ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Υλοποίηση 2D παιχνιδιού προσομοίωσης κατασκευής και διαχείρισης μέσω του framework libGDX

ΟΝΟΜΑΤΕΠΩΝΥΜΟ ΦΟΙΤΗΤΗ
Γούλας Ηλίας

ΕΠΙΒΛΕΠΩΝ: Ιωάννης Τζήμας

ΠΑΤΡΑ 2021

Περιεχόμενα

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέπων της παρούσας εργασίας, Ιωάννη Τζήμα, για την ευκαιρία που μου δόθηκε να αναπτύξω το συγκεκριμένο παιχνίδι στα πλαίσια της εργασίας.

Θα ήθελα επίσης να ευχαριστήσω την οικογένεια μου αλλα και τους φίλους μου για την υποστήριξή τους σε όλη την διάρκεια της εργασίας.

Περίληψη

Η διπλωματική εργασία έχει ως στόχο την κατα όσο δυνατόν πιο πολύπλευρη εξευρεύνηση της υλοποίησης ενός 2D παιχνιδιού προσομοίωσης κατασκευής και διαχείρισης και των συστημάτων που το αποτελούν χρησιμοποιώντας ως μέσο ανάπτυξης το application framework libGDX. Το εν λόγω framework που είναι βασισμένο στην Java,C και C++ και τα βασικα του στοιχεία θα αναλυθούν επίσης σε μία συνοπτική αναφορά με σκοπό την καλύτερη κατανόηση τους αλλα και την χρησιμότητα τους ως εργαλείων ανάπτυξης των συστημάτων του παιχνιδιού.

Λέξεις κλειδιά: 2D, Game, Development, libGDX, Framework, Java

Abstract

This thesis is aiming for a multifaceted exploration of the development and implementation of a 2D construction and management simulation game and the parts that make it up using the application framework libGDX.Said framework is also going to be briefly discussed in the scope of a better understanding of its usefulness as a development tool for the game's own systems.

Keywords: 2D, Game, Development, libGDX, Framework, Java

Κεφάλαιο - 1 Εισαγωγή

Η διαδικασία ανάπτυξης ηλεκτρονικών παιχνιδιών περικλείει μια μεγάλη ποικιλία αναγκαίων γνώσεων και συστημάτων που μπορεί να είναι χρήσιμες για την υλοποίησή του,ανάλογα με το είδος του παιχνιδιου.

Κάποιες από αυτές τις γνώσεις και τα συστήματα είναι συχνά κοινά, όπως η ανάγκη εμφάνισης γραφικών ή ήχου, αλλά ακόμα και κάτι πιο απλό όπως μαθηματικές πράξεις. Μπορεί επίσης κάποια κομμάτια να είναι δύσκολα στην υλοποίηση.
Δέν είναι πάντα εφικτό κάθε ξεχωριστό άτομο ή ομάδα ατόμων (π.χ. εταιρεία) να κατέχει τις πλήρεις γνώσεις για την ορθή και αποδοτική υλοποίηση των συστημάτων αυτων,καθώς και το χρόνο και πόρους για να το κάνει.

Σε αυτό το σημείο έρχεται χρήσιμη μια πληθώρα άμεσα διαθέσιμων πακέτων λογισμικού ποικίλων μορφών και δυνατοτήτων, από πολύ δημοφιλής μηχανών αναπτύξεων παιχνιδιών εμπορικής χρήσης όπως είναι η Unity ή η Unreal Engine, πιο εξειδικευμένα εργαλεία ως προς την ευκολία χρήση τους από άτομα χωρίς πλήρη ή ακόμα και εξαιρετικά περιορισμένη τεχνική γνωση όπως το GameMaker Studio 2, μέχρι και application frameworks όπως και το libGDX που αποτελείται μόνο από κώδικα με την απουσία κάποιου editor, χωρις αυτο να σημαίνει ότι είναι απαραίτητα μειονέκτημα.
Κάθε ένα από αυτα τα εργαλεία έχει την χρήση του και δεν είναι τα μοναδικά που υπάρχουν διαθέσιμα άλλα απλά κάποια παραδείγματα

Συνεπώς, υπάρχουν επιλογές που καλείται να αποφασίσει το κατα περίπτωση άτομο ή ομάδα που επιθυμεί να αναπτύξει ένα ηλεκτρονικό παιχνίδι νωρίς στην διαδικασία της ανάπτυξης, με την επιλογή αυτή να είναι κρίσιμη για όλη την υπόλοιπη διάρκεια της ανάπτυξης, καθιστώντας την ετσι μια απο τις σημαντικότερες επιλογές από αποψής χρόνου ή/και πόρων σε περίπτωση επιλογής που αργότερα ανακαλύπτεται λανθασμένη για τους στόχους της ομάδας.

Την ίδια επιλογή ήταν ανάγκη να αναλογιστούμε και για την ανάπτυξη τού παιχνιδιού που πραγματεύεται η παρούσα διπλωματική εργασία.
Οι λόγοι για τους οποίους η επιλογή αυτή ήταν το framework libGDX ως μέσο ανάπτυξης θα αναφερθούν στα επόμενα υποκεφάλαια, αφού πρώτα όμως ειπωθούν η γενική ιδέα αλλα και οι απαιτήσεις του παιχνιδιού καθώς και τους σχετικούς στόχους της εργασίας , με σκοπό την καλύτερη κατανόηση της επιλογής.

1.1 Fabritaku

Ο τίτλος που επιλέχθηκε για το υπό ανάπτυξη παιχνίδι ειναι “Fabritaku”.

Το παιχνίδι, σύμφωνα και με το θέμα της διπλωματικής ανήκει στο είδος παιχνιδιών προσομοίωσης κατασκευής και διαχείρισης αλλά αυτό δεν αρκεί για να προσδιοριστεί με ακρίβεια το περιεχόμενο του αλλά και οι στοχοι του.

Τα παιχνίδια προσομοίωσης κατασκευής και διαχείρισης μπορούν να ποικίλουν δραστικά σε μορφή και περιεχόμενο, απο την προσομοίωση της κατασκευής μιας πόλης, την συντήρηση και βελτίωση μιας αποικίας, την διαχείριση επιχειρήσεων μέχρι και την ανάπτυξη εργοστασιακών εγκαταστάσεων και όχι μόνο.

Το Fabritaku πιο συγκεκριμένα προσομοιώνει την ανακάλυψη, συγκομιδή και επεξεργασία πρώτων υλών (π.χ ξυλεία,μεταλλεύματα), την μετατροπή τους σε εργαλεία ή την κατανάλωσή τους. Ο παίκτης-χρήστης στην περίπτωση μας ελέγχει άμεσα τις κινήσεις ενός μόνο ατόμου που έχει στην διάθεση του έναν διαδικαστικά παραγμένο, τυχαίο, κόσμο για την διενέργεια των προσομοιώσεων.

Τίτλοι του ίδιου είδους παιχνιδιών υπήρξαν έμπνευση για την δημιουργία του Fabritaku, συγκεκριμένα τα παιχνίδια Factorio της Wube Software, Rimworld της Ludeon Studios και σε ένα σύνολο πακέτων τροποποιήσεων από πολλούς διαφορετικούς ερασιτέχνες ή επαγγελματίες προγραμματιστές βασισμένα πάνω στον τίτλο Minecraft της Mojang Studios.

Η πληροφορία που έγινε γρήγορα σημαντική, σίγουρα σε ποσό, άσχετα με το αν υποσυνείδητα ήταν σχετικά προφανής, είναι η διάρκεια ανάπτυξής τέτοιου είδους παιχνιδιών.
Για παράδειγμα, οι προαναφερθέντες τίτλοι Factorio και Rimworld ξεκίνησαν την δημιουργία τους αρχές προς μέσα της δεκαετίας του 2010, και εξακολουθούν να λαμβάνουν καινούργιες εκδόσεις, προσθήκες ή αναβαθμίσεις ακομα και σήμερα(2021).

Έγινε γνωστό λοιπόν ακόμα και από μία τέτοια επιφανειακή μελέτη οτι δεν είναι εφικτό ή αν είναι εφικτό θα ήταν εξαιρετικά δύσκολο και δαπανηρό να δημιουργηθεί μια πλήρης έκδοση του Fabritaku, συγκρίσιμη ή/και ανταγωνιστική εμπορικά στα χρονικά και χρηματικά πλαίσια μιας διπλωματικής, που δεν έχει ως στόχο την παραγωγή του παιχνιδιού αποκλειστικά αλλά και την ανάλυση του ίδιου του λογισμικού σε ακαδημαϊκό πλαίσιο.

Ωστόσο, η ανάπτυξή του λογισμικού κρίθηκε χρήσιμη παρά τα πιθανά προβλήματα αλλα ήταν σημαντικό να προσδιοριστούν επαρκώς οι στόχοι και οι παραχωρήσεις που έπρεπε να γίνουν για να είναι αποδεκτά και ωφέλιμα τόσο η διαδικασία αλλα καί το τελικό αποτέλεσμα.

1.2 Στόχοι και παραχωρήσεις

Δεδομένης της παραχώρησης που συζητήθηκε στο προηγούμενο υποκεφάλαιο περί χρονικών ορίων στα πλαίσια της εργασίας υπήρξε αναζήτηση σχετικά με την ύπαρξη ενός θεμελιώδους στόχου που να καθιστά την διαδικασία υλοποίησης χρήσιμη σε ακαδημαϊκό πλαίσιο, πέρα από την απλή ολοκλήρωση της.

Ο στόχος αυτός στην συγκεκριμένη περίπτωση έχει τελικά παραπάνω από μία πτυχές, που όμως συνδυάζονται ομαλά και συμβαδίζουν.
Αναφορικά, μία από τις πτυχές είναι η καταγραφή της διαδικασίας ανάπτυξης και τις επιλογές που πραγματοποιήθηκαν κατα την διάρκεια της.
Μία άλλη πτυχή είναι η ανάδειξη της ανάπτυξης ενός παιχνιδιού με την χρήση ενός application framework σε αντίθεση με την πιο συμβατική επιλογή χρήσης μίας μηχανής ανάπτυξης παιχνιδιών.
Μια ακόμα πτυχή που φάνηκε δυνητικά χρήσιμη είναι η καταγραφή και πιο λεπτομερής ανάλυση στα επιμέρους κομμάτια που χρειάστηκε να υλοποιηθούν.
Επίσης,

Σκοπός αυτών των στόχων είναι να παρουσιαστεί μια σφαιρική εικόνα σχετικά με το θέμα και να ολοκληρωθούν όσα περισσότερα συστήματα είναι δυνατόν με αποτέλεσμα την δυνατότητα χρήσης της διαδικασίας που επεξηγείται ως γενικό δείκτη κατεύθυνσης αλλα όχι ως απόλυτου οδηγου, καθώς κάθε διαδικασία ανάπτυξης έχει τους δικούς της στόχους αλλα και ιδιαιτερότητες.

Η κύρια παραχώρηση που πρέπει να τονιστεί έχει άμεση σχέση και με τους στόχους που ορίστηκαν.
Έχοντας σκοπό την κάλυψη όσο το δυνατόν μεγαλύτερο αριθμό (χρήσιμων) συστημάτων και έχοντας υπόψη τους χρονικούς περιορισμούς πρέπει να γίνει γνωστό ότι κάθε επί μέρους σύστημα δεν παρουσιάζεται αυτούσιο ούτε ως μοναδική, ούτε ως βέλτιστη λύση άλλα αποτελεί τουλάχιστον μια υλοποίηση που λειτουργεί στο τελικό αποτέλεσμα.

Μια ακόμα παραχώρηση που πρέπει να γίνει σχετίζεται με την ίδια την διαδικασία της ανάπτυξης.
Δεν ήταν γνωστές από την αρχή ολες οι λεπτομέρειες σχετικά με την πλατφόρμα ανάπτυξης, ούτε όλες οι απαιτήσεις του παιχνιδιού, η διαδικασία ανάπτυξης παιχνιδιών, λόγο της πολυπλοκότητας της, περιλαμβάνει ανακάλυψη και επαναπροσδιορισμό της τελικής μορφής του αποτελέσματος, με βάση τους στόχους της εργασίας.
Πρακτικά, αυτό σημαίνει ότι κάτι που αρχικά θεωρούταν εφικτό μπορεί να κατέληγε δύσκολο ή σημαντικά χρονοβόρο, ή ακόμα και μή επαρκώς χρήσιμο στα πλαίσια της εργασίας, με αποτέλεσμα την αφαίρεση του απο το λογισμικό.Σε αντίθεση, κατι που δεν ήταν προφανές απο την αρχή μπορεί να προστέθηκε στα σκόπιμα συστήματα κατα την διάρκεια της ανάπτυξης.

1.3 Σχεδιασμός και μέθοδος ανάπτυξης

Πριν την έναρξη της υλοποίησης του λογισμικού πραγματοποιήθηκε μια συλλογή των απαιτήσεων και των συστημάτων που ήταν ήδη γνωστό ή προφανές ότι θα έπρεπε να δημιουργιθούν.

Παρατηρήθηκε όμως ότι δεν ήταν γνωστό όλο το σύνολο των απαιτήσεων, ότι κάποια κομμάτια του λογισμικού δεν είχαν επαρκώς προσδιορίσιμη διαδικασία υλοποίησης, δεν ήταν βέβαια η δυνατότητα ολοκλήρωσης τους εντός του χρονικού ορίου ή η χρηστικότητα τους.

Έχοντας αυτά τα δεδομένα ήταν ξεκάθαρο ότι δεν μπορούσε να ακολουθηθεί μια μέθοδος ανάπτυξης που να ορίζει πλήρως κάθε κομμάτι του λογισμικού πριν την έναρξη υλοποίησης του.
Για την αντιμετώπιση των αγνώστων αυτών παραγόντων επιλέχθηκε ενα επαναληπτικό μοντέλο ανάπτυξής σε συνδυασμό με ανάπτυξη πρωτοτύπων.

Το επαναληπτικό μοντέλο ανάπτυξης ορίστηκε αρχικά σε κύκλους ανάπτυξης δύο εβδομάδων καθώς υλοποιούνταν τα κεντρικά και πιο γνωστά συστήματα και αυτά που τα υποστηρίζουν, ενώ σε μετέπειτα στάδιο ο κύκλος αυξήθηκε σε συχνότητα, έχοντας διάρκεια κύκλου μιας εβδομάδας, καθώς τα συστήματα ήταν λιγότερο γνωστά και έπρεπε να γίνει η τελική κρίση στο ποια συστηματα θα χρειαστεί να αφαιρεθουν απο το σχέδιο.

Και στα δύο είδη κύκλων ανάπτυξης η βασική διαδικασία ήταν κοινή, κατα την διάρκεια του κύκλου αναπτύσονταν συστηματα-πρωτότυπα ή πραγματοποιούνταν προσθήκες και βελτιώσεις σε προηγούμενος υλοποιημένα συστήματα.
Στο τέλος κάθε κύκλου θεωρήθηκε σκόπιμο να εκτελείται μια ανασκόπηση των προσθηκών και αλλαγών που πραγματοποιήθηκαν κατα την διάρκεια του με σκοπό την αξιολόγηση τους αλλα και την αξιολόγηση του συνολικού λογισμικού, επιτρέποντας έτσι την ορθή οριοθέτηση των στόχων και των ελλείψεων για τον κύκλο που ακολουθούσε.

Η ευέλικτη μέθοδος αυτή είχε ως αποτέλεσμα την συνεχή επαφή της υλοποίησης με τους στόχους, καθιστώντας δυνατή την υλοποίηση ενός συνόλου συστημάτων που κρίθηκε επαρκής, ως προς την ποιότητα και την ποσότητα, εχοντας ως σημείο αναφοράς σκοπούς της εργασίας.

1.4 Επιλογή λογισμικού ανάπτυξης - libGDX

Η επιλογή ενός framework έναντι μίας μηχανής ανάπτυξής παιχνιδιών για την ανάπτυξη του λογισμικού ήταν σχετικά εύκολη στην συγκεκριμένη περίπτωση.
Το παιχνίδι βασίζεται σε 2D γραφικά, το περιεχόμενο του παιχνιδιού παράγεται προγραμματιστικά οπότε δεν μας ήταν απαραίτητος καποιος editor που προσφέρουν οι μηχανές ανάπτυξης.Επισής, ένα framework, οπως το libGDX, θα μπορούσε να μας δώσει την ελευθερία να ορίσουμε την αρχιτεκτονική του παιχνιδιού οπως εμείς επιθυμούμε, εμπλουτίζοντας έτσι το περιεχόμενο της ανάλυσης μας.

Δεν υπάρχει όμως μόνο το libGDX ώς λύση, θα μπορούσε κάλλιστα να χρησιμοποιηθεί καποιο διαφορετικό πακέτο λογισμικού, όπως τα SDL2 ή liballeg(Allegro).
Για τους σκοπούς της εργασίας όμως κρίθηκε καλύτερη επιλογή το libGDX, διότι υπάρχει προηγούμενη εμπειρία εργασίας με αυτό και η γλώσσα προγραμματισμού που χρησιμοποιεί (Java) ήταν πιο γνώριμη σε σύγκριση με C/C++ των άλλων περιπτώσεων.
Η επιλογή του libGDX δεν ηταν μονο θέμα εμπειρίας φυσικά, το framework έχει αποδεδηγμένα ικανότητες παραγωγής αποτελεσμάτων, με τίτλους όπως το Slay the Spire, Mindustry και Space Haven να έχουν διανεμηθεί σε εμπορικές διαδικτυακές πλατφόρμες παιχνιδιών.

Μια πλήρη σύγκριση μεταξύ μηχανών παιχνιδιών και framework είναι εκτός του πλαισίου της εργασίας αλλα πρακτικά δεν είναι αναγκαία, εξαιτίας των περιορισμένων απαιτήσεων του παιχνιδιού μας.

Κεφάλαιο - 2 libGDX

Σαν τελευταίο στάδιο πρίν προχωρήσουμε στην αναφορά και ανάλυση των συστημάτων που υλοποιήθηκαν, θα αναφέρουμε στο παρόν κεφάλαιο τις δυνατοτητες αλλα και τον τρόπο λειτουργίας του libGDX.

2.1 Γενικά χαρακτηριστικά libGDX

Το libGDX παρέχει μέσω ενός API την δυνατότητα να διαθέσουμε μια εφαρμογή που βασίζεται πάνω σε αυτο στις πλατφόρμες Windows, Linux, macOS, Android, iOS και Web(μέσω GWT). Για το rendering σε όλες τις πλατφόρμες βασίζεται στο OpenGL ES 2.0/3.0.
Ο πηγαίος κώδικας είναι ανοιχτός, υπο την επιτρεπτική άδεια Apache 2.0 καθιστώντας δυνατή την χρήση του για εμπορικά και μη έργα, καθώς και την τροποποίηση του.
Βρίσκεται σε εξέλιξη εδώ και πάνω από μία δεκαετία και έχει ένα μεγάλο αριθμό άρθρων τεκμηρίωσης και εκμάθησης, και βοηθητικου λογισμικού, είτε αυτα προερχονται απο την ομάδα ανάπτυξής, είτε από την κοινότητα που έχει δημιουργηθεί γύρω του, ειτε απο τρίτες πηγές.

Παρέχει επίσης μια μεγάλη ποικιλία χαρακτηριστικών:

- Δυνατότητα αναπαραγωγής εφέ ήχου και μουσικής
- Δυνατότητα καταγραφής και χειρισμού μεθόδων εισόδου οπως ποντίκι, πληκτρολόγιο, οθόνη αφής, χειριστήριο αλλα και χειρονομιών
- Βοηθητικές συναρτήσεις μαθηματικών, βοηθητικές συναρτήσεις γεωμετρίας, μηχανές προσομοίωσης φυσικής
- Αφαιρέσεις πάνω στο σύστημα αρχείων, διαχείριση πόρων, σειριοποίηση JSON και XML
- Βοηθητικά συστήματα rendering υψηλού επιπεδου όπως sprite batching, texture atlases, particles, scene graphs
- Βιβλιοθήκη UI

Τα παραπάνω δεν ειναι μια εξαντλητική λίστα, καθώς παρέχει και στοιχεία π.χ. 3D rendering ή networking, αλλα κάποια από χαρακτηριστικά που θα συναντήσουμε κατα την ανάλυση του παιχνιδιού μας.

2.2 Διαδικασία προετοιμασίας περιβάλλοντος ανάπτυξης

Μια εφαρμογή βασισμένη στο libGDX έχει μια απλή διαδικασία δημιουργίας ενος καινούργιου project, χρησιμοποιώντας το εργαλείο gradle για την διαχείριση των εξαρτήσεων και του building.

1) **Δημιουργία περιβάλλοντος ανάπτυξης**

Επιλέγουμε ένα IDE που επιθυμούμε(ή command line), εγκαθιστούμε το απαραίτητο JDK και Android SDK αν δεν διαθέτετε μαζί με το IDE της επιλογής μας.

2) **Δημιουργία αρχείων του project**

Μέσω του Project Setup Tool(ή command line) επιλέγουμε τις πλατφόρμες για τις οποίες θέλουμε να αναπτύξουμε το λογισμικό μας, καθώς και τις επεκτασεις που επιθυμούμε

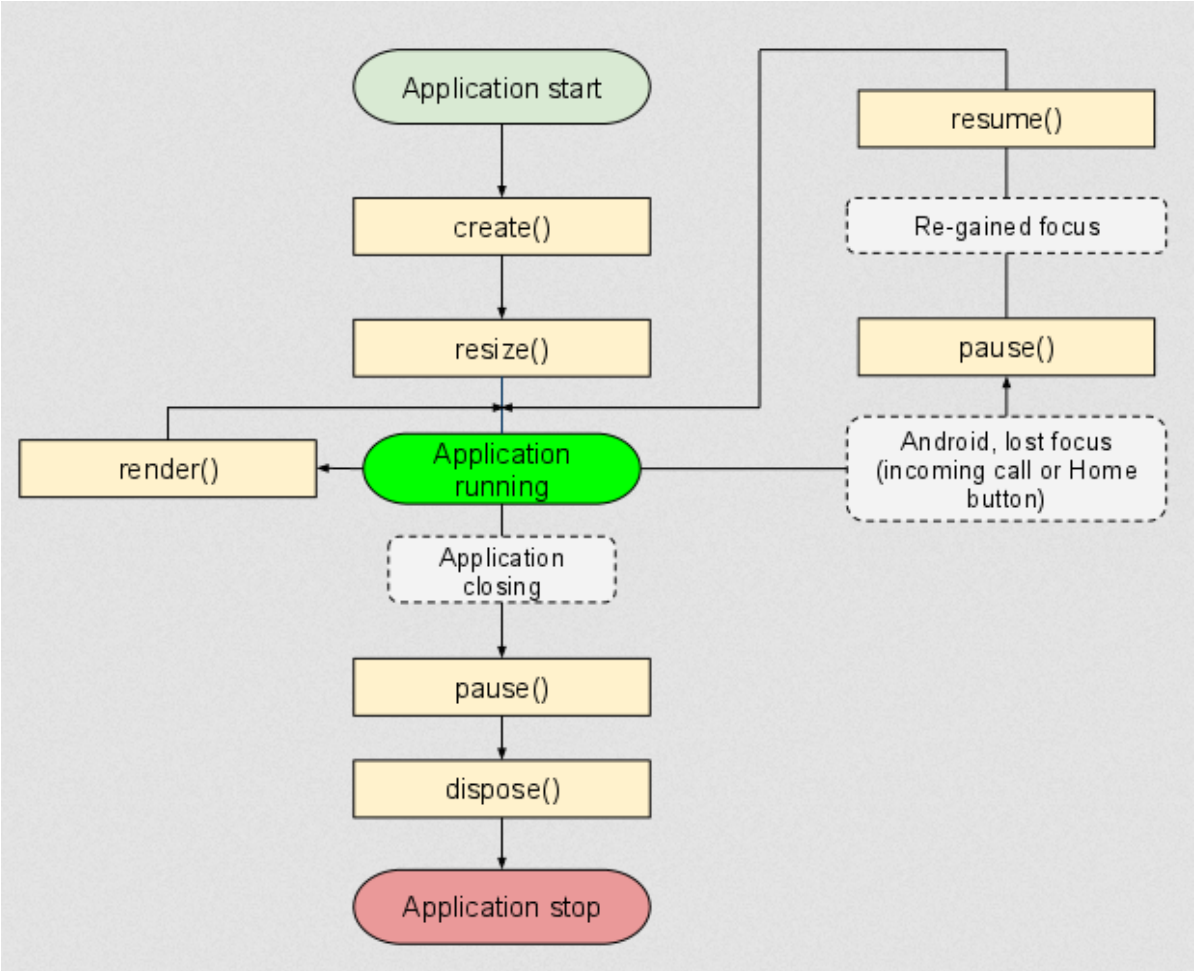
3) **Εισαγωγή του project**

Εισάγουμε τα αρχεία του project που παρήχθησαν στο βήμα 2) στο μέσο ανάπτυξης της επιλογής μας και ανανεώνουμε το project του gradle,αν χρειάζεται.

Μετα απο αυτά τα βήματα το project μας είναι έτοιμο για εκτέλεση/ανάπτυξη.

Το Project Setup Tool, μαζί με περισσότερες λεπτομέρειες για την άνω διαδικασία και με παραδείγματα δημοφιλών IDE μπορούν να βρεθούν στην ιστοσελίδα του libGDX.

2.3 Κύκλος ζωής μιας εφαρμογής libGDX



Εικόνα 1 - Κύκλος ζωής μιας εφαρμογής libGDX - Πηγή: libGDX wiki

Ο κύκλος ζωής μιας εφαρμογής libGDX δεν είναι ιδιαίτερα περίπλοκος, όπως φαίνεται και από την άνω απεικόνιση.

Κατα την δημιουργία της εφαρμογής μας καλείται η συνάρτηση create(), κατα την οποία μπορούμε να αρχικοποιήσουμε τις μεταβλητές και τα αντικείμενα που επιθυμούμε.

Στην συνέχεια, καλείται η συνάρτηση resize(), η οποία καλείται και κάθε άλλη φορά που η εφαρμογή μας αλλάζει διαστάσεις, επιτρέποντας μας να διαχειριστούμε την αλλαγή αυτη, για παράδειγμα να επαναπροσδιορίσουμε το μέγεθος του UI μας.

Η συνάρτηση render() καλείται κάθε φορά που πρέπει να γίνει απεικόνιση των γραφικών στην οθόνη και είναι και το μέρος που μπορούμε να ορίσουμε τον κώδικα του παιχνιδιού μας που χρειάζεται συνεχής ενημερώσεις.

Οι συναρτήσεις pause() και dispose() καλούνται πριν τον τερματισμό της εφαρμογής μας, και μας δίνουν την δυνατότητα να αποθηκεύσουμε την κατάσταση του παιχνιδιού μας και να διαχειριστούμε σωστά την αποδεσμευση των πόρων συστήματος που έχουμε χρεώσει αντίστοιχα.

Για την πλατφόρμα Android υπάρχει επίσης μια ακόμα περίπτωση κλήσης των συναρτήσεων pause() και resume(), για την σωστή διαχείριση των διακοπών της κανονικής λειτουργίας της εφαρμογής, π.χ. Λόγω τηλεφωνικής κλήσης ή πατήματος του Home Button.

Κεφάλαιο 3 - Εισαγωγή Ανάλυσης

Στο επόμενο κεφάλαιο θα αναλυθεί το κεντρικό θέμα της εργασίας, η υλοποίηση του παιχνιδιού με μονάδα αναφοράς το κατα περίπτωση σύστημα.

Στα πλαίσια της απόκτησης μιας ολοκληρωμένης εικόνας θα παρουσιάζεται πηγαίος κώδικας της εφαρμογής, θα επεξηγούνται λειτουργίες του libGDX και η άμεση εφαρμογή τους κατα κανόνα με βάση το συστημα, αλλα και όπου κρίνεται απαραίτητο ξεχωριστά κομμάτια.Με το ίδιο σκεπτικό, θα παρουσιάζονται επεξηγηματικές εικόνες σχετικα με τα συστήματα του παιχνιδιού.

3.1 Γενικές πληροφορίες υλοποίησης

Το Fabritaku, στα πλαίσια της εργασίας, έχει ως στόχο τις πλατφόρμες Windows, Linux, macOS και Android. Η υλοποίηση έχει γίνει με την χρήση της γλώσσας Java και περιλαμβάνει ένα μεγάλο σύνολο κλάσεων, αναγκαίο για την τμηματική αντιμετώπιση του λογικά πολύπλοκου και εκτενές σε αριθμό σειρών κώδικα.

3.2 Ταξινόμηση συστημάτων

Τα συστήματα που θα αναλυθούν ταξινομούνται σε λογικό επίπεδο σε:

- Κύρια Συστήματα
- Βοηθητικά Συστήματα
- Κοινά Συστήματα

Κύρια Συστήματα ονομάζονται τα συστήματα που πραγματοποιούν διεργασίες σημαντικές για την άρτια λειτουργία της εφαρμογής, συνήθως με συγκεκριμένο σκοπό λειτουργίας και δεν εξαρτώνται από κάποιο άλλο σύστημα.

Βοηθητικά Συστήματα ονομάζονται τα συστήματα που είναι εξαρτώμενα από κάποιο άλλο σύστημα στο οποίο προσφέρουν λειτουργίες ή δεδομένα και δεν τελούν ξεχωριστά κάποια χρήσιμη λειτουργία.

Κοινά Συστήματα μπορεί να μοιράζονται χαρακτηριστικά και απο τις δύο άλλες κατηγορίες αλλα ξεχωρίζουν στην λειτουργία τους λόγω της αυξημένης επικοινωνίας ή συζεύξης τους με άλλα συστήματα, άνω του ενός.

3.3 Μορφή κειμένου ανάλυσης

Κάθε σύστημα που αναλύεται θα έχει τουλάχιστον τις εξής πληροφορίες- υποκεφάλαια:

- Επισκόπηση λειτουργίας και στόχων
- Παρουσίαση και επεξήγηση κώδικα

Και προαιρετικά:

- Τεχνικές λεπτομέρειες
- Προτάσεις επέκτασης

Κεφάλαιο 4 - Συστήματα : Ανάλυση υλοποίησης - Επεξήγηση Συστημάτων - Τεχνικές Λεπτομέρειες

Ως σημείο εισαγωγής στήν ανάλυση των συστημάτων θα χρησιμεύσει το σύστημα οθόνων, το οποίο είναι ένα από τα πρώτα συστήματα που υλοποιήθηκαν.

4.1 Σύστημα Οθονών - Κοινό Σύστημα

4.1.1 Σύστημα Οθονών : Επισκόπηση λειτουργίας και στόχων

Το σύστημα οθονών είναι νοητικά υπεύθυνο για των διαχωρισμό των διαφορετικών οθονών διεπαφής με τον χρήστη, αλλα και σε πρακτικό επίπεδο για την διαχείριση μεταβάσεων απο την μία οθόνη στην άλλη καθώς και την τμηματοποίηση του κώδικα μεταξύ των οθονών.

Το σύστημα αυτό δεν αφορά τα ίδια τα στοιχεία διεπαφής χρήστη, το οποίο είναι ξεχωριστό σύστημα, αλλα τις μονάδες που τα περιλαμβάνουν.

Υπάρχουν δύο οθόνες που δημιουργήθηκαν, η οθόνη κεντρικού μενού και η οθόνη κόσμου. Κάθε οθόνη έχει την δική της λειτουργία και σκοπο, κριτήρια που επιτρέπουν τον διαχωρισμό τους.

Η οθόνη κεντρικού μενού είναι υπεύθυνη για την προβολή των επιλογών που έχει ο χρήστης κατα την έναρξη της εφαρμογής, με τις επιλογές να είναι :

- Εναρξη Παιχνιδιού (Play)
- Ρυθμίσεις (Settings)
- Ευχαριστίες (Credits)
- Έξοδος (Quit)

Απο πλευράς κώδικα, η οθόνη κεντρικού μενου είναι επίσης υπεύθυνη για την αρχικοποίηση διαφόρων αναγκαίων συστημάτων και πόρων.

Η οθόνη κόσμου (ή εναλλακτικά οθόνη παιχνιδιού) είναι υπεύθυνη για την προβολή της απεικόνισης του κόσμου και της κατάσταση του παιχνιδιού.

Οι ευθύνες που αναφέρθηκαν, εκτος απο την διαχείριση των μεταβάσεων, είναι στην πράξη λειτουργίες των άλλων συστημάτων, με το σύστημα οθονών να είναι ένα σημείο έναρξης-”δοχείο” για αυτές.

4.1.2 Σύστημα Οθονών : Παρουσίαση και επεξήγηση κώδικα

Το σύστημα οθονών σε μορφή κώδικα ουσιαστικά υλοποιεί τον κύκλο ζωής που αναφέρθηκε στο υποκεφάλαιο 2.3. Η λειτουργία αυτή γίνεται δυνατή χρησιμοποιώντας τις κλάσεις Game και το interface Screen του libGDX.

```
public class Fabritaku extends Game {

    @Override
    public void create () {
        Gdx.gl.glClearColor( red: 0,  green: 0,  blue: .2f,  alpha: 1);
        this.setScreen(new MainMenuScreen( game: this));
    }

    @Override
    public void render () {
        super.render();
    }

    @Override
    public void dispose () { this.getScreen().dispose(); }
}
```

Εικόνα 2 - Κώδικας της κλάσης Fabritaku

Επεκτείνοντας την κλάση Game μας διατίθενται ο μηχανισμός αλλαγής οθονών και το μόνο που χρειάζεται να κάνουμε είναι να ορίσουμε και να δημιουργήσουμε την οθόνη που επιθυμούμε, και να καλέσουμε τις συναρτήσεις της κλάσης Game που μας είναι απαραίτητες, εδώ την render() και την dispose().

Με την επέκταση αυτή, το σημείο εισαγωγής της εφαρμογής μας είναι μόνο λίγες γραμμές κώδικα, ανάλογα με την πλατφόρμα που εκτελείται κατα περίπτωση:

```
public class DesktopLauncher {
    public static void main (String[] arg) {
        Lwjgl3ApplicationConfiguration config = new Lwjgl3ApplicationConfiguration();
        new Lwjgl3Application(new Fabritaku(), config);
    }
}
```

Εικόνα 3 - Κώδικας του σημείου εισόδου στις πλατφόρμες Desktop

```
public class AndroidLauncher extends AndroidApplication {
    @Override
    protected void onCreate (Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        AndroidApplicationConfiguration config = new AndroidApplicationConfiguration();
        config.useAccelerometer = false;
        config.useCompass = false;
        initialize(new Fabritaku(), config);
    }
}
```

Εικόνα 4 - Κώδικας του σημείου εισόδου στην πλατφόρμα Android

Κάθε πλατφόρμα έχει το δικό της κομμάτι αρχικοποίησης, για παράδειγμα η χρήση του LWJGL στις πλατφόρμες Desktop, εξού και η ανάγκη για διαφοροποίηση.
Εδώ παρουσιάζεται και η δύναμη του libGDX, αφου ο άνω κώδικας είναι ένα από τα ελάχιστα κομμάτια κώδικα που χρειάζονται αποκλειστικά δικό τους κώδικα, με τον υπόλοιπο να είναι κοινό.

Στήν οθόνη αρχικού μενού (MainMenuScreen) αρχικοποιούμε τις μεταβλητές και τα αντικείμενά που αποτελούν μέρος αλλων συστηματων, καθώς και φορτώνουμε τους πόρους που μας ενδιαφέρουν, σε προετοιμασία για την έναρξη του παιχνιδιου.

```
public MainMenuScreen(final Fabritaku game){
    this.game=game;

    batch = new SpriteBatch();

    manager = new ResourceManager();

    region = manager.getMainTextureAtlas().findRegion( name: "COBBLE", index: 1);
    ui = new MainMenuUi(batch, manager.getUiTextureAtlas());
    Gdx.input.setInputProcessor(ui.getStage());
}
```

Εικόνα 5 - Κώδικας του constructor της κλάσης MainMenuScreen

Οι κλάσεις που υλοποιούν το interface Screen αντί για την συνάρτηση create() χρησιμοποιούν τον constructor τους για την διαδικασία αρχικοποίησης. Ομοίως και στον άνω κώδικα της εικόνας 5

Για την διαχείριση των λειτουργιών που γίνονται κάθε frame εκτελείται η συνάρτηση render().

```
@Override
public void render(float delta) {
    Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

    ui.update(delta);

    int tilesToFillScreenX = (int)ui.getStage().getWidth()/32;
    int tilesToFillScreenY = (int)ui.getStage().getHeight()/32;

    batch.begin();
    for (int i = 0; i <= tilesToFillScreenX; i++) {
        for (int j = 0; j <= tilesToFillScreenY; j++) {
            batch.draw(region, x: i*32, y: j*32);
        }
    }
    batch.end();

    ui.draw();

    if(ui.playTouched){
        game.setScreen(new WorldScreen(game, path: ui.playPath+"/world.ftwf",manager));
        this.dispose();
    }
    if(ui.quitTouched){
        Gdx.app.exit();
    }

}
```

Εικόνα 6 - Κώδικας της συνάρτησης render() της κλάσης MainMenuScreen

Η συνάρτηση render() στην περίπτωση αυτή έχει ως παράμετρο μια float τιμή delta, η οποία είναι ο χρόνος που έχει περάσει από την προηγούμενη κλήση της. Η τιμή αυτη μας βοηθάει να κάνουμε τους υπολογισμούς που χρειάζονται για να εξομαλύνουμε διάφορα κομμάτια του κώδικα μας που επηρεάζονται από τις διακυμάνσεις σε χρόνο από κλήση σε κλήση.

Κατα την κλήση της συνάρτησης, σύμφωνα και με την εικόνα 6, με την κλήση της συνάρτησης glClear() διαγράφουμε τα δεδομένα χρώματος που υπαρχουν στην (σε μνήμη,

όχι φυσική) οθόνη από το προηγούμενο frame, με την κατάλληλη παράμετρο, ετοιμάζοντας έτσι το έδαφος για το καινούργιο frame.

Με την συνάρτηση update() της διεπαφής χρήστη(το αντικείμενο ui) της επιτρέπουμε να επεξεργαστεί τα δεδομένα από το προηγούμενο frame, που ενδεχομένως να έλαβε.

Στην συνέχεια, μέσω του αντικειμένου batch της κλάσης SpriteBatch κάνουμε render οτι χρειάζεται, και καλούμε και την συνάρτηση ui.draw() για να πράξει το ίδιο για τα στοιχεία που το αποτελούν.

Τέλος, αν η διεπαφή χρήστη ui έχει λάβει κάποιο πάτημα σε κάποιο από τα μέλη-κουμπιά της, πράττουμε τις ανάλογες ενέργειες.

Ομοίως, η οθόνη κόσμου πράττει αντίστοιχες λειτουργίες.

```
WorldScreen(final Fabritaku game,String path,ResourceManager manager){
    this.game=game;

    this.manager= manager;

    registry= new Registry();

    renderer = new Renderer( batchSize: 8000,registry,manager);

    dispatcher = new Dispatcher();

    if(!Gdx.files.local(path).exists()){
        WorldGenerator generator = new WorldGenerator( worldWidth: 80, worldHeight: 80);
        generator.setSeed((int) TimeUtils.millis());
        generator.initRNG();
        generator.generationStageOne();
        generator.generationStageTwo();
        generator.generationStageThree();
        generator.toStorage(path);
        generator = null;
        //Hint the garbage collector to collect the world generation leftovers
        System.gc();
    }

    player = new Player();

    world = new World(Gdx.files.local(path),renderer,registry,dispatcher,player);
    world.chunkLoader.update();

    ui = new GameplayUi(renderer.batch, manager.getUiTextureAtlas(), manager.getMainTextureAtlas(),dispatcher,registry);
    ui.quickInventory.setInventory(player.quickInventory);
    ui.inventoryAndCrafting.setInventory(player.inventory);

    input = new InputHandler(dispatcher, ui.getStage(),ui.touchpads.movementTouchpad,ui.touchpads.interactionTouchpad);

    //Event setup
    dispatcher.addReceiver(Events.MOVEMENT,player);
    dispatcher.addReceiver(Events.WORLD_INTERACTION,world);
    dispatcher.addReceiver(Events.UI,ui);
    dispatcher.addReceiver(Events.INVENTORY,world);
}
```

Εικόνα 7 - Κώδικας του constructor της κλάσης WorldScreen

Στον constructor της οθόνης κόσμου αρχικοποιούνται όλα τα λογικά κομμάτια που έχουν σχέση με τις ανάγκες του παιχνιδιού, όπως ο χαρακτήρας του χρήστη και ο κόσμος που θα αλληλεπιδρά, άλλα και συστήματα που τα υποστηρίζουν, όπως ο renderer, το ui, και άλλα.

Αν στον κόσμο που επιλέχθηκε από την οθόνη κεντρικού μενού ο χρήστης δεν έχει αλληλεπιδράσει στο παρελθόν(ή αν δεν υπαρχουν προηγούμενα δεδομένα) τότε παράγεται ένας καινούργιος κόσμος.

Η συνάρτηση render() είναι επίσης όμοια:

```
@Override
public void render(float delta) {
    Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

    world.update(delta);
    input.update(delta);
    player.update(delta);

    if(Gdx.input.isKeyPressed(Input.Keys.Z)){
        renderer.translateCameraZoom( z: -5*delta);
    }
    if(Gdx.input.isKeyPressed(Input.Keys.X)){
        renderer.translateCameraZoom( z: 5*delta);
    }

    //Match camera to player
    renderer.camera.position.set(player.position,0);

    renderer.update();

    renderer.begin();
    renderer.render(world);
    renderer.end();

    ui.update(delta);

    ui.draw();

    renderer.begin();
    renderer.render(ui);
    renderer.end();


}
```

Εικόνα 8 - Κώδικας της συνάρτησης render() της κλάσης WorldScreen

Η κύρια διαφορά με την οθόνη κεντρικού μενού είναι η παρουσία των στοιχείων παιχνιδιού τα οποία χρήζουν επίσης ενημέρωση για να εκτελέσουν τις λειτουργίες τους, και η σειρά με την οποία γίνεται το rendering.

Πολλές άγνωστες κλάσεις στον κώδικα μας προς το παρόν, θα αναλυθούν όμως στην συνέχεια, στο σύστημα που ανήκουν

4.2 Σύστημα Πόρων - Κοινό Σύστημα

4.2.1 Σύστημα Πόρων : Επισκόπηση λειτουργίας και στόχων

Το σύστημα πόρων είναι ένα μικρό και απλό σύστημα που αποτελείται από μία μόνο κλάση, την ResourceManager, και έχει ως στόχο την φόρτωση και διάθεση των πόρων που επισυνάπτουμε στα αρχεία του παιχνιδιού(π.χ. Textures) σε κάθε άλλο σύστημα που χρειάζεται αυτές τις πληροφορίες.

4.2.2 Σύστημα Πόρων : Παρουσίαση και επεξήγηση κώδικα

```
public class ResourceManager implements Disposable {
    AssetManager assetManager;

    public ResourceManager(){
        assetManager = new AssetManager();
        assetManager.load( fileName: "main_texture.atlas",TextureAtlas.class);
        assetManager.load( fileName: "ui.atlas",TextureAtlas.class);
        assetManager.finishLoading();
    }

    public TextureAtlas getMainTextureAtlas() { return assetManager.get( fileName: "main_texture.atlas",TextureAtlas.class); }

    public TextureAtlas getUiTextureAtlas() { return assetManager.get( fileName: "ui.atlas",TextureAtlas.class); }

    @Override
    public void dispose() {
        assetManager.dispose();
    }
}
```

Εικόνα 9 - Κώδικας της κλάσης ResourceManager

Ο κώδικας είναι απλός, γιατί στην ουσία το σύστημα αυτο είναι μια wrapper κλάση βασισμένη στην κλάση AssetManager του libGDX, με λίγες βοηθητικές συναρτήσεις στοχευμένες στις περιπτώσεις χρήσης του παιχνιδιού μας.

Κατα την δημιουργία ενός ResourceManager δημιουργείται ένα αντικείμενο AssetManager και φορτώνονται οι αναγκαίοι πόροι κάνοντας χρήση του αντικειμένου αυτού.

Μεγάλης σημασίας είναι η συνάρτηση dispose(), που επιβάλλει το interface Disposable.

4.2.3 Σύστημα Πόρων : Τεχνικές λεπτομέρειες

Το interface Disposable παρέχεται απο το libGDX είναι ένας από τους λόγους που αυτή η μία κλάση αποτελεί μόνη της ένα σύστημα. Είναι άκρως σημαντικό να ελευθερώνονται οι πόροι που έχουμε στην μνήμη όταν έχει πάψει να υπάρχει η αναγκαιότητα της ύπαρξής τους, ειδάλλως απλά την χρεώνουμε χωρίς αιτία και μπορούν να προκληθούν διαρροές μνήμης.

Έχοντας όλους τους πόρους σε ενα κεντρικό σημείο μπορούμε να κρατάμε στενή επαφή με τον κύκλο ζωής τους, και να τις καταστρέφουμε όταν δεν χρειάζονται πια.

4.3 Σύστημα παραγωγής κόσμου - Κύριο Σύστημα

4.3.1 Σύστημα παραγωγής κόσμου : Επισκόπηση λειτουργίας και στόχων

Το σύστημα παραγωγής κόσμου είναι υπεύθυνο για την παραγωγή ενός καινούργιου κόσμου, τυχαία παραγόμενου κάθε φορά, με τον οποίο θα αλληλεπιδράσει ο χρήστης.Το σύστημα αυτο καλείται μόνο όταν δεν υπάρχουν δεδομένα για ένα κόσμο που επιθυμεί να αλληλεπιδράσει ο χρήστης, για να τον παράξει.

Ως seed ορίζεται η είσοδος που θα λάβει το συστημα αυτο οταν καλείται να παράξει έναν κόσμο, και δεδομένου ενός συγκεκριμένου seed η έξοδος του θα είναι η ίδια κάθε φορά.

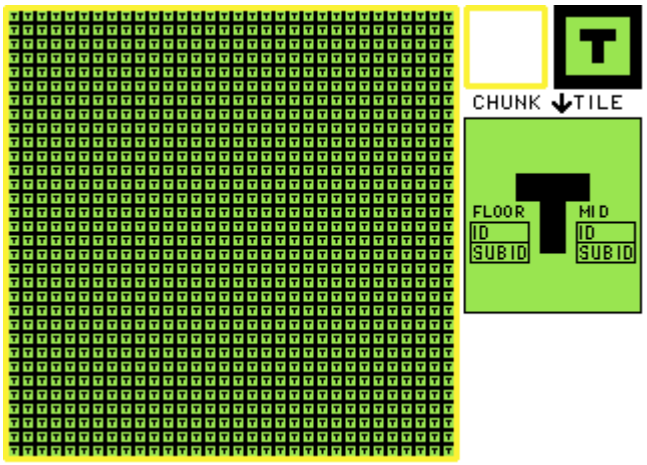
Η διαδικασία παραγωγής περιλαμβάνει τον ορισμό του seed και του μεγέθους του κόσμου, την αρχικοποίηση των εργαλειων που χρησιμοποιεί το σύστημα με βαση το seed, τρία ξεχωριστά στάδια παραγωγής, και τέλος την αποθήκευση των δεδομένων που παρήχθησαν στην μόνιμη μνήμη.

4.3.2 Σύστημα παραγωγής κόσμου : Μορφή Κόσμου

Πριν περάσουμε στην παράθεση του κώδικα, θα γίνει μια επισκόπηση της μορφής που έχει ένας κόσμος στο Fabritaku. Την μορφή αυτή μοιράζεται και το συστήμα κόσμου.

Η μορφή ενός κόσμου είναι ένας πίνακας τιμών σε πρώτο στάδιο από ομάδες κελιών μεγέθους 32 * 32, που ορίζονται ως Chunks, και σε δεύτερο στάδιο τα ίδια τα κελιά, που ορίζονται ως Tiles. Για τους σκοπούς του συστήματος παραγωγής κόσμου συγκεκριμένα η ομαδοποίηση σε Chunks αγνοείται και ορίζονται τα κελιά απευθείας.

Κάθε κελί έχει στο κατώτερο του επίπεδο ένα σύνολο τεσσάρων τιμών, ένα id και ένα subId για τον τύπο του εδάφους, που ορίζεται ως FLOOR, και ένα id και ένα subId για τον τύπο του στοιχείου που βρίσκεται πάνω από αυτό, που ορίζεται ως MID.
Σχηματικά:



Εικόνα 10 - Σχηματική απεικόνιση της μορφής μια ομάδας κελιών

4.3.3 Σύστημα παραγωγής κόσμου : Παρουσίαση και επεξήγηση κώδικα

Η διαδικασία που περιγράφηκε σε μορφή κώδικα έχει ως εξής, απο την κλήση της κατα τον constructor της οθόνης κόσμου:

```
if(!Gdx.files.local(path).exists()){
    WorldGenerator generator = new WorldGenerator( worldWidth: 80, worldHeight: 80);
    generator.setSeed((int) TimeUtils.millis());
    generator.initRNG();
    generator.generationStageOne();
    generator.generationStageTwo();
    generator.generationStageThree();
    generator.toStorage(path);
    generator = null;
    //Hint the garbage collector to collect the world generation leftovers
    System.gc();
}
```

Εικόνα 11 - Κώδικας για την κύρια περίπτωση χρήσης του συστήματος κόσμου

Ο κώδικας του συστήματος περιορίζεται στην κλαση WorldGenerator

```
/**
 * Initialize a world generation object
 * @param worldWidth horizontal size of the world,in chunks
 * @param worldHeight vertical size of the world,in chunks
 */
public WorldGenerator(int worldWidth ,int worldHeight){
    this.worldWidth = worldWidth;
    this.worldHeight = worldHeight;
    utils.WX = worldWidth * chunkWidth;
    utils.WY = worldHeight * chunkHeight;
    initArrays();
    this.internalRegistry = new TileRegistry();
}
```

Εικόνα 12 - Κώδικας του constructor της κλάσης WorldGenerator

Επιπρόσθετα της διαδικασίας που περιγράφηκε, αρχικοποιούνται οι πίνακες που αναπαριστούν τον κόσμο με βάση το μέγεθος που ζητήθηκε καθώς και εσωτερικοι βοηθητικοί μηχανισμοί.

Στην συνέχεια, αρχικοποιούνται οι γεννήτριες παραγωγής ψευδοτυχαίων αριθμών, εφόσον αυτο δεν έχει γίνει ήδη.

```
public void initRNG(){
    if(!generating) {
        this.generating = true;
        this.rng = new RandomXS128(seed);
        this.openSimplex2F = new OpenSimplex2F(seed);
    }
}
```

Εικόνα 13 - Κώδικας της αρχικοποίησης γεννητριών παραγωγής ψευδοτυχαίων αριθμών της κλάσης WorldGenerator

Τα τρία στάδια παραγωγής:

```
public void generationStageOne(){

    for (int i = 0; i < utils.WX; i++) {
        for (int j = 0; j < utils.WY; j++) {

            double t;
            t = (openSimplex2F.noise2_XBeforeY( x: i*Utils.NOISE_CONSTANT, y: j*Utils.NOISE_CONSTANT)
                    + 0.25 * openSimplex2F.noise2_XBeforeY( x: i*Utils.NOISE_CONSTANT_SECONDARY, y: j*Utils.NOISE_CONSTANT_SECONDARY))/1.25;

            int id;
            if (t>0.5d){
                id = Tiles.COBBLE;
            } else if (t>.35d){
                id = Tiles.DIRT;
            } else if (t>-0.4d){
                id = Tiles.GRASS;
            } else {
                id = Tiles.SAND;
            }
            worldIds[i][j][Tile.FLOOR] = id;

        }
    }
}
```

Εικόνα 14 - Κώδικας του πρώτου σταδίου παραγωγής της κλάσης WorldGenerator

Στο πρώτο στάδιο παράγεται το έδαφος του κόσμου, υπο την μορφή ενός heightmap, με τις μεγαλύτερες τιμές να συμβολίζουν βουνά, και καθως μικραίνουν παρυφές και πεδιάδες.

```
public void generationStageTwo(){
    long seed = this.seed % 100;
    OpenSimplex2F alt = new OpenSimplex2F(seed);

    for (int i = 0; i < utils.WX; i++) {
        for (int j = 0; j < utils.WY; j++) {

            double floraNoise = alt.noise3_Classic( x: i*Utils.NOISE_CONSTANT_VERY_HARSH, y: j*Utils.NOISE_CONSTANT_VERY_HARSH,Utils.FLORA_LAYER);

            switch (worldIds[i][j][Tile.FLOOR]){

                case Tiles.COBBLE:
                    worldIds[i][j][Tile.MID] = Tiles.COBBLE_WALL;
                    break;

                case Tiles.GRASS:
                    if(floraNoise>0.5d)
                        worldIds[i][j][Tile.MID] = Tiles.BUSH;
                    break;

                case Tiles.DIRT:
                    if(alt.noise3_Classic( x: i*Utils.NOISE_CONSTANT_HARSH, y: j*Utils.NOISE_CONSTANT_HARSH,Utils.FLINT_LAYER)>0.5d)
                        worldIds[i][j][Tile.FLOOR] = Tiles.FLINT_DIRT;
                    break;

                case Tiles.SAND:
                    if (floraNoise > 0.9d){
                        worldIds[i][j][Tile.MID] = Tiles.DRIED_BUSH;
                    } else if (floraNoise > 0.85d){
                        worldIds[i][j][Tile.MID] = Tiles.COTTON_PLANT;
                    }
            }

        }
    }
}
```

Εικόνα 15 - Κώδικας του δεύτερου σταδίου παραγωγής της κλάσης WorldGenerator

Στο δεύτερο στάδιο παράγονται οι πόροι του κόσμου(π.χ. Φυτά, μεταλλεύματα) , χρησιμοποιώντας ως βάση τα αποτελεσματα του πρωτου σταδίου, για την σωστή κατανομή τους.

```
public void generationStageThree(){
    for (int x = 0; x < utils.WX; x++) {
        for (int y = 0; y < utils.WY; y++) {

            worldSubIds[x][y][Tile.FLOOR] = (byte)rng.nextInt(internalRegistry.get(worldIds[x][y][Tile.FLOOR]).hasSubId ?
                    internalRegistry.get(worldIds[x][y][Tile.FLOOR]).subIdCount : 1);

            worldSubIds[x][y][Tile.MID]   = (byte)rng.nextInt(internalRegistry.get(worldIds[x][y][Tile.MID]).hasSubId ?
                    internalRegistry.get(worldIds[x][y][Tile.MID]).subIdCount : 1);
        }
    }
}
```

Εικόνα 16 - Κώδικας του τρίτου σταδίου παραγωγής της κλάσης WorldGenerator

Το τρίτο στάδιο ορίζει τα subId των κελιών, πληροφορία που αποσκοπεί στον οπτικό διαχωρισμό κελιών με το ίδιο id, προσπαθώντας με την χρήση διαφορετικών γραφικών βαση του subId να κάνει την οπτικά επαναλαμβανόμενη φύση των κελιών λιγότερο αντιληπτή.

4.3.4 Σύστημα παραγωγής κόσμου : Τεχνικές λεπτομέρειες

Με το πέρας του τρίτου σταδίου, τα δεδομένα αποθηκεύονται στην μόνιμη μνήμη, και γίνεται μια κλήση στον garbage collector, καθώς ο όγκος του αντικειμένου generator είναι μεγάλος, συνήθως της τάξεως των εκατοντάδων MB.

Για την διαδικασία της παραγωγής γίνεται εκτενής χρήση της υλοποίησης noise OpenSimplex2F του KdotJPG. Οι συναρτήσεις που παρέχει η υλοποίηση αυτή επιτρέπουν στο σύστημα παραγωγής κόσμων να παράξει έναν πιστευτό κόσμο με μικρά κομμάτια κώδικα. Ως είσοδο, λαμβάνουν τιμές double, όσες και ο αριθμός των διαστάσεων που πραγματεύονται, και ως έξοδο επιστρέφουν μια τιμή double.

Την double τιμή αυτή μπορούμε να χρησιμοποιήσουμε ως τυχαίο αριθμο για να παράξουμε τις τιμές για κελιά. Ως είσοδο τροφοδοτούμε την θέση x,y στο χώρο του κόσμου κάθε κελιού, αφου πρώτα την πολλαπλασιάσουμε με επιλεγμένες σταθερές, καθώς με μοναδιαίο βήμα (π.χ (0,0) -> (0,1)) οι τιμές είναι χαώδης. Για την παραγωγή των πόρων προσθέτουμε και μια επιλεγμένη τρίτη τιμη double, ανάλογα της ομάδας των πόρων.

4.3.5 Σύστημα παραγωγής κόσμου : Προτάσεις επέκτασης

Η διαδικασία παραγωγής που έχει υλοποιηθεί είναι αρκετά απλή και μπορεί να επεκταθεί με αρκετούς τρόπους.

Για παράδειγμα, μπορουν να προστεθούν παραπάνω πληροφορίες για τις παραμέτρους του κόσμου όπως θερμοκρασία, υγρασία, βροχόπτωση.

Μπορούν να τροποποιηθούν οι επιλεγμένες τιμές, ή να προστεθούν καινούριες, για να αλλάξει τελείως το αποτέλεσμα ή να προστεθούν σημεία ενδιαφέροντος.

Η παρούσα υλοποίηση αγνοεί την έννοια των Chunk και παράγει όλο τον κόσμο μαζί, χρησιμοποιώντας μεγάλο μέγεθος μνήμης, θα μπορούσε να μετατραπεί έτσι ώστε να μπορεί να παραχθεί κάθε κομμάτι ξεχωριστά, και ενδεχομένως να παραλληλοποιηθεί.

4.4 Σύστημα κόσμου - Κύριο Σύστημα

4.4.1 Σύστημα κόσμου : Επισκόπηση λειτουργίας και στόχων

Το σύστημα κόσμου αναλαμβάνει να διαχειρίζεται τα δεδομένα που σχετίζονται με την κατασταση του κόσμου και τα συστήματα που επιδρούν πάνω σε αυτα, συμπεριλαμβανομένου και του χαρακτήρα του παίκτη.

Για παράδειγμα, με τα δεδομένα που θα λάβει από το σύστημα παραγωγής κόσμου, θα αρχικοποιήσει τον κόσμο αυτο στην μνήμη, θα ενεργοποιήσει τα συστήματα που επιδρούν πάνω σε αυτόν και θα ξεκινήσει έτσι την διαδικασία του παιχνιδιού.

Προγραμματιστικά είναι ένα κεντρικό σημείο που συγκεντρώνει τα αντικείμενα που είναι απαραίτητα για την λειτουργία του κόσμου, ορίζει την σειρά αρχικοποίησης και ανανέωσης τους.

4.4.2 Σύστημα κόσμου : Παρουσίαση και επεξήγηση κώδικα

```
public World(FileHandle worldHandle, Renderer renderer, Registry registry, Dispatcher dispatcher, Player player){
    this.chunkLoader = new ChunkLoader( world: this,worldHandle);
    chunkLoader.initWorld();
    this.renderer = renderer;
    this.registry = registry;
    this.chunks = new IntMap<>();
    this.itemDrops = new ObjectSet<>();
    this.player = player;
    this.dispatcher = dispatcher;
    this.b2dWorld = new com.badlogic.gdx.physics.box2d.World(new Vector2( x: 0, y: 0), doSleep: true);
    this.systems = new Systems( world: this);
}

public void update(float delta){
    chunkLoader.update();
    systems.update(delta);
    b2dWorld.step(delta, velocityIterations: 6, positionIterations: 2);
}
```

Εικόνα 17 - Κώδικας αρχικοποίησης και ανανέωσης της κλάσης World

Η αρχική κλάση του συστήματος είναι η World, που αρχικοποιεί τα υπόλοιπα συστήματα και φροντίζει να τα ανανεώσει, όταν αυτό χρειάζεται.

Για λόγους τμηματοποίησης κώδικα η κλάση World δεν πράττει η ίδια όλες τις λειτουργίες που αφορούν τον κόσμο, που διενεργούνται από τα άλλα συστήματα, αλλα συγκεντρώνει τα δεδομένα που τον αφορούν στις κλάσεις Chunk και Tile (δομικά όπως αναφέρονται στο 4.3.2)

Η κλάση World παρέχει όμως βοηθητικές συναρτήσεις για την προσπέλαση των δεδομένων τις:

```java
public Chunk getChunk(int indexX ,int indexY){
    return chunks.get(indexX*worldHeight+indexY);
}

public Chunk getChunk(int index){
    return chunks.get(index);
}

@Null
public Chunk getChunkByPosition(float x,float y){
    if (validatePosition(x,y)){
        int chunkIndexX = (int)(x/chunkWidth);
        int chunkIndexY = (int)(y/chunkHeight);
        return getChunk(chunkIndexX,chunkIndexY);
    } else {
        return null;
    }
}

/**
 * Validates if the coordinates given are valid world coordinates
 * @param x coordinate in world units [0-worldWidth*chunkWidth)
 * @param y coordinate in world units [0-worldHeight*chunkHeight)
 * @return True if coordinates are valid, false if not
 */
public boolean validatePosition(float x, float y){
    return (x>= 0 && y>= 0 && x< worldWidth*chunkWidth && y< worldHeight*chunkWidth);
}

@Null
public Tile getTileByPosition(float x,float y){
    int chunkIndexX = (int)(x/chunkWidth);
    int chunkIndexY = (int)(y/chunkHeight);
    if (validatePosition(x,y) && getChunk(chunkIndexX, chunkIndexY)!= null){
        Chunk chunk = getChunk(chunkIndexX,chunkIndexY);
        return chunk.getTile(((int)(x - chunk.position.x)),(int)(y-chunk.position.y));
    } else {
        return null;
    }
}
```

Εικόνα 18 - Κώδικας βοηθητικών συναρτήσεων της κλάσης World

Κυρίως χρησιμεύει στην μετατροπή των εισόδων (συντεταγμένων κόσμου ή δεικτών) σε χρήσιμη πληροφορία, δηλαδή αντικείμενα Chunk ή Tile.

Οι κλάσεις Chunk και Tile βρίσκονται στην ίδια γραμμή σκέψης, διατηρούν δεδομένα χωρίς λειτουργίες, πλην μερικών βοηθητικών συναρτήσεων.

```java
public class Chunk {

    public final World world;

    public boolean dirty = false;

    public Vector2 position;
    public Vector3 center;
    public Vector3 dimensions;

    public IntMap<Tile> tiles;
    public IntMap<Body> bodies;

    public Chunk(float x, float y, World world){
        this.world = world;
        this.position = new Vector2(x,y);
        this.dimensions = new Vector3(world.chunkWidth,world.chunkHeight, z: 0);
        this.center = new Vector3( x: position.x+dimensions.x/2, y: position.y+dimensions.y/2, z: 0);
        tiles = new IntMap<>( initialCapacity: world.chunkWidth*world.chunkHeight);
        bodies = new IntMap<>( initialCapacity: world.chunkWidth*world.chunkHeight);
        for (int i = 0; i < world.chunkWidth; i++) {
            for (int j = 0; j < world.chunkHeight ; j++) {
                Tile tile = new Tile( x: position.x+i, y: position.y +j, index: i * world.chunkHeight + j);
                tiles.put(i * world.chunkHeight + j,tile);
            }
        }
    }

    public Tile getTile(int x, int y) {
        return tiles.get(x*((int)dimensions.y) + y);
    }

}
```

Εικόνα 19 - Κώδικας της κλάσης Chunk

Τα αντικείμενα dimensions και center χρησιμοποιούν διανύσματα τριών διαστάσεων(Vector3) για λόγους αποφυγής ενδιάμεσων αντικειμένων Vector2 για την χρήση συναρτήσεων της κλάσης OrthographicCamera του libGDX κατα την διαδικασία του rendering. Η τρίτη διάσταση μπορεί απλά να αγνοηθεί για τους σκοπούς μας.

Ένα αντικείμενο της κλάσης World έχει ένα σύνολο αντικειμένων Chunk, που με την σειρά του έχει ένα σύνολο αντικειμένων Tile.
Για τις πρώτες δύο κλάσεις γίνεται χρήση 1D IntMaps για τα σύνολα αυτά, με index που υπολογίζονται με την γενική μορφή index = x * yLength + y.

```java
public class Tile {
    public static final int FLOOR = 0;
    public static final int MID = 1;

    public static final int NUM_LAYERS =2;

    public Vector2 position;

    public int index;

    public int[] id = new int[NUM_LAYERS];
    public byte[] subId = new byte[NUM_LAYERS];

    Tile(float x,float y, int index){
        this.position = new Vector2(x,y);
        this.index = index;
    }

    public int getId(int layer) {
        return id[layer];
    }

    public Tile setId(int id,int layer) {
        this.id[layer] = id;
        return this;
    }

    public byte getSubId(int layer) {
        return subId[layer];
    }

    public Tile setSubId(byte subId,int layer) {
        this.subId[layer] = subId;
        return this;
    }
}
```

Εικόνα 20 - Κώδικας της κλάσης Tile

Τα αντικείμενα Tile χρησιμοποιούν δύο απλούς πίνακες για την διατήρηση των δεδομένων τους, και τις συναρτήσεις που τα μεταβάλουν.

4.4.3 Σύστημα κόσμου : Υποσύστημα διαχείρισης φόρτωσης Chunk - Βοηθητικό Σύστημα

4.4.3.1 Υποσύστημα διαχείρισης φόρτωσης Chunk - : Επισκόπηση λειτουργίας και στόχων

Στενά δεμένο με το σύστημα κόσμου είναι το υποσύστημα διαχείρισης φόρτωσης Chunk, στο εξής και γνωστό και ως σύστημα ChunkLoading, που είναι υπεύθυνο για δύο λειτουργίες:

1. Να πραγματοποιήσει τις διαδικασίες που χρειάζονται για να μεταφερθούν τα δεδομένα από την μόνιμη μνήμη στην μνήμη της εφαρμογής και να αρχικοποιηθούν τα αντίστοιχα αντικείμενα, και αντίστροφα να καταστραφούν και να μεταφερθούν στην μόνιμη μνήμη.
2. Να κρίνει πότε είναι η κατάλληλη στιγμή για αυτές τις διαδικασίες.

4.4.3.2 Υποσύστημα διαχείρισης φόρτωσης Chunk : Μορφή αρχείου μόνιμης μνήμης

Για την αναπαράσταση των δεδομένων των κύριων δεδομένων του κόσμου (όπως αναφέρονται στο κεφάλαιο 4.3.2) χρησιμοποιείται ένα αρχείο τύπου .ftwf, π.χ. <World Name>.ftwf.

Η μορφή του αρχείου έχει ως εξής:

HEADER:

Αριθμός Byte	Πληροφορία	Τύπος δεδομένου
0-3	Το seed του κόσμου	(int)
4-7	Το μεγεθος του κοσμου σε Chunk, οριζόντια	(int)
7-11	Το μεγεθος του κοσμου σε Chunk, κάθετα	(int)
11-15	Το μέγεθος των Chunk σε Tile, οριζόντια	(int)
16-19	Το μέγεθος των Chunk σε Tile, κάθετα	(int)

Τα byte μετα το 19ο είναι οργανωμένα σε ομάδες των 10248 byte, που αντιπροσωπεύουν ένα Chunk.
Για παράδειγμα:
Με δεδομένο H_S= 20 byte για το μέγεθος του header, C_S = 10248 το μέγεθος του Chunk και i τον δείκτη του Chunk

Αριθμός Byte	Πληροφορία	Τύπος δεδομένου
H_S + i * C_S + 0 μέχρι H_S + i * C_S + 3	Η οριζόντια θέση x του Chunk στον κόσμο	(int)
H_S + i * C_S + 4 μέχρι H_S + i * C_S + 7	Η κάθετη θέση y του Chunk στον κόσμο	(int)
H_S + i * C_S + 8 μέχρι H_S + i * C_S + 11	Τιμή id στο layer FLOOR του Tile με δείκτη 0	(int)
H_S + i * C_S + 12	Τιμή subid στο layer FLOOR του Tile με δείκτη 0	(byte)
H_S + i * C_S + 13 μέχρι H_S + i * C_S + 16	Τιμή id στο layer MID του Tile με δείκτη 0	(int)
H_S + i * C_S + 17	Τιμή subid στο layer MID του Tile με δείκτη 0	(byte)
...	...	...
H_S + i * C_S + 8 + (n*10)+ 0 μέχρι H_S + i * C_S + 8 + (n*10)+ 3	Τιμή id στο layer FLOOR του Tile με δείκτη n	(int)
H_S + i * C_S + 8 + (n*10)+ 4	Τιμή subid στο layer FLOOR του Tile με δείκτη n	(byte)
H_S + i * C_S + 8 + (n*10)+ 5 μέχρι H_S + i * C_S + 8 + (n*10)+ 8	Τιμή id στο layer MID του Tile με δείκτη n	(int)
H_S + i * C_S + 8 + (n*10)+ 9	Τιμή subid στο layer MID του Tile με δείκτη n	(byte)

4.4.3.3 Υποσύστημα διαχείρισης φόρτωσης Chunk : Παρουσίαση και επεξήγηση κώδικα

Το σύστημα αποτελείται από δύο κλάσεις, τις ChunkLoader και Log.
Η κλάση ChunkLoader είναι υπεύθυνη για τις λειτουργίες, χρησιμοποιώντας αντικείμενα Log για την διατήρηση πληροφοριών σχετικα με κάθε Chunk.

```
//World owning the loader
public final World world;
public FileHandle worldHandle;
//Logs used to keep track of time chunks have spent inactive
public ObjectMap<Chunk,Log> chunkLoadLogs;

public final IntSet shouldBeLoadedChunks;

public ChunkLoader(World world, FileHandle worldHandle){
    this.world = world;
    this.worldHandle = worldHandle;
    this.chunkLoadLogs = new ObjectMap<>();
    this.shouldBeLoadedChunks =new IntSet();
}

public void initWorld() {
    BufferedInputStream stream = new BufferedInputStream(worldHandle.read());
    byte[] worldData = new byte[Utils.HEADER_BYTE_SIZE];
    try {
        stream.read(worldData);
        //read seed
        world.seed = ByteBuffer.wrap(worldData, offset: 0, length: 4).getInt();
        //read world width
        world.worldWidth = ByteBuffer.wrap(worldData, offset: 4, length: 4).getInt();
        //read world height
        world.worldHeight = ByteBuffer.wrap(worldData, offset: 8, length: 4).getInt();
        //read chunk width
        world.chunkWidth = ByteBuffer.wrap(worldData, offset: 12, length: 4).getInt();
        //read chunk height
        world.chunkHeight = ByteBuffer.wrap(worldData, offset: 16, length: 4).getInt();

        stream.close();

    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Εικόνα 21 - Κώδικας αρχικοποίησης της κλάσης ChunkLoader

Κατα την αρχικοποίηση του ο ChunkLoader διαβάζει τα γενικά δεδομένα του κόσμου απο το header του αρχείου στην μόνιμη μνήμη, ετσι ωστε να μπορεί αρχικοποιήσει το αντικείμενο world.

Ο ChunkLoader κρατάει δύο σημαντικά σύνολα δεδομένων, τα αρχεία που έχουν χρονικές σφραγίδες για κάθε Chunk και τα Chunk στα οποία αντιστοιχούν στο ObjectMap chunkLoadLogs, καθώς και ένα σύνολο δεικτών απο Chunk που θα έπρεπε να είναι φορτωμένα στο IntSet shouldBeLoadedChunks.

Τα αρχεία με τις χρονικές σφραγίδες έχουν την εξής μορφή:

```
public class Log {
    boolean active;
    //Timestamp the chunk was last seen active
    long timeLastActiveMillis;

    public Log(boolean active) {
        this.active = active;
        this.timeLastActiveMillis = TimeUtils.millis();
    }

    public long timeSpentInactive() { return TimeUtils.timeSinceMillis(timeLastActiveMillis); }

    public boolean isActive() { return active; }
}
```

Εικόνα 22 - Κώδικας της κλάσης Log

Στα αρχεία αυτά μας ενδιαφέρει να αποθηκεύουμε το αν είναι αυτη την στιγμή ενεργό ένα Chunk, και αν δεν είναι, την τελευταία στιγμή που ήταν.
Με αυτές τις πληροφορίες μπορούμε να αποφασίσουμε πότε είναι η κατάλληλη στιγμή να αποδεσμεύσουμε ενα Chunk απο την μνήμη.

Την λειτουργία φόρτωσης και αποδέσμευσης αναλαμβάνει ο ChunkLoader, κάθε frame, εάν αυτή είναι αναγκαία.

```
public void update(){
    shouldBeLoadedChunks.clear();
    int x = (int)world.player.position.x/ world.chunkWidth;
    int y = (int)world.player.position.y/ world.chunkHeight;
    int playerChunkIndex = x* world.worldHeight + y;
    //Calculate which chunks should be loaded
    if (x>=0 && y>=0 && x< world.worldWidth && y< world.worldHeight) {
        shouldBeLoadedChunks.add(playerChunkIndex);
    }
```

Εικόνα 23 - Κώδικας της αρχής της συνάρτησης update της κλάσης ChunkLoader

Αρχικά ελέγχει το chunk στο οποίο βρίσκεται ο παίκτης και το προσθέτει στην λίστα με τα chunk που θα έπρεπε να είναι φορτωμένα, αφού πρώτα την αδειάσει.

```
int offset = world.worldHeight;
if(x>0 && y>0 && x< world.worldWidth-1 && y< world.worldHeight-1){
    shouldBeLoadedChunks.add(playerChunkIndex+1);//N Neighbour
    shouldBeLoadedChunks.add(playerChunkIndex+offset+1);//NE Neighbour
    shouldBeLoadedChunks.add(playerChunkIndex+offset);//E Neighbour
    shouldBeLoadedChunks.add(playerChunkIndex+offset-1);//SE Neighbour
    shouldBeLoadedChunks.add(playerChunkIndex-1);//S Neighbour
    shouldBeLoadedChunks.add(playerChunkIndex-offset-1);//SW Neighbour
    shouldBeLoadedChunks.add(playerChunkIndex-offset);//W Neighbour
    shouldBeLoadedChunks.add(playerChunkIndex-offset+1);//NW Neighbour
} else if (x==0 && y > 0 && y < world.worldHeight-1){
    shouldBeLoadedChunks.add(playerChunkIndex+1);//N Neighbour
    shouldBeLoadedChunks.add(playerChunkIndex+offset+1);//NE Neighbour
    shouldBeLoadedChunks.add(playerChunkIndex+offset);//E Neighbour
} else {
    if (x==0 && y==0){
        shouldBeLoadedChunks.add(playerChunkIndex+1);//N Neighbour
        shouldBeLoadedChunks.add(playerChunkIndex+offset+1);//NE Neighbour
        shouldBeLoadedChunks.add(playerChunkIndex+offset);//E Neighbour
    } else if(x==0 && y== world.worldHeight-1){
        shouldBeLoadedChunks.add(playerChunkIndex+offset);//E Neighbour
        shouldBeLoadedChunks.add(playerChunkIndex+offset-1);//SE Neighbour
        shouldBeLoadedChunks.add(playerChunkIndex-1);//S Neighbour
    } else if(x== world.worldWidth-1 && y == world.worldHeight-1){
        shouldBeLoadedChunks.add(playerChunkIndex-1);//S Neighbour
        shouldBeLoadedChunks.add(playerChunkIndex-offset-1);//SW Neighbour
        shouldBeLoadedChunks.add(playerChunkIndex-offset);//W Neighbour
    } else if (x== world.worldWidth-1 && y == 0){
        shouldBeLoadedChunks.add(playerChunkIndex+1);//N Neighbour
        shouldBeLoadedChunks.add(playerChunkIndex-offset);//W Neighbour
        shouldBeLoadedChunks.add(playerChunkIndex-offset+1);//NW Neighbour
    }
}
```

Εικόνα 24 - Κώδικας της συνάρτησης update της κλάσης ChunkLoader

Στην συνέχεια με βάση το chunk του παίκτη υπολογίζεται ποια άλλα chunk πρέπει να φορτωθούν βάση γειτονικοτητας.

```
//Load the chunks which are supposed to be loaded but are not
for (int index : shouldBeLoadedChunks.iterator().toArray().items) {
    if(index>=0 && index<= world.worldWidth* world.worldHeight-1){
        if(world.chunks.get(index)==null){
            loadChunk(index);
        }
        //Populate the activity timestamps
        chunkLoadLogs.get(world.getChunk(index)).timeLastActiveMillis = TimeUtils.millis();
        chunkLoadLogs.get(world.getChunk(index)).active = true;
    }
}

//Calculate which chunks do not need to be loaded and set them inactive
for (IntMap.Entry<Chunk> chunkEntry : world.chunks) {
    if(!shouldBeLoadedChunks.contains(chunkEntry.key)){
        if(world.getChunk(chunkEntry.key)!=null){
            chunkLoadLogs.get(world.getChunk(chunkEntry.key)).active = false;
        }
    }
}
//Unload inactive chunks if they've been inactive for too long
for (IntMap.Entry<Chunk> chunkEntry : world.chunks) {
    if(chunkLoadLogs.get(chunkEntry.value).timeSpentInactive() > Utils.CHUNK_INACTIVITY_TIME_LIMIT){
        unloadChunk(chunkEntry.key);
        //Break to avoid taking too long in a single frame if there's many chunks to unload
        break;
    }
}
```

Εικόνα 25 - Κώδικας της συνάρτησης update της κλάσης ChunkLoader

Αφού υπολογίσει ποια chunk πρέπει να είναι φορτωμένα μπορεί να προχωρήσει στις φορτώσεις και αποδεσμεύσεις.

Για τις φορτώσεις, ελέγχει αν το chunk είναι ήδη στην μνήμη. Αν το chunk δεν είναι στην μνήμη το φορτώνει και ανεξάρτητα απο το αν ήταν η όχι επικαιροποιεί το αντίστοιχο αντικείμενο Log.

Για τις αποδεσμεύσεις, ελέγχει τα chunk που είναι ήδη φορτωμένα και αν ο παίκτης δεν είναι σε κάποιο γειτονικό chunk τα ορίζει αδρανή.
Σε δεύτερο στάδιο, αν ένα chunk είναι αδρανές για ένα διάστημα μεγαλύτερο από αυτό που επιθυμούμε, ο ChunkLoader θα προχωρήσει στην αποδέσμευση του.

Κατα την διαδικασία της αποδέσμευσης αποφεύγουμε να αποδεσμευουμε παραπάνω από ένα chunk την φορά καθώς η αποθήκευση των δεδομένων στην μόνιμη μνήμη είναι χρονοβόρα και μπορεί να προκαλέσει μεγάλη καθυστέρηση στην ολοκλήρωση του frame, κατι που μπορεί να είναι αντιληπτό από τον χρήστη και μή επιθυμητό.

```
public void writeChunk(int index){
    try {
        RandomAccessFile file = new RandomAccessFile(worldHandle.file(), mode: "rw");
        ByteBuffer chunkData = ByteBuffer.allocate(Utils.CHUNK_BYTE_SIZE);
        //write chunk x
        chunkData.putInt((int)world.getChunk(index).position.x);
        //write chunk y
        chunkData.putInt((int)world.getChunk(index).position.y);
        for (int i = 0; i < world.chunkWidth; i++) {
            for (int j = 0; j < world.chunkHeight; j++) {

                //read tile floor id
                chunkData.putInt(world.getChunk(index).getTile(i, j).getId(Tile.FLOOR));
                //read tile floor subId
                chunkData.put(world.getChunk(index).getTile(i, j).getSubId(Tile.FLOOR));

                //read tile mid id
                chunkData.putInt(world.getChunk(index).getTile(i, j).getId(Tile.MID));
                //read tile mid subId
                chunkData.put(world.getChunk(index).getTile(i, j).getSubId(Tile.MID));

            }
        }
        file.seek( pos: Utils.HEADER_BYTE_SIZE +((long) index*Utils.CHUNK_BYTE_SIZE));
        file.write(chunkData.array());
        file.close();
    } catch (IOException e) {
        e.printStackTrace();
    }

}
```

Εικόνα 26 - Κώδικας αποδέσμευσης chunk της κλάσης ChunkLoader

```
public void loadChunk(int chunkIndex){
    BufferedInputStream stream = new BufferedInputStream(worldHandle.read());
    byte[] chunkData = new byte[Utils.CHUNK_BYTE_SIZE];
    try {
        stream.skip(Utils.HEADER_BYTE_SIZE);
        stream.skip( n: (long) chunkIndex*Utils.CHUNK_BYTE_SIZE);
        //read the chunk data
        stream.read(chunkData);
        //read chunk x and y from chunk data
        Chunk chunk = new Chunk(ByteBuffer.wrap(chunkData, offset: 0, length: 4).getInt(),ByteBuffer.wrap(chunkData, offset: 4, length: 4).getInt(),world);
        world.chunks.put(chunkIndex,chunk);
        for (int i = 0; i < world.chunkWidth; i++) {
            for (int j = 0; j < world.chunkHeight; j++) {

                int index = Utils.CHUNK_POS_BYTE_SIZE + ((i * world.chunkHeight + j) * 10);
                //read tile floor id
                world.getChunk(chunkIndex).getTile(i,j).setId(ByteBuffer.wrap(chunkData, offset: index +0, length: 4).getInt(), Tile.FLOOR);
                //read tile floor subId
                world.getChunk(chunkIndex).getTile(i,j).setSubId(ByteBuffer.wrap(chunkData, offset: index + 4, length: 1).get(),Tile.FLOOR);

                //read tile mid id
                world.getChunk(chunkIndex).getTile(i,j).setId(ByteBuffer.wrap(chunkData, offset: index +5, length: 4).getInt(),Tile.MID);
                //read tile mid subId
                world.getChunk(chunkIndex).getTile(i,j).setSubId(ByteBuffer.wrap(chunkData, offset: index +9, length: 1).get(),Tile.MID);

            }
        }
        stream.close();

    } catch (IOException e) {
        e.printStackTrace();
    }
    chunkLoadLogs.put(world.chunks.get(chunkIndex),new Log( active: true));
}
```

Εικόνα 27 - Κώδικας φόρτωσης chunk της κλάσης ChunkLoader

4.5 Σύστημα Registry - Κοινό Σύστημα

4.5.1 Σύστημα Registry : Επισκόπηση λειτουργίας και στόχων

Το σύστημα Registry διατηρεί και αρχικοποιεί ορισμούς τύπων αντικειμένων(π.χ Ξύλο ,Βαμβάκι ,Πυρόλιθος) και τύπων κελιών(π.χ. Χώμα, Γρασίδι, Άμμος) και συνταγών που ορίζουμε εμείς. Οι ορισμοί αυτοι συμπεριλαμβάνουν ιδιότητες αλλα και τα δεδομένα για κάθε τύπο, επιτρέποντας μας να προσθέτουμε ορισμούς με εύκολο τρόπο σε ένα κεντρικό σημείο.

Προγραμματιστικά το σύστημα αποτελείται από την κλάση Registry ως κεντρικό μέρος για την αναζήτηση όλων των ειδών ορισμών, ενώ εσωτερικά κάθε διαφορετικό είδος ορισμών δηλώνεται σε ξεχωριστές κλάσεις.

4.5.2 Σύστημα Registry : Παρουσίαση και επεξήγηση κώδικα

```
public class Registry {
    public static final Registry registry = new Registry();

    private final TileRegistry tileRegistry;
    private final ItemRegistry itemRegistry;
    private final RecipeRegistry recipeRegistry;

    public Registry(){
        tileRegistry = TileRegistry.tileRegistry;
        itemRegistry = ItemRegistry.itemRegistry;
        recipeRegistry = RecipeRegistry.recipeRegistry;
    }

    public ObjectSet<TileDefinition> getAllTiles() { return tileRegistry.getAll(); }

    public TileDefinition getT(int tileID) { return tileRegistry.get(tileID); }

    public ObjectSet<ItemDefinition> getAllItems() { return itemRegistry.getAll(); }

    public ItemDefinition getI(int itemID) { return itemRegistry.get(itemID); }

    public RecipeDefinition matchRecipe(CraftingInventory craftingInventory, int recipeType) {
        return recipeRegistry.matchRecipe(craftingInventory,recipeType);
    }
}
```

Εικόνα 28 - Κώδικας της κλάσης Registry

Η κεντρική κλάση αρχικοποιεί τα επιμέρους κομμάτια της και προσφέρει συναρτήσεις προσπέλασης των δεδομένων τους ετσι ωστε να μην χρειάζεται η απευθείας αναζήτηση κάποιου συγκεκριμένου κομματιού.

Το κάθε είδος χωρίζεται σε δύο κλασεις, μια που ονομαστικά δηλώνει τους ορισμούς και τις ιδιότητες, αλλα και σε μορφή (προγραμματιστικού) αντικειμένου τις τιμές αυτών για κάθε ορισμό, και μια άλλη που αρχικοποιεί αυτούς τους ορισμούς και τις τιμές τους, και τους τοποθετεί σε δομές δεδομένων.

Για τα αντικείμενα:
Η κλάση που ονομαστικά ορίζει τα αντικείμενα:

```java
public class ItemDefinition {
    public static final int STACK_SIZE_DEFAULT = 64;
    public static final int STACK_SIZE_NOT_STACKABLE = 1;

    public static class Items{
        public static final int PEBBLE = 0;
        public static final int STICK = 1;
        public static final int STRING = 2;
        public static final int ROCK = 3;
        public static final int WOODEN_LOG = 4;
        public static final int FLINT_AXE = 5;
        public static final int FLINT_PICKAXE = 6;
        public static final int FLINT = 7;
        public static final int COTTON = 8;
        public static final int IRON_CLUMP = 9;
        public static final int COAL_CLUMP = 10;
        public static final int IRON_INGOT = 11;
        public static final int BASKET = 12;
        public static final int METAL_FURNACE = 13;
        public static final int IRON_TROPHY = 14;
    }
    public static class Attributes{
        public static final int COMPONENT = 65636;
        public static final int TOOL = 65637;
        public static final int PLACEABLE = 65638;
        public static final int FURNACE_FUEL = 65544;
    }
    public String name;
    public int id;
    public IntSet attributes;
    public IntMap<Attribute> attributeValues;
    public boolean stackable;
    public byte maxStackSize;

    public boolean is(int attribute) { return attributes.contains(attribute); }
}
```

Εικόνα 29 - Κώδικας της κλάσης ItemDefinition

Και η κλάση που αρχικοποιεί τους ορισμούς και τις τιμές τους:

```java
public class ItemRegistry {

    private final IntMap<ItemDefinition> definitionIntMap;
    private final ObjectSet<ItemDefinition> definitionSet;

    /**
     * Populate Item definitions
     */
    public ItemRegistry() {...}

    public IntSet getByAttribute(int attribute) {
        IntSet set = new IntSet();
        for (ItemDefinition ItemDefinition : definitionSet) {
            if (ItemDefinition.attributes.contains(attribute)) {
                set.add(ItemDefinition.id);
            }
        }
        return set;
    }

    public ObjectSet<ItemDefinition> getAll() { return definitionSet; }


    public ItemDefinition get(int id) { return definitionIntMap.get(id); }
}
```

Εικόνα 30 - Κώδικας της κλάσης ItemRegistry

Στον constructor γίνεται η αρχικοποίηση των τιμών:

```
//STICK
def = new ItemDefinition();
def.name = "STICK";
def.id = Items.STICK;
def.attributes = new IntSet();
def.attributes.addAll(Attributes.COMPONENT);
definitionIntMap.put(Items.STICK,def);
definitionSet.add(def);

                        //ROCK
                        def = new ItemDefinition();
                        def.name = "ROCK";
                        def.id = Items.ROCK;
                        def.attributes = new IntSet();
                        def.attributes.addAll(Attributes.COMPONENT);
                        definitionIntMap.put(Items.ROCK,def);
                        definitionSet.add(def);

        //FLINT
        def = new ItemDefinition();
        def.name = "FLINT";
        def.id = Items.FLINT;
        def.attributes = new IntSet();
        def.attributes.addAll(Attributes.COMPONENT);
        definitionIntMap.put(Items.FLINT,def);
        definitionSet.add(def);
```

Εικόνα 31 - Κώδικας παραδειγμάτων αρχικοποίησης τιμών ορισμών αντικειμένων του constructor της κλάσης ItemRegistry

Για τα κελιά:

```
public class TileDefinition {

    public static class Tiles{
        public static final int AIR = 0;
        public static final int COBBLE = 1;
        public static final int SAND = 2;
        public static final int DIRT = 3;
        public static final int GRASS = 4;
        public static final int COBBLE_WALL = 5;
        public static final int BUSH = 6;
        public static final int FLINT_DIRT = 7;
        public static final int DRIED_BUSH = 8;
        public static final int COTTON_PLANT = 9;
    }

    public static class Attributes{
        public static final int EMPTY = 65536;
        public static final int FLOOR = 65537;
        public static final int MID = 65538;
        public static final int TOP = 65539;
        public static final int HARVESTABLE = 65540;
        public static final int FLORA = 65541;
        public static final int COLLIDABLE = 65542;
    }

    public String name;
    public int id;
    public boolean isRTE;
    public boolean hasSubId;
    public byte subIdCount;
    public IntSet attributes;
    public IntMap<Attribute> attributeValues;

    public boolean is(int attribute) { return attributes.contains(attribute); }
}
```

Εικόνα 32 - Κώδικας της κλάσης TileDefinition

```java
public class TileRegistry {
    public static final TileRegistry tileRegistry = new TileRegistry();

    private final IntMap<TileDefinition> definitionIntMap;
    private final ObjectSet<TileDefinition> definitionSet;

    /**
     * Populate tile definitions
     */
    public TileRegistry(){...}

    public IntSet getByAttribute(int attribute){
        IntSet set = new IntSet();
        for (TileDefinition tileDefinition : definitionSet) {
            if (tileDefinition.attributes.contains(attribute)){
                set.add(tileDefinition.id);
            }
        }
        return set;
    }

    public ObjectSet<TileDefinition> getAll() { return definitionSet; }

    public TileDefinition get(int id) { return definitionIntMap.get(id); }
}
```

Εικόνα 33 - Κώδικας της κλάσης TileRegistry

```java
//COBBLE
def = new TileDefinition();
def.name = "COBBLE";
def.id = Tiles.COBBLE;
def.isRTE = false;
def.hasSubId = true;
def.subIdCount = (byte) 3;
def.attributes = new IntSet();
def.attributes.addAll(Attributes.FLOOR);
def.attributeValues = new IntMap<>();
definitionIntMap.put(Tiles.COBBLE,def);
definitionSet.add(def);

//SAND
def = new TileDefinition();
def.name = "SAND";
def.id = Tiles.SAND;
def.isRTE = false;
def.hasSubId = true;
def.subIdCount = (byte) 3;
def.attributes = new IntSet();
def.attributes.addAll(Attributes.FLOOR);
def.attributeValues = new IntMap<>();
definitionIntMap.put(Tiles.SAND,def);
definitionSet.add(def);
```

```java
//COBBLE_WALL
def = new TileDefinition();
def.name = "COBBLE_WALL";
def.id = Tiles.COBBLE_WALL;
def.isRTE = false;
def.hasSubId = true;
def.subIdCount = (byte) 1;
def.attributes = new IntSet();
def.attributes.addAll(Attributes.MID,Attributes.HARVESTABLE,Attributes.COLLIDABLE);
def.attributeValues = new IntMap<>();
def.attributeValues.put(Attributes.HARVESTABLE,new Harvestable(Items.ROCK, amount: 5,Tiles.AIR));
def.attributeValues.put(Attributes.COLLIDABLE,new Collidable(Colliders.FULL_BOX));
definitionIntMap.put(Tiles.COBBLE_WALL,def);
definitionSet.add(def);
```

Εικόνα 34 - Κώδικας παραδειγμάτων αρχικοποίησης τιμών ορισμών κελιών του constructor της κλάσης TileRegistry

Όπως φαίνεται και από το παράδειγμα “COBBLE_WALL” στην εικόνα 34 κάποιοι ορισμοί ενδέχεται να απαιτούν ιδιότητες που χρειάζονται παραπάνω δεδομένα περα της απλής ύπαρξής της ιδιότητας.

Στο συγκεκριμένο παράδειγμα ορίζονται οι ιδιότητες τύπου Harvestable και Collidable:

```
public class Harvestable implements Attribute {
    //The item the harvestable tile produces
    public int itemID;
    //The amount of items the harvest produces
    public int amount;
    //The tile the given tile turns into after harvest
    public int replacementTileID;

    public Harvestable(int itemID, int amount, int replacementTileID) {
        this.itemID = itemID;
        this.amount = amount;
        this.replacementTileID = replacementTileID;
    }
}

public class Collidable implements Attribute{
    public static class Colliders{
        public static final int NONE = 0;
        public static final int FULL_BOX = 1;
        public static final int HALF_BOX = 2;
        public static final int QUARTER_BOX = 3;
    }
    public int colliderType;
    public Collidable(int colliderType){
        this.colliderType = colliderType;
    }
}
```

Εικόνα 35 - Κώδικας των ιδιοτήτων Harvestable και Collidable των αντίστοιχων κλάσεων

Ομοίως και για το Registry των συνταγών.

4.6 Σύστημα Event - Κοινό Σύστημα

4.6.1 Σύστημα Event : Επισκόπηση λειτουργίας και στόχων

Το σύστημα Event επιτρέπει στα συστήματα να επικοινωνούν μεταξύ τους ακόμα και αν δεν έχουν στενή σύνδεση.
Δημιουργεί επίσης ένα κοινό τρόπο επικοινωνίας, ορίζοντας ένα κύριο τρόπο διεκπεραίωσης κάθε τύπου Event ανάλογα τον παραλήπτη αλλα επιτρέποντας οποιοδήποτε σύστημα να δημιουργήσει και να αποστείλει Event.

Προγραμματιστικά το σύστημα αυτό υλοποιεί το Observer pattern, με κεντρικό κόμβο την κλάση Dispatcher, που εκτελεί χρέη καταγραφής των διάφορων ενδιαφερόμενων για κάθε τύπου Event αλλά και χρέη αγγελιοφόρου οταν ενα Event χρήζει διαβίβαση προς τυχόν ενδιαφερόμενους.

4.6.2 Σύστημα Event : Παρουσίαση και επεξήγηση κώδικα

Ο κώδικας για την κλάση Dispatcher είναι απλός, χρειάζονται μόνο πίνακες για την διατήρηση των ενδιαφερόμενων για κάθε τύπου Event, και οι αντίστοιχες μέθοδοι για καταγραφή τους και την αποστολή των Event σε αυτούς, ανα περίπτωση.

```
public class Dispatcher {

    Array<MovementReceiver> movementReceivers = new Array<>();
    Array<WorldInteractionReceiver> worldInteractionReceivers = new Array<>();
    Array<UiReceiver> uiReceivers = new Array<>();
    Array<InventoryReceiver> inventoryReceivers = new Array<>();
    Array<CraftingReceiver> craftingReceivers = new Array<>();
    Array<ItemSelectionReceiver> itemSelectionReceivers = new Array<>();
    Array<MessageReceiver> messageReceivers = new Array<>();

    public void addReceiver(int eventType, EventReceiver receiver){
        switch (eventType){...}
    }

    public void dispatch(int eventType, Payload payload){
        switch (eventType){...}
        payload.free();
    }


}
```

Εικόνα 36 - Κώδικας της κλάσης Dispatcher

Για την καταγραφή:

```
case Events.MESSAGE:
    messageReceivers.add((MessageReceiver) receiver);
    break;
```

Εικόνα 37 - Κώδικας περίπτωσης καταγραφής Receiver της addReveiver() της κλάσης Dispatcher

Και για την αποστολή:

```
case Events.MESSAGE:
    MessagePayload messagePayload = (MessagePayload) payload;
    for (MessageReceiver messageReceiver : messageReceivers) {
        messageReceiver.receive(messagePayload);
    }
    break;
```

Εικόνα 38 - Κώδικας περίπτωσης αποστολής Event της dispatch() της κλάσης Dispatcher

Για την υλοποίηση του μηχανισμού αυτου δημιουργήθηκαν τα interface EventReceiver και Payload.

Έστω <EventType> το όνομα ενός τύπου Event.
Κάθε τέτοιος τύπος Event ορίζεται από δύο στοιχεία, μια κλάση με όνομα <EventType>Payload που ορίζει τα πιθανά δεδομένα που μπορεί να φέρει ένα φορτίο(payload) του αντίστοιχου Event και ένα interface με όνομα <EventType>Receiver που κληρονομούν οι ενδιαφερόμενοι για να λάβουν τα αντίστοιχα Event αλλα και για να υλοποιήσουν τις διαδικασίες που επιθυμούν πάνω ή με βάση τα δεδομένα του φορτίου.

Για παράδειγμα, τα Event τύπου Message, που χρησιμοποιούνται για την αποστολή μηνυμάτων προς τον χρήστη:

```
public interface MessageReceiver extends EventReceiver {
    void receive(MessagePayload payload);
}

public class MessagePayload implements Payload {
    public static final Pool<MessagePayload> pool = Pools.get(MessagePayload.class, max: 10);

    private static final String emptyString = "";

    public String message;

    @Override
    public void reset() { message = emptyString; }

    @Override
    public void free() { pool.free( object: this); }
}
```

Εικόνα 39 - Αποκόμματα κώδικα των κλάσεων MessageReceiver και MessagePayload

Η διεπαφή χρήστη λαμβάνοντας ένα Event τύπου Message θα προχωρήσει στις προβλεπόμενες ενέργειες και μπορεί με την σειρά της να στείλει δικά της Events:

```
@Override
public void receive(MessagePayload payload) {
    text.setText(payload.message);
    mainTable.setVisible(true);
    if (ui.inventoryAndCrafting.mainTable.isVisible()){
        UiPayload uiPayload = UiPayload.pool.obtain();
        uiPayload.operation = UiPayload.Operations.TOGGLE;
        uiPayload.element = UiPayload.UiElements.INVENTORY_AND_CRAFTING;
        ui.dispatcher.dispatch(Events.UI,uiPayload);
    }
}
```

Εικόνα 40 - Απόκομμα κώδικα της κλάσης MessageUi

Το σύνολο των διαθέσιμων Event ορίζεται στην κλάση Events:

```
public class Events {

    public static final int MOVEMENT = 0;
    public static final int WORLD_INTERACTION = 1;
    public static final int UI = 2;
    public static final int INVENTORY = 3;
    public static final int CRAFTING = 4;
    public static final int ITEM_SELECTION = 5;
    public static final int MESSAGE = 6;
}
```

Εικόνα 41 - Κώδικας της κλάσης Events

4.6.3 Σύστημα Event : Τεχνικές λεπτομέρειες

Για την διαχείριση των αντικειμένων φορτίων χρησιμοποιείται το interface Poolable του libGDX το οποίο επιτρέπει την επαναχρησιμοποίηση αντικειμένων αντί για την δημιουργία καινούργιων κάθε φορά που χρειάζονται, μειώνοντας έτσι τις πιθανότητες να κληθεί ο garbage collector για να ελευθερώσει την μνήμη. Τα κάθε είδους φορτία κληρονομούν το interface αυτό από το interface Payload το οποίο και υλοποιούν.
Παράδειγμα της χρήσης αυτής παρουσιάζεται στην εικόνα 39.

Επίσης, για σκοπούς χωρικής βελτίωσης λόγο της εκτενής χρήση τους τόσο εδω, οσο και σε όλο το πρόγραμμα, επιλέχθηκε η χρήση public static final int και όχι enum για τον ορισμό των διαφόρων ειδών.

4.7 Σύστημα Rendering - Κύριο Σύστημα

4.7.1 Σύστημα Rendering : Επισκόπηση λειτουργίας και στόχων

Το σύστημα Rendering είναι υπεύθυνο για την απεικόνιση του κόσμου του παιχνιδιού στον παίκτη. Κάθε frame προσπελαύνει τα δεδομένα του κόσμου αλλα και της διεπαφής χρήστη και αποφασίζει τι πρέπει να απεικονιστεί και που πρέπει να απεικονιστεί.

Προγραμματιστικά αποτελείται από την κύρια κλάση Renderer και τις βοηθητικές κλάσεις ItemRegions, TileRegions και PlayerAnimations.
Η κλάση Renderer περιέχει τον κώδικα που απαιτείται για την απεικόνιση ενώ οι βοηθητικές κλάσεις περιέχουν συλλογές από τα γραφικά στοιχεία που χρειάζονται,ανα τύπο.

4.7.2 Σύστημα Rendering : Παρουσίαση και επεξήγηση κώδικα

```
public class Renderer implements Disposable {
    //How many tiles are rendered at once,from side to side
    public static final float VIEW_DISTANCE = 20f;
    public static final float CAMERA_ZOOM_MIN = 0.1f;
    public static final float CAMERA_ZOOM_MAX = 32f;


    public final SpriteBatch batch;
    public final OrthographicCamera camera;
    public final TextureAtlas atlas;
    public final Registry RY;
    public final TileRegions tileRegions;
    private final ItemRegions itemRegions;

    private final PlayerAnimations playerAnimations;
    private float playerStateTime;

    final Vector3 tempVector = new Vector3();

    public Renderer(int batchSize, Registry registry, ResourceManager manager){
        batch = new SpriteBatch(batchSize);

        int w = Gdx.graphics.getWidth();
        int h = Gdx.graphics.getHeight();

        camera = new OrthographicCamera(VIEW_DISTANCE, viewportHeight: VIEW_DISTANCE*h/w);
        batch.setProjectionMatrix(camera.combined);

        atlas = manager.getMainTextureAtlas();
        this.RY = registry;

        tileRegions = new TileRegions(registry, atlas);
        itemRegions = new ItemRegions(registry, atlas);

        playerAnimations = new PlayerAnimations(atlas);
        playerStateTime = 0f;

    }
```

Εικόνα 42 - Κώδικας αρχικοποίησης της κλάσης Renderer

Σημαντικό ρόλο στην λειτουργία της κλάσης έχουν τα αντικείμενα batch και camera των κλάσεων SpriteBatch και OrthographicCamera αντίστοιχα και παρέχονται από το libGDX.

Η κλάση SpriteBatch μας επιτρέπει να ομαδοποιούμε τις κλήσεις που κάνουμε στο υλικό υπεύθυνο για την απεικόνιση γραφικών μειώνοντας δραματικά των αριθμό κλήσεων που χρειάζονται για να ολοκληρωθεί ένα frame.

Στην συγκεκριμένη περίπτωση, χρειαζομαστε να απεικονίσουμε κατα περίπτωση ενδεχομένως και αριθμό στοιχείων άνω των 10.000-20.000.
Ένα τόσο μεγάλο φορτίο κλήσεων θα επιβαρύνει σημαντικά το υλικό και πιθανότατα θα καταστήσει το χρόνο που χρειάζεται για να ολοκληρωθεί ένα frame υπερβολικό, σε σημείο που το παιχνίδι να μην ανταποκρίνεται και η εμπειρία του χρήστη να είναι δυσάρεστη.

Αντί λοιπόν να κάνουμε 10.000-20.000 κλήσεις, μπορούμε χρησιμοποιώντας την κλάση SpriteBatch να τις ομαδοποίσουμε(στην περίπτωση μας σε ομάδες εώς 8000 στοιχεία) σε μονοψήφιο αριθμό.

Η κλάση OrthographicCamera μας παρέχει τις λειτουργίες που χρειάζεται η διαδικασία της απεικόνισης και μπορούμε να την φανταστούμε ως προς την χρήση της ως μια κάμερα που καταγράφει τον κόσμο μας.

Για παράδειγμα, μας δίνει την δυνατότητα να μετατρέψουμε συντεταγμένες οθόνης(σε pixel) σε συντεταγμένες κόσμου, καθώς και να κρατάμε τον οριζόντιο αριθμό κελιών που εμφανίζονται στην οθόνη σταθερό, ανεξάρτητα με την ανάλυση της.

Για κάθε frame υπάρχει μια επαναλαμβανόμενη διαδικασία:

```
public void render(World world){
    float delta =Gdx.graphics.getDeltaTime();

    camera.update();
    batch.setProjectionMatrix(camera.combined);

    for (IntMap.Entry<Chunk> chunk : world.chunks) {
        //Only render a chunk if it's visible
        if (camera.frustum.boundsInFrustum(chunk.value.center, chunk.value.dimensions)) {
            for (int x = 0; x < world.chunkWidth; x++) {
                for (int y = 0; y < world.chunkHeight; y++) {
                    Tile tile = chunk.value.getTile(x, y);
                    if (!RY.getT(tile.getId(FLOOR)).isRTE){
                        batch.draw(tileRegions.get(tile.id[FLOOR], tile.subId[FLOOR]), tile.position.x, tile.position.y,  width: 1,  height: 1);
                    } else {
                        world.tileEntities.get(world.systems.tileEntitySys.calcIndex(tile.position)).get(FLOOR).draw(batch);
                    }
                    if(!RY.getT(tile.getId(MID)).isRTE){
                        if (tileRegions.get(tile.id[MID], tile.subId[MID]) != null) {
                            batch.draw(tileRegions.get(tile.id[MID], tile.subId[MID]), tile.position.x, tile.position.y,  width: 1,  height: 1);
                        }
                    } else {
                        world.tileEntities.get(world.systems.tileEntitySys.calcIndex(tile.position)).get(MID).draw(batch);
                    }
                }
            }
        }
    }
```

Εικόνα 43 - Μέρος κώδικα της συναρτησης render(World world) της κλάσης Renderer

Αρχικά, ορίζεται το delta για το frame, ενημερώνεται η κάμερα, και ορίζεται η μήτρα προβολής του batch.

Σε πρώτο στάδιο κάθε chunk που είναι στην προσωρινή μνήμη ελέγχεται αν είναι ορατό από την κάμερα, και αν είναι τοτε προσθέτονται τα στοιχεία του στο τρέχων batch προς απεικόνιση με την χρήση της draw().
Για τα απλά στοιχεία κελιών απλά απεικονίζεται το αντίστοιχο γραφικό στοιχείο με βάση το id τους ενώ για τα πιο σύνθετα στοιχεία TileEntity(ή αλλίως RTE/Real-time TileEntity) καλείτε η δικιά τους μέθοδος απεικόνισης, καθώς η κατάστασή τους(και ενδεχομένως και η εμφάνιση τους) δεν είναι στατική.

```
    if (world.itemDrops != null) {
        for (ItemDrop itemDrop : world.itemDrops) {
            batch.draw(itemRegions.get(itemDrop.itemStack.id),
                    itemDrop.position.x, itemDrop.position.y,
                    width: .5f, height: .5f);
        }
    }

    //Accumulate state time for player animations
    playerStateTime += delta;
    //Render the player
    Vector2 playerPos = world.player.position;
    Vector2 offset = world.player.centerOffset;
    batch.draw(playerAnimations.getFrame(world.player.facingState,world.player.actionState,playerStateTime),
            x: playerPos.x + offset.x, y: playerPos.y + offset.y,
            width: 1, height: 1);
}
```

Εικόνα 44 - Μέρος κώδικα της συναρτησης render(World world) της κλάσης Renderer, συνέχεια της εικόνας 43.

Στην συνέχεια, αν υπάρχουν αντικείμενα στο έδαφος, προσθέτονται στο τρέχων batch.

Τέλος, απο άποψης κόσμου τουλάχιστον, υπολογίζεται η κατάσταση του παίκτη και προστίθεται στο batch το αντίστοιχο γραφικό στοιχείο.

Δεν εχει τελειωσει ομως όλη η διαδικασία απεικόνισης για το frame.

Μετά το τέλος της απεικόνισης του κόσμου, καλείτε η διεπαφή χρήστη να παραθέσει και να προσθέσει τα βασικά της στοιχεία στο batch.

Αφού ολοκληρωθεί και αυτή η διαδικασία καλείτε η συνάρτηση render αλλα με παράμετρο αυτή την φορά αντι για τον κόσμο την διεπαφή χρήστη, και θα προβεί στην απεικόνιση των εναπομειναντων στοιχείων της.

```
public void render(GameplayUi ui) {
    //Highlight the item the player is holding
    int srcFunc = batch.getBlendSrcFunc();
    int dstFunc = batch.getBlendDstFunc();
    batch.setBlendFunction(GL20.GL_DST_COLOR,GL20.GL_SRC_ALPHA);
    IndexedTable indexedTable = (IndexedTable) ui.quickInventory.mainTable.getChild(ui.quickInventory.selectedItem);
    ui.quickInventory.selectedItemDrawable.draw(batch,
            x: ui.quickInventory.mainTable.getX()+indexedTable.getX(),
            indexedTable.getY(),
            ui.quickInventory.cellSize,ui.quickInventory.cellSize);
    batch.setBlendFunction(srcFunc,dstFunc);

    //Render the inventory items
    if (ui.quickInventory.mainTable.isVisible()){
        renderInventoryItems(ui.quickInventory.mainTable, ui.quickInventory.mainTable);
    }

    if (ui.inventoryAndCrafting.inventoryTable.isVisible()) {
        renderInventoryItems(ui.inventoryAndCrafting.mainTable, ui.inventoryAndCrafting.inventoryTable);

    }

    if (ui.inventoryAndCrafting.craftingTable.isVisible()) {
        renderInventoryItems(ui.inventoryAndCrafting.mainTable, ui.inventoryAndCrafting.craftingTable);
    }

    if (ui.tileEntityUi.mainTable.isVisible()){
        renderInventoryItems(ui.tileEntityUi.mainTable, ui.tileEntityUi.mainTable);
    }
```

Εικόνα 45 - Μέρος κώδικα της συναρτησης render(GameplayUi ui) της κλάσης Renderer

Για την έμφαση που πρέπει οπτικά να δοθεί στην διεπαφή χρήστη στο αντικείμενο που κρατάει ο παίκτης, εργασία που ειναι πρωτη στην συνάρτηση, χρησιμοποιείται μια ξεχωριστή κλήση προς το υλικό, καθώς πρέπει να αλλάξουν οι μέθοδοι μίξης των χρωματων, έτσι ώστε να επιτευχθεί το επιθυμητό αποτέλεσμα.

Προχορόντας, για κάθε ορατό στοιχείο της διεπαφής χρήστη, απεικονίζονται τα γραφικά στοιχεία των αντικειμένων που κατέχει ο παίκτης, στο σωστό σημείο.

Την διαδικασία αυτή πραγματοποιεί η συνάρτηση renderInventoryItems().

Το μόνο που μένει να απεικονιστεί είναι το κείμενο που αντιστοιχεί στα άνω αντικείμενα, και υποδεικνύει τις ποσότητες τους.

Την διαδικασία αυτή πραγματοποιεί η συνάρτηση renderInventoryText().

Η σειρά με την οποία γίνεται η απεικόνιση των στοιχείων έχει σημασία, και θα αναλυθεί περεταίρω στο υποκεφάλαιο τεχνικών λεπτομερειών του κεφαλαίου αυτού.

Αυτό είναι το τέλος του frame, απο άποψη απεικόνισης, και όλη αυτή η διαδικασία θα επαναληφθεί κάθε frame που το πρόγραμμα βρίσκεται στην κανονική του ροή.

```
private void renderInventoryItems(Table topMostTable, Table mainTable){
    Inventory inventory = (Inventory) mainTable.getUserObject();
    //Go through all crafting inventory items
    for (Actor child : mainTable.getChildren()) {
        if (child instanceof IndexedTable){
            IndexedTable table = (IndexedTable) child;
            Actor actor = table.getChild( index: 0);
            //Accumulate relative coordinates,from the image to the main table
            float tX = 0;
            float tY = 0;
            while (actor != topMostTable.getParent()){
                tX += actor.getX();
                tY += actor.getY();
                actor = actor.getParent();
            }

            Image image = (Image) table.getChild( index: 0);
            //Find the relevant image by looking up the player's inventory
            ItemStack itemStack = inventory.get(table.getIndex());
            AtlasRegion atlasRegion = null;
            if(itemStack != null){
                ItemDefinition itemDef = RY.getI(itemStack.id);
                if (itemDef.is(Attributes.PLACEABLE)){
                    atlasRegion = tileRegions.get(((Placeable) itemDef.attributeValues.get(Attributes.PLACEABLE)).tileID);
                } else {
                    atlasRegion = itemRegions.get(itemStack.id);
                }
            }
            //Render the image in the accumulated position
            if (atlasRegion != null) {
                batch.draw(atlasRegion,tX,tY,image.getWidth(),image.getHeight());
            }
        }

    }
}
```

Εικόνα 46 - Κώδικας της συνάρτησης renderInventoryItems() της κλάσης Renderer

```
private void renderInventoryText(Table topMostTable, Table mainTable, float cellSizeMin, float cellSizeMax , float cellSize){
    for (Actor child : mainTable.getChildren()) {
        if (child instanceof IndexedTable) {
            Table table = (Table) child;
            Actor actor = table.getChild( index: 1);
            //Accumulate relative coordinates,from the label to the main table
            float tX = 0;
            float tY = 0;
            while (actor != topMostTable.getParent()) {
                tX += actor.getX();
                tY += actor.getY();
                actor = actor.getParent();
            }

            Label label = (Label) table.getChild( index: 1);
            //Render the label in the accumulated position using its font
            label.getStyle().font.getData().setScale(MathUtils.map(cellSizeMin, cellSizeMax,  outRangeStart: 0.25f,  outRangeEnd: .75f, cellSize));
            label.getStyle().font.draw(batch, label.getText().toString(),  x: tX + label.getWidth() / 2,  y: tY + label.getHeight() / 3);
        }
    }
}
```

Εικόνα 47 - Κώδικας της συνάρτησης renderInventoryText() της κλάσης Renderer

Οι βοηθητικές κλάσεις ItemRegions και TileRegions με βάση το σύστημα Registry τοποθετούν τα αντίστοιχα γραφικά στοιχεία σε δομές δεδομένων τύπου IntMap, έτσι ώστε να μπορεί να τις προσπελάσει οταν αυτό χρειαστεί η κλάση Renderer.

```
public class ItemRegions {
    IntMap<AtlasRegion> regions;

    public ItemRegions(Registry registry, TextureAtlas atlas) {
        regions = new IntMap<>();
        registerAtlasRegions(registry,atlas);
    }


    public void registerAtlasRegions(Registry registry, TextureAtlas atlas){
        for (ItemDefinition item : registry.getAllItems()) {
            AtlasRegion region = atlas.findRegion(item.name);
            if(region != null){
                regions.put(item.id,region);
            }
        }
    }

    public AtlasRegion get(int id) { return regions.get(id); }

}
```

Εικόνα 48 - Κώδικας της κλάσης ItemRegions

```
public class TileRegions {

    IntMap<IntMap<TextureAtlas.AtlasRegion>> regions;

    public TileRegions(Registry registry, TextureAtlas atlas) {
        regions = new IntMap<>( initialCapacity: 16);
        registerAtlasRegions(registry,atlas);
    }


    public void registerAtlasRegions(Registry registry, TextureAtlas atlas){
        for (TileDefinition tileDefinition : registry.getAllTiles()) {
            IntMap<TextureAtlas.AtlasRegion> map;

            if(!tileDefinition.isRTE){
                if(tileDefinition.hasSubId){
                    map = new IntMap<>(tileDefinition.subIdCount);
                    for (TextureAtlas.AtlasRegion region : atlas.findRegions(tileDefinition.name)) {
                        map.put(region.index,region);
                    }
                } else {
                    map = new IntMap<>( initialCapacity: 1);
                    map.put(0,atlas.findRegion(tileDefinition.name));
                }

            } else {
                map = new IntMap<>( initialCapacity: 1);
                map.put(0, atlas.findRegion(tileDefinition.defaultRegion));
            }
            regions.put(tileDefinition.id,map);
        }
    }

    public TextureAtlas.AtlasRegion get(int id) { return get(id, subId: 0); }

    public TextureAtlas.AtlasRegion get(int id,int subId) { return regions.get(id).get(subId); }

}
```

Εικόνα 49 - Κώδικας της κλάσης TileRegions

Η κλάση PlayerAnimations έχει παρόμοια λειτουργία, με την πρόσθετη δυνατότητα να μπορεί δεδομένης της κατάστασης του παίκτη να υπολογίσει το σωστό γραφικό στοιχείο.

```
public class PlayerAnimations {
    protected Animation<AtlasRegion> walkingSouth;
    protected Animation<AtlasRegion> walkingNorth;
    protected Animation<AtlasRegion> walkingWest;
    protected Animation<AtlasRegion> walkingEast;

    PlayerAnimations(TextureAtlas atlas){
        walkingNorth = new Animation<>( frameDuration: 0.3f, atlas.findRegions( name: "walking_north"));
        walkingEast  = new Animation<>( frameDuration: 0.3f, atlas.findRegions( name: "walking_east"));
        walkingSouth = new Animation<>( frameDuration: 0.3f, atlas.findRegions( name: "walking_south"));
        walkingWest  = new Animation<>( frameDuration: 0.3f, atlas.findRegions( name: "walking_west"));
    }

    AtlasRegion getFrame(int facingState ,int actionState ,float stateTime){
        switch (facingState){
            case States.FACING_NORTH:
                if(actionState==States.IDLE){
                    return walkingNorth.getKeyFrames()[1];
                } else {
                    return walkingNorth.getKeyFrame(stateTime, looping: true);
                }
            case States.FACING_EAST:
                if(actionState==States.IDLE){
                    return walkingEast.getKeyFrames()[1];
                } else {
                    return walkingEast.getKeyFrame(stateTime, looping: true);
                }
            case States.FACING_WEST:
                if(actionState==States.IDLE){
                    return walkingWest.getKeyFrames()[1];
                } else {
                    return walkingWest.getKeyFrame(stateTime, looping: true);
                }
            case States.FACING_SOUTH:
            default:
                if(actionState==States.IDLE){
                    return walkingSouth.getKeyFrames()[1];
                } else {
                    return walkingSouth.getKeyFrame(stateTime, looping: true);
                }
        }
    }
}
```

Εικόνα 50 - Κώδικας της κλάσης PlayerAnimations

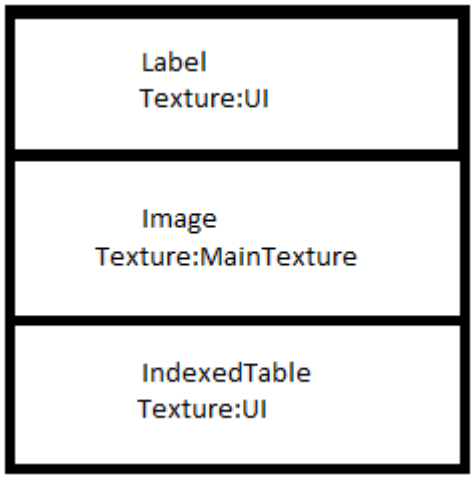
4.7.3 Σύστημα Rendering : Τεχνικές λεπτομέρειες

Όπως προαναφέρθηκε, η σειρά με την οποία θα προστεθούν τα στοιχεία στο batch και θα απεικονιστούν έχει σημασία.
Στην περίπτωση μας, δύο είναι οι λόγοι:

Η σειρά με την οποία θα προστεθούν τα στοιχεία είναι η σειρά με την οποία θα απεικονιστούν, οπότε η στρατηγική μας είναι να προσθέσουμε πρώτα τα στοιχεία του εδάφους το κόσμου, μετα τα στοιχεία που βρίσκονται πάνω στο έδαφος, μετα τον παίκτη, και τέλος τα στοιχεία διεπαφής χρήστη, ετσι ωστε να μην υπάρχει ανεπιθύμητη επικάλυψη των γραφικών μας στοιχείων.

Ο δεύτερος λόγος είναι για την αποφυγή αλλαγής texture, κάθε μία προκαλεί μια έξτρα κλήση προς το υλικό.
Στην περίπτωση μας έχουμε δύο texture, ενα για την διεπαφή χρήστη, έστω UI, και ένα για τα γραφικά του κόσμου(κελιά, αντικείμενα), έστω MainTexture.

Το πρόβλημα που παρουσιάστηκε σχετικά με την δεύτερη περίπτωση είναι στην διεπαφή χρήστη, όπου κάθε στοιχείο που αντιπροσωπεύει αντικείμενο έχει την εξής μορφή(που ορίσαμε εμείς).



Εικόνα 51 - Μορφή στοιχείου διεπαφής χρήστη που απεικονίζει αντικείμενο

Αν δεν κάναμε καμία αλλαγή, κάθε αντικείμενο θα προκαλούσε δύο έξτρα κλήσεις προς το υλικό, γεγονός μη επιθυμητό.

Για αυτό το λόγο δημιουργήθηκαν οι συναρτήσεις renderInventoryItems() και renderInventoryText(), και οι αντίστοιχες κλήσεις τους κατα την συναρτηση render(GameplayUi ui), έτσι ώστε να απεικονίζονται πρώτα όλα τα αντικείμενα και μετά από πάνω όλα τα στοιχεία κειμένου, με αποτέλεσμα να χρειάζονται δύο αλλαγές texture στο σύνολο για την διαδικασία, αντι για δύο αλλαγές ανα αντικείμενο.

4.8 Σύστημα διεπαφής χρήστη(UI) - Κύριο Σύστημα

4.8.1 Σύστημα διεπαφής χρήστη : Επισκόπηση λειτουργίας και στόχων

Το σύστημα διεπαφής χρήστη ορίζει την εμφάνιση και την λειτουργία των γραφικών στοιχείων με τα οποία αλληλεπιδρά ο παίκτης, που δεν ανήκουν στον κόσμο, στην οθόνη κεντρικού μενού και στην οθόνη κόσμου.

4.8.2 Σύστημα διεπαφής χρήστη : Παρουσίαση και επεξήγηση κώδικα

Το σύστημα διεπαφής χρήστη βασίζεται στο λογισμικό Scene2d και Scene2d.ui που παρέχεται απο το libGDX και μας δίνει την δυνατότητα να αφοσιωθούμε στην μορφή και τις λειτουργίες που θέλουμε να έχει η διεπαφή χρήστη μας, καθώς παρέχει ένα μεγάλο μέρος της λειτουργικότητας που είναι συχνα κοινή ανάμεσα στις διάφορες διεπαφές χρήστη.

Κάθε μία από τις δύο οθόνες του παιχνιδιού έχει δική της υλοποίηση της διεπαφής χρήστη, ανάλογα με τις ανάγκες κάθε φορά.
Όμως, και οι δύο κληρονομούν απο το interface BaseUi για να οριστούν οι βασικές τους λειτουργίες:

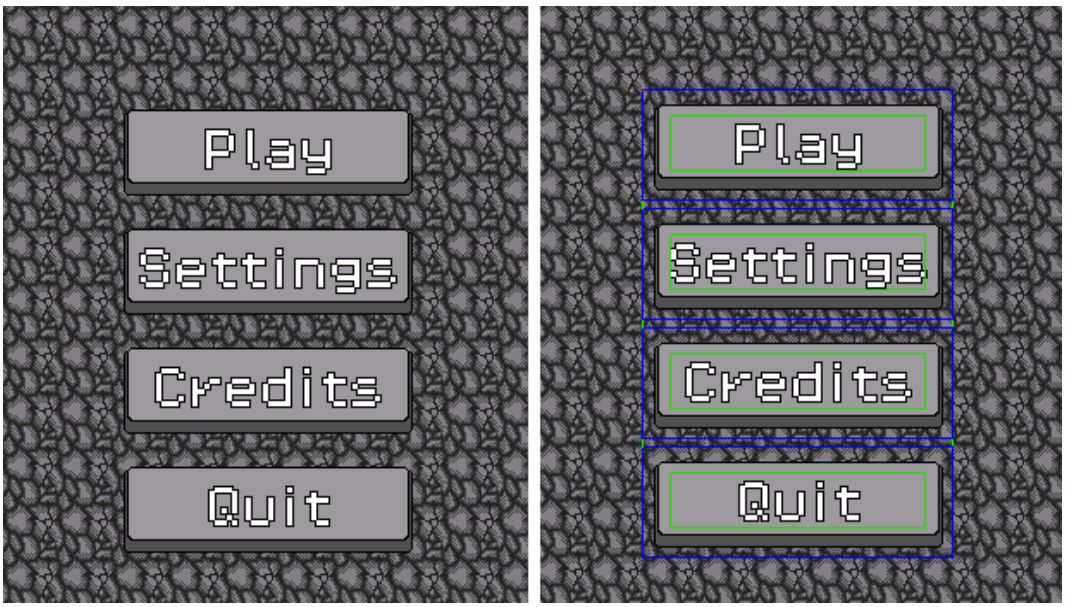
```
public interface BaseUi {
    void update(float delta);
    void draw();
    void resize(int width, int height);
    Stage getStage();
}
```

Εικόνα 52 - Κώδικας του interface BaseUi

Η οργάνωση της διεπαφής ορίζεται μέσω του ορισμού ενός συνόλου από αντικείμενα Actor που έχουν πληροφορίες όπως η θέση τους και το μέγεθός τους, αντικείμενα Group που μπορούν να έχουν ως παιδιά τους άλλα αντικείμενα Actor και ως κεντρικό σημείο ένα αντικείμενο Stage, που αποτελεί το αρχικό αντικείμενο.

Με την προσθήκη Actor και Group στο Stage δημιουργείται ένας γράφος, τον οποίο και επεξεργάζεται το Stage για να τα ενημερώνει και να τα απεικονίζει.

Στην συνέχεια θα αναδειχθούν μερικά παραδείγματα από τα στοιχεία της διεπαφής που υλοποιήθηκε:



Εικόνα 53 - Εμφάνιση της οθόνης κεντρικού μενού, χωρίς (αριστερά) και μέ (δεξιά) τις γραμμές που υποδεικνύουν την θέση και το μέγεθος κάθε Actor

```
public class MainMenuUi implements BaseUi,Disposable {

    final Stage stage;
    final Skin skin;
    final TextureAtlas uiAtlas;

    public final Table mainTable;
    public final VerticalGroup buttonGroup;

    public Table playTable;
    public Table settingsTable;
    public Table creditsTable;

    public TextButton backButton;
    public boolean backTouched;

    public boolean worldSelected = false;

    public boolean playTouched = false;
    public boolean playCreated = false;
    public boolean settingsTouched = false;
    public boolean settingsCreated = false;
    public boolean creditsTouched = false;
    public boolean creditsCreated = false;
    public boolean quitTouched = false;

    public String playPath;

    public MainMenuUi(final Batch batch, TextureAtlas uiAtlas) {
        stage = new Stage(new ScreenViewport(),batch);
        stage.setDebugAll(true);

        skin = new Skin(Gdx.files.internal( path: "ui.json"),uiAtlas);
        this.uiAtlas = uiAtlas;

        mainTable = new Table(skin);
        mainTable.setFillParent(false);
        mainTable.setX(stage.getWidth()/2);
        mainTable.setY(stage.getHeight()/2);
        stage.addActor(mainTable);
```

Εικόνα 54 - Κώδικας αρχικοποίησης της κλάσης MainMenuUi

Για την διεπαφή κεντρικού μενού αρχικοποιείται το Stage και φορτωνονται στο Skin οι διάφοροι ορισμοί γραφικών στοιχείων, τα οποία παρέχονται στο TextureAtlas. Στο Skin έχουμε εκτός του προγράμματος ορίσει μέσω του εργαλείου Skin Composer τις επιθυμητές προκαθορισμένες εμφανίσεις κάθε τύπου αντικειμένου Actor που μας ενδιαφέρει και το φορτώνουμε απο το αρχείο ui.json.Το αρχείο αυτο, όπως και τα Texture μας προστίθενται εσωτερικά στο τελικό jar και είναι ετσι διαθέσιμα.

Η διεπαφή κεντρικού μενού αποτελείται από ένα κεντρικό Table(mainTable) στο οποίο προστίθεται ένα σύνολο 4 κουμπιών(buttonGroup).
Κάθε ένα από τα τρία πρώτα αυτα κουμπιά έχει το δικό του αντίστοιχο Table που περιέχει με την σειρα του άλλα στοιχεία.
Το πρώτο κουμπί ορίζεται ως εξής:

```
final TextButton playButton = new TextButton( text: "Play",skin);
playButton.addListener(new ChangeListener() {
    @Override
    public void changed(ChangeEvent event, Actor actor) {
        if (playCreated){
            playTouched = true;
            mainTable.removeActor(buttonGroup);
            mainTable.addActor(playTable);
        } else {
            playTouched = true;
            playCreated = true;
            buttonGroup.setVisible(false);
            mainTable.removeActor(buttonGroup);

            final String savePath = "./Saves";

            if(!Gdx.files.local(savePath).exists()){
                Gdx.files.local(savePath).mkdirs();
            }

            playTable = new Table(skin);

            final List<String> list = new List<>(skin);
            list.setItems("World 1","World 2","World 3");
            list.addListener(new ClickListener(){
                @Override
                public void clicked(InputEvent event, float x, float y) {
                    worldSelected = true;
                    playPath = savePath +"/"+ list.getItemAt(y);
                }
            });

            playTable.add(list);
            mainTable.add(playTable);
        }
    }
});
```

Εικόνα 55 - Κώδικας απο τον constructor της κλάσης MainMenuUi, συνέχεια της εικόνας 54

Μέσω της προσθήκης ενος ChangeListener στο κουμπί, μας δίνεται η δυνατότητα να εκτελέσουμε κώδικα που επιθυμούμε οταν το κουμπί πατηθεί.
Το playButton όταν πατηθεί θα βγάλει το γκρουπ απο κουμπιά στο οποίο ανήκει απο το mainTable και θα προσθέσει ένα καινούργιο μενού, όπως ορίζεται στην εικόνα 55.

Υπάρχουν πολλά κομμάτια κώδικα που ορίζουν διεπαφή χρήστη, ωστόσο όμως είναι της ίδιας λογικής με τα παραπάνω οπότε θα παρατεθούν μόνο οι εξαιρέσεις και τα σημαντικά σημεία.

Ενα απο αυτα είναι η κλάση IndexedTable:

```
public class IndexedTable extends Table {
    int index;

    public IndexedTable(Skin skin, Dispatcher dispatcher){
        super(skin);

        final Image image = new Image();
        image.setVisible(false);
        this.addActor(image);

        final Label label = new Label( text: null,skin);
        label.setVisible(false);
        this.addActor(label);

        this.addListener(new InventoryListener( table: this,image,label,dispatcher));
    }

    public int getIndex() { return index; }

    public void setIndex(int index) { this.index = index; }
}
```

Εικόνα 56 - Κώδικας της κλάσης IndexedTable

Τα αντικείμενα της κλάσης IndexedTable χρησιμοποιούνται στην διεπαφή της οθόνης κόσμου, και χρησιμεύουν για την αναπαράσταση αντικειμένων που κατέχει ο παίκτης αλλα και λειτουργικά για την οργάνωση τους απο τον παίκτη μέσω της διεπαφής.

Την λειτουργικότητα αναλαμβάνει η κλάση InventoryListener :

```
@Override
public void drag(InputEvent event, float x, float y, int pointer) {
    super.drag(event, x, y, pointer);
    image.setPosition( x: x-image.getWidth()/2, y: y-image.getHeight()/2);
    label.setPosition( x: x-label.getWidth()/2, y: y-label.getHeight()/2);
}
```

Εικόνα 57 - Κώδικας της συνάρτησης drag της κλάσης InventoryListener

```java
@Override
public void dragStop(InputEvent event, float x, float y, int pointer) {
    super.dragStop(event, x, y, pointer);
    image.setPosition(image.getOriginX(), image.getOriginY());
    label.setPosition(label.getOriginX(), label.getOriginY());
    Stage stage = table.getStage();
    Actor actor = stage.hit(Gdx.input.getX(pointer), stageY: stage.getHeight() - Gdx.input.getY(pointer), touchable: true);
    if(actor instanceof IndexedTable){
        IndexedTable targetTable = (IndexedTable) actor;
        Table targetMainTable = (Table)targetTable.getParent();
        Table sourceMainTable = (Table)table.getParent();
        if(targetMainTable.getChild(targetTable.getIndex())!= null){
            Inventory sourceInventory = (Inventory) sourceMainTable.getUserObject();
            Inventory targetInventory = (Inventory) targetMainTable.getUserObject();

            if (sourceInventory == null || targetInventory == null){
                return;
            }

            ItemStack sourceStack = sourceInventory.get(table.getIndex());
            ItemStack targetStack = targetInventory.get(targetTable.getIndex());

            InventoryPayload payload = pool.obtain();
            payload.sourceInventory = sourceInventory;
            payload.targetInventory = targetInventory;

            if(sourceStack != null){
                payload.operation = InventoryPayload.Operations.SWAP;
                payload.targetIndex = targetTable.getIndex();
                payload.sourceIndex = table.getIndex();
                dispatcher.dispatch(Events.INVENTORY,payload);
            } else {
                payload.free();
            }
        }
}
}
```

Εικόνα 58 - Κώδικας της συνάρτησης dragStop της κλάσης InventoryListener

Μέσω του κώδικα αυτου επιτρέπεται η σύνταξη και η αποστολή Event τυπου Inventory που με την σειρά τους θα διεκπεραιωθούν απο τον EventHandler του κόσμου.

Μια άλλη εξαίρεση είναι η κλάση TileEntityUi που εκτός της ύπαρξης ενός κεντρικού Table δεν ορίζει η ίδια την μορφή της.
Η κλάση αυτή αντιπροσωπεύει μέρος της διεπαφής χρήστη κόσμου και ορίζεται κατα περίπτωση απο τα TileEntities που αλληλεπιδρά ο χρήστης μέσα στον κόσμο.

Για αυτό τον λόγο, ο ορισμός της μορφής της γίνεται από αντικείμενα τύπου UISpec:

```java
public class UISpec {

    public void setup(Table mainTable, Skin skin, Dispatcher dispatcher){

    }


}
```

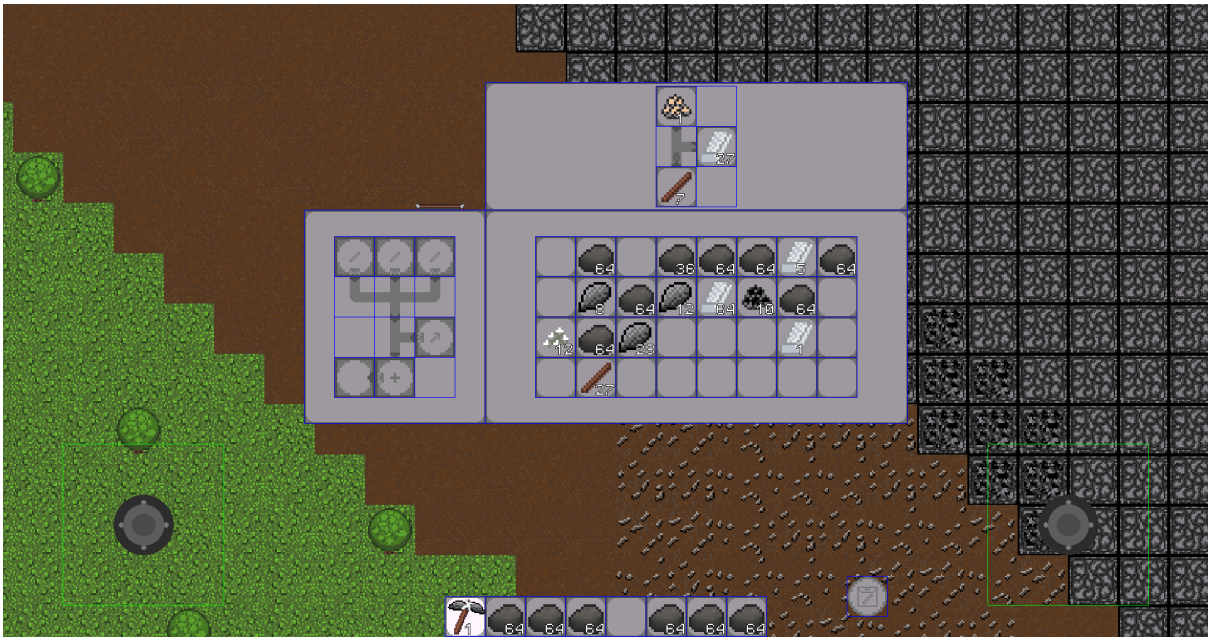
Εικόνα 59 - Κώδικας της κλάσης UISpec

Όμως η υλοποίηση του κώδικα δεν βρίσκεται ούτε εκεί, καθώς ορίζεται απευθείας, απο κάθε τύπο TileEntity, όπως για παράδειγμα το TileEntity Basket:

```
public static final UISpec basketUiSpec;

static {
    basketUiSpec = new UISpec(){
        @Override
        public void setup(Table mainTable, Skin skin, Dispatcher dispatcher) {

            for (int i = 0; i < INVENTORY_SIZE; i++) {
                IndexedTable indexedTable = new IndexedTable(skin,dispatcher);
                indexedTable.setBackground("item_background");
                indexedTable.setIndex(i);
                indexedTable.setTouchable(Touchable.enabled);
                mainTable.add(indexedTable);

                if (i == INVENTORY_SIZE/2 -1){
                    mainTable.row();
                }
            }

        }
    };
}
```

Εικόνα 60 - Κώδικας του ορισμού του UISpec της κλάσης Basket

Συνολικά, η διεπαφή χρήστη κόσμου αποτελείται από πολλές κλάσεις που ορίζουν διαφορετικά στοιχεία και έχει την εξής εμφάνιση:



Εικόνα 61 - Πλήρης απεικόνιση της κεντρικής διεπαφής χρήστη της οθόνης κόσμου

4.9 Σύστημα εισόδου - Κύριο Σύστημα

4.9.1 Σύστημα εισόδου : Επισκόπηση λειτουργίας και στόχων

Το σύστημα εισόδου είναι υπεύθυνο να λάβει, να επεξεργαστεί και ενδεχομένως να αποστείλει Events μέσω του συστήματος Event με βάση την είσοδο που λαμβάνει από τον χρήστη.

Η είσοδος μπορεί να προέρχεται είτε από την οθόνη αφής, είτε από το ποντίκι, είτε από το πληκτρολόγιο, ανάλογα την πλατφόρμα.

Προγραμματιστικά το σύστημα βασίζεται στην κλάση InputProcessor που παρέχεται από το libGDX και επιτρέπει σε εμάς να διαχειριστούμε τα event που λαμβάνει το παιχνίδι από τα αντίστοιχα μέσα εισόδου.

4.9.2 Σύστημα εισόδου : Παρουσίαση και επεξήγηση κώδικα

Η κεντρική κλάση του συστήματος είναι η InputHandler:

```
public class InputHandler implements EventCreator {

    Stage stage;
    UiProcessor uiProcessor;
    PlayerProcessor playerProcessor;
    WorldProcessor worldProcessor;
    InputMultiplexer multiplexer;
    Dispatcher dispatcher;

    public InputHandler(Dispatcher dispatcher, GameplayUi ui, World world){
        this.dispatcher = dispatcher;
        this.multiplexer = new InputMultiplexer();
        this.stage = ui.getStage();
        this.uiProcessor = new UiProcessor(dispatcher,world);
        playerProcessor = new PlayerProcessor(dispatcher,world,ui.touchInput.movementTouchpad,ui.touchInput.interactionTouchpad);
        worldProcessor = new WorldProcessor(dispatcher,world);

        multiplexer.addProcessor(stage);
        multiplexer.addProcessor(uiProcessor);
        multiplexer.addProcessor(playerProcessor);
        multiplexer.addProcessor(worldProcessor);
        Gdx.input.setInputProcessor(multiplexer);
    }

    public void update(float delta) { playerProcessor.update(); }

}
```

Εικόνα 62 - Κώδικας της κλάσης InputHandler

Η κλάση μας αυτή ορίζει ένα σύνολο από 3 InputProcessors, χωρισμένους σε διαφορετικές κλάσεις ανάλογα με το περιεχόμενο τον εισόδων που επεξεργάζονται.

Ο InputProcessor που παρέχεται από το Stage που είδαμε στο προηγούμενο κεφάλαιο επεξεργαζεται τις εισόδους που λαμβάνουν οι Actor του, και μαζί με τους άλλους 3 προστίθενται στον InputMultiplexer.

Η σειρά με την οποία προστέθηκαν στον InputMultiplexer(εφόσον δεν οριστεί κάποιος δείκτης) είναι και η σειρά με την οποία θα λάβουν τα event εισόδου οι InputProcessors.

Αν ένας InputProcessor επεξεργαστεί ένα event και επιθυμούμε να μήν το λάβουν οι υπόλοιποι μπορεί να επιστρέψει true στο αντίστοιχο event.

Για παράδειγμα, αν ο χρήστης προσπαθεί να αλληλεπιδράσει με την διεπαφή χρήστη, δεν θέλουμε ο κόσμος να λάβει τα αντίστοιχα events, καθώς δεν προορίζονται για αυτον.

Η κλάση InputProcessor μας παρέχει διάφορες συναρτήσεις που μπορούμε να κάνουμε override, όπως η keyDown(), που καλείτε κάθε φορά που πιέζεται ένα πλήκτρο.

```
@Override
public boolean keyDown(int keycode) {
    UiPayload payload = pool.obtain();
    switch (keycode){
        case Input.Keys.E:
            payload.element = UiElements.INVENTORY_AND_CRAFTING;
            payload.operation = Operations.TOGGLE;
            dispatcher.dispatch(Events.UI,payload);
            break;
        case Input.Keys.C:
            payload.element = UiElements.CRAFTING;
            payload.operation = Operations.TOGGLE;
            dispatcher.dispatch(Events.UI,payload);
            break;
        case Input.Keys.I:
            payload.element = UiElements.INVENTORY;
            payload.operation = Operations.TOGGLE;
            dispatcher.dispatch(Events.UI,payload);
            break;
        default:
            return false;
    }
    return true;
}
```

Εικόνα 63 - Κώδικας της συνάρτησης keyDown() της κλάσης UiProcessor

Με παράμετρο το κωδικό του πλήκτρου που πατήθηκε μπορούμε να χειριστούμε την είσοδο όπως επιθυμούμε, σε αυτήν την περίπτωση στέλνωντας Event προς την διεπαφή χρήστη, για να αλλάξει την ορατότητα διαφόρων στοιχείων της.

Μια άλλη συνάρτηση είναι η touchDown(), που καλείται όταν λαμβάνει είσοδο η οθόνη, άσχετα με το μέσο.
Έτσι μπορούμε να χειριστούμε την αλληλεπίδραση του χρήστη με τα κελιά του κόσμου, ως εξής:

```
@Override
public boolean touchDown(int screenX, int screenY, int pointer, int button) {
    WorldInteractionPayload payload = pool.obtain();
    if (button == Input.Buttons.LEFT) {
        Vector3 vec = world.renderer.unproject(screenX,screenY);
        Tile tile = world.getTileByPosition(vec.x, vec.y);

        ItemStack itemStack = world.player.quickInventory.get(world.player.selectedItem);
        ItemDefinition itemDef = null;
        if (itemStack != null){
            itemDef = world.registry.getI(itemStack.id);
        }

        if (tile != null && itemDef != null && itemDef.is(ItemDefinition.Attributes.PLACEABLE) && tile.getId(Tile.MID) == TileDefinition.Tiles.AIR){
            payload.interaction = Interactions.PLACE;
        } else {
            payload.interaction = Interactions.BREAK;
        }
    } else {
        payload.interaction = Interactions.INFO;
    }
    payload.source = Sources.PLAYER;
    payload.target = Targets.TILE;
    payload.inputMeans = InputMeans.MOUSE;
    payload.screenX = screenX;
    payload.screenY = screenY;
    dispatcher.dispatch(Events.WORLD_INTERACTION,payload);
    return true;
}
```

Εικόνα 64 - Κώδικας της συνάρτησης touchDown() της κλάσης WorldProcessor

4.9.3 Σύστημα εισόδου : Τεχνικές λεπτομέρειες

Το libGDX μας παρέχει μια μεγάλη διευκόλυνση στο χειρισμό των εισόδων προς την οθόνη, είτε το μέσο είναι μια οθόνη αφής, είτε το μέσο είναι ένα ποντίκι, δεν έχει κάποια διαφορά για εμάς, καθώς λαμβάνουμε συντεταγμένες οθόνης(screenX,ScreenY) σε pixel με βάση κάθε είσοδο.

Αν για κάποιο λόγο θέλουμε κάποια παραπάνω πληροφορία σχετικά με την είσοδο ανάλογα με την πλατφόρμα μας παρέχονται δύο ακόμα μεταβλητες.

Για τις πλατφόρμες που υποστηρίζουν MultiTouch μας παρέχεται στην μεταβλητή pointer ο δείκτης που αριθμεί το αντίστοιχο δάχτυλο. Για παράδειγμα, αν ένας χρήστης πατήσει με την σειρά 3 δάχτυλα και δεν τα αφαιρέσει πρίν ακουμπήσουν και τα 3, θα δημιουργηθούν 3 είσοδοι, με τιμές pointer 0 - 1 - 2.

Απο την άλλη πλευρά, όταν η είσοδος γίνεται από ένα ποντίκι μπορεί να μας ενδιαφέρει πιο κουμπί πατηθηκε(Π.χ Αριστερό/Δεξί), το οποίο παρέχεται στην μεταβλητή button. Στην περίπτωση της αφής, η μεταβλητή αυτή έχει την τιμή του αριστερού click.

4.10 Σύστημα οντοτήτων (Entity) - Κύριο Σύστημα

4.10.1 Σύστημα οντοτήτων : Επισκόπηση λειτουργίας και στόχων

Το σύστημα οντοτήτων ορίζει την ύπαρξη και τις λειτουργίες στοιχείων του κόσμου που είναι πιο περίπλοκα ως προς τις λειτουργίες τους και δεν αρκούνται σε ένα γενικό ορισμό όπως τα κελιά ή τα αντικείμενα, καθώς κάθε ύπαρξη τους είναι μοναδική.

Μία από αυτές τις οντότητες είναι και ο χαρακτήρας που χειρίζεται ο παίκτης

4.10.2 Σύστημα οντοτήτων : Παρουσίαση και επεξήγηση κώδικα

Η βασική κλάση για τις οντότητες είναι η κλάση Entity:

```
public abstract class Entity {
    public static class Types{
        public static final int ENTITY = 0;
        public static final int ITEM_DROP = 1;
        public static final int TILE_ENTITY = 2;
    }
    public int type;
    public Vector2 position;

}
```

Εικόνα 65 - Κώδικας της κλάσης Entity

Τα ορίσματα της είναι απλά, μια μεταβλητή για τον τύπο που είναι, και μία μεταβλητή για την θέση της οντότητα στον κόσμο.
Με αυτή την βάση όμως μπορούμε να χτίσουμε πιο πολύπλοκες οντότητες, όπως αυτή του παίκτη:

```
public class Player extends Entity implements MovementReceiver, ItemSelectionReceiver {

    public static class States{
        public static final int IDLE = 0;
        public static final int WALKING = 1;
        public static final int RUNNING = 2;

        public static final int FACING_NORTH = 6;
        public static final int FACING_EAST = 7;
        public static final int FACING_SOUTH = 8;
        public static final int FACING_WEST = 9;
    }

    public static class InventorySize {
        public static final int QUICK_INVENTORY_SIZE = 8;
        public static final int MAIN_INVENTORY_SIZE = 32;
        public static final int CRAFTING_INVENTORY_SIZE = 5;
    }
```

Εικόνα 66 - Κώδικας ορισμών σταθερών της κλάσης Player

```java
public Vector2 centerOffset = new Vector2( x: -0.5f, y: -0.5f);
public Vector2 bottomOffset = new Vector2( x: 0, y: -0.5f);
public Vector2 waistOffset = new Vector2( x: 0,  y: -0.25f);
public boolean gameWon;
public float speed = 3;
public float runningSpeed = 7;
public int reach = 3;
public int pickupRange = 2;

public int selectedItem;

public Vector2 spawnPoint;
public Vector2 direction;
public Vector2 temp = new Vector2();

public Body body;

/*
    States
 */
public int facingState = FACING_SOUTH;
public int actionState = IDLE;

public Inventory mainInventory;
public Inventory quickInventory;
public CraftingInventory craftingInventory;
```

Εικόνα 67 - Κώδικας ορισμών μεταβλητών της κλάσης Player

```java
public Player(String path){
    this.type = Types.ENTITY;
    this.position = new Vector2();
    this.spawnPoint = new Vector2();
    this.playerLoader = new PlayerLoader(path, player: this);

    this.mainInventory = new Inventory(MAIN_INVENTORY_SIZE);
    this.quickInventory = new Inventory(QUICK_INVENTORY_SIZE);

    SlotSpec slotSpec = new SlotSpec();
    slotSpec.set(SlotTypes.FIRST_COMPONENT, 0);
    slotSpec.set(SlotTypes.SECOND_COMPONENT, 1);
    slotSpec.set(SlotTypes.THIRD_COMPONENT, 2);
    slotSpec.set(SlotTypes.TOOL, 3);
    slotSpec.set(SlotTypes.OUTPUT, 4);
    this.craftingInventory = new CraftingInventory(CRAFTING_INVENTORY_SIZE,slotSpec);


    playerLoader.load();
}
```

Εικόνα 68 - Κώδικας του constructor της κλάσης Player

Όπως φαίνεται και στις εικόνες 65-67 η οντότητα που χειρίζεται ο παίκτης βασίζεται στην κλάση Entity και προσθέτει μια πληθωρα σταθερών και μεταβλητών που ορίζουν την κατάσταση της, για παράδειγμα αν τρέχει ή οχι, και τις ιδιότητες της, όπως την (μέγιστη) ταχύτητα της.

Ένας άλλος τύπος οντότητας είναι τα ItemDrops, τα οποία όπως υποδεικνύει το όνομα αναπαριστούν ένα αντικείμενο που βρίσκεται στο έδαφος(και μπορεί να μαζέψει ο παίκτης).

```
public class ItemDrop extends Entity {
    public ItemStack itemStack;

    public ItemDrop(float x, float y,int itemId,byte amount){
        this.type = Types.ITEM_DROP;
        this.position = new Vector2(x,y);
        this.itemStack = new ItemStack(itemId,amount);
    }
}
```

Εικόνα 69 - Κώδικας της κλάσης ItemDrop

Οι οντότητες αυτές επιπρόσθετα κατέχουν πληροφορία για το τί είδους αντικειμένου αναπαριστούν, και την ποσότητα του.

Ο τρίτος τύπος οντοτήτων είναι τα TileEntities. Τα TileEntities διαφοροποιούνται ως προς τα άλλα Entities διοτι ανήκουν σε κάποιο κελί του κόσμου, και αναπαριστώνται με το id τους στο αρχείο κελιών του κόσμου.

```
public abstract class TileEntity extends Entity {


    public static TileEntity newTE(World world, float x, float y, int layer, int id){...}

    public boolean hasUiSpec;
    public UISpec uiSpec;
    public boolean hasInventory;
    public Inventory inventory;

    public final World world;

    public int chunkIndex;
    public int tileIndex;
    public int layer;

    public final int id;
    public TileEntity(World world, float x, float y, int layer, int id){
        this.type = Types.TILE_ENTITY;
        this.world = world;
        this.position = new Vector2();
        this.position.set(x,y);
        this.chunkIndex = world.getChunkByPosition(x,y).index;
        this.tileIndex = world.getTileByPosition(x,y).index;
        this.layer = layer;
        this.id = id;
    }

    public void update(float delta) {

    }

    public void draw(SpriteBatch batch){

    }
}
```

Εικόνα 70 - Κώδικας της κλάσης TileEntity

Η κλάση TileEntity αποτελεί την βάση για την δημιουργία κελιών με λειτουργίες και όχι μόνο ιδιότητες ή λειτουργίες που υλοποιούνται από τρίτα συστήματα.

Ένα από αυτά είναι ένας φούρνος μεταλλευμάτων:

```
public class MetalFurnace extends TileEntity{

    public static final int INVENTORY_SIZE = 3;
    public static final int FUEL_SLOT = 2;
    public static final float MAX_FUEL = 500;
    public static final float CONSUMPTION_INTERVAL = 1f;
    public static final float CONSUMPTION_AMOUNT = 1f;

    public static final UISpec metalFurnaceUiSpec;

    static {...}

    final AtlasRegion regionActive;
    final AtlasRegion regionInactive;

    final CraftingInventory craftingInventory;

    boolean hasFuel;

    float fuelCounter;
    float intervalCounter;

    public MetalFurnace(World world, float x, float y, int layer) {
        super(world, x, y, layer, Tiles.METAL_FURNACE);
        this.regionActive = world.renderer.atlas.findRegion( name: "METAL_FURNACE_ACTIVE");
        this.regionInactive = world.renderer.atlas.findRegion( name: "METAL_FURNACE_INACTIVE");
        this.hasUiSpec = true;
        this.uiSpec = metalFurnaceUiSpec;
        this.hasInventory = true;
        SlotSpec slotSpec = new SlotSpec();
        slotSpec.set(SlotTypes.FIRST_COMPONENT,0);
        slotSpec.set(SlotTypes.OUTPUT, 1);
        this.craftingInventory = new CraftingInventory(INVENTORY_SIZE,slotSpec);
        this.inventory = craftingInventory;

    }
```

Εικόνα 71 - Κώδικας αρχικοποίησης της κλάσης MetalFurnace

Ο φούρνος μεταλλευμάτων ως κλάση ορίζει τις γενικές ιδιότητες του, καθώς και την διεπαφή χρήστη που χρειάζεται, ενώ κάθε στιγμιότυπο της κλάσης(δηλαδή καθε μοναδικός φούρνος) ορίζει την τρέχουσα του κατάσταση, όπως το αν έχει μεταλλεύματα ή/και καύσιμα υλικά.

Οι οντότητες TileEntities όπως ο φούρνος μεταλλευμάτων μπορούν να ορίσουν δικές τους λειτουργίες:

```java
@Override
public void update(float delta) {
    //Find out if the current input is a valid recipe
    RecipeDefinition recipe = world.registry.matchRecipe(craftingInventory, RecipeTypes.PROCESSING);

    //If there's a valid recipe start consuming fuel
    if (recipe != null && !hasFuel){
        consumeFuel();
    }

    if (intervalCounter >= CONSUMPTION_INTERVAL) {
        burn();
        resetInterval();
        if (recipe != null && hasFuel){
            smelt(recipe);
        }
    } else {
        intervalCounter += delta;
    }

    hasFuel = fuelCounter > 0;

}
```

Εικόνα 72 - Κώδικας της συνάρτησης update() της κλάσης MetalFurnace

4.10.3 Σύστημα οντοτήτων : Τεχνικές λεπτομέρειες

Τα δεδομένα της οντότητας του παίκτη αλλα καί τα δεδομένα των TileEntities αποθηκεύονται στην μόνιμη μνήμη σε αρχεία JSON, ενώ αντίθετα οι οντότητες τύπου ItemDrop απλά διαγράφονται κατα την έξοδο από το παιχνίδι.

Η αποθήκευση γίνεται μέσω ειδικών κλάσεων PlayerLoader και TileEntityLoader αντίστοιχα.

Οι κλάσεις αυτές κατα την έναρξη και την διάρκεια του παιχνιδιού φορτώνουν τα δεδομένα από την μόνιμη μνήμη και κατα την έξοδο του, ή την αν δεν είναι πια χρήσιμα καταγράφονται ξανά στην μόνιμη μνήμη, όπως τους ζητηθεί κατα περίπτωση από το σύστημα που θα τους καλέσει.

Η καταγραφή και φόρτωση γίνεται με την βοήθεια του libGDX με τις αντίστοιχες λειτουργίες που παρέχονται, με την γενικη διαδικασία να είναι κοινή ανάμεσα στις οντότητες, με τις λεπτομέρειες μόνο να είναι διαφορετικές.

Ο κώδικας για τον PlayerLoader για read και write αντίστοιχα αποτελεί καλό παράδειγμα της γενικής διαδικασίας.

```
public void read(){
    JsonValue playerInfo = reader.parse(playerFile).child;

    player.position.set(vec2f(playerInfo,position));
    player.spawnPoint.set(vec2f(playerInfo,spawnPoint));

    if (playerInfo.has(gameWon)){
        player.gameWon = playerInfo.get(gameWon).asBoolean();
    }

    if (playerInfo.has(quickInventory)){
        parseInventory(playerInfo.get(quickInventory), player.quickInventory);
    }

    if (playerInfo.has(mainInventory)){
        parseInventory(playerInfo.get(mainInventory), player.mainInventory);
    }

}
```

Εικόνα 73 - Κώδικας της συνάρτησης read() της κλάσης PlayerLoader

```
public void write(){
    JsonValue json = reader.parse(playerFile);
    JsonValue playerInfo = json.child;

    setVec2f(playerInfo,position,player.position);

    if (playerInfo.has(gameWon)){
        playerInfo.get(gameWon).set(player.gameWon);
    } else {
        playerInfo.addChild(gameWon,new JsonValue(player.gameWon));
    }

    if (playerInfo.has(quickInventory)) {
        playerInfo.remove(quickInventory);
    }
    playerInfo.addChild(quickInventory,serializeInventory(player.quickInventory));

    if (playerInfo.has(mainInventory)) {
        playerInfo.remove(mainInventory);
    }
    playerInfo.addChild(mainInventory,serializeInventory(player.mainInventory));

    playerFile.writeString(json.toString(), append: false);
}
```

Εικόνα 74 - Κώδικας της συνάρτησης write() της κλάσης PlayerLoader

Ενδεικτικά μπορούμε να δούμε και ένα μικρό παράδειγμα των αντίστοιχων αρχείων:

Player File

```
{
Player: {
        spawnPoint: [ 1280, 1280 ]
        position: [
                1845.6367
                1512.8154
        ]
        gameWon: true
        quickInventory: {
                0: [ 6, 1 ]
                5: [ 3, 64 ]
                2: [ 3, 64 ]
                7: [ 3, 64 ]
                4: [ 10, 64 ]
                1: [ 3, 64 ]
                6: [ 3, 64 ]
                3: [ 3, 64 ]
        }
        mainInventory: {
                0: [ 3, 64 ]
                13: [ 10, 36 ]
                5: [ 3, 64 ]
                18: [ 7, 28 ]
                10: [ 3, 64 ]
                2: [ 3, 64 ]
                15: [ 3, 64 ]
                7: [ 3, 64 ]
                20: [ 9, 1 ]
                12: [ 11, 64 ]
                25: [ 1, 27 ]
                4: [ 3, 64 ]
                17: [ 3, 64 ]
                9: [ 7, 8 ]
                22: [ 11, 1 ]
                1: [ 3, 64 ]
                14: [ 3, 64 ]
                6: [ 11, 5 ]
                19: [ 3, 64 ]
                11: [ 7, 12 ]
                3: [ 3, 64 ]
                16: [ 8, 12 ]
                8: [ 3, 64 ]
                21: [ 3, 14 ]
        }
}
}
```

itemID amount

TileEntity File

```
{
TEInfo: {
        3563: {
                3641716_13_1: {
                        position: [ 1422.0, 1396.0 ]
                        inventory: {
                                2: [ 10, 42 ]
                                1: [ 11, 5 ]
                        }
                }
                3644272_14_1: {
                        position: [ 1423.0, 1392.0 ]
                        inventory: {}
                }
                3646835_13_1: {
                        position: [ 1424.0, 1395.0 ]
                        inventory: {
                                2: [ 10, 51 ]
                                1: [ 11, 2 ]
                        }
                }
        }
        4607: {
                4724714_13_1: {
                        position: [ 1845.0, 1514.0 ]
                        inventory: {
                                0: [ 9, 1 ]
                                2: [ 1, 1 ]
                                1: [ 11, 43 ]
                        }
                }
                4717035_14_1: {
                        position: [ 1842.0, 1515.0 ]
                        inventory: {}
                }
        }
}
}
```

chunkIndex tileIndex tileID layer

Εικόνα 75 - Αρχεία στην μόνιμη μνήμη της οντότητας του παίκτη(αριστερά) και του συνόλου των οντοτήτων TileEntity(δεξιά)

4.11 Σύστημα αντικειμένων - Κύριο Σύστημα

4.11.1 Σύστημα αντικειμένων : Επισκόπηση λειτουργίας και στόχων

Το σύστημα αντικειμένων είναι υπεύθυνο για την διατήρηση και αναπαράσταση των φυσικών αντικειμένων του κόσμου σε λογικό επίπεδο, την διατήρηση τους σε λογικές μονάδες με μέγιστη χωρητικότητα, και την υποστήριξη των πληροφοριών που χρειάζονται επιπρόσθετα των λογικών αυτών μονάδων για τον συνδυασμό τους με σκοπό την παραγωγή τρίτων αντικειμένων.

4.11.2 Σύστημα αντικειμένων : Παρουσίαση και επεξήγηση κώδικα

Η βασική μονάδα που πραγματεύεται το σύστημα αντικειμένων είναι ένα ItemStack, το οποίο αντιπροσωπεύει ένα σύνολο φυσικών αντικειμένων ίδιου τύπου.

```
public class ItemStack {

    public int id;
    public byte amount;
    public ItemStack(int id, byte amount){
        this.id = id;
        this.amount = amount;
    }
}
```

Εικόνα 76 - Κώδικας της κλάσης ItemStack

Οι δύο πληροφορίες που χρειάζονται για την δημιουργία ενός ItemStack είναι ο τύπος του αντικειμένου, σε μορφή id, όπως ορίζονται στο σύστημα Registry, και η ποσότητα αυτού του αντικειμένου.

Η αμέσως μεγαλύτερη μονάδα που πραγματεύεται το σύστημα αντικειμένων είναι σύνολα απο ItemStack με μέγιστο μέγεθος σε αριθμό θυρών(Slots) που κάθε μία αντιστοιχεί σε ένα αντικείμενο(ή στην απουσία αντικειμένου), στο εξής και Inventory.

Αντικείμενα τύπου Inventory μπορούν να βρεθούν τόσο στον παίκτη όσο και σε TileEntities και αντιπροσωπεύουν το σύνολο των αντικειμένων που περιέχουν.

Τα Inventories εκτός από την αποθήκευση των ItemStack που περιέχουν υλοποιούν και βοηθητικές συναρτήσεις για την προσπέλαση των περιεχομένων τους.

```java
public class Inventory extends IntMap<ItemStack> {
    private static final Registry RY = Registry.registry;

    public static class Utils{...}

    public static class Conditions{...}

    int capacity;

    public Inventory(int capacity){
        super(capacity);
        this.capacity =capacity;
    }

    @Override
    public ItemStack get(int key) {
        if(key < capacity){
            return super.get(key);
        } else {
            return null;
        }
    }
```

Εικόνα 77 - Μέρος κώδικα της κλάσης Inventory

Το κύριο στοιχείο που μας ενδιαφέρει για την δημιουργία ενός Inventory είναι η χωρητικότητα του.

```java
public boolean hasEmptySlot(){
    for (int i = 0; i < capacity; i++) {
        if(get(i) == null) return true;
    }
    return false;
}

public int firstEmptySlot(){
    for (int index = 0; index < capacity; index++) {
        if(get(index) == null) return index;
    }
    return NOT_FOUND;
}

public boolean hasItem(int itemID){
    for (ItemStack value : values()) {
        if(value.id == itemID) return true;
    }
    return false;
}
```

Εικόνα 78 - Βοηθητικές συναρτήσεις της κλάσης Inventory

Υπάρχουν δύο ακόμα κλάσεις που επεκτείνουν την πληροφορία για τις θύρες ενός Inventory, οι CraftingInventory και SlotSpec, προσφέροντας πληροφορίες χρήσιμες για την διαδικασία παραγωγής τρίτων αντικειμένων.

```
public class CraftingInventory extends Inventory{
    public SlotSpec slotSpec;

    public CraftingInventory(int capacity, SlotSpec slotSpec) {
        super(capacity);
        this.slotSpec = slotSpec;
    }

    public ItemStack getSpecSlot(int slot) { return get(slotSpec.get(slot, NOT_FOUND)); }
}
```

Εικόνα 79 - Βοηθητικές συναρτήσεις της κλάσης Inventory

Η κλάση CraftingInventory επεκτείνει απευθείας την κλάση Inventory με την επιπρόσθετη πληροφορία σχετικά με τον ορισμό των τύπο των θυρών της, σε ένα αντικείμενο τύπου SlotSpec.

```
public class SlotSpec extends IntIntMap {

    public void set(int type,int slot){
        put(type,slot);
    }

}
```

Εικόνα 80 - Κώδικας της κλάσης SlotSpec

Οι ορισμοί των τύπων των θυρών ορίζονται στο σύστημα Registry:

```
public static class SlotTypes {
    public static final int FIRST_COMPONENT = 0;
    public static final int SECOND_COMPONENT = 1;
    public static final int THIRD_COMPONENT = 2;
    public static final int TOOL = 3;
    public static final int OUTPUT = 4;
}
```

Εικόνα 81 - Κώδικας της κλάσης SlotTypes, εσωτερικής κλάσης της κλάσης RecipeDefinition

Σε μορφή υλοποίησης, μπορούν να συνδυαστούν οι άνω κλάσεις για να οριστεί το CraftingInventory του παίκτη έτσι ώστε να μπορεί να παράγει τρία αντικείμενα από τα αντικείμενα που ηδη κατέχει:

```
SlotSpec slotSpec = new SlotSpec();
slotSpec.set(SlotTypes.FIRST_COMPONENT, 0);
slotSpec.set(SlotTypes.SECOND_COMPONENT, 1);
slotSpec.set(SlotTypes.THIRD_COMPONENT, 2);
slotSpec.set(SlotTypes.TOOL, 3);
slotSpec.set(SlotTypes.OUTPUT, 4);
this.craftingInventory = new CraftingInventory(CRAFTING_INVENTORY_SIZE,slotSpec);
```

Εικόνα 82 - Μέρος κώδικα του constructor της κλάσης Player

Και τέλος, η διαδικασία παραγωγής πραγματοποιείται από το σύστημα κόσμου, με την βοήθεια του συστήματος Registry, σχετικά με την εγκυρότητα της "συνταγής", δηλαδή ελέγχωντας αν τα αντικείμενα που προσπαθεί να συνδυάσει ο χρήστης αντιστοιχούν σε κάποιο τρίτο ορισμένο αντικείμενο.

```
@Override
public void receive(CraftingPayload payload) {
    CraftingInventory craftingInventory = world.player.craftingInventory;
    //Find out if there's a matching recipe according to the input
    RecipeDefinition recipe = world.registry.matchRecipe(craftingInventory, RecipeTypes.CRAFTING);
    if (recipe != null){
        ItemStack outputStack = craftingInventory.getSpecSlot(SlotTypes.OUTPUT);
        //If there is empty space in the output complete the craft
        if (outputStack == null){
            craftingInventory.put(SlotTypes.OUTPUT,new ItemStack(recipe.itemID,(byte)recipe.amount));
            for (IntIntMap.Entry entry : recipe.itemsRequired) {
                craftingInventory.get(entry.key).amount -= recipe.amountsRequired.get(entry.key,NOT_FOUND);
                if(craftingInventory.get(entry.key).amount==0){
                    craftingInventory.remove(entry.key);
                }
            }
        } else {
            //Or if the occupied slot is matching the item to be crafted
            if(outputStack.id == recipe.itemID && (outputStack.amount + recipe.amount) <= RY.getI(outputStack.id).maxStackSize){
                outputStack.amount += recipe.amount;
                for (IntIntMap.Entry entry : recipe.itemsRequired) {
                    craftingInventory.get(entry.key).amount -= recipe.amountsRequired.get(entry.key,NOT_FOUND);
                    if(craftingInventory.get(entry.key).amount==0){
                        craftingInventory.remove(entry.key);
                    }
                }
            }
        }
    }
}
```

Εικόνα 83 - Μέρος κώδικα της κλάσης EventHandler

4.12 Σύνολο υποσυστημάτων κόσμου - Βοηθητικά Συστήματα

Στο τελευταίο αυτό υποκεφάλαιο ανάλυσης θα δούμε ένα σύνολο από μικρά συστήματα που προσφέρουν λειτουργικότητα στον κόσμο του παιχνιδιού

4.12.1 Σύνολο υποσυστημάτων κόσμου : Επισκόπηση λειτουργίας και στόχων

- Υποσύστημα EventHandler

  Το σύστημα EventHandler, όπως ορίζει και το όνομα του, διεκπεραιώνει τα Event που δέχεται ο κόσμος μέσω του συστήματος Event

- Υποσύστημα ItemPickup

  Το σύστημα ItemPickup προσδίδει λειτουργικότητα στις οντότητες τύπου ItemDrop, επιτρέποντας του παίκτη να τις συλλέξει και να προσθέσει το αντίστοιχο αντικείμενο στο Inventory του.

- Υποσύστημα Physics

  Το σύστημα Physics παρέχει την δυνατότητα να προσθέσουμε στα στοιχεία του κόσμου μας ιδιότητες φυσικής και να τους επιτρέψουμε να αλληλεπιδράσουν μεταξύ τους

- Υποσύστημα TileEntity

  Το σύστημα TileEntity φορτώνει, αποθηκεύει, αρχικοποιεί και ενημερώνει τα TileEntities που υπάρχουν στον κόσμο.

4.12.2 Σύνολο υποσυστημάτων κόσμου : Παρουσίαση και επεξήγηση κώδικα

Τα υποσυστήματα αυτά χρησιμοποιούν ως βάση και επεκτείνουν την κλάση WorldSystem, κυρίως για την αποφυγή επανάληψης κώδικα:

```
public abstract class WorldSystem implements EventCreator {
    final World world;
    final Dispatcher dispatcher;

    public WorldSystem(World world) {
        this.world = world;
        this.dispatcher = world.dispatcher;
    }

    public abstract void update(float delta);
}
```

Εικόνα 84 - Κώδικας της κλάσης WorldSystem

- Υποσύστημα EventHandler

Το σύστημα Eventhandler λαμβάνει και διεκπεραιώνει Events όπως την διαδικασία παραγωγής αντικειμένων ή την μεταφορά αντικειμένων από ένα Inventory σε άλλο.

```java
@Override
public void receive(CraftingPayload payload) {
    CraftingInventory craftingInventory = world.player.craftingInventory;
    //Find out if there's a matching recipe according to the input
    RecipeDefinition recipe = world.registry.matchRecipe(craftingInventory, RecipeTypes.CRAFTING);
    if (recipe != null){
        ItemStack outputStack = craftingInventory.getSpecSlot(SlotTypes.OUTPUT);
        //If there is empty space in the output complete the craft
        if (outputStack == null){
            craftingInventory.put(SlotTypes.OUTPUT,new ItemStack(recipe.itemID,(byte)recipe.amount));
            for (IntIntMap.Entry entry : recipe.itemsRequired) {
                craftingInventory.get(entry.key).amount -= recipe.amountsRequired.get(entry.key,NOT_FOUND);
                if(craftingInventory.get(entry.key).amount==0){
                    craftingInventory.remove(entry.key);
                }
            }
        } else {
            //Or if the occupied slot is matching the item to be crafted
            if(outputStack.id == recipe.itemID && (outputStack.amount + recipe.amount) <= RY.getI(outputStack.id).maxStackSize){
                outputStack.amount += recipe.amount;
                for (IntIntMap.Entry entry : recipe.itemsRequired) {
                    craftingInventory.get(entry.key).amount -= recipe.amountsRequired.get(entry.key,NOT_FOUND);
                    if(craftingInventory.get(entry.key).amount==0){
                        craftingInventory.remove(entry.key);
                    }
                }
            }
        }
    }
}
```

Εικόνα 85 - Κώδικας παραγωγής αντικειμένων της κλάσης EventHandler

```java
@Override
public void receive(InventoryPayload payload) {
    Inventory src = payload.sourceInventory;
    Inventory target = payload.targetInventory;


    switch (payload.operation){
        case InventoryPayload.Operations.ADD:
            if(target.hasItem(payload.itemID) && target.firstSlotOfItem(payload.itemID, Conditions.NOT_FULL) != NOT_FOUND){
                int index = target.firstSlotOfItem(payload.itemID, Conditions.NOT_FULL);
                ItemStack itemStack = target.get(index);
                byte maxStackSize = RY.getI(itemStack.id).maxStackSize;
                if(itemStack.amount + payload.amount <= maxStackSize){
                    itemStack.amount += payload.amount;
                } else {
                    int carry = itemStack.amount + payload.amount - maxStackSize;
                    itemStack.amount = maxStackSize;
                    if(target.hasEmptySlot()){
                        target.put(target.firstEmptySlot(),new ItemStack(payload.itemID,(byte)carry));
                    }
                }

            } else {
                ItemStack itemStack = new ItemStack(payload.itemID,payload.amount);
                target.put(payload.targetIndex, itemStack);
            }
            break;

        case InventoryPayload.Operations.SWAP:
            ItemStack stack = src.remove(payload.sourceIndex);
            if(target.get(payload.targetIndex) != null){
                src.put(payload.sourceIndex, target.remove(payload.targetIndex));
            }
            target.put(payload.targetIndex, stack);
            break;
    }
}
```

Εικόνα 86 - Κώδικας μεταφοράς αντικειμένων μεταξύ Inventories της κλάσης EventHandler

- Υποσύστημα ItemPickup

Η μοναδική λειτουργία του συστήματος είναι να ελέγχει αν ο παίκτης μπορεί να συλλέξει απο το έδαφος κάθε αντικείμενο.

```
@Override
public void update(float delta) {
    for (ItemDrop itemDrop : itemDrops) {
        //Calculate if the player is near enough for pickup
        if(itemDrop != null && itemDrop.position.dst(player.position) < player.pickupRange){
            //Smooth out the motion by interpolating the movement
            tempVec.set(player.position.x + player.waistOffset.x,player.position.y + player.waistOffset.y);
            itemDrop.position.interpolate(tempVec, alpha: 0.01f, Interpolation.fastSlow);
            if(itemDrop.position.dst(tempVec) < player.pickupRange/4f) {
                InventoryPayload payload = pool.obtain();
                payload.operation = Operations.ADD;
                payload.itemID = itemDrop.itemStack.id;
                payload.amount = itemDrop.itemStack.amount;
                //Figure out if the player inventories can accept the item
                if (player.quickInventory.hasEmptySlot() || player.quickInventory.firstSlotOfItem(payload.itemID, Conditions.NOT_FULL) != Utils.NOT_FOUND) {
                    payload.targetInventory = player.quickInventory;
                    if(player.quickInventory.hasEmptySlot()){
                        payload.targetIndex = payload.targetInventory.firstEmptySlot();
                    } else {
                        payload.targetIndex = payload.targetInventory.firstSlotOfItem(payload.itemID, Conditions.NOT_FULL);
                    }
                } else if (player.mainInventory.hasEmptySlot() || player.mainInventory.firstSlotOfItem(payload.itemID, Conditions.NOT_FULL) != Utils.NOT_FOUND) {
                    payload.targetInventory = player.mainInventory;
                    if(player.mainInventory.hasEmptySlot() || player.mainInventory.firstSlotOfItem(payload.itemID, Conditions.NOT_FULL) != Utils.NOT_FOUND ){
                        payload.targetIndex = payload.targetInventory.firstEmptySlot();
                    } else {
                        payload.targetIndex = payload.targetInventory.firstSlotOfItem(payload.itemID, Conditions.NOT_FULL);
                    }
                } else {
                    payload.free();
                    continue;
                }
                dispatcher.dispatch(Events.INVENTORY, payload);
                itemDrop.itemStack = null;
                itemDrops.remove(itemDrop);
            }
        }
    }
}
```

Εικόνα 87 - Κώδικας συλλογής αντικειμένων από τον παίκτη, της κλάσης ItemPickupSys

- Υποσύστημα Physics

Το σύστημα Physics υλοποιεί τις ανάγκες που έχουμε από τα στοιχεία του κόσμου μας να αλληλεπιδράσουν με βάση κανόνων της φυσικής, όπως το να μην μπορεί να διαπεράσει ο παίκτης μας στερεά αντικείμενα.

Τις δυνατότητές του τις λαμβάνει απο την μηχανή φυσική Box2D που παρέχεται απο το libGDX.

Κατα την δημιουργία του συστήματος αρχικοποιούνται κοινοί ορισμοί φυσικών σωμάτων, τους οποίους στην συνέχεια μπορεί να προσθέσει στα στοιχεία του κόσμου μας για να επιτρέψει έτσι την αλληλεπίδραση των σωμάτων απο την μηχανή φυσικής.

Στην διάρκεια εκτέλεσης του παιχνιδιού το σύστημα αυτο συνεχώς ορίζει φυσικά σώματα στα στοιχεία(εφόσον το απαιτούν) που βρίσκονται κοντά στον παίκτη, ενώ αν ο παίκτης απομακρυνθεί από ένα προηγουμένως ορισμένο στοιχείο αφαιρεί το σώμα αυτό από την προσομοίωση της μηχανής φυσικής.

```
public PhysicsSys(World world) {
    super(world);
    b2dWorld = world.b2dWorld;
    tileBodyDef = new BodyDef();

    fullBox = new PolygonShape();
    fullBox.setAsBox( hx: 0.5f, hy: 0.5f);

    halfBox = new PolygonShape();
    halfBox.setAsBox( hx: 0.25f, hy: 0.25f);

    quarterBox = new PolygonShape();
    quarterBox.setAsBox( hx: 0.125f, hy: 0.125f);


    BodyDef playerDef = new BodyDef();
    playerDef.type = BodyDef.BodyType.DynamicBody;
    playerDef.position.set(world.player.position);
    playerDef.fixedRotation = true;

    world.player.body = b2dWorld.createBody(playerDef);

    FixtureDef fixtureDef = new FixtureDef();
    fixtureDef.shape = halfBox;
    fixtureDef.density = 1f;
    fixtureDef.restitution = 0f;


    world.player.body.createFixture(fixtureDef);

}
```

Εικόνα 88 - Κώδικας του constructor της κλάσης PhysicsSys

```java
@Override
public void update(float delta) {
    Player p = world.player;
    //Ensure tiles near player have physics bodies enabled
    for (Chunk chunk : world.chunks.values()) {
        for (IntMap.Entry<Tile> tile : chunk.tiles) {
            //Only generate colliders within close range
            if (p.position.dst(tile.value.position) < PHYS_ENABLE_DISTANCE){
                TileDefinition tileDef = world.registry.getT(tile.value.getId(Tile.MID));
                if ( tileDef.is(Attributes.COLLIDABLE) && chunk.bodies.get(tile.value.index) == null ){
                    temp.set(tile.value.position.x + physPosOffset.x, tile.value.position.y + physPosOffset.y);
                    tileBodyDef.position.set(temp);

                    Body body = world.b2dWorld.createBody(tileBodyDef);
                    chunk.bodies.put(tile.value.index,body);

                    int colliderType = ((Collidable)tileDef.attributeValues.get(Attributes.COLLIDABLE)).colliderType;
                    switch (colliderType){

                        case Colliders.FULL_BOX:
                            body.createFixture(fullBox, density: 0f);
                            break;

                        case Colliders.HALF_BOX:
                            body.createFixture(halfBox, density: 0f);
                            break;

                        case Colliders.QUARTER_BOX:
                            body.createFixture(quarterBox, density: 0f);
                            break;

                        default:
                            body.createFixture(fullBox, density: 0f);

                    }


                }
```

Εικόνα 89 - Κώδικας δημιουργίας σωμάτων και προσθήκης τους στην μηχανή φυσικής, της κλάσης PhysicsSys

```java
            //If the tile is too far, delete any colliders
            } else {
                if(chunk.bodies.get(tile.value.index) != null){
                    world.b2dWorld.destroyBody(chunk.bodies.remove(tile.value.index));
                }
            }

        }
    }


}
```

Εικόνα 90 - Κώδικας καταστροφής σωμάτων και αφαίρεσης τους απο την μηχανή φυσικής, της κλάσης PhysicsSys

- Υποσύστημα TileEntity

Το υποσύστημα TileEntity ελέγχει κάθε frame αν υπάρχουν κελιά τύπου TileEntity στον κόσμο που δεν έχουν το αντίστοιχο αντικείμενο καταχωρημένο, και αν δεν είναι τότε τα δημιουργεί.

Επίσης, ενημερώνει το σύνολο των αντικειμένων τύπου TileEntity που υπάρχουν, δίνοντας τους έτσι ευκαιρία να εκτελέσουν τις λειτουργίες τους.

```
@Override
public void update(float delta) {
    Registry RY = world.registry;
    for (Chunk chunk : world.chunks.values()) {
        for (Tile tile : chunk.tiles.values()) {
            int tileIndex = calcIndex(tile.position);
            boolean isFloorRTE = RY.getT(tile.getId(Tile.FLOOR)).isRTE;
            boolean isMidRTE = RY.getT(tile.getId(Tile.MID)).isRTE;
            if (isFloorRTE || isMidRTE){
                if (tileEntities.get(tileIndex) == null){
                    tileEntities.put(tileIndex,new IntMap<TileEntity>());
                }
            } else {
                continue;
            }

            if ( isFloorRTE && tileEntities.get(tileIndex).get(Tile.FLOOR) == null) {
                tileEntities.get(tileIndex).put(Tile.FLOOR,TileEntity.newTE(world,tile.position.x,tile.position.y,Tile.FLOOR, tile.getId(Tile.FLOOR)));
            }
            if ( isMidRTE && tileEntities.get(tileIndex).get(Tile.MID) == null) {
                tileEntities.get(tileIndex).put(Tile.MID,TileEntity.newTE(world,tile.position.x,tile.position.y,Tile.MID, tile.getId(Tile.MID)));
            }
        }
    }

    for (IntMap.Entry<IntMap<TileEntity>> entry : tileEntities) {
        for (IntMap.Entry<TileEntity> tileEntityEntry : entry.value) {
            tileEntityEntry.value.update(delta);
        }
    }
}
```

Εικόνα 91 - Κώδικας δημιουργίας και ενημέρωσης των TileEntities, της κλάσης TileEntitySys

Κεφάλαιο 5 - Επίλογος Ανάλυσης

Κατα την διάρκεια της ανάλυσης παρουσιάστηκε μεγάλο μέρος του κώδικα ενώ παραλείφθηκε το υπόλοιπο καθώς κρίθηκε οτι δεν θα βοηθούσε περαιτέρω στην κατανόηση της υλοποίησης.

Στα ίδια πλαίσια, πιστεύουμε ότι η καλύτερη κατανάλωση του περιεχομένου της παρόντος εργασίας είναι παράλληλα με την εκτέλεση του παιχνιδιού.

Πολλές από τις αλληλεπιδράσεις μεταξύ των συστημάτων θα γίνουν ευκολότερα αντιληπτές, τόσο σε σκόπο όσο και σε λειτουργία αν μετά την πρώτη ανάγνωση τους ως ξεχωριστά συστήματα αναζητηθούν κατα την εκτέλεση του παιχνιδιού.

Κεφάλαιο 6 - Συμπεράσματα και προτάσεις περαιτέρω ανάπτυξης

Η διαδικασία της ανάπτυξης του Fabritaku είχε πολλές προκλήσεις αλλα ταυτόχρονα ήταν και μεγάλη ευκαιρία μάθησης.

Ως συμπέρασμα, κρίνουμε την συνολική διαδικασία ανάπτυξης ενός παιχνιδιού μέσω ενός game development framework ως μια άρτια ευκαιρία εξάσκησης των προγραμματιστικών ικανοτήτων ενός ατόμου.

Η διαδικασία περιλαμβάνει την οριοθέτηση των σκοπών του παιχνιδιού, τον ορισμό της μεθόδου ανάπτυξης, την κατανόηση των χρονικών και χρηματικών απαιτήσεων, την επιλογή του κατάλληλου ανα περίπτωση εργαλείου ανάπτυξης, την αναζήτηση και κατανόηση τεκμηριωσης διαφόρων λογισμικών καθώς και την επίλυση προβλημάτων.

Θέλουμε επίσης να τονίσουμε την χρησιμότητα που παρουσίασαν λογισμικά version control και IDE στην ανάπτυξη του παιχνιδιού και να επιβεβαιώσουμε τις δυνατότητες του libGDX.

Υπάρχουν διάφοροι τρόποι που θα μπορούσε κάποιος να αναπτύξει ή/και να προσθέσει στην ιδέα αυτής της εργασίας.

- Θα μπορούσε να παρέμβει σε υπάρχοντα συστήματα και να τα επεκτείνει ή να δημιουργήσει καινούργια.
- Θα μπορούσε να δημιουργήσει το αντίστοιχο παιχνίδι με κάποια άλλη γλώσσα προγραμματισμού και εργαλεία ανάπτυξης
- Θα μπορούσε να τροποποιήσει την αρχιτεκτονική του παιχνιδιού έτσι ώστε να μπορεί να υποστηρίξει παραπάνω από έναν παίκτη στον ίδιο κόσμο ταυτόχρονα.

Πίνακας Συντομογραφιών

GWT	Google Web Toolkit
IDE	Integrated Development Environment
SDK	Software Development Kit
JSON	JavaScript Object Notation
XML	Extensible Markup Language
UI	User Interface

Βιβλιογραφία

1. Stefan Gustavson (2005) - Simplex noise demystified
2. Marcus Toftedahl, Henrik Engström (2019) - Taxonomy of Game Engines and the Tools that Drive the Industry
3. https://www.redblobgames.com/maps/terrain-from-noise/
4. https://github.com/KdotJPG/OpenSimplex2
5. https://libgdx.com/dev/setup/
6. http://www.cs.cornell.edu/courses/cs3152/2021sp/resources/engine/
7. https://libgdx.com/dev/simple-game/
8. https://libgdx.com/dev/simple-game-extended/
9. https://github.com/rafaskb/awesome-libgdx
10. https://github.com/libgdx/libgdx/wiki/The-application-framework
11. https://github.com/libgdx/libgdx/wiki/The-life-cycle
12. https://github.com/libgdx/libgdx/wiki/Managing-your-assets
13. https://github.com/libgdx/libgdx/wiki/File-handling
14. https://github.com/libgdx/libgdx/wiki/Skin-Composer
15. https://github.com/EsotericSoftware/tablelayout