ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
UNIVERSITY OF PIRAEUS

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

ΤΜΗΜΑ ΒΙΟΜΗΧΑΝΙΚΗΣ ΔΙΟΙΚΗΣΗΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ

ΠΜΣ ΣΤΗ ΒΙΟΜΗΧΑΝΙΚΗ ΔΙΟΙΚΗΣΗ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑ

ΕΙΔΙΚΕΥΣΗ:ΔΙΟΙΚΗΣΗ LOGISTICS

Διαδικασία Picking με Σύγχρονες Μεθόδους (Unity-AR)

Δημήτριος Πέτρου

Α.Μ. : TML2123

Επιβλέπον Καθηγητής:
Χονδροκούκης Γρηγόριος

Πειραιάς, Δεκέμβρης 2024

Περιεχόμενα

1. Εισαγωγή

Η διαχείριση των αποθηκευτικών διαδικασιών αποτελεί ένα ζωτικής σημασίας τμήμα της εφοδιαστικής αλυσίδας για πολλές επιχειρήσεις. Οι αποθηκευτικοί χώροι αποτελούν τον τομέα όπου αποθηκεύονται και διατίθενται τα προϊόντα, είτε αυτά είναι καταναλωτικά αγαθά είτε είδη παραγωγής. Η αποτελεσματική διαχείριση της διαδικασίας συλλογής, που αναφέρεται στην επιλογή και τη συλλογή των προϊόντων από τις αποθηκευτικές θέσεις για την ικανοποίηση των παραγγελιών, έχει αναδειχθεί ως κρίσιμη για την αύξηση της αποτελεσματικότητας και την μείωση των λαθών στην εφοδιαστική αλυσίδα.

Στην παρούσα διπλωματική εργασία, εστιάζουμε στη χρήση του αλγορίθμου Dijkstra για την επίλυση του προβλήματος αυτού. Ο αλγόριθμος Dijkstra αποτελεί έναν από τους πιο διακεκριμένους αλγορίθμους γράφων, ιδανικός για την εύρεση των συντομότερων διαδρομών μεταξύ σημείων ενός γράφου. Αυτός ο αλγόριθμος επιλύει αποτελεσματικά το πρόβλημα του πολύ-διαδρόμου και μπορεί να εφαρμοστεί σε πληθώρα πεδίων, συμπεριλαμβανομένου του χώρου της αποθήκης.

Επίσης, θα εξετάσουμε λεπτομερώς τη θεωρία πίσω από τον αλγόριθμο Dijkstra, θα παρουσιάσουμε τον σχεδιασμό και την υλοποίηση της εφαρμογής, και θα αξιολογήσουμε τα αποτελέσματα και τις επιδόσεις της. Επιπλέον θα εξετάσουμε πιθανές βελτιώσεις και μελλοντικές επεκτάσεις της εφαρμογής.

Στην προσπάθεια να αξιοποιήσουμε αυτόν τον αλγόριθμο, αναπτύξαμε μια εφαρμογή σε Unity που ενσωματώνει τον αλγόριθμο Dijkstra για τη βελτιστοποίηση της διαδικασίας συλλογής.

Μέσω ενός ευέλικτου περιβάλλοντος προσομοίωσης, η εφαρμογή αυτή διευκολύνει την αξιολόγηση των επιδόσεων του αλγορίθμου Dijkstra σε διάφορες συνθήκες αποθήκης.

Η παρούσα εργασία αποτελεί μια ανάλυση της συνάφειας του αλγορίθμου Dijkstra με το πρόβλημα της βέλτιστης διαδικασίας συλλογής σε αποθήκη και ένα βήμα προς τη δημιουργία και αξιοποίηση μιας νέας προσέγγισης για την βελτιστοποίηση των διαδικασιών συλλογής σε εφοδιαστικές αλυσίδες.

2. Ανάγκη για γρήγορη και αποτελεσματική συλλογή

Η ανάγκη για γρήγορη και αποτελεσματική συλλογή στη σύγχρονη εποχή είναι αναπόφευκτη και αποτελεί κρίσιμο παράγοντα για την αειφόρο και αποδοτική λειτουργία των αποθηκών και την ικανοποίηση των αυξημένων απαιτήσεων των σύγχρονων καταναλωτών. Παράγοντες όπως η ταχύτητα, η ακρίβεια, η αυξημένη ζήτηση, ο ανταγωνισμός, οι τεχνολογικές εξελίξεις και η επαναστατική εξέλιξη του εμπορίου έχουν διαμορφώσει το περιβάλλον των επιχειρήσεων και την καθημερινότητα των καταναλωτών.

Ας αναλύσουμε αναλυτικά κάθε έναν από αυτούς τους παράγοντες:

1. **Αυξημένη Ζήτηση και Ανταγωνισμός:** Η αυξημένη ζήτηση για προϊόντα και υπηρεσίες σε όλους τους κλάδους έχει δημιουργήσει έναν υψηλό ανταγωνισμό μεταξύ των επιχειρήσεων. Για να διατηρήσουν το μερίδιο της αγοράς και να ικανοποιήσουν τους πελάτες τους, οι επιχειρήσεις πρέπει να προσφέρουν γρήγορες παραδόσεις. Ο αλγόριθμος Dijkstra μπορεί να βοηθήσει στη βελτίωση της διαδικασίας συλλογής, επιλέγοντας τις βέλτιστες διαδρομές για την εξυπηρέτηση των παραγγελιών με τον ταχύτερο τρόπο.

2. **Ταχύτητα Παράδοσης:** Οι καταναλωτές αναζητούν όχι μόνο ποιοτικά προϊόντα, αλλά και γρήγορες παραδόσεις. Η επιθυμία για άμεση ικανοποίηση των αναγκών τους έχει καταστήσει την ταχύτητα παράδοσης έναν κρίσιμο παράγοντα. Ο αλγόριθμος Dijkstra ως γραφοθεωριτικός αλγόριθμος όπως αναφέρει ο Frederick S. Hillier et al., μπορεί να βελτιστοποιήσει τη διαδικασία συλλογής προσδιορίζοντας τις διαδρομές που εξασφαλίζουν τις γρηγορότερες παραδόσεις.

3. **Εξοικονόμηση Κόστους:** Η αποτελεσματική διαδικασία συλλογής συμβάλλει στη μείωση των λαθών, της χρονικής απώλειας και των ανθρώπινων πόρων. Αυτό έχει ως αποτέλεσμα την εξοικονόμηση κόστους, καθώς οι επιχειρήσεις μπορούν να λειτουργούν πιο αποτελεσματικά και αποδοτικά.

4. **Απαιτήσεις E-Commerce:** Το ηλεκτρονικό εμπόριο έχει αλλάξει τον τρόπο που αγοράζουμε προϊόντα. Οι καταναλωτές προτιμούν την άμεση παράδοση των αγορών τους. Επομένως, οι επιχειρήσεις πρέπει να είναι σε θέση να ανταποκριθούν σε αυτές τις απαιτήσεις με ταχύτητα και ακρίβεια.

5. **Πολυπλοκότητα Αποθηκευτικών Περιβαλλόντων:** Οι μοντέρνες αποθήκες συχνά έχουν σύνθετες δομές με δεκάδες, αν όχι εκατοντάδες, τοποθεσίες αποθήκευσης. Η αποτελεσματική διαχείριση των αποθηκευτικών αυτών χώρων απαιτεί τον υπολογισμό των βέλτιστων διαδρομών για τη διαδικασία συλλογής, κάτι που μπορεί να επιτευχθεί με τον αλγόριθμο Dijkstra. (Sunil Chopra et al.)

Η παγκοσμιοποίηση του εμπορίου και η άνοδος του ηλεκτρονικού εμπορίου έχουν δημιουργήσει ένα περιβάλλον όπου οι καταναλωτές έχουν αυξημένες προσδοκίες για γρήγορες και αποτελεσματικές παραδόσεις. Οι πελάτες περιμένουν τα προϊόντα τους να παραδίδονται σε ελάχιστο χρόνο, κάτι που έχει δημιουργήσει αναγκαιότητα για αποτελεσματική διαχείριση των αποθηκευτικών διαδικασιών.

Ο ανταγωνισμός στην αγορά έχει ενταθεί σημαντικά, με τις επιχειρήσεις να αναζητούν συνεχώς τρόπους για τη βελτίωση της απόδοσης και τη μείωση των λειτουργικών κοστών. Το γρήγορο και αποτελεσματικό picking επιτρέπει την εξοικονόμηση χρόνου και πόρων, επιτρέποντας στις επιχειρήσεις να ανταποκριθούν στη ζήτηση με μεγαλύτερη αποτελεσματικότητα.

Η τεχνολογική πρόοδος έχει επίσης συντελέσει στην αύξηση της ανάγκης για γρήγορη διαδικασία συλλογής. Η αυτοματοποίηση των διαδικασιών, η χρήση ρομποτικής και η εφαρμογή προηγμένων συστημάτων διαχείρισης αποθηκών έχουν δημιουργήσει νέες δυνατότητες για τη βελτιστοποίηση του picking. (Michael ten Hompel) Ο αλγόριθμος Dijkstra μπορεί να ενσωματωθεί σε αυτά τα συστήματα για την εύρεση των βέλτιστων διαδρομών που πρέπει να ακολουθήσουν τα ρομπότ ή οι εργαζόμενοι προκειμένου να συλλέξουν τα απαιτούμενα εμπορεύματα.

Επιπλέον, η σύγχρονη κοινωνία έχει αυξημένες απαιτήσεις όσον αφορά την προστασία του περιβάλλοντος. Οι αποθήκες και οι επιχειρήσεις πρέπει να λειτουργούν με τρόπο που να μειώνει τον αντίκτυπο στο περιβάλλον. Ο γρήγορος και αποτελεσματικός αλγόριθμος picking μπορεί να συμβάλλει στη μείωση της ενεργειακής κατανάλωσης και της εκπομπής αερίων, μειώνοντας τον αντίκτυπο των επιχειρήσεων στο περιβάλλον.

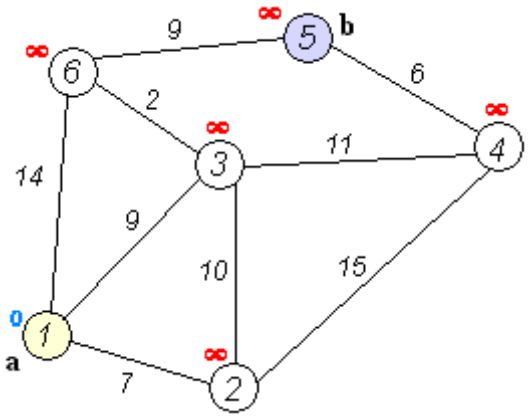
Τελικά, οι προκλήσεις και οι απαιτήσεις της σύγχρονης εποχής έχουν οδηγήσει στην ανάγκη για γρήγορο και αποτελεσματικό picking στις αποθήκες. Ο αλγόριθμος Dijkstra, με την ικανότητά του να εντοπίζει τις βέλτιστες διαδρομές, αποτελεί ένα από τα εργαλεία που μπορούν να βοηθήσουν στην αντιμετώπιση αυτών των απαιτήσεων και να διαμορφώσουν το μέλλον της αποθηκευτικής διαχείρισης.

3. Αλγόριθμος Dijkstra

3.1 Εισαγωγή

Ο αλγόριθμος του Dijkstra χρησιμοποιείται για την εύρεση της συντομότερης διαδρομής από έναν αρχικό κόμβο σε έναν κόμβο στόχο σε ένα σταθμισμένο γράφημα. Ο αλγόριθμος υπάρχει σε πολλές παραλλαγές, οι οποίες αρχικά χρησιμοποιήθηκαν για την εύρεση της συντομότερης διαδρομής μεταξύ δύο δεδομένων κόμβων. Τώρα χρησιμοποιούνται πιο συχνά για την εύρεση της συντομότερης διαδρομής από την πηγή προς όλους τους άλλους κόμβους στο γράφημα, παράγοντας ένα δέντρο συντομότερης διαδρομής.

3.2 Ανάλυση Αλγορίθμου Dijkstra



Εικόνα 3.1: Γράφος με μη αρνητικά βάρη στις ακμές

Η ανάλυση του αλγορίθμου Dijkstra αποτελεί μια σημαντική κατανόηση της λειτουργίας του και των βημάτων που ακολουθεί για την εύρεση της συντομότερης διαδρομής από έναν αρχικό κόμβο προς όλους τους υπόλοιπους κόμβους ενός γράφου. Ο αλγόριθμος Dijkstra είναι ένας γραμμικός αλγόριθμος που βρίσκει τον συντομότερο δρόμο σε έναν γράφο με μη αρνητικά βάρη σε ακμές. (Εικόνα 3.1) (Mitchel T. Keller, Alan Tucker)

Ακολουθεί η ανάλυση των βασικών βημάτων του αλγορίθμου Dijkstra:

1. **Αρχικοποίηση:** Ξεκινώντας από τον αρχικό κόμβο, αναθέτουμε σε κάθε άλλο κόμβο αρχικά άπειρη απόσταση και τον ορίζουμε ως "μη επισκεπτόμενο".

2. **Επιλογή Κόμβου:** Επιλέγουμε τον μη επισκεπτόμενο κόμβο με τη μικρότερη γνωστή απόσταση από τον αρχικό κόμβο. Αυτός ο κόμβος θα είναι ο πρώτος που θα επισκεφτούμε.

3. **Ενημέρωση Αποστάσεων:** Για τους γειτονικούς κόμβους του επιλεγμένου κόμβου, υπολογίζουμε τη συνολική απόσταση από τον αρχικό κόμβο μέσω του επιλεγμένου κόμβου. Εάν η νέα απόσταση είναι μικρότερη από την προηγούμενη γνωστή απόσταση, τότε ενημερώνουμε τη γνωστή απόσταση και τον προκάτοχο του γειτονικού κόμβου.

4. **Σήμανση Επισκεπτόμενου:** Έπειτα από την επεξεργασία όλων των γειτονικών κόμβων του επιλεγμένου κόμβου, σημαίνουμε τον επιλεγμένο κόμβο ως "επισκεπτόμενο".

5. **Επανάληψη:** Επαναλαμβάνουμε τα βήματα 2-4 μέχρι να έχουμε επισκεφτεί όλους τους κόμβους ή μέχρι να φτάσουμε σε στόχο, ενώ σε κάθε βήμα επιλέγουμε τον επόμενο μη επισκεπτόμενο κόμβο με τη μικρότερη γνωστή απόσταση.

6. **Ολοκλήρωση:** Όταν έχουμε επισκεφτεί όλους τους κόμβους ή έχουμε φτάσει στον στόχο, ο αλγόριθμος Dijkstra έχει υπολογίσει τις συντομότερες αποστάσεις από τον αρχικό κόμβο σε όλους τους υπόλοιπους κόμβους.

Εφαρμογή του αλγορίθμου με παράδειγμα

Ας δούμε ένα παράδειγμα έχοντας δεδομένα τα παραπάνω για να κατανοήσουμε καλύτερα τον αλγόριθμο του Dijkstra.

Έστω ότι έχουμε τον παρακάτω γράφο με 5 κορυφές (A, B, C, D, E) και τις εξής ακμές με βάρη:

- A -> B: 4
- A -> C: 2
- B -> C: 5
- B -> D: 10
- C -> D: 3
- C -> E: 4
- D -> E: 1

Θέλουμε να βρούμε τη συντομότερη διαδρομή από την κορυφή A σε όλες τις άλλες κορυφές.

1. **Αρχικοποίηση**: Ορίζουμε την απόσταση της A ως 0 και όλων των άλλων κορυφών ως ∞.
    - A: 0
    - B: ∞
    - C: ∞
    - D: ∞
    - E: ∞
2. **Πρώτη Επαναληπτική Διαδικασία**:
    - Επιλέγουμε την A (η μόνη με απόσταση 0).
    - Ελέγχουμε τις γειτονικές κορυφές της A:
        - Η απόσταση προς την B γίνεται 4 (A -> B).
        - Η απόσταση προς την C γίνεται 2 (A -> C).
    - Οι ενημερωμένες αποστάσεις είναι:
        - A: 0
        - B: 4
        - C: 2
        - D: ∞
        - E: ∞
3. **Δεύτερη Επαναληπτική Διαδικασία**:
    - Επιλέγουμε την C (η μικρότερη απόσταση μεταξύ των μη επισκεπτόμενων).
    - Ελέγχουμε τις γειτονικές κορυφές της C:

        - Η απόσταση προς την B δεν ενημερώνεται (C -> B είναι 5, άρα συνολικά 7, μεγαλύτερο από το 4).
        - Η απόσταση προς την D γίνεται 5 (C -> D).
        - Η απόσταση προς την E γίνεται 6 (C -> E).
    - Οι ενημερωμένες αποστάσεις είναι:
        - A: 0
        - B: 4
        - C: 2
        - D: 5
        - E: 6
4. **<u>Τρίτη Επαναληπτική Διαδικασία</u>**:
    - Επιλέγουμε την B (η μικρότερη απόσταση).
    - Ελέγχουμε τις γειτονικές κορυφές της B:
        - Η απόσταση προς την D μέσω της B είναι 14, που είναι μεγαλύτερο από το 5, οπότε δεν ενημερώνεται.
    - Οι αποστάσεις παραμένουν ως έχουν:
        - A: 0
        - B: 4
        - C: 2
        - D: 5
        - E: 6
5. **<u>Τέταρτη Επαναληπτική Διαδικασία</u>**:
    - Επιλέγουμε την D.
    - Ελέγχουμε τις γειτονικές κορυφές της D:
        - Η απόσταση προς την E μέσω της D είναι 6, ίδια με την υπάρχουσα απόσταση, οπότε δεν αλλάζει.
    - Οι αποστάσεις παραμένουν ως έχουν.
6. **<u>Ολοκλήρωση</u>**:
    - Οι αποστάσεις από την A προς τις άλλες κορυφές είναι:
        - A -> B: 4
        - A -> C: 2

- A -> D: 5
- A -> E: 6

3.3 Βελτιστοποίηση Αλγορίθμου Dijkstra

Ο αλγόριθμος Dijkstra βελτιστοποιείται με τη χρήση κατάλληλων δομών δεδομένων, όπως οι προτεραιότητες (heaps) όπως αναφέρει ο Thomas H. Cormen et al. για την αποτελεσματική επιλογή του επόμενου κόμβου με τη μικρότερη γνωστή απόσταση. Ο αλγόριθμος αυτός βρίσκει εφαρμογή σε πολλά πεδία, όπως στον τομέα της δρομολόγησης σε δίκτυα, την αναζήτηση σε μηχανές αναζήτησης και την ανάλυση δεδομένων.

Η πιο συνηθισμένη και αποδοτική υλοποίηση του αλγορίθμου του Dijkstra γίνεται με τη χρήση ενός σωρού μικρότερης τιμής (min-heap), όπως είναι η διωνυμική ή η σωρός Fibonacci. Ο σωρός επιτρέπει τη γρήγορη επιλογή της κορυφής με τη μικρότερη απόσταση και την αποτελεσματική ενημέρωση των αποστάσεων.

- **Χρόνος Εκτέλεσης**: O((V+E)logV), όπου V είναι ο αριθμός των κορυφών και E ο αριθμός των ακμών.
- **Πλεονέκτημα**: Πολύ αποδοτικό για πυκνούς και μεγάλους γράφους.
- **Μειονέκτημα**: Η χρήση και διαχείριση σωρών μπορεί να είναι πιο πολύπλοκη.

Αυτή η υλοποίηση του αλγορίθμου επιτρέπει την αποτελεσματική διαχείριση και ενημέρωση των αποστάσεων, μειώνοντας σημαντικά τον συνολικό χρόνο εκτέλεσης.

Ο αλγόριθμος Dijkstra αποτελεί ιδανική επιλογή για τη βελτιστοποίηση της διαδικασίας συλλογής σε ένα αποθηκευτικό περιβάλλον, λόγω των εξής αιτιολογιών:

1. **Βέλτιστη Διαδρομή:** Ο κύριος στόχος της διαδικασίας picking είναι η εύρεση της βέλτιστης διαδρομής για τη συλλογή εμπορευμάτων από την αποθήκη. Ο αλγόριθμος Dijkstra εξασφαλίζει την εύρεση της συντομότερης διαδρομής ανάμεσα στα ράφια, με βάση τις αποστάσεις μεταξύ των τοποθεσιών των εμπορευμάτων.

2. **Δυναμική Ενημέρωση:** Στη διαδικασία picking, οι συνθήκες μπορεί να αλλάζουν συχνά, καθώς νέες παραγγελίες μπορεί να προστίθενται ή να αφαιρούνται. Ο αλγόριθμος Dijkstra μπορεί να αντιμετωπίσει αυτήν τη δυναμική, ενημερώνοντας τις αποστάσεις και τις διαδρομές κατάλληλα όταν υπάρχουν αλλαγές.

3. **Μη Αρνητικά Βάρη:** Ο αλγόριθμος Dijkstra απαιτεί μη αρνητικά βάρη στις ακμές του γράφου. Στη διαδικασία picking, οι αποστάσεις μεταξύ των τοποθεσιών δεν μπορούν να είναι αρνητικές, καθιστώντας τον αλγόριθμο κατάλληλο για αυτήν την περίπτωση.

4. **Σύνθετο Αποθηκευτικό Δίκτυο:** Σε μεγάλα αποθηκευτικά δίκτυα, ο αριθμός των τοποθεσιών και η πολυπλοκότητα των διαδρομών μπορεί να είναι υψηλή. Ο αλγόριθμος Dijkstra μπορεί να αντιμετωπίσει αποθηκευτικά δίκτυα με πολλές τοποθεσίες και σύνθετες διαδρομές με αποδοτικότητα.

Υπάρχουν, επίσης, επιπλέον αλγόριθμοι που μπορούν να χρησιμοποιηθούν σε παρόμοιες καταστάσεις. Ας εξετάσουμε συνοπτικά κάποιους από αυτούς:

1. **Αλγόριθμος Βελλμάν-Φορντ:** Παρόμοιος με τον αλγόριθμο Dijkstra, αλλά μπορεί να αντιμετωπίσει και αρνητικά βάρη σε ακμές, αλλά με περιορισμένη απόδοση σε σύγκριση με τον Dijkstra σε θετικά βάρη.

2. **Αλγόριθμος A*:** Ο αλγόριθμος A* συνδυάζει την αναζήτηση με τον υπολογισμό του εκτιμώμενου κόστους για τον τερματισμό. Χρησιμοποιείται ευρέως σε προβλήματα που απαιτούν αναζήτηση σε χώρους καταστάσεων, όπως την προγραμματισμένη πλοήγηση.

3. **Αλγόριθμος Κοντότερου Μονοπατιού:** Είναι ένας αλγόριθμος που βρίσκει την κοντινότερη ακμή από τον τρέχοντα κόμβο, χωρίς να χρειάζεται να εκτελεστεί πλήρης διάσχιση του γράφου. Χρησιμοποιείται συχνά σε κοινωνικά δίκτυα.

4. **Αλγόριθμοι Δυαδικής Αναζήτησης:** Αν και φαίνονται παρόμοιοι με τον Dijkstra, οι αλγόριθμοι δυαδικής αναζήτησης όπως ο αλγόριθμος Αναζήτησης σε Πλάτος Πρώτο (BFS) μπορούν να εφαρμοστούν σε προβλήματα βραχυκυκλωμένων διαδρομών, παρότι δεν είναι άμεσα συγκρίσιμοι με τον Dijkstra.

Η επιλογή του κατάλληλου αλγορίθμου για τη διαδικασία picking είναι κρίσιμη για την αποτελεσματικότητα και την απόδοση του συστήματος. (Wayne L. Winston)

Στην περίπτωση της διαδικασίας picking, η επιλογή του αλγορίθμου Dijkstra έναντι άλλων αλγορίθμων έγινε για τους εξής παρακάτω λόγους:

1. **Συντομότερη Διαδρομή:** Ο αλγόριθμος Dijkstra εντοπίζει τη συντομότερη διαδρομή με βάση τα μη αρνητικά βάρη στις ακμές του γράφου. Στη διαδικασία picking, η εύρεση των βέλτιστων διαδρομών με βάση την ελάχιστη απόσταση μπορεί να είναι κρίσιμη για τη γρήγορη και αποτελεσματική εκτέλεση των παραγγελιών.

2. **Αποφυγή Αρνητικών Βαρών:** Ο Dijkstra απαιτεί μη αρνητικά βάρη στις ακμές, κάτι που είναι σύνηθες σε προβλήματα picking. Αυτό εξασφαλίζει την ασφάλεια και την ορθότητα των δρομολογήσεων.

3. **Ακρίβεια και Ευελιξία:** Ο αλγόριθμος Dijkstra παρέχει ακριβή αποτελέσματα και μπορεί να προσαρμοστεί σε ποικίλες περιπτώσεις, ανάλογα με τα βάρη και τις απαιτήσεις του προβλήματος.

4. **Διασφάλιση Καλής Υπηρεσίας:** Η συντομότερη διαδρομή στη διαδικασία picking συνεπάγεται τη γρηγορότερη εξυπηρέτηση των παραγγελιών, προσφέροντας μία ανώτερη ποιότητα υπηρεσίας στους πελάτες.

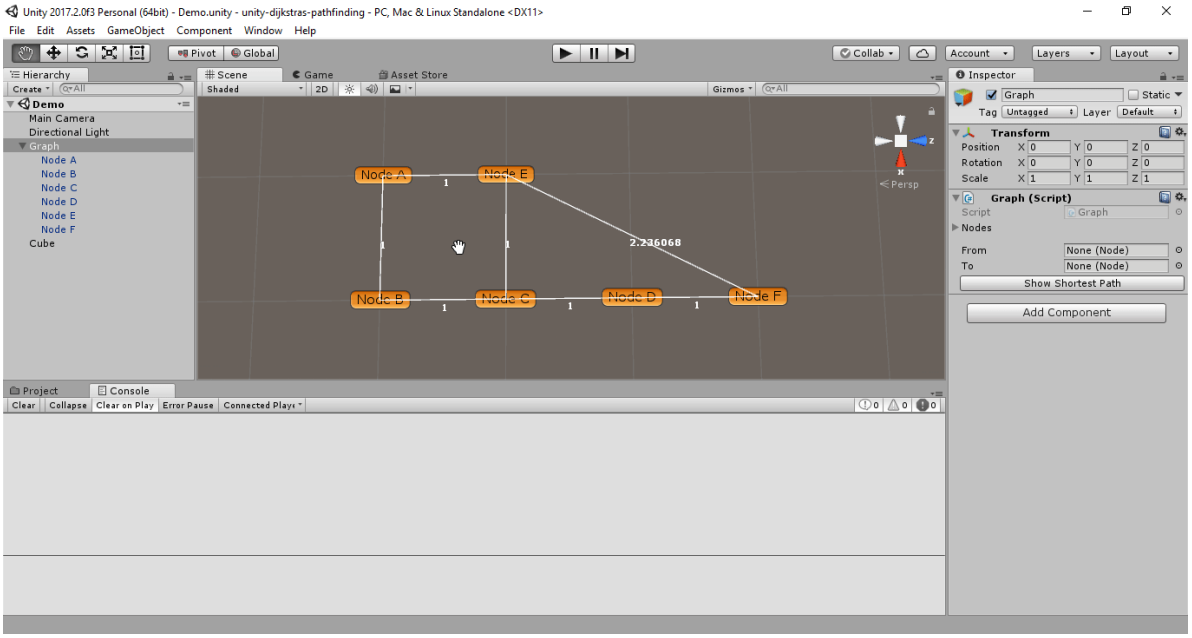
Εν κατακλείδι, ο αλγόριθμος Dijkstra είναι κατάλληλος για τη διαδικασία picking λόγω της ικανότητάς του να προσφέρει βέλτιστες και γρήγορες διαδρομές με αξιοπιστία και ακρίβεια, ανταποκρινόμενος στις απαιτήσεις της σύγχρονης εποχής για γρήγορη και αποτελεσματική αποθηκευτική διαχείριση. (Gwynne Richards, Hartmut Stadtler et al.)

3.4 Αλγόριθμος Dijkstra και Unity

Η εφαρμογή του αλγορίθμου Dijkstra στη Unity είναι ένα εξαιρετικό παράδειγμα συνδυασμού ενός αλγορίθμου εύρεσης συντομότερης διαδρομής με ένα περιβάλλον ανάπτυξης παιχνιδιών και διαδραστικών εφαρμογών. Η Unity παρέχει τα εργαλεία για να ενσωματώσουμε και να οπτικοποιήσουμε την πορεία ενός αντικειμένου μέσα από ένα γράφο, όπως απαιτείται στον αλγόριθμο του Dijkstra, κάτι που αποδεικνύεται πολύ χρήσιμο σε περιπτώσεις όπως η πλοήγηση χαρακτήρων σε ένα παιχνίδι ή σε εφαρμογές που αφορούν χάρτες και δρομολόγηση.

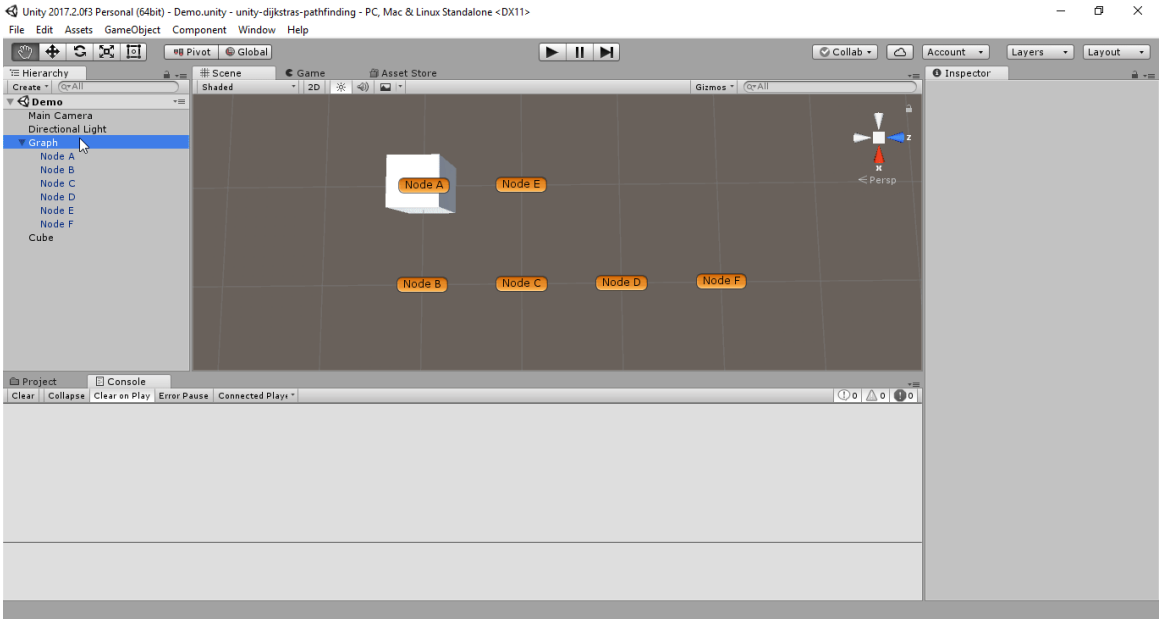
Ο αλγόριθμος Dijkstra χρησιμοποιείται για τον υπολογισμό της συντομότερης διαδρομής από έναν αρχικό κόμβο προς έναν τελικό κόμβο, λαμβάνοντας υπόψη το κόστος που απαιτείται για να περάσει από τους ενδιάμεσους κόμβους. Στο συγκεκριμένο παράδειγμα, όπως αναφέρεται

και στο forum της Unity (Unity Forum), έχουμε δύο κόμβους, τον κόμβο **A** και τον κόμβο **F**, και θέλουμε να βρούμε τη διαδρομή με το μικρότερο συνολικό κόστος.



Εικόνα 3.4.1: Dijkstra και Unity

Στη Unity, η υλοποίηση γίνεται μέσω ενός απλού τρισδιάστατου αντικειμένου, ενός **Cube**, το οποίο ακολουθεί τη διαδρομή που υπολογίζεται με τον αλγόριθμο Dijkstra. Το αντικείμενο Cube μετακινείται μέσα στον χώρο ακολουθώντας την πιο αποδοτική διαδρομή από την αρχή ως το τέλος, όπως προκύπτει από τον αλγόριθμο.(Εικόνα 3.4.2)



Εικόνα 3.4.2: Dijkstra και Unity

Δομή και Λειτουργία του Γράφου στη Unity

Για την εφαρμογή του Dijkstra στη Unity, δημιουργούμε έναν γράφο που αποτελείται από κόμβους και ακμές. Οι κόμβοι αντιπροσωπεύουν τις θέσεις στις οποίες μπορεί να μετακινηθεί το Cube, και οι ακμές που συνδέουν τους κόμβους αντιπροσωπεύουν τις διαδρομές που μπορεί να ακολουθήσει. Κάθε ακμή έχει ένα συγκεκριμένο "βάρος", το οποίο είναι το κόστος για τη μετάβαση από τον έναν κόμβο στον άλλο. Το κόστος αυτό μπορεί να είναι μια απόσταση, ο χρόνος ή οποιοδήποτε άλλο μέτρο.

Μέσα στη Unity, οι κόμβοι και οι ακμές αναπαριστώνται συνήθως με αντικείμενα και γραμμές που δημιουργούνται δυναμικά στον χώρο. Ο αλγόριθμος Dijkstra εφαρμόζεται σε αυτό το δίκτυο κόμβων και ακμών, υπολογίζοντας το μικρότερο κόστος από έναν αρχικό κόμβο προς έναν τελικό, λαμβάνοντας υπόψη τα ενδιάμεσα σημεία.

Υλοποίηση του Αλγορίθμου στη Unity

Η υλοποίηση του αλγορίθμου Dijkstra στη Unity γίνεται μέσω ενός script που καθορίζει τα βήματα που πρέπει να ακολουθήσει το Cube για να φτάσει στον προορισμό του. Το script περιλαμβάνει μερικά βασικά στοιχεία του αλγορίθμου Dijkstra:

- **Αρχικοποίηση των Κόμβων**: Κάθε κόμβος έχει μία αρχική απόσταση, η οποία τίθεται στο άπειρο (∞), εκτός από τον αρχικό κόμβο που έχει απόσταση 0.
- **Επιλογή του Κόμβου με το Μικρότερο Κόστος**: Ο αλγόριθμος επιλέγει τον κόμβο με το μικρότερο κόστος από την αρχική θέση και ενημερώνει τις αποστάσεις για τους γειτονικούς κόμβους.
- **Ενημέρωση Αποστάσεων**: Για κάθε γειτονικό κόμβο, ο αλγόριθμος ελέγχει αν η νέα απόσταση μέσω του τρέχοντος κόμβου είναι μικρότερη από την υπάρχουσα απόσταση. Αν ναι, η απόσταση αυτή ενημερώνεται.
- **Επαναληπτική Διαδικασία**: Τα παραπάνω βήματα επαναλαμβάνονται μέχρι να βρεθεί η συντομότερη διαδρομή προς τον τελικό κόμβο.

Μέσα στη Unity, αυτή η διαδικασία υλοποιείται συνήθως με τη χρήση διαφόρων κλάσεων και μεθόδων, που βοηθούν στην οργάνωση του κώδικα και τη διαχείριση του γραφήματος.

Ενσωμάτωση του Αντικειμένου Cube για την Απεικόνιση της Διαδρομής

Η χρήση του αντικειμένου Cube για την απεικόνιση της διαδρομής είναι μια οπτική προσέγγιση που επιτρέπει στους χρήστες να βλέπουν τη διαδρομή που ακολουθείται στο χώρο της Unity. Το Cube τοποθετείται αρχικά στην αρχική θέση (κόμβο A) και κινείται προς τους ενδιάμεσους κόμβους μέχρι να φτάσει στον τελικό κόμβο (κόμβο F).

Αυτή η διαδικασία γίνεται δυναμικά, επιτρέποντας στο Cube να ενημερώνει τη θέση του σε πραγματικό χρόνο. Η χρήση της Unity προσφέρει την ευκολία της τρισδιάστατης προβολής και της φυσικής κίνησης του αντικειμένου, δίνοντας στους χρήστες τη δυνατότητα να κατανοήσουν την πορεία του αλγορίθμου μέσα από οπτική αναπαράσταση.

4. Μηχανή Unity

4.1 Εισαγωγή

Η Unity είναι μια από τις πιο διαδεδομένες και ισχυρές μηχανές ανάπτυξης παιχνιδιών στον κόσμο. Από την κυκλοφορία της το 2005, έχει κερδίσει την εμπιστοσύνη πολλών προγραμματιστών, από ανεξάρτητους έως μεγάλες εταιρείες ανάπτυξης, λόγω της ευελιξίας, των εργαλείων και των δυνατοτήτων που προσφέρει. Η Unity υποστηρίζει τόσο την ανάπτυξη 2D όσο και 3D παιχνιδιών και παρέχει μια σειρά από χαρακτηριστικά που διευκολύνουν τη διαδικασία ανάπτυξης, καθιστώντας την ιδανική για πολλούς τύπους έργων.

4.2 Ιστορικό και Ανάπτυξη

Η Unity Technologies ιδρύθηκε το 2004 από τους David Helgason, Joachim Ante και Nicholas Francis, με στόχο να δημιουργήσει μια μηχανή παιχνιδιών που θα ήταν προσβάσιμη σε όλους. Από την αρχή, η Unity προγραμματίστηκε για να επιτρέψει στους προγραμματιστές να αναπτύσσουν παιχνίδια εύκολα και γρήγορα, χωρίς να χρειάζεται να έχουν προηγούμενη εμπειρία στο προγραμματισμό (Unity Technologies, 2020).

Η δημιουργία της Unity ήταν αποτέλεσμα της ανάγκης για ένα εργαλείο που θα απλοποιούσε την ανάπτυξη παιχνιδιών, επιτρέποντας στους προγραμματιστές να επικεντρωθούν στις δημιουργικές πτυχές αντί για τις τεχνικές. Πριν την Unity, η ανάπτυξη ψηφιακών παιχνιδιών απαιτούσε σύνθετες γνώσεις προγραμματισμού και τη χρήση εξειδικευμένων εργαλείων που δεν ήταν εύκολα προσβάσιμα. Η Unity έκανε τη διαδικασία πιο εύκολη, προσφέροντας μια δωρεάν έκδοση για νέους προγραμματιστές και μικρές ομάδες, δίνοντας ταυτόχρονα τη δυνατότητα για ανάπτυξη σε πολλές πλατφόρμες.

Η Unity άρχισε ως μια μηχανή αποκλειστικά για Mac, αλλά γρήγορα επεκτάθηκε για να υποστηρίξει άλλες πλατφόρμες, όπως τα Windows, τις κονσόλες και τα κινητά τηλέφωνα. Σήμερα, η Unity υποστηρίζει περισσότερες από 25 πλατφόρμες, περιλαμβάνοντας τόσο συμβατές πλατφόρμες όπως το PC, το PlayStation και το Xbox όσο και κινητές πλατφόρμες όπως το iOS και το Android (Gamasutra, 2021).

Από την πρώτη της κυκλοφορία μέχρι σήμερα, η Unity εξελίσσεται συνεχώς, ενσωματώνοντας νέα χαρακτηριστικά, δυνατότητες και εργαλεία. Σήμερα, με την υποστήριξη τόσο του 2D όσο και του 3D περιβάλλοντος, η Unity θεωρείται μία από τις κορυφαίες μηχανές ανάπτυξης παιχνιδιών παγκοσμίως, υποστηρίζοντας όχι μόνο PC και κονσόλες, αλλά και κινητές συσκευές, VR και AR εφαρμογές, web παιχνίδια και άλλα.

4.3 Χαρακτηριστικά της Unity

Πολλαπλές Πλατφόρμες

Η Unity επιτρέπει την ανάπτυξη παιχνιδιών για πολλές πλατφόρμες με μία μόνο βάση κώδικα. Αυτό σημαίνει ότι οι προγραμματιστές μπορούν να δημιουργήσουν το παιχνίδι τους και να το δημοσιεύσουν σε διαφορετικές πλατφόρμες χωρίς να χρειαστεί να ξαναγράψουν τον κώδικα. Αυτή η δυνατότητα εξοικονομεί χρόνο και πόρους και καθιστά τη Unity ιδανική επιλογή για indie developers και μεγάλες εταιρείες (Unity Technologies, 2020).

Γραφικά και 3D Σχεδίαση

Η Unity προσφέρει ισχυρές δυνατότητες γραφικών για τη δημιουργία ρεαλιστικών και εντυπωσιακών τρισδιάστατων σκηνών. Με εργαλεία όπως το Shader Graph, οι προγραμματιστές μπορούν να δημιουργήσουν σύνθετα shaders χωρίς να χρειάζεται να γράψουν κώδικα. Επιπλέον, η υποστήριξη VR (Virtual Reality) και AR (Augmented Reality) διευρύνει τις δυνατότητες ανάπτυξης σε νέες διαστάσεις (Unity Learn, 2023).

Κατάστημα Πόρων

Ένα από τα πιο δυνατά σημεία της Unity είναι το Unity Asset Store, μια πλατφόρμα όπου οι προγραμματιστές μπορούν να αγοράσουν και να πωλήσουν πόρους όπως μοντέλα, ήχους, scripts και εργαλεία ανάπτυξης. Αυτό επιτρέπει στους προγραμματιστές να εξοικονομούν χρόνο, καθώς μπορούν να χρησιμοποιήσουν έτοιμα assets αντί να τα δημιουργήσουν από την αρχή (Unity Technologies, 2020).

Φυσική και Κίνηση

Η Unity περιλαμβάνει έναν ισχυρό φυσικό κινητήρα (Physics Engine), που επιτρέπει την εύκολη προσομοίωση ρεαλιστικών κινήσεων και αλληλεπιδράσεων στον κόσμο του παιχνιδιού. Οι μηχανισμοί φυσικής υποστηρίζουν στοιχεία όπως η βαρύτητα, η τριβή και οι συγκρούσεις, κάνοντας τις κινήσεις και τις αντιδράσεις των αντικειμένων ρεαλιστικές. Αυτό καθιστά τη Unity ιδανική για παιχνίδια που βασίζονται σε φυσικά μοντέλα, όπως παιχνίδια οδήγησης, πλατφόρμες και άλλα παιχνίδια που απαιτούν ρεαλιστικές κινήσεις.

Οπτικά Εφέ και Rendering

Η Unity παρέχει εξαιρετικά εργαλεία rendering, επιτρέποντας τη δημιουργία πολύπλοκων και ρεαλιστικών γραφικών. Διαθέτει το High Definition Render Pipeline (HDRP) για υψηλής ποιότητας γραφικά σε υπολογιστές και κονσόλες και το Universal Render Pipeline (URP), που είναι κατάλληλο για φορητές συσκευές και για ανάπτυξη που απαιτεί πιο αποδοτική απόδοση. Οι προγραμματιστές μπορούν επίσης να δημιουργήσουν προσαρμοσμένα shaders και οπτικά εφέ χρησιμοποιώντας το Shader Graph, ένα εργαλείο που επιτρέπει τη δημιουργία shaders με τη χρήση γραφικού περιβάλλοντος.

Σενάριο και Λογική (Scripting)

Η Unity χρησιμοποιεί τη γλώσσα C# για την ανάπτυξη σεναρίων (scripting), κάτι που επιτρέπει στους προγραμματιστές να δημιουργούν λογική για το παιχνίδι τους. Η C# είναι μια γλώσσα προγραμματισμού ευρέως διαδεδομένη και αναγνωρισμένη, γεγονός που καθιστά εύκολη την εκμάθησή της για πολλούς προγραμματιστές (Microsoft, 2022). Η Unity παρέχει ένα καλά οργανωμένο API, με λειτουργίες και μεθόδους που διευκολύνουν τον χειρισμό αντικειμένων, τις αλληλεπιδράσεις, την κίνηση και πολλά άλλα.

Διαχείριση Δεδομένων και Αποθήκευση

Η Unity υποστηρίζει την αποθήκευση δεδομένων, επιτρέποντας τη δημιουργία συστημάτων αποθήκευσης παιχνιδιού, όπως για τη διάσωση προόδου του παίκτη, τη διαχείριση αποθήκευσης δεδομένων στον υπολογιστή και στο διαδίκτυο. Επίσης, υπάρχουν πολλές βιβλιοθήκες και επεκτάσεις που διευκολύνουν τη διαχείριση και επεξεργασία δεδομένων.

Δικτύωση και Πολλαπλοί Παίκτες

Η Unity υποστηρίζει τη δημιουργία παιχνιδιών με πολλούς παίκτες (multiplayer games) μέσω του ενσωματωμένου συστήματος δικτύωσης και προσφέρει επίσης εργαλεία και plugins για πιο προχωρημένα δίκτυα και online αλληλεπίδραση. Με αυτά τα εργαλεία, οι προγραμματιστές μπορούν να δημιουργήσουν παιχνίδια που υποστηρίζουν συνδεδεμένο παιχνίδι, online διαγωνισμούς και άλλα χαρακτηριστικά που απαιτούν δικτύωση.

Εργαλεία Επεξεργασίας και Ανάπτυξης

Η Unity περιλαμβάνει έναν εξαιρετικό editor, ο οποίος επιτρέπει την προεπισκόπηση και τη δοκιμή των παιχνιδιών σε πραγματικό χρόνο. Ο editor αυτός διευκολύνει την τροποποίηση των αντικειμένων, των υλικών και των ρυθμίσεων του παιχνιδιού, παρέχοντας μια πλήρη επισκόπηση του project.

Φιλική προς το Χρήστη Διεπαφή

Η διεπαφή της Unity είναι σχεδιασμένη να είναι φιλική προς το χρήστη, με εργαλείο drag-and-drop και δυνατότητες προεπισκόπησης που επιτρέπουν στους προγραμματιστές να βλέπουν τις αλλαγές σε πραγματικό χρόνο. Αυτή η προσέγγιση διευκολύνει τη διαδικασία ανάπτυξης και ενισχύει την παραγωγικότητα (Game Developer, 2021).

4.4 Διάρθρωση της Unity

Σκηνές

Η Unity οργανώνει το περιβάλλον ανάπτυξης σε σκηνές, οι οποίες είναι αυτοτελή περιβάλλοντα που περιλαμβάνουν αντικείμενα, φωτισμό και κάμερες. Οι σκηνές επιτρέπουν στους προγραμματιστές να οργανώνουν τα διάφορα μέρη του παιχνιδιού τους και να εργάζονται σε αυτά ξεχωριστά (Unity Manual, 2022).

Αντικείμενα και Εξαρτήματα Unity

Ο πυρήνας της Unity βασίζεται στα Game Objects, τα οποία είναι τα θεμελιώδη αντικείμενα που απαρτίζουν τη σκηνή. Κάθε αντικείμενο (Game Object) μπορεί να έχει διάφορα εξαρτήματα – μέρη (Components) που καθορίζουν τη συμπεριφορά του, όπως φυσική, ήχο ή γραφικά. Αυτή η αρχιτεκτονική επιτρέπει στους προγραμματιστές να δημιουργούν σύνθετα αντικείμενα συνδυάζοντας απλά στοιχεία (Unity Manual, 2022).

Συστήματα Φυσικής

Η Unity περιλαμβάνει ένα ισχυρό σύστημα φυσικής που προσομοιώνει τον τρόπο με τον οποίο τα αντικείμενα αλληλεπιδρούν στο περιβάλλον. Με τη χρήση 2D και 3D φυσικής, οι προγραμματιστές μπορούν να δημιουργήσουν ρεαλιστικά αποτελέσματα κίνησης και σύγκρουσης (Unity Physics Documentation, 2023).

4.5 Δημιουργία Παιχνιδιών

Σχεδιασμός Παιχνιδιού

Ο σχεδιασμός παιχνιδιού είναι μια πολύπλοκη διαδικασία που περιλαμβάνει τη δημιουργία των κανόνων και των μηχανισμών του παιχνιδιού. Με τη Unity, οι προγραμματιστές μπορούν να πειραματιστούν με διαφορετικές μηχανικές και να δημιουργήσουν πρωτότυπα παιχνίδια (K. R. Schubert, 2018).

Διαχείριση Πόρων

Η διαχείριση πόρων είναι κρίσιμη για την επιτυχία κάθε παιχνιδιού. Η Unity προσφέρει εργαλεία για την εύκολη διαχείριση και οργάνωση όλων των assets του παιχνιδιού. Οι προγραμματιστές μπορούν να κατηγοριοποιήσουν και να αναζητήσουν εύκολα τους πόρους τους, διευκολύνοντας τη διαδικασία ανάπτυξης (Unity Manual, 2022).

Δοκιμές και Βελτίωση

Η διαδικασία δοκιμών είναι καθοριστική για την ανάπτυξη παιχνιδιών. Η Unity παρέχει εργαλεία για τη δοκιμή και την ανάλυση της απόδοσης του παιχνιδιού, επιτρέποντας στους προγραμματιστές να εντοπίζουν και να διορθώνουν σφάλματα πριν από την κυκλοφορία (A. A. Ramani, 2020).

4.6 Κοινότητα και Υποστήριξη

Κοινότητα Unity

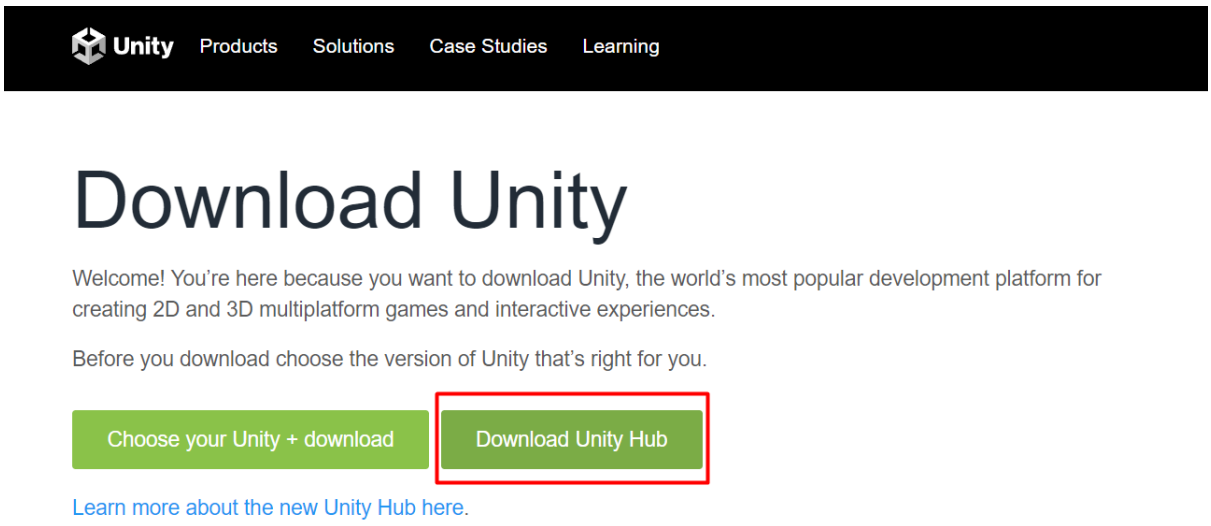
Η κοινότητα της Unity είναι μεγάλη και ενεργή, προσφέροντας πόρους, tutorials και υποστήριξη σε προγραμματιστές σε όλο τον κόσμο. Η ύπαρξη φόρουμ, ομάδων στο Discord και άλλων πηγών ενισχύει την αλληλεπίδραση μεταξύ των χρηστών και την ανταλλαγή γνώσεων (Unity Community, 2023).

Εκπαιδευτικοί Πόροι

Η Unity παρέχει μια πληθώρα εκπαιδευτικών πόρων, όπως διαδικτυακά μαθήματα, tutorials και documentation. Αυτοί οι πόροι είναι χρήσιμοι για αρχάριους και έμπειρους προγραμματιστές και διευκολύνουν την εκμάθηση των δυνατοτήτων της μηχανής (Unity Learn, 2023).

4.7 Οδηγός Εγκατάστασης

Για να εγκαταστήσουμε το Unity hub πάμε στην σελίδα https://unity3d.com/get-unity/download και κατεβάζουμε από το Downloads Section την δωρεάν έκδοση του Unity Hub. (Unity)

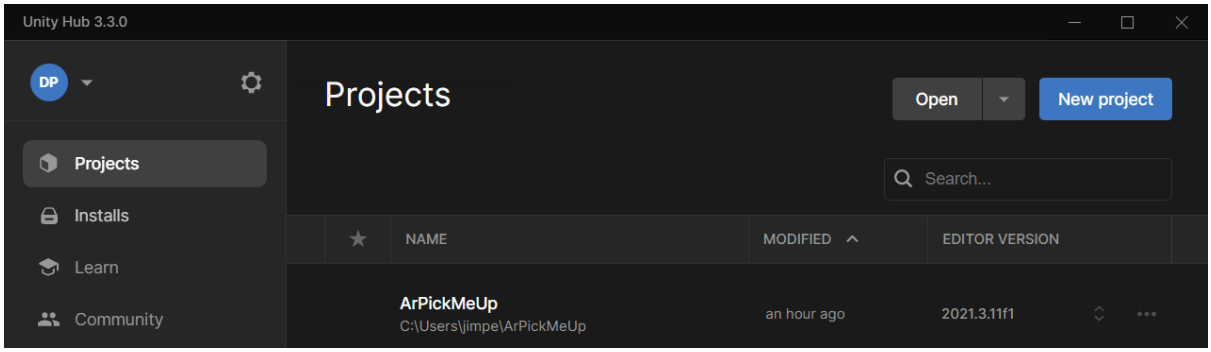


Εικόνα 4.7.1: Ιστότοπος Unity

Εφόσον το κατεβάσουμε κάνουμε την απαραίτητη εγκατάσταση που μας ζητάει και εφόσον την ολοκληρώσουμε μας ζητάει να φτιάξουμε δωρεάν λογαριασμό για να συνδεθούμε στο Unity Hub.

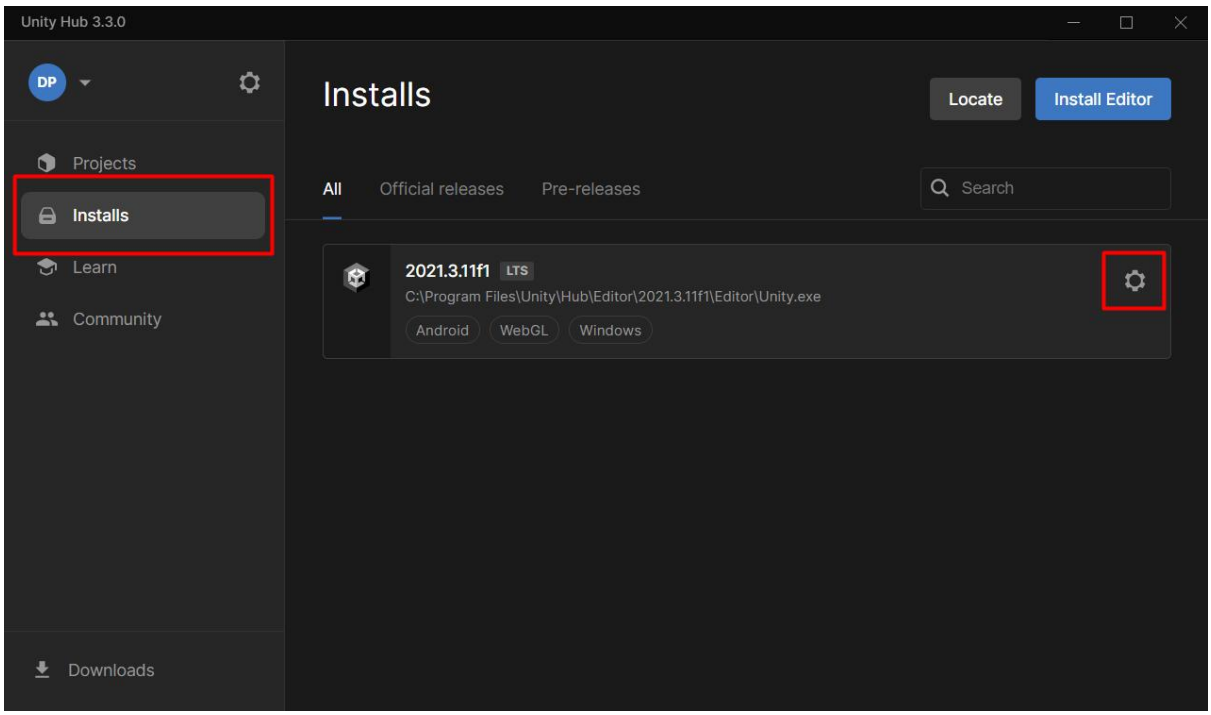
Εικόνα 4.7.2: Unity Installation

Εφόσον φτιάξουμε λογαριασμό και συνδεθούμε μας βάζει στην εικόνα 4.7.3.



Εικόνα 4.7.3: Περιβάλλον Unity Hub

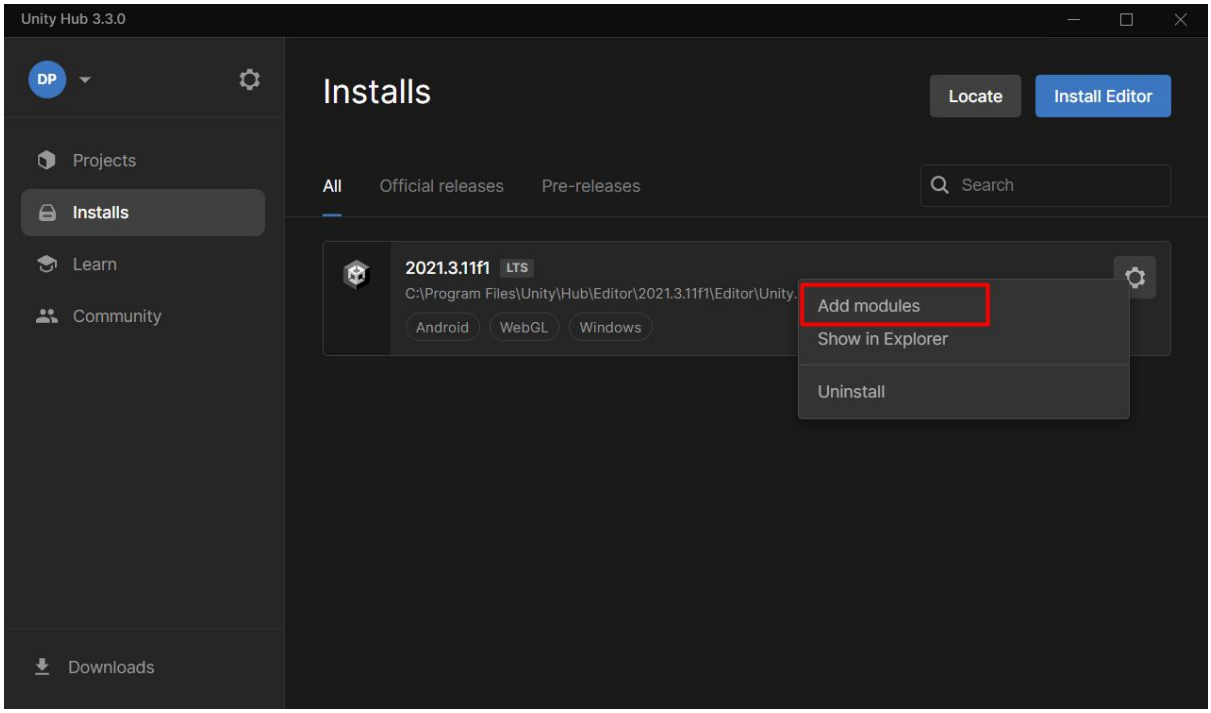
Κατά την πρώτη εγκατάσταση πάμε στο section Installs και πατάμε το γρανάζι για να μας βγάλει τις παρακάτω επιλογές



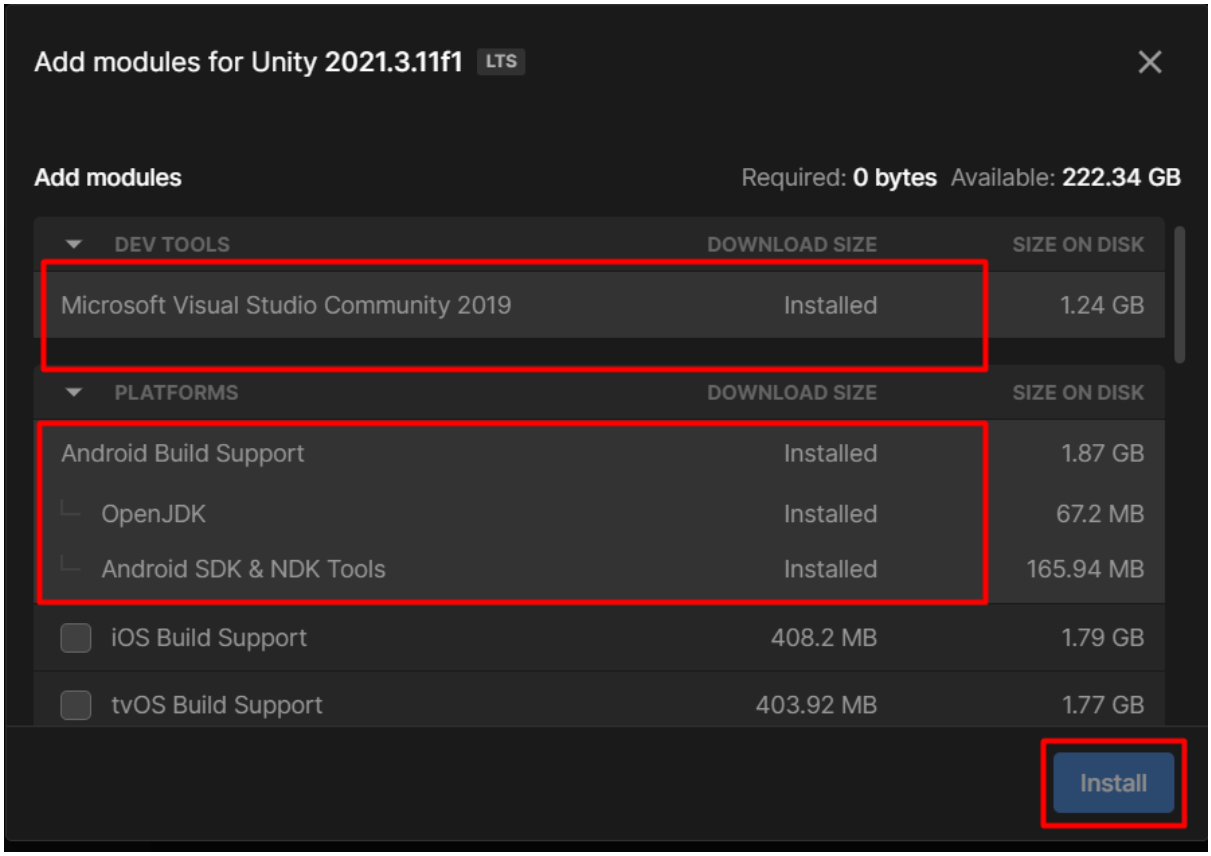
Εικόνα 4.7.4: Απεικόνιση εγκαταστάσεων στο Unity Hub

Πατώντας το γρανάζι επιλέγουμε το add modules όπως φαίνεται στην εικόνα 4.7.5.

Μας ανοίγει το Section των Modules που θέλουμε να εγκαταστήσουμε. Πατάμε να εγκαταστήσουμε την έκδοση του Visual Studio που θέλουμε και επίσης εφόσον η εφαρμογή μας θα παίζει σε Android κινητά εγκαθιστούμε ότι αφορά Android εγκατάσταση.



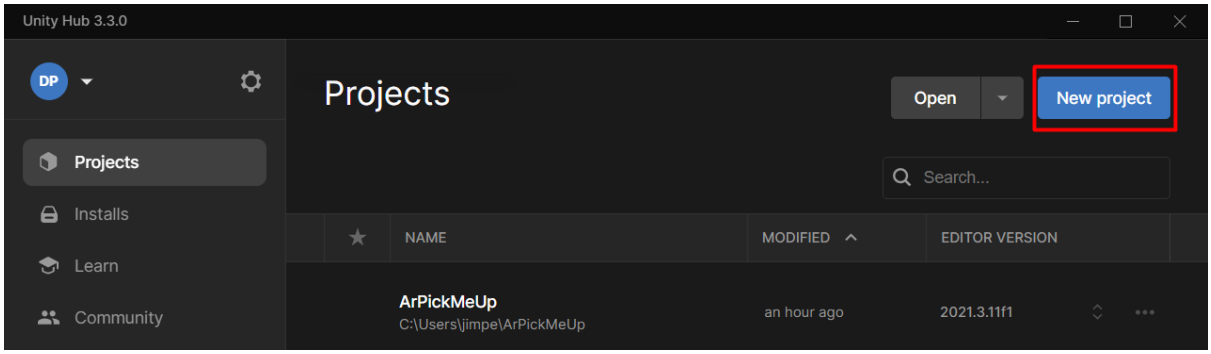
Εικόνα 4.7.5: Εγκατάσταση προσθέτων στο περιβάλλον της Unity.



Εικόνα 4.7.6: Πρόσθετα προς εγκατάσταση για την ορθή λειτουργία της εφαρμογής.

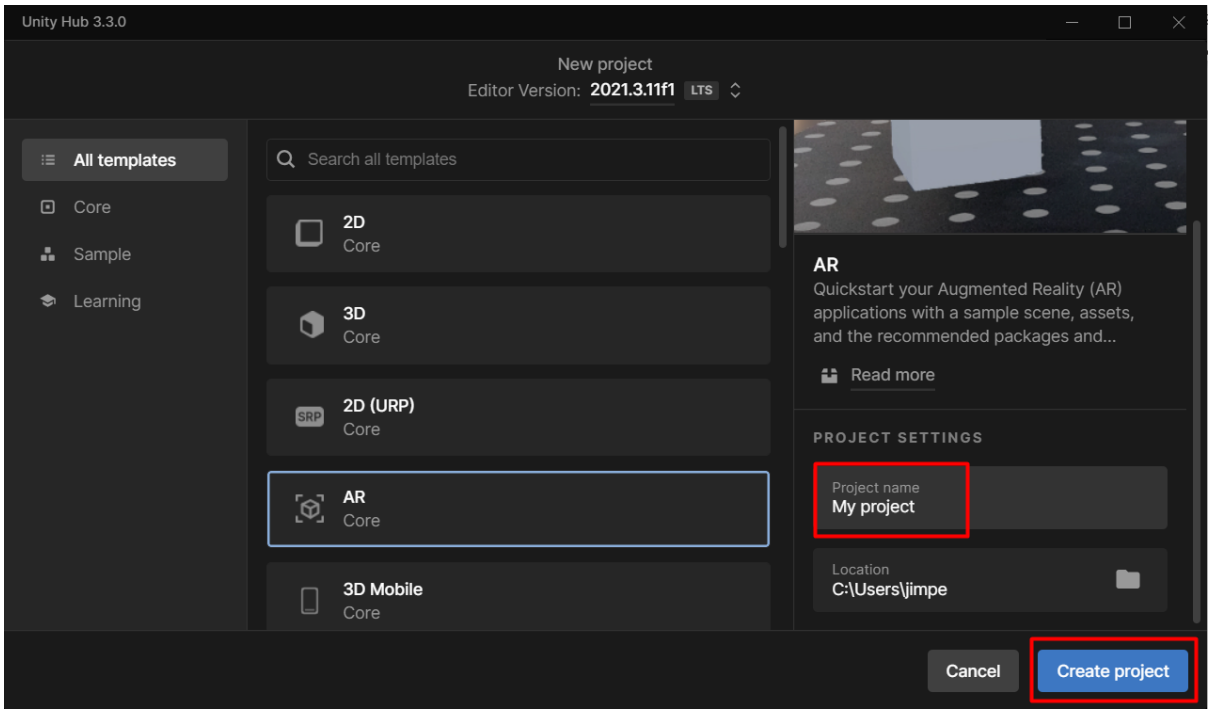
4.8 Δημιουργία Εφαρμογής

Για να δημιουργήσουμε νέο project εφόσον είμαστε στην αρχική οθόνη του Unity Hub πατάμε το κουμπί «New Project»



Εικόνα 4.8.1: Δημιουργία νέου έργου

Μας ανοίγει νέο παράθυρο έτσι ώστε να επιλέξουμε τί τύπου project θέλουμε να δημιουργήσουμε. Στην περίπτωσή μας επιλέγουμε το AR Core, βάζουμε όνομα που θέλουμε να έχει το project και πατάμε το κουμπί Create Project.



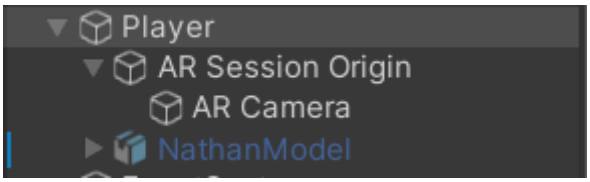
Εικόνα 4.8.2: Δημιουργία νέου έργου

5. Η Εφαρμογή μας

Το project μας, ArPickMeUp, έχει σχεδιαστεί για να προσφέρει μια διαδραστική εμπειρία συλλογής κιβωτίων μέσα σε ένα τρισδιάστατο περιβάλλον, αξιοποιώντας τις δυνατότητες της μηχανής Unity. Η εφαρμογή αυτή στοχεύει στην προσομοίωση μιας διαδικασίας συλλογής προϊόντων σε μια αποθήκη, όπου ο παίκτης έχει την ευκαιρία να αλληλεπιδράσει με το περιβάλλον και να συλλέξει κιβώτια που βρίσκονται μέσα στην εικονική μας αποθήκη. Στο παρακάτω κείμενο, θα εξετάσουμε τη διαδικασία δημιουργίας αυτής της εφαρμογής, από την αρχική σκηνή μέχρι τις μηχανισμούς αλληλεπίδρασης.

5.1 Κάμερα (Παίχτης)

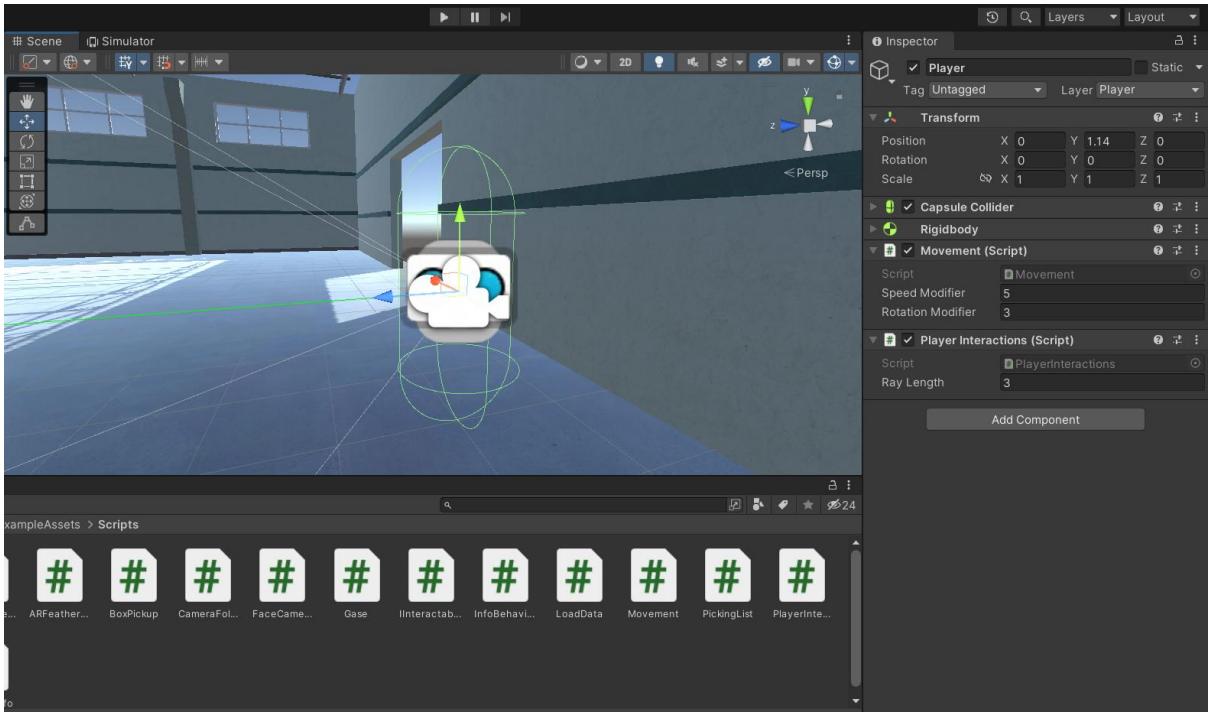
Η κάμερά μας αποτελείται και από τον «Picker» (Παίχτη) της εφαρμογής.



Εικόνα 5.1.1: Δομή κάμερας και «παίχτη» στο Unity περιβάλλον.

Ο Παίχτης μας βρίσκεται στο σημείο (0, 0, 0) και περιλαμβάνει δύο (2) Script. Το Movement script που αποτελεί την κίνηση του παίχτη μας και το Player Interactions το οποίο περιλαμβάνει τις αλληλεπιδράσεις του παίχτη. Στην δική μας περίπτωση ο παίχτης μας συλλέγει κιβώτια.

Η διαδικασία συλλογής των κιβωτίων θα περιλαμβάνει τη χρήση ενός Collider που θα ανιχνεύει όταν ο παίκτης πλησιάζει ένα κιβώτιο. Στον κάθε κιβωτίο θα προστεθεί ένας Collider με το Trigger ενεργοποιημένο. Όταν ο παίκτης εισέλθει σε αυτό το trigger, θα καλέσουμε μια μέθοδο που θα «συλλέγει» το κιβώτιο.



Εικόνα 5.1.2: Απεικόνιση κάμερας και script που είναι «δεμένα» σε αυτή.

Movement Script

Στον κώδικα που παρατίθεται παρακάτω, στην κλάση μας Movement χρησιμοποιούμε μέσα την συνάρτηση Update(), η οποία καλείται σε κάθε frame.

```
public class Movement : MonoBehaviour
{

    private Touch touch;
    [SerializeField] float speedModifier;
    [SerializeField] float rotationModifier;
    int xInput, yInput;


    void Update()
    {
        try
        {
            touch = Input.GetTouch(0);

            if (touch.phase.Equals(TouchPhase.Moved))
            {
                Vector3 cameraRelative = new Vector3(0f, 0f, touch.deltaPosition.normalized.y) * speedModifier * Time.deltaTime;
                transform.position += Camera.main.transform.TransformDirection(cameraRelative);


                transform.rotation *= Quaternion.AngleAxis(touch.deltaPosition.normalized.x, Vector3.up);
            }
        }
        catch (ArgumentException)
        {

        }
```

```
    }

    private static Quaternion GyroToUnity(Quaternion q)
    {
        return new Quaternion(q.x, q.y, -q.z, -q.w);
    }
}
```

Σε κάθε frame ουσιαστικά δημιουργούμε ένα νέο αντικείμενο τύπου Vector3 που ονομάζεται cameraRelative, το οποίο είναι ίσο με την κανονικοποιημένη τιμή y της ιδιότητας deltaPosition του αντικειμένου αφής, πολλαπλασιαζόμενη με το speedModifier και το Time.deltaTime. Οι τιμές x και z του Vector3 ορίζονται στο 0.

Η θέση του μετασχηματισμού αλλάζει προσθέτοντας τον μετασχηματισμό της κύριας κάμερας προς την κατεύθυνση του σχετικού διανύσματος της κάμερας. Το σχετικό διάνυσμα κάμερας είναι το διάνυσμα στο οποίο η κάμερα είναι σε σχέση με τον μετασχηματισμό.

Η περιστροφή του αντικειμένου πολλαπλασιάζεται με ένα τεταρτοταγές που αντιπροσωπεύει μια περιστροφή γύρω από τον άξονα y με μια γωνία ίση με τη συνιστώσα x της κανονικοποιημένης θέσης δέλτα της τρέχουσας επαφής.

Player Interactions

```
public class PlayerInteractions : MonoBehaviour
{
    [SerializeField] float rayLength;

    RaycastHit hitInfo;
    private void Update()
    {
        Ray forwardRay = new Ray(Camera.main.transform.position, Camera.main.transform.forward);
        Ray upRay = new Ray(Camera.main.transform.position + Vector3.up, Camera.main.transform.forward);
        Ray downRay = new Ray(Camera.main.transform.position - Vector3.up, Camera.main.transform.forward);

        if (Physics.Raycast(forwardRay, out hitInfo, rayLength)
            || Physics.Raycast(upRay, out hitInfo, rayLength)
            || Physics.Raycast(downRay, out hitInfo, rayLength))
        {
            Debug.Log($"Looking at {hitInfo.transform.name}");
            if (Input.GetKeyDown(KeyCode.Mouse0))
            {
                IInteractable interaction = hitInfo.transform.GetComponent<IInteractable>();

                if (interaction != null)
                {
                    interaction.PickUpInteraction();
                }
            }
        }
    }
```

```
    private void OnDrawGizmos()
    {
        Debug.DrawRay(Camera.main.transform.position, Camera.main.transform.forward * rayLength, Color.green);
    }
}
```

Δημιουργούμε ένα νέο αντικείμενο τύπου Ray, χρησιμοποιώντας τη θέση της κύριας κάμερας και την προς τα εμπρός κατεύθυνση της κύριας κάμερας ως αρχή και κατεύθυνση της ακτίνας. Το Ray μπορεί να χρησιμοποιηθεί για την ανίχνευση αντικειμένων στη σκηνή προς την κατεύθυνση που βλέπει η κάμερα.

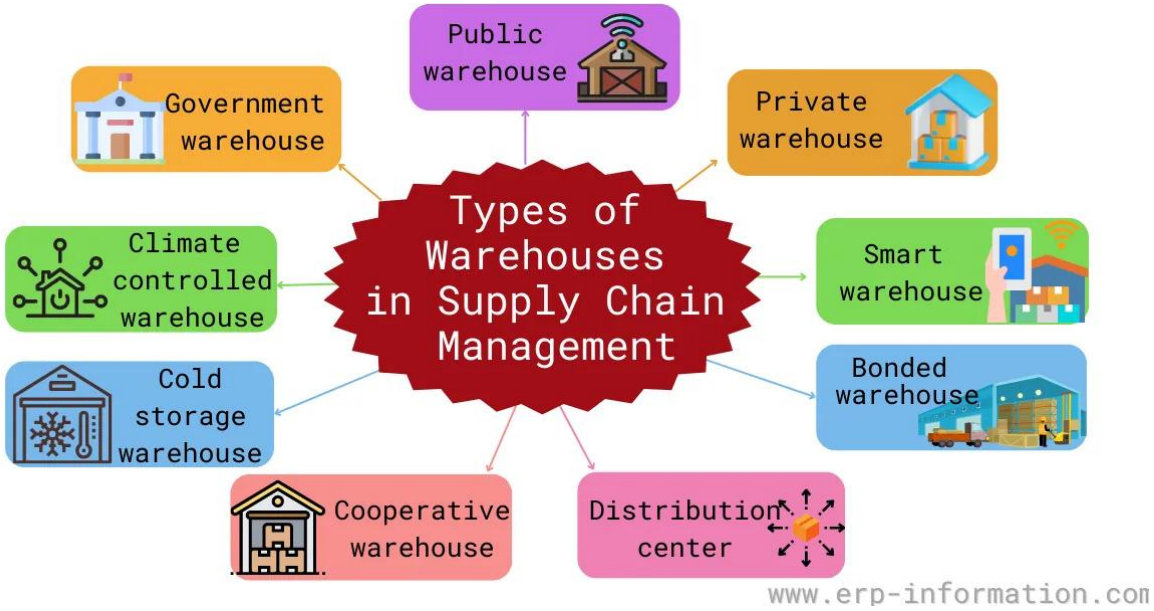
Ομοίως φτιάχνουμε και άλλα 2 αντικείμενα τύπου Ray τα οποία προσδιορίζουν ακτίνες που κοιτάνε προς τα πάνω upRay και προς τα κάτω downRay.

Και ο κώδικας ελέγχει εάν ένα Raycast (που είναι μια αόρατη γραμμή που χρησιμοποιείται για την ανίχνευση της παρουσίας αντικειμένων) «χτυπά» αντικείμενα σε μια ορισμένη απόσταση. Αν είναι υπαρκτά, εκτυπώνει τι αντικείμενο χτυπάει. Εάν ο χρήστης πατήσει το κουμπί του ποντικιού στον υπολογιστή ή αγγίξει την οθόνη του κινητού αν η εφαρμογή τρέχει σε αυτό, θα ελέγξει εάν το αντικείμενο έχει στοιχείο τύπου “IIinteractable”. Εάν το κάνει δηλαδή αν το αντικείμενο είναι τύπου “IInteractable”, θα καλέσει τη μέθοδο "PickUpInteraction" σε αυτό η οποία βρίσκεται στην κλάση BoxPickUp την οποία «κληρονομεί» κάθε κιβώτιο που είναι για συλλογή μέσα στην αποθήκη μας και θα την αναλύσουμε στο αντίστοιχο section που αφορά τα κιβώτια.

5.2 Αποθήκη

Η αποθήκη είναι μια μεγάλη εμπορική ή βιομηχανική εγκατάσταση που χρησιμοποιείται για την αποθήκευση και τη διανομή αγαθών και πρώτων υλών. Είναι συνήθως ένας μεγάλος, ανοιχτός χώρος με ψηλά ταβάνια και πολλαπλούς αποθηκευτικούς χώρους, όπως ράφια και παλέτες. Οι αποθήκες έχουν σχεδιαστεί για να φιλοξενούν τη μετακίνηση εμπορευμάτων, με αποβάθρες φόρτωσης για φορτηγά, περονοφόρα και άλλα οχήματα. Εκτός από την αποθήκευση, οι αποθήκες μπορούν επίσης να χρησιμοποιηθούν για δραστηριότητες όπως η διαχείριση αποθεμάτων, η εκπλήρωση παραγγελιών και η συσκευασία. Οι αποθήκες χρησιμοποιούνται συνήθως από κατασκευαστές, χονδρεμπόρους, λιανοπωλητές και άλλες επιχειρήσεις που εμπλέκονται στην αλυσίδα εφοδιασμού.

Τύποι αποθηκών



Εικόνα 5.2.1: Συνηθισμένοι τύποι αποθηκών.

Υπάρχουν πολλοί διαφορετικοί τύποι αποθηκών (Εικόνα 5.2.1), ο καθένας με τα δυνατά και τα αδύνατα σημεία του. Το είδος της αποθήκης που χρησιμοποιείτε θα εξαρτηθεί από τις ανάγκες της επιχείρησής. (ShipCalm)

Ας δούμε τους εννέα πιο συνηθισμένους τύπους διαχείρισης εφοδιαστικής αλυσίδας αποθήκης. Αυτοί είναι: Δημόσια αποθήκη, Ιδιωτική αποθήκη, Έξυπνη αποθήκη, Αποθήκη με Ομολογίες, Κέντρο Διανομής, Συνεταιριστική Αποθήκη, Αποθήκη Ψυχρής Αποθήκευσης, Αποθήκη Ελεγχόμενης Κλίματος και Κυβερνητική Αποθήκη.

1. **Δημόσια αποθήκη**

Είναι ένας αποθηκευτικός χώρος που ανήκει σε κρατικούς φορείς. Ένα άτομο ή ένας οργανισμός μπορεί να χρησιμοποιήσει αυτόν τον τύπο αποθήκης πληρώνοντας ενοίκιο.

Λόγω της έλλειψης τεχνολογικών δυνατοτήτων, οι δημόσιες αποθήκες τείνουν να είναι πολύ βασικές στο περιβάλλον.

Μια δημόσια αποθήκη είναι η πλέον κατάλληλη για την προσωρινή αποθήκευση των προϊόντων τους για ηλεκτρονικό εμπόριο, νεοφυείς επιχειρήσεις και βιομηχανίες μικρής κλίμακας.

Τοποθεσία: Βρίσκεται σε περιοχές εύκολα προσβάσιμες με όλα τα μέσα μεταφοράς.

Ασφάλεια: Η κυβέρνηση διασφαλίζει την ασφάλεια των αποθηκευμένων αγαθών παρέχοντας φρουρούς ασφαλείας και κάμερες.

Κόστος: Το κόστος χρήσης μιας δημόσιας αποθήκης είναι χαμηλότερο από αυτό μιας ιδιωτικής.

2. **Ιδιωτική αποθήκη**

Είναι ένας αποθηκευτικός χώρος που ανήκει σε ιδιώτες κατασκευαστές, διανομείς, χονδρέμπορους κ.λπ. Επομένως, είναι λίγο ακριβός σε σύγκριση με τις δημόσιες αποθήκες.

Ωστόσο, εξακολουθεί να είναι προσιτό για startups και βιομηχανίες μικρής κλίμακας.

Τα βασικά χαρακτηριστικά του περιλαμβάνουν:

Ευελιξία: Ο χώρος στην αποθήκη μπορεί να χρησιμοποιηθεί και να διευθετηθεί ανάλογα με τις ανάγκες του ιδιοκτήτη.

Προσαρμογή: Ο ιδιοκτήτης μπορεί να προσαρμόσει την αποθήκη σύμφωνα με τις επιχειρηματικές του ανάγκες.

Ασφάλεια: Οι ιδιωτικές αποθήκες έχουν υψηλή ασφάλεια καθώς δεν είναι ανοιχτές στο κοινό.

3. **Έξυπνη αποθήκη**

Είναι ένας προηγμένος τύπος αποθήκης στον οποίο μπορείτε να αποθηκεύετε και να διαχειρίζεστε τα αντικείμενα αυτόματα με τη βοήθεια της Τεχνητής Νοημοσύνης.

Αυτό περιλαμβάνει τη διαχείριση λογισμικού και την εκτέλεση εργασιών όπως η συλλογή, η ζύγιση, η συσκευασία, η μεταφορά και η αποθήκευση αγαθών με χρήση drones.

Έτσι, αυτός ο τύπος αποθήκης κάνει τη δουλειά εύκολη και αυξάνει τις πωλήσεις και την παραγωγικότητα μειώνοντας τα ανθρώπινα λάθη. Ως εκ τούτου, εξέχοντες πωλητές ηλεκτρονικού εμπορίου όπως η Amazon και η Alibaba χρησιμοποιούν αυτόν τον τύπο αποθήκης.

Τα βασικά χαρακτηριστικά του περιλαμβάνουν:

Αυτοματισμός: Όλες οι εργασίες σε αυτή την αποθήκη γίνονται αυτόματα με τη βοήθεια μηχανών.

Γρήγορο και αποτελεσματικό: Είναι γρήγορο και αποτελεσματικό καθώς οι μηχανές κάνουν τη δουλειά και δεν υπάρχει περιθώριο για ανθρώπινα λάθη.

Υψηλή ασφάλεια: Οι έξυπνες αποθήκες έχουν αυξημένη ασφάλεια καθώς είναι εξοπλισμένες με κάμερες CCTV και άλλα συστήματα ασφαλείας.

4. **Ομολογιακή αποθήκη**

Ένας συγκεκριμένος χώρος αποθήκευσης χρησιμοποιείται για την αποθήκευση εισαγόμενων αντικειμένων πριν από την πληρωμή των δασμών. Μπορείτε επίσης να διατηρήσετε τα εισαγόμενα προϊόντα για μεγάλο χρονικό διάστημα. Τέτοιες αποθήκες απαιτούν άδεια από την κυβέρνηση.

Το πιο κρίσιμο πλεονέκτημα αυτής της αποθήκης είναι ότι δεν χρειάζεται να πληρώσετε δασμούς έως ότου τα αντικείμενά σας απελευθερωθούν και πουληθούν, επειδή οι δασμοί για τα εισαγόμενα αντικείμενα θα είναι πολύ υψηλοί.

Τα βασικά χαρακτηριστικά του περιλαμβάνουν:

- Μπορείτε να αποθηκεύσετε εισαγόμενα προϊόντα για σύντομο ή μεγάλο χρονικό διάστημα.
- Δεν χρειάζεται να πληρώσετε δασμούς μέχρι να απελευθερωθούν τα αντικείμενα.
- Είναι κατάλληλο για την αποθήκευση υψηλής αξίας και ευαίσθητων αντικειμένων.

5. **Κέντρο διανομής**

Είναι χώρος αποθήκευσης, ειδικά για προσωρινή αποθήκευση εμπορευμάτων. Σε αντίθεση με άλλες αποθήκες, τα κέντρα διανομής ενδέχεται να διατηρούν υλικά προσωρινά ή για σύντομο χρονικό διάστημα, με πολύ υψηλότερο ποσοστό εισερχόμενων και εξερχόμενων προϊόντων.

Όταν λαμβάνετε πολλά υλικά και μπορείτε να τα διανείμετε σε διαφορετικούς μεταπωλητές στην αλυσίδα εφοδιασμού, το κέντρο διανομής βοηθάει πολύ.

Εκτός από την αποθήκευση αποθέματος, τα κέντρα διανομής διαχειρίζονται επίσης την ολοκλήρωση παραγγελιών και προετοιμάζουν αποστολές, οι οποίες αποτελούν αναπόσπαστο κομμάτι της σύνδεσης προμηθευτών και πελατών

Τα βασικά χαρακτηριστικά του περιλαμβάνουν:

- Είναι καλά εξοπλισμένο με προηγμένες τεχνολογίες.
- Παρέχει υπηρεσίες προστιθέμενης αξίας όπως υπηρεσίες pick & pack, docking κ.λπ.

- Είναι κατάλληλο για προσωρινή ή σύντομη αποθήκευση εμπορευμάτων.

6. Συνεταιριστική αποθήκη

Η συνεταιριστική αποθήκη είναι ένας αποθηκευτικός χώρος που ανήκει από κοινού και λειτουργεί από μια ομάδα εταιρειών ή συνεταιριστικών οργανώσεων όπως αγρότες, υφαντές κ.λπ. Ο κύριος σκοπός αυτού του τύπου αποθήκης είναι η εξοικονόμηση κόστους και η αύξηση της αποδοτικότητας.

Σε αυτή την αποθήκη, τόσο τα μέλη της συνεταιριστικής οργάνωσης όσο και οι ξένοι μπορούν να αποθηκεύουν τα εμπορεύματά τους. Ωστόσο, τα μέλη θα έχουν κάποια μείωση στο ενοίκιο.

Τα βασικά χαρακτηριστικά του περιλαμβάνουν:

- Σε αυτή την αποθήκη, τόσο τα μέλη της συνεταιριστικής οργάνωσης όσο και οι ξένοι μπορούν να αποθηκεύουν τα εμπορεύματά τους.
- Παρέχει έναν ιδανικό τρόπο σύνδεσης επιχειρήσεων με παρόμοια αποθέματα.
- Βοηθά τις εταιρείες να μειώσουν το κόστος, να μοιράζονται πόρους, να μειώνουν την αλληλεπικάλυψη υπηρεσιών και να συγκεντρώνουν αγοραστική δύναμη.

7. Ψυκτική αποθήκη

Είναι ένας χώρος αποθήκευσης που αποθηκεύει προϊόντα ευαίσθητα στη θερμοκρασία, όπως φάρμακα, καλλυντικά, είδη διατροφής και ποτά. Αυτός ο τύπος αποθήκης είναι ειδικά σχεδιασμένος για να διατηρεί χαμηλή θερμοκρασία.

Το κύριο πλεονέκτημα αυτής της αποθήκης είναι ότι βοηθά στη διατήρηση της ποιότητας των εμπορευμάτων για μεγάλο χρονικό διάστημα. Επιπλέον, η θερμοκρασία εντός της ψυκτικής αποθήκης διατηρείται με χρήση προηγμένων συστημάτων ψύξης.

Τα βασικά χαρακτηριστικά του περιλαμβάνουν:

- Οι αποστολές εντός και εκτός ψυκτικών αποθηκών είναι επίσης υπό ψύξη.
- Συνήθως το χρησιμοποιούν οι φαρμακευτικές εταιρείες.
- Η διαχείριση της αποθήκης πρέπει να διατηρεί το περιβάλλον κρύο με πρόσθετη συντήρηση και επιτήρηση.

8. Αποθήκη ελεγχόμενη από το κλίμα

Μια ελεγχόμενη από το κλίμα αποθήκη είναι μια εγκατάσταση αποθήκευσης όπου η θερμοκρασία και η υγρασία ρυθμίζονται προσεκτικά για την προστασία των αποθηκευμένων αντικειμένων. Οι ελεγχόμενες από το κλίμα αποθήκες συχνά αποθηκεύουν ευαίσθητα υλικά, όπως φάρμακα, ηλεκτρονικά είδη και τρόφιμα.

Η θερμοκρασία και η υγρασία σε μια ελεγχόμενη από το κλίμα αποθήκη ελέγχονται από ένα σύστημα θέρμανσης, εξαερισμού και κλιματισμού (HVAC). Το σύστημα HVAC διατηρεί τη θερμοκρασία εντός ενός καθορισμένου εύρους και διαχειρίζεται το επίπεδο υγρασίας για να αποτρέψει τη δημιουργία συμπυκνωμάτων σε αντικείμενα ή μέσα στο ίδιο το κτίριο.

Οι ελεγχόμενες από το κλίμα αποθήκες συνήθως χρησιμοποιούν μονωμένους τοίχους και οροφές για να ελαχιστοποιήσουν την απώλεια ή το κέρδος θερμότητας. Η αποθήκη μπορεί επίσης να διαθέτει συστήματα συναγερμού που ειδοποιούν το προσωπικό εάν διαφέρουν τα επίπεδα θερμοκρασίας ή υγρασίας.

Τα βασικά χαρακτηριστικά του περιλαμβάνουν:

- Η θερμοκρασία, η υγρασία και ο αερισμός πρέπει να παρακολουθούνται και να ελέγχονται.
- Τα συστήματα συναγερμού ειδοποιούν το προσωπικό σε περίπτωση αλλαγής θερμοκρασίας ή υγρασίας.

9. Κυβερνητική αποθήκη

Μια κρατική αποθήκη είναι ένα κτίριο ή συγκρότημα κτιρίων που έχει σχεδιαστεί για την αποθήκευση μεγάλων ποσοτήτων αγαθών και υλικών για λογαριασμό της κυβέρνησης. Αυτές οι αποθήκες συνήθως διαχειρίζονται μια κυβερνητική υπηρεσία ή τμήμα και μπορούν να χρησιμοποιηθούν για την αποθήκευση οτιδήποτε, από τρόφιμα και ιατρικές προμήθειες μέχρι όπλα και πυρομαχικά.

Σε ορισμένες περιπτώσεις, οι κρατικές αποθήκες μπορεί επίσης να είναι ανοιχτές στο κοινό, επιτρέποντας στους ανθρώπους να αγοράζουν προϊόντα και υλικά απευθείας.

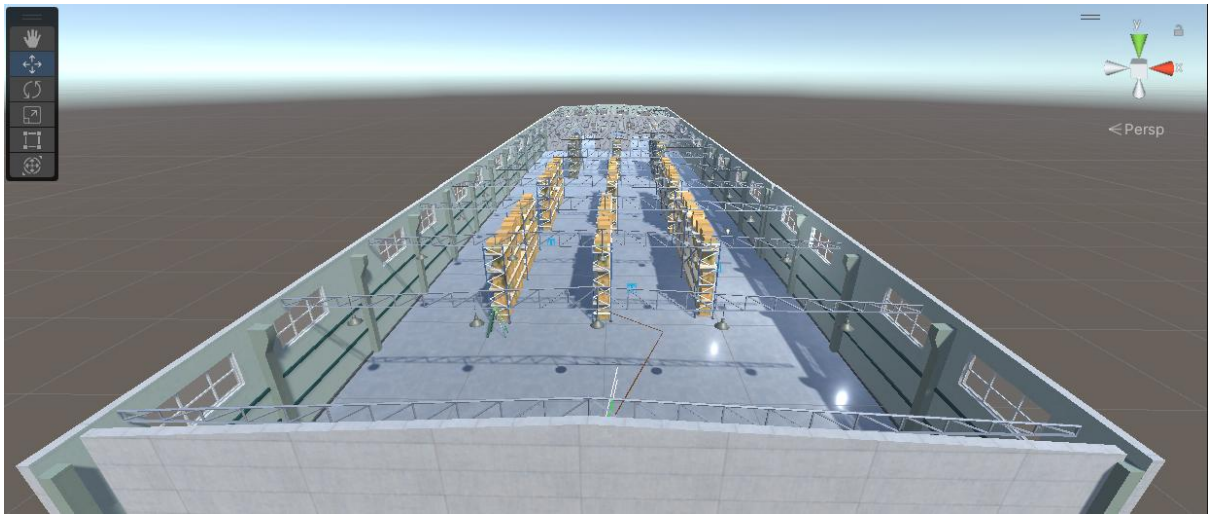
Οι κρατικές αποθήκες είναι καλύτερη επιλογή από τις δημόσιες, καθώς έχουν συνήθως μεγαλύτερη ασφάλεια, η οποία είναι κρίσιμη για ορισμένα προϊόντα

Τα βασικά χαρακτηριστικά του περιλαμβάνουν:

- Κατάλληλο για αποθήκευση μεγάλων ποσοτήτων αποθέματος.

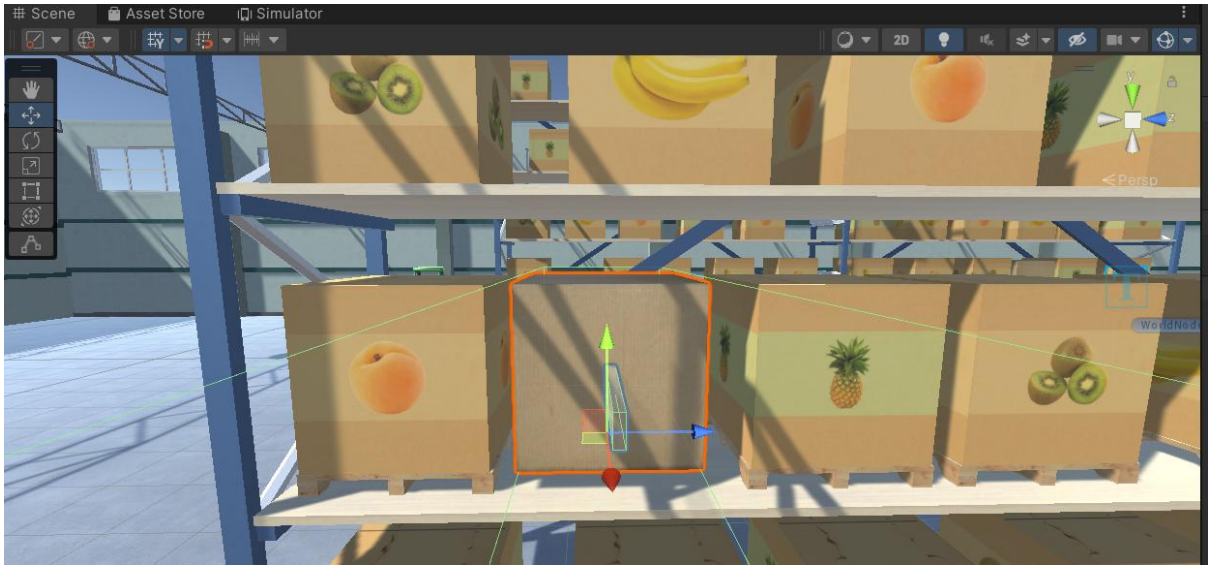
- Επιτρέπει στους ανθρώπους να αγοράζουν προϊόντα και υλικά απευθείας.
- Πιο οικονομικά αποδοτικό και παρέχει ασφάλεια.

Στο πλαίσιο της εργασίας έχουμε φτιάξει μια αποθήκη (Εικόνα 5.2.2) η οποία αποτελείται από 4 διαδρόμους και ράφια με κιβώτια τα οποία είναι στατικά. Μέσα σε αυτά τα ράφια έχουμε τοποθετήσει κιβώτια τα οποία συλλέγονται και κάνουν Interaction με τον picker (παίχτη) της εφαρμογής μας.



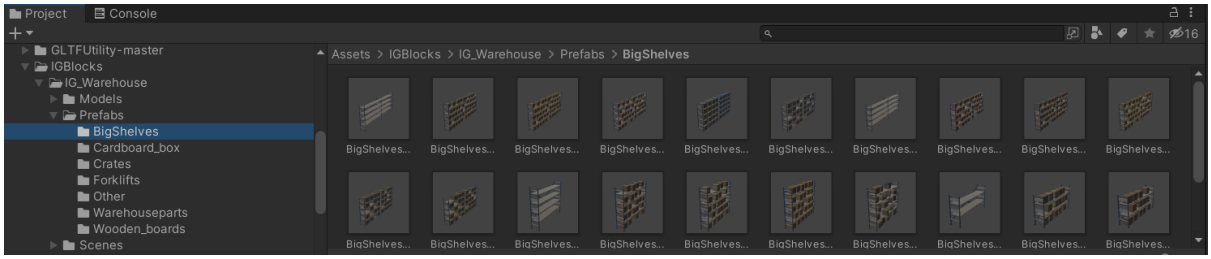
Εικόνα 5.2.2: Άποψη της αποθήκης μας στο περιβάλλον Unity.

Όπως βλέπουμε στην εικόνα 5.2.3 το κουτί που συλλέγεται ξεχωρίζει από τα στατικά κουτιά του αντικειμένου του ραφιού. (Μπήκαν κιβώτια που συλλέγονται σε κενές θέσεις των ραφιών χάριν της διαδικασίας της εφαρμογής)



Εικόνα 5.2.3: Ενεργό κουτί που μπορεί να συλλεχθεί από τον picker.

Κάθε αντικείμενο της αποθήκης μας τοποθετήθηκε από μία συλλογή αντικειμένων (Prefabs) που αφορούνε αποθήκες.



Εικόνα 5.2.4: Συλλογή αντικειμένων για την δημιουργία του «κόσμου» στο Unity.

5.3 Διαδικασία Συλλογής

Η διαδικασία συλλογής, επίσης γνωστή ως διαλογή ή παραλαβή εμπορευμάτων, αποτελεί κρίσιμο στάδιο στην αλυσίδα παροχής εφοδιασμού και τομέας της αποθήκευσης. Πρόκειται για τη διαδικασία όπου τα προϊόντα επιλέγονται από τις αποθηκευτικές θέσεις, συλλέγονται και προετοιμάζονται για την αποστολή στους πελάτες. Αν και μπορεί να φαίνεται απλή, η διαδικασία συλλογής παίζει ζωτικό ρόλο στην επιτυχημένη λειτουργία της αλυσίδας εφοδιασμού. Περιλαμβάνει την επιλογή αγαθών από τοποθεσίες αποθήκευσης για την εκπλήρωση της παραγγελίας ή την αποστολή. Συνήθως πραγματοποιείται από το προσωπικό της αποθήκης χρησιμοποιώντας συσκευές χειρός, καρότσια συλλογής ή άλλο εξοπλισμό.

Η διαδικασία συλλογής συνήθως περιλαμβάνει τα ακόλουθα βήματα:

Παραλαβή και επεξεργασία παραγγελίας: Οι παραγγελίες λαμβάνονται από πελάτες ή δημιουργούνται εσωτερικά και υποβάλλονται σε επεξεργασία για τον προσδιορισμό των ειδών, των ποσοτήτων και των απαιτήσεων παράδοσης.

Τοποθεσία και ανάκτηση αντικειμένων: Το προσωπικό της αποθήκης χρησιμοποιεί συστήματα διαχείρισης αποθεμάτων ή άλλα εργαλεία για να εντοπίσει και να ανακτήσει τα είδη που απαιτούνται για την παραγγελία.

Ενοποίηση παραγγελίας: Αφού επιλεγούν όλα τα είδη, ενοποιούνται ή ομαδοποιούνται για συσκευασία και αποστολή.

Ποιοτικός έλεγχος: Οι παραγγελίες ελέγχονται για να διασφαλιστεί ότι έχουν επιλεγεί τα σωστά είδη και ποσότητες και ότι είναι σε καλή κατάσταση.

Συσκευασία και αποστολή: Η παραγγελία συσκευάζεται και προετοιμάζεται για αποστολή, συμπεριλαμβανομένης της επισήμανσης, της τεκμηρίωσης και της φόρτωσης στο όχημα παράδοσης.

Η αποτελεσματικότητα της διαδικασίας συλλογής επηρεάζεται από πολλούς παράγοντες, μεταξύ των οποίων:

Τοπολογία της Αποθήκης: Η δομή της αποθήκης, όπως η τοποθεσία των ραφιών, οι διάδρομοι και οι περιοχές αποθήκευσης, επηρεάζει τον τρόπο που πρέπει να πραγματοποιείται η διαδικασία συλλογής. Ο σωστός σχεδιασμός της τοπολογίας μπορεί να μειώσει τον χρόνο που απαιτείται για την αναζήτηση και τη συλλογή των προϊόντων.

Μέθοδος Συλλογής: Υπάρχουν διάφορες μέθοδοι συλλογής, όπως το μονοπάτι της συλλογής (batch picking), όπου πολλά προϊόντα συλλέγονται ταυτόχρονα, και το μονοπάτι της μονάδας (piece picking), όπου τα προϊόντα συλλέγονται ένα-ένα. Η επιλογή της κατάλληλης μεθόδου επηρεάζει τον τρόπο λειτουργίας της αποθήκης.

Τεχνολογία και Εξοπλισμός: Η χρήση τεχνολογίας όπως τα αυτοματοποιημένα συστήματα συλλογής, ρομπότ και τα τροχοφόρα ράφια μπορεί να βελτιώσει σημαντικά την αποτελεσματικότητα της διαδικασίας.

Ανθρώπινος Παράγοντας: Ο ρόλος των ανθρώπων που εκτελούν τη διαδικασία συλλογής είναι κρίσιμος. Η κατάλληλη εκπαίδευση, η οργάνωση των κινήσεων και η γνώση του περιβάλλοντος της αποθήκης μπορούν να βοηθήσουν στην αύξηση της ταχύτητας και της ακρίβειας.

Χρόνος: Ο περιορισμένος χρόνος για την ολοκλήρωση της διαδικασίας συλλογής απαιτεί τον καλό σχεδιασμό και την αποτελεσματική διαχείριση των παραπάνω παραγόντων.

Ο συνδυασμός αυτών των παραγόντων επηρεάζει την απόδοση και την αποτελεσματικότητα της διαδικασίας συλλογής. Η βελτιστοποίηση της διαδικασίας προσφέρει αύξηση της παραγωγικότητας, μείωση του χρόνου που απαιτείται για την παραλαβή των προϊόντων και βελτίωση της εμπειρίας των πελατών.

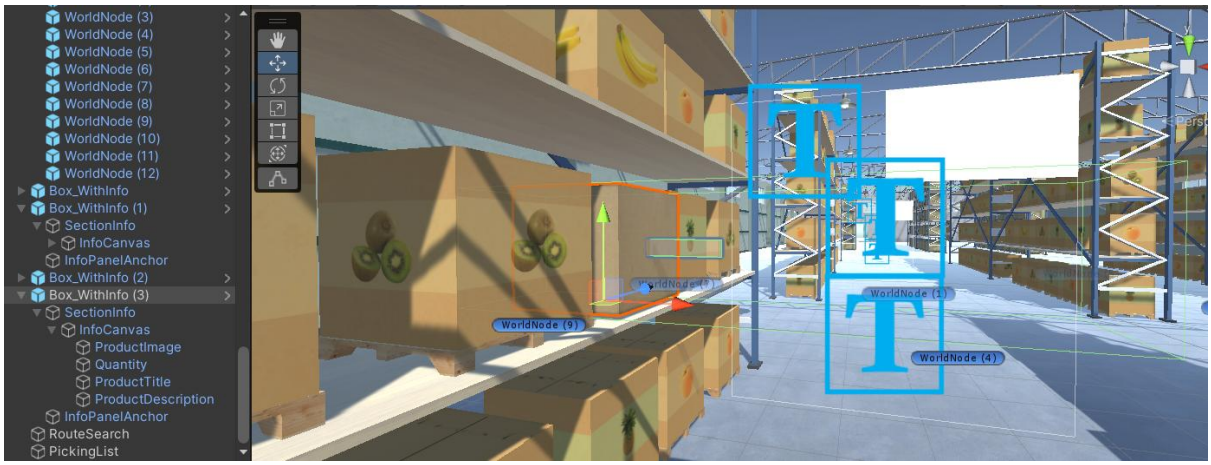
Συνολικά, η αναγνώριση και η διαχείριση αυτών των παραγόντων αποτελούν ζωτικής σημασίας κομμάτι της προσπάθειας για τη βελτιστοποίηση της διαδικασίας συλλογής σε αποθήκη, προκειμένου να επιτευχθεί υψηλή αποτελεσματικότητα και απόδοση.

Η διαδικασία συλλογής είναι κρίσιμη για τη διασφάλιση της ακριβούς και έγκαιρης εκπλήρωσης της παραγγελίας και απαιτεί προσεκτικό σχεδιασμό, οργάνωση και εκτέλεση για την ελαχιστοποίηση των σφαλμάτων και τη μεγιστοποίηση της αποτελεσματικότητας. Οι διαχειριστές αποθήκης μπορούν να χρησιμοποιήσουν μια σειρά από στρατηγικές για τη βελτιστοποίηση της διαδικασίας παραλαβής, όπως η συλλογή κατά παρτίδες, η επιλογή ζώνης ή η επιλογή κυμάτων, ανάλογα με τη φύση των εμπορευμάτων, τον όγκο της παραγγελίας και άλλους παράγοντες.

Η διαδικασία συλλογής στην εφαρμογή μας γίνεται στην διαδικασία συλλογής των κιβωτίων που το περιγράφουμε στο επόμενο τμήμα. Ο σκοπός μας άλλωστε είναι να βελτιώσουμε την διαδικασία συλλογής έτσι ώστε να γίνεται με βέλτιστο τρόπο.

5.4 Κιβώτια

Τα κιβώτια αποτελούν μέρος της αποθήκης και κάθε ένα από αυτά έχει την δικιά του λειτουργικότητα καθώς ο συλλέκτης (παίχτης) πρέπει να τα συλλέξει. Κάθε κιβώτιο έχει την δυνατότητα να συλλεχθεί και επίσης έχει την παρουσίαση του προϊόντος που υπάρχει μέσα σε αυτό.



Εικόνα 5.4.1: Δείχνει την δομή των πληροφοριών ενός κιβωτίου

Στην εικόνα 5.4.1 φαίνεται η δομή του κάθε κιβωτίου. Το κάθε κιβώτιο περιέχει τον κώδικα BoxPickUp το οποίο την στιγμή που συλλέγεται γίνεται το αντικείμενο ανενεργό (Inactive) ώστε να εξαφανιστεί από την «σκηνή» και αμέσως μετά στο κώδικα καλούμε την μέθοδο GetShortestPath περνώντας τον κόμβο του κιβωτίου και τον επόμενο συντομότερο κόμβο (Node nextPacketNode = LoadData.S.GetNextPacket();) Παρατίθεται ο κώδικας παρακάτω.

```
public class BoxPickup : MonoBehaviour, IInteractable
{
   [SerializeField] Node _ownNode;

   private void Awake()
   {
      _ownNode = GetComponent<Node>();
   }

   public void PickUpInteraction()
   {
      gameObject.SetActive(false);

      //When finding object then return to the shortest node of object
      Node nextPacketNode = LoadData.S.GetNextPacket();
```

```
        Path path = Graph.S.GetShortestPath(_ownNode.connections[0], nextPacketNode);

        FindObjectOfType<TraverseGraph>().ShowNextPath(path);
    }
}
```

Αυτός ο κώδικας ορίζει την κλάση BoxPickup, η οποία υλοποιεί τη διεπαφή IInteractable και έχει ως κύριο ρόλο την αλληλεπίδραση του παίκτη με ένα αντικείμενο στη σκηνή (κιβώτιο).

Η κλάση BoxPickup υλοποιεί τη διεπαφή IInteractable, πράγμα που σημαίνει ότι θα πρέπει να παρέχει μια συγκεκριμένη μέθοδο ή μεθόδους που να καθορίζονται από τη διεπαφή. Από τον κώδικα, φαίνεται ότι η μέθοδος PickUpInteraction είναι αυτή η υλοποίηση και καλείται όταν το αντικείμενο "αλληλεπιδράται" (όταν ο παίκτης το συλλέγει).

- Η _ownNode είναι μια αναφορά στον κόμβο Node που σχετίζεται με το αντικείμενο αυτό, δηλαδή τον κόμβο στον οποίο ανήκει το BoxPickup στον γράφο της σκηνής.
- Δηλώνεται ως SerializeField ώστε να μπορεί να ρυθμιστεί μέσω του Inspector στο Unity, αν και στη συνέχεια, αυτή η αναφορά παίρνει τιμή μέσα στον κώδικα.

Η PickUpInteraction είναι η κύρια μέθοδος αλληλεπίδρασης με το αντικείμενο. Αναλύεται ως εξής:

- **Απενεργοποίηση του αντικειμένου**: Το gameObject.SetActive(false); κρύβει το αντικείμενο από τη σκηνή, επειδή έχει συλλεχθεί από τον παίκτη.
- **Εύρεση επόμενου πακέτου**: Καλεί τη μέθοδο GetNextPacket() από την LoadData, για να βρει τον επόμενο κόμβο-πακέτο στον οποίο πρέπει να πάει ο παίκτης.
- **Εύρεση συντομότερης διαδρομής**: Χρησιμοποιεί τον Graph.S.GetShortestPath() για να βρει την συντομότερη διαδρομή από τον πρώτο κόμβο του connections στον nextPacketNode. Ο πίνακας _ownNode.connections περιέχει συνδεδεμένους κόμβους, και η [0] είναι ο πρώτος συνδεδεμένος κόμβος.
- **Εμφάνιση διαδρομής**: Εντοπίζει το component TraverseGraph στη σκηνή με FindObjectOfType<TraverseGraph>() και καλεί τη ShowNextPath(path), για να εμφανιστεί η διαδρομή που πρέπει να ακολουθήσει ο παίκτης.

Τα κιβώτια παίρνουν τις πληροφορίες (θέση και πληροφορίες είδους) από ένα json αρχείο το οποίο είναι της παρακάτω μορφής.


```
"products": [
   {
    "index": "0",
    "product_name": "Safe Products Kerala Sweet Banana Chips (500g)",
    "product_pos": {
     "x": "-0.25",
     "y": "1.82",
     "z": "22.348"
    },
    "product_qty": "10",
    "product_info": "Sweet banana chips are the most popular chips variety in Kerala, and a top of the list item for every tourist visiting Kerala. Prepared using the locally grown “Nendran” Bananas, the variant of Bananas to make Chips which is available only in Kerala.",
    "product_img": "Product_Images/banana-chips"
   }]
```


Κάθε πληροφορία του json αρχείου επεξηγείται παρακάτω:

Index: αριθμητική σειρά προϊόντος

Product_name: η ονομασία του προϊόντος

Product_pos: η θέση του κιβωτίου του προϊόντος στον «κόσμο».

Product_qty: η ποσότητα του προϊόντος που περιέχεται στο κιβώτιο

Product_info: Η περιγραφή του προϊόντος.

Product_img: Η εικόνα του προϊόντος.

5.5 Φορτωτής Προϊόντων

Εφόσον γίνει η φυσική φόρτωση της αποθήκης πρέπει να γίνει απεικόνιση των κιβωτίων με τις πληροφορίες του κάθε είδους όπως αναφέρονται στο προηγούμενο τμήμα στο πρόγραμμά μας.

Παρατίθεται κομμάτι κώδικα που διαβάζει το .Json αρχείο με τις κατάλληλες πληροφορίες έτσι ώστε να δημιουργηθούν τα αντίστοιχα αντικείμενα και να μετασχηματιστούν (transform) στην «σκηνή» μας.

```
public class LoadData : MonoBehaviour
{
   public static LoadData S;

   [SerializeField] ProductList plist;
   [SerializeField] GameObject[] objcube;
```

```
    string jsonFilePath = string.Empty;
    private string jsonString;
    int currentPacket = -1;

    private void Awake()
    {
        S = this;
        jsonFilePath = Application.dataPath + "/Plugins/data.json";

        Debug.Log(jsonFilePath);
    }

    // Start is called before the first frame update
    void Start()
    {
        ProductList productList = Load();
        for (int i = 0; i < objcube.Length; i++)
        {
            objcube[i].transform.position = productList.products[i].product_pos;
        }
    }

    public Node GetNextPacket()
    {
        ++currentPacket;

        if (currentPacket >= objcube.Length)
            Debug.Log("No more packets to retrieve");

        return objcube[currentPacket].GetComponent<Node>();
    }

    private ProductList Load()
    {
        //Load
        if (File.Exists(jsonFilePath))
        {
            jsonString = File.ReadAllText(jsonFilePath);
            Debug.Log("File Loaded...");

            plist = JsonUtility.FromJson<ProductList>(jsonString);
            /* Debug.Log(plist.products[0].product_pos);
            Debug.Log(plist.products[0].product_img); */
            return plist;
        }
        else
        {
            Debug.Log("No Load...");
            return null;
        }
    }

    public ProductList ProductList => plist;

    private void OnDestroy()
    {
        S = null;
    }
}


[System.Serializable]
```

```
public class ProductList
{
    public List<Product> products;
}

[System.Serializable]
public class Product
{
    public string product_name;
    public Vector3 product_pos;
    public int product_qty;
    public string product_info;
    public string product_img;
}
```

Ανάλυση Κώδικα

Η κλάση Load Data κληρονομεί από την κλάση MonoBehaviour, κάτι που της επιτρέπει να εκμεταλλεύεται τις βασικές δυνατότητες του Unity για scripting (όπως τα events Awake, Start και OnDestroy).

public static LoadData S;

Η μεταβλητή S χρησιμοποιείται ως ένα στατικό singleton, δηλαδή μια αναφορά προς το μοναδικό instance της κλάσης LoadData που υπάρχει στη σκηνή. Αυτό διευκολύνει την πρόσβαση στην κλάση LoadData από άλλες κλάσεις χωρίς να χρειάζεται αναζήτηση μέσω του Unity API (π.χ. με FindObjectOfType<LoadData>()).

```
[SerializeField] ProductList plist;
    [SerializeField] GameObject[] objcube;

    string jsonFilePath = string.Empty;
    private string jsonString;
    int currentPacket = -1;
```

- Η μεταβλητή plist είναι μια λίστα αντικειμένων τύπου Product, τα οποία θα φορτώνονται από ένα αρχείο JSON.
- objcube: πίνακας από GameObject, που αναμένεται να περιέχει κύβους ή άλλα αντικείμενα της σκηνής.
- jsonFilePath: Η διαδρομή του αρχείου JSON από το οποίο θα φορτώνονται τα δεδομένα.
- jsonString: Σειρά χαρακτήρων που αποθηκεύει το περιεχόμενο του αρχείου JSON.
- currentPacket: Δείκτης για παρακολούθηση του τρέχοντος "πακέτου" (αντικειμένου) στον πίνακα objcube.

Η μέθοδος Awake εκτελείται κατά την εκκίνηση της κλάσης, αρχικοποιώντας την στατική μεταβλητή S με την τρέχουσα αναφορά. Ρυθμίζει επίσης τη διαδρομή του αρχείου JSON.

Η μέθοδος Start εκτελείται αμέσως μετά την μέθοδο Awake και εδώ:

- Φορτώνει δεδομένα από το αρχείο JSON μέσω της Load().
- Ενημερώνει τη θέση κάθε αντικειμένου στον πίνακα objcube με βάση τη θέση που καθορίζεται στο product_pos από τα προϊόντα του productList.

Η μέθοδος GetNextPacket χρησιμοποιείται για να επιστρέφει το επόμενο αντικείμενο από τον πίνακα objcube. Αφού αυξήσει τον δείκτη currentPacket, ελέγχει αν έχει φτάσει στο τέλος του πίνακα και εμφανίζει ένα μήνυμα αν δεν υπάρχουν άλλα αντικείμενα. Στη συνέχεια, επιστρέφει το Node component του τρέχοντος αντικειμένου στον πίνακα.

Η μέθοδος Load φορτώνει το περιεχόμενο του JSON αρχείου. Εάν το αρχείο υπάρχει, το διαβάζει και το αποθηκεύει στην plist μέσω του JsonUtility.FromJson, και επιστρέφει τη λίστα προϊόντων, διαφορετικά, επιστρέφει null και εμφανίζει μήνυμα σφάλματος.

Εικόνα 5.5.1: Εμφανίζει τις πληροφορίες του είδους που περιέχονται στο κιβώτιο το οποίο μπορεί να συλλεχθεί

Εικόνα 5.5.2: Εμφανίζει όπως και στην 5.5.1 στοιχεία του είδους όπως εικόνα, ποσότητα, τίτλος και περιγραφή.

5.6 Κόμβοι

Ο «κόσμος» μας, δηλαδή η αποθήκη, περιλαμβάνει κόμβους. Οι κόμβοι βρίσκονται στους διαδρόμους της αποθήκης και ενώνονται μεταξύ τους δημιουργώντας το μονοπάτι στο οποίο θα κινηθεί και θα ακολουθήσει ο picker έτσι ώστε να συλλέξει τα κιβώτια.

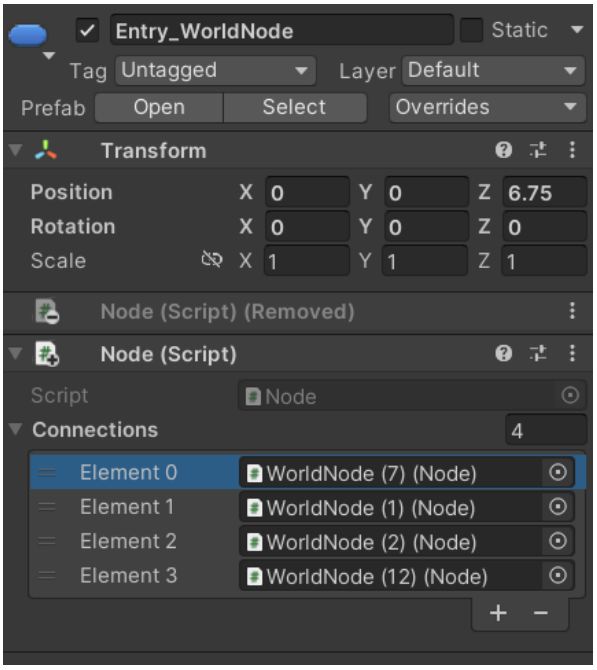
Έτσι λοιπόν έχουμε τον αρχικό κόμβο «Entry_WorldNode» από τον οποίο ξεκινάει ο συλλέκτης (picker) όπως φαίνεται στην εικόνα 5.6.1.

Ο αρχικός κόμβος, στην εφαρμογή μας χαρακτηρίζεται ως «Entry_WorldNode», αποτελεί το σημείο εκκίνησης για τον συλλέκτη. Από αυτόν τον κόμβο ξεκινούν όλες οι διαδρομές, οι οποίες αναπτύσσονται μέσα στην αποθήκη. Η σύνδεση του «Entry_WorldNode» με τους κοντινούς του κόμβους, δηλαδή εκείνους που βρίσκονται στην αρχή των διαδρόμων της αποθήκης, είναι κρίσιμη για την εύρεση των βέλτιστων διαδρομών.

Εικόνα 5.6.1: Κάτοψη κόμβων αρχικού (Entry_WorldNode) και γειτονικών αυτού.

Ο αρχικός κόμβος ενώνεται με όλους τους κοντινούς του κόμβους και φτιάχνει όλους τους πιθανούς συνδυασμούς έτσι ώστε να υπολογιστεί η βέλτιστη διαδρομή μεταξύ των κόμβων.

Η διαδικασία δημιουργίας των διαδρομών περιλαμβάνει την ένωση του αρχικού κόμβου με όλους τους γειτονικούς του κόμβους. Κάθε κόμβος μπορεί να συνδεθεί με τους τέσσερις κοντινότερους κόμβους του, οι οποίοι μπορεί να βρίσκονται αριστερά, δεξιά, μπροστά ή πίσω. Αυτή η αρχή είναι ο πυρήνας της οργάνωσης της αποθήκης και βοηθά στην εύρεση των βέλτιστων διαδρομών.



Εικόνα 5.6.2: Ένωση του αρχικού κόμβου με του γειτονικούς σε αυτόν κόμβους.
(Συνδέσεις με κόμβους)

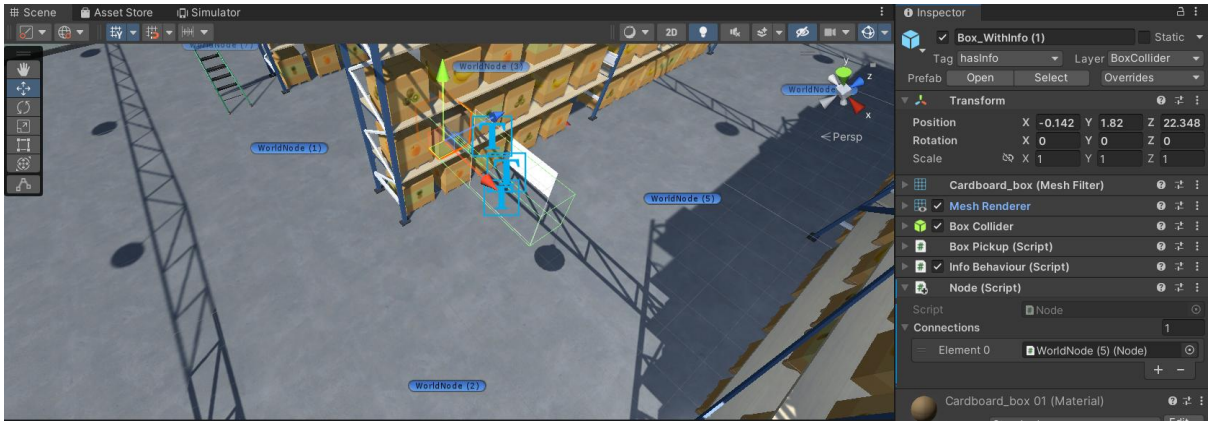
Οι κόμβοι με τους οποίους ενώνεται ο αρχικός κόμβος θα ενωθούν με την σειρά τους με τους κοντινούς τους κόμβους συμπεριλαμβανομένου και του αρχικού εφόσον αυτός είναι στους κοντινούς τους κόμβους.

Με αυτό τον τρόπο ενώνουμε όλους τους κόμβους που βρίσκονται στην αποθήκη.

Κιβώτια ως Κόμβοι

Εκτός από τους κόμβους που αναπαριστούν τις θέσεις αποθήκευσης, τα κιβώτια που περιέχουν τα προϊόντα αποτελούν επίσης κόμβους στο γράφο. Είναι σημαντικό να υπολογίζεται η βέλτιστη διαδρομή για την πρόσβαση σε κάθε κιβώτιο, καθώς αυτή η πληροφορία μπορεί να βελτιώσει τη διαδικασία συλλογής.

Κάθε κιβώτιο συνδέεται με τον κοντινό κόμβο του που βρίσκεται στον διάδρομο. Αυτή η σύνδεση επιτρέπει στον συλλέκτη να κινείται από τον κύριο κόμβο της αποθήκης προς τα κιβώτια, ακολουθώντας τις βέλτιστες διαδρομές που έχουν υπολογιστεί. Σημαντική είναι η δυνατότητα ενσωμάτωσης αλγορίθμων για τη βελτίωση της διαχείρισης αποθεμάτων και της διαδικασίας συλλογής. (Εικόνα 5.6.3)



Εικόνα 5.6.3: Απεικόνιση κιβωτίου που αποτελεί και κόμβο στον γράφο.

6. Συμπεράσματα

Η εφαρμογή του αλγορίθμου Dijkstra για τη βέλτιστη διαδικασία συλλογής σε μια αποθήκη μέσω του περιβάλλοντος ανάπτυξης παιχνιδιών Unity ανοίγει νέους ορίζοντες για τη βελτιστοποίηση των διαδικασιών συλλογής σε εφοδιαστικές αλυσίδες. Η ισχύς του αλγορίθμου Dijkstra σε συνδυασμό με τις δυνατότητες του Unity δημιουργεί ένα εργαλείο που μπορεί να έχει σημαντική επίδραση στην αποτελεσματικότητα, την οικονομία και την αειφορία των διαδικασιών συλλογής σε αποθήκες.

Συγκεκριμένα δίνεται η δυνατότητα για:

Καινοτομία στην Βελτιστοποίηση: Η ενσωμάτωση του αλγορίθμου Dijkstra σε μια εφαρμογή σε Unity για τη βέλτιστη διαδικασία συλλογής είναι ένα παράδειγμα πρωτοποριακής εφαρμογής τεχνολογίας σε επιχειρησιακά προβλήματα.

Βελτίωση Επιδόσεων: Η εφαρμογή του αλγορίθμου Dijkstra μειώνει τον χρόνο που απαιτείται για τη διαδικασία συλλογής, βελτιώνοντας έτσι την επίδοση της αποθήκης και μειώνοντας τον κίνδυνο σφαλμάτων.

Εξοικονόμηση Πόρων: Μειώνοντας τις αποστάσεις που διανύονται, η εφαρμογή συμβάλλει στην εξοικονόμηση καυσίμων και πόρων, μειώνοντας τα κόστη λειτουργίας.

Εκπαίδευση και Κατάρτιση: Η ανάπτυξη μιας τέτοιας εφαρμογής μπορεί να λειτουργήσει ως εκπαιδευτικό εργαλείο για τους εργαζόμενους, εκπαιδεύοντάς τους για τη βέλτιστη χρήση των διαδρομών στην αποθήκη.

Μεταφορά Τεχνολογίας: Η εμπειρία από την ανάπτυξη αυτής της εφαρμογής μπορεί να μεταφερθεί και να εφαρμοστεί σε άλλα πεδία, επεκτείνοντας τις δυνατότητες και την εφαρμογές της τεχνολογίας αυτής.

Σε γενικές γραμμές, η εφαρμογή του αλγορίθμου Dijkstra για τη βέλτιστη διαδικασία συλλογής σε μία αποθήκη μέσω του Unity συνδυάζει την τεχνολογία και τη διοίκηση αποθήκης για να επιτύχει αυξημένη αποτελεσματικότητα, αειφορία και επίδοση στη διαχείριση αποθήκης. Μέσω αυτής της εφαρμογής, είμαστε σε θέση να εξερευνήσουμε τη δύναμη της τεχνολογίας για την βελτίωση των επιχειρησιακών διαδικασιών και την ανάδειξη νέων διαδραστικών προσεγγίσεων για την αποτελεσματική διαχείριση των αποθηκευτικών διαδικασιών.

7. Προτάσεις Για Μελλοντική Έρευνα

Σε αυτήν την ενότητα, θα εξετάσουμε διάφορες προοπτικές για μελλοντική έρευνα και ανάπτυξη, που μπορούν να εμπλουτίσουν και να επεκτείνουν το θέμα της εφαρμογής του αλγορίθμου Dijkstra για τη βέλτιστη διαδικασία συλλογής σε μία αποθήκη.

Αναβαθμίσεις του Αλγορίθμου Dijkstra: Μία πιθανή κατεύθυνση είναι η μελέτη και η ανάπτυξη βελτιωμένων εκδοχών του αλγορίθμου Dijkstra. Μπορεί να εξεταστεί η ενσωμάτωση τεχνικών για την αντιμετώπιση περιπλοκοτήτων όπως η χρήση προτεραιοτήτων, τον εντοπισμό βέλτιστων μονοπατιών για πολλούς στόχους, ή η χρήση εξελικτικών αλγορίθμων για την βελτιστοποίηση του χρόνου εκτέλεσης.

Εφαρμογή σε Πραγματικό Χρόνο: Μία προκλητική προοπτική είναι η μετάβαση από την προσομοίωση σε πραγματικό χρόνο. Η ανάπτυξη ενός συστήματος που εφαρμόζει τον αλγόριθμο Dijkstra σε πραγματικό περιβάλλον αποθήκης, λαμβάνοντας υπόψη την πραγματική διαθεσιμότητα των δρόμων, των εμποδίων και των αλληλεπιδράσεων, θα αποτελούσε σημαντική εξέλιξη.

Βελτιστοποίηση με Μηχανική Μάθηση: Η εφαρμογή τεχνικών μηχανικής μάθησης, όπως τα νευρωνικά δίκτυα, μπορούν να βελτιώσουν την προσαρμογή του αλγορίθμου Dijkstra στις πολύπλοκες συνθήκες ενός πραγματικού περιβάλλοντος αποθήκης. Με την ανάλυση των δεδομένων της λειτουργίας της αποθήκης, μπορούν να προβλεφθούν προτάσεις για βέλτιστες διαδρομές.

Ανάπτυξη Εφαρμογών Ενδοεπικοινωνίας: Η ενσωμάτωση λειτουργιών ενδοεπικοινωνίας μεταξύ των εργαζομένων και της εφαρμογής μπορεί να βελτιώσει την αποτελεσματικότητα της διαδικασίας συλλογής. Εργαλεία όπως ειδοποιήσεις, προβολή πληροφοριών και αλληλεπίδραση στην πραγματικότητα ενδυναμώνουν την αποδοτικότητα των διαδικασιών.

Συγκεκριμένα θα μπορούσε να ερευνηθεί περισσότερο η αποδοτικότητα του αλγορίθμου Dijkstra έναντι κάποιου ή κάποιων εναλλακτικών αλγορίθμων όπως παραδείγματος χάρη οι αλγόριθμοι που αναφέρονται στην παρούσα εργασία ορίζοντας σαφή κριτήρια, διατηρώντας σταθερές κάποιες μεταβλητές και μετρώντας την αποδοτικότητα ποσοτικά. Για παράδειγμα, σε μία δεδομένη αποθήκη Χ τ.μ. με δεδομένη την διαμόρφωσή της και τον εξοπλισμό της να μετρηθεί ο χρόνος απόκρισης (σχεδιασμός διαδρομής) και ο χρόνος ολοκλήρωσης μίας διαδρομής. Ενδιαφέρον θα είχε να συγκρίνουμε την αποδοτικότητα ενός συστήματος

διατηρώντας σταθερό το χρόνο απόκρισης και ολοκλήρωσης της διαδρομής μεταβάλλοντας τα τετραγωνικά μέτρα ή και την διάταξή της.

Με βάση την βιβλιογραφία ο χρόνος σχεδιασμού μιας διαδρομής από τον αλγόριθμο Dijkstra θα μπορούσε να βελτιωθεί σημαντικά χρησιμοποιώντας μηχανισμούς Cashing για τους υπολογισμούς των διαδρομών. Αυτή είναι μία τεχνική η οποία θα μπορούσε να ελαχιστοποιήσει την κατανάλωση πόρων του συστήματος όπως επεξεργαστική ισχύς και μνήμη.

Συνολικά, η Μελλοντική Εργασία αναδεικνύει τον ρόλο της συνεχούς έρευνας και ανάπτυξης στον τομέα της βελτιστοποίησης της διαδικασίας συλλογής. Μέσα από την εξερεύνηση αυτών των προοπτικών, μπορούν να δημιουργηθούν νέες προσεγγίσεις και λύσεις που θα ενισχύσουν την αποτελεσματικότητα, την ταχύτητα και την ακρίβεια των διαδικασιών παράδοσης στον τομέα της αποθήκης.

Βιβλιογραφία

1. Frederick S. Hillier and Gerald J. Lieberman (Ninth Edition), *"Introduction to Operations Research"*, http://www.mim.ac.mw/books/Introduction%20to%20Operations%20Research%209th%20edition.pdf

2. Gwynne Richards (2nd Edition), *"Warehouse Management: A Complete Guide to Improving Efficiency and Minimizing Costs in the Modern Warehouse"*, http://dspace.vnbrims.org:13000/xmlui/bitstream/handle/123456789/4567/Warehouse%20Management%20A%20Complete%20Guide%20to%20Improving%20Efficiency%20and%20Minimizing%20Costs%20in%20the%20Modern%20Warehouse.pdf?sequence=1

3. Wayne L. Winston (Fourth Edition),*"Operations Research: Applications and Algorithms"*, https://books.google.gr/books?hl=el&lr=&id=Y9NYEAAAQBAJ&oi=fnd&pg=PR1&dq=Operations+Research:+Applications+and+Algorithms+pdf&ots=2XeKj2h73M&sig=eOeUzCkY_b8lgmKuEgpD-yCCwDA&redir_esc=y#v=onepage&q=Operations%20Research%3A%20Applications%20and%20Algorithms%20pdf&f=false

4. Mitchel T. Keller and William T. Trotter (2017), *"Applied Combinatorics"*, https://www.rellek.net/book-2017/app-comb-2017.pdf
5. Alan Tucker (2012), *"Applied Combinatorics"*, Wiley, https://www.isinj.com/mt-usamo/Applied%20Combinatorics%20(6th%20Edition)%20by%20Alan%20Tucker%20Wiley%20(2012).pdf
6. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein (Fourth Edition), *"Introduction to Algorithms"*, The MIT Press, https://dl.ebooksworld.ir/books/Introduction.to.Algorithms.4th.Leiserson.Stein.Rivest.Cormen.MIT.Press.9780262046305.EBooksWorld.ir.pdf

7. Hartmut Stadtler, Christoph Kilger and Herbert Meyr (4th Edition), *"Supply Chain Management and Advanced Planning: Concepts, Models, Software, and Case Studies"*, Springer, https://mynotesonsystemicthinking.files.wordpress.com/2011/02/scm-and-adv-planning.pdf

8. Michael ten Hompel and Thorsten Schmidt ,*"Warehouse Management: Automation and Organisation of Warehouse and Order Picking Systems"*, https://download.e-bookshelf.de/download/0000/0109/28/L-G-0000010928-0002343854.pdf
9. Sunil Chopra and Peter Meindl (Fifth Edition), *"Supply Chain Management: Strategy, Planning, and Operation"*, Pearson, https://base-logistique-services.com/storage/app/media/Chopra_Meindl_SCM.pdf
10. ShipCalm (2022), *"10 Types of Warehouses: What Are They and How to Choose the Right One"*, https://www.shipcalm.com/blog/warehouse-types/
11. Unity Forum (2023), https://forum.unity.com/
12. Unity (2023), https://unity.com
13. Unity Technologies. (2020). *Unity User Manual*. Retrieved from Unity Manual
14. Gamasutra. (2021). *The Evolution of Unity: 2005 to 2021*. Retrieved from Gamasutra
15. GamesIndustry.biz. (2022). *How Unity Became the Go-To Engine for Indie Games*. Retrieved from GamesIndustry.biz
16. Microsoft. (2022). *C# Programming Guide*. Retrieved from Microsoft Docs
17. Game Developer. (2021). *Unity: The Complete Guide to Game Development*. Retrieved from Game Developer
18. Unity Learn. (2023). *Unity Learn: Your Learning Platform for Game Development*. Retrieved from Unity Learn
19. Unity Community. (2023). *Unity Community Forum*. Retrieved from Unity Forum
20. K. R. Schubert. (2018). *Game Design Workshop: A Playcentric Approach to Creating Innovative Games*. CRC Press.

Παράρτημα Α: Ο κώδικας της εφαρμογής

Anchor Creator

```
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.XR.ARFoundation;
using UnityEngine.XR.ARSubsystems;

//
// This script allows us to create anchors with
// a prefab attached in order to visbly discern where the anchors are created.
// Anchors are a particular point in space that you are asking your device to track.
//

[RequireComponent(typeof(ARAnchorManager))]
[RequireComponent(typeof(ARRaycastManager))]
[RequireComponent(typeof(ARPlaneManager))]
public class AnchorCreator : MonoBehaviour
{
    // This is the prefab that will appear every time an anchor is created.
    [SerializeField]
    GameObject m_AnchorPrefab;

    public GameObject AnchorPrefab
    {
        get => m_AnchorPrefab;
        set => m_AnchorPrefab = value;
    }

    // Removes all the anchors that have been created.
    public void RemoveAllAnchors()
    {
        foreach (var anchor in m_AnchorPoints)
        {
            Destroy(anchor);
        }
        m_AnchorPoints.Clear();
    }
```

```
    // On Awake(), we obtains a reference to all the required components.
    // The ARRaycastManager allows us to perform raycasts so that we know where to place an anchor.
    // The ARPlaneManager detects surfaces we can place our objects on.
    // The ARAnchorManager handles the processing of all anchors and updates their position and rotation.
    void Awake()
    {
        m_RaycastManager = GetComponent<ARRaycastManager>();
        m_AnchorManager = GetComponent<ARAnchorManager>();
        m_PlaneManager = GetComponent<ARPlaneManager>();
        m_AnchorPoints = new List<ARAnchor>();
    }

    void Update()
    {
        // If there is no tap, then simply do nothing until the next call to Update().
        if (Input.touchCount == 0)
            return;

        var touch = Input.GetTouch(0);
        if (touch.phase != TouchPhase.Began)
            return;

        if (m_RaycastManager.Raycast(touch.position, s_Hits, TrackableType.PlaneWithinPolygon))
        {
            // Raycast hits are sorted by distance, so the first one
            // will be the closest hit.
            var hitPose = s_Hits[0].pose;
            var hitTrackableId = s_Hits[0].trackableId;
            var hitPlane = m_PlaneManager.GetPlane(hitTrackableId);

            // This attaches an anchor to the area on the plane corresponding to the raycast hit,
            // and afterwards instantiates an instance of your chosen prefab at that point.
            // This prefab instance is parented to the anchor to make sure the position of the prefab is consistent
            // with the anchor, since an anchor attached to an ARPlane will be updated automatically by the
ARAnchorManager as the ARPlane's exact position is refined.
            var anchor = m_AnchorManager.AttachAnchor(hitPlane, hitPose);
            Instantiate(m_AnchorPrefab, anchor.transform);

            if (anchor == null)
            {
                Debug.Log("Error creating anchor.");
            }
            else
            {
                // Stores the anchor so that it may be removed later.
                m_AnchorPoints.Add(anchor);
            }
        }
    }

    static List<ARRaycastHit> s_Hits = new List<ARRaycastHit>();

    List<ARAnchor> m_AnchorPoints;

    ARRaycastManager m_RaycastManager;

    ARAnchorManager m_AnchorManager;

    ARPlaneManager m_PlaneManager;
}
```

Box Pick Up

```
using UnityEngine;

public class BoxPickup : MonoBehaviour, IInteractable
{
   [SerializeField] Node _ownNode;

   private void Awake()
   {
      _ownNode = GetComponent<Node>();
   }

   public void PickUpInteraction()
   {
      gameObject.SetActive(false);

      //When finding object then return to the shortest node of object
      Node nextPacketNode = LoadData.S.GetNextPacket();
      Path path = Graph.S.GetShortestPath(_ownNode.connections[0], nextPacketNode);

      FindObjectOfType<TraverseGraph>().ShowNextPath(path);
   }
}
```

Camera Follow

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

public class CameraFollow : MonoBehaviour
{
   public Transform target;
   public Vector3 offset;

   // Update is called once per frame
   void Update()
   {
      transform.position = target.position + offset;
   }
}
```

Face Camera

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

public class FaceCamera : MonoBehaviour
{
   Transform mainCamera;
   Vector3 targetAngle = Vector3.zero;
   // Start is called before the first frame update
   void Start()
   {
```

```
        mainCamera = Camera.main.transform;
    }

    // Update is called once per frame
    void Update()
    {
        transform.LookAt(mainCamera);
        targetAngle = transform.localEulerAngles;
        targetAngle.x = 0;
        targetAngle.z = 0;
        transform.rotation = Quaternion.Euler(targetAngle);
    }
}
```

Gaze

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

public class Gase : MonoBehaviour
{
    [SerializeField] float gaseDistance = 5;

    List<InfoBehaviour> infolist = new List<InfoBehaviour>();
    RaycastHit hit;

    // Start is called before the first frame update
    void Start()
    {
        infolist = FindObjectsOfType<InfoBehaviour>().ToList();
    }

    // Update is called once per frame
    void Update()
    {
        if (Physics.Raycast(transform.position, transform.forward, out hit, gaseDistance)
            || Physics.Raycast(transform.position + transform.up, transform.forward, out hit, gaseDistance)
            || Physics.Raycast(transform.position - transform.up, transform.forward, out hit, gaseDistance))
        {
            GameObject go = hit.collider.gameObject;
            if (go.CompareTag("hasInfo"))
            {
                OpenInfo(go.GetComponent<InfoBehaviour>());
            }
        }
        else
        {
            CloseAll();
        }
    }

    void OpenInfo(InfoBehaviour infoBehaviour)
    {
        foreach (InfoBehaviour info in infolist)
        {
```

```
            if (info == infoBehaviour)
            {
                info.OpenInfo();
            }
            else
            {
                info.CloseInfo();
            }

        }
    }

    void CloseAll()
    {
        foreach (InfoBehaviour info in infolist)
        {
            info.CloseInfo();
        }
    }

    private void OnDrawGizmos()
    {
        Gizmos.DrawRay(transform.position + transform.up, transform.forward * gaseDistance);
        Gizmos.DrawRay(transform.position, transform.forward * gaseDistance);
    }
}
```

Interactable

```
public interface IInteractable
{
    void PickUpInteraction();
}
```

Info Behaviour

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

public class InfoBehaviour : MonoBehaviour
{
    const float SPEED = 6f;
    [SerializeField] Transform SectionInfo;

    Vector3 desiredScale = Vector3.zero;

    private void Start()
    {
        CloseInfo();
    }

    // Update is called once per frame
    void Update()
    {
        SectionInfo.localScale = Vector3.Lerp(SectionInfo.localScale, desiredScale, Time.deltaTime * SPEED);
    }
```

```
    public void OpenInfo()
    {
        desiredScale = Vector3.one;
    }

    public void CloseInfo()
    {
        desiredScale = Vector3.zero;
    }
}
```

Load Data

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

[DefaultExecutionOrder(150)]
public class LoadData : MonoBehaviour
{
    public static LoadData S;

    [SerializeField] ProductList plist;
    [SerializeField] GameObject[] objcube;

    string jsonFilePath = string.Empty;
    private string jsonString;
    int currentPacket = -1;

    private void Awake()
    {
        S = this;
        jsonFilePath = Application.dataPath + "/Plugins/data.json";

        Debug.Log(jsonFilePath);
    }

    // Start is called before the first frame update
    void Start()
    {
        ProductList productList = Load();
        for (int i = 0; i < objcube.Length; i++)
        {
            objcube[i].transform.position = productList.products[i].product_pos;
        }
    }

    public Node GetNextPacket()
    {
        ++currentPacket;

        if (currentPacket >= objcube.Length)
            Debug.Log("No more packets to retrieve");

        return objcube[currentPacket].GetComponent<Node>();
    }
```

```
    private ProductList Load()
    {
        //Load
        if (File.Exists(jsonFilePath))
        {
            jsonString = File.ReadAllText(jsonFilePath);
            Debug.Log("File Loaded...");

            plist = JsonUtility.FromJson<ProductList>(jsonString);
            /* Debug.Log(plist.products[0].product_pos);
            Debug.Log(plist.products[0].product_img); */
            return plist;
        }
        else
        {
            Debug.Log("No Load...");
            return null;
        }
    }

    public ProductList ProductList => plist;

    private void OnDestroy()
    {
        S = null;
    }
}


[System.Serializable]
public class ProductList
{
    public List<Product> products;
}

[System.Serializable]
public class Product
{
    public string product_name;
    public Vector3 product_pos;
    public int product_qty;
    public string product_info;
    public string product_img;
}
```

Movement

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

public class Movement : MonoBehaviour
{

    private Touch touch;
    [SerializeField] float speedModifier;
    [SerializeField] float rotationModifier;
```

```
    int xInput, yInput;


    // Update is called once per frame
    void Update()
    {
        try
        {
            touch = Input.GetTouch(0);

            if (touch.phase.Equals(TouchPhase.Moved))
            {
                Vector3 cameraRelative = new Vector3(0f, 0f, touch.deltaPosition.normalized.y) * speedModifier *
Time.deltaTime;
                transform.position += Camera.main.transform.TransformDirection(cameraRelative);

                //Un-comment to enable finger rotation
                transform.rotation *= Quaternion.AngleAxis(touch.deltaPosition.normalized.x, Vector3.up);
            }
        }
        catch (ArgumentException)
        {
            //nop...
        }
    }

    private static Quaternion GyroToUnity(Quaternion q)
    {
        return new Quaternion(q.x, q.y, -q.z, -q.w);
    }
}
```

Player Interactions

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

public class PlayerInteractions : MonoBehaviour
{
    [SerializeField] float rayLength;

    RaycastHit hitInfo;
    private void Update()
    {
        Ray forwardRay = new Ray(Camera.main.transform.position, Camera.main.transform.forward);
        Ray upRay = new Ray(Camera.main.transform.position + Vector3.up, Camera.main.transform.forward);
        Ray downRay = new Ray(Camera.main.transform.position - Vector3.up, Camera.main.transform.forward);

        if (Physics.Raycast(forwardRay, out hitInfo, rayLength)
            || Physics.Raycast(upRay, out hitInfo, rayLength)
            || Physics.Raycast(downRay, out hitInfo, rayLength))
        {
            Debug.Log($"Looking at {hitInfo.transform.name}");
            if (Input.GetKeyDown(KeyCode.Mouse0))
            {
                IInteractable interaction = hitInfo.transform.GetComponent<IInteractable>();

                if (interaction != null)
                {
```

```
                interaction.PickUpInteraction();
            }
        }
    }
}

    private void OnDrawGizmos()
    {
        Debug.DrawRay(Camera.main.transform.position, Camera.main.transform.forward * rayLength, Color.green);
    }
}
```

Load Product Info

```
using UnityEngine;
using UnityEngine.UI;
using TMPro;

[DefaultExecutionOrder(500)]
public class ProductInfo : MonoBehaviour
{
    [SerializeField] int index;
    [SerializeField] Image objImage;
    [SerializeField] TextMeshProUGUI objQuantity;
    [SerializeField] TextMeshProUGUI objTitle;
    [SerializeField] TextMeshProUGUI objDescription;

    // Start is called before the first frame update
    void Start()
    {
        LoadProductInfo();
    }

    public void LoadProductInfo()
    {
        if (index < 0 || index > LoadData.S.ProductList.products.Count)
            throw new System.IndexOutOfRangeException();

        //Cache the texture from the Resources folder
        Texture2D tempTexture = Resources.Load<Texture2D>(LoadData.S.ProductList.products[index].product_img);

        //Create a new sprite based on dimensions of loaded texture
        Sprite createdSprite = Sprite.Create(tempTexture, new Rect(0, 0, tempTexture.width, tempTexture.height), new
Vector2(0.5f, 0.5f));

        //Apply the created sprite to image component.
        objImage.sprite = createdSprite;
        objQuantity.SetText($"{LoadData.S.ProductList.products[index].product_qty}");
        objTitle.SetText($"{LoadData.S.ProductList.products[index].product_name}");
        objDescription.SetText($"{LoadData.S.ProductList.products[index].product_info}");
    }
}
```

Traverse Graph

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

public class TraverseGraph : MonoBehaviour
{
    LineRenderer lineRenderer;
    Path activePath;
    IEnumerator activeCoroutine;

    private void Awake()
    {
        lineRenderer = GetComponent<LineRenderer>();
    }

    private void Start()
    {
        activePath = Graph.S.CalculateInitialPath();
        ShowNextPath(activePath);
    }

    public void ShowNextPath(Path nextPath)
    {
        activePath = nextPath;

        activeCoroutine = ShowPath();
        StartCoroutine(activeCoroutine);
    }

    IEnumerator ShowPath()
    {
        List<Vector3> positions = new List<Vector3>();

        foreach (var node in activePath.nodes)
        {
            Vector3 tempPos = node.transform.position;
            tempPos.y = 0f;

            positions.Add(tempPos);

            //Debug.Log(node.transform.position);
        }

        lineRenderer.positionCount = positions.Count;
        lineRenderer.SetPositions(positions.ToArray());

        yield return null;
    }

    private void OnDestroy()
    {
        StopAllCoroutines();
    }
}
```