is carrying a box of colorful cupcakes across the CandyCity bridge.'
- 'Alex and his friends are racing down the chocolate river in candy cane boats, while holding bags full of jellybeans.'

Διάγραμμα 19 - Παραδείγματα φράσεων που χρησιμοποιούνται σε tasks



Διάγραμμα 20 - Παραδείγματα λέξεων που χρησιμοποιούνται σε tasks

Self-Adaptive Simulations Box

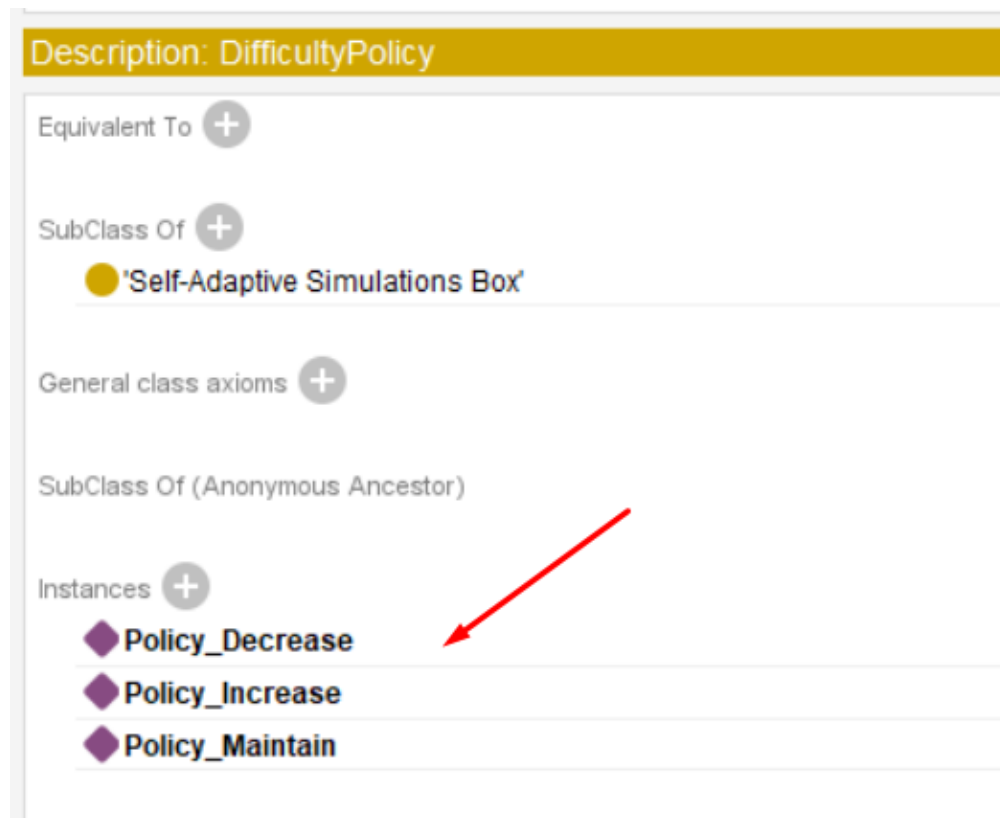
Το Self-Adaptive Simulations Box ομαδοποιεί την προσαρμοστική λογική, με βασικές κλάσεις του prototype να είναι (1) «DifficultyProfile» όπου είναι υπεύθυνη για να «δειχνει» το επίπεδο δυσκολίας ενός παίκτη (Beginner, Intermediate και Advanced) έτσι ώστε να φιλτράρονται τα tasks με τη δυσκολία που πρέπει για τον κάθε παίκτη. (2) «DifficultyPolicy» όπου αντιπροσωπεύει τα τρία policies που υπάρχουν στο σύστημα για την αλλαγή της δυσκολίας του παίκτη με βάση το performance του. (Policy_Increase → Ανεβάζει το επίπεδο δυσκολίας αν ο παίκτης τα πάει καλά στα tasks του, Policy_Maintain → Κρατάει τη δυσκολία στο ίδιο επίπεδο αν ο παίκτης έχει μέτρια απόδοση, Policy_Decrease → Μειώνει τη δυσκολία αν ο παίκτης δεν έχει καλή απόδοση στα tasks). Αυτές οι policies βασίζονται σε μετρικές όπως το average accuracy που είχε ο παίκτης μετά την ολοκλήρωση κάποιου sequence απο tasks (ολοκλήρωση του NPC) και πόσα hints χρησιμοποίησε. Με βάση των 3 κανόνων αλλάζεται το

«policy_delta» σε -1/0/+1 ανάλογα με το ποιο policy χρησιμοποιήθηκε (-1 μειώνει το επίπεδο, 0 το κρατάει σταθερό, +1 αυξάνει το επίπεδο δυσκολίας).

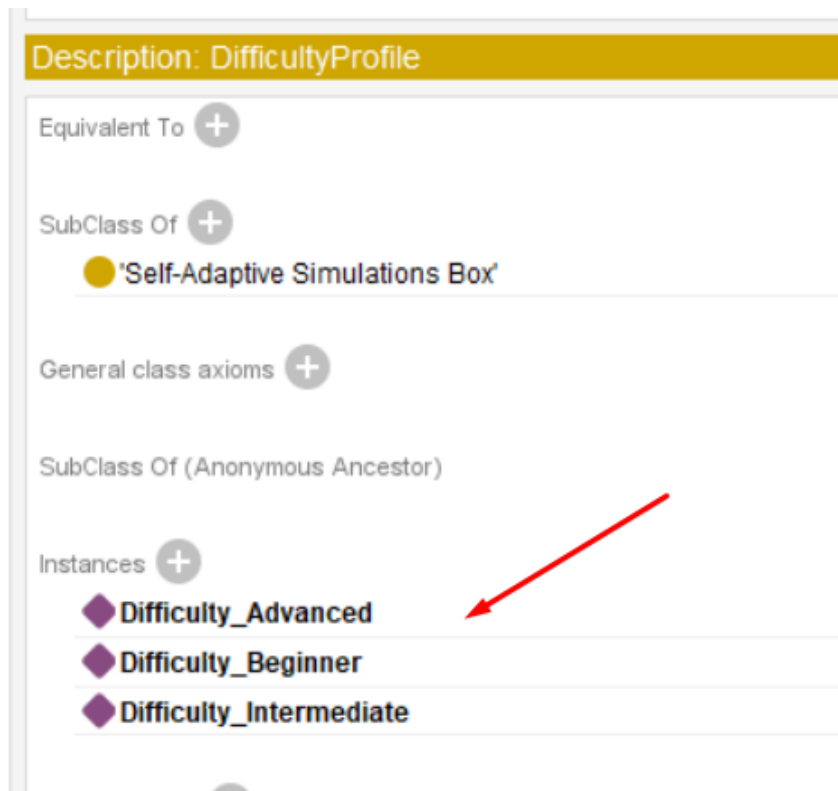
Οι κλάσεις Difficulty, AdaptationDecision και DigitalTwin δεν χρησιμοποιούνται στο παρών prototype αλλά δημιουργήθηκαν για μελλοντική δουλειά και πρόσθεση digital twins. (Βλέπε Διάγραμμα 21,22 και 23)



Διάγραμμα 21 - Self-Adaptive Simulations Box



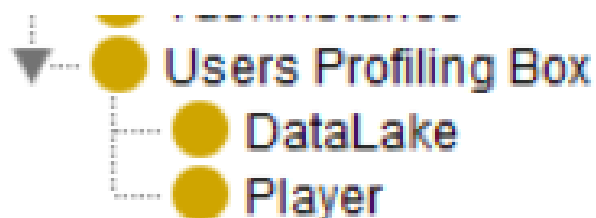
Διάγραμμα 22 - Policies



Διάγραμμα 23 - Difficulties

Users Profiling Box

Το Users Profiling Box έχει δημιουργηθεί για μελλοντικούς σκοπούς προσθήκης data lakes για την αποθήκευση δεδομένων και δεν χρησιμοποιείται για το παρών prototype. (Βλέπε Διάγραμμα 24).



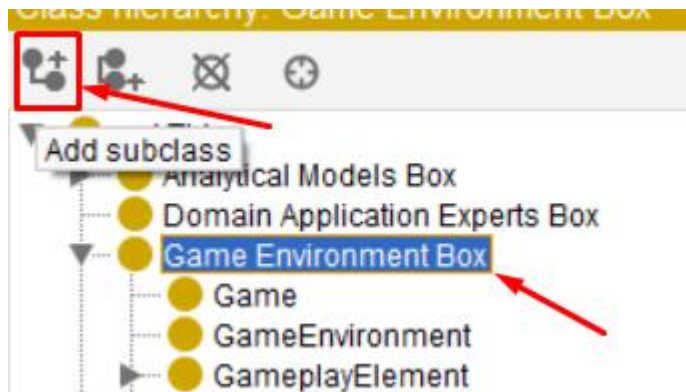
Διάγραμμα 24 - Users Profiling Box - Future Work

4.3.2.2 Δημιουργία κλάσεων και υποκλάσεων (Πρακτικό)

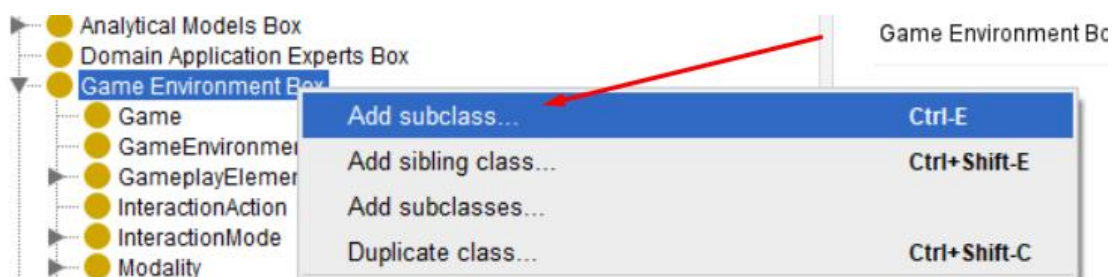
Αρχικά η «κύρια» κλάση είναι η owl:Thing η οποία δημιουργείται αυτόματα απο το Protege κατα την δημιουργία της οντολογίας και στη συνέχεια προσθέτουμε subclasses πάνω σε αυτή.

Η δημιουργία των classes γίνεται με τον εξής τρόπο:

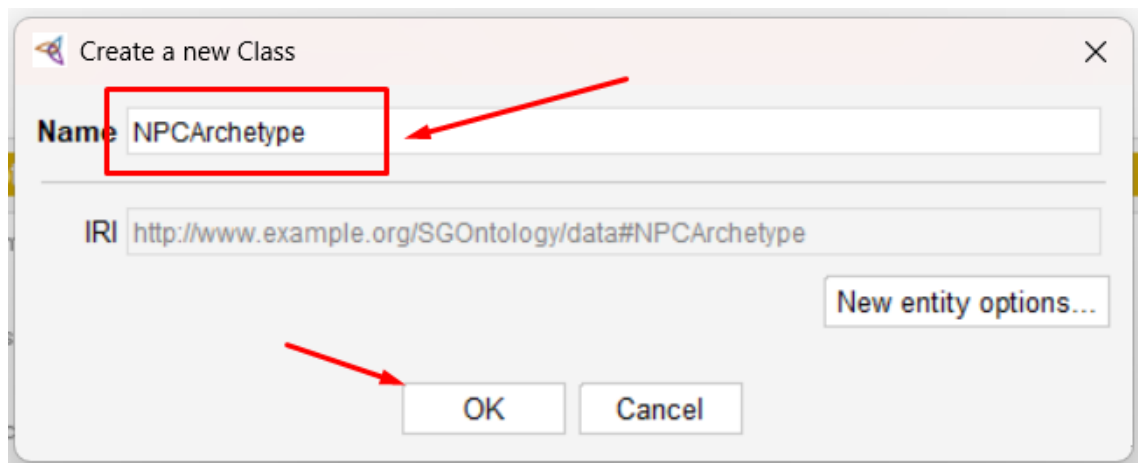
Επιλογή κλάσης-γονέας (π.χ Game Environment Box) → Κουμπί Add Subclass ή δεξί κλικ + add subclass → Εισαγωγή ονόματος subclass (π.χ NPCArchetype) → Η νέα κλάση δημιουργείται με αυτόματη σχέση Subclass Of προς τον γονέα. (Βλεπε Διάγραμμα 25,26,27, 28 και 29).



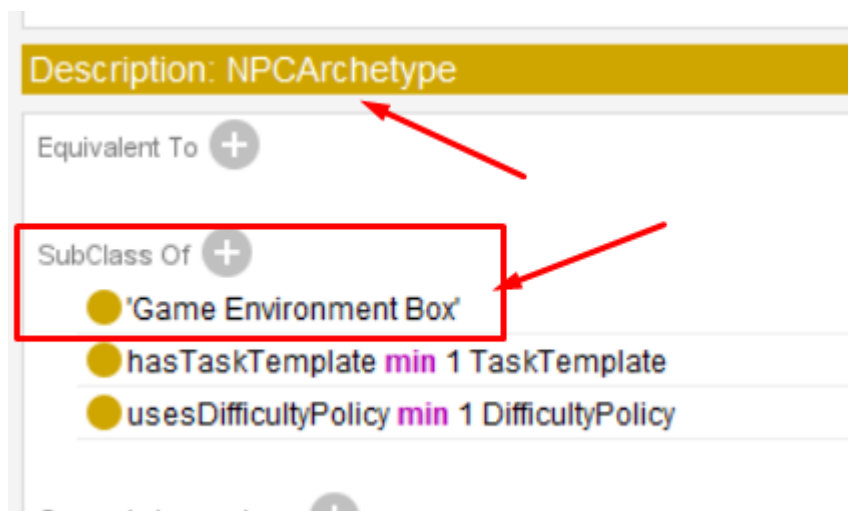
Διάγραμμα 25 - Add subclass



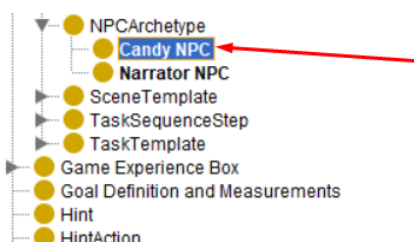
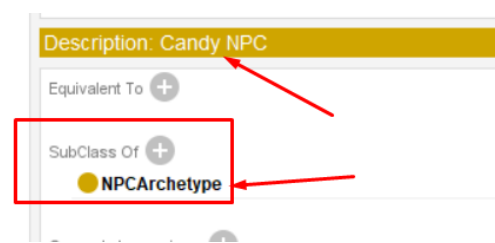
Διάγραμμα 26 - Add subclass



Διάγραμμα 27 - Subclass Name



Διάγραμμα 28 - Parent-child class relation



Διάγραμμα 29 - Parent-Child class relation

4.3.3 Object properties → Domain και Range

Μετά τον ορισμό των κλάσεων, το επόμενο βήμα είναι ο ορισμός των *object properties*. Στην OWL, μια object property είναι τύπος σχέσης μεταξύ δύο individuals. Οι object properties βασικά δηλώνουν *ποιος* συνδέεται με *ποιον* και με *ποια* έννοια (π.χ. «αυτό το NPC προσφέρει αυτό το task»).

Στο αρχείο SGOntology_core_v2.ttl κάθε property δηλώνεται ως owl:ObjectProperty και συνοδεύεται από rdfs:domain και rdfs:range, που δεν είναι «τιμές» στο παιχνίδι αλλά τυπολογικοί περιορισμοί: το *subject* της σχέσης πρέπει να ανήκει στο domain και το *object* στο range. Έτσι η οντολογία λέει ότι, π.χ., η hasTaskTemplate συνδέει NPCArchetype → TaskTemplate και όχι αυθαίρετα άλλους συνδυασμούς κλάσεων. Σχηματίζουν βασικά μία αλυσίδα. (Βλέπε Διάγραμμα 30 και 31)

Annotations: NPCArchetype

Annotations +

rdfs:comment [in core]
Defines a type/role of NPC. Every NPCArchetype must have at least one TaskTemplate and at least one DifficultyPolicy.

Description: NPCArchetype

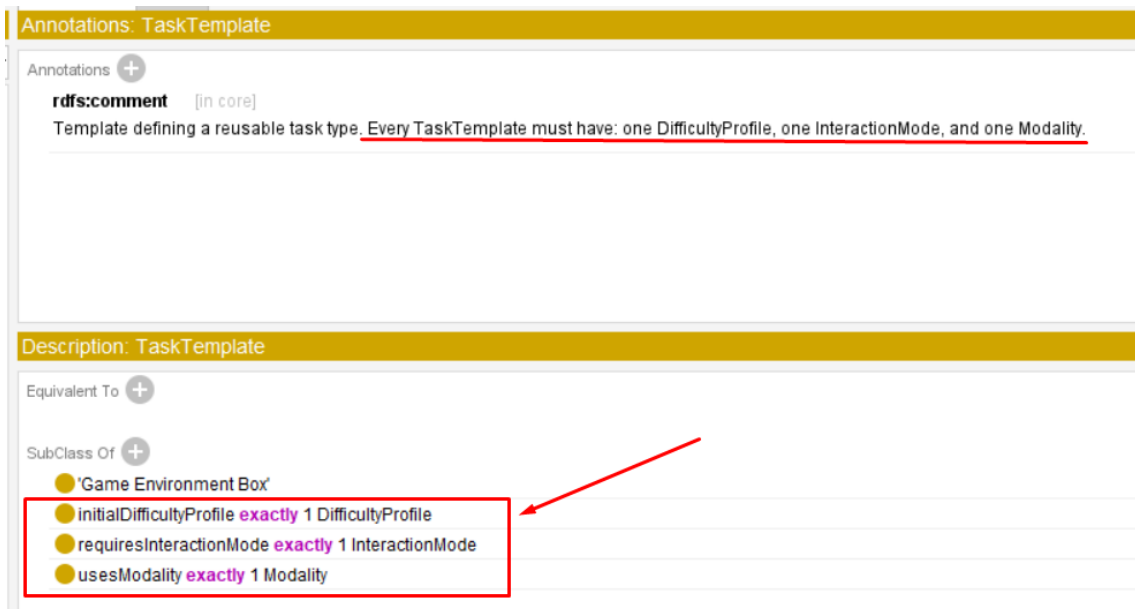
Equivalent To +

SubClass Of +

- 'Game Environment Box'
- hasTaskTemplate **min 1** TaskTemplate
- usesDifficultyPolicy **min 1** DifficultyPolicy

General class axioms +

Διάγραμμα 30 - Range



Διάγραμμα 31 - Triples and restrictions in Protege

Στα πιο πάνω διαγράμματα (Διάγραμμα 30 και figure 31), φαίνεται η αλυσίδα των κλάσεων με τα object properties τους και το πώς ενώνονται μεταξύ τους.

NPCArchetype → hasTaskTemplate → min 1 → TaskTemplate

NPCArchetype → usesDifficultyPolicy → min 1 → DifficultyPolicy,

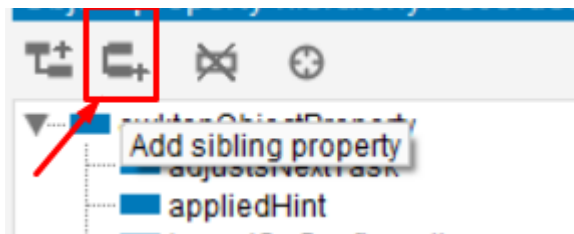
Και TaskTemplate → has initialDifficultyProfile → exactly 1 → DifficultyProfile
requiresInteractionMode → exactly 1 → InteractionMode,

usesModality → Exactly 1 → Modality

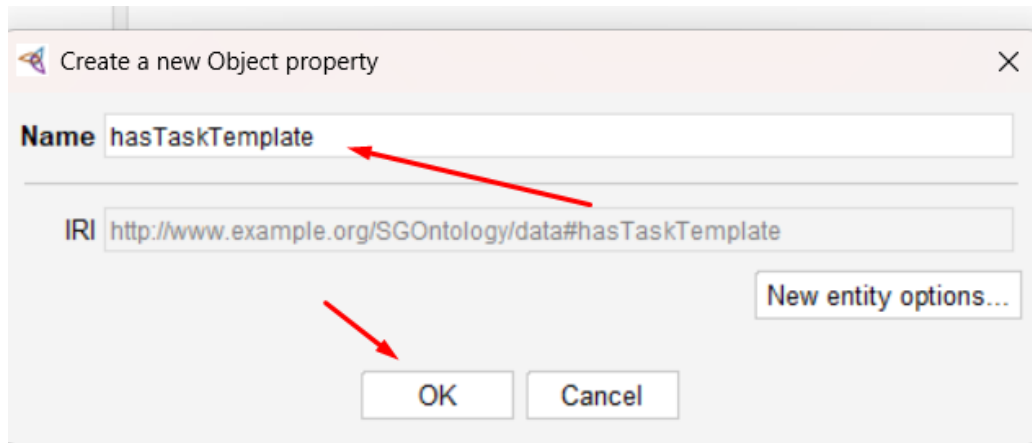
(όπου μεταφράζεται σε: Κάθε NPCArchetype πρέπει να έχει τουλάχιστον ένα TaskTemplate και τουλάχιστον ένα Difficulty Policy όπου κάθε TaskTemplate πρέπει να έχει ακριβώς ένα DifficultyProfile, ακριβώς ένα InteractionMode και ακριβώς ένα Modality). Έτσι δημιουργείται μια αλυσίδα σχέσης μεταξύ των κλάσεων.

4.3.3.1 Δημιουργία Object Properties

Για την δημιουργία των Object Properties, ανοίγουμε το Object Properties Tab στο Protégé και πατάμε το κουμπί «add object property» → Εισαγωγή ονόματος Object Property.(Βλέπε Διάγραμμα 32 και 33).



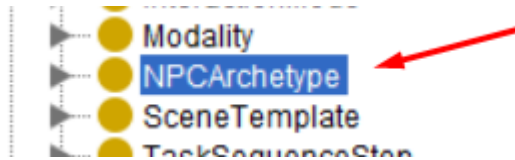
Διάγραμμα 32 - Adding a new object property



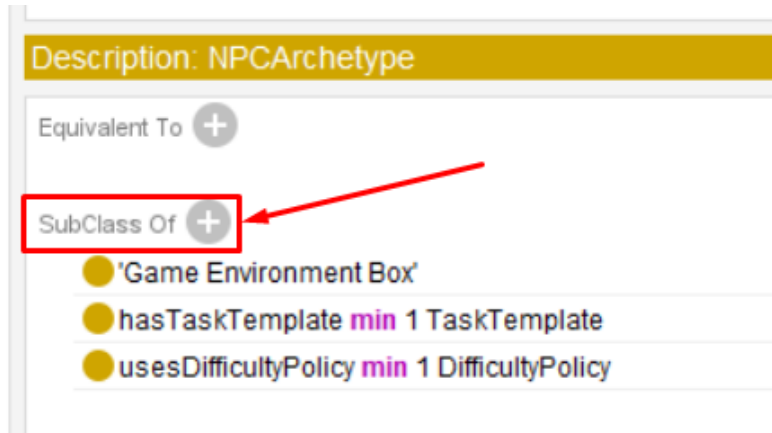
Διάγραμμα 33 - Name input for new Object property

4.3.3.2 Προσθήκη Object Property σε Domain και Range

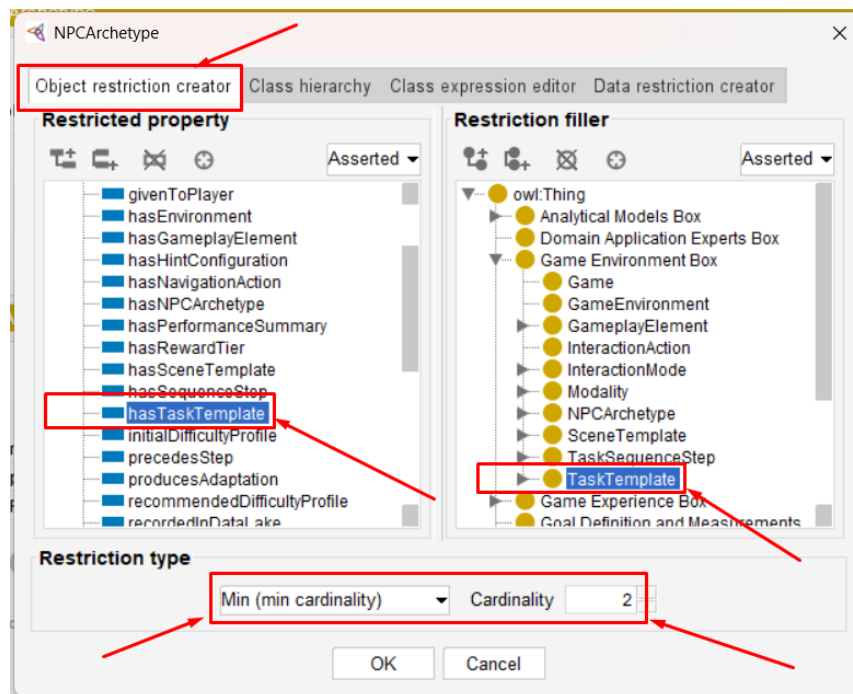
Για να προσθέσουμε ένα Object property στις 2 κλάσεις (Domain και Range), επιλέγουμε τη κλάση που θέλουμε ως Domain → Έπειτα στο description της κλάσης πατάμε το κουμπί (+) δίπλα απο το «Subclass Of» → απο το παράθυρο που ανοίγει ανοίγουμε το tab «Object restriction creator» και επιλέγουμε το Object Property απο τον «Restricted property» πίνακα με όλα τα object properties και έπειτα την κλάση (Range) απο τον «Restriction filler» πίνακα → προχωράμε στο restriction type και επιλέγουμε (some/only/min/exactly/max) και το cardinality (τον αριθμό που θέλουμε). (Βλέπε Διάγραμμα 34, 35 και 36).



Διάγραμμα 34 – Class in Protege



Διάγραμμα 35 - Where to add object properties



Διάγραμμα 36 - Adding object properties to domain and range

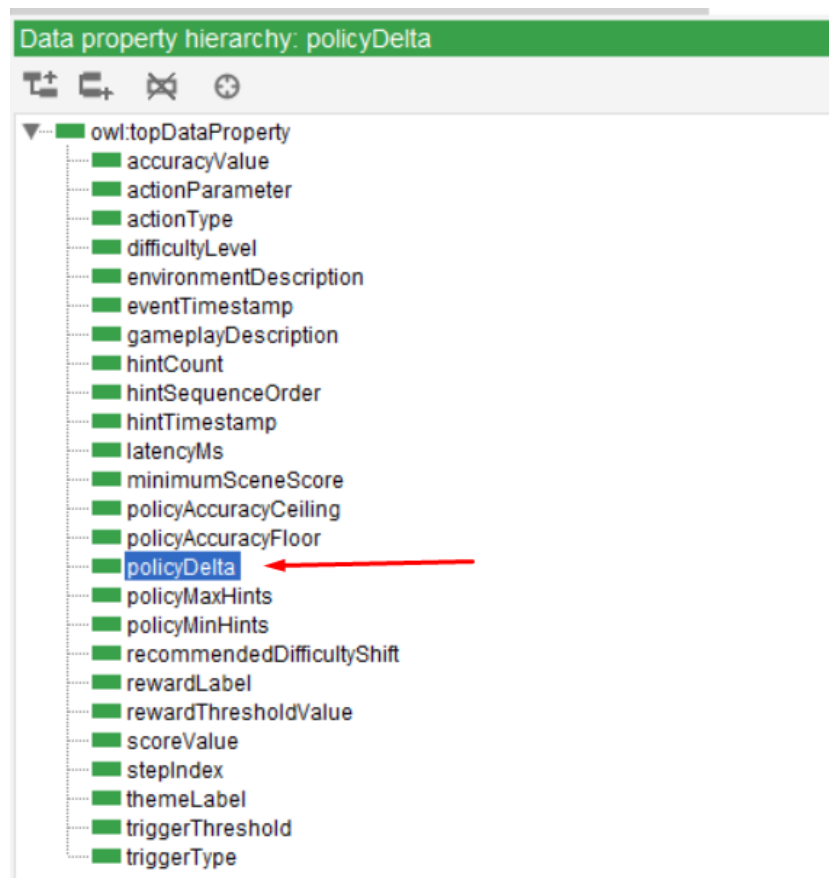
4.3.4 Datatype properties και τιμές policies

Οι datatype properties είναι ιδιότητες που συνδέουν κάποιο individual με μία απλή τιμή. Το range τους είναι ένα datatype από το XML Schema (xsd:integer, xsd:decimal, xsd:string κτλ.), ενώ στις object properties το range είναι μια άλλη κλάση. Οι datatype properties βασικά προσθέτουν τα αριθμητικά ή κειμενικά χαρακτηριστικά.

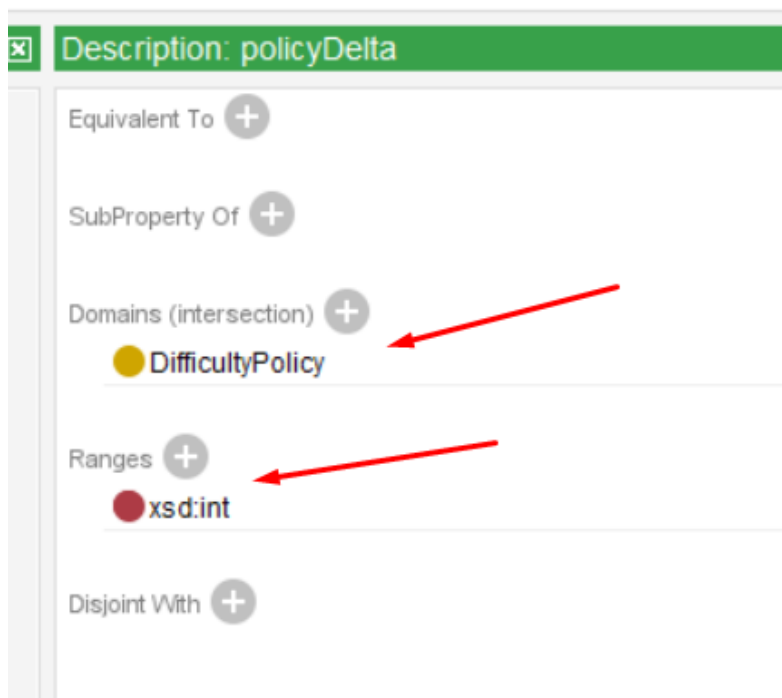
Ορίστηκαν στο Data properties tab του protege, με domain την αντίστοιχη κλάση (π.χ DifficultyPolicy για τα policies). Οι τιμές δηλώνονται ως assertions πάνω σε Named Individuals (π.χ PolicyIncrease) στο data.ttl.

Π.χ policyDelta (4.3.2.1 → Self-Adaptive Simulations Box), όπου μιλήσαμε για το πώς αλλάζει το επίπεδο δυσκολίας ενός παίκτη (-1/0/+1).

Έχει δημιουργηθεί ένα datatype property «policyDelta» το οποίο έχει ως Domain την κλάση «DifficultyPolicy» και ως Range το «xsd:int» όπου θα παίρνει τις τιμές (-1/0/+1) που θα του δίνονται. (Βλέπε Διάγραμμα 37 και 38).



Διάγραμμα 37 – Data type property in protege



Διάγραμμα 38 – Domain and Range of a data property in protege

Σημείωση: Η διαδικασία δημιουργίας και προσθήκης των *data properties* είναι ακριβώς η ίδια με την διαδικασία δημιουργίας και προσθήκης των *Object properties*

4.3.5 Περιορισμοί OWL (restrictions) -- παραδείγματα από την οντολογία

Οι περιορισμοί χρησιμοποιήθηκαν έτσι ώστε να υπάρχει μία ισορροπία και συνέπεια στην οντολογία και γενικά στο παιχνίδι, χωρίς να υπάρχουν π.χ άδειες σκηνές ή NPCs χωρίς tasks/διαλόγους ή rewards κτλπ.

Στο Protégé: Class → Description → Add subclass (anonymous restriction).

Χρησιμοποιήθηκαν κυρίως `minQualifiedCardinality` και `qualifiedCardinality`.

Μερικά παραδείγματα:

Game → πρέπει να έχει τουλάχιστον 1 `SceneTemplate`:

```

## http://www.example.org/SGOntology/core#Game
go:Game rdf:type owl:Class ;
    rdfs:subClassOf sgo:GameEnvironmentBox ,
        [ rdf:type owl:Restriction ;
          owl:onProperty sgo:hasSceneTemplate ;
          owl:minQualifiedCardinality "1"^^xsd:nonNegativeInteger ;
          owl:onClass sgo:SceneTemplate
        ] ;

```

Διάγραμμα 39 – Game restrictions in ttl

NPCArchetype → πρέπει να έχει τουλάχιστον 1 TaskTemplate και 1 DifficultyPolicy

```

## http://www.example.org/SGOntology/core#NPCArchetype
go:NPCArchetype rdf:type owl:Class ;
    rdfs:subClassOf sgo:GameEnvironmentBox ,
        [ rdf:type owl:Restriction ;
          owl:onProperty sgo:hasTaskTemplate ;
          owl:minQualifiedCardinality "1"^^xsd:nonNegativeInteger ;
          owl:onClass sgo:TaskTemplate
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty sgo:usesDifficultyPolicy ;
          owl:minQualifiedCardinality "1"^^xsd:nonNegativeInteger ;
          owl:onClass sgo:DifficultyPolicy
        ] ;

```

Διάγραμμα 40 – NPCArchetype restrictions in ttl

SceneTemplate → πρέπει να έχει τουλάχιστον 1 NPC, 1 TaskSequenceStep, ακριβώς 1 GameEnvironment / GameplayElement / RewardScheme:

```

## http://www.example.org/SGOntology/core#SceneTemplate
go:SceneTemplate rdf:type owl:Class ;
    rdfs:subClassOf sgo:GameEnvironmentBox ,
        [ rdf:type owl:Restriction ;
          owl:onProperty sgo:hasNPCArchetype ;
          owl:minQualifiedCardinality "1"^^xsd:nonNegativeInteger ;
          owl:onClass sgo:NPCArchetype
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty sgo:hasSequenceStep ;
          owl:minQualifiedCardinality "1"^^xsd:nonNegativeInteger ;
          owl:onClass sgo:TaskSequenceStep
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty sgo:hasEnvironment ;
          owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
          owl:onClass sgo:GameEnvironment
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty sgo:hasGameplayElement ;
          owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
          owl:onClass sgo:GameplayElement
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty sgo:referencesRewardScheme ;
          owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
          owl:onClass sgo:RewardScheme
        ] ;

```

Διάγραμμα 41 – Scene template restrictions in ttl

TaskTemplate → πρέπει να έχει ακριβώς 1 DifficultyProfile, 1 InteractionMode, 1 Modality

```
## http://www.example.org/SGOntology/core#TaskTemplate
go:TaskTemplate rdf:type owl:Class ;
                rdfs:subClassOf sgo:GameEnvironmentBox ,
                [ rdf:type owl:Restriction ;
                  owl:onProperty sgo:initialDifficultyProfile ;
                  owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
                  owl:onClass sgo:DifficultyProfile
                ] ,
                [ rdf:type owl:Restriction ;
                  owl:onProperty sgo:requiresInteractionMode ;
                  owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
                  owl:onClass sgo:InteractionMode
                ] ,
                [ rdf:type owl:Restriction ;
                  owl:onProperty sgo:usesModality ;
                  owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
                  owl:onClass sgo:Modality
                ] ;
```

Διάγραμμα 42 – Task Template restrictions in ttl

4.3.6 Individuals στο data.ttl

Στο data.ttl υπάρχουν όλα τα individuals δηλαδή συγκεκριμένα instances απο υπάρχουσες κλάσεις (π.χ Class – NPCArchetype → Individual – NPC_Candy_Fox) Το Candy_Fox είναι ένα συγκεκριμένο instance της κλάσης NPCArchetype.

Αντιπροσωπεύουν βασικά τα πραγματικά δεδομένα του παιχνιδιού.

Κάθε individual έχει συνήθως:

1. rdf:type → Σε ποιά κλάση ανήκει (π.χ TaskTemplate)
2. Object property assertions → Συνδέσεις με άλλα individuals (π.χ hasTaskTemplate)
3. Datatype property assertions → αριθμοι/κείμενα (π.χ policyDelta, difficultyLevel , rdfs:label)

Παράδειγμα NPC:

- Τύπος: NPCArchetype
- Συνδέεται με tasks: CandyFox_Task1_Beginner
- *Συνδέεται με* policies: Policy_Increase, Policy_Maintain, Policy_Decrease
- Label: «NPC Candy Fox»

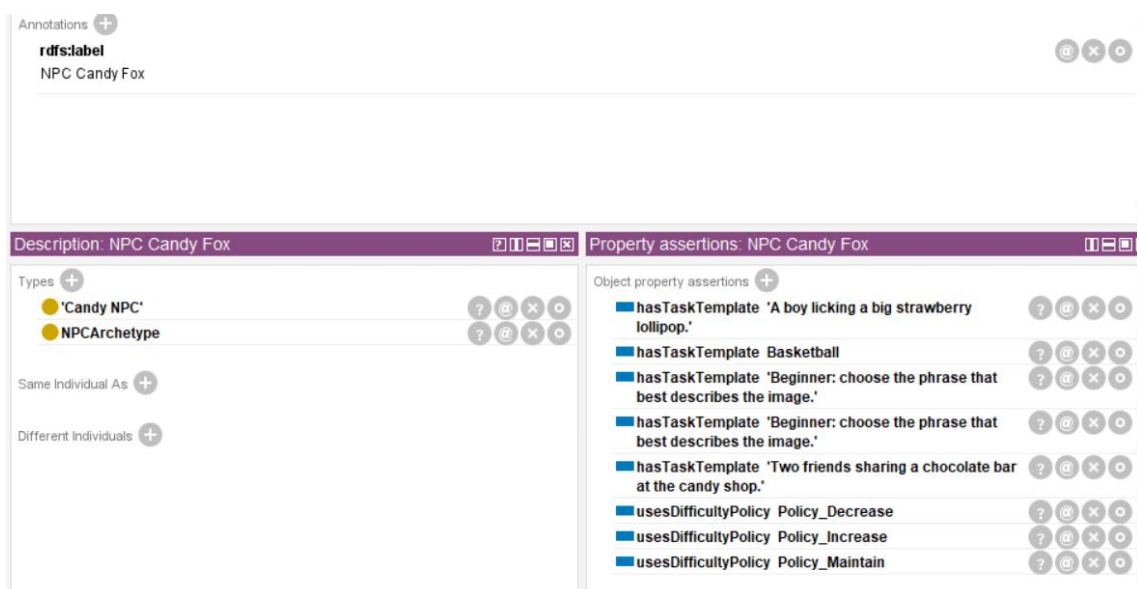
```

### http://www.example.org/SGOntology/data#NPC_Candy_Fox
:NPC_Candy_Fox rdf:type owl:NamedIndividual ,
                sgo:NPCArchetype ;
                sgo:hasTaskTemplate :CandyFox_Task1_Beginner ,
                                   :CandyFox_Task2_Beginner ,
                                   :CandyFox_Task3_Beginner ,
                                   :CandyFox_Task4_Beginner ,
                                   :CandyFox_Task5_Beginner ;
                sgo:usesDifficultyPolicy :Policy_Decrease ,
                                       :Policy_Increase ,
                                       :Policy_Maintain ;
                rdfs:label "NPC Candy Fox" .

```

Διάγραμμα 43 - Individual in ttl file

NPC Candy Fox individual στο Protégé: (Βλέπε Διάγραμμα 44)



Διάγραμμα 44 – NPC Individual: Protege

4.3.7 Reasoner και έλεγχος συνέπειας

Η επαλήθευση της οντολογίας γίνεται σε δύο επίπεδα. Στο Protege, με OWL reasoner (Pellet), ελέγχονται η συνέπεια κλάσεων, ικανοποίηση restrictions από τα individuals και αν τα domain/range είναι σωστά πριν την εξαγωγή των core.ttl και data.ttl.

4.3.8 SPARQL

Έγινε χρήση SPARQL Queries manually στο Protégé για σκοπούς έλεγχου της οντολογίας και της ροής του παιχνιδιού. Παρακάτω φαίνονται κάποια παραδείγματα SPARQL Query και τα αποτελέσματά τους.

Candy City Scene: NPCs και ο αριθμός των tasks που προσφέρουν.

```
SPARQL query:

PREFIX sgo: <http://www.example.org/SGOntology/core#>
PREFIX data: <http://www.example.org/SGOntology/data#>

SELECT ?npc ?npcLabel (COUNT(?task) AS ?taskCount)
WHERE {
  data:SceneTemplate_CandyCity a sgo:SceneTemplate ;
    sgo:hasNPCArchetype ?npc .
  ?npc rdfs:label ?npcLabel .
  OPTIONAL { ?npc sgo:hasTaskTemplate ?task . }
}
GROUP BY ?npc ?npcLabel
ORDER BY ?npcLabel
```

Διάγραμμα 45 – SPARQL Query

npc	npcLabel	taskCount
NPC Candy Fox	"NPC Candy Fox" http://www.w3.org/2001/XMLSchema#string	"5" http://www.w3.org/2001/XMLSchema#integer
NPC Candy Princess	"NPC Candy Princess" http://www.w3.org/2001/XMLSchema#string	"6" http://www.w3.org/2001/XMLSchema#integer
Narrator Completion	"Narrator Completion" http://www.w3.org/2001/XMLSchema#string	"1" http://www.w3.org/2001/XMLSchema#integer
Narrator Dialog	"Narrator Dialog" http://www.w3.org/2001/XMLSchema#string	"1" http://www.w3.org/2001/XMLSchema#integer
Narrator Guide	"Narrator Guide" http://www.w3.org/2001/XMLSchema#string	"1" http://www.w3.org/2001/XMLSchema#integer

Διάγραμμα 46 - SPARQL Query result

NPC Candy Fox: Όλα τα tasks του NPC και τι modality είναι

```
SPARQL query:

PREFIX sgo: <http://www.example.org/SGOntology/core#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?task ?taskLabel ?modality
WHERE {
  <http://www.example.org/SGOntology/data#NPC_Candy_Fox>
    sgo:hasTaskTemplate ?task .
  ?task sgo:usesModality ?modality .
  OPTIONAL { ?task rdfs:label ?taskLabel }
}
ORDER BY ?task
```

Διάγραμμα 47 - SPARQL Query

task	taskLabel	
A boy licking a big strawberry lollipop.	"A boy licking a big strawberry lollipop." ¹ <http://www.w3.org/2001/XMLSchema#string>	Modality_ImageToPhrase
Two friends sharing a chocolate bar at the candy shop.	"Two friends sharing a chocolate bar at the candy shop." ¹ <http://www.w3.org/2001/XMLSchema#string>	Modality_ImageToPhrase
Beginner: choose the phrase that best describes the image.	"Beginner: choose the phrase that best describes the image." ¹ <http://www.w3.org/2001/XMLSchema#string>	Modality_ImageToText
Beginner: choose the phrase that best describes the image.	"Beginner: choose the phrase that best describes the image." ¹ <http://www.w3.org/2001/XMLSchema#string>	Modality_ImageToText
Basketball	"Basketball" ¹ <http://www.w3.org/2001/XMLSchema#string>	Modality_WordToImage

Διάγραμμα 48 - SPARQL Query result

NPC Candy Fox: Όλα τα tasks του NPC, τα labels τους και σε ποιά difficulty εμφανίζονται αυτά τα tasks

```

PREFIX sgo: <http://www.example.org/SGOntology/core#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?task ?taskLabel ?profile ?level
WHERE {
  <http://www.example.org/SGOntology/data#NPC_Candy_Fox>
    sgo:hasTaskTemplate ?task .
  ?task sgo:initialDifficultyProfile ?profile .
  ?profile sgo:difficultyLevel ?level .
  OPTIONAL { ?task rdfs:label ?taskLabel }
}
ORDER BY ?level ?task

```

Διάγραμμα 49 - SPARQL Query

task	taskLabel	profile	level
A boy licking a big strawberry lollipop.	"A boy licking a big strawberry lollipop." ¹ <http://www.w3.org/2001/XMLSchema#string>	Difficulty_Beginner	"1" ¹ <http://www.w3.org/2001/XMLSchema#integer>
Two friends sharing a chocolate bar at the candy shop.	"Two friends sharing a chocolate bar at the candy shop." ¹ <http://www.w3.org/2001/XMLSchema#string>	Difficulty_Beginner	"1" ¹ <http://www.w3.org/2001/XMLSchema#integer>
Beginner: choose the phrase that best describes the image.	"Beginner: choose the phrase that best describes the image." ¹ <http://www.w3.org/2001/XMLSchema#string>	Difficulty_Beginner	"1" ¹ <http://www.w3.org/2001/XMLSchema#integer>
Beginner: choose the phrase that best describes the image.	"Beginner: choose the phrase that best describes the image." ¹ <http://www.w3.org/2001/XMLSchema#string>	Difficulty_Beginner	"1" ¹ <http://www.w3.org/2001/XMLSchema#integer>
Basketball	"Basketball" ¹ <http://www.w3.org/2001/XMLSchema#string>	Difficulty_Beginner	"1" ¹ <http://www.w3.org/2001/XMLSchema#integer>

Διάγραμμα 50 - SPARQL Query result

4.3.8 Export Turtle και τοποθέτηση σε StreamingAssets

Η οντολογία γίνεται export σε 2 .ttl αρχεία και αποθηκεύεται στο φάκελο StreamingAssets του Unity για να διαβαστεί και να μπορεί ο TaskManager να τα φορτώσει.

File → Save ή Export → Turtle (.ttl) και μετά κάνουμε αντίγραφα στο Unity:

Assets/StreamingAssets/SGOntology_core_v2.ttl και SGOntology_data_v2.ttl

Στο build, τα αρχεία αντιγράφονται στο StreamingAssets του executable και ο OntologyLoader τα διαβάζει στο runtime με Application.streamingAssetsPath.

4.3.9 Policies – ερμηνεία τιμών στο runtime

Policy_Increase: accuracy 80%-100%, hints 0-1, delta +1

Policy_Decrease: accuracy 70%-85%, hints 3-5, delta -1

Policy_Maintain: accuracy 70%-85%, hints 2-3, delta 0

Συγκρίνεται το μέσο accuracy και totalHints με τα διαστήματα που υπάρχουν στο TTL. Το πρώτο policy που ταιριάζει «κερδίζει» (Βλέπε Διάγραμμα 52) . Επίσης υπάρχει fallback από URI substring αν αποτύχει το parsing των delta και τα delta δίνονται απο κώδικα στο Unity (Βλέπε Διάγραμμα 53) , λόγο προβλημάτων που εμφανίστηκαν στο parsing.

```
:Policy_Decrease ...
    sgo:policyAccuracyFloor 0.70 ;
    sgo:policyAccuracyCeiling 0.85 ;
...
:Policy_Increase ...
    sgo:policyAccuracyFloor 0.80 ;
...
:Policy_Maintain ...
    sgo:policyAccuracyFloor 0.70 ;
```

Διάγραμμα 51 - floor/ceilings from ontology ttl

```
DifficultyPolicy matchingPolicy = null;
Debug.Log($"[DifficultyAdjuster] Sequence summary: AvgAccuracy={averageAccuracy}, TotalHints={totalHints}.
foreach (var policy in allPolicies)
{
    bool accuracyMatch = averageAccuracy >= policy.PolicyAccuracyFloor &&
        averageAccuracy <= policy.PolicyAccuracyCeiling;
    GetEffectiveHintRange(policy, out int minHints, out int maxHints);
    bool hintMatch = totalHints >= minHints &&
        totalHints <= maxHints;
    ...
    if (accuracyMatch && hintMatch)
    {
        matchingPolicy = policy;
        ...
        break;
    }
}

if (matchingPolicy == null)
{
    Debug.Log($"[DifficultyAdjuster] No policy matched. Difficulty unchanged.");
    return player.DifficultyProfileUri;
}
```

Διάγραμμα 52 – Difficulty policies

```
private static int GetEffectivePolicyDelta(DifficultyPolicy policy)
{
    if (policy == null) return 0;
    if (policy.PolicyDelta != 0) return policy.PolicyDelta;
    if (string.IsNullOrEmpty(policy.Uri)) return 0;
    if (policy.Uri.IndexOf("Increase", System.StringComparison.OrdinalIgnoreCase) >= 0) return 1;
    if (policy.Uri.IndexOf("Decrease", System.StringComparison.OrdinalIgnoreCase) >= 0) return -1;
    return 0;
}
```

Διάγραμμα 53 – Parsing fail

Σημείωση: *Known issue/bug, για accuracy < 70% ή hints > 5 κανένα policy δεν κάνει match και το difficulty παραμένει όπως ήταν*

4.3.10 Τα τρία NPCs και σειρά περιεχομένου

Στο παιχνίδι πρέπει να ακολουθείται μία σειρά για να μπορέσει ο παίκτης να προχωρήσει στο παιχνίδι. Οι επισκέψεις του στα NPC είναι οι εξής:

NPC_Candy_Fox → NPC_VendingMachine → NPC_CandyPrincess

Αν ο παίκτης αποφασίσει να αγνοήσει την σειρά τότε δεν θα μπορεί να προχωρήσει στο παιχνίδι και θα λαμβάνει διαλόγους απο τα NPCs που θα του λενε πως δεν έχει έρθει ακόμη η σειρά τους ή οτι εχουν ήδη ολοκληρωθεί αν γίνει ξανά επίσκεψη τους απο τον παίκτη μετά την ολοκλήρωσή τους.

Επίσης υπάρχει μια σειρά απο milestones όπου γίνεται trigger και εμφανίζεται ο narrator για να καθοδηγήσει τον παίκτη στο πώς και πού να προχωρήσει μέσω URIs.

Η διαδικασία αυτή είναι one-time follow-up, δηλαδή αν εμφανιστεί ο narrator και ο παίκτης τον αγνοήσει ή κλείσει το παράθυρο, τότε δεν θα ξανά εμφανιστεί.

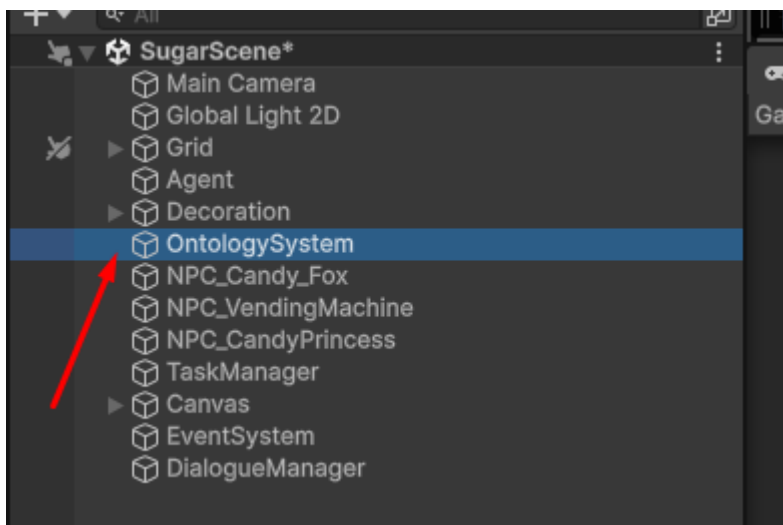
4.4 Ενσωμάτωση Οντολογίας στο Unity

Τα αρχεία .ttl τοποθετήθηκαν στον φάκελο StreamingAssets της Unity. Ο TaskManager τα φορτώνει κατά την εκκίνηση, δημιουργώντας dictionary URI → TaskData. Κάθε NPC φέρει URI που αντιστοιχεί σε instance της οντολογίας. Κατά την αλληλεπίδραση

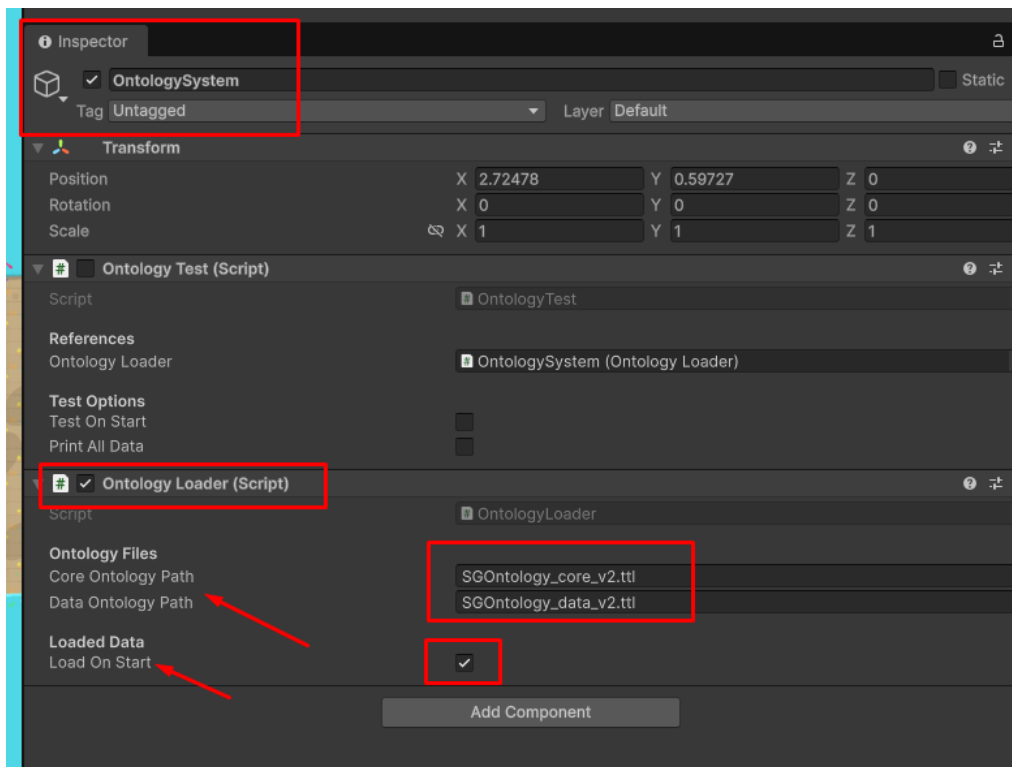
με NPC, το σύστημα ανακτά τα tasks μέσω οντολογίας και φορτώνει το περιεχόμενο από τα TaskData assets.

4.4.1 Ρύθμιση Unity: **OntologyLoader** και **TaskManager** στο Scene

Στη σκηνή του παιχνιδιού στο Unity (Sugar Scene) δημιουργήθηκε ένα GameObject που ονομάστηκε **OntologySystem** και περιέχει στο inspector window του σαν component το **OntologyLoader** script έτσι ώστε να μπορεί να διαβάζει τα αρχεία .ttl που έχουμε δημιουργήσει και να τα χρησιμοποιεί στο Unity. Στο inspector υπάρχουν fields όπου προστίθενται τα αρχεία .ttl και η επιλογή Load On Start (Βλέπε Διάγραμμα 54 και 55)



Διάγραμμα 54 – OntologySystem Unity



Διάγραμμα 55 - Ontology System Inspector

Στο Play, το Awake του OntologyLoader καλεί την LoadOntology() πριν ο TaskManager ζητήσει tasks. (Βλέπε Διάγραμμα 56 και 57).

```
private void Awake()
{
    if (loadOnStart)
    {
        LoadOntology();
    }
}
```

Διάγραμμα 56 - load on start code

```

public void LoadOntology()
{
    parsedTriples.Clear();

    string corePath = GetFilePath(coreOntologyPath);
    if (File.Exists(corePath))
        LoadTTLFile(corePath, isCore: true);
    ...
    string dataPath = GetFilePath(dataOntologyPath);
    if (File.Exists(dataPath))
        LoadTTLFile(dataPath, isCore: false);
    ...

    MapTriplesToDomainObjects();
}

```

Διάγραμμα 57 - Load ttl files

Στο Unity όλα τα Awake τρέχουν πριν απο οποιοδήποτε start. Αρα όταν ο παίκτης ζητήσει tasks τα dictionaries είναι ήδη γεμάτα.

Ο TaskManager δεν καλεί την LoadOntology, απλά χρησιμοποιεί τον loader. (Βλέπε Διάγραμμα 58)

```

public List<TaskTemplate> GetTasksForNPC(string npcArchetypeUri)
{
    if (ontologyLoader == null)
    {
        Debug.LogWarning("[TaskManager] OntologyLoader not found.");
    }
}

```

Διάγραμμα 58 - Task manager looking for loader

4.4.2 Parse και MapTriplesToDomainObjects

Ο parser διαβάζει prefixes και triples (subject, predicate, object).

Η MapTriplesToDomainObjects() δημιουργεί αντικείμενα απο rdf:type όπως NPCArchetype , TaskTemplate κτλπ, και συμπληρώνει λίστες απο object properties (hasTaskTemplate → HasTaskTemplateUri και αποθηκεύει labels rdfs:label για την εμφάνιση τους στο UI.

4.4.3 TaskManager – runtime queries

Ο TaskManager είναι το κεντρικό component του Unity που συνδέει το gameplay με την οντολογία. Μετά τη φόρτωση των αρχείων .ttl από τον OntologyLoader, παίρνει για κάθε NPC τα διαθέσιμα TaskTemplate μέσω του hasTaskTemplate και επιλέγει αυτά που έχουν το ίδιο επίπεδο δυσκολίας με το τρέχον επίπεδο του παίκτη (initialDifficultyProfile – DifficultyProfileUri). Επίσης βλέπει ποια tasks έχουν ήδη ολοκληρωθεί μέσω του profile του παίκτη (CompletedTaskUris στο αρχείο JSON που κρατάει όλα τα profile και performances των παικτών) για να μην τα ξανά εμφανίσει.

Λειτουργεί ως singleton στη σκηνή, έτσι ώστε scripts όπως NPCInteractionHandler και UI να ζητούν εργασίες χωρίς να διαβάζουν απευθείας από τα αρχεία .ttl.

Η οντολογία παραμένει η «πηγή», ενώ η πρόοδος και η προσαρμογή δυσκολίας γίνονται στο runtime.

Singleton = όλα τα scripts καλούν «TaskManager.Instance» αντί να φορτώνουν την οντολογία από μόνοι τους

```
private void Awake()
{
    if (Instance != null && Instance != this)
    {
        Destroy(gameObject);
        return;
    }
    Instance = this;
}
```

Διάγραμμα 59 – Awake function

GetTasksForNPC → Βλέπει τι προσφέρει αυτό το NPC

- Βρίσκει το individual NPC (π.χ NPC_Candy_Fox) από τα δεδομένα που έφτασαν από το hasTaskTemplate στο TTL
- Για κάθε URI task, φέρνει το TaskTemplate (label, modality, difficulty profile, κτλπ)

- Επιστρέφει όλα τα tasks που προσφέρει το NPC

```
var npc = ontologyLoader.GetNPCArchetype(trimmedUri);
...
foreach (var taskUri in npc.HasTaskTemplateUris)
{
    var task = ontologyLoader.GetTaskTemplate(taskUri);
    if (task != null)
        tasks.Add(task);
}
return tasks;
```

Διάγραμμα 60 – Finds NPC & tasks

GetTasksForNPCAtDifficulty → Βλέπει ποια από αυτά τα tasks ταιριάζουν στο επίπεδο δυσκολίας του. Δηλαδή παίρνει τη λίστα απο το GetTasksForNPC και συγκρίνει το local name του προφιλ του παίκτη με του initialDifficultyProfile του κάθε task.

Αν κάνουν match, τότε τα κρατάει. Π.χ – Παίκτης με Difficulty_Beginner → Κρατάει μόνο tasks με initialDifficultyProfile → Difficulty_Beginner.

```
var allTasks = GetTasksForNPC(npcArchetypeUri);
...
string diffLocal = GetLocalName(playerDifficultyUri);
foreach (var task in allTasks)
{
    string taskDiffLocal = GetLocalName(task.InitialDifficultyProfileUri);
    if (string.Equals(taskDiffLocal, diffLocal, System.StringComparison.OrdinalIgnoreCase))
        filtered.Add(task);
}
```

Διάγραμμα 61 – Finds tasks for NPC

HasCompletedTask → Βλέπει ποια tasks έχουν ήδη ολοκληρωθεί

Ψάχνει στη λίστα CompletedTaskUris του PlayerProfile που είναι αποθηκευμένο στο JSON αρχείο, το taskUri και απαντάει true ή false.

```
foreach (var completedUri in GameManager.Instance.CurrentPlayer.CompletedTaskUris)
{
    if (string.Equals(completedUri?.Trim(), trimmedUri, System.StringComparison.OrdinalIgnoreCase))
        return true;
}
return false;
```

Διάγραμμα 62 – Checks for completed tasks

Όταν ένα task ολοκληρωθεί προστίθεται το uri του στην λίστα

```
if (!alreadyCompleted)
{
    player.CompletedTaskUris.Add(normalizedUri);
}
```

Διάγραμμα 63 – Add URI in completedTask list

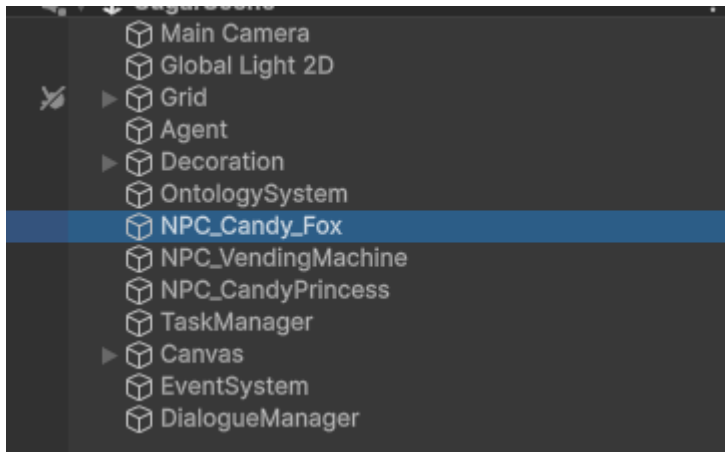
4.4.4 Σύνδεση URI: InteractiveNPC και TaskData

Η σύνδεση των URI της οντολογίας και του Unity είναι ίσως το σημαντικότερο κομμάτι στο μοντέλο. Όλα βασίζονται στο αν τα URI κάνουν match μεταξύ τους αλλιώς δεν δουλεύει τίποτα ή δουλεύει λάθος.

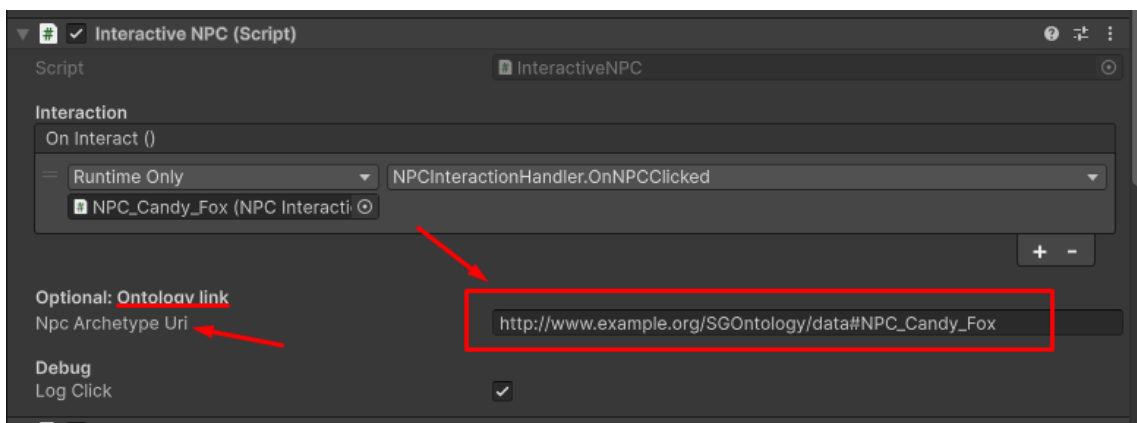
InteractiveNPC.npcArchetypeUri = ακριβώς το URI του individual στο data.ttl

InteractiveNPC.npcArchetypeUri→

http://www.example.org/SGOntology/data#NPC_Candy_Fox (Βλέπε Διάγραμμα 64 και 65)



Διάγραμμα 64 – NPC_Candy_Fox in Unity



Διάγραμμα 65 - URI Match

4.4.5 TaskData: Δομή και σχέση με την οντολογία

Η οντολογία ορίζει τι είδους task είναι (modality, δυσκολία, σύνδεση με NPC) και ποιο task τρέχει (URI), αλλά δεν μπορεί να κρατήσει sprites, λίστες εικόνων ή κείμενα κουμπιών.

Το TaskData είναι ScriptableObject στο Unity και λειτουργεί ως ένας πίνακας αντιστοίχισης. Κάθε γραμμή (TaskDataEntry) δένεται στο ίδιο URI με το αντίστοιχο TaskTemplate individual (taskUri ↔ task.Uri μέσω του TryGetEntry). Έτσι το σύστημα παραμένει ontology-driven στη λογική, ενώ το περιεχόμενο παραμένει εκεί που ανήκει (assets του editor). Βασικά αποφασίζει το οπτικό και λεκτικό περιεχόμενο για τα tasks όπως εικόνες επιλογής, φράσεις που χρησιμοποιούνται, εικόνες που πρέπει να περιγράψει ο παίκτης (stimulusImage) και τα hints (τι είδος hint, με ποια σειρά να εμφανίζονται κτλ.).

Χωρίς TaskData η οντολογία θα ήταν πλήρης σημασιολογικά αλλά άδεια οπτικά.

Inspector
Task Data (Task Data)

Script: TaskData

Task entries
Choose a task type for each row. Only the fields used for that type are shown. Other data stays saved but hidden. Runtime still follows the ontology modality.

Entry 0: CandyFox_Task1_Beginner

Task URI: http://www.example.org/SGOntology/data#CandyFox_Task1_Beginner

Task type: Image → phrase

Fields (image → phrase)

Images (4)

- Element 0: beginner_1_wrong2_0
- Element 1: beginner_1_wrong_0
- Element 2: beginner_1_wrong3_0
- Element 3: beginner_1_correct_0

Correct Image Index: 3

Hints (3)

Element 0

Hint Type: Highlight Key Words

Key Words (3)

- Element 0: boy
- Element 1: strawberry
- Element 2: lollipop

Dim Count: 0

Text Clue:

Element 1

Element 2

Διάγραμμα 66 - TaskData

Entry 2: CandyFox_Task3_Beginner

Task URI: http://www.example.org/SGOntology/data#CandyFox_Task3_Beginner

Task type: Image → text (pick phrase)

Fields (image → text)

Stimulus Image: Main Menu Image_0

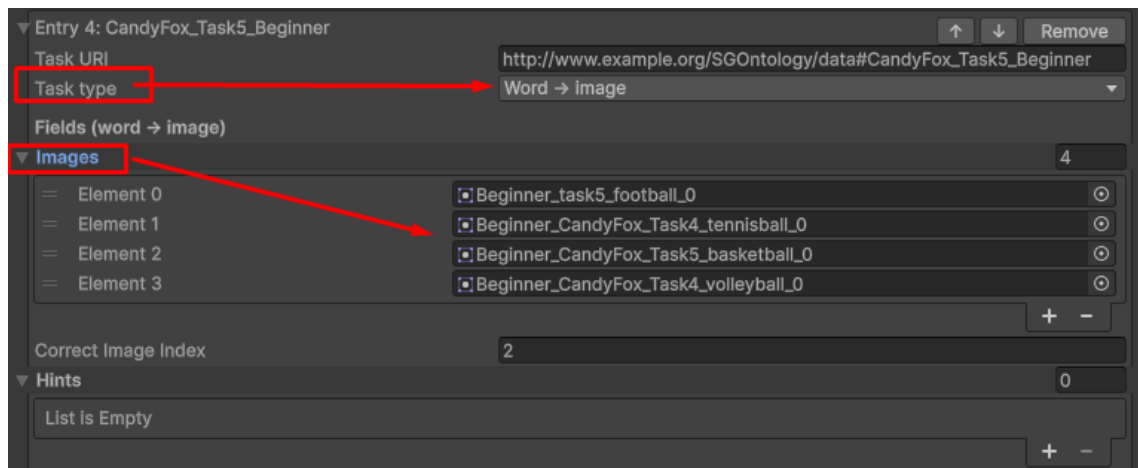
Choice Phrases (4)

- Element 0: A colourful playground with candy lollipops.
- Element 1: A beach full of chocolate bars
- Element 2: A colourful playground with chocolate all over it
- Element 3: A beach with candy lollipops

Correct Choice Index: 0

Hints: 0

Διάγραμμα 67 - TaskData Image-to-text



Διάγραμμα 68 - TaskData - Word to Image

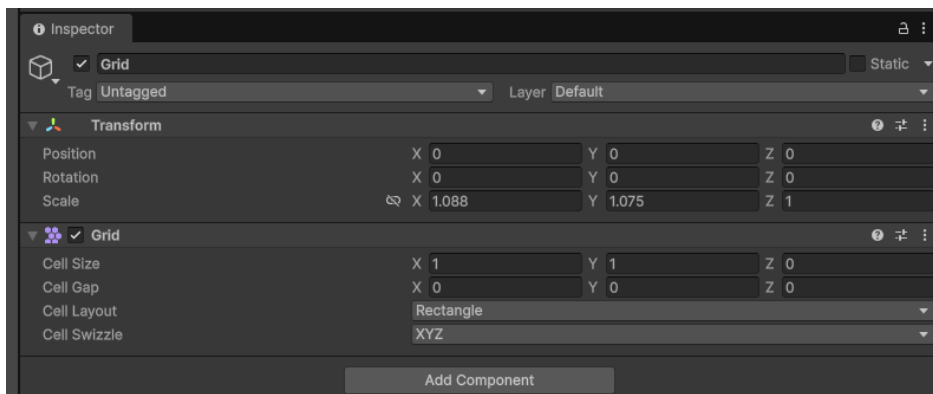
4.4.6 Δημιουργία σκηνής (Περιβάλλον)

Αρχικά δεν θα υπήρχε παιχνίδι χωρίς τη δημιουργία του οπτικού περιβάλλοντος της σκηνής. Το περιβάλλον χτίστηκε στο Editor του Unity με 2D Tilemap και διακοσμητικά sprites τα οποία έγιναν download απο το Unity Store.

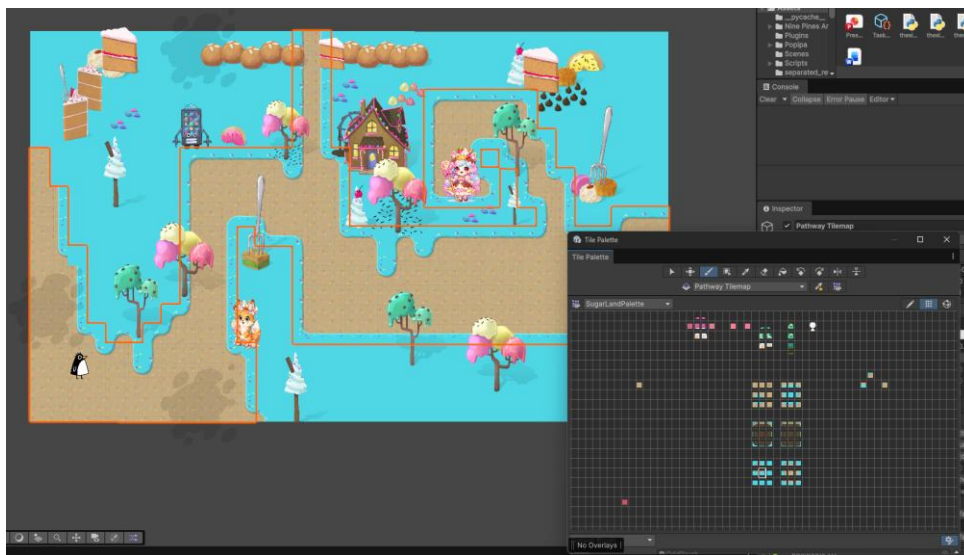
Βρέθηκαν (tiles - μικρά τετραγωνάκια γραφικών όπου αποτέλεσαν το «πάτωμα» της σκηνής όπως και διακοσμητικά sprites και agents) που χρησιμοποιήθηκαν για την δημιουργία του tilemap του παιχνιδιού και ως NPCs.

Τα συγκεκριμένα tiles και sprites προστέθηκαν σε ένα tile palette στο Unity και ξεκίνησε το χτίσιμο του tilemap πάνω σε ένα grid. Πρώτα έγινε η δημιουργία του grid (Βλέπε Διάγραμμα 69 και 70) . Το tilemap είναι πολλά μικρά τετραγωνάκια (tiles), όχι ένα ενιαίο background.

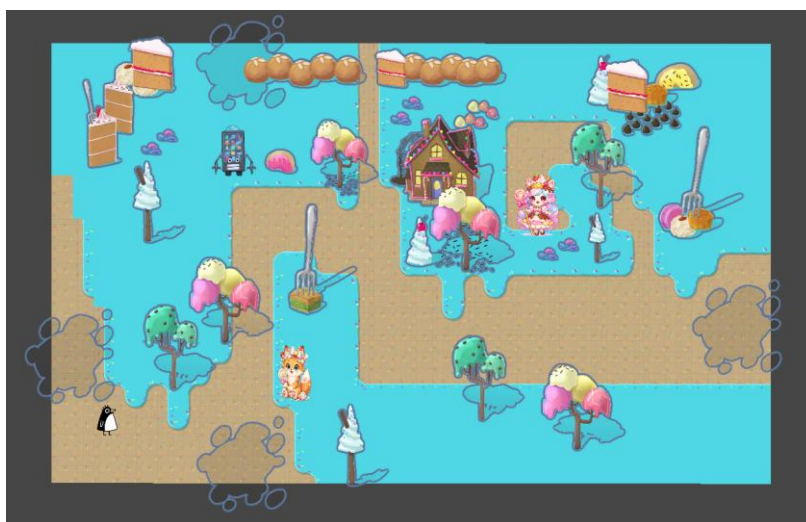
Τα διακοσμητικά sprites χρησιμοποιήθηκαν μετά τη δημιουργία του tilemap για decoration της σκηνής.(Βλέπε Διάγραμμα 71)



Διάγραμμα 69 - Unity Grid



Διάγραμμα 70 - Tile Palette



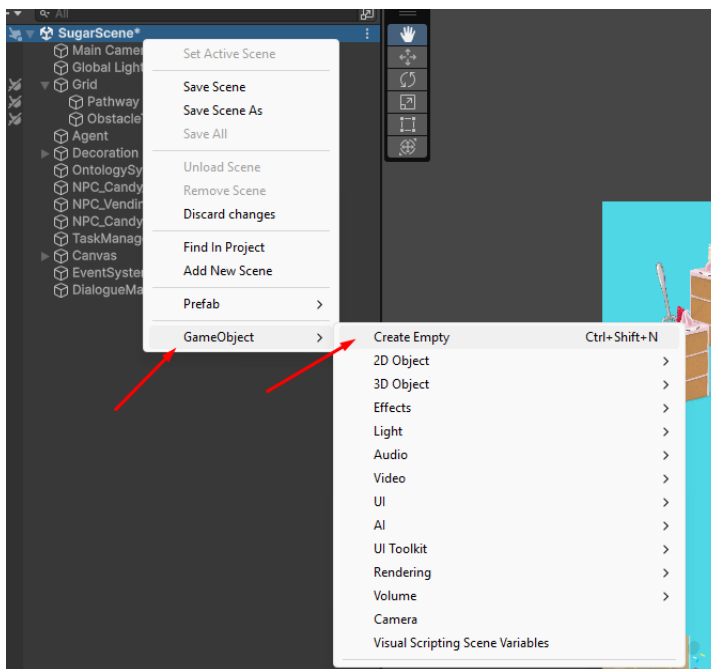
Διάγραμμα 71 - Decorations

4.4.7 Πρόσθεση GameObjects, Panels, Text, Images και components στο Unity (Πρακτικό).

4.4.7.1 Δημιουργία GameObject

Για τη δημιουργία GameObjects (π.χ OntologySystem, TaskManager κτλπ), πρέπει να ακολουθηθούν τα βήματα:

Δεξί κλικ στη σκηνή SugarScene → GameObject → Create Empty (Βλέπε Διάγραμμα 72)



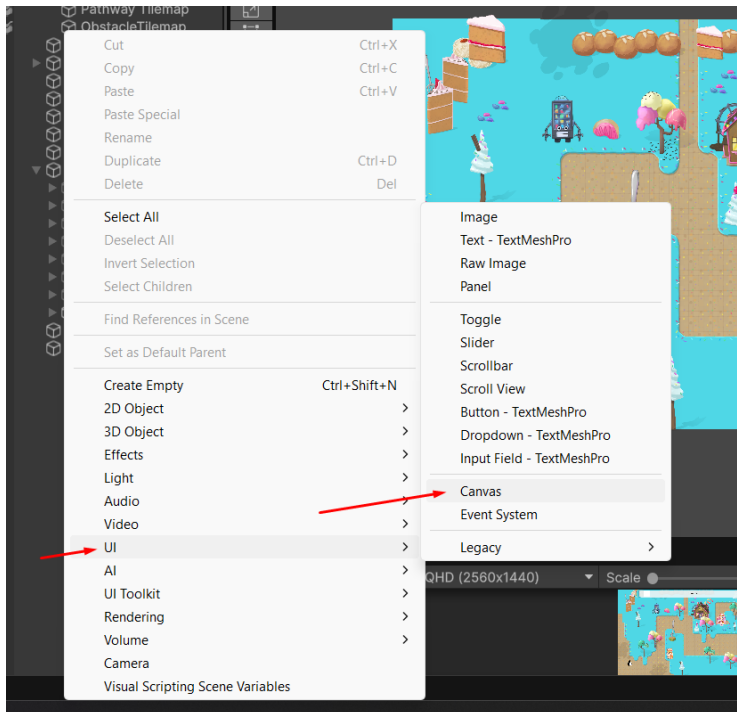
Διάγραμμα 72 - Create GameObject

4.4.7.2 Δημιουργία Canvas & Panel

Για τη δημιουργία Panel (π.χ Panel Phrase to Image, Reward Panel, Task Complete panel) πρέπει να ακολουθηθούν τα βήματα:

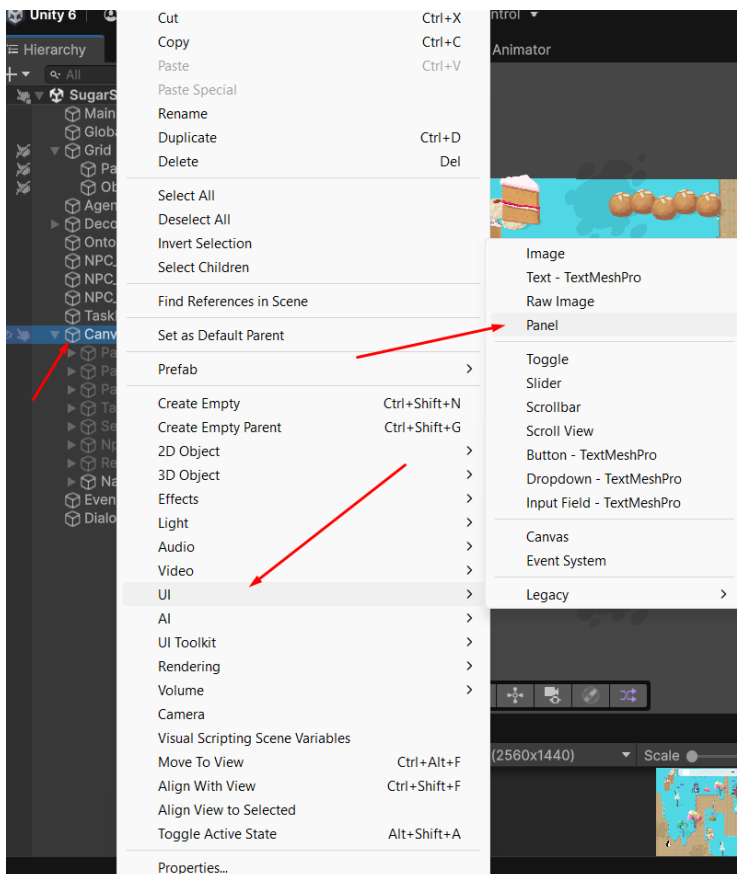
Δημιουργία ενός UI Canvas → Δημιουργία Panel μέσα στον canvas

Πως δημιουργούμε UI Canvas: Δεξί κλικ στο hierarchy → UI → Canvas (Διάγραμμα 73)



Διάγραμμα 73 - Create Canvas

Πως δημιουργούμε Panel: Δεξι κλικ στο Canvas → UI → Panel (Βλέπε Διάγραμμα 74)



Διάγραμμα 74 - Create Panel

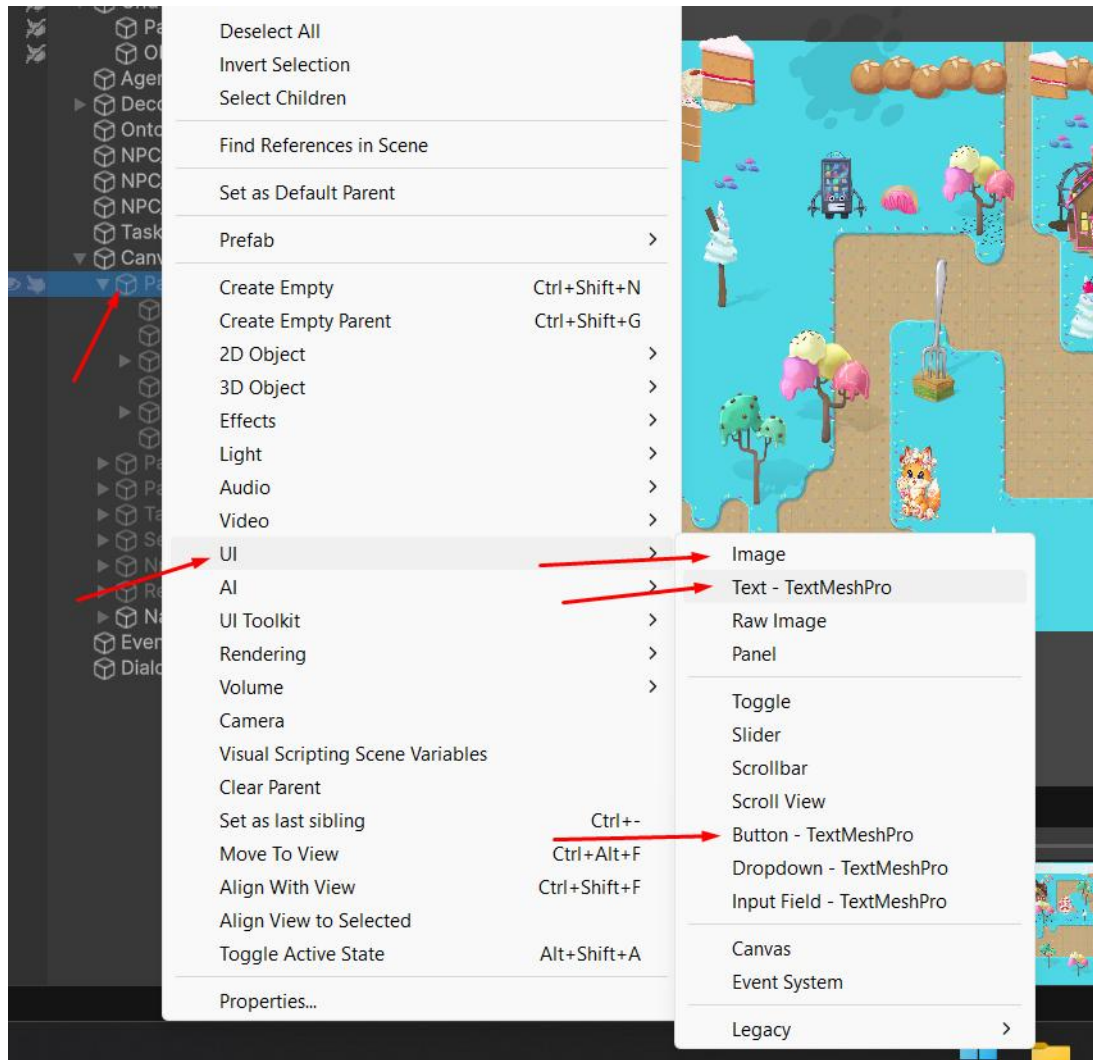
4.4.7.3 Προσθήκη εικόνας, κουμπιού και text στο panel

Για να προσθέσουμε εικόνες, κουμπιά και text στο panel που δημιουργήσαμε ακολουθούμε τα εξής βήματα: (Βλέπε Διάγραμμα 75)

Δεξί κλικ στο panel → UI → Image

Δεξί κλικ στο panel → UI → Text - TextMeshPro

Δεξί κλικ στο panel → UI → Button – TextMeshPro

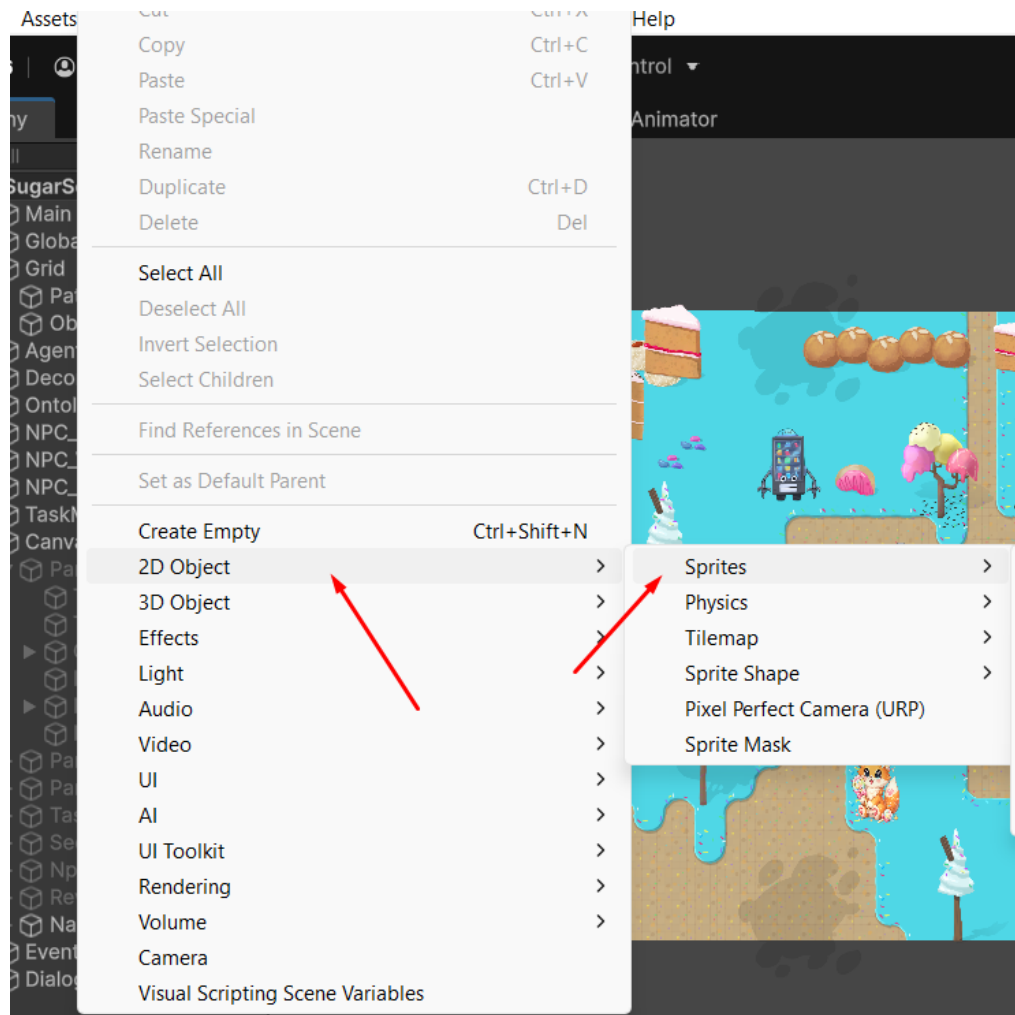


Διάγραμμα 75 – Create Panel

4.4.7.4 Προσθήκη Agent και NPC

Για να προσθέσουμε Agent και NPC στη σκηνή, ακολουθούμε τα εξής βήματα:

Δεξί κλικ στο hierarchy → 2D Object → Sprites (Βλέπε Διάγραμμα 76)

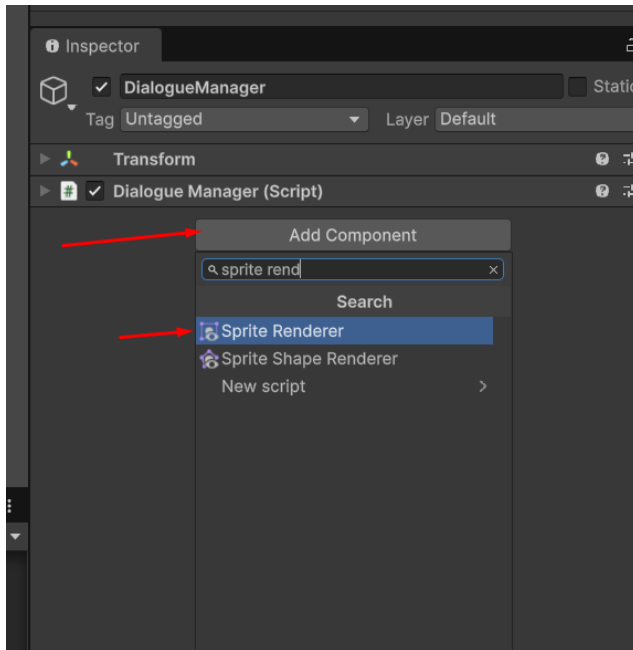


Διάγραμμα 76 - Create Sprites

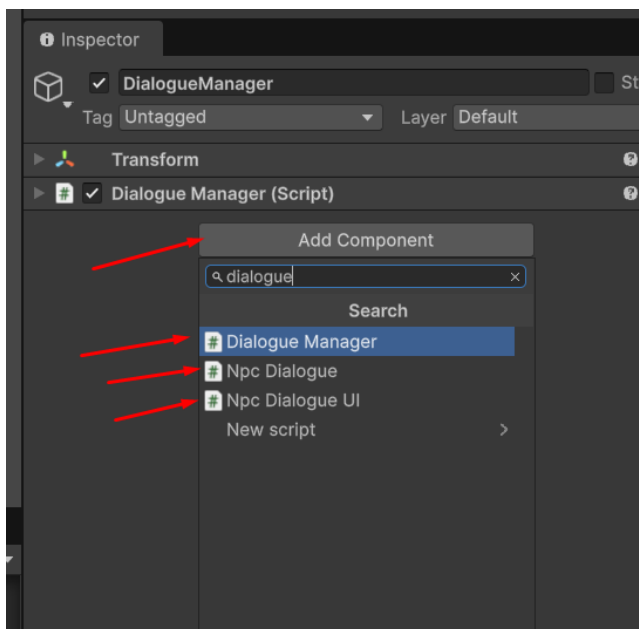
4.4.7.5 Προσθήκη components στο inspector

Τα components είναι ένα πάρα πολύ σημαντικό κομμάτι στη δημιουργία των παραπάνω (Game Objects, NPCs, Agents, Panels, Text, Images) αφού εδώ ερχόμαστε και προσθέτουμε όλα τα scripts που έχουμε δημιουργήσει για να δουλεύουν στο παιχνίδι, αλλά και έτοιμα components που προσφέρει το Unity που μας βοηθάνε στην ομάλη και σωστή λειτουργία τους. (π.χ Sprite Renderer , Box Collider 2D)

Στο inspector βρίσκουμε ένα κουμπί «Add Component» που μας δίνει τη δυνατότητα να προσθέσουμε components είτε έτοιμα απο το Unity ή δικά μας scripts. (Βλέπε Διάγραμμα 77 και 78)

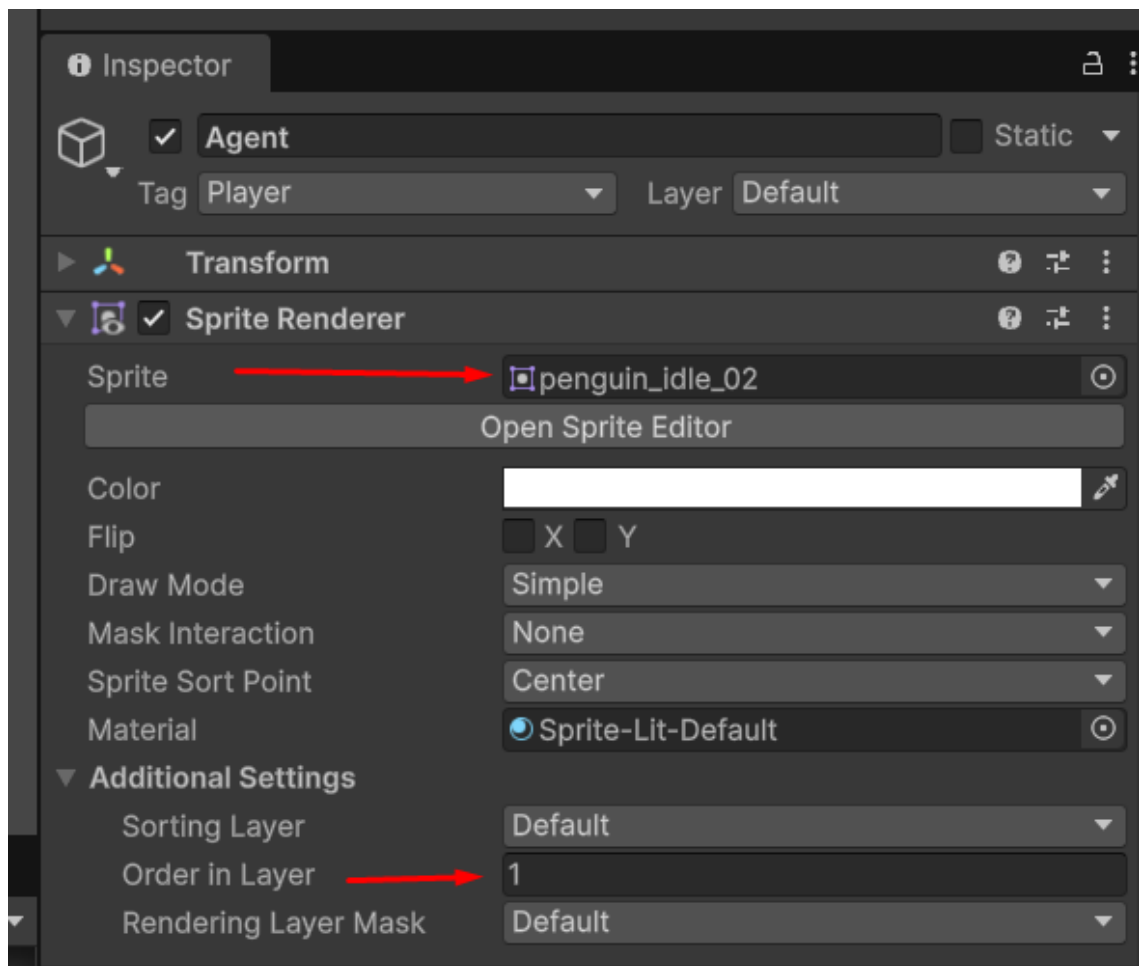


Διάγραμμα 77 - Pre-made Components



Διάγραμμα 78 - Script Components

Παράδειγμα component → Sprite Renderer



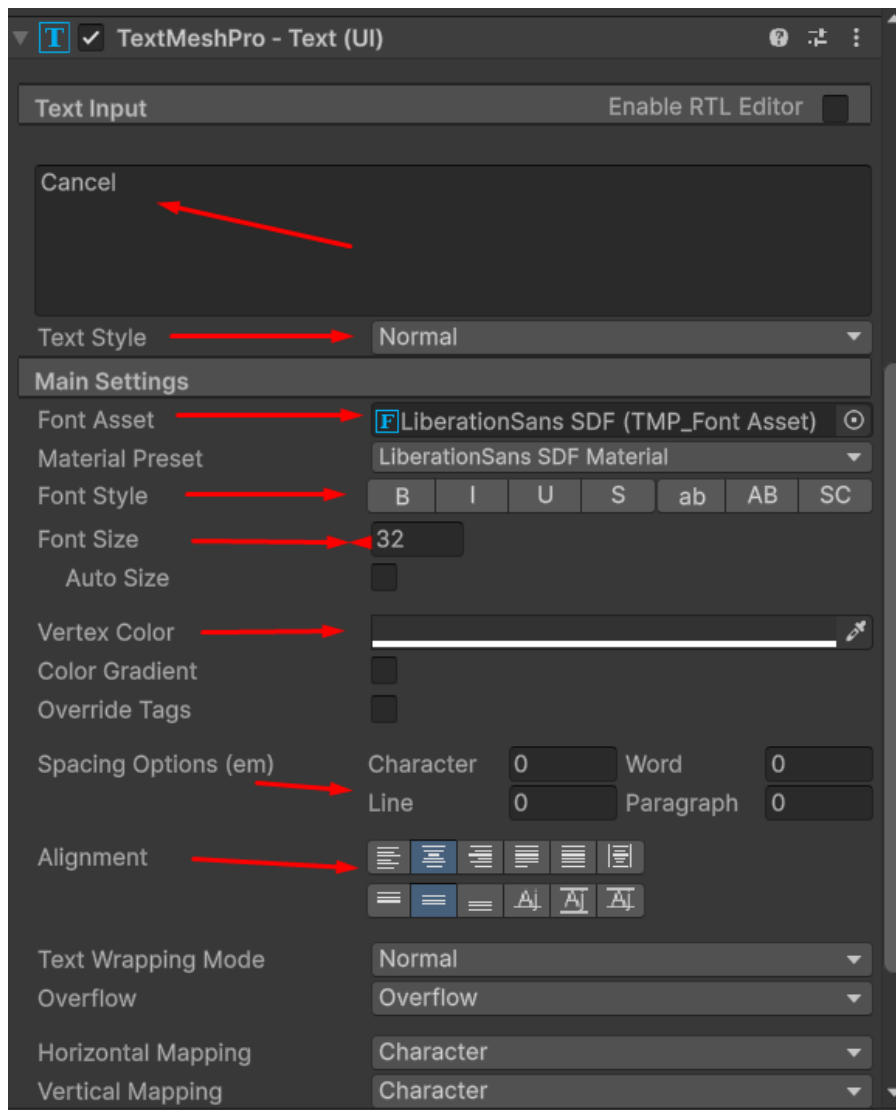
Διάγραμμα 79 - Sprite Renderer

Το Sprite Renderer μας δίνει τη δυνατότητα να προσθέσουμε το sprite (εμφάνιση του agent) με μόνο ένα κουμπί όπως και το order του agent in Layer (π.χ αν ένα δέντρο που χρησιμοποιήθηκε με layer 0 αλλά ο agent με layer 1, τότε ο agent μπορεί να κάνει overlap το δέντρο όταν περπατήσει απο πάνω του (αν δεν υπάρχει barrier). Αυτό μας βοηθάει πολύ στην κατασκευή και decoration της σκηνής και ποιά sprites μπορούν να κάνουν overlap άλλα.

Παράδειγμα component → TextMeshPro – Text (UI)

Με το TextMeshPro – Text (UI) μας δίνεται η δυνατότητα να προσθέσουμε text στα panels μας ή σε κουμπιά, να ρυθμίζουμε μέγεθος γραμμάτων, text style, γραμματοσειρές, χρώμα γραμμάτων, spacings, alignment κτλπ.

Είναι ένα πολυ σημαντικό εργαλείο για panels και διαλόγους. (Βλέπε Διάγραμμα 80)



Διάγραμμα 80 - TextMeshPro

4.5 Σύστημα Εισόδου και Αποθήκευση Προφίλ

Το σύστημα εισόδου ξεκινάει από το Main Menu όπου υλοποιείται από το MainMenuController. Ο παίκτης εισάγει όνομα (nameInput) και κωδικό (passwordInput) για να δημιουργήσει profile ή να συνδεθεί στο παιχνίδι αν έχει ήδη έτοιμο profile. Η μέθοδος OnStartGame() είναι το κεντρικό σημείο ελέγχου των στοιχείων και της δημιουργίας/αποθήκευσης των profile.

4.5.1 Login – πλήρης ανάλυση και κώδικας

Στο Main Menu ο παίκτης εισάγει τα στοιχεία του για να συνδεθεί στο profile του (Όνομα και Password) και πατάει το κουμπί Start Game για να συνδεθεί και να

προχωρήσει στο παιχνίδι. Πριν ο παίκτης περάσει στη σκηνή του παιχνιδιού γίνονται οι ελέγχοι (1) PasswordInput Missing, (2) NameInput Missing, (3) password match (hash), (4) GameManager.Instance, (5) GetByName case-insensitive.

Αν ο παίκτης δεν έχει profile και συμπληρώσει τα name και password inputs, τότε δημιουργείται το profile του παίκτη και αποθηκεύεται στο JSON αρχείο “PlayerProfile” ως εξής (UUID, Όνομα, URI Player_ «ονομα», Difficulty_Beginner, LastSceneName = firstSceneName, hashed password). (Βλέπε εικόνα 84 και figure 92,93)

Αν όμως ο παίκτης έχει ήδη profile και βάλει τα σωστά στοιχεία θα προχωρήσει κανονικά με το profile του στο παιχνίδι (Βλέπε εικόνα 82,83 και figure 94). Αν όμως ο παίκτης έχει ήδη profile και βάλει λάθος κωδικό θα πάρει μήνυμα «wrong password» (Βλέπε εικόνα 82 και figure 59,60), αν αφήσει κενό το name input θα πάρει μήνυμα «Please enter your name» (Βλέπε εικόνα 18 και figure 91,92) , αν αφήσει κενό το password input θα πάρει μήνυμα «Enter your password». (Βλέπε εικόνα 82 και figure 89,90)

Αν τα στοιχεία του παίκτη είναι σωστά ή αν είναι η πρώτη του φορά στο παιχνίδι, τότε βάζει τον παίκτη στο GameManager.CurrentPlayer όπου διατηρείται με DontDestroyOnLoad (για να μην καταστραφεί το GameObject της παλιάς σκηνής (Main Menu) όταν φορτωθεί η καινούρια σκηνή), και φορτώνει τη τελευταία σκηνή του παίκτη (LastSceneName) ή την πρώτη σκηνή αν είναι η πρώτη του φορά (FirstSceneName – SugarScene). (Βλέπε εικόνα 82 και 85)

Οι κωδικοί αποθηκεύονται μόνο ως hash (PBKDF2-SHA256, 10000 iterations, 16-byte salt + 32-byte hash, Base64) και όχι σαν plain text στο JSON, για την προστασία των κωδικών. (Βλέπε εικόνα 86)

```

public void OnStartGame()
{
    string rawName = nameInput != null ? nameInput.text.Trim() : "";
    string password = passwordInput != null ? passwordInput.text : "";

    if (passwordInput == null)
    {
        Debug.LogWarning("[MainMenu] Password Input is not assigned. Assign it");
        SetMessage("Password field missing. Check setup.");
        return;
    }

    if (string.IsNullOrEmpty(rawName))
    {
        SetMessage("Please enter your name.");
        return;
    }
}

```

Διάγραμμα 81 – OnStartGame → Null name

```

    if (string.IsNullOrEmpty(password))
    {
        SetMessage("Enter your password.");
        return;
    }
    if (!PasswordHelper.VerifyPassword(password, existing.PasswordHash))
    {
        SetMessage("Wrong password.");
        return;
    }
    GameManager.Instance.CurrentPlayer = existing;
    SetMessage("Welcome back, " + existing.Name + "!");
    string scene = string.IsNullOrEmpty(existing.LastSceneName) ? firstSceneName : existing.LastSceneName;
    SceneManager.LoadScene(scene);
    return;
}

```

Διάγραμμα 82 - OnStartGame → Null password & wrong password

```

var existing = PlayerProfileManager.GetByName(rawName);

```

Διάγραμμα 83 - OnStartGame → if profile exists

```

string safeName = rawName.Replace(" ", "_");
string playerUri = "http://www.example.org/SGOntology/data#Player_" + safeName;
var profile = new PlayerProfile
{
    Id = Guid.NewGuid().ToString(),
    Name = rawName,
    Uri = playerUri,
    DifficultyProfileUri = "http://www.example.org/SGOntology/data#Difficulty_Beginner",
    LastSceneName = firstSceneName,
    PasswordHash = PasswordHelper.HashPassword(password)
};

```

Διάγραμμα 84 – Player profile data

```

    GameManager.Instance.CurrentPlayer = profile;
    SetMessage("Profile created. Starting game...");
    SceneManager.LoadScene(firstSceneName);
}

```

Διάγραμμα 85 – Profile creation

```

public static string HashPassword(string password)
{
    if (string.IsNullOrEmpty(password))
        return "";

    byte[] salt = new byte[SaltSize];
    using (var rng = RandomNumberGenerator.Create())
    {
        rng.GetBytes(salt);
    }

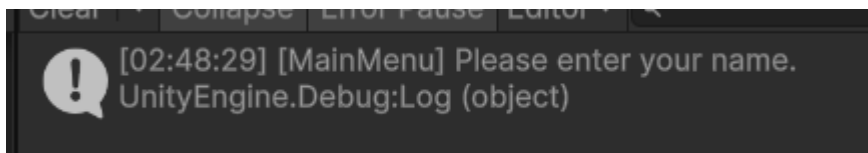
    byte[] hash = Pbkdf2(Encoding.UTF8.GetBytes(password), salt, Iterations, HashSize);
    byte[] combined = new byte[SaltSize + HashSize];
    Buffer.BlockCopy(salt, 0, combined, 0, SaltSize);
    Buffer.BlockCopy(hash, 0, combined, SaltSize, HashSize);
    return Convert.ToBase64String(combined);
}

```

Διάγραμμα 86 – Password Hash



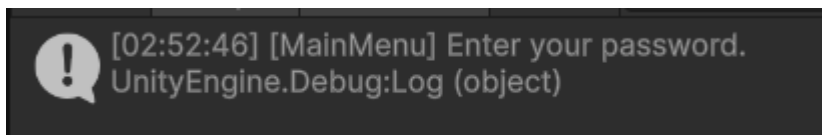
Διάγραμμα 87 - Login without name



Διάγραμμα 88 - Login without name debug



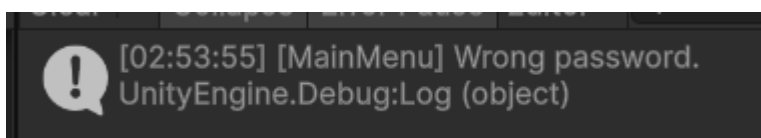
Διάγραμμα 89 - Login without password



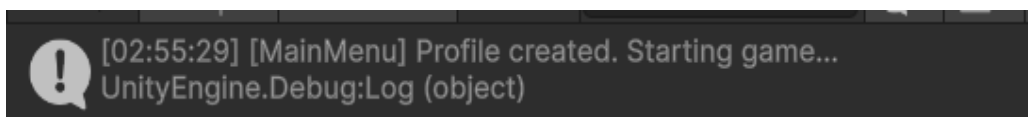
Διάγραμμα 90 - Login without password debug



Διάγραμμα 91 - Login with wrong password



Διάγραμμα 92 - Login with wrong password debug



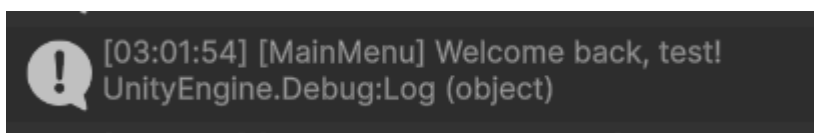
Διάγραμμα 93 - Profile Creation debug message

```

"profiles": [
  {
    "Id": "aa4ee7be-9578-45fc-90d7-c3f90d44151c",
    "Name": "test",
    "Uri": "http://www.example.org/SGOntology/data#Player_test",
    "DifficultyProfileUri": "http://www.example.org/SGOntology/data#Difficulty_Beginner",
    "LastSceneName": "SugarScene",
    "PasswordHash": "mcyNn5MBFWGMxzsi+sJFfUv+3517u06y0APv80C6i9F+qYypMzLBJj42GyqIyR04",
    "CompletedTaskUris": [
      "narrator://seen/Narrator_Welcome_Seen"
    ],
    "PerformanceHistory": [],
    "NpcRewardsEarned": 0,
    "SceneRewardsEarned": 0
  },
],

```

Διάγραμμα 94 - Profile Created in JSON file



Διάγραμμα 95 - Welcome back debug message

4.5.2 PlayerProfileManager και JSON

Τα runtime δεδομένα δεν μπαίνουν στην οντολογία αλλά στο JSON αρχείο PlayerProfiles.json όπου περιέχει πεδία όπως Id, Name, Uri, DifficultyProfileUri, LastSceneName, PasswordHash, CompletedTaskUris, PerformanceHistory, NpcRewardsEarned και SceneRewardsEarned.

Τα CompletedTaskUris χρησιμοποιούνται για να μὴν επαναλαμβάνονται tasks ή narrator dialogues (markers) που ἤδη ἔχει ολοκληρώσει ο παίκτης

Το PlayerProfileManager φορτώνει τα προφίλ από PlayerProfiles.json (LoadAll με in-memory cache) και ενημερώνει εγγραφές με βάση το UUID (UpdateProfile → SaveAll), διατηρώντας την πρόοδο, δυσκολία και το ιστορικό επιδόσεων εκτός της οντολογίας. (Βλέπε Διάγραμμα 96 και 97)

```

public static List<PlayerProfile> LoadAll()
{
    if (_cached != null)
        return _cached;
    ...
    string json = File.ReadAllText(FilePath);
    var wrapper = JsonUtility.FromJson<ProfileListWrapper>(json);
    if (wrapper?.profiles != null)
        _cached = wrapper.profiles;
    ...
    return _cached;
}

```

Διάγραμμα 96 – Player Profile LoadAll()

```

public static void UpdateProfile(PlayerProfile profile)
{
    if (profile == null || string.IsNullOrEmpty(profile.Id))
        return;
    var all = LoadAll();
    int i = all.FindIndex(p => p.Id == profile.Id);
    if (i >= 0)
        all[i] = profile;
    else
        all.Add(profile);
    SaveAll(all);
}

```

Διάγραμμα 97 – Update Profile

4.6 Κίνηση Παίκτη (PlayerMovement)

Ο παίκτης κινείται με δεξί κλικ. Το script PlayerMovement (MoveToMouse.cs) μετατρέπει τη θέση ποντικιού σε world coordinates και κινείται προς το mouse με σταθερή ταχύτητα. (Βλέπε Διάγραμμα 98)

Η επιλογή της κίνησης του παίκτη με απλά κλικ στο mouse έγινε για την ευκολία των χρηστών να μετακινούν τον παίκτη τους μόνο με ένα πάτημα κουμπιού σε αντίθεση με άλλες μεθόδους όπως WASD ή άλλων κουμπιών του πληκτρολογίου που μπορεί να ήταν πιο δύσκολο, ειδικά στα παιδιά.

```

using UnityEngine;
using UnityEngine.InputSystem;

public class PlayerMovement : MonoBehaviour
{
    public float speed = 5f;
    private Vector3 target;
    private Mouse mouse;

    void Start()
    {
        target = transform.position;
        mouse = Mouse.current;
    }

    void Update()
    {
        if (mouse == null) return;

        // Right-click: set move target (anywhere, no restrictions)
        if (mouse.rightButton.wasPressedThisFrame)
        {
            Vector3 mousePosition = Camera.main.ScreenToWorldPoint(mouse.position.ReadValue());
            mousePosition.z = transform.position.z;
            target = mousePosition;
        }

        transform.position = Vector3.MoveTowards(transform.position, target, speed * Time.deltaTime);
    }
}

```

Διάγραμμα 98 - Player Movement

4.7 NPC, Πρόοδος και Διάλογοι

Όσο αφορά τα NPCs και την πρόοδο του παίκτη στο παιχνίδι, έχει υλοποιηθεί ένα progression line με τη σειρά που πρέπει να ολοκληρωθούν τα NPCs και τα tasks τους αλλά και να γίνουν trigger τα guides του narrator που βοηθούν στην ομαλή λειτουργία του παιχνιδιού και καθοδηγούν τον παίκτη.

Ο GameManager είναι υπεύθυνος για την τύρηση της σειράς και γενικά της ροής του παιχνιδιού. Αν ο παίκτης αγνοήσει τη σειρά και προσπαθήσει να μιλήσει με διαφορετικό NPC απο αυτο που πρέπει θα λάβει συγκεκριμένα μηνύματα που θα τον καθοδηγήσουν.

Οι διαλόγοι συντονίζονται μέσω του DialogueManager.

4.7.1.1 NPC Clicks, Proximity και Progression Order

Κάθε φορά που γίνεται κλικ σε ένα NPC τρέχει η παρακάτω αλυσίδα:

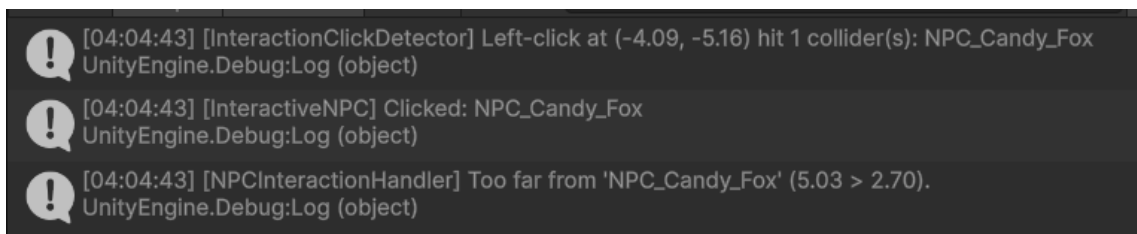
InteractionClickDetector → Physics2D.OverlapPointAll → InteractiveNPC → TriggerInteraction() → OnInteract → NPCInteractionHandler.OnNPCClicked().

Όπου γίνεται detect το αριστερό κλικ που έκανε ο παίκτης, μετατρέπεται η θέση του ποντικού σε world point και ελέγχονται 2D colliders, ελέγχετε αν εκεί που έκανε κλίπ ο παίκτης υπάρχει κάτι clickable (NPC), γίνεται trigger το interaction και τρέχει ο NPCInteractionHandler.OnNPCClicked() το οποίο φορτώνει tasks απο την οντολογία μέσω του TaskManager αλλά πριν ελέγχετε το proximity του παίκτη με το NPC (Πόσο μακριά/κοντά είναι ο παίκτης απο το NPC που έκανε κλικ), (Βλέπε εικόνα 27)

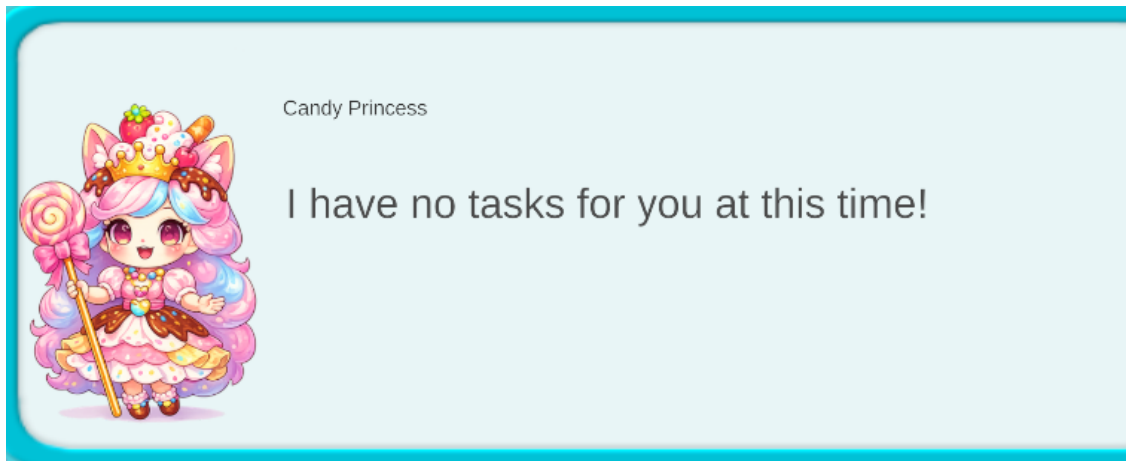
Αν όμως ο παίκτης είναι πιο μακριά απο το proximity limit του NPC, τότε δεν θα μπλοκάρει και θα επιστρέψει 0 tasks.

Επίσης γίνεται έλεγχος του progression order για να είμαστε σίγουροι οτι ο παίκτης προχωράει σωστά στο παιχνίδι με τα εξής: IsInteractionBlockedByNpcOrder() → GameManager.enforceNpcOrder + λίστα npcProgressionOrderUris (Candy Fox → Vending Machine → Candy Princess).

Αν γίνει κλικ σε NPC με λάθος σειρά τότε ο παίκτης θα λάβει μήνυμα απο το NPC «I have no tasks for you at this time!» ή «You have already completed my tasks!» (Βλέπε Διάγραμμα 100 και 101)



Διάγραμμα 99 - Click Detection & NPC click



Διάγραμμα 100 - Wrong NPC Order



Διάγραμμα 101 - NPC Already completed

4.7.2 Διαλόγοι NPC (DialogueManager, NpcDialogue)

Το κάθε NPC προσφέρει τους δικούς του intro και outro διαλόγους με click-to-advance στον επόμενο διάλογο.

ShowIntro → click-to-advance → Tasks → ShowOutro

```
public void OnNPCClicked() {
    if (!IsPlayerInRange()) return;
    if (IsInteractionBlockedByNpcOrder()) return;
    if (npcDialogue != null && DialogueManager.Instance != null &&
        npcDialogue.HasIntro)
        DialogueManager.Instance.ShowIntro(npcDialogue, StartTasksForNpc);
    else
        StartTasksForNpc();
}
```

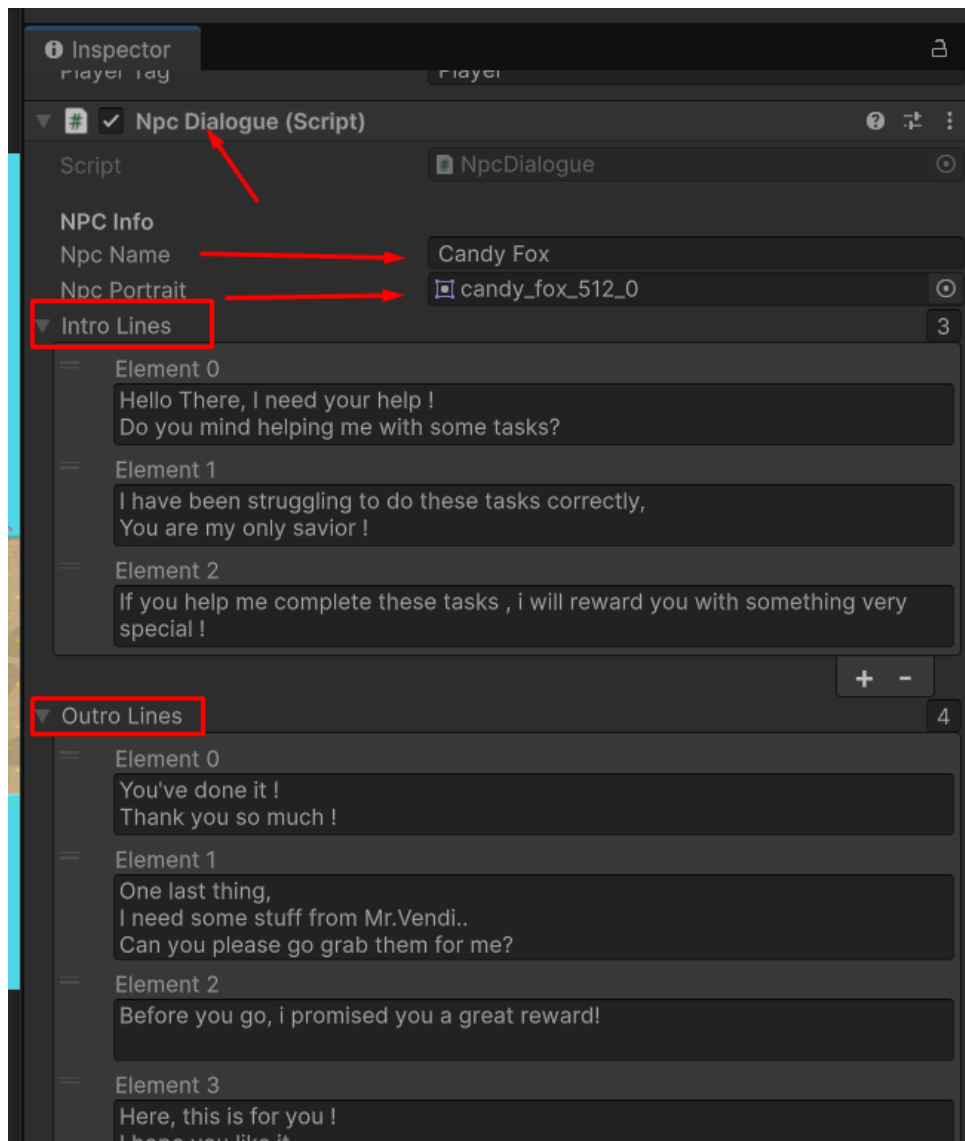
```

public void Advance() {
    _currentIndex++;
    if (_currentIndex >= _currentLines.Count) {
        dialoguePanel.SetActive(false);
        _onFinished?.Invoke();
    } else dialogueText.text = _currentLines[_currentIndex];
}

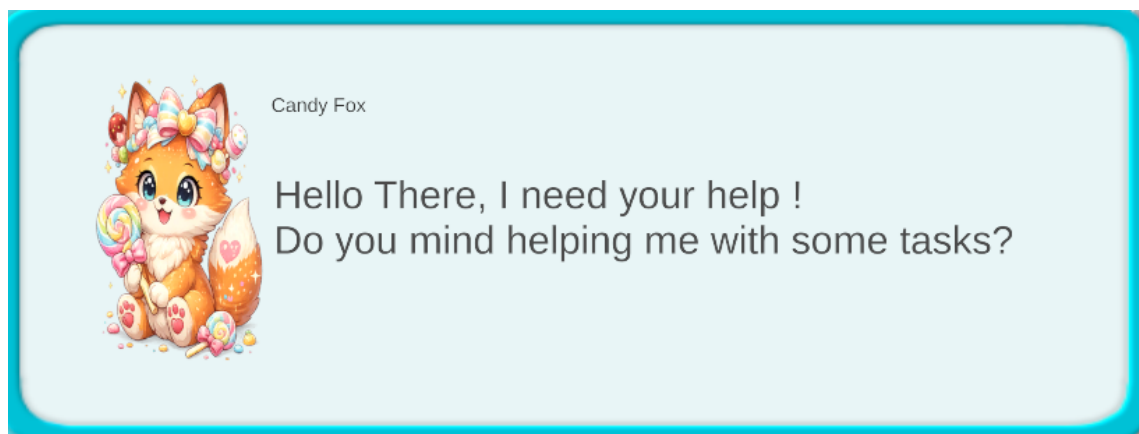
```

Οι διαλόγοι που χρησιμοποιούνται για κάθε npc, βρίσκονται στο Unity → NPC (π.χ Candy_Fox) στο component script «Npc Dialogue» με script «NpcDialogue» όπου εδώ προσθέτουμε το όνομα και το sprite του NPC που θα εμφανίζεται στον διάλογο αλλά και τα intro – outro dialogue lines με τη σειρά που θέλουμε να είναι (Βλέπε Διάγραμμα 102 και 103).

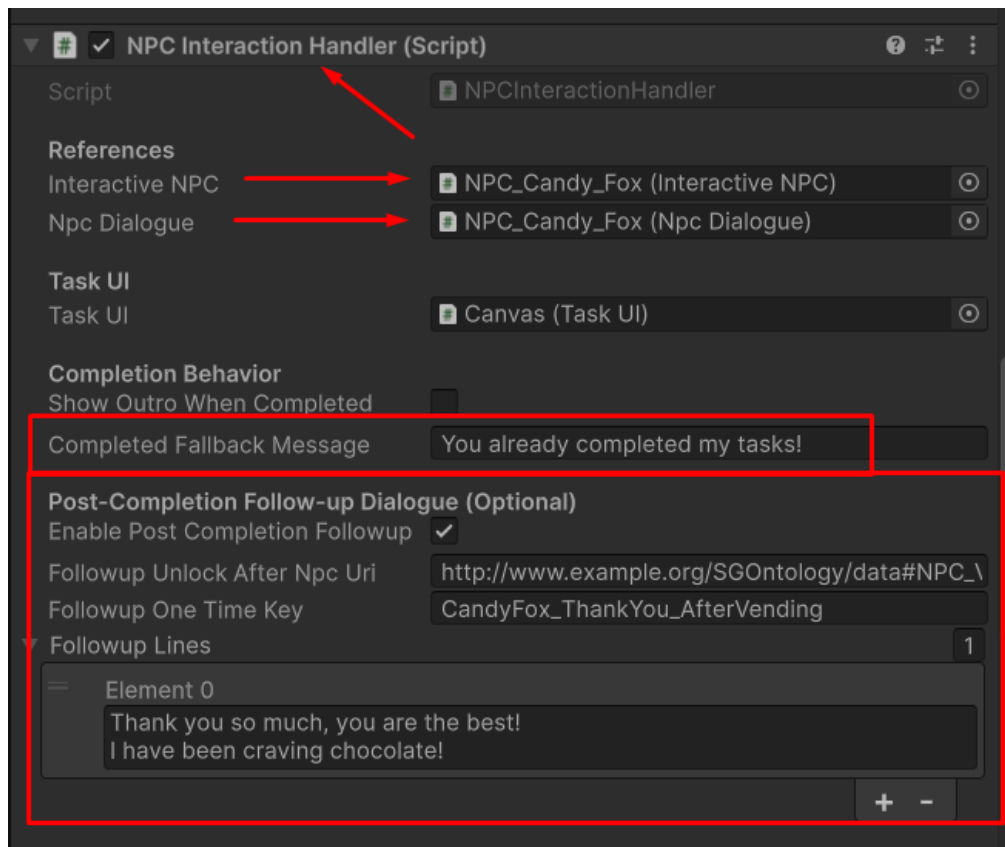
Οι διαλόγοι μετά την ολοκλήρωση των tasks του npc και οι followup διαλόγοι (αν υπάρχουν) βρίσκονται στο component NPC Interaction Handler και γίνονται trigger μετά απο συγκεκριμένα events. (Own NPC Completion ή Followup unlock μετά απο ολοκλήρωση κάποιου άλλου NPC) (Βλέπε Διάγραμμα 104).



Διάγραμμα 102 - NPC Dialogue Component



Διάγραμμα 103 - Intro Dialogue – Candy Fox



Διάγραμμα 104 - Completed & followup dialogues

4.8 Tasks, Task UI και Hints

Το TaskUI δημιουργήθηκε για να χειρίζεται διαφορετικά task panels για τα διαφορετικά task modalities που υπάρχουν στο prototype που δημιουργήθηκε. Είναι υπεύθυνο για να «ανοίγει» το σωστό task panel, να εμφανίζει εικόνες και φράσεις από το TaskData, να μετράει hints και λάθη και να στέλνει τα αποτελέσματα στο PerformanceTracker και στο TaskManager.

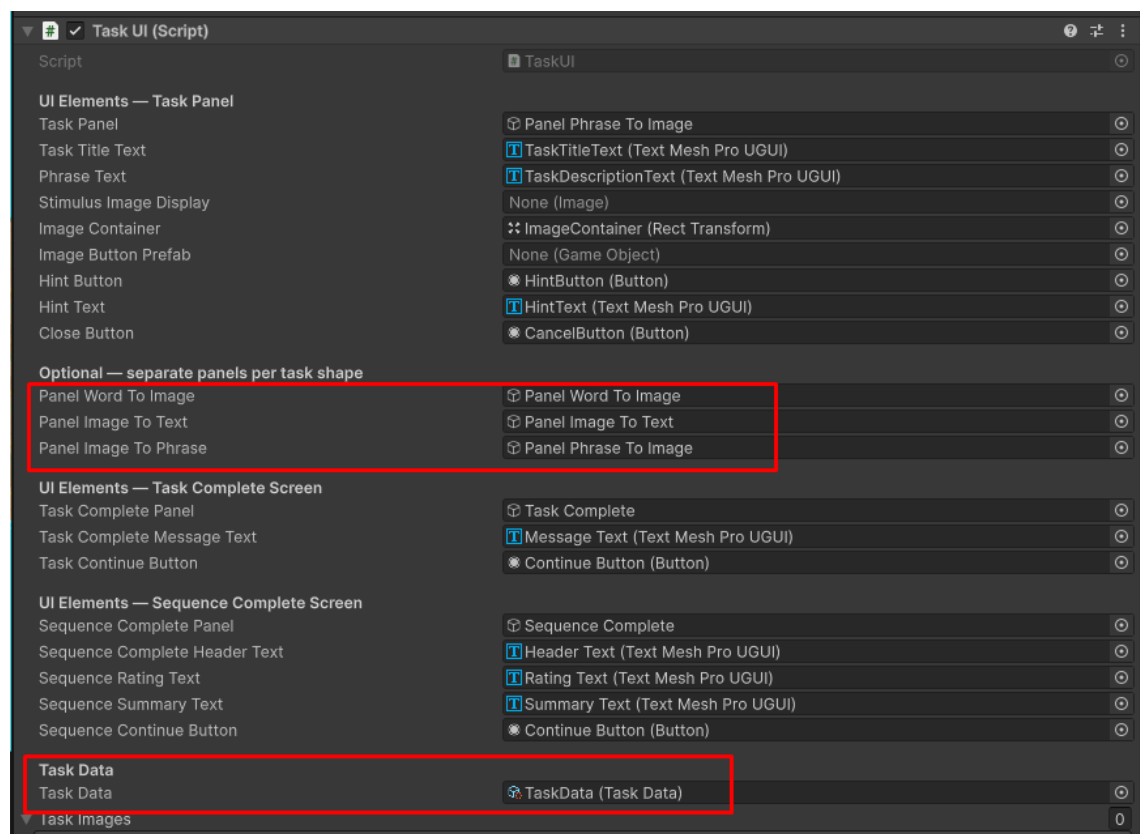
4.8.1 Επιλογή Tasks και Difficulty ανά NPC

Η επιλογή των tasks που θα εμφανίσει ένα NPC εξαρτάται από το difficulty του παίκτη αλλά και με ποίο NPC αλληλεπιδρά ο παίκτης. Κάθε NPC έχει το δικό του σετ (sequence) από tasks για συγκεκριμένα difficulties (Beginner, Intermediate, Advanced) τα οποία εμφανίζονται αν το difficulty του παίκτη κάνει match με το initialDifficulty των tasks στο συγκεκριμένο NPC.

Αναλυτικά:

Όταν γίνει κλικ σε ένα NPC ξεκινάει η «StartTasksForNPC» όπου πέρνει απο την οντολογία ποιό NPC έγινε κλικ (npcUri) και σε ποιά σκηνή βρίσκεται (sceneUri), μετά διαβάζει το τρέχων επίπεδο δυσκολίας του παίκτη (DifficultyProfileUri) απο το JSON file και τρέχει την GetTasksForNPCAtDifficulty για να φέρει μόνο τα tasks αυτού του NPC που στην οντολογία έχουν initialDifficultyProfile ίδιο με του παίκτη. (π.χ όλα τα beginner tasks για το NPC Candy Fox). Μετά τρέχει την HasCompletedTask και διαλέγει το πρώτο task που δεν έχει ηδη ολοκληρωθεί απο τον παίκτη. Τέλος δημιουργείται ένα νέο sequence (BeginNewSequence) και τρέχει το StartTask και το TaskUI.ShowTask για να εμφανιστεί το task στην οθόνη του παίκτη.

Όταν όλα τα task ολοκληρωθούν τότε το NPC θεωρείται ολοκληρωμένο (isNpcFullyCompleted). Αυτό χρησιμοποιείται στο progression order που δίνει τη δυνατότητα στο επόμενο NPC να γίνει «unlock» και κλειδώνει το ολοκληρωμένο NPC. (Βλέπε Διάγραμμα 105-112)



Διάγραμμα 105 - TaskUI component

```
// NPCInteractionHandler.cs
string npcUri = interactiveNPC.npcArchetypeUri;
string sceneUri = GetCurrentSceneUri();
string playerDifficultyUri = GameManager.Instance.CurrentPlayer.DifficultyProfileUri;

var tasks = TaskManager.Instance.GetTasksForNPCAtDifficulty(npcUri, playerDifficultyUri);

foreach (var task in tasks) {
    if (!TaskManager.Instance.HasCompletedTask(task.Uri)) {
        availableTask = task;
        break;
    }
}

PerformanceTracker.BeginNewSequence();
TaskManager.Instance.StartTask(availableTask, sceneUri, npcUri);
taskUI.ShowTask(availableTask, npcUri);
```

Διάγραμμα 106 - Επιλογή task

```
// TaskManager.cs - GetTasksForNPCAtDifficulty
var allTasks = GetTasksForNPC(npcArchetypeUri);
string diffLocal = GetLocalName(playerDifficultyUri);

foreach (var task in allTasks) {
    if (GetLocalName(task.InitialDifficultyProfileUri) == diffLocal)
        filtered.Add(task);
}

return filtered;
```

Διάγραμμα 107 - Difficulty Filter

```
foreach (var completedUri in CurrentPlayer.CompletedTaskUris) {
    if (completedUri == taskUri) return true;
}

return false;
```

Διάγραμμα 108 - Completed task check

```
// true αν ΟΛΑ τα tasks μιας ομάδας είναι completed
foreach (var difficultyGroup in groups) {
    if (all tasks in group are completed) return true;
}
return false;
```

Διάγραμμα 109 - Ελεγχος ολοκλήρωσης NPC

```
PerformanceTracker.RecordSequenceComplete(npcUri, sceneUri);

// PerformanceTracker.cs
decimal averageAccuracy = sumAccuracy / taskCount;
int totalHints = /* άθροισμα hints */;
DifficultyAdjuster.EvaluateAndUpdateDifficultyFromSequence(
    player, averageAccuracy, totalHints, sceneUri, npcUri);
```

Διάγραμμα 110 - performance tracker

```
// DifficultyAdjuster.cs
bool accuracyMatch = averageAccuracy >= policy.PolicyAccuracyFloor
    && averageAccuracy <= policy.PolicyAccuracyCeiling;
bool hintMatch = totalHints >= minHints && totalHints <= maxHints;

if (accuracyMatch && hintMatch) {
    int newLevel = Mathf.Clamp(currentLevel + policy.PolicyDelta, 1, 3);
    player.DifficultyProfileUri = newDifficulty.Uri;
}
```

Διάγραμμα 111 - Difficulty Adjuster

```
string next_npcUri = GetNextNPCInProgressionOrder(); // πρώτος μη-ολοκληρωμένος στη λίστα

// NPCInteractionHandler.cs
if (next_npcUri != clicked_npcUri)
    return; // «δεν έχω tasks» – μπλοκάρει λάθος NPC
```

Διάγραμμα 112 - Progression Order

4.8.1 Επιλογή panel ανα modality

Για την επιλογή του σωστού panel, το TaskManager καλεί την TaskUI.ShowTask(task). Το UI επιλέγει το layout απο το usesModality του TaskTemplate, (Modality_WordToImage, Modality_ImageToText, Modality_ImageToPhrase). Ετσι η οντολογία καθορίζει τον τύπο της άσκησης και το Unity εμφανίζει το αντίστοιχο panel.

- Διαβάζεται το usesModality από το task μέσω UsesModalityUri
- Φορτώνεται το TaskData entry με το ίδιο task URI (εικόνες, φράσεις, hints).
- Image→Text: αν το modality είναι ImageToText και υπάρχουν ≥ 2 choicePhrases στο TaskData τότε εμφανίζεται το panel Image To Text. Αλλιώς δίνεται warning και γίνεται fallback σε Word To Image (known issue → χρειάζεται αφαίρεση του fallback , έγινε χρήση μόνο για δοκιμαστικούς σκοπούς)
- Image→Phrase: αν το modality είναι ImageToPhrase εμφανίζεται panel Phrase To Image.
- Αλλιώς: Panel Word To Image

```
bool ontologyWantsImageToText = task.UsesModalityUri.Contains("Modality_ImageToText");
bool useImageToText = ontologyWantsImageToText && choicePhrases.Count >= 2;

GameObject panelRoot = ResolveTaskPanelRoot(useImageToText, task);
HideAllTaskPanels();
activeMainPanel.SetActive(true);
```

Διάγραμμα 113 - TaskUI script

```
if (useImageToText)           return panelImageToText;
if (IsImageToPhrase(task))    return panelImageToPhrase;
return panelWordToImage;      // default
```

Διάγραμμα 114 - TaskUI panel decision

4.8.2 Hint System

Στο σύστημα χρησιμοποιείται hint system όπου ο παίκτης λαμβάνει hints αν επιλέξει λάθος απάντηση ή αν πατήσει το κουμπί «Get Hint». Όλα τα hints βρίσκονται στο TaskData για το κάθε task ξεχωριστά με τις ιδιότητες τους.

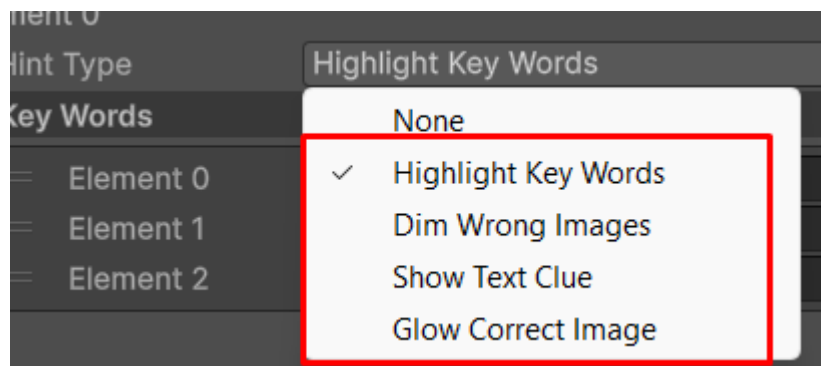
Είδη Hint: (Βλέπε Διάγραμμα 115 και 116)

Highlight Key Words: όπου γίνονται highlight key words που επιλέγονται

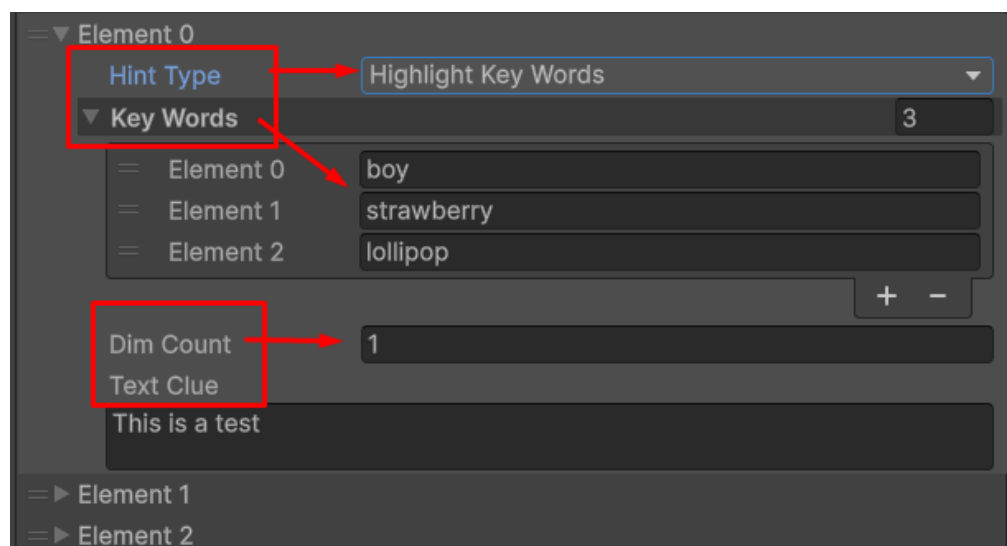
Dim Wrong Images: αφαίρεση λάθος εικόνας που δεν έχει επιλεγεί ακόμα με dim count για την απόφαση πόσων εικόνων να αφαιρεθούν.

Show Text Clue: Δίνεται ένα βοηθητικό text clue στον παίκτη

Glow Correct Image: Εμφανίζεται ένα glow boarder γύρω από τη σωστή εικόνα.



Διάγραμμα 115 - Είδη Hint



Διάγραμμα 116 - Hint UI in Unity

4.9 Sequence, Rewards και Performance Tracking

Για το performance και rewards χρησιμοποιείται ο PerformanceTracker όπου καταγράφει ορθότητα, hints και χρόνο που έκανε ο παίκτης να ολοκληρώσει τα tasks αλλά και το RewardManager που υπολογίζει reward tiers (Bronze/Silver/Gold/Diamond) ανεξάρτητα από το difficulty.

Τα rewards δίνονται στον παίκτη μετά τον outro διάλογο του NPC

4.9.1 Sequence και σύνδεση Dialog & Reward

Η ροή sequence completion → διαλόγων → reward έχει ως εξής:

ShowSequenceCompleteScreen → outro dialogue → RaiseNpcSequenceFlowFinished → RewardManager.TryShowNextPopup. Queue pendingNpcRewards → reward

Δηλαδή, όταν ο παίκτης ολοκληρώσει τα tasks του, εμφανίζεται το Completion Screen με το average performance του για το sequence απο tasks που ολοκλήρωσε, μετά ο outro διάλογος του npc και καλείτε ο RewardManager για να αποφασίσει τι reward πρέπει να πάρει ο παίκτης. Τα rewards μπαίνουν σε ένα queue μέχρι ο παίκτης να τελειώσει με τους διαλόγους του NPC, έτσι ώστε να μην υπάρχει overlap μεταξύ των δύο panel. Όταν ολοκληρωθεί ο διάλογος του NPC, τότε εμφανίζεται το reward panel και ο παίκτης λαμβάνει το reward του ανάλογα με το performance που είχε. (Βλέπε Διάγραμμα 117-120)

```
decimal avgAccuracy = seqTotalAccuracy / seqTaskCount;
decimal avgScore    = seqTotalScore / seqTaskCount;
sequenceSummaryText.text =
    $"Average Score: {avgScore:F0} / 100\nTotal Hints: {seqTotalHints}\n...";
```

Διάγραμμα 117 – Average accuracy/score

```
// TaskUI.cs – OnSequenceContinueClicked → EndSequenceFlow
DialogueManager.Instance.ShowOutro(npcDialogue, EndSequenceFlow);

void EndSequenceFlow() {
    HideTask();
    OnNPCSequenceFlowFinished?.Invoke(finishedNPCUri);
}
```

Διάγραμμα 118 – Outro Dialogue & End Sequence

```
// RewardManager.cs – OnNPCSequenceFlowFinished
if (_pendingNPCRewards.TryGetValue(key, out var npcReward)) {
    _pendingNPCRewards.Remove(key);
    _popupQueue.Enqueue(npcReward);
}
TryShowNextPopup();
```

Διάγραμμα 119 – Reward queue

```
// RewardManager.cs – TryShowNextPopup
var reward = _popupQueue.Dequeue();
rewardPopupPanel.SetActive(true); // trophy + title + description
```

Διάγραμμα 120 - Reward dequeue

4.9.2 Rewards – Υπολογισμός και εμφάνιση

Τα tiers των tasks υπολογίζονται από το μέσο όρο του score (0-100) και δεν επηρεάζονται από το difficulty.

Score: 0 - 60 = Bronze

Score: 60 - 79 = Silver

Score: 80 - 99 = Gold

Score: 100 = Diamond

Τα rewards εμφανίζονται μόνο 1 φορά ανά NPC (one-time markers).

Ο παίκτης λαμβάνει επίσης ένα reward με τον ίδιο τρόπο όταν ολοκληρώσει επιτυχώς όλα τα NPCs της σκηνής.

```
private static TrophyTier ResolveTierFromScore(decimal score) {  
    if (score >= 100m) return TrophyTier.Diamond;  
    if (score >= 80m) return TrophyTier.Gold;  
    if (score >= 60m) return TrophyTier.Silver;  
    return TrophyTier.Bronze;  
}
```

Παράδειγμα κώδικα reward tier

4.9.2.1 Rewards – Αριθμητικό παράδειγμα

Scores σε 3 tasks: 90, 85, 70 → average 81.67 → Gold Reward

4.9.3 PerformanceTracker

Το PerformanceTracker κρατά τι έκανε ο παίκτης σε κάθε task και στο τέλος μιας ακολουθίας NPC, υπολογίζει μέσους όρους και τρέχει policies για δυσκολία.

Για τη καταγραφή της επίδοσης του παίκτη χρησιμοποιείται η RecordTaskCompletion όπου καλείτε μετά την ολοκλήρωση ενός task.

- Φτιάχνει TaskPerformance: accuracy, score, hints, χρόνος, scene, timestamp και difficulty πριν το task
- Προσθέτει το task URI στο CompletedTaskUris
- Το προσθέτει στο PerformanceHistory του παίκτη στο JSON
- Κάνει update to profile του παίκτη στο PlayerProfiles αρχείο

Μετά τρέχει η RecordSequenceComplete όπου αθροίζει όλα τα tasks και βγάζει μέσο accuracy, μέσο score και σύνολο hints, όπου ο DifficultyAdjuster τα συγκρίνει με τα policies και ενημερώνει το player.DifficultyProfileUri (αν ταιριάζει κάποιο Policy) και αποθηκεύει το profile.

```

public static void RecordTaskCompletion(
    string taskUri, string sceneUri,
    decimal accuracy, decimal score, int hintCount, long latencyMs)
{
    var player = GameManager.Instance.CurrentPlayer;
    string difficultyBefore = player.DifficultyProfileUri;

    var performance = new TaskPerformance {
        TaskUri = taskUri,
        SceneUri = sceneUri,
        Timestamp = DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss"),
        Accuracy = (double)accuracy,
        Score = (double)score,
        HintCount = hintCount,
        LatencyMs = latencyMs,
        DifficultyProfileUriBefore = difficultyBefore
    };

    // Πρόσθεσε task στο CompletedTaskUris
    if (!alreadyCompleted)
        player.CompletedTaskUris.Add(normalizedUri);

    _currentSequencePerformances.Add(performance); // buffer για sequence
    performance.DifficultyProfileUriAfter = difficultyBefore;
    player.PerformanceHistory.Add(performance);

    PlayerProfileManager.UpdateProfile(player); // JSON
}

```

Διάγραμμα 121 – RecordTaskCompletion

```

public static void RecordSequenceComplete(string npcUri, string sceneUri)
{
    foreach (var p in _currentSequencePerformances) {
        sumAccuracy += (decimal)p.Accuracy;
        sumScore += (decimal)p.Score;
        totalHints += p.HintCount;
    }
    decimal averageAccuracy = sumAccuracy / count;
    decimal averageScore = sumScore / count;

    string newDifficulty = DifficultyAdjuster.EvaluateAndUpdateDifficultyFromSequence(
        player, averageAccuracy, totalHints, sceneUri, npcUri);

    foreach (var p in _currentSequencePerformances)
        p.DifficultyProfileUriAfter = newDifficulty;

    _currentSequencePerformances.Clear();
    PlayerProfileManager.UpdateProfile(player);

    OnSequenceCompleted?.Invoke(npcUri, sceneUri);
    OnSequenceCompletedWithScore?.Invoke(npcUri, sceneUri, averageScore);
}

```

Διάγραμμα 122 - RecordSequenceComplete

4.10 Σύστημα Narrator

Ο narrator εμφανίζεται στο παιχνίδι για να καθοδηγήσει τον παίκτη έτσι ώστε να υπάρχει μια ομαλή ροή στο παιχνίδι, χωρίς ο παίκτης να «κολλάει» και να μήν γνωρίζει τι πρέπει να κάνει και που να προχωρίσει.

Γίνεται χρήση του NarratorGuideUI όπου εμφανίζει μηνύματα σε milestones όπως (είσοδος στο παιχνίδι και ολοκλήρωση NPC).

Κάθε μήνυμα είναι one-time με click-to-advance και δεν μπορεί να ξανα εμφανιστεί.

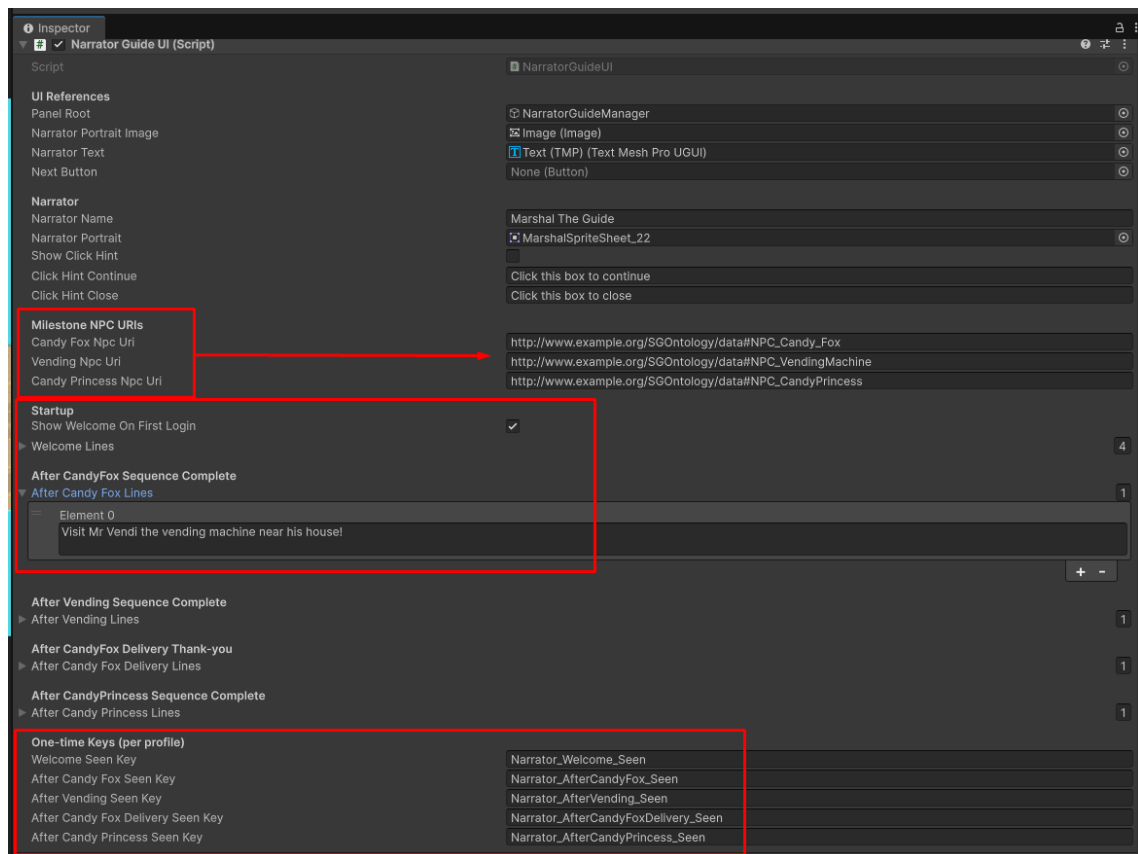
Χρήσιμοποιεί milestones URIs απο τα NPCs και απο events όπως «OnSequenceCompleted» και γίνονται marked ως «seen» όταν εμφανιστεί ο narrator στην οθόνη του παίκτη.

```
private void OnSequenceCompleted(string npcUri, string sceneUri) {
    if (UriEqual(npcUri, candyFoxNPCUri) &&
        !HasSeenMarker(afterCandyFoxSeenKey))
        ShowLines(afterCandyFoxLines, afterCandyFoxSeenKey);
}
private void MarkSeen(string key) {
    player.CompletedTaskUris.Add("narrator://seen/" + key.Trim());
    PlayerProfileManager.UpdateProfile(player);
}
```

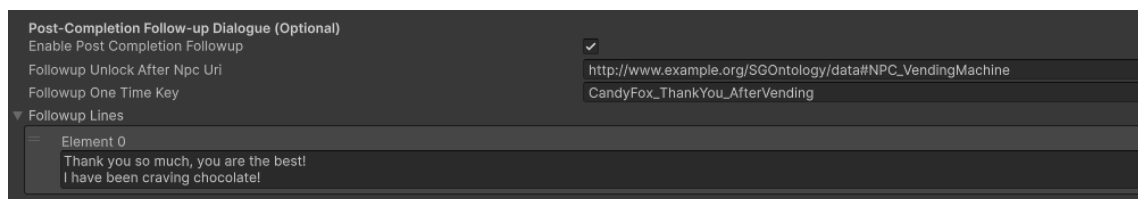
Παράδειγμα κώδικα

4.10.1 Narrator -- milestones

Welcome (Start) → after CandyFox sequence → after VendingMachine → CandyFox thank-you (OnOneTimeFollowupShown) → after CandyPrincess (Βλέπε Διάγραμμα 123 και 124)



Διάγραμμα 123 - Narrator UI Component



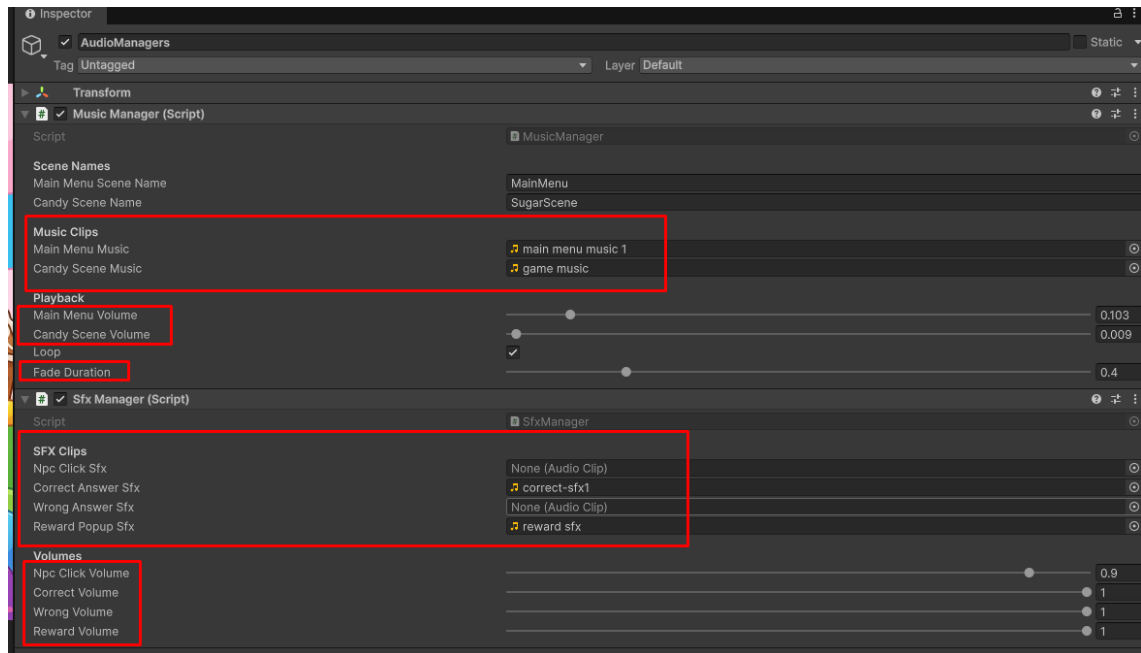
Διάγραμμα 124 - Followup dialogue for Candy Fox

4.11 Audio System

Η μουσική και τα sound effects παίζουν σημαντικό ρόλο στην εντύπωση του παίκτη για το παιχνίδι αλλά κάνοντας το παιχνίδι πιο ψυχαγωγικό χωρίς να χάνει τους εκπαιδευτικούς του σκοπούς. Αντιθέτως κρατάει τον χρήστη στο παιχνίδι χωρίς να δημιουργεί την αίσθηση «βαρεμάρας» που μπορεί να επηρεάσει τον χρήστη σε θέματα performance ή ακόμη και στη συνέχιση του παιχνιδιού.

Τα sound effects δίνουν επίσης ανατροφοδότηση στον χρήστη με ήχους όταν γίνεται κλικ σε κάποιο NPC, όταν επιλέγει λάθος ή σωστές απαντήσεις και όταν λαμβάνει rewards.

Η κάθε σκηνή (Main Menu, SugarScene) έχει τη δικιά της μουσική και sound volume adjustments με fade-in fade-out έτσι ώστε να μην γίνεται απότομη αλλαγή στη μουσική. (Βλέπε Διάγραμμα 125)



Διάγραμμα 125 - Audio Manage

5 Κεφάλαιο 5: Επίδειξη Συστήματος

5.1 Σκοπός Κεφαλαίου

Το παρόν κεφάλαιο παρουσιάζει τη λειτουργία του συστήματος από τη σκοπιά του τελικού χρήστη, με τη περιγραφή κάθε οθόνης και σενάριο χρήσης, καθώς και μετρικές αποτίμησης βάσει δοκιμαστικών εκτελέσεων.

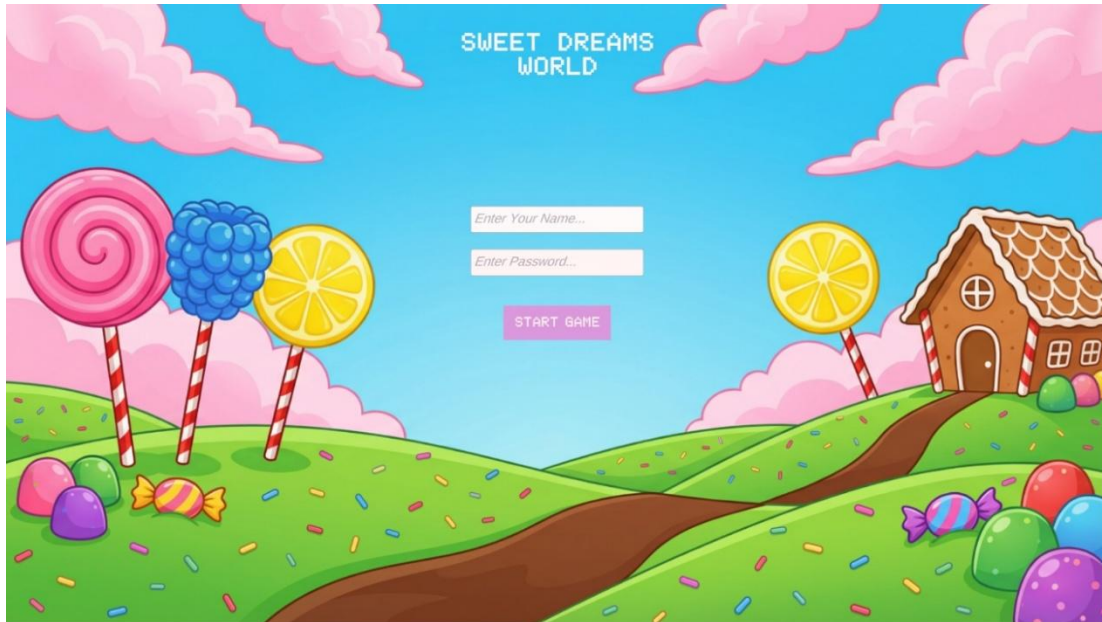
5.2 Σενάριο Επίδειξης

Το σενάριο επίδειξης αποτελείται από τα εξής βήματα:

1. Εκκίνηση εφαρμογής και προβολή Main Menu με επιλογή έναρξης παιχνιδιού.
2. Μετάβαση στη σκηνή παιχνιδιού με fade-in μουσική.
3. Εμφάνιση narrator με εισαγωγική οδηγία (one-time trigger).
4. Πλοήγηση χαρακτήρα και αλληλεπίδραση με τον πρώτο NPC (proximity check + order check).
5. Εκτέλεση tasks διαφορετικών modalities με παροχή hints αν χρειαστεί.
6. Ηχητικό feedback για κάθε απάντηση (σωστό/λάθος SFX).
7. Ολοκλήρωση sequence και εμφάνιση rewards.
8. Narrator guidance για τον επόμενο στόχο του παίκτη.
9. Συνέχιση progression με τα επόμενα NPCs μέχρι την ολοκλήρωση της σκηνής.

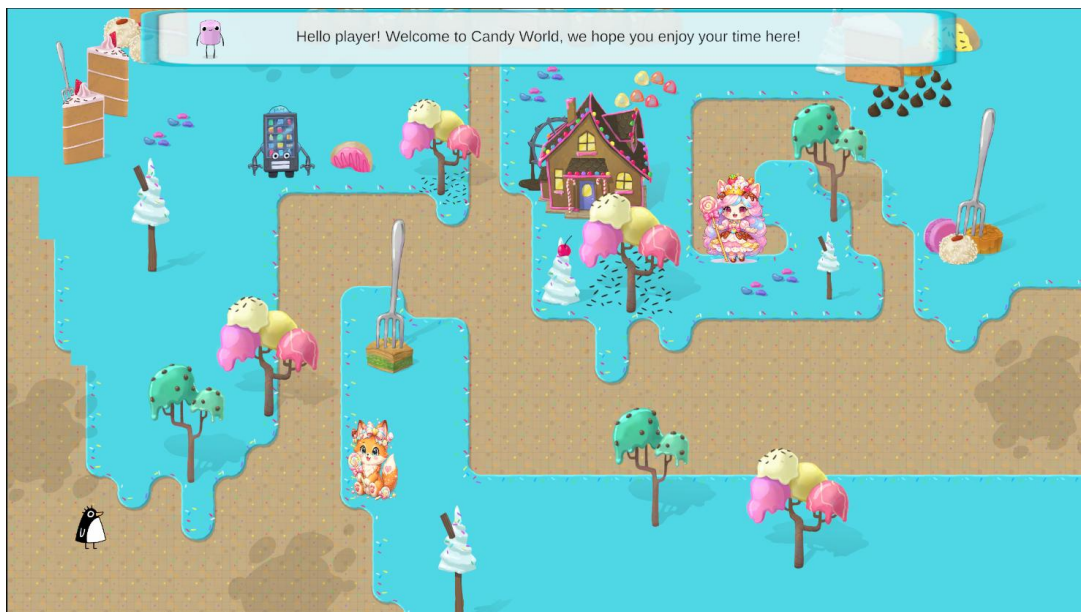
5.3 Παρουσίαση Οθονών

Main Menu -- Αρχική οθόνη με επιλογή έναρξης παιχνιδιού όπου ο χρήστης πρέπει να αναρτήσει όνομα και κωδικό για να συνδεθεί στο profile του αν έχει ήδη ή για τη δημιουργία του.



Διάγραμμα 126 - Main Menu

Έναρξη παιχνιδιού - Αρχική οθόνη όπου φαίνεται η πρώτη σκηνή του παιχνιδιού με όλα τα NPCs της και το Welcome message του Narrator.



Διάγραμμα 127 - Έναρξη παιχνιδιού

NPC Interaction -- Αλληλεπίδραση παίκτη με NPC μετά από proximity και order check.
(Κάθε NPC περιέχει αρχικό διάλογο - Task Sequence – Τελικό διάλογο).



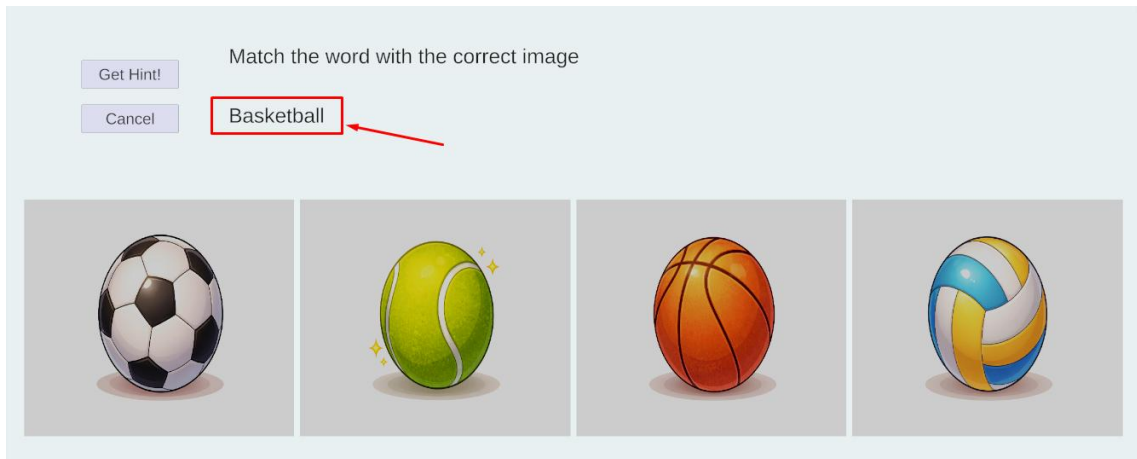
Διάγραμμα 128 - NPC Interaction

Image-to-Phrase panel – Εμφάνιση του πρώτου task μετά τον into διάλογο του NPC --
Ο παίκτης επιλέγει την εικόνα που αντιστοιχεί στη πρόταση που του δίνεται.



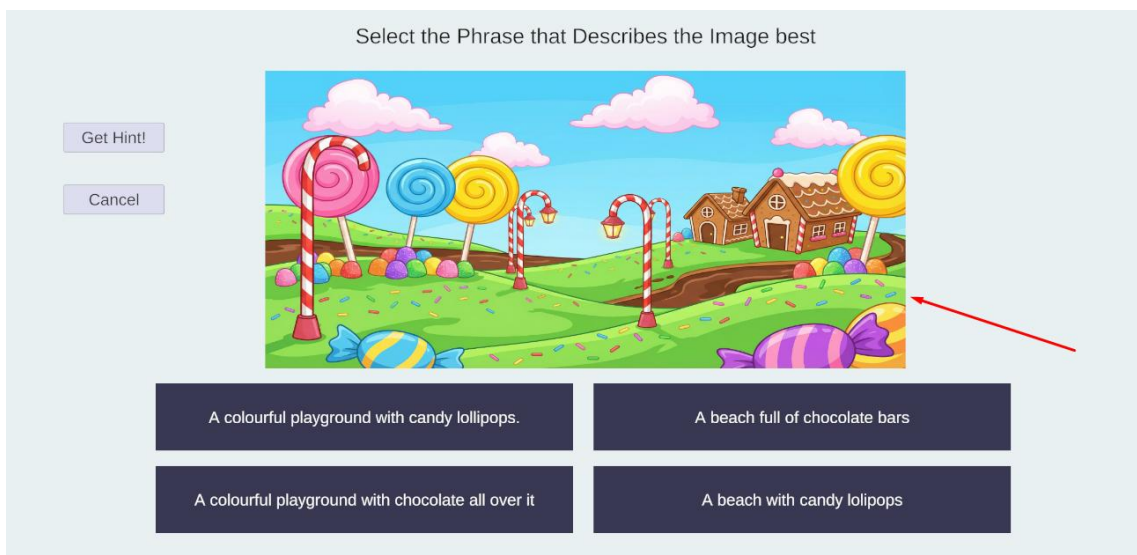
Διάγραμμα 129 - Image-to-Phrase panel

Word-to-Image panel -- Ο παίκτης επιλέγει την εικόνα που αντιστοιχεί στη λέξη που του δίνεται.



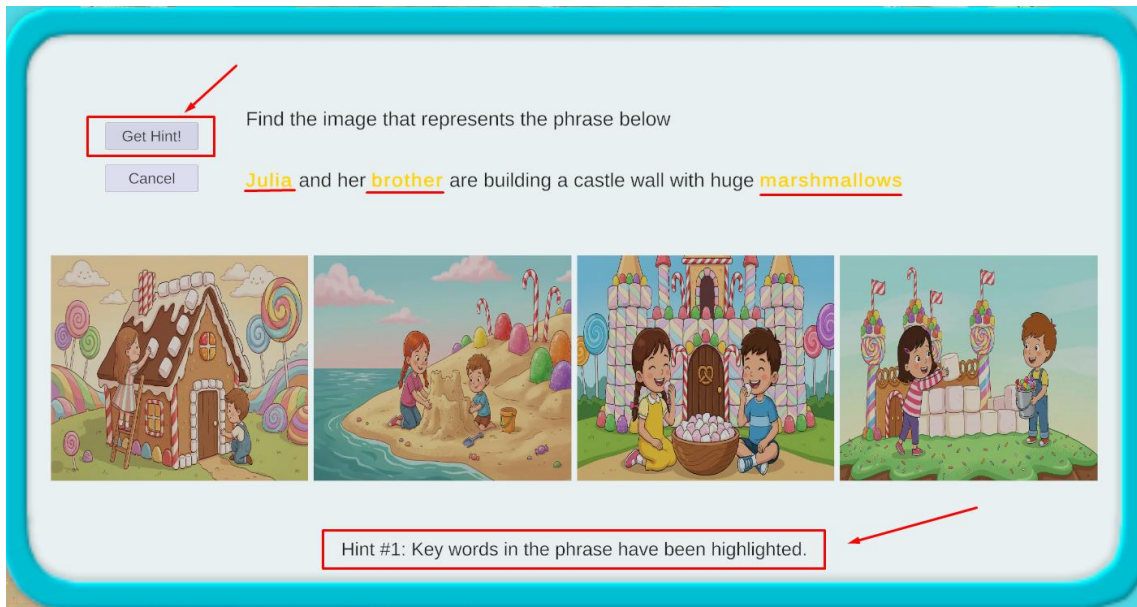
Διάγραμμα 130 - Word-to-Image panel

Image-to-Text panel -- Ο παίκτης επιλέγει τη πρόταση που αντιστοιχεί καλύτερα στην εικόνα που του δίνεται.



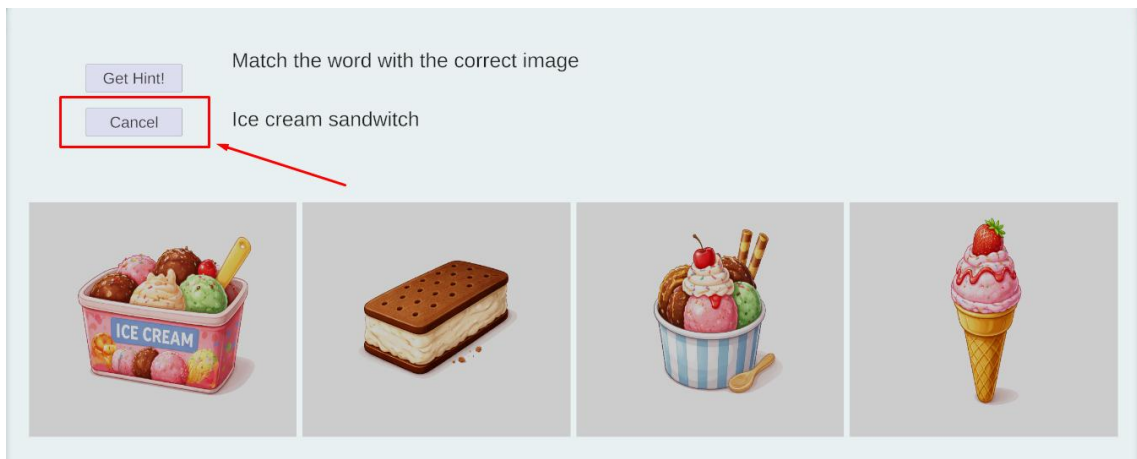
Διάγραμμα 131 - Image-to-Text panel

Hint button -- Εμφάνιση βοήθειας (hint) πατώντας το κουμπί «Get Hint!». (Hints δίνονται **και** όταν δοθεί λάθος απάντηση από τον παίκτη).



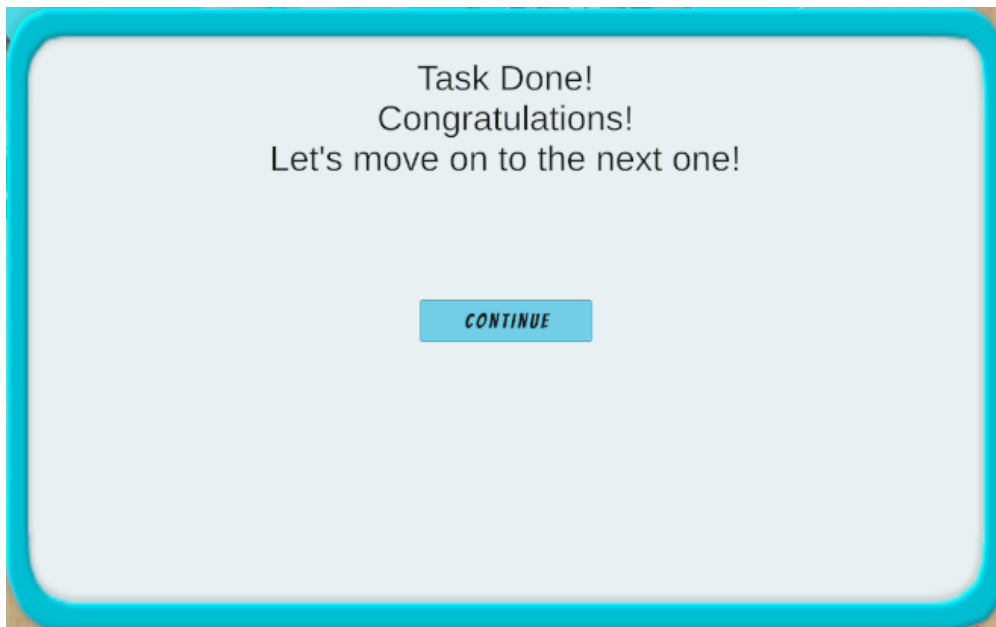
Διάγραμμα 132 - Hint button

Cancel button -- Κλείσιμο παραθύρου task (Επαναφορά στο ίδιο task όταν γίνει ξανά interaction με το NPC).



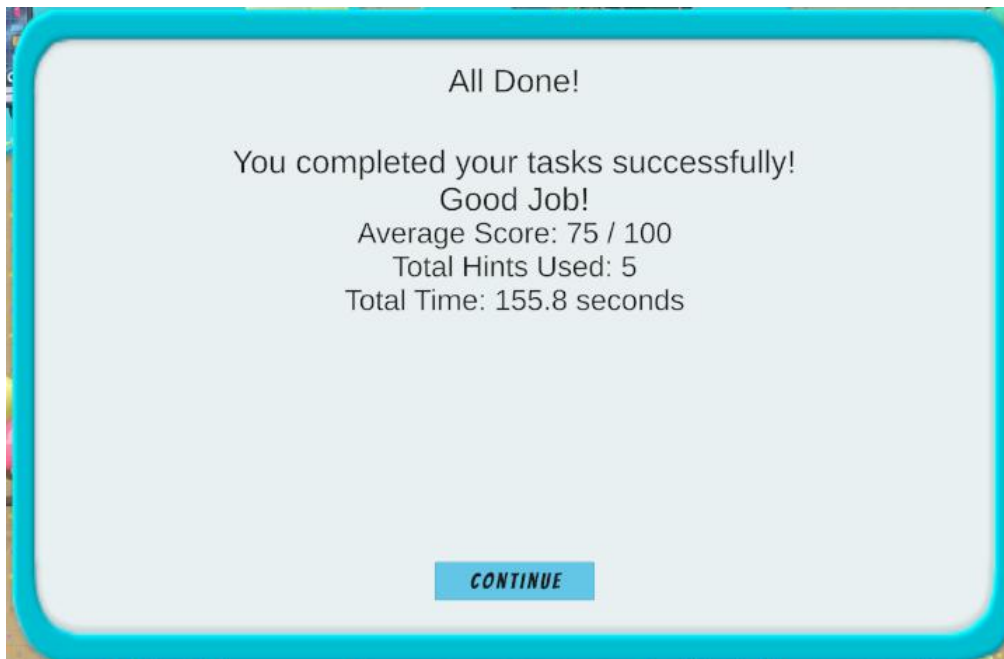
Διάγραμμα 133 - Cancel button

Progress Congratulations panel μετά απο την ολοκλήρωση ενός task.



Διάγραμμα 134 - Progress Congratulations panel

Task sequence report panel μετά απο την ολοκλήρωση task sequence (Ολοκλήρωση όλων των task για ένα NPC) με το average score, total hints και το χρόνο που χρειάστηκε για την ολοκλήρωση.



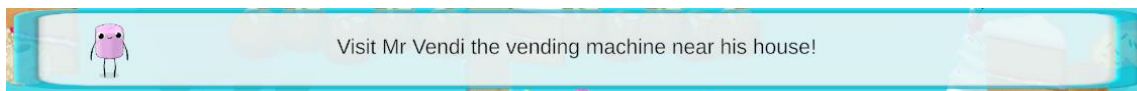
Διάγραμμα 135 - Task sequence report panel

Task Reward popup -- Εμφάνιση reward tier (π.χ. Gold) μετά ολοκλήρωση NPC sequence.

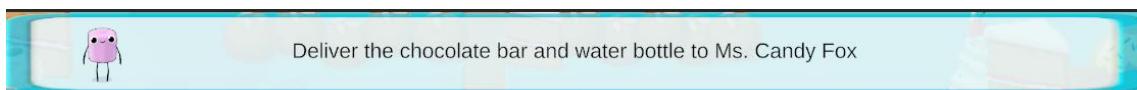


Διάγραμμα 136 - Task Reward popup

Narrator panel -- Εμφάνιση οδηγιών του αφηγητή σε milestones.



Διάγραμμα 137 - Narrator panel



Διάγραμμα 138 - Narrator panel

5.4 Μετρικές Αποτίμησης

Για κάθε δοκιμαστικό run καταγράφηκαν οι εξής μετρικές:

- Ποσοστό ολοκλήρωσης ανά task (%).
- Μέσο score ανά NPC sequence.
- Αριθμός hints που χρησιμοποιήθηκαν ανά task.
- Χρόνος ολοκλήρωσης ανά task (δευτερόλεπτα).

5.5 Συζήτηση Αποτελεσμάτων

Η επίδειξη αναδεικνύει ότι η οντολογική οργάνωση υποστηρίζει καθαρό mapping περιεχομένου χωρίς ασυνέπειες. Η progression λογική βελτιώνει τη συνοχή εμπειρίας, ενώ rewards και audio αυξάνουν το engagement. Κύριοι περιορισμοί: απαιτείται μεγαλύτερη αξιολόγηση με πραγματικούς χρήστες και πιο αναλυτικό analytics dashboard.

6 Κεφάλαιο 6: Συμπεράσματα και Μελλοντική Εργασία

6.1 Συμπεράσματα

Η παρούσα εργασία ανέπτυξε ένα εκπαιδευτικό παιχνίδι 2D με ενσωμάτωση οντολογικής μοντελοποίησης γνώσης, αντιμετωπίζοντας το πρόβλημα της δομημένης διαχείρισης περιεχομένου σε game environments. Αποδείχθηκε ότι η οντολογική μοντελοποίηση μπορεί να ενσωματωθεί πρακτικά σε Unity υλοποίηση, παρέχοντας σαφή διαχωρισμό δομικής γνώσης (οντολογία) από game logic (Unity scripts).

Η εφαρμογή καλύπτει end-to-end ροή: από ontology-driven task selection έως runtime interaction, scoring, narration, rewards και audio feedback. Κεντρικό αρχιτεκτονικό συμπέρασμα: η συνέπεια URIs μεταξύ οντολογίας και Unity mapping αποτελεί βασική προϋπόθεση ορθής λειτουργίας, ενώ η event-driven αρχιτεκτονική διατηρεί καθαρό διαχωρισμό ευθυνών.

6.2 Επίτευξη Στόχων

Οι αρχικοί στόχοι (Κεφάλαιο 1) επιτεύχθηκαν σε μεγάλο βαθμό:

- Δημιουργήθηκε λειτουργικό οντολογικό μοντέλο με κλάσεις NPC, Task, Modality, DifficultyLevel.
- Πραγματοποιήθηκε επιτυχής σύνδεση με Unity μέσω URI-based lookup και TaskData ScriptableObjects.
- Υλοποιήθηκαν και οι τρεις task modalities με δυναμική δημιουργία UI.
- Ο progression μηχανισμός λειτουργεί με σειρά NPCs και proximity constraints.
- Τα συστήματα narrator, rewards και audio προστέθηκαν επιτυχώς.

6.3 Περιορισμοί

- Δεν έχει ολοκληρωθεί μεγάλης κλίμακας αξιολόγηση με πραγματικούς χρήστες.
- Η ανάλυση runtime δεδομένων (JSON logs) βρίσκεται σε αρχικό στάδιο.
- Ο adaptation engine είναι απλός και δεν προσαρμόζει δυναμικά τη δυσκολία.
- Το πρωτότυπο καλύπτει μόνο τον πρώτο κόσμο.

6.4 Μελλοντική Εργασία

1. Ανάπτυξη analytics dashboard με οπτικοποίηση δεδομένων από JSON/DB έτσι ώστε να υπάρχει analytical feedback που θα βοηθήσει σε μελλοντικά updates/αλλαγές και για τη δημιουργία του πραγματικού παιχνιδιού.
2. Προσθήκη νέων κόσμων, NPCs και task types.
3. Υλοποίηση adaptation engine με ιστορικό επιδόσεων για δυναμική ρύθμιση δυσκολίας.
4. Εκτενής πειραματική αξιολόγηση με στατιστική ανάλυση σε πραγματικό πληθυσμό.
5. Αυτοματοποιημένοι έλεγχοι συνέπειας URI mapping κατά build time.
6. Χρήση μοντέλων όπου θα δίνουν στοχευμένα hints στους παίκτες και όχι standard hints.

6.5 Τελική Αποτίμηση

Το τελικό αποτέλεσμα είναι ένα σταθερό και επεκτάσιμο πρωτότυπο που συνδυάζει semantic μοντελοποίηση και game development σε πρακτικό πλαίσιο πτυχιακής εργασίας. Η εργασία αποδεικνύει ότι η ενοποίηση οντολογιών με game engines είναι εφικτή και προσφέρει ουσιαστικά πλεονεκτήματα, αποτελώντας βάση για περαιτέρω ερευνητική εξέλιξη.

7 Βιβλιογραφία

- Bonnechère, B., Migeotte, P.-F., Forthomme, B., Croisier, J.-L., Schwartz, C., & Bavier, A. (2021). Serious games in healthcare: A meta-review. *Σχετικό περιοδικό*.
- Diamantini, C., Lo Giudice, P., Maddalena, L., Meo, R., Musarella, L., Potena, D., Rosaci, D., & Sarné, G. M. L. (2019). Enriching data lake metadata through semantics. *Proceedings of the International Conference on Extending Database Technology (EDBT)*.
- Hai, R., Geisler, S., & Quix, C. (2016). Constance: An intelligent data lake system. *Proceedings of the 2016 International Conference on Management of Data (SIGMOD)* (pp. 2097–2100). ACM.
- Kickmeier-Rust, M. D., & Albert, D. (2010). Micro-adaptivity: Protecting immersion in didactically adaptive digital educational games. *Journal of Computer Assisted Learning*, 26(2), 95–105.
- Lamb, R., Annetta, L., Firestone, J., & Etopio, E. (2018). A meta-analysis with examination of moderators of student cognition, affect, and learning outcomes while using serious educational games. *Computers in Human Behavior*, 80, 158–167.
- Makransky, G., & Lilleholt, L. (2018). A structural equation modeling investigation of the emotional value of immersive virtual reality in education. *Educational Technology Research and Development*, 66(5), 1141–1164.
- Musen, M. A. (2015). The Protégé project: A look back and a look forward. *AI Matters*, 1(4), 4–12.
- Nichols, M., Cator, K., & Torres, M. (2020). *Challenge based learning guide*. Digital Promise.
- Sawadogo, P., & Darmont, J. (2021). On data lake architectures and metadata management. *Journal of Intelligent Information Systems*, 56(1), 97–120.

Spruit, M., & Jagesar, R. (2016). Iterative metadata management in self-adaptive serious games.

Proceedings of the 2016 International Conference on Information Systems (ICIS).

Taib, R., Tedman, R., & Chiou, E. K. (2019). Targeting social learning and engagement using a cooperative multi-player serious game.

Computers & Education, 131, 450–461.

W3C. (2012).

OWL 2 Web Ontology Language document overview (second edition). World Wide Web Consortium. <https://www.w3.org/TR/owl2-overview/>

Loizou, S., & Andreou, A. S. (2025). A framework for standardizing the development of serious games with real-time self-adaptation capabilities using digital twins.

Technologies, 13(8), Article 369. <https://doi.org/10.3390/technologies13080369>