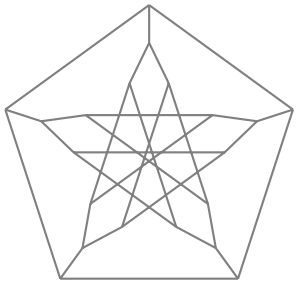
Πανεπιστήμιο Δυτικής Μακεδονίας
Σχολή Θετικών Επιστημών
Τμήμα Πληροφορικής

Δημιουργία δια-πλατφορμικού παιχνιδιού δύο διαστάσεων χρησιμοποιώντας προσέγγιση βασισμένη στα δεδομένα

Πτυχιακή Εργασία

του

ΓΙΑΠΟΤΖΗ ΓΕΩΡΓΙΟΥ



Επιβλέπων: Γεώργιος Σίσιας
Επιστ. Συνεργάτης

Καστοριά, Μάιος 2021

Πανεπιστήμιο Δυτικής Μακεδονίας
Σχολή Θετικών Επιστημών
Τμήμα Πληροφορικής

Δημιουργία δια-πλατφορμικού παιχνιδιού δύο διαστάσεων χρησιμοποιώντας προσέγγιση βασισμένη στα δεδομένα

Πτυχιακή Εργασία

του

ΓΙΑΠΟΤΖΗ ΓΕΩΡΓΙΟΥ

Επιβλέπων: Γεώργιος Σίσιας
Επιστ. Συνεργάτης

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 27α Μαΐου 2021.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Δημήτριος Ι. Βέργαδος	Σπυρίδων Νικολάου	Νίκος Δημόκας
Επίκουρος Καθηγητής	Λέκτορας	Επίκουρος Καθηγητής

Καστορια, Μάιος 2021

Πανεπιστήμιο Δυτικής Μακεδονίας
Σχολή Θετικών Επιστημών
Τμήμα Πληροφορικής

Περίληψη

Κανείς δεν μπορεί να αντικρούσει το γεγονός πως τα βιντεοπαιχνίδια στις μέρες μας είναι μία αρκετά διαδεδομένη δραστηριότητα που πολλοί κάνουν στον ελεύθερό τους χρόνο, μικροί και μεγάλοι. Σε συνδυασμό με την εξέλιξη της τεχνολογίας η βιομηχανία αυτή γνώρισε αρκετά μεγάλη ανάπτυξη, κάτι που οδήγησε στην δημιουργία καινοτόμων ιδεών, όπως για παράδειγμα η χρήση μάσκας επαυξημένης πραγματικότητας. Είναι συνηθισμένο φαινόμενο ένα παιχνίδι να υποστηρίζει περισσότερες από μία πλατφόρμες, πχ. Windows, Linux, Mac, δίνοντας έτσι στον χρήστη ένα εύρος από επιλογές και την ελευθερία να απολαύσει την εμπειρία, ανάλογα με την προτίμηση του, σε όποια πλατφόρμα αυτός επιθυμεί. Οι παραπάνω λόγοι γέννησαν νέες τεχνικές βελτιστοποίησης, κατά κύριο λόγο στον τομέα της απόδοσης των προγραμμάτων ώστε να μπορούν να ανταπεξέλθουν με τις υπόλοιπες αλλαγές και μία από αυτές είναι ο προγραμματισμός βασισμένος στα δεδομένα.

Η συγκεκριμένη εργασία μελετά τον τρόπο κατασκευής παιχνιδιών δύο διαστάσεων, κάνοντας χρήση του προγραμματισμού βασισμένου στα δεδομένα έναντι του πιο διαδεδομένου αντικειμενοστραφή προγραμματισμού και ερευνά κατά πόσο αυτή η απόφαση θα μπορούσε να βοηθήσει τόσο τον προγραμματιστή στην δουλειά του όσο και να βελτιώσει την εμπειρία του τελικού χρήστη.

Λέξεις Κλειδιά

Προγραμματισμός βασισμένος στα δεδομένα, Αντικειμενοστραφής προγραμματισμός, βιντεοπαιχνίδια

Abstract

No one can deny the fact that video games nowadays are a fairly common activity that many do in their spare time, young and old. Combined with the evolution of technology, this industry has experienced considerable growth, which has led to the creation of innovative ideas, such as the use of augmented reality masks. It is a common phenomenon for a game to support more than one platform, eg Windows, Linux, Mac, thus giving the user a range of options and the freedom to enjoy the experience, depending on his preference, on any platform he wishes. The above reasons gave birth to new optimization techniques, mainly in the field of performance of programs so that they can cope with the rest of the changes, and one of those techniques being data-oriented programming.

This thesis examines how to create two-dimensional games, using data-oriented programming versus the most common object-oriented programming, and investigates whether this decision could help both the developer in his work and improve the end-user experience.

Keywords

Data Oriented Programming, Object Oriented Programming, Video Games

Περιεχόμενα

Περίληψη **1**

Abstract **2**

Κατάλογος σχημάτων **5**

Κατάλογος πινάκων **6**

1 Εισαγωγή **7**
1.1 Στόχος της εργασίας 7
1.2 Διάρθρωση της εργασίας 7

2 Θεωρητικό υπόβαθρο **8**
2.1 Τι σημαίνει προγραμματισμός βασισμένος στα δεδομένα 8
2.1.1 Πλεονεκτήματα και Μειονεκτήματα 8
2.1.2 Χρήσεις 9
2.2 CPU Cache 9
2.2.1 Cache Miss και Cache Hit 10
2.2.2 Cache lines 10
2.2.3 Cache Prefetch 11
2.3 Locality of reference 11
2.3.1 Γιατί είναι σημαντικό το locality of reference 11
2.4 Design Patterns 11
2.4.1 Entity Component System 12
2.4.2 Game loop 12
2.4.3 State Pattern 13
2.5 Κάμερα 13
2.5.1 View 13
2.5.2 Viewport 14
2.6 Σχεδιοκίνηση 14

3 Υλοποίηση **15**
3.1 Σημείο εισόδου εφαρμογής 15
3.2 Κλάση Game 15
3.3 Κλάση World 16

3.4 Κλάση Context 17
3.5 Καταστάσεις και στοίβα καταστάσεων 17
3.5.1 StateStack 17
3.5.2 State 18
3.6 Components 20
3.7 Systems 21
3.7.1 System Manager 21
3.8 Διαχείρηση πόρων 23
3.9 Μουσική και Ηχητικά εφέ 24
3.9.1 Music 24
3.9.2 SoundBuffer και Sound 25
3.10 Βοηθητικές κλάσεις 25
3.10.1 Συμπληρωματικές κλάσεις για την Box2D 26
3.10.2 Λοιπές 27
3.11 Εργαλεία 27
3.11.1 CMake 27
3.11.2 entt 28
3.11.3 SFML 28
3.11.4 Box2D 28
3.11.5 spdlog 29
3.11.6 fmt 29
3.11.7 Tiled 29
3.11.8 tmxlite 29

4 Πείραμα **30**
4.1 Εφαρμογή σχεδιασμένη με OOP 30
4.2 Εφαρμογή σχεδιασμένη με DOP 31
4.3 Αποτελέσματα και Παρατηρήσεις 31

5 Συμπεράσματα και Μελλοντικές επεκτάσεις **33**

Παραρτήματα **34**

Α΄ Παράρτημα **35**
Α΄.1 Κώδικας πειράματος 35
Α΄.1.1 Κοινές κλάσεις μεταξύ των δύο προγραμμάτων 35
Α΄.1.2 Πρόγραμμα σχεδιασμένο με OOP 36
Α΄.1.3 Πρόγραμμα σχεδιασμένο με DOP 44

Β΄ Παράρτημα **58**
Β΄.1 Κώδικας του κυρίως προγράμματος της εργασίας 58

Βιβλιογραφία **163**

Κατάλογος σχημάτων

2.1 Δεδομένα του προγράμματος μέσα στην μνήμη με χρήση του OOP, © Wilmer Lin [21] 8
2.2 Δεδομένα του προγράμματος μέσα στην μνήμη με χρήση του DOP, © Wilmer Lin [21] 9
2.3 Cache και κύρια μνήμη, © William Stallings [9] 10
2.4 Αναπαράσταση καρέ ανά δευτερόλεπτο, © Drew Campbell [12] 13
2.5 Λειτουργίες ενός View, © Laurent Gomila[16] 14
2.6 Σχεδιοκίνηση του κυρίως χαρακτήρα καθώς περπατάει 14

3.1 Αρνητικές επιπτώσεις της διακύμανσης του χρόνου dt στον κώδικα της φυσικής, © Artur Moreira, Henrik Vogelius Hansson, and Jan Haller [1] . 16
3.2 Title Screen 18
3.3 Main Menu 19
3.4 Game Screen 19
3.5 Pause Screen 20
3.6 Λογότυπο CMake, Πηγή: https://cmake.org/files/logos/. Ημερομηνία πρόσβασης: 21-4-2021 27
3.7 Λογότυπο entt, Πηγή: https://github.com/skypjack/entt. Ημερομηνία πρόσβασης: 21-4-2021 28
3.8 Λογότυπο SFML, Πηγή: https://www.sfml-dev.org/download/goodies/. Ημερομηνία πρόσβασης: 21-4-2021 28
3.9 Λογότυπο Box2D, Πηγή: https://en.wikipedia.org/wiki/File:Box2D_logo.svg. Ημερομηνία πρόσβασης: 21-4-2021 28
3.10 Λογότυπο Tiled, Πηγή: https://www.mapeditor.org/. Ημερομηνία πρόσβασης: 21-4-2021 29

4.1 Παράθυρο εφαρμογής σχεδιασμένης με OOP 30
4.2 Παράθυρο εφαρμογής σχεδιασμένης με DOP 31

Κατάλογος πινάκων

4.1 Μέτρηση απόδοσης 32

Κεφάλαιο 1

Εισαγωγή

1.1 Στόχος της εργασίας

Στόχος είναι η δημιουργία ενός παιχνιδιού, το οποίο θα εξυπηρετήσει στην παρουσίαση και ανάλυση του προγραμματισμού με βάση τα δεδομένα (DOP), όπου θα δράσει ως ένα παράδειγμα χρήσης αυτού του τρόπου και θα δώσει την δυνατότητα να αναδειχτούν τα πλεονεκτήματα αλλά και τα μειονεκτήματα του. Γίνεται σύγκριση με τον πιο διαδεδομένο τύπο προγραμματισμού, τον Αντικειμενοστραφή προγραμματισμό (OOP) για την απόκτηση μιας ευρύτερης ιδέας και γνώμης πάνω στο θέμα.

1.2 Διάρθρωση της εργασίας

Η εργασία είναι χωρισμένη σε 5 κεφάλαια. Αρχικά γίνεται μία επισκόπηση του θεωρητικού μέρους για να γίνουν αντιληπτές οι βασικές έννοιες που σχετίζονται με το προγραμματισμό βασισμένο στα δεδομένα, στην συνέχεια γίνεται εκτενής αναφορά στα εργαλεία και τις τεχνικές που χρησιμοποιήθηκαν για να ολοκληρωθεί η εργασία, έπειτα ένα κομμάτι είναι αφιερωμένο στο πείραμα που δημιουργήθηκε για λόγους σύγκρισης OOP με DOP με στόχο την ανάδειξη των διαφορών μεταξύ των δύο τρόπων στην πράξη και τέλος ακολουθεί η ανάλυση των αποτελεσμάτων και των συμπερασμάτων που προέκυψαν.

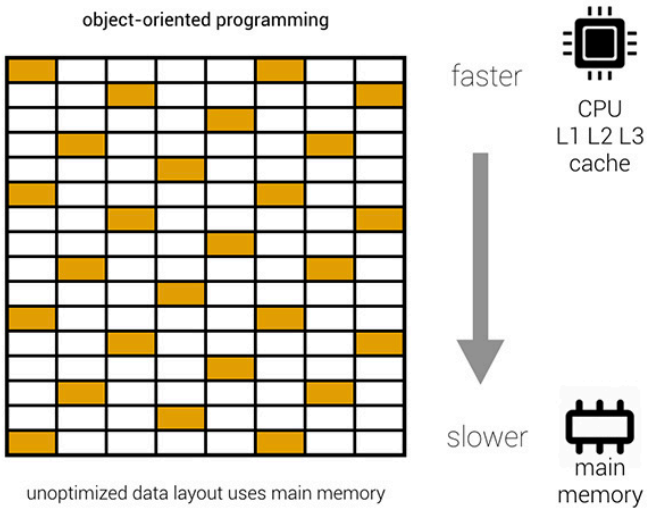
Κεφάλαιο 2

Θεωρητικό υπόβαθρο

2.1 Τι σημαίνει προγραμματισμός βασισμένος στα δεδομένα

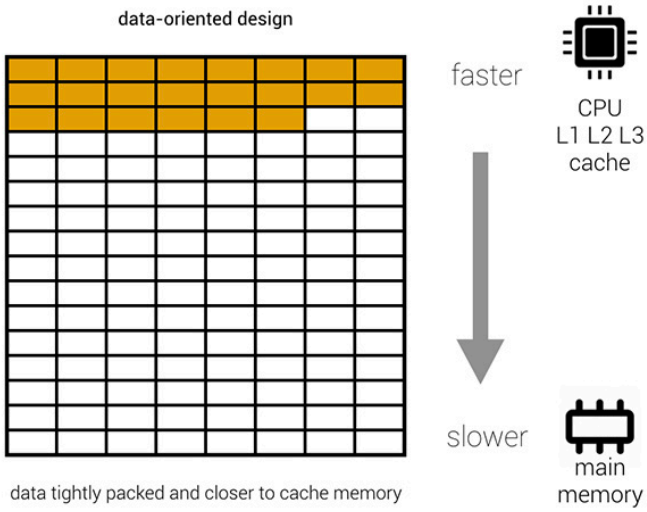
Ο προγραμματισμός βασισμένος στα δεδομένα(DOP) είναι μια διαφορετική προσέγγιση στον σχεδιασμό προγραμμάτων η οποία στοχεύει στην βελτιστοποίηση της χρήσης της μνήμης cache του επεξεργαστή του υπολογιστή. Η τεχνική αυτή οργανώνει τον κώδικα γύρω από τα δεδομένα του προβλήματος που προσπαθούμε να λύσουμε. Αυτό σημαίνει ότι οδηγούμαστε σε έναν διαχωρισμό ανάμεσα στον κώδικα και τα δεδομένα με αποτέλεσμα το πρόβλημα να γίνεται πιο εύκολα αντιληπτό από τον προγραμματιστή και έτσι να αντιμετωπίζεται με μεγαλύτερη ευκολία.

2.1.1 Πλεονεκτήματα και Μειονεκτήματα



Σχήμα 2.1: *Δεδομένα του προγράμματος μέσα στην μνήμη με χρήση του OOP, © Wilmer Lin [21]*

Όταν προγραμματίζουμε χρησιμοποιώντας OOP, τα δεδομένα μέσα μνήμη είναι σκορπισμένα και όχι σε γειτονικά κελιά, πράγμα που σημαίνει ότι μέρος των δεδομένων που



Σχήμα 2.2: *Δεδομένα του προγράμματος μέσα στην μνήμη με χρήση του DOP, © Wilmer Lin [21]*

χρειάζεται το πρόγραμμα θα βρίσκονται σε πιο αργή μνήμη(βλ. Σχήμα 2.1). Σε αντίθεση με το OOP, το DOP είναι φιλικό ως προς την μνήμη και προσπαθεί να αποθηκεύσει τα δεδομένα με όσα κενά το λιγότερο γίνεται ανάμεσα τους. Με αυτόν τον τρόπο τα χρήσιμα δεδομένα ως προς το πρόγραμμα είναι κοντά το ένα με το άλλο και μένουν όσο γίνεται περισσότερο στην γρήγορη μνήμη του υπολογιστή(βλ Σχήμα 2.2).

Το σημαντικότερο και ίσως το μοναδικό μειονέκτημα είναι ότι για να μπορέσει κάποιος να εφαρμόσει το DOP στην πράξη, θα πρέπει να αλλάξει τον τρόπο σκέψης του και γενικότερα τον τρόπο επίλυσης προβλημάτων που έχει συνηθίσει. Αυτό συμβαίνει επειδή η πλειοψηφία των προγραμματιστών μαθαίνει να αντιμετωπίζει προβλήματα με αντικειμενοστραφή ή άλλο τρόπο.

2.1.2 Χρήσεις

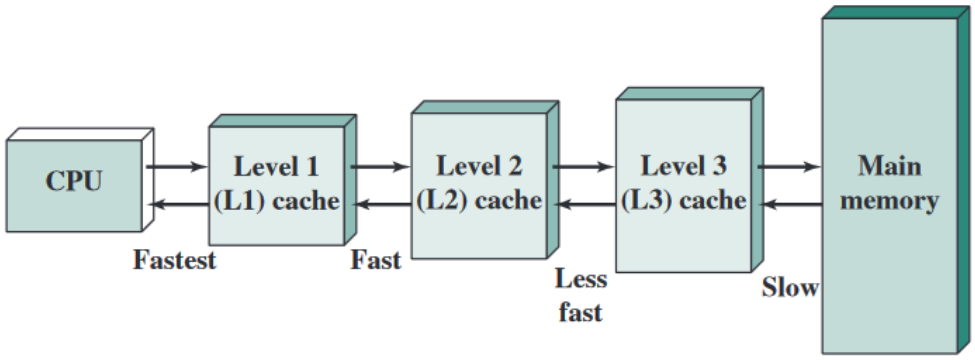
Το DOP χρησιμοποιείται κατά κύριο λόγο στην δημιουργία βιντεοπαιχνιδιών. Στην συγκεκριμένη βιομηχανία ένας από τους κύριους στόχους είναι η μεγιστοποίηση της απόδοσης και πολλές φορές οι απαιτήσεις για να τρέξουν τέτοιου είδους προγράμματα είναι συνήθως μεγάλες, οπότε χρειάζεται ένας σχετικά ικανός υπολογιστής για να μπορέσει να ανταπεξέλθει. Για αυτόν τον λόγο, η σημαντική βελτίωση που προσφέρει σε σχέση με το OOP είναι που το κάνει την καλύτερη επιλογή.

2.2 CPU Cache

Για να καταλάβουμε τον λόγο που η πρόσβαση στην μνήμη επηρεάζει την απόδοση, πρέπει πρώτα να κατανοήσουμε τον τρόπο με τον οποίο οι μοντέρνοι επεξεργαστές γράφουν και διαβάζουν την μνήμη. Η πρόσβαση στην κύρια μνήμη του υπολογιστή, την

μνήμη RAM, είναι μία αρκετά αργή διαδικασία, έτσι για να μειωθεί το κόστος διαβάσματος και γραψίματος σε αυτή την μνήμη, οι μοντέρνοι επεξεργαστές χρησιμοποιούν μία ή και περισσότερες μνήμες υψηλής ταχύτητας, τις λεγόμενες μνήμες cache.

Στην μνήμη cache αποθηκεύονται αντίγραφα των δεδομένων που βρίσκονται στην κύρια μνήμη και χρησιμοποιούνται περισσότερο από τα υπόλοιπα. Ακόμα, η μνήμη cache είναι τοποθετημένη κοντά στον επεξεργαστή με αποτέλεσμα η απόσταση που χρειάζεται για να μεταφερθούν τα δεδομένα να είναι μικρή, άρα και πιο γρήγορη από αν γινόταν μεταφορά από την κύρια μνήμη. Οι μοντέρνοι επεξεργαστές αποτελούνται από πολλαπλές στρώσεις αυτής της μνήμης οι οποίες ονομάζονται L1, L2, L3 και μερικοί επεξεργαστές σπάνια περιέχουν και τέταρτο επίδεδο το L4.



Σχήμα 2.3: *Cache και κύρια μνήμη, © William Stallings [9]*

Στις επόμενες υποενότητες θα παρουσιαστούν μερικές βασικές ορολογίες για την λειτουργία της μνήμης cache που είναι απαραίτητες για την κατανόηση του στόχου αυτής της εργασίας.

2.2.1 Cache Miss και Cache Hit

Το cache miss είναι ένα φαινόμενο που προκύπτει όταν ο επεξεργαστής κάνει απόπειρα να διαβάσει ή να γράψει δεδομένα στην μνήμη cache και δεν τα καταφέρνει με αποτέλεσμα να χρειαστεί να επικοινωνήσει με την κύρια μνήμη. Κατά την διάρκεια αυτού του φαινομένου ο επεξεργαστής περιμένει μέχρι να ολοκληρωθεί η διαδικασία επικοινωνίας με την κύρια μνήμη. Το ιδανικό που θα θέλαμε να γίνεται είναι το cache hit, όπου ο επεξεργαστής βρίσκει αυτό που χρειάζεται μέσα στην γρήγορη μνήμη cache και δεν χρειάζεται να επικοινωνήσει με την αργή κύρια μνήμη.

2.2.2 Cache lines

Για να αντιμετωπιστεί το φαινόμενο του cache miss όσο γίνεται καλύτερα, φορτώνει περισσότερα δεδομένα στην cache από όσα ζητήθηκαν. Για παράδειγμα αν γίνει απόπειρα να διαβαστούν τα περιεχόμενα μιας ακέραιας μεταβλητής int, που είναι τυπικά 32 ή 64 bit ανάλογα την αρχιτεκτονική του συστήματος, αντί να διαβάσει μόνο αυτήν την μεταβλητή θα διαβάσει ένα μεγαλύτερο σε μέγεθος κομμάτι μέσα στην μνήμη που

θα περιέχει αυτό που ζητήθηκε με την προϋπόθεση πως τα δεδομένα σε αυτό το κομμάτι να είναι γειτονικά. Η ιδέα πίσω από όλο αυτό είναι πως γενικά τα προγράμματα προσπελάζουν την μνήμη σειριακά και ενώ η πρώτη απόπειρα διαβάσματος μπορεί να είναι αποτυχημένη (cache miss), οι απόπειρες που θα ακολουθήσουν στην συνέχεια θα είναι επιτυχείς (cache hit). Το μέγεθος ενός cache line είναι τυπικά 64 bytes.

2.2.3 Cache Prefetch

Το cache prefetch είναι μία ειδική λειτουργία της μνήμης cache η οποία αυτό που κάνει είναι να "μαντεύει" ποια δεδομένα θα χρησιμοποιηθούν από την κύρια μνήμη και δημιουργεί αντίγραφα αυτών πριν ακόμα χρειαστούν. Αποτέλεσμα αυτού είναι να μειώνετε το φαινόμενο του cache miss και ως εκ τούτου η πρόσβαση σε αυτά τα δεδομένα να γίνει πολύ γρηγορότερα από τον επεξεργαστή.

2.3 Locality of reference

Locality of reference είναι η τάση του επεξεργαστή να προσπελάζει τις ίδιες διευθύνσεις μνήμης επαναληπτικά σε μικρό χρονικό διάστημα.[9] Υπάρχουν δύο κατηγορίες και είναι οι εξής:

- Temporal: Αναφέρεται στην τάση του επεξεργαστή να προσπελάζει τις θέσεις μνήμης που έχουν χρησιμοποιηθεί πρόσφατα[9]
- Spatial: Σχετίζεται με την χρήση των δεδομένων που είναι σχετικά κοντά μέσα στην μνήμη και αντικατοπτρίζει την τάση των επεξεργαστών να προσπελάζουν ακολουθιακά τα δεδομένα μέσα στην μνήμη, όπως για παράδειγμα διασχίζοντας έναν πίνακα μίας διάστασης.[9]

2.3.1 Γιατί είναι σημαντικό το locality of reference

Υπολογιστικά συστήματα που παρουσιάζουν το φαινόμενο του locality of reference είναι καλοί υποψήφιοι για βελτιστοποίηση απόδοσης με την χρήση τεχνικών όπως prefetching και caching και μπορούμε να την πετύχουμε με την χρήση του προγραμματισμού βασισμένου στα δεδομένα, και όπως θα δούμε στο κεφαλαίο 4 το οποίο ερευνά την βελτίωση στην απόδοση ενός προγράμματος κατασκευασμένου με OOP και DOP, τα αποτελέσματα είναι αρκετά ικανοποιητικά.

2.4 Design Patterns

Τα design patterns είναι επαναχρησιμοποιήσιμες λύσεις σε προβλήματα που βρίσκουμε ξανά και ξανά μπροστά μας στον τομέα σχεδίασης λογισμικού. Στην πραγματικότητα δεν είναι κάτι που μπορεί να χρησιμοποιηθεί άμεσα για την επίλυση όλων

των προβλημάτων, αλλά δύναται να ενεργήσει ως ένα πρότυπο το οποίο θα συμβάλλει στην σχεδίαση του λογισμικού που θα είναι η πραγματική επίλυση στο πρόβλημα. Υπάρχουν 3 κατηγορίες και είναι οι εξής:

- Structural patterns, τα οποία επιτρέπουν την καλύτερη οργάνωση κλάσεων και αντικειμένων σε μεγαλύτερες και πιο περίπλοκες δομές
- Behavioural patterns, τα οποία αναγνωρίζουν τρόπους επικοινωνίας μεταξύ αντικειμένων
- Creational patterns, τα οποία δίνουν την δυνατότητα δημιουργίας αντικειμένων με βάση κάποια κριτήρια και με ελεγχόμενο τρόπο.

Στα υποκεφάλαια που ακολουθούν περιγράφονται τα design patterns που χρησιμοποιήθηκαν για την δημιουργία της εργασίας.

2.4.1 Entity Component System

Το ECS χρησιμοποιείτε κυρίως στην δημιουργία βιντεοπαιχνιδιών και απαρτίζεται από 3 στοιχεία,

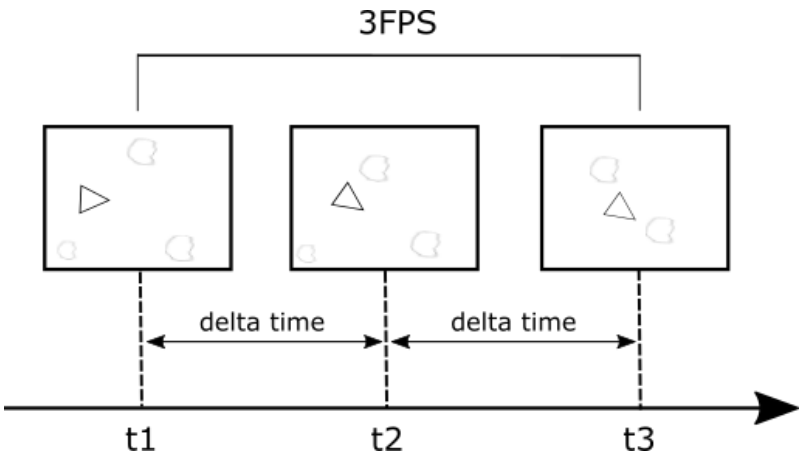
- τα Entities
- τα Components
- και τα Systems

Βασική ιδέα του ECS είναι τα λεγόμενα Entities. Τα Entities δεν είναι τίποτα παραπάνω από ένας αναγνωριστικός unsigned integer, ένα “δοχείο“ για τα Components που είναι συσχετισμένα με αυτό. Τα Components είναι σκέτα δεδομένα όπως για παράδειγμα το Transform μιας οντότητας μέσα στον κόσμο. Η σχέση που υπάρχει ανάμεσα στα Entities και τα Components είναι ένα προς πολλά. Τέλος, τα Systems χρησιμοποιούνται για να τροποποιήσουν, επεξεργαστούν, προσθέσουν ή διαγράψουν Components από τα Entities.

2.4.2 Game loop

Το game loop τρέχει συνεχώς κατά την διάρκεια του παιχνιδιού. Κάθε επανάληψη ονομάζεται καρέ. Σε κάθε καρέ, επεξεργάζεται η είσοδος από τον χρήστη, ενημερώνεται η κατάσταση του παιχνιδιού και απεικονίζεται το παιχνίδι στην οθόνη. Αν μετρήσουμε πόσο γρήγορα πραγματοποιείται ένας κύκλος παιχνιδιού παίρνουμε τα “καρέ ανά δευτερόλεπτο“, το λεγόμενο FPS. Αν το παιχνίδι τρέχει πολύ γρήγορα, δηλαδή έχει πολλά FPS, τότε κυλάει ομαλά και γρήγορα. Αν τρέχει αργά τότε έχει ελάχιστα FPS. Αυτό σημαίνει πως σε διαφορετικό hardware θα υπάρχει διαφορετικό FPS πράγμα που δεν είναι αυτό που θέλουμε να πετύχουμε. Το πρόβλημα αυτό λύνεται με την χρήση του delta time ή αλλιώς χρόνο dt.

Ο χρόνος dt συμβολίζει την διαφορά του χρόνου που πέρασε από το προηγούμενο καρέ που ζωγραφίστηκε και του τρέχων καρέ. Για παράδειγμα, αν υποθέσουμε



Σχήμα 2.4: *Αναπαράσταση καρέ ανά δευτερόλεπτο, © Drew Campbell [12]*

ότι έχουμε το σενάριο του Σχήματος 2.4, ο χρόνος dt για τα δύο πρώτα καρέ ορίζεται ως:

$$dt = t2 - t1$$

Με την χρήση του χρόνου dt πετυχαίνουμε ανεξαρτητοποίηση του προγράμματος από τις διαφορές στην απόδοση του κάθε hardware.

2.4.3 State Pattern

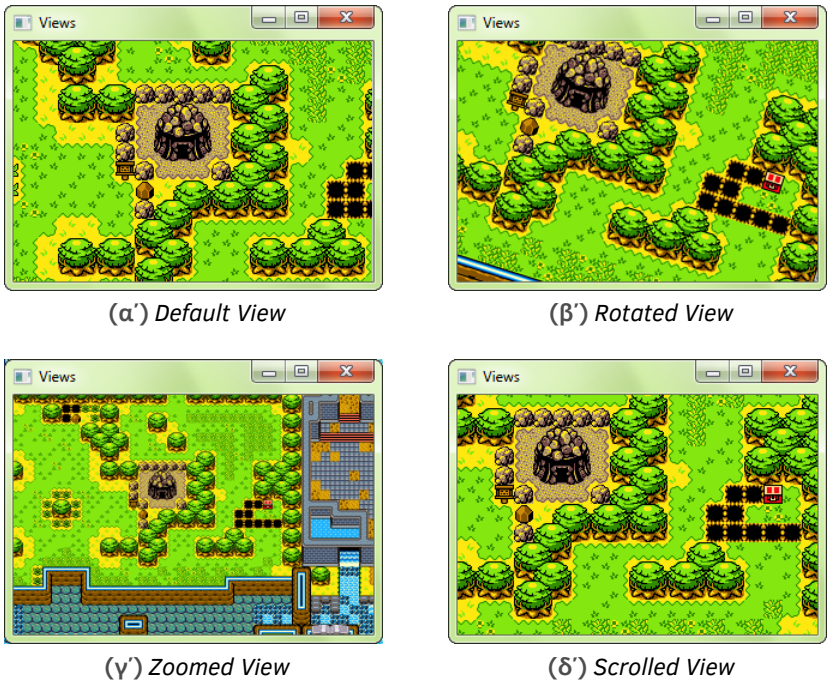
Το State Pattern επιτρέπει σε ένα αντικείμενο να αλλάξει την συμπεριφορά του όταν αλλάξει κάτι στην εσωτερική του κατάσταση. Είναι ένας πιο κομψός τρόπος να αλλάζουμε την συμπεριφορά του αντικειμένου κατά την διάρκεια εκτέλεσης του προγράμματος, αποφεύγοντας έτσι να γράφουμε περίπλοκες συνθήκες για τον έλεγχο της κατάστασης του αντικειμένου προσδίδοντας επίσης καλύτερη οργάνωση στον κώδικα του προγράμματος.

2.5 Κάμερα

Στα παιχνίδια ένα σύνηθες φαινόμενο ο κόσμος να είναι μεγαλύτερος από το ίδιο το παράθυρο. Αυτό που βλέπουμε εμείς στην πραγματικότητα είναι ένα μικρό κομμάτι αυτού του μεγαλύτερου κόσμου. Εφόσον το παράθυρο δρα με αυτόν τον τρόπο θα πρέπει να προσδιοριστεί ο χώρος στον οποίο θα δείχνει το παράθυρο, όπως επίσης το που και πως θα απεικονίζεται αυτή η περιοχή. Αυτά τα δύο πράγματα τυγχάνει να είναι και τα βασικά χαρακτηριστικά των Views της SFML.

2.5.1 View

Το View αναπαριστά την ιδέα της κάμερας δύο διαστάσεων. Ορίζει ένα ορθογώνιο μέσα στον χώρο που θέτει ποιο μέρος του κόσμου θα απεικονίζεται, δηλαδή το υποσύνολο του κόσμου που θα βλέπει ο χρήστης. Ακόμη, το View υποστηρίζει κάποιες χρήσιμες λειτουργίες όπως το ζουμ, η κύλιση και η περιστροφή.



(α΄) *Default View* (β΄) *Rotated View*

(γ΄) *Zoomed View* (δ΄) *Scrolled View*

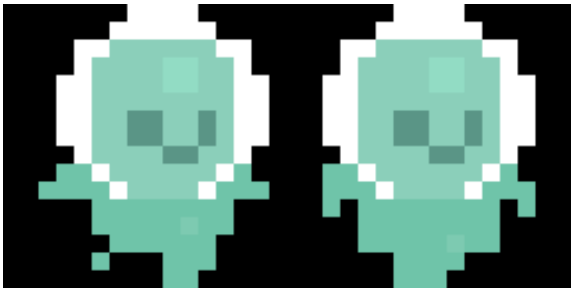
Σχήμα 2.5: *Λειτουργίες ενός View, © Laurent Gomila[16]*

2.5.2 Viewport

Το Viewport είναι ένα ορθογώνιο το οποίο ορίζει ποια περιοχή του παραθύρου θα χρησιμοποιηθεί. Οι συντεταγμένες του είναι κανονικοποιημένες, δηλαδή αν ένα Viewport ξεκινάει από το 0,0 και φτάνει έως το 1,1 δηλώνει πως θα χρησιμοποιηθεί ολόκληρο το παράθυρο.

2.6 Σχεδιοκίνηση

Το Animation είναι μία μέθοδος φωτογράφησης διαδοχικών εικόνων ώστε να δημιουργηθεί η ψευδαίσθηση της κίνησης. Επειδή τα μάτια μας μπορούν να διατηρήσουν μια εικόνα μόνο για περίπου 1/10 του δευτερολέπτου, όταν πολλές εικόνες εμφανίζονται διαδοχικά, ο εγκέφαλος τις συνδυάζει σε μία κινούμενη εικόνα. Στην Εικόνα 2.6 φαίνεται η σχεδιοκίνηση κίνησης του παίκτη που αποτελείτε από δύο καρέ.



Σχήμα 2.6: *Σχεδιοκίνηση του κυρίως χαρακτήρα καθώς περπατάει*

Κεφάλαιο 3

Υλοποίηση

3.1 Σημείο εισόδου εφαρμογής

```
int main()
{
    Game game;
    game.Run();

    return 0;
}
```

Η μοναδική δουλειά της συνάρτησης main είναι να καλεί την συνάρτηση Run() της κλάσης Game για να προχωρήσει το πρόγραμμα στο game loop.

3.2 Κλάση Game

Η σημαντικότερη κλάση η οποία ρυθμίζει την ροή του προγράμματος. Στον κατασκευαστή της γίνεται η αρχικοποίηση του παραθύρου, φορτώνονται οι πόροι στην μνήμη(εικόνες κλπ.) και δημιουργούνται οι καταστάσεις της εφαρμογής.

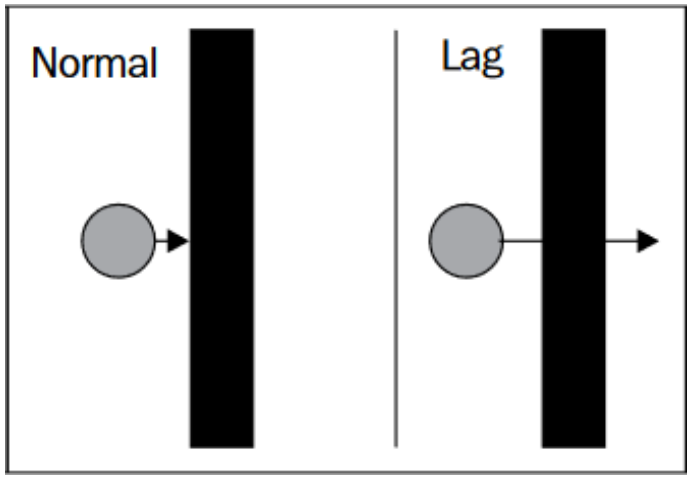
Στην συνέχεια η ροή του προγράμματος περνάει στο game loop. Εκεί υπολογίζεται ο χρόνος dt που είναι απαραίτητος καθόλη τη διάρκεια της εκτέλεσης του προγράμματος. Με βάση αυτό που προσπαθούμε να πετύχουμε:

- HandleEvents(): γίνεται έλεγχος για την ύπαρξη γεγονότων από τον χρήστη ώστε να γίνει η ανάλογη διαχείριση τους
- Update(): ενημερώνει την κατάσταση ολόκληρου του παιχνιδιού
- Render(): διαχειρίζεται την απεικόνιση των οντοτήτων στην οθόνη

Εδώ είναι σημαντικό να τονιστεί πως στην συνάρτηση που είναι υπεύθυνη για την ενημέρωση της εφαρμογής, την Update(), δίνεται ως όρισμα μία σταθερά ονόματι TimePerFrame και όχι ο χρόνος dt. Αυτό γίνεται γιατί οι μεγάλες διακυμάνσεις του χρόνου dt που μπορεί να υπάρξουν θα επηρεάσουν αρνητικά την εκτέλεση του προγράμματος. Για παράδειγμα όπως φαίνεται στην Εικόνα3.1, αν υποθέσουμε οτι ο κώδικας φυσικής

που μετακινεί τα αντικείμενα δεχόταν μεταβαλλόμενο αριθμό dt και αυτός ο αριθμός στο τωρινό καρέ ήταν για κάποιο λόγο κατά πολύ μεγαλύτερος σε σχέση με το προηγούμενο καρέ, τότε θα είχαμε το φαινόμενο όπου το ένα σώμα θα περνούσε μέσα από το άλλο ενώ κανονικά θα έπρεπε να υπάρξει σύγκρουση.

```
void Game::Run()
{
    ...

    while (m_window.isOpen())
    {
        ...

            if(!m_IsPaused)
                Update(TimePerFrame);

        ...
    }
}
```



Σχήμα 3.1: *Αρνητικές επιπτώσεις της διακύμανσης του χρόνου dt στον κώδικα της φυσικής, © Artur Moreira, Henrik Vogelius Hansson, and Jan Haller [1]*

3.3 Κλάση World

Η ανάγκη για την ύπαρξη μίας κλάσης η οποία θα περιέχει και θα διαχειρίζεται διάφορα δεδομένα και συναρτήσεις γέννησε την κλάση World. Αυτή η κλάση είναι η καρδιά της εφαρμογής. Περιγράφει την “σκηνή“ του παιχνιδιού, δηλαδή περιέχει οτιδήποτε έχει να κάνει με το παιχνίδι, όπως τον κόσμο της φυσικής, τον κατάλογο με τις οντότητες, την κάμερα όπως επίσης είναι υπεύθυνη και για την διαχείριση του HUD(πχ. μετρητής νομισμάτων), τον χρόνο που έχει απομείνει, αυτή είναι που δημιουργεί και καταστρέφει ό,τι χρειάζεται να φορτωθεί στην μνήμη για να μπορέσει ο χρήστης να παίξει το παιχνίδι.

3.4 Κλάση Context

Μία σημαντική βοηθητική κλάση για την λειτουργία του προγράμματος που περιέχει μεταβλητές οι οποίες χρειάζεται να είναι προσβάσιμες από ολόκληρο το πρόγραμμα.

```
struct Context
{
    ...

    sf::RenderWindow* window;
    FontHolder* fonts;
    TextureHolder* textures;
    MusicPlayer* music;
    SoundEffectPlayer* sounds;
    bool isPaused;
    std::unordered_map<entt::entity, b2Body*> enttToBody;
};
```

Περιέχει δείκτη προς το παράθυρο της εφαρμογής, δείκτη προς τα hashmaps που διαχειρίζονται του πόρους, μία bool μεταβλητή που γίνεται true όταν η εφαρμογή περάσει στην κατάσταση παύσης και τέλος μια βοηθητική δομή που αντιστοιχίζει τα entity ids με τα σώματα τους για βελτίωση της απόδοσης ώστε να μην χρειάζεται σε κάθε καρέ να ψάχνουμε ολόκληρο τον κατάλογο οντοτήτων.

3.5 Καταστάσεις και στοίβα καταστάσεων

Η διαχείριση των καταστάσεων της εφαρμογής γίνεται με την βοήθεια δύο κλάσεων, της StateStack και της αφηρημένης κλάσης State.

3.5.1 StateStack

Η StateStack όπως υποδηλώνει και το όνομα της δρα σαν στοίβα η οποία δέχεται αντικείμενα της κλάσης State. Υποστηρίζει τις κλασσικές λειτουργίες της στοίβας όπως push, pop, clear και isEmpty. Επίσης περιέχει την πρότυπη συνάρτηση registerState(...) η οποία χρησιμοποιείται για την δημιουργία των αντικειμένων της κλάσης State.

```
template<typename T>
void StateStack::registerState(States::ID stateID)
{
    m_Factories[stateID] = [this]()
    {
        return State::Ptr(new T(*this, m_Context));
    };
}
```

Τέλος περιέχει την δομή PendingChange. Η δουλειά της είναι να ολοκληρώνει ενέργειες όπως push, pop κλπ. στο τέλος κάθε καρέ, καλώντας την βοηθητική συνάρτηση applyPendingChange(), για να αποφευχθούν προβλήματα στην λογική του προγράμματος.

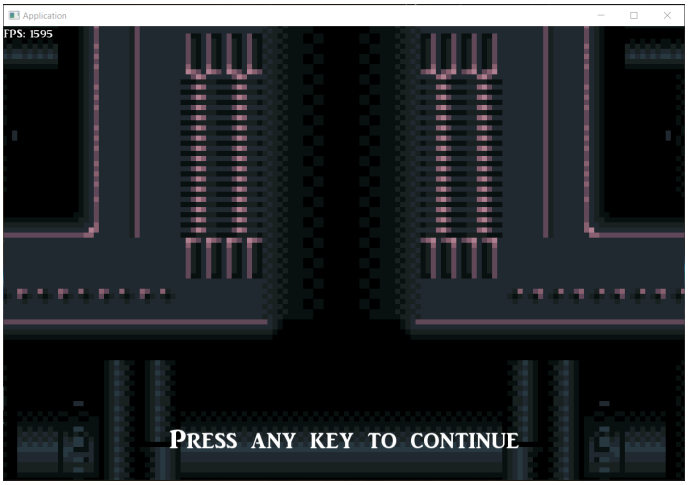
```
class StateStack : private sf::NonCopyable
{
    . . .

    void applyPendingChange();

private:
    struct PendingChange
    {
        PendingChange(Action action, States::ID stateID = States::ID::None)
            ;

        Action action;
        States::ID stateID;
    };
    . . .
};
```

3.5.2 State

Η κλάση State μοντελοποιεί την ιδέα της κατάστασης της εφαρμογής. Περιλαμβάνει τις αμιγώς εικονικές συναρτήσεις update(), η render() και η handleEvents() για την καλύτερη οργάνωση του κώδικα. Ακόμα, έχει έναν δείκτη που δείχνει στην στοίβα καταστάσεων για να μπορεί να αλληλεπιδρά με αυτήν και τις βοηθητικές συναρτήσεις. Τέλος περιέχει και ένα αντικείμενο της κλάσης Context για να έχει πρόσβαση στα δεδομένα που είναι κοινά και απαραίτητα σε όλες τις καταστάσεις.

Title Screen State

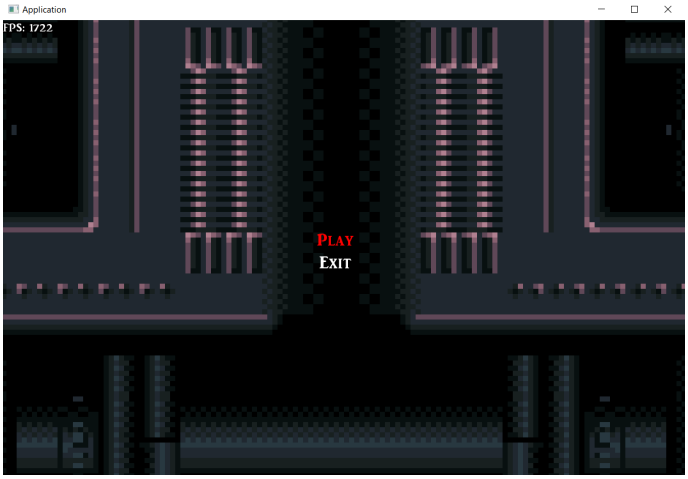
Κατά την έναρξη της εφαρμογής, η αρχική κατάσταση της εφαρμογής είναι το Title State(βλ. Εικόνα3.2). Αυτή η κατάσταση περιμένει να πατηθεί οποιοδήποτε κουμπί από το πληκτρολόγιο για να προχωρήσει στο κυρίως μενού.



Σχήμα 3.2: *Title Screen*

Menu State

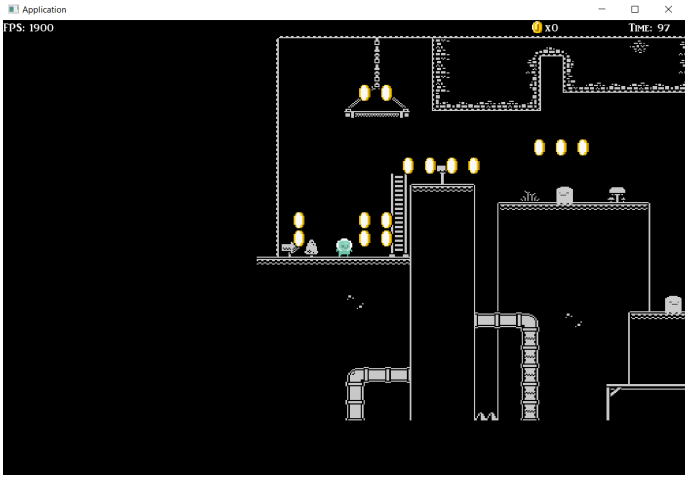
Έπειτα, μέσα στην κατάσταση μενού, δίνει στον χρήστη επιλογή να τερματήσει την εφαρμογή πατώντας “Exit“ ή να ξεκινήσει το παιχνίδι πατώντας “Play“. Αν επιλεχθεί το “Play“, τότε η κατάσταση θα μεταβεί στο Game State(βλ. Εικόνα 3.3).



Σχήμα 3.3: *Main Menu*

Game State

Μέσα στην κατάσταση παιχνιδιού ο χρήστης έχει την δυνατότητα να μεταβεί στην Pause State πατώντας το κουμπί esc και μετά πατώντας το κουμπί backspace να γυρίσει πίσω στην κατάσταση μενού ή πατώντας το esc για άλλη μια φορά να βγει από την κατάσταση παύσης και να γυρίσει πίσω στην κατάσταση παιχνιδιού(βλ. Εικόνα 3.4).

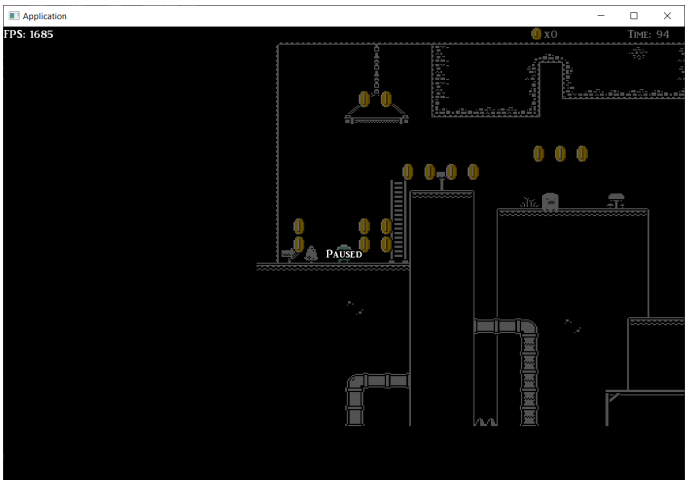


Σχήμα 3.4: *Game Screen*

Pause State

Δεν υπάρχει κάτι ιδιαίτερο σε αυτήν την κατάσταση, το μόνο που κάνει είναι να σταματάει το παιχνίδι μέχρι ο χρήστης να βγει από αυτήν την κατάσταση ή να γυρίσει

πίσω στο κυρίως μενού(βλ. Εικόνα 3.5).



Σχήμα 3.5: *Pause Screen*

3.6 Components

Σε αυτό το κομμάτι θα παρουσιαστούν τα components τα οποία μπορούν να έχουν όλες οι οντότητες τις εφαρμογής.

C_Animation

Κάθε animation component έχει από ένα αντικείμενο της κλάσης AnimatedSprite. Η κλάση αυτή περιέχει διάφορες χρήσιμες συναρτήσεις για την διαχείριση των textures που είναι φορτωμένα στην μνήμη ώστε να πετύχει το αποτέλεσμα της σχεδιοκίνησης.

C_Rigidbody

Το Rigidbody Component περιέχει έναν δείκτη προς αντικείμενο b2Body της βιβλιοθήκης Box2D. Η προσομοίωση του κόσμου φυσικής γίνεται με σώματα αυτού του τύπου τα οποία πρέπει να είναι αναγκαστικά δείκτες εξαιτίας του API της Box2D.

C_Camera

Περιέχει πληροφορίες για το view, τον στόχο τον οποίον θα ακολουθεί η κάμερα και το προκαθορισμένο view που δημιουργεί το παράθυρο κατά την δημιουργία του.

C_Raycast

Περιέχει πληροφορίες σχετικά το σώμα που χρησιμοποιεί η προσομοίωση της φυσικής, όπως τις γωνίες του σχήματος του ώστε να μπορούν να γίνουν οι υπολογισμοί για τη θέση που θα ξεκινούν οι ακτίνες, τον αριθμό των ακτίνων και την απόσταση

μεταξύ τους, όπως επίσης και πληροφορίες που θα προκύψουν αν αυτές οι ακτίνες συγκρουστούν με άλλα σώματα μέσα στην κόσμο της φυσικής.

C_Tag

Είναι μια σειρά από αναγνωριστικά Components. Η μόνη χρήση τους είναι να δίνουν έναν τρόπο στο πρόγραμμα να ξεχωρίζει τις οντότητες μεταξύ τους όποτε χρειαστεί όπως για παράδειγμα μέσα στα συστήματα που τις διαχειρίζονται.

C_Tilemap

Περιέχει πληροφορίες για το tilemap, όπως το path μέσα στο σύστημα αρχείων που βρίσκεται το αρχείο .tmx που περιέχει τις πληροφορίες που περιγράφουν το πως πρέπει να απεικονιστεί το tilemap στο παράθυρο.

3.7 Systems

3.7.1 System Manager

Η κλάση System Manager είναι αυτή που μεταχειρίζεται όλα τα εγγεγραμμένα συστήματα του προγράμματος που βρίσκονται σε ένα vector από δείκτες σε συστήματα. Ακόμα, διαθέτει κάποιες βοηθητικές συναρτήσεις για την αρχικοποίηση, την εισαγωγή και διαγραφή συστημάτων σε αυτό το vector. Επίσης περιέχει τις συναρτήσεις update, handleEvents και draw/render για να κρατηθεί ο κώδικας όσο πιο καθαρός και συντηρήσιμος γίνεται.

```
class SystemManager
{
public:
    SystemManager() = default;

    void update(sf::Time dt);
    void handleEvents(const sf::Event& event);
    void draw();

    void addSystem(std::unique_ptr<BaseSystem> sys);
    void deleteAllSystems();

    void initSystems();

private:
    std::vector<std::unique_ptr<BaseSystem>> m_Systems{};
```

Animation System

Η δουλειά του συστήματος σχεδιοκίνησης είναι να ενημερώνει όλα τα Animation components των οντοτήτων ώστε η λειτουργία της σχεδιοκίνησης να ανταποκρίνεται

στις ενέργειες τους μέσα στον χώρο.

Camera System

Αυτό το σύστημα είναι υπεύθυνο για ό,τι αφορά την κάμερα. Διαχειρίζεται το view του παραθύρου, το μετακινεί ανάλογα την θέση του παίκτη μέσα στον κόσμο ώστε αυτός να βρίσκεται πάντα στο κέντρο του και διαθέτει ακόμα κάποιες ειδικές λειτουργίες όπως ζουμ και μετακίνηση της μέσα στον κόσμο.

Collision System

Το συγκεκριμένο σύστημα δίνει "μάτια" στις οντότητες του κόσμου του παιχνιδιού. Είναι αυτό που καλεί την συνάρτηση Step() της Box2D για να ενημερωθεί ο κόσμος φυσικής και πετάει τις ακτίνες από τα σώματα που χρειάζεται να αντιλαμβάνονται το περιβάλλον τους.

Move System

Η δουλειά αυτού του συστήματος είναι να ενημερώνει την θέση όλων των οντοτήτων που έχουν το Rigidbody Component. Εξαιτίας της ιδιαιτερότητας της βιβλιοθήκης Box2D, η οποία χρησιμοποιεί τις μονάδες μέτρησης MKS(meters, kilograms, and seconds) για την σωστή και ακριβή ενημέρωση των οντοτήτων πρέπει οι τιμές που επιστρέφονται από την Box2D να μετατρέπονται στο σύστημα μονάδων μέτρησης που χρησιμοποιεί η SFML, πχ. να μετατραπούν τα ακτίνια σε μοίρες, τα μέτρα σε εικονοστοιχεία κλπ. Επίσης μετακινεί και το Animation Component, εάν έχει, της οντότητας έτσι ώστε να συμβαδίζει με το σώμα που είναι δεμένο.

Render System

Όπως υποδηλώνει και το όνομα του είναι υπεύθυνο για την απεικόνιση των οντοτήτων που πρέπει να ζωγραφιστούν όπως τα animations και το tilemap. Ακόμα αυτό το σύστημα προσφέρει με το πάτημα του κουμπιού Num1 πληροφορίες που μπορούν να χρησιμοποιηθούν για αποσφαλμάτωση και με το πάτημα του κουμπιού Num2 πληροφορίες για τον κόσμο προσωμοίωσης της φυσικής από την Box2D.

Player Controller System

Το σύστημα υπεύθυνο για την διαχείριση του παίκτη. Ελέγχει την οντότητα με το PlayerTag Component, δηλαδή τον κυρίως χαρακτήρα, και ρυθμίζει τα πάντα για αυτήν όπως τις ενέργειες που μπορεί να εκτελέσει μέσα στον κόσμο, θέτει την ταχύτητα του μέσα στον κόσμο αλλά και την μέγιστη ταχύτητα που μπορεί να έχει, το ποιο animation πρέπει να παίξει όταν ο παίκτης κινείται ή όταν κάνει άλμα κλπ. Τέλος χρησιμοποιώντας τις πληροφορίες που παρέχονται από το Raycast Component μπορεί να αλληλεπιδρά με άλλες οντότητες όπως οι εχθροί ανιχνεύοντας πότε ακούμπησε ο παίκτης έναν εχθρό από πάνω ώστε να τον καταστρέψει και να πάρει μία ώθηση ταχύτητας προς τα πάνω.

Enemy Controller System

Το σύστημα υπεύθυνο για την διαχείριση των εχθρών. Μετατρέπει τις οντότητες με το EnemyTag Component σε “ευφυείς“ οντότητες. Δηλαδή ανιχνεύει χρησιμοποιώντας τις πληροφορίες που παρέχονται από το Raycast Component πότε αυτές οι οντότητες είναι έτοιμες να πέσουν από την πλατφόρμα πάνω στην οποία βρίσκονται και αλλάζει το velocity vector τους προς την αντίθετη κατεύθυνση ώστε να τις κρατήσει εκεί πάνω. Τέλος από αυτό το σύστημα ανιχνεύεται αν ο παίκτης ακούμπησε έναν εχθρό από δεξιά, αριστερά ή από κάτω ώστε να τερματίσει το παιχνίδι με GameOver για τον παίκτη.

Tilemap System

Ένα από τα σημαντικότερα συστήματα καθώς αυτό είναι που φορτώνει το tilemap του παιχνιδιού στην μνήμη, αναλύοντας με την βιβλιοθήκη tmxlite το αρχείο που βγήκε από την εφαρμογή σχεδίασης tilemap ονόματι Tiled και περιέχει τις απαραίτητες πληροφορίες για το που και πως να γίνει η απεικόνιση των tiles, όπως και πληροφορίες σαν την αρχική θέση του παίκτη, των εχθρών, των αντικειμένων που μπορούν να συλλεχθούν αλλά και τα σχήματα που θα χρησιμοποιήσει ο κόσμος της φυσικής για τους υπολογισμούς του.

3.8 Διαχείρηση πόρων

Η διαχείριση όλων των πόρων που χρησιμοποιεί η εφαρμογή, κατ εξαίρεση των ηχητικών εφέ και της μουσικής που προϋποθέτουν ειδική μεταχείριση κατά το φόρτωμά τους στην μνήμη, γίνεται από την πρότυπη κλάση ResourceHolder. Η ResourceHolder δέχεται δύο πρότυπες παραμέτρους:

- **Resource:** Ο τύπος του πόρου, πχ. ID::Texture.
- **Identifier:** Το αναγνωριστικό ID για την πρόσβαση στον πόρο. Ο τύπος είναι συνήθως ένα enum όμως δεν είναι και απαραίτητο, μπορεί να χρησιμοποιηθεί οποιοσδήποτε τύπος έχει υπερφορτωμένο τον τελεστή <, πχ. std::string

Αποθηκεύει τους πόρους που έχουν φορτωθεί σε ένα hash map με key το Identifier και value ένα unique_ptr προς το Resource.

```
std::map<Identifier, std::unique_ptr<Resource>> m_ResourceMap;
```

Για την καλύτερη διαχείριση του κώδικα χρησιμοποιήθηκαν ψευδώνυμα που αναπαριστούν τους διάφορους πόρους που μπορούμε να φορτώσουμε στην μνήμη και γίνεται με τον τρόπο που φαίνεται στο παρακάτω κομμάτι του κώδικα:

```
using TextureHolder = ResourceHolder<sf::Texture, Textures::ID>;
using FontHolder = ResourceHolder<sf::Font, Fonts::ID>;
using SoundBufferHolder = ResourceHolder<sf::SoundBuffer, Sounds::ID>;
```

3.9 Μουσική και Ηχητικά εφέ

3.9.1 Music

Η SFML χρησιμοποιεί την κλάση sf::Music για την μεταχείριση της μουσικής. Αυτή η κλάση συμπεριφέρεται διαφορετικά σε σχέση με τις υπόλοιπες κλάσεις που μεταχειρίζονται πόρους και αυτό διότι τα μουσικά κομμάτια είναι συνήθως μεγάλα σε έκταση οπότε αν φορτωθούν ολόκληρα θα πιάνουν πολύ χώρο μέσα στην μνήμη. Έτσι η κλάση sf::Music τα μεταδίδει κατευθείαν από τον σκληρό δίσκο. Αυτό σημαίνει πως ένα μικρό μέρος της μουσικής θα είναι φορτωμένο την φορά στην μνήμη, με τα καινούργια να φορτώνονται όταν αυτό φτάσει στο τέλος του. Αυτό σημαίνει πως το αρχείο από το οποίο θα μεταδίδεται η μουσική θα πρέπει να είναι μόνιμα διαθέσιμο στο πρόγραμμα μέχρι να σταματήσει. Όλη η διαχείριση της μουσικής γίνεται με την κλάση MusicPlayer. Ως αποτέλεσμα της ιδιαιτερότητας που αναφέρθηκε δεν χρησιμοποιείται η κλάση ResourceHolder αλλά όλη η δουλειά γίνεται μέσα στην ίδια την κλάση MusicPlayer. Στον contructor αρχικοποιεί και αποθηκεύει τα κομμάτια σε ένα hash map με key ένα αναγνωριστικό ID και value το path προς το αρχείο της μουσικής.

```
m_Filenames[Music::ID::MenuTheme] = "../res/Music/Menu_Select.ogg";
m_Filenames[Music::ID::GameTheme] = "../res/Music/Stage 1.ogg";
```

Επίσης προσφέρει μερικές βοηθητικές συναρτήσεις για περισσότερη ευκολία χρήσης, όπως η play() που ξεκινά να μεταδίδει το κομμάτι, η stop() που σταματάει την μετάδοση και η setPaused() που κάνει παύση την μετάδοση.

```
void MusicPlayer::play(Music::ID id)
{
    auto filename = m_Filenames.at(id);

    if (!m_Music.openFromFile(filename))
    {
        throw std::runtime_error("Couldn't load " + filename + ".");
    }

    m_Music.setVolume(m_Volume);
    m_Music.setLoop(true);
    m_Music.play();
}

void MusicPlayer::stop()
{
    m_Music.stop();
}

void MusicPlayer::setPaused(bool paused)
{
    if (paused)
        m_Music.pause();
    else
        m_Music.play();
```

```
}
```

3.9.2 SoundBuffer και Sound

Η SFML προσφέρει δύο κλάσεις για την μεταχείριση των ήχων που είναι μικροί σε μέγεθος για να χωράνε με άνεση στην μνήμη και να :

- **sf::SoundBuffer:** Χρησιμοποιείται για την φόρτωση ηχητικών δειγμάτων στην μνήμη που ορίζουν έναν ήχο. Το sound buffer αποθηκεύει τα δεδομένα ενός ήχου, τα οποία είναι ένας πίνακας από δείγματα ήχου. Ένα ηχητικό δείγμα αποτελείται από 16bit ακέραιους, οι οποίοι ορίζουν το εύρος της κυματομορφής του ήχου σε μία συγκεκριμένη χρονική στιγμή. Το sound buffer επιτρέπει την πρόσβαση και την μετατροπή των δειγμάτων αλλά από μόνο του δεν είναι ικανό να παίξει έναν ήχο. Οι τύποι αρχείων που υποστηρίζει είναι τα .OGG, .WAV και .AIFF ενώ το πρότυπο MP3 δεν υποστηρίζεται λόγο της περιοριστικής άδειας που διαθέτει.
- **sf::Sound:** Είναι η κλάση που χρησιμοποιεί η SFML για να παίζει ήχους. Για να μπορέσει αυτή η κλάση να παίξει έναν ήχο πρέπει να την προμηθεύσουμε με ένα SoundBuffer και έπειτα να καλέσουμε την συνάρτηση μέλος της Sound play(). Παραπάνω από ένας ήχοι γίνεται να παίζουν ταυτόχρονα όπως επίσης πολλαπλά Sound μπορούν να παίζουν έναν ήχο χρησιμοποιόντας το ίδιο SoundBuffer.

Στο παιχνίδι, τα πάντα τακτοποιεί η κλάση SoundEffectPlayer. Στον constructor αρχικοποιεί και αποθηκεύει τα ηχητικά δείγματα κάνοντας χρήση της κλάσης ResourceHolder:

```
m_SoundBuffers.load(Sounds::ID::CoinPickup, "../res/SoundEffects/coin.wav")
m_SoundBuffers.load(Sounds::ID::Pause, "../res/SoundEffects/pause.wav");
m_SoundBuffers.load(Sounds::ID::Squash, "../res/SoundEffects/squash.wav");
```

Όσων αναφορά τους ήχους, γίνεται χρήση της std::list για την αποθήκευση τους και χρησιμοποιείται κυρίως για να γνωρίζει το πρόγραμμα ποιοι ήχοι παίζουν σε κάποια συγκεκριμένη χρονική στιγμή.

```
SoundBufferHolder     m_SoundBuffers;
std::list<sf::Sound> m_Sounds;
```

3.10 Βοηθητικές κλάσεις

Σε αυτό το μέρος θα παρουσιαστούν κλάσεις που είναι προαιρετικές για την εκτέλεση του προγράμματος και ο λόγος ύπαρξης τους είναι να κάνουν την ζωή του προγραμματιστή πιο εύκολη.

3.10.1 Συμπληρωματικές κλάσεις για την Box2D

Όταν τα σώματα κινούνται μέσα στην κόσμο της φυσικής και συγκρούονται μεταξύ τους, η Box2D θα αναλάβει να ανιχνεύσει την σύγκρουση και να την διαχειριστεί κατάλληλα χωρίς εμείς να κάνουμε τίποτα. Το θέμα είναι όμως ότι στα παιχνίδια συνήθως όταν προκύπτουν συγκρούσεις πρέπει να γίνονται συγκεκριμένες ενέργειες ανάλογα με την περίσταση, πχ. όταν ο παίκτης συγκρούεται με έναν εχθρό πρέπει το πρόγραμμα να αποφασίσει εάν το παιχνίδι συνεχίζεται ή τελειώνει. Η βιβλιοθήκη Box2D μας δίνει την δυνατότητα να αντλήσουμε πληροφορίες σχετικά με τα σώματα που υπάρχουν μέσα στον κόσμο την προσομοίωσης. Αυτό το πετυχαίνουμε με την χρήση των λεγόμενων callback συναρτήσεων. Η διαδικασία είναι σχετικά απλή, κληρονομούμε από αφηρημένες κλάσεις που είναι μέρος της βιβλιοθήκης και καλούνται για παράδειγμα κάθε φορά που θα προκύψει μία σύγκρουση, κάνοντας την ροή του προγράμματος να περνά μέσα από την callback συνάρτησή μας.

ContactListener

Η συγκεκριμένη κλάση θα μας ενημερώνει για κάθε σύγκρουση που θα προκύπτει ανάμεσα σε δύο σώματα. Στο περίπτωσή του δικού μας προγράμματος μας ενήμερώνει όταν ο παίκτης έχει ακουμπήσει νομίσματα και μονόδρομες πλατφόρμες.

QueryCallback

Πολλές φορές σε παιχνίδια θέλουμε να μάθουμε αν υπάρχουν σώματα σε ένα σημείο μέσα στον κόσμο. Με την QueryCallback παίρνουμε όλα τα σώματα από μία περιοχή που έχουμε κάνει aabb query. Ο έλεγχος θα περάσει στην κλάση μας για κάθε σώμα που θα βρίσκεται μέσα στο aabb που δημιουργήσαμε σε εκείνο το σημείο.

RaycastCallback

Δίνοντας τον αρχικό και τον τελικό προορισμό μπορούμε με την βοήθεια της Box2D να κάνουμε μία προβολή ακτίνας για την ανίχνευση σωμάτων. Θα επιστραφούν όσα σώματα χτύπησε η ακτίνα.

FixtureUserData

Η Box2D μας δίνει την δυνατότητα να αποθηκεύσουμε μέσα στο σώμα ένα αντικείμενο που θα περιέχει κάποιες επιπλέον χρήσιμες πληροφορίες δικής μας κατασκευής. Αυτό γίνεται με την χρήση του δείκτη userdata που εμπεριέχεται μέσα στην βιβλιοθήκη.

SFMLDebugDraw

Αυτή η κλάση είναι υπεύθυνη για την απεικόνιση πληροφοριών για τα σώματα που υπάρχουν μέσα στον κόσμο της φυσικής με σκοπό την βοήθεια στην αποσφαλμάτωση.

3.10.2 Λοιπές

Math

Περιέχει χρήσιμες μαθηματικές συναρτήσεις για περίπλοκους υπολιγισμούς.

Body Creator

Με πληροφορίες που αντλεί απο το tilemap, δημιουργεί όλα τα αντικείμενα που υπάρχουν στον κόσμο του παιχνιδιού όπως η αρχική θέση του παίκτη, τα σχήματα των αντικειμένων φυσικής που θα χρησιμοποιήσει η Box2D κλπ.

Timer

Προσομοιώνει ένα ρολόι αντίστροφης μέτρησης. Η κύρια χρήση του είναι μετρά αντίστροφα και αν το ρολόι φτάσει στο μηδέν το παιχνίδι να τερματήσει.

Utility

Περιέχει διάφορες χρήσιμες συναρτήσεις και όχι μόνο που χρησιμοποιούνται συχνά σε πολλά μέρη μέσα στο πρόγραμμα.

3.11 Εργαλεία

Στην συνέχεια γίνεται μία γρήγορη ανασκόπηση των εργαλείων που χρησιμοποιήθηκαν για να έρθει εις πέρας αυτή η εργασία.

3.11.1 CMake

Σχήμα 3.6: *Λογότυπο CMake, Πηγή: https://cmake.org/files/logos/. Ημερομηνία πρόσβασης: 21-4-2021*

Το CMake είναι μία οικογένεια διαπλατφορμικών εργαλείων ανοιχτού κώδικα που έχουν σχεδιαστεί για την κατασκευή λογισμικού. Χρησιμοποιείται για να ελέγχει την διαδικασία της μεταγλώττισης.

Σχήμα 3.7: *Λογότυπο entt, Πηγή: https://github.com/skypjack/entt. Ημερομηνία πρόσβασης: 21-4-2021*

3.11.2 entt

Το entt είναι ένα header αρχείο ανοιχτού κώδικα. Προσφέρει το Entity Component System που χρησιμοποιείται για την κατασκευή παιχνιδιών και όχι μόνο γραμμένη με την γλώσσα C++.

3.11.3 SFML

Σχήμα 3.8: *Λογότυπο SFML, Πηγή: https://www.sfml-dev.org/download/goodies/. Ημερομηνία πρόσβασης: 21-4-2021*

Η SFML παρέχει μια απλή διεπαφή για διάφορα στοιχεία του υπολογιστή, για να διευκολύνει την ανάπτυξη παιχνιδιών και εφαρμογών πολυμέσων. Αποτελείται από πέντε ενότητες: το System, το Window, το Graphics, το Audio και το Network. Ακόμα, με την SFML, η εφαρμογή μας μπορεί να μεταγλωττιστεί για Window, MacOS και Linux.

3.11.4 Box2D

Σχήμα 3.9: *Λογότυπο Box2D, Πηγή: https://en.wikipedia.org/wiki/File:Box2D_logo.svg. Ημερομηνία πρόσβασης: 21-4-2021*

Η Box2D είναι μία βιβλιοθήκη προσομοίωσης δισδιάστατων σωμάτων για παιχνίδια. Μπορεί να χρησιμοποιηθεί για να κάνει να αντικείμενα να κινούνται με ρεαλιστικό τρόπο και να κάνει τα παιχνίδια πιο διαδραστικά.

3.11.5 spdlog

Βιβλιοθήκη που χρησιμοποιείται για logging. Στην συγκεκριμένη εργασία χρησιμοποιήθηκε μόνο για αποσφαλμάτωση

3.11.6 fmt

Βιβλιοθήκη μορφοποίησης κειμένου ανοιχτού κώδικα που προσφέρει μία γρήγορη και ασφαλής εναλλακτική για το iostream.

3.11.7 Tiled

Σχήμα 3.10: *Λογότυπο Tiled, Πηγή: https://www.mapeditor.org/. Ημερομηνία πρόσβασης: 21-4-2021*

Το Tiled map editor είναι ένα εργαλείο ανοιχτού κώδικα για την δημιουργία tilemap.

3.11.8 tmxlite

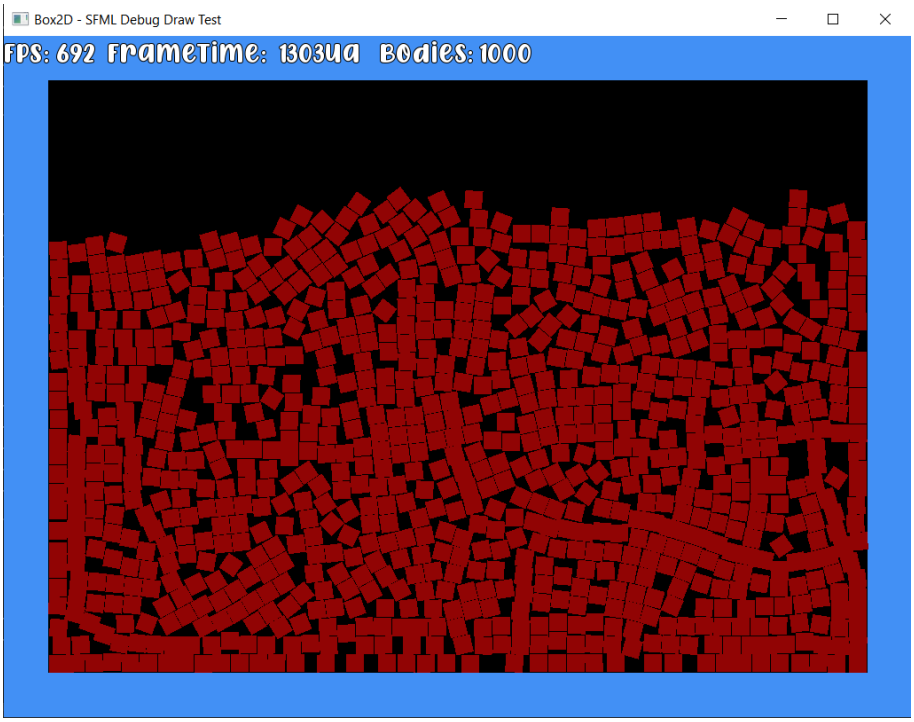
Μία βιβλιοθήκη ανάλυσης .tmx αρχείων tilemap, που παράγει το Tiled map editor.

Κεφάλαιο 4

Πείραμα

Για της ανάγκες της εργασίας, διότι δεν ήταν εφικτό να κατασκευαστεί ολόκληρο το παιχνίδι και με τους δύο τρόπους σχεδίασης OOP και DOP, δημιουργήθηκαν 2 μικρές εφαρμογές για την επίδειξη της διαφοράς στην απόδοση που έχουν αυτοί οι δύο σχεδιαστικοί τρόποι. Οι εφαρμογές αυτό που κάνουν είναι να δημιουργούν, με την χρήση της SFML για την απεικόνιση και της Box2D για την προσομοίωση της φυσικής, έναν αριθμό από κινούμενα ορθογώνια που περικλείονται από στατικά ορθογώνια για να μην φύγουν εκτός της οθόνης. Οι δύο εφαρμογές έχουν ακριβώς το ίδιο αποτέλεσμα και η μόνη διαφορά είναι στον τρόπο που το πετυχαίνουν.

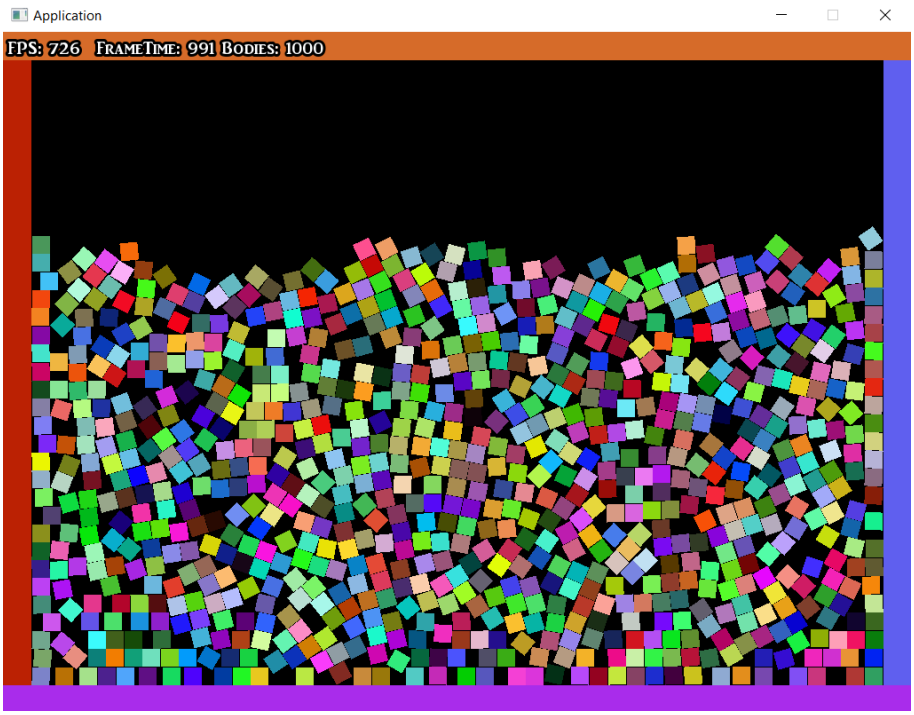
4.1 Εφαρμογή σχεδιασμένη με OOP



Σχήμα 4.1: *Παράθυρο εφαρμογής σχεδιασμένης με OOP*

Στο Σχήμα 4.1 απεικονίζεται η πρώτη εφαρμογή. Στο πάνω αριστερό μέρος τυπώνονται διάφορες χρήσιμες πληροφορίες όπως τα καρέ ανά δευτερόλεπτο, ο χρόνος dt σε microsecond και ο αριθμός των σωμάτων που παίρνουν μέρος στην προσομοίωση της φυσικής. Όλα τα σώματα έχουν το ίδιο μέγεθος και βάρος και όλα μπορούν να περιστραφούν. Τέλος, με την χρήση του ποντικιού ο χρήστης μπορεί να επιλέξει οποιοδήποτε ορθογώνιο μέσα στην οθόνη, πλην αυτών που αποτελούν το περίβλημα, και να το κυλάει μέσα στην οθόνη επηρεάζοντας έτσι και τα υπόλοιπα.

4.2 Εφαρμογή σχεδιασμένη με DOP



Σχήμα 4.2: *Παράθυρο εφαρμογής σχεδιασμένης με DOP*

Στο Σχήμα 4.2 απεικονίζεται η δεύτερη εφαρμογή. Η διαφορά στα χρώματα δεν έχει καμία απολύτως σημασία και έγινε για να είναι πιο εύκολη η διαχώρισή τους. Όπως και στην πρώτη εφαρμογή, έτσι και εδώ στο πάνω αριστερό μέρος υπάρχουν οι χρήσιμες πληροφορίες, όπως επίσης υπάρχει και η λειτουργία της επιλογής των ορθογωνίων.

4.3 Αποτελέσματα και Παρατηρήσεις

Η σύγκριση ανάμεσα στις δύο εφαρμογές έγινε με τον εξής τρόπο. Κατά την εκκίνηση τους καταγράφηκαν ο αριθμός των καρέ ανά δευτερόλεπτο και ο χρόνος dt σε microsecond για τα πρώτα 20 καρέ. Στην συνέχεια βγήκε ο μέσος όρος αυτών και αποτυπώθηκε στον παραρκάτω πίνακα. Να σημειωθεί πως οι μετρήσεις γίνανε στο λειτουργικό σύστημα Windows 10, με επεξεργαστή Intel® i7-8750H και 16GB RAM και κάρτα γραφικών την ενσωματωμένη στον επεξεργαστή Intel® UHD Graphics 630.

	OOP				DOP			
Number of objects	AVG FPS	AVERAGE FRAME TIME	MIN FPS	MAX FPS	AVG FPS	AVERAGE FRAME TIME	MIN FPS	MAX FPS
100	2149.2	598.2μs	2128	2197	2205.95	1281.1μs	2197	2257
500	1392.65	1897.6μs	1361	1430	1557.8	1785.25μs	1485	1648
1000	693.45	2194μs	641	722	793.05	2065.35μs	763	815
1500	283.4	4301.5μs	241	313	431.85	3479.4μs	395	460
2000	1.75	7578563μs	1	14	8.9	545963.9μs	1	40

Πίνακας 4.1: *Μέτρηση απόδοσης*

Όπως μπορούμε να διακρίνουμε, για λίγα αντικείμενα η απόδοση είναι σχετικά στα ίδια στάδια. Η διαφορά στην απόδοση αρχίζει να φαίνεται για τα καλά όταν τα αντικείμενα πληθαίνουν, σύμφωνα με τον πίνακα, από τα 1000 αντικείμενα και πάνω η διαφορά στον αριθμό των καρέ ανά δευτερόλεπτο και του χρόνου dt είναι αισθητή καθώς όπως μπορούμε να διακρίνουμε η διαφορά στα τα καρέ φτάνει έως και τα 150. Τέλος, κάτι που παρατηρήθηκε είναι ότι στην εφαρμογή με το OOP ο αριθμός fps είχε πολλές διακυμάνσεις σε αντίθεση με την εφαρμογή με το DOP όπου φαινόταν να είναι σχετικά πολύ πιο σταθερός.

Ο ολοκληρωμένος κώδικας για αυτά τα δύο προγράμματα βρίσκεται στο παράρτημα Α.

Κεφάλαιο 5

Συμπεράσματα και Μελλοντικές επεκτάσεις

Η χρήση του σχεδιασμού DOP σίγουρα μας προσφέρει αρκετά πλεονεκτήματα. Όμως αυτό δεν σημαίνει πως είναι η καλύτερη επιλογή για να χρησιμοποιηθεί παντού. Οι χρήσεις του για την ώρα είναι περιορισμένες, συγκεκριμένα στον τομέα της κατασκευής μηχανών γραφικών και βιντεοπαιχνιδιών. Σίγουρα ο OOP παραμένει η πιο διαδεδομένη σχεδίαση λόγω του ότι για τους περισσότερους είναι πιο εύκολη στην χρήση και καλύτερη επιλογή για την πλειοψηφία των προβλημάτων, όπως για παράδειγμα η ανάπτυξη συστήματος Graphical User Interface (GUI), όπου η χρήση του DOP θα την έκανε αδικαιολόγητα δυσκολότερη. Είναι κάτι που πρέπει να χρησιμοποιούμε ανάλογα με το πρόβλημα το οποίο προσπαθούμε να λύσουμε, επειδή το πρόβλημα είναι αυτό που θα καθορίσει την καλύτερη επιλογή και θα μας οδηγήσει στην τελική απόφαση.

Όσων αναφορά τις μελλοντικές αναβαθμίσεις, οι ιδέες για ένα παιχνίδι είναι πάρα πολλές, αλλά για αρχή στην TODO λίστα είναι η προσθήκη ειδικών εφέ με την χρήση shaders που θα προσφέρει στο παιχνίδι το στοιχείο του φωτισμού, η δημιουργία particle system που θα χρησιμοποιηθεί για την προσομοίωση φωτιάς, καπνού, υγρών κλπ. Ακόμα η αναβάθμιση στην φυσική και κυρίως στην κίνηση του κυρίως χαρακτήρα είναι αρκετά επιθυμητή για να επιφέρει μεγαλύτερη ικανοποίηση στον χρήστη που θα ελέγχει τον κύριο χαρακτήρα του παιχνιδιού.

Παραρτήματα

Παράρτημα Α΄

Παράρτημα

Α΄.1 Κώδικας πειράματος

Α΄.1.1 Κοινές κλάσεις μεταξύ των δύο προγραμμάτων

QueryCallback.h

```
#pragma once
#include <Box2D/Box2D.h>

//Class copied from http://code.google.com/p/box2d/source/browse/trunk/
    Box2D/Testbed/Framework/Test.cpp
//Copyright (c) 2011 Erin Catto http://box2d.org
class QueryCallback : public b2QueryCallback
{
public:
    QueryCallback(const b2Vec2& point)
    {
        m_point = point;
        m_fixture = NULL;
    }

    bool ReportFixture(b2Fixture* fixture)
    {
        b2Body* body = fixture->GetBody();
        if (body->GetType() == b2_dynamicBody)
        {
            bool inside = fixture->TestPoint(m_point);
            if (inside)
            {
                m_fixture = fixture;

                // We are done, terminate the query.
                return false;
            }
        }

        // Continue the query.
```

```
        return true;
    }

    b2Vec2 m_point;
    b2Fixture* m_fixture;
};
```

Α΄.1.2 Πρόγραμμα σχεδιασμένο με OOP

Main.cpp

```
#include "Game.h"


int main()
{
    Game game;
    game.Run();

    return 0;
}
```

Game.h

```
#pragma once
#include <SFML/Graphics.hpp>
#include <Box2D/Box2D.h>

#include <vector>
#include <sstream>

#include "QueryCallback.h"

class Game
{
public:
    Game();

    void Run();

    sf::RenderWindow& getWindow() { return window; }
    b2World& getWorld() { return m_world; }


private:
    void HandleEvents(sf::Event&);
    void Update(sf::Time);
    void Render(sf::Time&);

    void updateStatistics(sf::Time);
```

```
    void CreateBoxes();
    void createWalls();
    void centerOrigin(sf::RectangleShape& rect);
    template <typename T>
    T radToDeg(T radians);
    template <typename T>
    T degToRad(T degrees);

    // Utility functions to scale sf::Vector to box2d Vector and vice versa
    template<class T>
    b2Vec2 sfVecToB2Vec(sf::Vector2<T> vector);
    sf::Vector2f b2VecTosfVec(b2Vec2 vector);

    sf::RenderWindow window;
    sf::Font mainFont;
    sf::Text m_StatisticsText;
    sf::Time m_StatisticsUpdateTime;
    std::size_t m_StatisticsNumFrames;

    b2World m_world;
    std::vector<b2Body*> bodies;

    b2MouseJoint* mouseJoint;
    b2Body* ground;

    const double PI = 3.14159265358979323846264338327;
    /* constants for time step and physics accuracy */
    const sf::Time TimePerFrame = sf::seconds(1 / 60.f);
    const int velocityIterations = 8;
    const int positionIterations = 3;
};
```

Game.cpp

```
#include "Game.h"

#include <random>

namespace sfdd
{
    const float SCALE = 50.f;
}

Game::Game() : window(sf::VideoMode(1024, 768, 32), "Box2D - SFML Debug
    Draw Test"),
               m_world(b2Vec2(0.f, 9.81f))
{

    if (!mainFont.loadFromFile("../res/Font/CHICKEN Pie.ttf"))
    {
        exit(EXIT_FAILURE);
```

```
    }

    m_StatisticsText.setFont(mainFont);
    m_StatisticsText.setFillColor(sf::Color::White);
    m_StatisticsText.setOutlineThickness(1);
    m_StatisticsText.setOutlineColor(sf::Color::Black);

    m_world.SetAllowSleeping(true);

    createWalls();
    CreateBoxes();
}

void Game::HandleEvents(sf::Event& event)
{
    while (window.pollEvent(event))
    {
        if (event.type == sf::Event::Closed || (event.type == sf::Event::
            KeyPressed && event.key.code == sf::Keyboard::Escape))
            window.close();

        // Following three events are copied almost completely from http://
            code.google.com/p/box2d/source/browse/trunk/Box2D/Testbed/
            Framework/Test.cpp
        // Copyright (c) 2011 Erin Catto http://box2d.org
        if (event.type == sf::Event::MouseButtonPressed && event.
            mouseButton.button == sf::Mouse::Left && mouseJoint == nullptr)
        {
            b2Vec2 mousePos = sfVecToB2Vec(sf::Mouse::getPosition(window));

            // Make a small box.
            b2AABB aabb;
            b2Vec2 d;
            d.Set(0.001f, 0.001f);
            aabb.lowerBound = mousePos + d;
            aabb.upperBound = mousePos - d;

            // Query the world for overlapping shapes.
            QueryCallback callback(mousePos);
            m_world.QueryAABB(&callback, aabb);

            if (callback.m_fixture)
            {
                b2Body* body = callback.m_fixture->GetBody();
                b2MouseJointDef md;
                md.bodyA = ground; //If bounding box body is used instead,
                    the dynamic bodes can be dragged over the bounding box
                    and we don't want that
                md.bodyB = body;
                md.target = mousePos;
                md.maxForce = 1000.0f * body->GetMass();
                mouseJoint = (b2MouseJoint*)m_world.CreateJoint(&md);
```

```
                body->SetAwake(true);
            }
        }
        else if (event.type == sf::Event::MouseMoved && mouseJoint !=
            nullptr)
        {
            b2Vec2 mousePos = sfVecToB2Vec(sf::Mouse::getPosition(window));
            mouseJoint->SetTarget(mousePos);
        }
        else if (event.type == sf::Event::MouseButtonReleased && event.
            mouseButton.button == sf::Mouse::Left && mouseJoint != nullptr)
        {
            m_world.DestroyJoint(mouseJoint);
            mouseJoint = nullptr;
        }
    }
}

void Game::Update(sf::Time fixedDt)
{
    m_world.Step(1 / 60.f, velocityIterations, positionIterations);

    for (auto& b : bodies)
    {
        if (b->GetType() == b2BodyType::b2_staticBody) continue;

        for (auto* fixture = b->GetFixtureList(); fixture; fixture =
            fixture->GetNext())
        {
            if (fixture != nullptr && fixture->GetUserData() != nullptr)
            {
                auto shape = static_cast<sf::RectangleShape*>(fixture->
                    GetUserData());

                shape->setPosition(b2VecTosfVec(b->GetPosition()));
                shape->setRotation(radToDeg(b->GetAngle()));
            }
        }
    }
}

void Game::Run()
{
    sf::Clock clock;
    sf::Time timeSinceLastUpdate = sf::Time::Zero;

    while (window.isOpen())
    {
        sf::Event event;
        sf::Time dt = clock.restart();
        timeSinceLastUpdate += dt;
        while (timeSinceLastUpdate > TimePerFrame)
```

```
        {
            timeSinceLastUpdate -= TimePerFrame;
            HandleEvents(event);
            Update(TimePerFrame);
        }
        Render(dt);
        updateStatistics(dt);
    }
}

void Game::Render(sf::Time &deltaTime)
{
    window.clear();

    for(auto &b : bodies)
        for (auto* fixture = b->GetFixtureList(); fixture; fixture =
            fixture->GetNext())
        {
            if (fixture != nullptr && fixture->GetUserData() != nullptr)
            {
                auto shape = static_cast<sf::RectangleShape*>(fixture->
                    GetUserData());
                window.draw(*shape);
            }
        }

    window.draw(m_StatisticsText);

    window.display();
}

void Game::updateStatistics(sf::Time dt)
{
    m_StatisticsUpdateTime += dt;
    m_StatisticsNumFrames += 1;
    if (m_StatisticsUpdateTime >= sf::seconds(1.0f))
    {
        m_StatisticsText.setString("FPS: " + std::to_string(
            m_StatisticsNumFrames) + "  FrameTime:  " + std::to_string(dt.
            asMicroseconds()) + "μs" + "   Bodies: " + std::to_string(bodies
            .size()));
        m_StatisticsUpdateTime -= sf::seconds(1.0f);
        m_StatisticsNumFrames = 0;
    }
}

//Converts SFML's vector to Box2D's vector and downscales it so it fits
    Box2D's MKS units
template<typename T >
b2Vec2 Game::sfVecToB2Vec(sf::Vector2<T> vector)
{
    return b2Vec2(vector.x / sfdd::SCALE, vector.y / sfdd::SCALE);
```

```
}

sf::Vector2f Game::b2VecTosfVec(b2Vec2 vector)
{
    return sf::Vector2f(vector.x * sfdd::SCALE, vector.y * sfdd::SCALE);
}

void Game::CreateBoxes()
{
    for (auto i = 0; i < 999; ++i)
    {
        sf::RectangleShape *rect = new sf::RectangleShape(sf::Vector2f(20.f
            , 20.f));
        rect->setFillColor(sf::Color(145, 4, 4));
        centerOrigin(*rect);

        // Create the body definition
        sf::Vector2f position(static_cast<float>(rand() % 1000),
            static_cast<float>(rand() % 600));
        auto bodySize = rect->getSize();
        b2BodyDef bodyDef;
        bodyDef.type = b2_dynamicBody;
        bodyDef.position = sfVecToB2Vec(position);
        //bodyDef.fixedRotation = true;

        // Create and register the body in the world
        b2Body* body = m_world.CreateBody(&bodyDef);
        bodies.push_back(body);
        // Fixture shape
        b2PolygonShape bShape;
        b2Vec2 size = sfVecToB2Vec(bodySize / 2.f);
        bShape.SetAsBox(size.x, size.y);

        // Fixture definition
        b2FixtureDef fixture;
        fixture.shape = &bShape;
        fixture.density = 1.f;
        fixture.friction = 1.f;
        fixture.restitution = 0.f;
        fixture.userData = rect;
        body->CreateFixture(&fixture);
    }
}

void Game::createWalls()
{
    /* Create the bounding box */
    b2BodyDef boundingBoxDef;
    boundingBoxDef.type = b2_staticBody;
    float xPos = (window.getSize().x / 2.f) / sfdd::SCALE;
    float yPos = 0.5f;
    boundingBoxDef.position.Set(xPos, yPos);
```

```
b2Body* boundingBoxBody = m_world.CreateBody(&boundingBoxDef);
bodies.push_back(boundingBoxBody);

b2PolygonShape boxShape;
boxShape.SetAsBox(window.getSize().x / sfdd::SCALE, 0.5f, b2Vec2(0.f,
    0.f), 0.f);
auto fixture1 = boundingBoxBody->CreateFixture(&boxShape, 1.0); //Top

//Top
{
    auto size = sf::Vector2f(static_cast<float>(window.getSize().x),
        static_cast<float>(1.f * sfdd::SCALE));
    sf::RectangleShape *rect = new sf::RectangleShape(size);
    centerOrigin(*rect);
    rect->setFillColor(sf::Color(66, 144, 245));
    rect->setPosition(xPos * sfdd::SCALE, yPos * sfdd::SCALE);

    fixture1->SetUserData(rect);
}

yPos = (window.getSize().y) / sfdd::SCALE - 1.f;
boxShape.SetAsBox((window.getSize().x) / sfdd::SCALE, 0.5f, b2Vec2(0.f,
     yPos), 0.f);
auto fixture2 = boundingBoxBody->CreateFixture(&boxShape, 1.f); //
    Bottom

//Bottom
{
    auto size = sf::Vector2f(static_cast<float>(window.getSize().x),
        static_cast<float>(1.f * sfdd::SCALE));
    sf::RectangleShape *rect = new sf::RectangleShape(size);
    centerOrigin(*rect);
    rect->setFillColor(sf::Color(66, 144, 245));
    rect->setPosition(xPos * sfdd::SCALE, yPos * sfdd::SCALE + 25.f);

    fixture2->SetUserData(rect);
}

xPos -= 0.5f;
boxShape.SetAsBox(0.5f, (window.getSize().y) / sfdd::SCALE, b2Vec2(-
    xPos, 0.f), 0.f);
auto fixture3 = boundingBoxBody->CreateFixture(&boxShape, 1.f);//Left

//Left
{
    auto size = sf::Vector2f(static_cast<float>(1.f * sfdd::SCALE),
        static_cast<float>(window.getSize().y) * 2.f);
    sf::RectangleShape *rect = new sf::RectangleShape(size);
    centerOrigin(*rect);
    rect->setFillColor(sf::Color(66, 144, 245));
    rect->setPosition(25.f, yPos * sfdd::SCALE);
```

```
        fixture3->SetUserData(rect);
    }

    boxShape.SetAsBox(0.5f, (window.getSize().y) / sfdd::SCALE, b2Vec2(xPos
        , 0.f), 0.f);
    auto fixture4 = boundingBoxBody->CreateFixture(&boxShape, 1.f);//Right

    //Right
    {
        auto size = sf::Vector2f(1.f * sfdd::SCALE,
            static_cast<float>(window.getSize().y) * 2.f);
        sf::RectangleShape *rect = new sf::RectangleShape(size);
        centerOrigin(*rect);
        rect->setFillColor(sf::Color(66, 144, 245));
        rect->setPosition(xPos * sfdd::SCALE + window.getSize().x / 2.f,
            yPos * sfdd::SCALE);

        fixture4->SetUserData(rect);
    }

    /* Mouse Joint */
    mouseJoint = nullptr;
    b2BodyDef bodyDef;
    ground = m_world.CreateBody(&bodyDef);
}

void Game::centerOrigin(sf::RectangleShape& rect)
{
    const sf::FloatRect bounds = rect.getLocalBounds();
    rect.setOrigin(std::floor(bounds.left + bounds.width / 2.f), std::floor
        (bounds.top + bounds.height / 2.f));
}

template <typename T>
T Game::radToDeg(T radians)
{
    return (180 / PI) * radians;
}

template <typename T>
T Game::degToRad(T degrees)
{
    return (PI / 180) * degrees;
}
```

Α΄.1.3 Πρόγραμμα σχεδιασμένο με DOP

Main.cpp

```
#include "core/Game.h"


int main()
{
    Game game;
    game.Run();

    return 0;
}
```

Game.h

```
#pragma once

#include <entt/entt.hpp>
#include <Box2D/Box2D.h>
#include <SFML/Graphics/Text.hpp>

#include "../core/EntityManager.hpp"
#include "../Utils/Utility.hpp"

class Game : private sf::NonCopyable
{
public:
    Game();
    void Run();

private:
    void HandleEvents();
    void Update(sf::Time dt);
    void Render();

    void loadResources();
    void updateStatistics(sf::Time dt);
    void buildScene();
    void createWalls();

private:
    sf::RenderWindow m_window;
    entt::registry m_Registry;
    b2World m_B2DWorld;
    Context m_Context;

    sf::Font m_font;

    EntityManager em;
```

```
    std::unordered_map<entt::entity, b2Body*> bodies;

    sf::Text m_StatisticsText;
    sf::Time m_StatisticsUpdateTime;
    std::size_t m_StatisticsNumFrames;

    b2MouseJoint* mouseJoint;
    b2Body* ground;
};
```

Game.cpp

```
#include "Game.h"
#include "../Components/C_Body.hpp"
#include "../Components/C_Rigidbody.hpp"
#include "../Systems/MoveSystem.hpp"
#include "../Systems/CollisionSystem.hpp"
#include "../Systems/RenderSystem.hpp"
#include "../Utils/QueryCallback.h"

#include <SFML/Window/Event.hpp>

#include <memory>

const sf::Time TimePerFrame = sf::seconds(1 / 60.f);

Game::Game() :
    m_window(sf::VideoMode(1024, 768), "Application",
            sf::Style::Close, sf::ContextSettings(0,0,0,1,1)),
    m_B2DWorld(b2Vec2(0, 9.81)),
    m_Context(m_window, m_Registry, m_B2DWorld),
    m_StatisticsText(),
    m_StatisticsUpdateTime(),
    m_StatisticsNumFrames(0)
{
    m_window.setKeyRepeatEnabled(false);

    m_B2DWorld.SetAllowSleeping(true);

    loadResources();
    buildScene();

    m_StatisticsText.setFont(m_font);
    m_StatisticsText.setOutlineThickness(3);
    m_StatisticsText.setPosition(5.f, 5.f);
    m_StatisticsText.setCharacterSize(20u);


    /* Mouse Joint */
    mouseJoint = nullptr;
    b2BodyDef bodyDef;
```

```
    ground = m_B2DWorld.CreateBody(&bodyDef);
}

void Game::Run()
{
    sf::Clock clock;
    sf::Time timeSinceLastUpdate = sf::Time::Zero;

    while (m_window.isOpen())
    {
        const sf::Time dt = clock.restart();
        timeSinceLastUpdate += dt;
        while (timeSinceLastUpdate > TimePerFrame)
        {
            timeSinceLastUpdate -= TimePerFrame;
            HandleEvents();
            Update(TimePerFrame);
        }
        updateStatistics(dt);
        Render();
    }
}

void Game::HandleEvents()
{
    sf::Event event{};
    while (m_window.pollEvent(event))
    {
        if (event.type == sf::Event::Closed)
            m_window.close();

        // Following three events are copied almost completely from http://
            code.google.com/p/box2d/source/browse/trunk/Box2D/Testbed/
            Framework/Test.cpp
        // Copyright (c) 2011 Erin Catto http://box2d.org
        if (event.type == sf::Event::MouseButtonPressed && event.
            mouseButton.button == sf::Mouse::Left && mouseJoint == nullptr)
        {
            b2Vec2 mousePos = utils::sfVecToB2Vec(sf::Mouse::getPosition(
                m_window));

            // Make a small box.
            b2AABB aabb;
            b2Vec2 d;
            d.Set(0.001f, 0.001f);
            aabb.lowerBound = mousePos + d;
            aabb.upperBound = mousePos - d;

            // Query the world for overlapping shapes.
            QueryCallback callback(mousePos);
            m_B2DWorld.QueryAABB(&callback, aabb);
```

```
            if (callback.m_fixture)
            {
                b2Body* body = callback.m_fixture->GetBody();
                b2MouseJointDef md;
                md.bodyA = ground; //If bounding box body is used instead,
                    the dynamic bodes can be dragged over the bounding box
                    and we don't want that
                md.bodyB = body;
                md.target = mousePos;
                md.maxForce = 1000.0f * body->GetMass();
                mouseJoint = (b2MouseJoint*)m_B2DWorld.CreateJoint(&md);
                body->SetAwake(true);
            }
        }
        else if (event.type == sf::Event::MouseMoved && mouseJoint !=
            nullptr)
        {
            b2Vec2 mousePos = utils::sfVecToB2Vec(sf::Mouse::getPosition(
                m_window));
            mouseJoint->SetTarget(mousePos);
        }
        else if (event.type == sf::Event::MouseButtonReleased && event.
            mouseButton.button == sf::Mouse::Left && mouseJoint != nullptr)
        {
            m_B2DWorld.DestroyJoint(mouseJoint);
            mouseJoint = nullptr;
        }
    }
}

void Game::Update(sf::Time dt)
{
    em.update(dt);
}

void Game::Render()
{
    m_window.clear();

    em.draw();

    m_window.draw(m_StatisticsText);

    m_window.setView(m_window.getDefaultView());
    m_window.display();
}

void Game::loadResources()
{
    m_font.loadFromFile("../res/Font/ARJULIAN.ttf");
}
```

```
void Game::updateStatistics(sf::Time dt)
{
    m_StatisticsUpdateTime += dt;
    m_StatisticsNumFrames += 1;
    if (m_StatisticsUpdateTime >= sf::seconds(1.0f))
    {
        m_StatisticsText.setString("FPS: " + std::to_string(
            m_StatisticsNumFrames) + "  FrameTime: " + std::to_string(dt.
            asMicroseconds()) + "μs" + "  Bodies: " + std::to_string(bodies.
            size()));
        m_StatisticsUpdateTime -= sf::seconds(1.0f);
        m_StatisticsNumFrames = 0;
    }
}

void Game::buildScene()
{
    createWalls();

    std::srand(std::time(nullptr));
    for (auto i = 0; i < 1000; ++i)
    {
        const auto entity = m_Registry.create();
        m_Registry.emplace<Body>(entity, sf::RectangleShape(sf::Vector2f
            (20.f, 20.f)));

        // Create the body definition
        sf::Vector2f position(static_cast<float>(rand() % 1100),
            static_cast<float>(rand() % 700));
        auto bodySize = m_Registry.view<Body>().get<Body>(entity);
        b2BodyDef bodyDef;
        bodyDef.type = b2_dynamicBody;
        bodyDef.position = utils::sfVecToB2Vec(position);
        //bodyDef.fixedRotation = true;

        // Create and register the body in the world
        b2Body* body = m_B2DWorld.CreateBody(&bodyDef);

        // Fixture shape
        b2PolygonShape bShape;
        b2Vec2 size = utils::sfVecToB2Vec(bodySize.shape.getSize() / 2.f);
        bShape.SetAsBox(size.x, size.y);

        // Fixture definition
        b2FixtureDef fixture;
        fixture.shape = &bShape;
        fixture.density = 1.f;
        fixture.friction = 1.f;
        fixture.restitution = 0.f;
        body->CreateFixture(&fixture);

        bodies[entity] = body;
```

```
        m_Registry.emplace<C_Rigidbody>(entity, body);
    }

    em.addSystem(std::make_unique<MoveSystem>(m_Context));
    em.addSystem(std::make_unique<CollisionSystem>(m_Context));
    em.addRenderSystem(std::make_unique<RenderSystem>(m_Context));
}

void Game::createWalls()
{
    /* Create the bounding box */
    b2BodyDef boundingBoxDef;
    boundingBoxDef.type = b2_staticBody;
    float xPos = (m_window.getSize().x / 2.f) / sfdd::SCALE;
    float yPos = 0.5f;
    boundingBoxDef.position.Set(xPos, yPos);

    b2Body* boundingBoxBody = m_B2DWorld.CreateBody(&boundingBoxDef);

    b2PolygonShape boxShape;
    boxShape.SetAsBox(m_window.getSize().x / sfdd::SCALE, 0.5f, b2Vec2(0.f,
        0.f), 0.f);
    boundingBoxBody->CreateFixture(&boxShape, 1.0); //Top

    //Top
    {
        auto size = sf::Vector2f(static_cast<float>(m_window.getSize().x),
            static_cast<float>(1.f * sfdd::SCALE));
        sf::RectangleShape rect(size);
        rect.setPosition(xPos * sfdd::SCALE, yPos * sfdd::SCALE);

        const auto entity = m_Registry.create();
        //Add body to entity-bodies map
        bodies[entity] = boundingBoxBody;
        m_Registry.emplace<Body>(entity, rect);
        m_Registry.emplace<C_Rigidbody>(entity, boundingBoxBody);
    }

    yPos = (m_window.getSize().y) / sfdd::SCALE - 1.f;
    boxShape.SetAsBox((m_window.getSize().x) / sfdd::SCALE, 0.5f, b2Vec2(0.
        f, yPos), 0.f);
    boundingBoxBody->CreateFixture(&boxShape, 1.f); //Bottom

    //Bottom
    {
        auto size = sf::Vector2f(static_cast<float>(m_window.getSize().x),
            static_cast<float>(1.f * sfdd::SCALE));
        sf::RectangleShape rect(size);
        rect.setPosition(xPos * sfdd::SCALE, yPos * sfdd::SCALE + 16.f);

        const auto entity = m_Registry.create();
```

```
        //Add body to entity-bodies map
        bodies[entity] = boundingBoxBody;
        m_Registry.emplace<Body>(entity, rect);
        m_Registry.emplace<C_Rigidbody>(entity, boundingBoxBody);
    }

    xPos -= 0.5f;
    boxShape.SetAsBox(0.5f, (m_window.getSize().y) / sfdd::SCALE, b2Vec2(-
        xPos, 0.f), 0.f);
    boundingBoxBody->CreateFixture(&boxShape, 1.f);//Left

    //Left
    {
        auto size = sf::Vector2f(static_cast<float>(1.f * sfdd::SCALE),
            static_cast<float>(m_window.getSize().y) * 2.f);
        sf::RectangleShape rect(size);
        rect.setPosition(16.f, yPos * sfdd::SCALE);

        const auto entity = m_Registry.create();
        //Add body to entity-bodies map
        bodies[entity] = boundingBoxBody;
        m_Registry.emplace<Body>(entity, rect);
        m_Registry.emplace<C_Rigidbody>(entity, boundingBoxBody);
    }

    boxShape.SetAsBox(0.5f, (m_window.getSize().y) / sfdd::SCALE, b2Vec2(
        xPos, 0.f), 0.f);
    boundingBoxBody->CreateFixture(&boxShape, 1.f);//Right

    //Right
    {
        auto size = sf::Vector2f(1.f * sfdd::SCALE,
            static_cast<float>(m_window.getSize().y) * 2.f);
        sf::RectangleShape rect(size);
        rect.setPosition(xPos * sfdd::SCALE + m_window.getSize().x / 2.f,
            yPos * sfdd::SCALE);

        const auto entity = m_Registry.create();
        //Add body to entity-bodies map
        bodies[entity] = boundingBoxBody;
        m_Registry.emplace<Body>(entity, rect);
        m_Registry.emplace<C_Rigidbody>(entity, boundingBoxBody);
    }
}
```

EntityManager.h

```
#pragma once

#include <vector>
#include <memory>
```

```
#include <entt/entity/registry.hpp>
#include <SFML/System/Time.hpp>

#include "../Systems/BaseSystem.hpp"

class EntityManager
{
public:
    EntityManager() = default;
    EntityManager(entt::registry* reg);

    void update(sf::Time dt);
    void draw();

    void addSystem(std::unique_ptr<BaseSystem> sys);
    void addRenderSystem(std::unique_ptr<BaseSystem> sys);

private:
    entt::registry* m_Registry{};
    std::vector<std::unique_ptr<BaseSystem>> m_Systems{};
    std::vector<std::unique_ptr<BaseSystem>> m_RenderSystems{};
};
```

EntityManager.cpp

```
#include "EntityManager.hpp"

EntityManager::EntityManager(entt::registry* reg) :
    m_Registry(reg)
{
}

void EntityManager::update(sf::Time dt)
{
    for (auto& system : m_Systems)
        system->update(dt);
}

void EntityManager::draw()
{
    for (auto& system : m_RenderSystems)
        system->update(sf::Time());
}

void EntityManager::addSystem(std::unique_ptr<BaseSystem> sys)
{
    m_Systems.push_back(std::move(sys));
}

void EntityManager::addRenderSystem(std::unique_ptr<BaseSystem> sys)
```

```
{
    m_RenderSystems.push_back(std::move(sys));
}
```

C_Body

```
#pragma once

#include <SFML/Graphics.hpp>
#include <Box2D/Common/b2Math.h>
#include "../Utils/Utility.hpp"

struct Body
{
    Body() = default;
    Body(sf::RectangleShape rect) :
        shape(std::move(rect))
    {
        utils::centerOrigin(shape);
        shape.setFillColor(sf::Color(rand() % 255, rand() % 255, rand() %
            255));
    }

    sf::RectangleShape shape{};
};
```

C_Rigidbody

```
#pragma once

#include <Box2D/Dynamics/b2Body.h>

struct C_Rigidbody
{
    b2Body* rigidbody{};
};
```

BaseSystem.h

```
#pragma once

#include "../Utils/Context.hpp"

namespace sf
{
    class Time;
}
```

```
class BaseSystem
{
public:
    BaseSystem() = default;
    BaseSystem(Context& context);

    virtual void update(sf::Time dt) = 0;

protected:
    Context* m_Context;
};
```

BaseSystem.cpp

```
#include "BaseSystem.hpp"

BaseSystem::BaseSystem(Context& context) :
    m_Context(&context)
{
}
```

CollisionSystem.h

```
#pragma once

#include "BaseSystem.hpp"
#include <SFML/System/Vector2.hpp>

class Body;

class CollisionSystem : public BaseSystem
{
public:
    CollisionSystem() = default;
    explicit CollisionSystem(Context& context);

    void update(sf::Time dt) override;
};
```

CollisionSystem.cpp

```
#include "CollisionSystem.hpp"
#include "../Components/C_Body.hpp"

CollisionSystem::CollisionSystem(Context& context) :
    BaseSystem(context)
{
}
```

```
void CollisionSystem::update(sf::Time dt)
{
    m_Context->world->Step(1 / 60.f, 8, 5);
}
```

MoveSystem.h

```
#pragma once

#include "BaseSystem.hpp"

class MoveSystem : public BaseSystem
{
public:
    MoveSystem() = default;
    MoveSystem(Context& context);

    void update(sf::Time dt) override;
};
```

MoveSystem.cpp

```
#include "MoveSystem.hpp"

#include "../Utils/Math.hpp"
#include "../Components/C_Body.hpp"
#include "../Components/C_Rigidbody.hpp"

#include <SFML/System/Time.hpp>

MoveSystem::MoveSystem(Context& context) :
    BaseSystem(context)
{
}

void MoveSystem::update(sf::Time dt)
{
    m_Context->registry->view<Body, C_Rigidbody>().each([&](auto& body,
        auto& rb)
        {
            if(rb.rigidbody->GetType() != b2_staticBody)
            {
                body.shape.setRotation(math::radToDeg(rb.rigidbody->
                    GetAngle()));
                body.shape.setPosition(utils::B2VecToSFVec<sf::Vector2f>(rb
                    .rigidbody->GetPosition()));
            }
        });
}
```

RenderSystem.h

```
#pragma once

#include "BaseSystem.hpp"

class RenderSystem : public BaseSystem
{
public:
    RenderSystem() = default;
    RenderSystem(Context& context);

    void update(sf::Time dt) override;
};
```

RenderSystem.cpp

```
#include "RenderSystem.hpp"
#include "../Components/C_Body.hpp"

#include <SFML/System/Time.hpp>

RenderSystem::RenderSystem(Context& context) :
BaseSystem(context)
{
}

void RenderSystem::update(sf::Time dt)
{
    m_Context->registry->view<Body>().each([&](auto& body)
        {
            m_Context->window->draw(body.shape);
        });
}
```

Utility.h

```
#pragma once

#include <SFML/Window/Keyboard.hpp>
#include <SFML/Graphics/RectangleShape.hpp>
#include <SFML/System/Vector2.hpp>
#include <Box2D/Box2D.h>
#include <cassert>
#include <cmath>


namespace sfdd
{
    const float SCALE = 32.f;
```

```
}

namespace utils
{
    void centerOrigin(sf::RectangleShape& rect);

    //Converts SFML's vector to Box2D's vector and downscales it so it fits
        Box2D's MKS units
    template <typename T>
    b2Vec2 sfVecToB2Vec(sf::Vector2<T> vector)
    {
        return b2Vec2(vector.x / sfdd::SCALE, vector.y / sfdd::SCALE);
    }

    // Convert Box2D's vector to SFML vector
    template <typename T>
    T B2VecToSFVec(const b2Vec2 &vector, bool scaleToPixels = true)
    {
        return T(vector.x * (scaleToPixels ? sfdd::SCALE : 1.f), vector.y *
            (scaleToPixels ? sfdd::SCALE : 1.f));
    }

}
```

Utility.cpp

```
#include "Utility.hpp"

namespace utils
{
    void centerOrigin(sf::RectangleShape& rect)
    {
        const sf::FloatRect bounds = rect.getLocalBounds();
        rect.setOrigin(std::floor(bounds.left + bounds.width / 2.f), std::
            floor(bounds.top + bounds.height / 2.f));
    }
}
```

Context.h

```
#pragma once

#include <entt/entity/registry.hpp>
#include <SFML/Graphics/RenderWindow.hpp>
#include <Box2D/Box2D.h>

#include "Utility.hpp"

struct Context
{
```

```
    Context() = default;
    Context(sf::RenderWindow& win, entt::registry& reg, b2World& wr)
        :
        window(&win),
        registry(&reg),
        world(&wr)
    {
    }

    sf::RenderWindow* window;
    entt::registry* registry;
    b2World* world;
};
```

Math.h

```
#pragma once

#include <cmath>
#include <SFML/System/Vector2.hpp>

namespace math
{
    constexpr const double PI = 3.14159265358979323846264338327;

    template <typename T>
    T radToDeg(T radians)
    {
        return (180 / PI) * radians;
    }

    template <typename T>
    T degToRad(T degrees)
    {
        return (PI / 180) * degrees;
    }
}
```

Παράρτημα Β΄

Παράρτημα

Β΄.1 Κώδικας του κυρίως προγράμματος της εργασίας

Το project μπορεί να βρεθεί στον προσωπικό μου λογαριασμό Github: `https://github.com/ggiap/2D_Game` μαζί με οδηγίες για εγκατάσταση.

main.cpp

```
#include "core/Game.h"

int main()
{
    Game game;
    game.Run();

    return 0;
}
```

Game.h

```
#pragma once

#include "../States/StateStack.h"
#include "../Utils/ResourceHolder.h"
#include "../Utils/Utility.hpp"
#include "../core/MusicPlayer.hpp"
#include "../core/SoundEffectPlayer.hpp"

class Game : private sf::NonCopyable
{
public:
    Game();
    void Run();

private:
    void HandleEvents();
```

```
    void Update(sf::Time dt);
    void Render();

    void loadResources();
    void registerStates();
    void updateStatistics(sf::Time dt);

private:
    const sf::Time TimePerFrame = sf::seconds(1 / 60.f);

    sf::RenderWindow  m_window;
    FontHolder        m_Fonts;
    TextureHolder     m_Textures;
    StateStack        m_StateStack;
    MusicPlayer       m_MusicPlayer;
    SoundEffectPlayer m_SoundPlayer;

    sf::Text          m_StatisticsText;
    sf::Time          m_StatisticsUpdateTime;
    std::size_t       m_StatisticsNumFrames;

    bool              m_IsPaused{ false };
};
```

Game.cpp

```
#include "Game.h"

#include "../States/GameState.h"
#include "../States/MenuState.hpp"
#include "../States/SplashScreenState.hpp"
#include "../States/PauseState.hpp"

#include <SFML/Window/Event.hpp>

Game::Game() :
    m_window(sf::VideoMode(1200, 800), "Application",
            sf::Style::Close | sf::Style::Resize, sf::ContextSettings
                (0,0,0,1,1)),
    m_Fonts(),
    m_MusicPlayer(),
    m_SoundPlayer(),
    m_StateStack(Context(m_window, m_Fonts, m_Textures, m_MusicPlayer,
        m_SoundPlayer)),
    m_StatisticsText(),
    m_StatisticsUpdateTime(),
    m_StatisticsNumFrames(0)
{
    m_window.setKeyRepeatEnabled(false);

    loadResources();
```

```
    m_StatisticsText.setFont(m_Fonts.get(Fonts::ID::ARJULIAN));
    m_StatisticsText.setOutlineThickness(3.f);
    m_StatisticsText.setOutlineColor(sf::Color::Black);
    m_StatisticsText.setCharacterSize(20u);

    registerStates();
    m_StateStack.pushState(States::ID::Title);
}

void Game::Run()
{
    sf::Clock clock;
    sf::Time timeSinceLastUpdate = sf::Time::Zero;

    while (m_window.isOpen())
    {
        const sf::Time dt = clock.restart();
        timeSinceLastUpdate += dt;
        while (timeSinceLastUpdate > TimePerFrame)
        {
            timeSinceLastUpdate -= TimePerFrame;
            HandleEvents();

            // Check inside this loop, because stack might be empty before
                the update() call
            if (m_StateStack.isEmpty())
                m_window.close();

            if(!m_IsPaused)
                Update(TimePerFrame);
        }

        if (!m_IsPaused)
        {
            updateStatistics(dt);
            Render();
        }
    }
}

void Game::HandleEvents()
{
    sf::Event event{};
    while (m_window.pollEvent(event))
    {
        m_StateStack.handleEvent(event);

        if (event.type == sf::Event::LostFocus)
        {
            m_IsPaused = true;
        }
```

```
        if (event.type == sf::Event::GainedFocus)
        {
            m_IsPaused = false;
        }

        if (event.type == sf::Event::Closed)
            m_window.close();
    }
}

void Game::Update(sf::Time dt)
{
    m_StateStack.update(dt);

    m_StatisticsText.setPosition(m_window.getView().getCenter().x -
        m_window.getView().getSize().x / 2.f,
                                 m_window.getView().getCenter().y -
                                     m_window.getView().getSize().y / 2.f);
    m_StatisticsText.setScale(m_window.getView().getSize().x /  m_window.
        getDefaultView().getSize().x,
                              m_window.getView().getSize().y /  m_window.
                                  getDefaultView().getSize().y);
}

void Game::Render()
{
    m_window.clear();
    m_StateStack.draw();
    m_window.draw(m_StatisticsText);
    m_window.display();
}

void Game::loadResources()
{
    m_Fonts.load(Fonts::ID::ARJULIAN, "../res/Font/ARJULIAN.ttf");
    m_Textures.load(Textures::ID::MonochromeSpriteSheet, "../res/Sprites/
        Tilemap/monochrome_tilemap_transparent_packed.png");
    m_Textures.load(Textures::ID::CharactersSpriteSheet, "../res/Sprites/
        Tilemap/tilemap_packed_16x16_2.png");
    m_Textures.load(Textures::ID::MainBackground, "../res/Sprites/
        MainBackground.png");
    m_Textures.load(Textures::ID::CoinAnimation, "../res/Sprites/Tilemap/
        Coin_Animation.png");
}

void Game::registerStates()
{
    m_StateStack.registerState<SplashScreenState>(States::ID::Title);
    m_StateStack.registerState<MenuState>(States::ID::Menu);
    m_StateStack.registerState<GameState>(States::ID::Game);
    m_StateStack.registerState<PauseState>(States::ID::Pause);
```

```
}

void Game::updateStatistics(sf::Time dt)
{
    m_StatisticsUpdateTime += dt;
    m_StatisticsNumFrames += 1;
    if (m_StatisticsUpdateTime >= sf::seconds(1.0f))
    {
        m_StatisticsText.setString("FPS: " + std::to_string(
            m_StatisticsNumFrames));
        m_StatisticsUpdateTime -= sf::seconds(1.0f);
        m_StatisticsNumFrames = 0;
    }
}
```

World.h

```
#pragma once

#include <map>

#include <SFML/Graphics/RenderWindow.hpp>
#include <SFML/Graphics/View.hpp>
#include <SFML/Window/Event.hpp>

#include "../core/SystemManager.hpp"
#include "../Utils/Utility.hpp"
#include "../Utils/Timer.hpp"
#include "../Utils/Animation.h"
#include "../Utils/ContactListener.hpp"

struct Context;

class World
{
public:
    explicit World(Context& context);
    ~World();

    void update(sf::Time dt);
    void handleEvents(const sf::Event& event);
    void draw();

    b2World* getB2World();
    entt::registry* getEntityRegistry();
    float getRemainingTime();
    bool& sfmlDebugging();
    bool& b2dDebugging();
    void spawnEnemy();
    std::vector<entt::entity>& getMarkedEntities();
    unsigned& getNumberOfEnemies();
```

```
    bool& GameOver();

private:
    void buildScene();
    void createAnimations();
    void createPlayer();
    void createCamera();
    void createTilemap();
    void createEnemies();
    void createCoins();
    void destroyMarkedEntities();

    void unloadScene();
    void updateHUD(const sf::Time& dt);

private:
    sf::View m_WorldView;
    sf::FloatRect m_WorldBounds;

    Context* m_Context;
    SystemManager m_SystemManager;
    std::unique_ptr<b2World> m_b2World;
    entt::registry m_WorldRegistry;

    std::unordered_map<Animations::ID, Animation*> anims;

    std::vector<entt::entity> markedForDestruction;

    ContactListener m_ContactListener;

    Timer m_CountdownTimer;
    sf::Text m_TimerLabel;

    bool sfmlDebug;
    bool b2dDebug;
    bool gameOver;

    unsigned m_numberOfEnemies;

    sf::Sprite m_coinHudSprite;
    sf::Text m_coinCounLabel;
    unsigned m_coinCount;
};
```

World.cpp

```
#include "Game.h"

#include "../States/GameState.h"
#include "../States/MenuState.hpp"
#include "../States/SplashScreenState.hpp"
```

```
#include "../States/PauseState.hpp"

#include <SFML/Window/Event.hpp>

Game::Game() :
    m_window(sf::VideoMode(1200, 800), "Application",
            sf::Style::Close | sf::Style::Resize, sf::ContextSettings
                (0,0,0,1,1)),
    m_Fonts(),
    m_MusicPlayer(),
    m_SoundPlayer(),
    m_StateStack(Context(m_window, m_Fonts, m_Textures, m_MusicPlayer,
        m_SoundPlayer)),
    m_StatisticsText(),
    m_StatisticsUpdateTime(),
    m_StatisticsNumFrames(0)
{
    m_window.setKeyRepeatEnabled(false);

    loadResources();

    m_StatisticsText.setFont(m_Fonts.get(Fonts::ID::ARJULIAN));
    m_StatisticsText.setOutlineThickness(3.f);
    m_StatisticsText.setOutlineColor(sf::Color::Black);
    m_StatisticsText.setCharacterSize(20u);

    registerStates();
    m_StateStack.pushState(States::ID::Title);
}

void Game::Run()
{
    sf::Clock clock;
    sf::Time timeSinceLastUpdate = sf::Time::Zero;

    while (m_window.isOpen())
    {
        const sf::Time dt = clock.restart();
        timeSinceLastUpdate += dt;
        while (timeSinceLastUpdate > TimePerFrame)
        {
            timeSinceLastUpdate -= TimePerFrame;
            HandleEvents();

            // Check inside this loop, because stack might be empty before
                the update() call
            if (m_StateStack.isEmpty())
                m_window.close();

            if(!m_IsPaused)
                Update(TimePerFrame);
        }
```

```
        if (!m_IsPaused)
        {
            updateStatistics(dt);
            Render();
        }
    }
}

void Game::HandleEvents()
{
    sf::Event event{};
    while (m_window.pollEvent(event))
    {
        m_StateStack.handleEvent(event);

        if (event.type == sf::Event::LostFocus)
        {
            m_IsPaused = true;
        }

        if (event.type == sf::Event::GainedFocus)
        {
            m_IsPaused = false;
        }

        if (event.type == sf::Event::Closed)
            m_window.close();
    }
}

void Game::Update(sf::Time dt)
{
    m_StateStack.update(dt);

    m_StatisticsText.setPosition(m_window.getView().getCenter().x -
        m_window.getView().getSize().x / 2.f,
                                 m_window.getView().getCenter().y -
                                     m_window.getView().getSize().y / 2.f);
    m_StatisticsText.setScale(m_window.getView().getSize().x /  m_window.
        getDefaultView().getSize().x,
                              m_window.getView().getSize().y /  m_window.
                                  getDefaultView().getSize().y);
}

void Game::Render()
{
    m_window.clear();
    m_StateStack.draw();
    m_window.draw(m_StatisticsText);
    m_window.display();
}
```

```
void Game::loadResources()
{
    m_Fonts.load(Fonts::ID::ARJULIAN, "../res/Font/ARJULIAN.ttf");
    m_Textures.load(Textures::ID::MonochromeSpriteSheet, "../res/Sprites/
        Tilemap/monochrome_tilemap_transparent_packed.png");
    m_Textures.load(Textures::ID::CharactersSpriteSheet, "../res/Sprites/
        Tilemap/tilemap_packed_16x16_2.png");
    m_Textures.load(Textures::ID::MainBackground, "../res/Sprites/
        MainBackground.png");
    m_Textures.load(Textures::ID::CoinAnimation, "../res/Sprites/Tilemap/
        Coin_Animation.png");
}

void Game::registerStates()
{
    m_StateStack.registerState<SplashScreenState>(States::ID::Title);
    m_StateStack.registerState<MenuState>(States::ID::Menu);
    m_StateStack.registerState<GameState>(States::ID::Game);
    m_StateStack.registerState<PauseState>(States::ID::Pause);
}

void Game::updateStatistics(sf::Time dt)
{
    m_StatisticsUpdateTime += dt;
    m_StatisticsNumFrames += 1;
    if (m_StatisticsUpdateTime >= sf::seconds(1.0f))
    {
        m_StatisticsText.setString("FPS: " + std::to_string(
            m_StatisticsNumFrames));
        m_StatisticsUpdateTime -= sf::seconds(1.0f);
        m_StatisticsNumFrames = 0;
    }
}
```

SystemManager.h

```
#pragma once

#include <vector>
#include <memory>

#include <entt/entity/registry.hpp>
#include <SFML/System/Time.hpp>

#include "../Systems/BaseSystem.hpp"

class SystemManager
{
public:
    SystemManager() = default;
```

```
    void update(sf::Time dt);
    void handleEvents(const sf::Event& event);
    void draw();

    void addSystem(std::unique_ptr<BaseSystem> sys);
    void deleteAllSystems();

    void initSystems();

private:
    std::vector<std::unique_ptr<BaseSystem>> m_Systems{};
};
```

SystemManager.cpp

```
#include "SystemManager.hpp"

void SystemManager::update(sf::Time dt)
{
    for (auto& system : m_Systems)
        system->update(dt);
}

void SystemManager::handleEvents(const sf::Event &event)
{
    for (auto& system : m_Systems)
        system->handleEvents(event);
}

void SystemManager::draw()
{
    for (auto& system : m_Systems)
        system->draw();
}

void SystemManager::addSystem(std::unique_ptr<BaseSystem> sys)
{
    m_Systems.push_back(std::move(sys));
}

void SystemManager::deleteAllSystems()
{
    m_Systems.clear();
}

void SystemManager::initSystems()
{
    for (auto& system : m_Systems)
        system->init();
}
```

MusicPlayer.h

```
#pragma once

#include <map>

#include "../Utils/Utility.hpp"

#include <SFML/Audio/Music.hpp>

class MusicPlayer : sf::NonCopyable
{
public:
    MusicPlayer();

    void play(Music::ID id);
    void stop();
    void setPaused(bool paused);

private:
    sf::Music m_Music;
    std::map<Music::ID, std::string> m_Filenames;
    float m_Volume;
};
```

MusicPlayer.cpp

```
#include "MusicPlayer.hpp"

MusicPlayer::MusicPlayer() :
    m_Music(),
    m_Filenames(),
    m_Volume(100.f)
{
    m_Filenames[Music::ID::MenuTheme] = "../res/Music/Menu_Select.ogg";
    m_Filenames[Music::ID::GameTheme] = "../res/Music/Stage 1.ogg";
}

void MusicPlayer::play(Music::ID id)
{
    auto filename = m_Filenames.at(id);

    if (!m_Music.openFromFile(filename))
    {
        throw std::runtime_error("Couldn't load " + filename + ".");
    }

    m_Music.setVolume(m_Volume);
    m_Music.setLoop(true);
    m_Music.play();
}
```

```
void MusicPlayer::stop()
{
    m_Music.stop();
}

void MusicPlayer::setPaused(bool paused)
{
    if (paused)
        m_Music.pause();
    else
        m_Music.play();
}
```

SoundEffectPlayer.h

```
#pragma once

#include "../Utils/Utility.hpp"
#include "../Utils/ResourceHolder.h"

#include <SFML/Audio/Sound.hpp>
#include <SFML/Audio/SoundBuffer.hpp>

#include <list>

class SoundEffectPlayer : public sf::NonCopyable
{
public:
    SoundEffectPlayer();

    void play(Sounds::ID id);

private:
    SoundBufferHolder m_SoundBuffers;
    std::list<sf::Sound> m_Sounds;
};
```

SoundEffectPlayer.cpp

```
#include "SoundEffectPlayer.hpp"

SoundEffectPlayer::SoundEffectPlayer() :
    m_SoundBuffers(),
    m_Sounds()
{
    m_SoundBuffers.load(Sounds::ID::CoinPickup, "../res/SoundEffects/coin.
        wav");
    m_SoundBuffers.load(Sounds::ID::Pause, "../res/SoundEffects/pause.wav")
        ;
```

```
    m_SoundBuffers.load(Sounds::ID::Squash, "../res/SoundEffects/squash.wav
        ");
}

void SoundEffectPlayer::play(Sounds::ID id)
{
    m_Sounds.push_back(sf::Sound(m_SoundBuffers.get(id)));
    m_Sounds.back().setVolume(10.f);
    m_Sounds.back().play();
}
```

C_Animation.h

```
#pragma once

#include "Utils/AnimatedSprite.h"

struct C_Animation
{
    C_Animation() = default;
    C_Animation(sf::Time frameTime, bool paused, bool looped) :
    animatedSprite(frameTime, paused, looped)
    {

    }

    AnimatedSprite animatedSprite{};
};
```

C_Camera.h

```
#pragma once

#include <entt/entity/registry.hpp>
#include <SFML/Graphics/View.hpp>

struct C_Camera
{
    C_Camera() = default;
    C_Camera(sf::View view, entt::entity target) :
        view(view),
        defaultView(view),
        target(target)
    {

    }

    sf::View view{};
    sf::View defaultView{};
    entt::entity target{entt::null};
```

```
};
```

C_Raycast.h

```
#pragma once

#include <Box2D/Common/b2Math.h>
#include <entt/entity/entity.hpp>

#include <vector>
#include <array>


struct CollisionInfo
{
    bool collisionAbove{false};
    bool collisionBelow{false};
    bool collisionLeft {false};
    bool collisionRight{false};
    bool groundCheckLeft{false};
    bool groundCheckRight{false};
    bool climbingLadder{false};
    entt::entity entityBelow{entt::null};
    entt::entity entityAbove{entt::null};
    entt::entity entityLeft {entt::null};
    entt::entity entityRight{entt::null};
    b2Vec2 normalBelow{};

    void reset()
    {
        collisionAbove = false;
        collisionBelow = false;
        collisionLeft  = false;
        collisionRight = false;
        groundCheckLeft = false;
        groundCheckRight = false;
        climbingLadder = false;
        entityBelow = entt::null;
        entityAbove = entt::null;
        entityLeft = entt::null;
        entityRight = entt::null;
        normalBelow = b2Vec2();
    }
};

struct RaycastOrigins
{
    b2Vec2 topLeft{}, topRight{};
    b2Vec2 bottomLeft{}, bottomRight{};
};
```

```
struct C_Raycast
{
    C_Raycast() = default;

    CollisionInfo collisionInfo{};
    RaycastOrigins raycastOrigins{};
    float rayLength{2.f};
    int horizontalRayCount{};
    int verticalRayCount{};
    float dstBetweenRays = .8f;
    float horizontalRaySpacing{};
    float verticalRaySpacing{};

    std::vector<std::array<sf::Vertex, 2>> raycasts;
};
```

C_Rigidbody.h

```
#pragma once

#include <Box2D/Dynamics/b2Body.h>

struct C_Rigidbody
{
    b2Body* rigidbody{};
};
```

C_Tag.h

```
#pragma once

// The following components are used as entity tags
struct C_PlayerTag
{

};

struct C_EnemyTag
{

};

struct C_Coin
{

};

struct C_OneWayPlatform
{
```

```
};

struct C_Spikes
{

};

struct C_Ladder
{

};
```

C_Tilemap.h

```
#pragma once

#include "../Tilemaps/SFMLOrthogonalLayer.hpp"

#include <vector>
#include <memory>

struct C_Tilemap
{
    C_Tilemap() = default;
    explicit C_Tilemap(const std::string &path) : m_MapPath(path)
    {
    }

    std::string m_MapPath{};
    std::vector<std::unique_ptr<MapLayer>> m_TileLayers;
    std::vector<std::unique_ptr<tmx::ObjectGroup>> m_ObjectLayers;
};
```

BaseSystem.h

```
#pragma once

struct Context;
class World;

namespace sf
{
    class Time;
    class Event;
}

class BaseSystem
{
public:
    BaseSystem() = default;
```

```
    explicit BaseSystem(Context& context, World *world);

    virtual void update(sf::Time& dt) { }
    virtual void handleEvents(const sf::Event& event) { }
    virtual void draw() { }

    virtual void init() { }

protected:
    Context* m_Context{};
    World* m_World;
};
```

BaseSystem.cpp

```
#include "BaseSystem.hpp"
#include "../core/World.h"

BaseSystem::BaseSystem(Context& context, World *world) :
    m_Context(&context),
    m_World(world)
{

}
```

AnimationSystem.h

```
#pragma once

#include "BaseSystem.hpp"

#include <SFML/System/Time.hpp>

struct Context;

class AnimationSystem : public BaseSystem
{
public:
    AnimationSystem() = default;
    explicit AnimationSystem(Context &context, World *world);

    void update(sf::Time& dt) override;
};
```

AnimationSystem.cpp

```
#include "AnimationSystem.hpp"
#include "../Components/C_Animation.hpp"
```

```
#include "../Utils//Context.hpp"
#include "../core/World.h"

AnimationSystem::AnimationSystem(Context& context, World *world) :
BaseSystem(context, world)
{

}

void AnimationSystem::update(sf::Time& dt)
{
    auto view = m_World->getEntityRegistry()->view<C_Animation>();

    for (auto& entity : view)
    {
        auto& animComp = m_World->getEntityRegistry()->get<C_Animation>(
            entity);

        animComp.animatedSprite.update(dt);
    }
}
```

CameraSystem.h

```
#pragma once

#include "BaseSystem.hpp"

namespace sf
{
    class View;
    template<typename T>
    class Vector2;
}
struct C_Camera;

class CameraSystem : public BaseSystem
{
public:
    CameraSystem() = default;
    explicit CameraSystem(Context& context, World* world);

    void update(sf::Time& dt) override;
    void handleEvents(const sf::Event& event) override;

private:
    void cameraPanning();
    void zoomViewAt(sf::Vector2<int> pixel, float zoom);
    void resetView();

    bool freeRoaming{false};
```

```
};
```

CameraSystem.cpp

```
#include "CameraSystem.hpp"
#include "../Utils/Context.hpp"
#include "../Components/C_Camera.hpp"
#include "../core/World.h"

#include <SFML/System/Time.hpp>
#include <SFML/System/Vector2.hpp>
#include <SFML/Window/Event.hpp>

CameraSystem::CameraSystem(Context &context, World* world) :
BaseSystem(context, world),
freeRoaming(false)
{ }

void CameraSystem::update(sf::Time& dt)
{
    m_World->getEntityRegistry()->view<C_Camera>().each([&](auto& entity,
        auto& cameraComp)
    {
        if(freeRoaming == false)
        {
            auto targetPos = utils::B2VecToSFVec<sf::Vector2f>(m_Context->
                enttToBody[cameraComp.target]->GetPosition());
            cameraComp.view.setCenter(targetPos);
            m_Context->window->setView(cameraComp.view);
        }
        else
        {
          cameraPanning();
        }
    });
}

void CameraSystem::handleEvents(const sf::Event& event)
{
    if( sf::Keyboard::isKeyPressed(sf::Keyboard::Up) ||
        sf::Keyboard::isKeyPressed(sf::Keyboard::Down)||
        sf::Keyboard::isKeyPressed(sf::Keyboard::Right)||
        sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
    {
        freeRoaming = true;
    }

    if (event.type == sf::Event::MouseWheelScrolled)
    {
        freeRoaming = true;
        auto zoomAmount = 1.1f;
```

```
        if (event.mouseWheelScroll.delta > 0)
            zoomViewAt({event.mouseWheelScroll.x, event.mouseWheelScroll.y
                }, (1.f / zoomAmount));
        else if (event.mouseWheelScroll.delta < 0)
            zoomViewAt({event.mouseWheelScroll.x, event.mouseWheelScroll.y
                }, zoomAmount);
    }

    if (sf::Keyboard::isKeyPressed(sf::Keyboard::I))
    {
        freeRoaming = false;
        resetView();
    }
}

void CameraSystem::cameraPanning()
{
    freeRoaming = true;

    m_World->getEntityRegistry()->view<C_Camera>().each([&](auto& entity,
        auto& cameraComp)
    {
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up))
        {
            cameraComp.view.move({0.f, -10.f});
            m_Context->window->setView(cameraComp.view);
        }
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down))
        {
            cameraComp.view.move({0.f, 10.f});
            m_Context->window->setView(cameraComp.view);
        }
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right))
        {
            cameraComp.view.move({10.f, 0.f});
            m_Context->window->setView(cameraComp.view);
        }
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
        {
            cameraComp.view.move({-10.f, 0.f});
            m_Context->window->setView(cameraComp.view);
        }
    });
}

void CameraSystem::zoomViewAt(sf::Vector2<int> pixel, float zoom)
{
    m_World->getEntityRegistry()->view<C_Camera>().each([&](auto& entity,
        auto& cameraComp)
    {
        const sf::Vector2f beforeCoord{ m_Context->window->mapPixelToCoords
            (pixel) };
```

```
        cameraComp.view.zoom(zoom);
        m_Context->window->setView(cameraComp.view);
        const sf::Vector2f afterCoord{ m_Context->window->mapPixelToCoords(
            pixel) };
        const sf::Vector2f offsetCoords{ beforeCoord - afterCoord };
        cameraComp.view.move(offsetCoords);
        m_Context->window->setView(cameraComp.view);
    });
}

void CameraSystem::resetView()
{
    m_World->getEntityRegistry()->view<C_Camera>().each([&](auto& entity,
        auto& cameraComp)
    {
        cameraComp.view = cameraComp.defaultView;
    });
}
```

CollisionSystem.h

```
#pragma once

#include "BaseSystem.hpp"
#include "../Utils/RayCastCallback.hpp"

#include <SFML/System/Vector2.hpp>
#include <SFML/Graphics.hpp>

#include <array>
#include <vector>

class CollisionSystem : public BaseSystem
{
public:
    CollisionSystem() = default;
    explicit CollisionSystem(Context& context, World* world);

    void update(sf::Time& dt) override;
    void init() override;

private:
    void handleRaycasts();
    void handleLadders();
    void CalculateRaySpacing();
    void UpdateRaycastOrigins();
    void createLine(b2Vec2 &origin, b2Vec2 &direction, std::vector<std::
        array<sf::Vertex, 2>>& lines);

    RayCastCallback m_Callback{};
};
```

CollisionSystem.cpp

```
#include "CollisionSystem.hpp"
#include "../Utils/Context.hpp"
#include "../Utils/Math.hpp"
#include "../Utils/FixtureUserData.hpp"
#include "../Components/C_Tag.h"
#include "../Components/C_Rigidbody.hpp"
#include "../Components/C_Raycast.hpp"
#include "../core/World.h"

#include <SFML/System/Time.hpp>
#include <SFML/Graphics/CircleShape.hpp>

CollisionSystem::CollisionSystem(Context& context, World* world) :
    BaseSystem(context, world),
    m_Callback()
{
    init();
}

void CollisionSystem::update(sf::Time& dt)
{
    m_World->getB2World()->Step(dt.asSeconds(), 8, 10);

    UpdateRaycastOrigins();
    handleRaycasts();
    handleLadders();
}

void CollisionSystem::init()
{
    CalculateRaySpacing();
}

void CollisionSystem::handleRaycasts()
{
    auto view = m_World->getEntityRegistry()->view<C_Raycast>();

    for(auto entity : view)
    {
        auto &rb = m_World->getEntityRegistry()->get<C_Rigidbody>(entity);
        auto &raycastComp = m_World->getEntityRegistry()->get<C_Raycast>(
            entity);
        raycastComp.collisionInfo.reset();

        for(b2Fixture* fixture = rb.rigidbody->GetFixtureList(); fixture;
            fixture = fixture->GetNext())
        {
            // Keep enemies on platforms
            if (m_World->getEntityRegistry()->has<C_EnemyTag>(entity))
            {
```

```
            b2Vec2 rayOrigin = raycastComp.raycastOrigins.bottomLeft +
                utils::sfVecToB2Vec(math::VECTOR_UP);
            b2Vec2 rayDir = rayOrigin + utils::sfVecToB2Vec(math::
                VECTOR_DOWN * (raycastComp.rayLength * 2));
            m_World->getB2World()->RayCast(&m_Callback, rayOrigin,
                rayDir);
            if (m_Callback.m_fixture != nullptr)
            {
                raycastComp.collisionInfo.groundCheckLeft = true;
            }

            createLine(rayOrigin, rayDir, raycastComp.raycasts);

            m_Callback = RayCastCallback();

            rayOrigin = raycastComp.raycastOrigins.bottomRight + utils
                ::sfVecToB2Vec(math::VECTOR_UP);
            rayDir = rayOrigin + utils::sfVecToB2Vec(math::VECTOR_DOWN
                * (raycastComp.rayLength * 2));
            m_World->getB2World()->RayCast(&m_Callback, rayOrigin,
                rayDir);
            if (m_Callback.m_fixture != nullptr)
            {
                raycastComp.collisionInfo.groundCheckRight = true;
            }

            createLine(rayOrigin, rayDir, raycastComp.raycasts);

            m_Callback = RayCastCallback();
        }

        // Check Above
        for (int i = 0; i < raycastComp.verticalRayCount; ++i)
        {
            b2Vec2 rayOrigin = raycastComp.raycastOrigins.topLeft +
                utils::sfVecToB2Vec(math::VECTOR_DOWN / 2.f +
                math::VECTOR_RIGHT * (raycastComp.verticalRaySpacing *
                    i));
            b2Vec2 rayDir = rayOrigin + utils::sfVecToB2Vec(math::
                VECTOR_UP * raycastComp.rayLength);

            createLine(rayOrigin, rayDir, raycastComp.raycasts);

            m_World->getB2World()->RayCast(&m_Callback, rayOrigin,
                rayDir);
            if (m_Callback.m_fixture != nullptr)
            {
                auto userData = static_cast<FixtureUserData*>(
                    m_Callback.m_fixture->GetUserData());
                if (userData == nullptr) continue;

                raycastComp.collisionInfo.collisionAbove = true;
```

```
                raycastComp.collisionInfo.entityAbove = userData->
                    entity;
                break;
            }
            m_Callback = RayCastCallback();
        }
        m_Callback = RayCastCallback();

        // Check Below
        for (int i = 0; i < raycastComp.verticalRayCount; ++i)
        {
            b2Vec2 rayOrigin = raycastComp.raycastOrigins.bottomRight +
                 utils::sfVecToB2Vec(math::VECTOR_UP / 2.f +
                math::VECTOR_LEFT * (raycastComp.verticalRaySpacing * i
                    ));
            b2Vec2 rayDir = rayOrigin + utils::sfVecToB2Vec(math::
                VECTOR_DOWN * raycastComp.rayLength);

            createLine(rayOrigin, rayDir, raycastComp.raycasts);

            m_World->getB2World()->RayCast(&m_Callback, rayOrigin,
                rayDir);
            if (m_Callback.m_fixture != nullptr)
            {
                auto userData = static_cast<FixtureUserData*>(
                    m_Callback.m_fixture->GetUserData());
                if (userData == nullptr) continue;

                if (m_World->getEntityRegistry()->has<C_Spikes>(
                    userData->entity) &&
                    m_World->getEntityRegistry()->has<C_PlayerTag>(
                        entity))
                    m_World->GameOver() = true;

                raycastComp.collisionInfo.collisionBelow = true;
                raycastComp.collisionInfo.entityBelow = userData->
                    entity;
                //raycastComp.collisionInfo.normalBelow = m_Callback.
                    m_normal;
                break;
            }
            m_Callback = RayCastCallback();
        }
        m_Callback = RayCastCallback();

        // Check Right
        for (int i = 0; i < raycastComp.horizontalRayCount - 1; ++i)
        {
            b2Vec2 rayOrigin = raycastComp.raycastOrigins.topRight +
                utils::sfVecToB2Vec(math::VECTOR_LEFT / 2.f +
                math::VECTOR_DOWN * (raycastComp.horizontalRaySpacing *
                     i));
```

```
            b2Vec2 rayDir = rayOrigin + (utils::sfVecToB2Vec(math::
                VECTOR_RIGHT * raycastComp.rayLength));

            createLine(rayOrigin, rayDir, raycastComp.raycasts);

            m_World->getB2World()->RayCast(&m_Callback, rayOrigin,
                rayDir);
            if (m_Callback.m_fixture != nullptr)
            {
                auto userData = static_cast<FixtureUserData*>(
                    m_Callback.m_fixture->GetUserData());
                if (userData == nullptr) continue;

                raycastComp.collisionInfo.collisionRight = true;
                raycastComp.collisionInfo.entityRight = userData->
                    entity;
                break;
            }
            m_Callback = RayCastCallback();
        }
        m_Callback = RayCastCallback();

        // Check Left
        for (int i = 0; i < raycastComp.horizontalRayCount - 1; ++i)
        {
            b2Vec2 rayOrigin = raycastComp.raycastOrigins.bottomLeft +
                utils::sfVecToB2Vec(math::VECTOR_RIGHT / 2.f +
                math::VECTOR_UP * (raycastComp.horizontalRaySpacing * i
                    ));
            auto rayDir = rayOrigin + utils::sfVecToB2Vec(math::
                VECTOR_LEFT * raycastComp.rayLength);

            createLine(rayOrigin, rayDir, raycastComp.raycasts);

            m_World->getB2World()->RayCast(&m_Callback, rayOrigin,
                rayDir);
            if(m_Callback.m_fixture != nullptr)
            {
                auto userData = static_cast<FixtureUserData*>(
                    m_Callback.m_fixture->GetUserData());
                if (userData == nullptr) continue;

                raycastComp.collisionInfo.collisionLeft = true;
                raycastComp.collisionInfo.entityLeft = userData->entity
                    ;
                break;
            }
            m_Callback = RayCastCallback();
        }
        m_Callback = RayCastCallback();
    }
}
```

```
}

void CollisionSystem::handleLadders()
{
    b2Fixture* playerFixture = nullptr;
    FixtureUserData* playerUserData = nullptr;
    auto playerView = m_World->getEntityRegistry()->view<C_PlayerTag>();
    for (auto entity : playerView)
    {
        playerFixture = m_Context->enttToBody[entity]->GetFixtureList();
        playerUserData = static_cast<FixtureUserData*>(playerFixture->
            GetUserData());
    }

    b2Fixture* ladderFixture = nullptr;
    auto ladderView = m_World->getEntityRegistry()->view<C_Ladder>();
    for (auto entity : ladderView)
    {
        ladderFixture = m_Context->enttToBody[entity]->GetFixtureList();

        if (!ladderFixture || !playerFixture) return;

        b2AABB ladderAABB = ladderFixture->GetAABB(0);
        b2AABB playerAABB = playerFixture->GetAABB(0);

        if (ladderAABB.Contains(playerAABB))
        {
            auto& raycastComp = m_World->getEntityRegistry()->get<C_Raycast
                >(playerUserData->entity);
            raycastComp.collisionInfo.climbingLadder = true;
        }
    }
}

void CollisionSystem::CalculateRaySpacing()
{
    auto view = m_World->getEntityRegistry()->view<C_Raycast>();

    for(auto entity : view)
    {
        auto &rb = m_World->getEntityRegistry()->get<C_Rigidbody>(entity);
        auto &raycastComp = m_World->getEntityRegistry()->get<C_Raycast>(
            entity);
        for(b2Fixture* fixture = rb.rigidbody->GetFixtureList(); fixture;
            fixture = fixture->GetNext())
        {
            auto userData = static_cast<FixtureUserData*>(fixture->
                GetUserData());
            if (userData == nullptr || userData->shape == nullptr) continue
                ;

            auto bounds = userData->shape->getGlobalBounds();
```

```
            float boundsWidth = bounds.width;
            float boundsHeight = bounds.height;

            raycastComp.horizontalRayCount = std::ceil(boundsHeight /
                raycastComp.dstBetweenRays);
            raycastComp.verticalRayCount = std::ceil(boundsWidth /
                raycastComp.dstBetweenRays);

            raycastComp.horizontalRaySpacing = boundsHeight / (raycastComp.
                horizontalRayCount);
            raycastComp.verticalRaySpacing = boundsWidth / (raycastComp.
                verticalRayCount);

            raycastComp.horizontalRayCount += 1;
            raycastComp.verticalRayCount += 1;
        }
    }
}

void CollisionSystem::UpdateRaycastOrigins()
{
    m_World->getEntityRegistry()->view<C_Raycast, C_Rigidbody>().each([&](
        auto entity, auto& raycastComp, auto& rb)
    {
        raycastComp.raycasts.clear();
        auto fixture = rb.rigidbody->GetFixtureList();

        auto userData = static_cast<FixtureUserData*>(fixture->GetUserData
            ());
        if (userData == nullptr || userData->shape == nullptr) return;

        auto bounds = userData->shape->getGlobalBounds();
        auto &size = userData->shape->getSize();

        raycastComp.raycastOrigins.topLeft = utils::sfVecToB2Vec(
                sf::Vector2f(bounds.left, bounds.top));
        raycastComp.raycastOrigins.topRight = utils::sfVecToB2Vec(
                sf::Vector2f(bounds.left + size.x, bounds.top + .1f));
        raycastComp.raycastOrigins.bottomLeft = utils::sfVecToB2Vec(
                sf::Vector2f(bounds.left, bounds.top + size.y - .1f));
        raycastComp.raycastOrigins.bottomRight = utils::sfVecToB2Vec(
                sf::Vector2f(bounds.left + size.x, bounds.top + size.y));
    });
}

void CollisionSystem::createLine(b2Vec2& origin, b2Vec2& direction, std::
    vector<std::array<sf::Vertex, 2>>& lines)
{
    std::array<sf::Vertex, 2> line;
    line[0].position = utils::B2VecToSFVec<sf::Vector2f>(origin);
    line[1].position = utils::B2VecToSFVec<sf::Vector2f>(direction);
```

```
    line[0].color = sf::Color::Red;
    line[1].color = sf::Color::Red;
    lines.push_back(line);
}
```

EnemyControllerSystem.h

```
#pragma once

#include "BaseSystem.hpp"
#include "../Utils/RayCastCallback.hpp"

class EnemyControllerSystem : public BaseSystem
{
public:
    EnemyControllerSystem() = default;
    explicit EnemyControllerSystem(Context &context, World *world);

    void update(sf::Time & dt) override;

private:
    RayCastCallback m_Callback{};
};
```

EnemyControllerSystem.cpp

```
#include "EnemyControllerSystem.hpp"

#include "../Utils/Context.hpp"
#include "../Utils/Math.hpp"
#include "../core/World.h"
#include "../Components/C_Tag.h"
#include "../Components/C_Rigidbody.hpp"
#include "../Components/C_Raycast.hpp"
#include "../Components/C_Animation.hpp"

#include <SFML/System/Time.hpp>

EnemyControllerSystem::EnemyControllerSystem(Context &context, World* world
    ) :
    BaseSystem(context, world),
    m_Callback()
{
}

void EnemyControllerSystem::update(sf::Time& dt)
{
    m_World->getEntityRegistry()->view<C_EnemyTag, C_Rigidbody, C_Raycast,
        C_Animation>().each([&](auto entity, auto& rb, auto& raycastComp,
        auto& animComp)
```

```
    {
        b2Vec2 velocity = m_Context->enttToBody[entity]->
            GetLinearVelocity();

        if (velocity.x < 0.f)
            velocity.x += -1.f;
        else
            velocity.x += 1.f;

        if (raycastComp.collisionInfo.collisionBelow && raycastComp.
            collisionInfo.groundCheckLeft == false)
            if (velocity.x < 0.f)
                velocity.x = std::abs(velocity.x);

        if (raycastComp.collisionInfo.collisionBelow && raycastComp.
            collisionInfo.groundCheckRight == false)
            if (velocity.x > 0.f)
                velocity.x = -std::abs(velocity.x);

        if(raycastComp.collisionInfo.collisionLeft)
            velocity.x = std::abs(velocity.x);

        if (raycastComp.collisionInfo.collisionRight)
            velocity.x = -std::abs(velocity.x);


        velocity.x = std::clamp(velocity.x, -1.f, 1.f);
        velocity.y = std::clamp(velocity.y, -10.f, 10.f);
        m_Context->enttToBody[entity]->SetLinearVelocity(velocity);

        if (velocity.x < -0.1f)
            animComp.animatedSprite.setScale(sf::Vector2f(1.f, 1.f));
        else if (velocity.x > 0.1f)
            animComp.animatedSprite.setScale(sf::Vector2f(-1.f, 1.f));

        if (raycastComp.collisionInfo.collisionLeft || raycastComp.
            collisionInfo.collisionRight || raycastComp.collisionInfo.
            collisionBelow)
        {
            if(raycastComp.collisionInfo.entityLeft != entt::null &&
                m_World->getEntityRegistry()->has<C_PlayerTag>(
                    raycastComp.collisionInfo.entityLeft))
                m_World->GameOver() = true;
            else if(raycastComp.collisionInfo.entityRight != entt::null
                &&
                m_World->getEntityRegistry()->has<C_PlayerTag>(
                    raycastComp.collisionInfo.entityRight))
                m_World->GameOver() = true;
            else if (raycastComp.collisionInfo.entityBelow != entt::
                null &&
                m_World->getEntityRegistry()->has<C_PlayerTag>(
                    raycastComp.collisionInfo.entityBelow))
```

```
                    m_World->GameOver() = true;
                }
            });
}
```

MoveSystem.h

```
#pragma once

#include "BaseSystem.hpp"

class MoveSystem : public BaseSystem
{
public:
    MoveSystem() = default;
    explicit MoveSystem(Context& context, World *world);

    void update(sf::Time& dt) override;
};
```

MoveSystem.cpp

```
#include "MoveSystem.hpp"

#include "../Utils/Math.hpp"
#include "../Utils/FixtureUserData.hpp"
#include "../Utils//Context.hpp"
#include "../Components/C_Rigidbody.hpp"
#include "../Components/C_Animation.hpp"
#include "../Components/C_Raycast.hpp"
#include "../Components/C_Tag.h"
#include "../core/World.h"

#include <SFML/System/Time.hpp>

MoveSystem::MoveSystem(Context& context, World *world) :
    BaseSystem(context, world)
{

}

void MoveSystem::update(sf::Time& dt)
{
    m_World->getEntityRegistry()->view<C_Rigidbody>().each([&](auto entity,
        auto& rb)
        {
            if(rb.rigidbody->GetType() != b2_staticBody)
            {
                for(auto fixture = rb.rigidbody->GetFixtureList(); fixture
                    != nullptr; fixture = fixture->GetNext())
```

```
                {
                    auto userData = static_cast<FixtureUserData*>(fixture->
                        GetUserData());
                    if (userData == nullptr || userData->shape == nullptr)
                        continue;

                    userData->shape->setPosition(utils::B2VecToSFVec<sf::
                        Vector2f>(rb.rigidbody->GetPosition()));
                    userData->shape->setRotation(math::radToDeg(rb.
                        rigidbody->GetAngle()));
                }
            }

            if(m_World->getEntityRegistry()->has<C_Animation>(entity))
            {
                auto &anim = m_World->getEntityRegistry()->get<C_Animation
                    >(entity);

                auto bodyPos = utils::B2VecToSFVec<sf::Vector2f>(rb.
                    rigidbody->GetPosition());
                anim.animatedSprite.setPosition(bodyPos);
            }
        });
}
```

PlayerControllerSystem.h

```
#pragma once

#include "BaseSystem.hpp"
#include "../Utils/Utility.hpp"

namespace sf
{
    class Time;
}

struct Context;

class PlayerControllerSystem : public BaseSystem
{
public:
    PlayerControllerSystem() = default;
    explicit PlayerControllerSystem(Context& context, World *world);

    void update(sf::Time& dt) override;

private:
    GameObjectState::ID m_State{};
    float m_lowJumpMultiplier;
    float m_fallMultiplier;
```

```
};
```

PlayerCOntrollerSystem.cpp

```
#include "PlayerControllerSystem.hpp"
#include "../Utils/AnimatedSprite.h"
#include "../Utils/Context.hpp"
#include "../Utils/Math.hpp"
#include "../Utils/FixtureUserData.hpp"
#include "../Components/C_Tag.h"
#include "../Components/C_Animation.hpp"
#include "../Components/C_Raycast.hpp"
#include "../core/World.h"
#include "../core/SoundEffectPlayer.hpp"

#include <SFML/Window/Event.hpp>

#include <cmath>
#include <spdlog/spdlog.h>

PlayerControllerSystem::PlayerControllerSystem(Context& context, World*
    world) :
    BaseSystem(context, world),
    m_State(),
    m_lowJumpMultiplier(1.08f)
{

}

void PlayerControllerSystem::update(sf::Time& dt)
{
    auto registry = m_World->getEntityRegistry();
    auto view = registry->view<C_PlayerTag, C_Animation, C_Raycast>();

    for (auto& entity : view)
    {
        auto& anim = view.get<C_Animation>(entity);
        auto& raycastComp = view.get<C_Raycast>(entity);

        auto userData = static_cast<FixtureUserData*>(m_Context->enttToBody
            [entity]->GetFixtureList()->GetUserData());
        if (userData == nullptr || userData->shape == nullptr) continue;

        b2Vec2 velocity = m_Context->enttToBody[entity]->GetLinearVelocity
            ();
        velocity.x -= velocity.x > 0 ? .2f : -.2f;
        if (velocity.x < .21f && velocity.x > -.21f) velocity.x = 0;

        if (sf::Keyboard::isKeyPressed(sf::Keyboard::A))
        {
            if (!raycastComp.collisionInfo.collisionLeft)
```

```
            velocity.x -= 0.6f;

        if (raycastComp.collisionInfo.collisionBelow)
            m_State = GameObjectState::ID::Walking;
    }

    if (sf::Keyboard::isKeyPressed(sf::Keyboard::D))
    {
        if (!raycastComp.collisionInfo.collisionRight)
            velocity.x += 0.6f;

        if (raycastComp.collisionInfo.collisionBelow)
            m_State = GameObjectState::ID::Walking;
    }
    velocity.x = math::lerp(velocity.x, velocity.x, 0.5f);

    if (raycastComp.collisionInfo.entityBelow != entt::null)
    {
        if (registry->has<C_EnemyTag>(raycastComp.collisionInfo.
            entityBelow))
        {
            m_Context->sounds->play(Sounds::ID::Squash);
            spdlog::info("Killed enemy, numOfEnemies: {}", --m_World->
                getNumberOfEnemies());
            auto* body = m_Context->enttToBody[raycastComp.
                collisionInfo.entityBelow];
            for (auto* fixture = body->GetFixtureList(); fixture;
                fixture = fixture->GetNext())
                delete fixture->GetUserData();

            m_World->getB2World()->DestroyBody(body);
            registry->destroy(raycastComp.collisionInfo.entityBelow);
            m_Context->enttToBody.erase(raycastComp.collisionInfo.
                entityBelow);

            velocity.y = 0;
            velocity.y -= 5.f;
        }
    }

    if (sf::Keyboard::isKeyPressed(sf::Keyboard::Space))
    {
        if (raycastComp.collisionInfo.collisionBelow)
            velocity = { velocity.x, velocity.y - 50.f };

        //m_Context->enttToBody[entity]->ApplyLinearImpulseToCenter(
            b2Vec2(0.f, -3.f), true);
        //m_Context->enttToBody[entity]->ApplyForceToCenter(b2Vec2(0.f,
             -200.f), true);

        m_State = GameObjectState::ID::Jumping;
    }
```

```
if (velocity.y < 0 && !sf::Keyboard::isKeyPressed(sf::Keyboard::
    Space))
{
    velocity = { velocity.x, velocity.y + utils::sfVecToB2Vec(math
        ::VECTOR_DOWN).y * m_World->getB2World()->GetGravity().y * (
        m_lowJumpMultiplier - 1) * dt.asMilliseconds() };
}
velocity.x = math::lerp(velocity.x, velocity.x, 0.8f);

if (!raycastComp.collisionInfo.collisionBelow)
    m_State = GameObjectState::ID::Jumping;

if ((velocity.x >= -0.1f || velocity.x <= 0.1f) && raycastComp.
    collisionInfo.collisionBelow)
    m_State = GameObjectState::ID::Standing;

if ((velocity.x < -0.1f || velocity.x > 0.1f) && raycastComp.
    collisionInfo.collisionBelow)
    m_State = GameObjectState::ID::Walking;


if (sf::Keyboard::isKeyPressed(sf::Keyboard::W))
{
    if (raycastComp.collisionInfo.climbingLadder)
    {
        velocity.y = 0.f;
        velocity.y = -2.f;
        m_State = GameObjectState::ID::ClimbingLadder;
    }
}

velocity.x = std::clamp(velocity.x, -5.f, 5.f);
velocity.y = std::clamp(velocity.y, -10.f, 10.f);
m_Context->enttToBody[entity]->SetLinearVelocity(velocity);

if (velocity.x < -0.1f)
    anim.animatedSprite.setScale(sf::Vector2f(-1.f, 1.f));
else if (velocity.x > 0.1f)
    anim.animatedSprite.setScale(sf::Vector2f(1.f, 1.f));

switch (m_State)
{
case GameObjectState::ID::Standing:
    anim.animatedSprite.play(Animations::ID::Standing);
    break;

case GameObjectState::ID::Walking:
    anim.animatedSprite.play(Animations::ID::Walking);
    break;

case GameObjectState::ID::Jumping:
```

```
            anim.animatedSprite.play(Animations::ID::Jumping);
            break;

        case GameObjectState::ID::ClimbingLadder:
            anim.animatedSprite.play(Animations::ID::ClimbingLadder);
            break;
        }
    }
}
```

RenderSystem.h

```
#pragma once

#include "BaseSystem.hpp"
#include "../Utils/SFMLDebugDraw.h"

#include <entt/entity/entity.hpp>

struct C_Rigidbody;

class RenderSystem : public BaseSystem
{
public:
    RenderSystem() = default;
    explicit RenderSystem(Context& context, World *world);

    void draw() override;

private:
    void drawDebugInfo(entt::entity& entity, C_Rigidbody& rb);

    SFMLDebugDraw debugDraw;
};
```

RenderSystem.cpp

```
#include "RenderSystem.hpp"
#include "../Components/C_Animation.hpp"
#include "../Components/C_Rigidbody.hpp"
#include "../Components/C_Raycast.hpp"
#include "../Components/C_Tilemap.hpp"
#include "../Components/C_Tag.h"
#include "../Utils/Context.hpp"
#include "../Utils/Math.hpp"
#include "../Utils/FixtureUserData.hpp"
#include "../core/World.h"

#include <vector>
```

```
RenderSystem::RenderSystem(Context& context, World *world) :
BaseSystem(context, world),
debugDraw()
{
    // Initialize SFML Debug Draw
    debugDraw.SetWindow(m_Context->window);
    debugDraw.ClearFlags(debugDraw.e_aabbBit & debugDraw.e_centerOfMassBit
        &
                         debugDraw.e_jointBit & debugDraw.e_pairBit &
                             debugDraw.e_shapeBit);
    debugDraw.SetFlags( debugDraw.e_jointBit | debugDraw.e_shapeBit); //
        Debug draw shapes and joints only
    m_World->getB2World()->SetDebugDraw(&debugDraw);
}


void RenderSystem::draw()
{
    m_World->getEntityRegistry()->view<C_Tilemap>().each([&](auto entity,
        auto& tilemap)
    {
        for(const auto &l : tilemap.m_TileLayers)
            m_Context->window->draw(*l);
    });

    m_World->getEntityRegistry()->view<C_Animation>().each([&](auto entity,
         auto &anim)
    {
        m_Context->window->draw(anim.animatedSprite);
    });

    if (m_World->b2dDebugging())
        m_World->getB2World()->DrawDebugData();

    if (m_World->sfmlDebugging())
    {
        m_World->getEntityRegistry()->view<C_Rigidbody>().each([&](auto
            entity, auto& rb)
            {
                drawDebugInfo(entity, rb);
            });
    }
}

void RenderSystem::drawDebugInfo(entt::entity& entity, C_Rigidbody& rb)
{
    for(b2Fixture* fixture = m_Context->enttToBody[entity]->GetFixtureList
        (); fixture != nullptr; fixture = fixture->GetNext())
    {
        auto userData = static_cast<FixtureUserData*>(fixture->GetUserData
            ());
        if (userData == nullptr || userData->shape == nullptr) continue;
```

```
        m_Context->window->draw(*userData->shape);

        if(m_World->getEntityRegistry()->has<C_Raycast>(entity))
        {
            auto &raycastComp = m_World->getEntityRegistry()->get<C_Raycast
                >(entity);

            for (auto& ray : raycastComp.raycasts)
            {
                sf::Vertex line[2];
                line[0].position = ray[0].position;
                line[1].position = ray[1].position;
                line[0].color = ray[0].color;
                line[1].color = ray[1].color;
                m_Context->window->draw(line, 2, sf::Lines);
            }
        }
    }
}
```

TilemapSystem.h

```
#pragma once

#include "BaseSystem.hpp"

class TilemapSystem : public BaseSystem
{
public:
    TilemapSystem() = default;
    explicit TilemapSystem(Context &context, World *world);

    void update(sf::Time &dt) override;
};
```

TilemapSystem.cpp

```
#include "TilemapSystem.hpp"

#include "../Utils/Context.hpp"
#include "../Utils/BodyCreator.h"
#include "../Utils/FixtureUserData.hpp"
#include "../Components/C_Tilemap.hpp"
#include "../Components/C_Rigidbody.hpp"
#include "../Components/C_Tag.h"
#include "../core/World.h"

#include <tmxlite/Layer.hpp>
#include <tmxlite/Map.hpp>
```

```
TilemapSystem::TilemapSystem(Context &context, World *world) : BaseSystem(
    context, world)
{
    m_World->getEntityRegistry()->view<C_Tilemap>().each([&](auto entity,
        auto& tilemapComp)
    {
        tmx::Map map;
        map.load(tilemapComp.m_MapPath);
        auto& layers = map.getLayers();

        size_t i = 0;
        for (const auto& l : layers)
        {
            if (l->getType() == tmx::Layer::Type::Tile)
            {
                tilemapComp.m_TileLayers.push_back(std::make_unique<
                    MapLayer>(map, i));
                ++i;
            }
            else if (l->getType() == tmx::Layer::Type::Object)
            {
                auto objLayer = l->template getLayerAs<tmx::ObjectGroup>();
                tilemapComp.m_ObjectLayers.push_back(std::make_unique<tmx::
                    ObjectGroup>(objLayer));
                if (l->getName() == "Static Object Layer")
                {
                    const auto& objects = l->getLayerAs<tmx::ObjectGroup>()
                        .getObjects();
                    for (const auto& obj : objects)
                    {
                        auto entity = m_World->getEntityRegistry()->create
                            ();
                        m_Context->enttToBody[entity] = BodyCreator::
                            createBody(*m_World->getB2World(), obj,
                            b2BodyType::b2_staticBody);

                        m_World->getEntityRegistry()->emplace<C_Rigidbody>(
                            entity, m_Context->enttToBody[entity]);

                        FixtureUserData* userData = new FixtureUserData();
                        userData->entity = entity;
                        m_Context->enttToBody[entity]->GetFixtureList()->
                            SetUserData(userData);
                    }
                }
                else if (l->getName() == "Dynamic Object Layer")
                {
                    auto objects = l->getLayerAs<tmx::ObjectGroup>().
                        getObjects();
                    for (const auto& obj : objects)
                    {
```

```
                auto entity = m_World->getEntityRegistry()->create
                    ();
                m_Context->enttToBody[entity] = BodyCreator::
                    createBody(*m_World->getB2World(), obj,
                    b2BodyType::b2_dynamicBody);

                m_World->getEntityRegistry()->emplace<C_Rigidbody>(
                    entity, m_Context->enttToBody[entity]);

                FixtureUserData* userData = new FixtureUserData();
                userData->entity = entity;
                m_Context->enttToBody[entity]->GetFixtureList()->
                    SetUserData(userData);
            }
        }

        else if (l->getName() == "Kinematic Object Layer")
        {
            auto objects = l->getLayerAs<tmx::ObjectGroup>().
                getObjects();
            for (const auto& obj : objects)
            {
                auto entity = m_World->getEntityRegistry()->create
                    ();
                m_Context->enttToBody[entity] = BodyCreator::
                    createBody(*m_World->getB2World(), obj,
                    b2BodyType::b2_kinematicBody);

                m_World->getEntityRegistry()->emplace<C_Rigidbody>(
                    entity, m_Context->enttToBody[entity]);

                FixtureUserData* userData = new FixtureUserData();
                userData->entity = entity;
                m_Context->enttToBody[entity]->GetFixtureList()->
                    SetUserData(userData);

                if (obj.getName().find("Platform") != std::string::
                    npos)
                {
                    m_World->getEntityRegistry()->emplace<
                        C_OneWayPlatform>(entity);
                    b2Filter filter;
                    filter.categoryBits = BodyCategory::
                        OneWayPlatform;
                    filter.maskBits = BodyCategory::Enemy |
                        BodyCategory::Player;
                    m_Context->enttToBody[entity]->GetFixtureList()
                        ->SetFilterData(filter);
                }

                if (obj.getName().find("Spikes") != std::string::
                    npos)
```

```
                        {
                            m_World->getEntityRegistry()->emplace<C_Spikes
                                >(entity);
                        }

                        if (obj.getName().find("Ladder") != std::string::
                            npos)
                        {
                            m_Context->enttToBody[entity]->GetFixtureList()
                                ->SetSensor(true);
                            m_World->getEntityRegistry()->emplace<C_Ladder
                                >(entity);
                            b2Filter filter;
                            filter.categoryBits = BodyCategory::Ladder;
                            filter.maskBits = BodyCategory::Player;
                            m_Context->enttToBody[entity]->GetFixtureList()
                                ->SetFilterData(filter);
                        }
                    }
                }
            }
        }
    });
}

void TilemapSystem::update(sf::Time &dt)
{
    m_World->getEntityRegistry()->view<C_Tilemap>().each([&](auto entity,
        auto &tilemapComp)
    {
        for(const auto &l : tilemapComp.m_TileLayers)
            l->update(dt);
    });
}
```

State.h

```
#pragma once
#include <memory>

#include "Utils/Utility.hpp"
#include "../Utils/Context.hpp"

class StateStack;

class State
{
public:
    using Ptr = std::unique_ptr<State>;

public:
```

```
    State(StateStack& stack, Context context);
    virtual ~State() {}

    virtual void draw() = 0;
    virtual bool update(sf::Time dt) = 0;
    virtual bool handleEvent(const sf::Event& event) = 0;

protected:
    void requestStackPush(States::ID stateID) const;
    void requestStackPop() const;
    void requestStackClear() const;

private:
    StateStack* m_Stack;

protected:
    Context m_Context;
};
```

State.cpp

```
#include "State.h"
#include "StateStack.h"

#include <utility>


State::State(StateStack& stack, Context context) :
    m_Stack(&stack),
    m_Context(std::move(context))
{
}

void State::requestStackPush(States::ID stateID) const
{
    m_Stack->pushState(stateID);
}

void State::requestStackPop() const
{
    m_Stack->popState();
}

void State::requestStackClear() const
{
    m_Stack->clearStates();
}
```

StateStack.h

```
#pragma once
#include <vector>
#include <map>
#include <functional>
#include <cassert>

#include <SFML/System/Time.hpp>
#include <SFML/System/NonCopyable.hpp>

#include "State.h"
#include "../Utils/Context.hpp"
#include "../Utils/Utility.hpp"

class StateStack : private sf::NonCopyable
{
public:
    enum Action
    {
        Push,
        Pop,
        Clear
    };

public:
    StateStack() = default;
    explicit StateStack(Context context);

    template<typename T>
    void registerState(States::ID stateID);

    void update(sf::Time dt);
    void draw();
    void handleEvent(const sf::Event& event);

    void pushState(States::ID stateID);
    void popState();
    void clearStates();

    bool isEmpty() const;

private:
    State::Ptr createState(States::ID stateID);
    void applyPendingChange();

private:
    struct PendingChange
    {
        PendingChange(Action action, States::ID stateID = States::ID::None)
            ;

        Action action;
        States::ID stateID;
```

```
    };

private:
    std::vector<State::Ptr> m_Stack{};
    std::vector<PendingChange> m_PendingList{};
    Context m_Context{};
    std::map<States::ID, std::function<State::Ptr()>> m_Factories{};
};

template<typename T>
void StateStack::registerState(States::ID stateID)
{
    m_Factories[stateID] = [this]()
    {
        return State::Ptr(new T(*this, m_Context));
    };
}
```

StateStack.cpp

```
#include "StateStack.h"

#include <utility>

StateStack::StateStack(Context context) :
    m_Stack(),
    m_PendingList(),
    m_Context(std::move(context)),
    m_Factories()
{
}

State::Ptr StateStack::createState(States::ID stateID)
{
    const auto found = m_Factories.find(stateID);

    assert(found != m_Factories.end());

    return found->second();
}

void StateStack::applyPendingChange()
{
    for (auto change : m_PendingList)
    {
        switch (change.action)
        {
        case Action::Push:
            m_Stack.push_back(createState(change.stateID));
            break;
```

```
        case Action::Pop:
            m_Stack.pop_back();
            break;

        case Action::Clear:
            m_Stack.clear();
            break;

        default:
            break;
        }
    }

    m_PendingList.clear();
}

void StateStack::update(sf::Time dt)
{
    for (auto iter = m_Stack.rbegin(); iter != m_Stack.rend(); ++iter)
    {
        if (!(*iter)->update(dt))
            break;
    }

    applyPendingChange();
}

void StateStack::draw()
{
    for (auto& state : m_Stack)
        state->draw();
}

void StateStack::handleEvent(const sf::Event& event)
{
    for (auto iter = m_Stack.rbegin(); iter != m_Stack.rend(); ++iter)
    {
        if (!(*iter)->handleEvent(event))
            break;
    }

    applyPendingChange();
}

void StateStack::pushState(States::ID stateID)
{
    m_PendingList.emplace_back(Push, stateID);
}

void StateStack::popState()
{
    m_PendingList.emplace_back(Pop);
```

```
}

void StateStack::clearStates()
{
    m_PendingList.emplace_back(Clear);
}

bool StateStack::isEmpty() const
{
    return m_Stack.empty();
}

StateStack::PendingChange::PendingChange(Action action, States::ID stateID)
    :
    action(action), stateID(stateID)
{
}
```

GameState.h

```
#pragma once
#include "../States/State.h"
#include "../core/World.h"

class StateStack;

class GameState : public State
{
public:
    GameState(StateStack &stack, Context& context);

    void draw() override;
    bool update(sf::Time dt) override;
    bool handleEvent(const sf::Event& event) override;

private:
    World m_World;

    b2MouseJoint* mouseJoint;
    b2Body* ground;

    sf::Text youLoseLabel;
    sf::Text victoryLabel;
};
```

GameState.cpp

```
#include "GameState.h"
#include "StateStack.h"
#include "../Utils/QueryCallback.h"
```

```
#include "../Components/C_Rigidbody.hpp"
#include "../core/MusicPlayer.hpp"
#include "../core/SoundEffectPlayer.hpp"

#include <SFML/Window/Event.hpp>

GameState::GameState(StateStack& stack, Context& context) :
    State(stack, context),
    m_World(context)
{
    auto m_B2World = m_World.getB2World();

    /* Mouse Joint */
    mouseJoint = nullptr;
    b2BodyDef bodyDef;
    ground = m_B2World->CreateBody(&bodyDef);

    youLoseLabel.setString("YOU LOSE");
    youLoseLabel.setCharacterSize(60u);
    youLoseLabel.setFont(m_Context.fonts->get(Fonts::ID::ARJULIAN));
    youLoseLabel.setOutlineThickness(3.f);
    youLoseLabel.setOutlineColor(sf::Color::Black);
    utils::centerOrigin(youLoseLabel);

    victoryLabel.setString("VICTORY");
    victoryLabel.setCharacterSize(60u);
    victoryLabel.setFont(m_Context.fonts->get(Fonts::ID::ARJULIAN));
    victoryLabel.setOutlineThickness(3.f);
    victoryLabel.setOutlineColor(sf::Color::Black);
    utils::centerOrigin(victoryLabel);

    m_Context.music->play(Music::ID::GameTheme);
}

void GameState::draw()
{
    m_World.draw();

    if (m_World.getRemainingTime() <= 0 || m_World.GameOver())
        m_Context.window->draw(youLoseLabel);
    if (m_World.getNumberOfEnemies() == 0)
        m_Context.window->draw(victoryLabel);
}

bool GameState::update(const sf::Time dt)
{
    if (m_World.getNumberOfEnemies() == 0)
    {
        victoryLabel.setPosition(m_Context.window->getView().getCenter());
        victoryLabel.setScale(m_Context.window->getView().getSize().x /
            m_Context.window->getDefaultView().getSize().x,
```

```
                                m_Context.window->getView().getSize().y /
                                    m_Context.window->getDefaultView().getSize
                                    ().y);

            return false;
        }
        if (m_World.getRemainingTime() <= 0 || m_World.GameOver())
        {
            /*requestStackPop();
            requestStackPush(States::Game);*/

            youLoseLabel.setPosition(m_Context.window->getView().getCenter());
            youLoseLabel.setScale(m_Context.window->getView().getSize().x /
                m_Context.window->getDefaultView().getSize().x,
                                m_Context.window->getView().getSize().y /
                                    m_Context.window->getDefaultView().getSize
                                    ().y);

            return false;
        }

        m_World.update(dt);

        return true;
}

bool GameState::handleEvent(const sf::Event& event)
{
        m_World.handleEvents(event);

        if (event.type == sf::Event::KeyPressed && event.key.code == sf::
            Keyboard::P && m_World.getNumberOfEnemies() != 0)
        {
            m_World.spawnEnemy();
        }

        if (event.type == sf::Event::KeyPressed && event.key.code == sf::
            Keyboard::R)
        {
            requestStackPop();
            requestStackPush(States::ID::Game);
        }

        if (event.type == sf::Event::KeyPressed && event.key.code == sf::
            Keyboard::Num1)
        {
            m_World.sfmlDebugging() = !m_World.sfmlDebugging();
        }

        if (event.type == sf::Event::KeyPressed && event.key.code == sf::
            Keyboard::Num2)
        {
```

```
        m_World.b2dDebugging() = !m_World.b2dDebugging();
    }

    if (((event.type == sf::Event::KeyPressed && (event.key.code == sf::
        Keyboard::BackSpace || event.key.code == sf::Keyboard::Escape))) &&
        (m_World.getRemainingTime() <= 0 || m_World.getNumberOfEnemies() ==
            0))
    {
        requestStackPop();
        requestStackPush(States::ID::Menu);
        m_Context.isPaused = false;
    }
    else if (event.type == sf::Event::KeyPressed && event.key.code == sf::
        Keyboard::Escape)
    {
        m_Context.sounds->play(Sounds::ID::Pause);
        requestStackPush(States::ID::Pause);
        m_Context.isPaused = true;
    }

    auto m_B2World = m_World.getB2World();

    // Following three events are copied almost completely from http://code
        .google.com/p/box2d/source/browse/trunk/Box2D/Testbed/Framework/Test
        .cpp
    // Copyright (c) 2011 Erin Catto http://box2d.org
    if (event.type == sf::Event::MouseButtonPressed && event.mouseButton.
        button == sf::Mouse::Left && mouseJoint == nullptr)
    {
        b2Vec2 mousePos = utils::sfVecToB2Vec(m_Context.window->
            mapPixelToCoords(sf::Mouse::getPosition(*m_Context.window)));

        // Make a small box.
        b2AABB aabb;
        b2Vec2 d;
        d.Set(0.001f, 0.001f);
        aabb.lowerBound = mousePos + d;
        aabb.upperBound = mousePos - d;

        // Query the world for overlapping shapes.
        QueryCallback callback(mousePos);
        m_B2World->QueryAABB(&callback, aabb);

        if (callback.m_fixture)
        {
            b2Body* body = callback.m_fixture->GetBody();
            b2MouseJointDef md;
            md.bodyA = ground; //If bounding box body is used instead, the
                dynamic bodies can be dragged over the bounding box and we
                don't want that
            md.bodyB = body;
            md.target = mousePos;
```

```
            md.maxForce = 1000.0f * body->GetMass();
            mouseJoint = (b2MouseJoint*)m_B2World->CreateJoint(&md);
            body->SetAwake(true);
            body->SetBullet(true);
        }
    }
    else if (event.type == sf::Event::MouseMoved && mouseJoint != nullptr)
    {
        b2Vec2 mousePos = utils::sfVecToB2Vec(m_Context.window->
            mapPixelToCoords(sf::Mouse::getPosition(*m_Context.window)));
        mouseJoint->SetTarget(mousePos);
    }
    else if (event.type == sf::Event::MouseButtonReleased && event.
        mouseButton.button == sf::Mouse::Left && mouseJoint != nullptr)
    {
        mouseJoint->GetBodyB()->SetBullet(false);
        m_B2World->DestroyJoint(mouseJoint);
        mouseJoint = nullptr;
    }

    return true;
}
```

PauseState.h

```
#pragma once

#include "State.h"

class PauseState : public State
{
public:
    PauseState(StateStack& stack, Context& context);
    ~PauseState();

    void draw() override;
    bool update(sf::Time dt) override;
    bool handleEvent(const sf::Event& event) override;

private:
    sf::Text m_Text;
};
```

PauseState.cpp

```
#include "PauseState.hpp"
#include "../Utils/Utility.hpp"
#include "../utils/Context.hpp"
#include "../core/MusicPlayer.hpp"
```

```
#include <SFML/Window/Event.hpp>

PauseState::PauseState(StateStack& stack, Context& context) :
    State(stack, context)
{
    m_Text.setFont(m_Context.fonts->get(Fonts::ID::ARJULIAN));
    m_Text.setString("Paused");
    m_Text.setCharacterSize(20u);
    m_Text.setOutlineThickness(3.f);
    utils::centerOrigin(m_Text);

    m_Context.music->setPaused(true);
}

PauseState::~PauseState()
{
    m_Context.music->setPaused(false);
}

void PauseState::draw()
{
    sf::RectangleShape background;
    background.setFillColor(sf::Color(0, 0, 0, 150));
    background.setSize(sf::Vector2f(m_Context.window->getSize()));

    m_Context.window->draw(background);
    m_Context.window->draw(m_Text);
}

bool PauseState::update(sf::Time dt)
{
    m_Text.setPosition(m_Context.window->getView().getCenter());
    m_Text.setScale(m_Context.window->getView().getSize().x / m_Context.
        window->getDefaultView().getSize().x,
                    m_Context.window->getView().getSize().y / m_Context.
                        window->getDefaultView().getSize().y);

    return false;
}

bool PauseState::handleEvent(const sf::Event& event)
{
    if (event.type != sf::Event::KeyPressed)
        return false;

    if (event.key.code == sf::Keyboard::Escape)
    {
        requestStackPop();
        m_Context.isPaused = false;
    }

    if (event.key.code == sf::Keyboard::BackSpace)
```

```
    {
        requestStackClear();
        requestStackPush(States::ID::Menu);
    }

    return false;
}
```

MenuState.h

```
#pragma once

#include "State.h"
#include <vector>

class MenuState : public State
{
public:
    MenuState(StateStack& stack, Context& context);

    void draw() override;
    bool update(sf::Time dt) override;
    bool handleEvent(const sf::Event& event) override;

private:
    void updateOptionsText();


private:
    enum OptionNames
    {
        Play,
        Exit,
    };

    std::vector<sf::Text> m_Options;
    std::size_t optionIndex;

    sf::Sprite      m_BackgroundSprite;
};
```

MenuState.cpp

```
#include "MenuState.hpp"
#include "../Utils/Utility.hpp"
#include "../Components/C_Camera.hpp"
#include "../core/MusicPlayer.hpp"

#include <SFML/Window/Event.hpp>
```

```
MenuState::MenuState(StateStack& stack, Context& context) :
    State(stack, context),
    m_Options(),
    optionIndex(0)
{
    auto& font = context.fonts->get(Fonts::ID::ARJULIAN);

    sf::Text playText;
    playText.setFont(font);
    playText.setString("Play");
    utils::centerOrigin(playText);
    m_Options.push_back(playText);

    sf::Text exitText;
    exitText.setFont(font);
    exitText.setString("Exit");
    utils::centerOrigin(exitText);
    m_Options.push_back(exitText);

    updateOptionsText();

    m_Context.music->play(Music::ID::MenuTheme);

    auto windowSize = m_Context.window->getSize();
    m_BackgroundSprite.setTexture(m_Context.textures->get(Textures::ID::
        MainBackground));
    m_BackgroundSprite.setScale(sf::Vector2f(4.5f, 4.5f));
    utils::centerOrigin(m_BackgroundSprite);
    m_BackgroundSprite.setPosition(windowSize.x / 2.f, windowSize.y / 2.2f)
        ;
}

void MenuState::draw()
{
    m_Context.window->setView(m_Context.window->getDefaultView());

    m_Context.window->draw(m_BackgroundSprite);

    for (const auto& text : m_Options)
        m_Context.window->draw(text);
}

bool MenuState::update(sf::Time dt)
{
    for (std::size_t i = 0; i < m_Options.size(); ++i)
    {
        m_Options.at(i).setPosition(m_Context.window->getView().getCenter()
            - sf::Vector2f(15.f, 15.f) + sf::Vector2f(0, i * 40.f));
    }

    return true;
```

```
}

bool MenuState::handleEvent(const sf::Event& event)
{
    // The demonstration menu logic
    if (event.type != sf::Event::KeyPressed)
        return false;

    if (event.key.code == sf::Keyboard::Return)
    {
        if (optionIndex == Play)
        {
            requestStackPop();
            requestStackPush(States::ID::Game);
        }
        else if (optionIndex == Exit)
        {
            // The exit option was chosen, by removing itself, the stack
                will be empty, and the game will know it is time to close.
            requestStackPop();
        }
    }

    else if (event.key.code == sf::Keyboard::Up)
    {
        // Decrement and wrap-around
        if (optionIndex > 0)
            optionIndex--;
        else
            optionIndex = m_Options.size() - 1;

        updateOptionsText();
    }

    else if (event.key.code == sf::Keyboard::Down)
    {
        // Increment and wrap-around
        if (optionIndex < m_Options.size() - 1)
            optionIndex++;
        else
            optionIndex = 0;

        updateOptionsText();
    }

    return true;
}

void MenuState::updateOptionsText()
{
    if (m_Options.empty())
        return;
```

```
    // White color for all text options
    for (auto& text : m_Options)
        text.setFillColor(sf::Color::White);

    // Red color for selected option
    m_Options[optionIndex].setFillColor(sf::Color::Red);
}
```

SplashScreenState.h

```
#pragma once

#include "State.h"
#include <SFML/Graphics/Text.hpp>
#include <SFML/Graphics/Sprite.hpp>
#include "../Utils/ResourceHolder.h"

class SplashScreenState : public State
{
public:
    SplashScreenState(StateStack& stack, Context context);

    virtual void    draw();
    virtual bool    update(sf::Time dt);
    virtual bool    handleEvent(const sf::Event& event);

private:
    sf::Sprite      m_BackgroundSprite;
    sf::Text        m_Text;

    bool            m_ShowText;
    sf::Time        m_TextEffectTime;
};
```

SplashScreenState.cpp

```
#include "SplashScreenState.hpp"
#include "../Utils/Utility.hpp"

#include <SFML/Window/Event.hpp>

SplashScreenState::SplashScreenState(StateStack& stack, Context context) :
    State(stack, context),
    m_BackgroundSprite(),
    m_Text(),
    m_ShowText(true),
    m_TextEffectTime(sf::Time::Zero)
{
    m_Text.setCharacterSize(50u);
```

```
    m_Text.setFont(context.fonts->get(Fonts::ID::ARJULIAN));
    m_Text.setString("Press any key to continue");
    utils::centerOrigin(m_Text);
    sf::Vector2f windowSize(context.window->getSize().x, context.window->
        getSize().y);
    m_Text.setPosition(windowSize.x / 2.f, windowSize.y / 1.1f);

    m_BackgroundSprite.setTexture(m_Context.textures->get(Textures::ID::
        MainBackground));
    m_BackgroundSprite.setScale(sf::Vector2f(4.5f, 4.5f));
    utils::centerOrigin(m_BackgroundSprite);
    m_BackgroundSprite.setPosition(windowSize.x / 2.f, windowSize.y / 2.2f)
        ;
}

void SplashScreenState::draw()
{
    m_Context.window->draw(m_BackgroundSprite);

    if (m_ShowText)
        m_Context.window->draw(m_Text);
}

bool SplashScreenState::update(sf::Time dt)
{
    m_TextEffectTime += dt;

    if (m_TextEffectTime > sf::seconds(0.5f))
    {
        m_ShowText = !m_ShowText;
        m_TextEffectTime = sf::Time::Zero;
    }

    return true;
}

bool SplashScreenState::handleEvent(const sf::Event& event)
{
    if (event.type == sf::Event::KeyPressed)
    {
        requestStackPop();
        requestStackPush(States::ID::Menu);
    }

    return true;
}
```

AnimatedSprite.h

```
#pragma once
#include <unordered_map>
```

```
#include <SFML/Graphics/RenderTarget.hpp>
#include <SFML/System/Time.hpp>
#include <SFML/Graphics/Drawable.hpp>
#include <SFML/Graphics/Transformable.hpp>
#include <SFML/Graphics/Sprite.hpp>
#include <SFML/System/Vector2.hpp>

#include "Animation.h"
#include "Utility.hpp"

class AnimatedSprite : public sf::Drawable, public sf::Transformable
{
public:
    explicit AnimatedSprite(sf::Time frameTime = sf::seconds(0.2f), bool
        paused = false, bool looped = true);

    void update(sf::Time deltaTime);
    void setAnimation(const Animations::ID id);
    void setFrameTime(sf::Time time);
    void addAnimation(Animations::ID id, Animation* animation);
    void play();
    void play(const Animations::ID id);
    void pause();
    void stop();
    void setLooped(bool looped);
    void setColor(const sf::Color& color);
    const Animation* getAnimation() const;
    sf::FloatRect getLocalBounds() const;
    sf::FloatRect getGlobalBounds() const;
    bool isLooped() const;
    bool isPlaying() const;
    sf::Time getFrameTime() const;
    void setFrame(std::size_t newFrame, bool resetTime = true);

private:
    std::unordered_map<Animations::ID, Animation*> animations;
    const Animation* m_animation;
    sf::Time m_frameTime;
    sf::Time m_currentTime;
    std::size_t m_currentFrame;
    bool m_isPaused;
    bool m_isLooped;
    sf::Sprite m_Sprite;

    void draw(sf::RenderTarget& target, sf::RenderStates states) const
        override;
};
```

AnimatedSprite.cpp

```
#include "AnimatedSprite.h"

#include <utility>
#include <spdlog/spdlog.h>

AnimatedSprite::AnimatedSprite(sf::Time frameTime, bool paused, bool looped
    ) :
animations(),m_animation(nullptr), m_frameTime(frameTime),
m_currentFrame(0), m_isPaused(paused), m_isLooped(looped),
m_Sprite()
{

}

void AnimatedSprite::setAnimation(const Animations::ID id)
{
    m_animation = animations[id];
    m_Sprite.setTexture(*m_animation->getSpriteSheet());
    m_currentFrame = 0;
    setFrame(m_currentFrame);
}

void AnimatedSprite::setFrameTime(sf::Time time)
{
    m_frameTime = time;
}
void AnimatedSprite::addAnimation(Animations::ID id, Animation *animation)
{
    animations[id] = animation;
}

void AnimatedSprite::play()
{
    m_isPaused = false;
}

void AnimatedSprite::play(const Animations::ID id)
{
    if (getAnimation() != animations[id])
    {
        setAnimation(id);
    }
    play();
}

void AnimatedSprite::pause()
{
    m_isPaused = true;
}

void AnimatedSprite::stop()
{
```

```
    m_isPaused = true;
    m_currentFrame = 0;
    setFrame(m_currentFrame);
}

void AnimatedSprite::setLooped(bool looped)
{
    m_isLooped = looped;
}

void AnimatedSprite::setColor(const sf::Color& color)
{
    m_Sprite.setColor(color);
}

const Animation* AnimatedSprite::getAnimation() const
{
    return m_animation;
}

sf::FloatRect AnimatedSprite::getLocalBounds() const
{
    return m_Sprite.getLocalBounds();
}

sf::FloatRect AnimatedSprite::getGlobalBounds() const
{
    return m_Sprite.getGlobalBounds();
}

bool AnimatedSprite::isLooped() const
{
    return m_isLooped;
}

bool AnimatedSprite::isPlaying() const
{
    return !m_isPaused;
}

sf::Time AnimatedSprite::getFrameTime() const
{
    return m_frameTime;
}

void AnimatedSprite::setFrame(std::size_t newFrame, bool resetTime)
{
    if (m_animation)
    {
        sf::IntRect rect = m_animation->getFrame(newFrame);
        m_Sprite.setTextureRect(rect);
    }
```

```
    if (resetTime)
        m_currentTime = sf::Time::Zero;
}

void AnimatedSprite::update(sf::Time deltaTime)
{
    // if not paused and we have a valid animation
    if (!m_isPaused && m_animation)
    {
        // add delta time
        m_currentTime += deltaTime;

        // if current time is bigger then the frame time advance one frame
        if (m_currentTime >= m_frameTime)
        {
            // reset time, but keep the remainder
            m_currentTime = sf::microseconds(m_currentTime.asMicroseconds()
                % m_frameTime.asMicroseconds());

            // get next Frame index
            if (m_currentFrame + 1 < m_animation->getSize())
                m_currentFrame++;
            else
            {
                // animation has ended
                m_currentFrame = 0; // reset to start

                if (!m_isLooped)
                {
                    m_isPaused = true;
                }

            }

            // set the current frame, not reseting the time
            setFrame(m_currentFrame, false);
        }
    }
}

void AnimatedSprite::draw(sf::RenderTarget& target, sf::RenderStates states
    ) const
{
    if (m_animation)
    {
        states.transform *= getTransform();
        target.draw(m_Sprite, states);
    }
}
```

Animation.h

```
#pragma once
#include <vector>

#include <SFML/Graphics/Rect.hpp>
#include <SFML/Graphics/Texture.hpp>

class Animation
{
public:
    Animation();

    void addFrame(sf::IntRect rect);
    void setSpriteSheet(const sf::Texture& texture);
    const sf::IntRect& getFrame(std::size_t n) const;
    std::size_t getSize() const;
    const sf::Texture* getSpriteSheet() const;

private:
    std::vector<sf::IntRect> m_frames;
    const sf::Texture* m_texture;
};
```

Animation.cpp

```
#include "Animation.h"
#include <assert.h>

Animation::Animation() : m_texture(nullptr)
{

}

void Animation::addFrame(sf::IntRect rect)
{
    m_frames.push_back(rect);
}

void Animation::setSpriteSheet(const sf::Texture& texture)
{
    m_texture = &texture;
}

const sf::Texture* Animation::getSpriteSheet() const
{
    return m_texture;
}

std::size_t Animation::getSize() const
{
```

```
    return m_frames.size();
}

const sf::IntRect& Animation::getFrame(std::size_t n) const
{
    //assert(m_frames.size() == 0 && n > 0);
    return m_frames[n];
}
```

BodyCreator.h

```
/*********************************************************************
Matt Marchant 2013 - 2016
SFML Tiled Map Loader - https://github.com/bjorn/tiled/wiki/TMX-Map-Format
http://trederia.blogspot.com/2013/05/tiled-map-loader-for-sfml.html

Some of the hull decimation code is a based on an ActionScript class
by Antoan Angelov. See:
http://www.emanueleferonato.com/2011/09/12/create-non-convex-complex-shapes
    -with-box2d/

Zlib License:

This software is provided 'as-is', without any express or
implied warranty. In no event will the authors be held
liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose,
including commercial applications, and to alter it and redistribute
it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented;
you must not claim that you wrote the original software.
If you use this software in a product, an acknowledgment
in the product documentation would be appreciated but
is not required.

2. Altered source versions must be plainly marked as such,
and must not be misrepresented as being the original software.

3. This notice may not be removed or altered from any
source distribution.
*********************************************************************/

/*********************************************************************
 ************************** CAUTION ********************************

 ******* This is an altered version of the original software *********

*********************************************************************/
```

```
#ifndef TMXTEST_BODYCREATOR_H
#define TMXTEST_BODYCREATOR_H

#include <vector>
#include <memory>

#include <SFML/System/Vector2.hpp>
#include <Box2D/Box2D.h>
#include <tmxlite/Object.hpp>

#include "../Utils/Utility.hpp"


struct BodyCreator
{
    static float getWinding(const sf::Vector2f &p1, const sf::Vector2f &p2,
        const sf::Vector2f &p3)
    {
        return p1.x * p2.y + p2.x * p3.y + p3.x * p1.y - p1.y * p2.x - p2.y
            * p3.x - p3.y * p1.x;
    }

    static void createFixture(const std::vector<sf::Vector2<float>> &points
        , b2Body *body)
    {
        bool clockwise = (getWinding(points[0], points[1], points[2]) > 0.f
            );
        size_t s = points.size();
        std::unique_ptr<b2Vec2[]> vertices(new b2Vec2[s]);
        for (auto i = 0u; i < s; i++)
        {
            //reverse direction if verts wound clockwise
            int index = (clockwise) ? (s - 1) - i : i;
            vertices[i] = utils::sfVecToB2Vec(points[index]);
        }

        b2PolygonShape ps;
        ps.Set(vertices.get(), points.size());
        b2FixtureDef f;
        f.density = body->GetType() == b2BodyType::b2_dynamicBody ? 1.f :
            0.f;
        f.shape = &ps;
        f.filter.categoryBits = BodyCategory::Other;

        body->CreateFixture(&f);
    }

    static b2Body *createBody(b2World &world, const tmx::Object &object,
        b2BodyType bodyType)
    {
        tmx::Object::Shape objShape = object.getShape();
```

```
        auto halfObjSize = utils::sfVecToB2Vec(object.getAABB().width / 2.f
             , object.getAABB().height /2.f);
        auto objPos = utils::sfVecToB2Vec(object.getPosition().x, object.
            getPosition().y) + halfObjSize;
        auto objSize = utils::sfVecToB2Vec(object.getAABB().width, object.
            getAABB().height);

        std::vector<sf::Vector2f> shape{};
        for (const auto &point : object.getPoints())
        {
            shape.emplace_back(point.x, point.y);
        }
        auto pointsCount = object.getPoints().size();

        b2BodyDef bodyDef;
        bodyDef.type = bodyType;
        bodyDef.position.Set(objPos.x, objPos.y);
        b2Body *retBody = nullptr;
        b2FixtureDef fixtureDef;
        fixtureDef.density = bodyType == b2BodyType::b2_dynamicBody ? 1.f :
             0.f;
        fixtureDef.filter.categoryBits = BodyCategory::Other;

        if (objShape == tmx::Object::Shape::Polyline)
        {
            retBody = world.CreateBody(&bodyDef);

            if (pointsCount < 3)
            {
                b2EdgeShape edgeShape;

                edgeShape.Set(utils::sfVecToB2Vec(shape[0].x, shape[0].y),
                    utils::sfVecToB2Vec(shape[1].x, shape[1].y));

                fixtureDef.shape = &edgeShape;
                retBody->CreateFixture(&fixtureDef);
            }
            else
            {
                std::unique_ptr<b2Vec2[]> vertices(new b2Vec2[pointsCount])
                    ;
                for (auto i = 0u; i < pointsCount; i++)
                    vertices[i] = utils::sfVecToB2Vec(shape[i].x, shape[i].
                        y);

                b2ChainShape chainShape;
                chainShape.CreateChain(vertices.get(), pointsCount);

                fixtureDef.shape = &chainShape;

                retBody->CreateFixture(&fixtureDef);
            }
```

```
        }
        else if (objShape == tmx::Object::Shape::Ellipse)
        {
            b2CircleShape c;
            c.m_radius = objSize.x / 2.f;

            fixtureDef.restitution = .9f;

            fixtureDef.shape = &c;

            retBody = world.CreateBody(&bodyDef);
            retBody->CreateFixture(&fixtureDef);
        }
        else if (objShape == tmx::Object::Shape::Rectangle)
        {
            b2PolygonShape ps;
            ps.SetAsBox(objSize.x / 2.f, objSize.y / 2.f);
            fixtureDef.shape = &ps;

            retBody = world.CreateBody(&bodyDef);
            retBody->CreateFixture(&fixtureDef);
        }
        else //simple convex polygon
        {
            retBody = world.CreateBody(&bodyDef);

            createFixture(shape, retBody);
        }

        return retBody;
    }
};

#endif //TMXTEST_BODYCREATOR_H
```

ContactListener.h

```
#pragma once

#include <Box2D/Box2D.h>

class World;
struct Context;
struct FixtureUserData;

class ContactListener : public b2ContactListener
{
public:
    ContactListener() = default;
    ContactListener(Context* context, World* world);
    ~ContactListener() override = default;
```

```
    void BeginContact(b2Contact* contact) override;
    void EndContact(b2Contact* contact) override;
    void PreSolve(b2Contact* contact, const b2Manifold* oldManifold)
        override;
    void PostSolve(b2Contact* contact, const b2ContactImpulse* impulse)
        override;

    void handleOneWayPlatforms(b2Contact* contact);
    void handleCoins();

    void initMemberData(b2Contact* contact);

    Context* m_Context;
    World* m_World;

    b2Fixture* fixtureA;
    b2Fixture* fixtureB;
    FixtureUserData* userDataA;
    FixtureUserData* userDataB;
};
```

ContactListener.cpp

```
#include "ContactListener.hpp"
#include "Context.hpp"
#include "FixtureUserData.hpp"
#include "../core/World.h"
#include "../core/SoundEffectPlayer.hpp"
#include "../Components/C_Tag.h"
#include "../Components/C_Raycast.hpp"


ContactListener::ContactListener(Context* context, World* world) :
    m_Context(context), m_World(world)
{
}

void ContactListener::BeginContact(b2Contact* contact)
{
    initMemberData(contact);

    handleCoins();
}

void ContactListener::EndContact(b2Contact* contact)
{

}
```

```
void ContactListener::PreSolve(b2Contact* contact, const b2Manifold*
    oldManifold)
{
    initMemberData(contact);

    handleOneWayPlatforms(contact);
}

void ContactListener::PostSolve(b2Contact* contact, const b2ContactImpulse*
     impulse)
{

}

void ContactListener::handleOneWayPlatforms(b2Contact* contact)
{
    //check if one of the fixtures is the platform
    b2Fixture* platformFixture = nullptr;
    b2Fixture* otherFixture = nullptr;
    if (m_World->getEntityRegistry()->has<C_OneWayPlatform>(userDataA->
        entity))
    {
        platformFixture = fixtureA;
        otherFixture = fixtureB;
    }
    else if (m_World->getEntityRegistry()->has<C_OneWayPlatform>(userDataB
        ->entity))
    {
        platformFixture = fixtureB;
        otherFixture = fixtureA;
    }

    if (!platformFixture || !otherFixture)
        return;

    b2AABB platformAABB = platformFixture->GetAABB(0);
    b2AABB otherAABB = otherFixture->GetAABB(0);

    auto platformHeight = platformAABB.upperBound.y - platformAABB.
        lowerBound.y;
    auto halfplatformHeight = platformHeight * 0.5f;

    if (otherAABB.upperBound.y > platformAABB.lowerBound.y +
        halfplatformHeight &&
        otherAABB.lowerBound.y < platformAABB.upperBound.y)
    {
        contact->SetEnabled(false);
    }
}

void ContactListener::handleCoins()
{
```

```
    FixtureUserData* coinUserData = nullptr;
    if (m_World->getEntityRegistry()->has<C_Coin>(userDataA->entity))
    {
        coinUserData = userDataA;
    }
    else if (m_World->getEntityRegistry()->has<C_Coin>(userDataB->entity))
    {
        coinUserData = userDataB;
    }

    if (coinUserData == nullptr) return;

    m_Context->sounds->play(Sounds::ID::CoinPickup);
    m_World->getMarkedEntities().push_back(coinUserData->entity);
}

void ContactListener::initMemberData(b2Contact* contact)
{
    fixtureA = contact->GetFixtureA();
    fixtureB = contact->GetFixtureB();

    userDataA = static_cast<FixtureUserData*>(fixtureA->GetUserData());
    userDataB = static_cast<FixtureUserData*>(fixtureB->GetUserData());
}
```

Context.h

```
#pragma once

#include <unordered_map>

#include <SFML/Graphics/RenderWindow.hpp>

#include "Utility.hpp"

class MusicPlayer;
class SoundEffectPlayer;

struct Context
{
    Context() = default;
    Context(sf::RenderWindow& win, FontHolder& font, TextureHolder& texture
        , MusicPlayer& music, SoundEffectPlayer& sounds,
            bool isPaused = false) :
                window(&win),
                fonts(&font),
                textures(&texture),
                music(&music),
                sounds(&sounds),
                isPaused(isPaused),
                enttToBody()
```

```
    {

    }

    sf::RenderWindow*   window;
    FontHolder*         fonts;
    TextureHolder*      textures;
    MusicPlayer*        music;
    SoundEffectPlayer*  sounds;
    bool                isPaused;
    std::unordered_map<entt::entity, b2Body*> enttToBody;
};
```

FixtureUserData.h

```
#pragma once

#include <entt/entt.hpp>


namespace sf
{
    class RectangleShape;
}


struct FixtureUserData
{
    sf::RectangleShape* shape;
    entt::entity entity;
};
```

Math.h

```
#pragma once

#include <cmath>
#include <SFML/System/Vector2.hpp>

namespace math
{
    constexpr const double PI = 3.14159265358979323846264338327;
    const sf::Vector2f VECTOR_UP     = { 0.f, -1.f};
    const sf::Vector2f VECTOR_DOWN   = { 0.f,  1.f};
    const sf::Vector2f VECTOR_RIGHT  = { 1.f,  0.f};
    const sf::Vector2f VECTOR_LEFT   = {-1.f,  0.f};
    const sf::Vector2f VECTOR_ZERO   = { 0.f,  0.f};

    template <typename T>
    T dotProduct(const sf::Vector2<T>& v, const sf::Vector2<T>& u)
```

```
{
    return v.x * u.x + v.y * u.y;
}

/*template <typename T>
T crossProduct(const sf::Vector2<T>& v, const sf::Vector2<T>& u)
{
    return v.x * u.y - v.y * u.x;
}*/

template <typename T>
T magnitude(const sf::Vector2<T>& v)
{
    return std::sqrt(dotProduct(v, v));
}

template <typename T>
T squaredLength(const sf::Vector2<T>& v)
{
    return dotProduct(v, v);
}

template <typename T>
T distance(const sf::Vector2<T>& v, const sf::Vector2<T>& u)
{
    return magnitude(v - u);
}

template <typename T>
sf::Vector2<T> normalize(const sf::Vector2<T>& v)
{
    assert(v != sf::Vector2<T>());
    return v / magnitude(v);
}

template <typename T>
T angle(const sf::Vector2<T>& v, const sf::Vector2<T>& u)
{
    T m = std::sqrt(squaredLength(v) * squaredLength(u));
    return std::acos(dotProduct(v, u) / m);
}

template <typename T>
T radToDeg(T radians)
{
    return (180 / PI) * radians;
}

template <typename T>
T degToRad(T degrees)
{
    return (PI / 180) * degrees;
```

```
    }

    template <typename T>
    T lerp(const T& v0, const T& v1, float t)
    {
        return (1 - t) * v0 + t * v1;
    }
}
```

QueryCallback.h

```
#pragma once
#include <Box2D/Dynamics/b2WorldCallbacks.h>
#include <Box2D/Dynamics/b2Fixture.h>

//Class copied from http://code.google.com/p/box2d/source/browse/trunk/
    Box2D/Testbed/Framework/Test.cpp
//Copyright (c) 2011 Erin Catto http://box2d.org
class QueryCallback : public b2QueryCallback
{
public:
    explicit QueryCallback(const b2Vec2& point)
    {
        m_point = point;
        m_fixture = nullptr;
    }

    bool ReportFixture(b2Fixture* fixture) override
    {
        b2Body* body = fixture->GetBody();
        if (body->GetType() == b2_dynamicBody)
        {
            bool inside = fixture->TestPoint(m_point);
            if (inside)
            {
                m_fixture = fixture;

                // We are done, terminate the query.
                return false;
            }
        }

        // Continue the query.
        return true;
    }

    b2Vec2 m_point;
    b2Fixture* m_fixture;
};
```

RaycastCallback.h

```
#pragma once

#include <Box2D/Dynamics/b2WorldCallbacks.h>
#include <Box2D/Common/b2Math.h>

class b2Fixture;
class World;
struct Context;

class RayCastCallback : public b2RayCastCallback
{
public:
    RayCastCallback() = default;
    RayCastCallback(Context* context, World* world);
    ~RayCastCallback() override = default;

    float32 ReportFixture(b2Fixture* fixture, const b2Vec2& point,
                          const b2Vec2& normal, float32 fraction)
                              override;

    b2Fixture* m_fixture;
    b2Vec2 m_point;
    b2Vec2 m_normal;
    float32 m_fraction;

    Context* m_Context;
    World* m_World;
};
```

RaycastCallback.cpp

```
#include "RayCastCallback.hpp"
#include "Context.hpp"
#include "FixtureUserData.hpp"
#include "../core/World.h"
#include "../Components/C_Tag.h"

#include <Box2D/Dynamics/b2Fixture.h>

RayCastCallback::RayCastCallback(Context* context, World* world) :
    m_Context(context), m_World(world),
                          m_fixture(nullptr), m_fraction(), m_normal
                              (), m_point()
{
}

float RayCastCallback::ReportFixture(b2Fixture *fixture, const b2Vec2 &
    point, const b2Vec2 &normal, float32 fraction)
{
```

```
    b2Filter filterData = fixture->GetFilterData();

    if (filterData.categoryBits == BodyCategory::Coin || filterData.
        categoryBits == BodyCategory::Ladder)
    {
        return -1.f;
    }

    m_fixture = fixture;
    m_point = point;
    m_normal = normal;
    m_fraction = fraction;

    return fraction;
}
```

ResourceHolder.h

```
#pragma once
#include <string>
#include <memory>
#include <stdexcept>
#include <map>
#include <cassert>

template<typename Resource, typename Identifier>
class ResourceHolder
{
public:
    void load(Identifier, const std::string&);
    template<typename Parameter>
    void load(Identifier, const std::string&, Parameter);

    void unload(Identifier);
    void unloadAll();

    Resource& get(Identifier);
    const Resource& get(Identifier) const;

private:
    std::map<Identifier, std::unique_ptr<Resource>> m_ResourceMap;
};


template<typename Resource, typename Identifier>
inline void ResourceHolder<Resource, Identifier>::load(Identifier ID, const
     std::string& filename)
{
    std::unique_ptr<Resource> resource(new Resource);

    if (!resource->loadFromFile(filename))
```

```
    {
        throw std::runtime_error("ResourceHolder::load - Failed to load " +
            filename);
    }

    auto inserted = m_ResourceMap.insert(std::make_pair(ID, std::move(
        resource)));

    assert(inserted.second);
}

template<typename Resource, typename Identifier>
template<typename Parameter>
inline void ResourceHolder<Resource, Identifier>::load(Identifier ID, const
     std::string& filename, Parameter secondParam)
{
    std::unique_ptr<Resource> resource(new Resource);

    if (!resource->loadFromFile(filename, secondParam))
    {
        throw std::runtime_error("ResourceHolder::load - Failed to load " +
            filename);
    }

    auto inserted = m_ResourceMap.insert(std::make_pair(ID, std::move(
        resource)));

    assert(inserted.second);
}

template<typename Resource, typename Identifier>
inline void ResourceHolder<Resource, Identifier>::unload(Identifier ID)
{
    auto found = m_ResourceMap.find(ID);
    found->second.reset();
    m_ResourceMap.erase(found);
}


template<typename Resource, typename Identifier>
inline void ResourceHolder<Resource, Identifier>::unloadAll()
{
    m_ResourceMap.clear();
}

template<typename Resource, typename Identifier>
inline Resource& ResourceHolder<Resource, Identifier>::get(Identifier ID)
{
    auto found = m_ResourceMap.find(ID);
    assert(found != m_ResourceMap.end());

    return *found->second;
```

```
}

template<typename Resource, typename Identifier>
inline const Resource& ResourceHolder<Resource, Identifier>::get(Identifier
    ID) const
{
    const auto found = m_ResourceMap.find(ID);
    assert(found != m_ResourceMap.cend());

    return *found->second;
}
```

SFMLDebugDraw.h

```
// Debug draw for Box2D 2.2.1 - 2.3.0 for SFML 2.0 - SFMLDebugDraw.h
// Copyright (C) 2013  Matija Lovrekovic
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program.  If not, see <http://www.gnu.org/licenses/>.

#ifndef SFMLDEBUGDRAW_H
#define SFMLDEBUGDRAW_H

#include <Box2D/Box2D.h>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>

#include <cmath>

#include "../Utils/Utility.hpp"

class SFMLDebugDraw : public b2Draw
{
private:
    sf::RenderWindow* m_window{};
public:
    SFMLDebugDraw() = default;
    explicit SFMLDebugDraw(sf::RenderWindow *window);

    void SetWindow(sf::RenderWindow* window) { m_window = window; }
```

```
    /// Convert Box2D's OpenGL style color definition[0-1] to SFML's color
        definition[0-255], with optional alpha byte[Default - opaque]
    static sf::Color GLColorToSFML(const b2Color &color, sf::Uint8 alpha =
        255)
    {
        return sf::Color(static_cast<sf::Uint8>(color.r * 255), static_cast
            <sf::Uint8>(color.g * 255), static_cast<sf::Uint8>(color.b *
            255), alpha);
    }

    /// Draw a closed polygon provided in CCW order.
    void DrawPolygon(const b2Vec2* vertices, int32 vertexCount, const
        b2Color& color) override;

    /// Draw a solid closed polygon provided in CCW order.
    void DrawSolidPolygon(const b2Vec2* vertices, int32 vertexCount, const
        b2Color& color) override;

    /// Draw a circle.
    void DrawCircle(const b2Vec2& center, float radius, const b2Color&
        color) override;

    /// Draw a solid circle.
    void DrawSolidCircle(const b2Vec2& center, float radius, const b2Vec2&
        axis, const b2Color& color) override;

    /// Draw a point
    void DrawPoint(const b2Vec2& p, float size, const b2Color& color)
        override;

    /// Draw a line segment.
    void DrawSegment(const b2Vec2& p1, const b2Vec2& p2, const b2Color&
        color) override;

    /// Draw a transform. Choose your own length scale.
    void DrawTransform(const b2Transform& xf) override;
};
#endif //SFMLDEBUGDRAW_H
```

SFMLDebugDraw.cpp

```
// Debug draw for Box2D 2.2.1 - 2.3.0 for SFML 2.0 - SFMLDebugDraw.cpp
// Copyright (C) 2013  Matija Lovrekovic
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program.  If not, see <http://www.gnu.org/licenses/>.

#include "SFMLDebugDraw.h"

SFMLDebugDraw::SFMLDebugDraw(sf::RenderWindow *window) : m_window(window)
    {}

void SFMLDebugDraw::DrawPolygon(const b2Vec2* vertices, int32 vertexCount,
    const b2Color& color)
{
    sf::ConvexShape polygon(vertexCount);
    sf::Vector2f center;
    for(int i = 0; i < vertexCount; i++)
    {
        //polygon.setPoint(i, utils::B2VecToSFVec<sf::Vector2f>(vertices[i
            ]));
        sf::Vector2f transformedVec = utils::B2VecToSFVec<sf::Vector2f>(
            vertices[i]);
        polygon.setPoint(i, sf::Vector2f(std::floor(transformedVec.x), std
            ::floor(transformedVec.y))); // flooring the coords to fix
            distorted lines on flat surfaces
    }
                                        // they still show up though.. but
        less frequently
    polygon.setOutlineThickness(-1.f);
    polygon.setFillColor(sf::Color::Transparent);
    polygon.setOutlineColor(SFMLDebugDraw::GLColorToSFML(color));

    m_window->draw(polygon);
}

void SFMLDebugDraw::DrawSolidPolygon(const b2Vec2* vertices, int32
    vertexCount, const b2Color& color)
{
    sf::ConvexShape polygon(vertexCount);
    for(int i = 0; i < vertexCount; i++)
    {
        //polygon.setPoint(i, utils::B2VecToSFVec<sf::Vector2f>(vertices[i
            ]));
        sf::Vector2f transformedVec = utils::B2VecToSFVec<sf::Vector2f>(
            vertices[i]);
        polygon.setPoint(i, sf::Vector2f(std::floor(transformedVec.x), std
            ::floor(transformedVec.y))); // flooring the coords to fix
            distorted lines on flat surfaces
    }
                                        // they still show up though.. but
        less frequently
    polygon.setOutlineThickness(-1.f);
```

```
    polygon.setFillColor(SFMLDebugDraw::GLColorToSFML(color, 60));
    polygon.setOutlineColor(SFMLDebugDraw::GLColorToSFML(color));

    m_window->draw(polygon);
}

void SFMLDebugDraw::DrawCircle(const b2Vec2& center, float radius, const
    b2Color& color)
{
    sf::CircleShape circle(radius * utils::PIXELS_PER_METERS);
    circle.setOrigin(radius * utils::PIXELS_PER_METERS, radius * utils::
        PIXELS_PER_METERS);
    circle.setPosition(utils::B2VecToSFVec<sf::Vector2f>(center));
    circle.setFillColor(sf::Color::Transparent);
    circle.setOutlineThickness(-1.f);
    circle.setOutlineColor(SFMLDebugDraw::GLColorToSFML(color));

    m_window->draw(circle);
}

void SFMLDebugDraw::DrawSolidCircle(const b2Vec2& center, float radius,
    const b2Vec2& axis, const b2Color& color)
{
    sf::CircleShape circle(radius * utils::PIXELS_PER_METERS);
    circle.setOrigin(radius * utils::PIXELS_PER_METERS, radius * utils::
        PIXELS_PER_METERS);
    circle.setPosition(utils::B2VecToSFVec<sf::Vector2f>(center));
    circle.setFillColor(SFMLDebugDraw::GLColorToSFML(color, 60));
    circle.setOutlineThickness(1.f);
    circle.setOutlineColor(SFMLDebugDraw::GLColorToSFML(color));

    b2Vec2 endPoint = center + radius * axis;
    sf::Vertex line[2] =
    {
        sf::Vertex(utils::B2VecToSFVec<sf::Vector2f>(center), SFMLDebugDraw
            ::GLColorToSFML(color)),
        sf::Vertex(utils::B2VecToSFVec<sf::Vector2f>(endPoint),
            SFMLDebugDraw::GLColorToSFML(color)),
    };

    m_window->draw(circle);
    m_window->draw(line, 2, sf::Lines);
}

void SFMLDebugDraw::DrawPoint(const b2Vec2& p, float size, const b2Color&
    color)
{
    sf::CircleShape circle;
    circle.setRadius(size);
    circle.setOrigin(circle.getRadius(), circle.getRadius());
    circle.setFillColor(SFMLDebugDraw::GLColorToSFML(color));
    circle.setPosition(utils::B2VecToSFVec<sf::Vector2f>(p));
```

```
    m_window->draw(circle);
}

void SFMLDebugDraw::DrawSegment(const b2Vec2& p1, const b2Vec2& p2, const
    b2Color& color)
{
    sf::Vertex line[] =
    {
        sf::Vertex(utils::B2VecToSFVec<sf::Vector2f>(p1), SFMLDebugDraw::
            GLColorToSFML(color)),
        sf::Vertex(utils::B2VecToSFVec<sf::Vector2f>(p2), SFMLDebugDraw::
            GLColorToSFML(color))
    };

    m_window->draw(line, 2, sf::Lines);
}

void SFMLDebugDraw::DrawTransform(const b2Transform& xf)
{
    float lineLength = 0.4f;

    /*b2Vec2 xAxis(b2Vec2(xf.p.x + (lineLength * xf.q.c), xf.p.y + (
        lineLength * xf.q.s)));*/
    b2Vec2 xAxis = xf.p + lineLength * xf.q.GetXAxis();
    sf::Vertex redLine[] =
    {
        sf::Vertex(utils::B2VecToSFVec<sf::Vector2f>(xf.p), sf::Color::Red)
            ,
        sf::Vertex(utils::B2VecToSFVec<sf::Vector2f>(xAxis), sf::Color::Red
            )
    };

    // You might notice that the ordinate(Y axis) points downward unlike
        the one in Box2D testbed
    // That's because the ordinate in SFML coordinate system points
        downward while the OpenGL(testbed) points upward
    /*b2Vec2 yAxis(b2Vec2(xf.p.x + (lineLength * -xf.q.s), xf.p.y + (
        lineLength * xf.q.c)));*/
    b2Vec2 yAxis = xf.p + lineLength * xf.q.GetYAxis();
    sf::Vertex greenLine[] =
    {
        sf::Vertex(utils::B2VecToSFVec<sf::Vector2f>(xf.p), sf::Color::
            Green),
        sf::Vertex(utils::B2VecToSFVec<sf::Vector2f>(yAxis), sf::Color::
            Green)
    };

    m_window->draw(redLine, 2, sf::Lines);
    m_window->draw(greenLine, 2, sf::Lines);
}
```

Timer.h

```
#pragma once

#include <SFML/System/Time.hpp>

class Timer
{
public:
    Timer() = default;
    Timer(const sf::Time& seconds);

    void start();
    void update(const sf::Time &dt);
    void stop();
    float getRemainingTime();
    void setTimer(const sf::Time &amount);
    void addTime(const sf::Time &amount);
    void reset();
    bool isRunning();

private:
    bool isActive{ false };
    sf::Time seconds{};
    sf::Time remainingTime{};
};
```

Timer.cpp

```
#include "Timer.hpp"

Timer::Timer(const sf::Time& seconds) : seconds(seconds), remainingTime(
    seconds)
{

}

void Timer::start()
{
    isActive = true;
}

void Timer::update(const sf::Time& dt)
{
    if (isActive)
    {
        remainingTime -= dt;

        if (remainingTime.asSeconds() <= 0)
            isActive = false;
    }
```

```
}

void Timer::stop()
{
    isActive = false;
}

float Timer::getRemainingTime()
{
    return remainingTime.asSeconds();
}

void Timer::setTimer(const sf::Time& amount)
{
    seconds = amount;
    remainingTime = seconds;
}

void Timer::addTime(const sf::Time& amount)
{
    remainingTime += amount;
}

void Timer::reset()
{
    remainingTime = seconds;
}

bool Timer::isRunning()
{
    return isActive;
}
```

Utility.h

```
#pragma once

#include <SFML/Window/Keyboard.hpp>
#include <SFML/Graphics/Sprite.hpp>
#include <SFML/Graphics/Text.hpp>
#include <SFML/Graphics/RectangleShape.hpp>
#include <SFML/System/Vector2.hpp>
#include <Box2D/Box2D.h>

#include <cassert>
#include <cmath>
#include <entt/entity/registry.hpp>

#include "../Utils/ResourceHolder.h"
```

```
// ======= FORWARD DECLARATIONS =======
namespace sf
{
    class Texture;
    class Font;
    class Sprite;
    class Text;
    class RectangleShape;
    class SoundBuffer;
}

template<typename Resource, typename Identifier>
class ResourceHolder;

class AnimatedSprite;

namespace tmx
{
    class Object;
}
// ====================================


namespace BodyCategory
{
    enum ID
    {
        Player = 0x01,
        Enemy = 0x02,
        OneWayPlatform = 0x04,
        Ladder = 0x08,
        Coin = 0x16,
        Other = 0x32,
    };
}

namespace Music
{
    enum class ID
    {
        MenuTheme,
        GameTheme,
    };
}

namespace Sounds
{
    enum class ID
    {
        Squash,
        CoinPickup,
        Pause,
```

```
    };
}

namespace Textures
{
    enum class ID
    {
        CharactersSpriteSheet,
        MonochromeSpriteSheet,
        MainBackground,
        CoinAnimation,
    };
}

namespace Fonts
{
    enum class ID
    {
        ARJULIAN,
    };
}

namespace Category
{
    enum class ID
    {
        None = 0,
        Scene = 1 << 0,
        Player = 1 << 1,
        Enemy = 1 << 2,
    };
}

namespace GameObjectState
{
    enum class ID
    {
        Standing,
        Walking,
        Jumping,
        ClimbingLadder,
    };
}

namespace Animations
{
    enum class ID
    {
        Standing,
        Walking,
        Jumping,
        ClimbingLadder,
```

```
        EnemyMoving,
        Coin,
    };
}

namespace States
{
    enum class ID
    {
        None,
        Game,
        Menu,
        Settings,
        Loading,
        Pause,
        Title
    };
}

using TextureHolder = ResourceHolder<sf::Texture, Textures::ID>;
using FontHolder = ResourceHolder<sf::Font, Fonts::ID>;
using SoundBufferHolder = ResourceHolder<sf::SoundBuffer, Sounds::ID>;

namespace utils
{
    void centerOrigin(sf::Sprite& sprite);
    void centerOrigin(sf::Text& text);
    void centerOrigin(sf::RectangleShape& rect);
    void centerOrigin(sf::Shape& rect);
    void centerOrigin(AnimatedSprite& animatedSprite);
    tmx::Object getObjectByName(entt::registry &reg, const std::string&
        layerName, const std::string& name);
    std::vector<tmx::Object> getObjectsByName(entt::registry &reg, const
        std::string& layerName, const std::string& substr);
    const char* getKeyName(const sf::Keyboard::Key key);

    // Box2D to SFML space conversion constant
    const auto PIXELS_PER_METERS = 32.f;

    //Converts SFML's vector to Box2D's vector and downscales it so it fits
         Box2D's MKS(meters, kilograms, seconds) units
    b2Vec2 sfVecToB2Vec(const float x, const float y);

    template <typename T>
    b2Vec2 sfVecToB2Vec(sf::Vector2<T> vector)
    {
        return b2Vec2(vector.x / PIXELS_PER_METERS, vector.y /
            PIXELS_PER_METERS);
    }

    // Convert Box2D's vector to SFML vector [Default - scales the vector
        up by PIXELS_PER_METERS constants amount]
```

```
    template <typename T>
    T B2VecToSFVec(const b2Vec2 &vector, bool scaleToPixels = true)
    {
        return T(vector.x * (scaleToPixels ? PIXELS_PER_METERS : 1.f),
            vector.y * (scaleToPixels ? PIXELS_PER_METERS : 1.f));
    }
}
```

Utility.cpp

```
#include "Utility.hpp"
#include "AnimatedSprite.h"
#include <../Components/C_Tilemap.hpp>

#include <tmxlite/Object.hpp>

namespace utils
{
    void centerOrigin(sf::Sprite& sprite)
    {
        const sf::FloatRect bounds = sprite.getLocalBounds();
        sprite.setOrigin(std::floor(bounds.left + bounds.width / 2.f), std
            ::floor(bounds.top + bounds.height / 2.f));
    }

    void centerOrigin(sf::Text& text)
    {
        const sf::FloatRect bounds = text.getLocalBounds();
        text.setOrigin(std::floor(bounds.left + bounds.width / 2.f), std::
            floor(bounds.top + bounds.height / 2.f));
    }

    void centerOrigin(sf::RectangleShape& rect)
    {
        const sf::FloatRect bounds = rect.getLocalBounds();
        rect.setOrigin(std::floor(bounds.left + bounds.width / 2.f), std::
            floor(bounds.top + bounds.height / 2.f));
    }

    void centerOrigin(sf::Shape& shape)
    {
        const sf::FloatRect bounds = shape.getLocalBounds();
        shape.setOrigin(std::floor(bounds.left + bounds.width / 2.f), std::
            floor(bounds.top + bounds.height / 2.f));
    }

    void centerOrigin(AnimatedSprite& animatedSprite)
    {
        auto bounds = animatedSprite.getLocalBounds();
        animatedSprite.setOrigin(std::floor(bounds.left + bounds.width / 2.
            f),
```

```
                                        std::floor(bounds.top + bounds.
                                            height / 2.f));
}

b2Vec2 sfVecToB2Vec(const float x, const float y)
{
    return {x / PIXELS_PER_METERS, y / PIXELS_PER_METERS};
}

tmx::Object getObjectByName(entt::registry &reg, const std::string&
    layerName, const std::string& objName)
{
    tmx::Object retObj;

    reg.view<C_Tilemap>().each([&](auto entity, auto &c_t)
    {
        for (const auto &objGroup : c_t.m_ObjectLayers)
        {
            if (objGroup->getName() == layerName)
            {
                for (const auto &obj : objGroup->getObjects())
                {
                    if (obj.getName() == objName)
                    {
                        retObj = obj;
                    }
                }
            }
        }
    });

    assert(retObj.getUID() != 0);

    return retObj;
}

std::vector<tmx::Object> getObjectsByName(entt::registry& reg, const
    std::string& layerName, const std::string& substr)
{
    std::vector<tmx::Object> objects{};

    reg.view<C_Tilemap>().each([&](auto entity, auto& c_t)
        {
            for (const auto& objGroup : c_t.m_ObjectLayers)
            {
                if (objGroup->getName() == layerName)
                {
                    for (const auto& obj : objGroup->getObjects())
                    {
                        if (obj.getName().find(substr) != std::string::
                            npos)
                        {
```

```
                                objects.push_back(obj);
                            }
                        }
                    }
                }
            });

    return objects;
}

const char* getKeyName(const sf::Keyboard::Key key)
{
    switch (key) {
    default:
    case sf::Keyboard::Unknown:
        return "Unknown";
    case sf::Keyboard::A:
        return "A";
    case sf::Keyboard::B:
        return "B";
    case sf::Keyboard::C:
        return "C";
    case sf::Keyboard::D:
        return "D";
    case sf::Keyboard::E:
        return "E";
    case sf::Keyboard::F:
        return "F";
    case sf::Keyboard::G:
        return "G";
    case sf::Keyboard::H:
        return "H";
    case sf::Keyboard::I:
        return "I";
    case sf::Keyboard::J:
        return "J";
    case sf::Keyboard::K:
        return "K";
    case sf::Keyboard::L:
        return "L";
    case sf::Keyboard::M:
        return "M";
    case sf::Keyboard::N:
        return "N";
    case sf::Keyboard::O:
        return "O";
    case sf::Keyboard::P:
        return "P";
    case sf::Keyboard::Q:
        return "Q";
    case sf::Keyboard::R:
        return "R";
```

```
        case sf::Keyboard::S:
            return "S";
        case sf::Keyboard::T:
            return "T";
        case sf::Keyboard::U:
            return "U";
        case sf::Keyboard::V:
            return "V";
        case sf::Keyboard::W:
            return "W";
        case sf::Keyboard::X:
            return "X";
        case sf::Keyboard::Y:
            return "Y";
        case sf::Keyboard::Z:
            return "Z";
        case sf::Keyboard::Num0:
            return "Num0";
        case sf::Keyboard::Num1:
            return "Num1";
        case sf::Keyboard::Num2:
            return "Num2";
        case sf::Keyboard::Num3:
            return "Num3";
        case sf::Keyboard::Num4:
            return "Num4";
        case sf::Keyboard::Num5:
            return "Num5";
        case sf::Keyboard::Num6:
            return "Num6";
        case sf::Keyboard::Num7:
            return "Num7";
        case sf::Keyboard::Num8:
            return "Num8";
        case sf::Keyboard::Num9:
            return "Num9";
        case sf::Keyboard::Escape:
            return "Escape";
        case sf::Keyboard::LControl:
            return "LControl";
        case sf::Keyboard::LShift:
            return "LShift";
        case sf::Keyboard::LAlt:
            return "LAlt";
        case sf::Keyboard::LSystem:
            return "LSystem";
        case sf::Keyboard::RControl:
            return "RControl";
        case sf::Keyboard::RShift:
            return "RShift";
        case sf::Keyboard::RAlt:
            return "RAlt";
```

```
        case sf::Keyboard::RSystem:
            return "RSystem";
        case sf::Keyboard::Menu:
            return "Menu";
        case sf::Keyboard::LBracket:
            return "LBracket";
        case sf::Keyboard::RBracket:
            return "RBracket";
        case sf::Keyboard::SemiColon:
            return "SemiColon";
        case sf::Keyboard::Comma:
            return "Comma";
        case sf::Keyboard::Period:
            return "Period";
        case sf::Keyboard::Quote:
            return "Quote";
        case sf::Keyboard::Slash:
            return "Slash";
        case sf::Keyboard::BackSlash:
            return "BackSlash";
        case sf::Keyboard::Tilde:
            return "Tilde";
        case sf::Keyboard::Equal:
            return "Equal";
        case sf::Keyboard::Dash:
            return "Dash";
        case sf::Keyboard::Space:
            return "Space";
        case sf::Keyboard::Return:
            return "Return";
        case sf::Keyboard::BackSpace:
            return "BackSpace";
        case sf::Keyboard::Tab:
            return "Tab";
        case sf::Keyboard::PageUp:
            return "PageUp";
        case sf::Keyboard::PageDown:
            return "PageDown";
        case sf::Keyboard::End:
            return "End";
        case sf::Keyboard::Home:
            return "Home";
        case sf::Keyboard::Insert:
            return "Insert";
        case sf::Keyboard::Delete:
            return "Delete";
        case sf::Keyboard::Add:
            return "Add";
        case sf::Keyboard::Subtract:
            return "Subtract";
        case sf::Keyboard::Multiply:
            return "Multiply";
```

```
        case sf::Keyboard::Divide:
            return "Divide";
        case sf::Keyboard::Left:
            return "LeftArrow";
        case sf::Keyboard::Right:
            return "RightArrow";
        case sf::Keyboard::Up:
            return "UpArrow";
        case sf::Keyboard::Down:
            return "DownArrow";
        case sf::Keyboard::Numpad0:
            return "Numpad0";
        case sf::Keyboard::Numpad1:
            return "Numpad1";
        case sf::Keyboard::Numpad2:
            return "Numpad2";
        case sf::Keyboard::Numpad3:
            return "Numpad3";
        case sf::Keyboard::Numpad4:
            return "Numpad4";
        case sf::Keyboard::Numpad5:
            return "Numpad5";
        case sf::Keyboard::Numpad6:
            return "Numpad6";
        case sf::Keyboard::Numpad7:
            return "Numpad7";
        case sf::Keyboard::Numpad8:
            return "Numpad8";
        case sf::Keyboard::Numpad9:
            return "Numpad9";
        case sf::Keyboard::F1:
            return "F1";
        case sf::Keyboard::F2:
            return "F2";
        case sf::Keyboard::F3:
            return "F3";
        case sf::Keyboard::F4:
            return "F4";
        case sf::Keyboard::F5:
            return "F5";
        case sf::Keyboard::F6:
            return "F6";
        case sf::Keyboard::F7:
            return "F7";
        case sf::Keyboard::F8:
            return "F8";
        case sf::Keyboard::F9:
            return "F9";
        case sf::Keyboard::F10:
            return "F10";
        case sf::Keyboard::F11:
            return "F11";
```

```
        case sf::Keyboard::F12:
            return "F12";
        case sf::Keyboard::F13:
            return "F13";
        case sf::Keyboard::F14:
            return "F14";
        case sf::Keyboard::F15:
            return "F15";
        case sf::Keyboard::Pause:
            return "Pause";
        }
    }
}
```

SFMLOrthogonalLayer.h

```
/*********************************************************************
(c) Matt Marchant & contributors 2016 - 2019
http://trederia.blogspot.com

tmxlite - Zlib license.

This software is provided 'as-is', without any express or
implied warranty. In no event will the authors be held
liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose,
including commercial applications, and to alter it and redistribute
it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented;
you must not claim that you wrote the original software.
If you use this software in a product, an acknowledgment
in the product documentation would be appreciated but
is not required.

2. Altered source versions must be plainly marked as such,
and must not be misrepresented as being the original software.

3. This notice may not be removed or altered from any
source distribution.
*********************************************************************/

/*
Creates an SFML drawable from an Orthogonal tmx map layer.
This is an example of drawing with SFML - not all features
are implemented.
*/

#ifndef SFML_ORTHO_HPP_
#define SFML_ORTHO_HPP_
```

```
#include <tmxlite/Map.hpp>
#include <tmxlite/TileLayer.hpp>
#include <tmxlite/detail/Log.hpp>

#include <SFML/Graphics/Drawable.hpp>
#include <SFML/Graphics/RenderStates.hpp>
#include <SFML/Graphics/RenderTarget.hpp>
#include <SFML/Graphics/Vertex.hpp>
#include <SFML/Graphics/Texture.hpp>
#include <SFML/Graphics/Transformable.hpp>
#include <SFML/System/Time.hpp>

#include <memory>
#include <vector>
#include <array>
#include <map>
#include <string>
#include <limits>
#include <iostream>
#include <cmath>


class MapLayer final : public sf::Drawable
{
public:
    MapLayer(const tmx::Map& map, std::size_t idx)
    {
        const auto& layers = map.getLayers();
        if (map.getOrientation() == tmx::Orientation::Orthogonal &&
            idx < layers.size() && layers[idx]->getType() == tmx::Layer::
                Type::Tile)
        {
            //round the chunk size to the nearest tile
            const auto tileSize = map.getTileSize();
            m_chunkSize.x = std::floor(m_chunkSize.x / tileSize.x) *
                tileSize.x;
            m_chunkSize.y = std::floor(m_chunkSize.y / tileSize.y) *
                tileSize.y;
            m_MapTileSize.x = map.getTileSize().x;
            m_MapTileSize.y = map.getTileSize().y;
            const auto& layer = layers[idx]->getLayerAs<tmx::TileLayer>();
            createChunks(map, layer);

            auto mapSize = map.getBounds();
            m_globalBounds.width = mapSize.width;
            m_globalBounds.height = mapSize.height;
        }
        else
        {
            std::cout << "Not a valid orthogonal layer, nothing will be
                drawn." << std::endl;
```

```
    }
}

~MapLayer() = default;
MapLayer(const MapLayer&) = delete;
MapLayer& operator = (const MapLayer&) = delete;

const sf::FloatRect& getGlobalBounds() const { return m_globalBounds; }

void setTile(int tileX, int tileY, tmx::TileLayer::Tile tile, bool
    refresh = true)
{
    sf::Vector2u chunkLocale;
    const auto& selectedChunk = getChunkAndTransform(tileX, tileY,
        chunkLocale);
    selectedChunk->setTile(chunkLocale.x, chunkLocale.y, tile, refresh)
        ;
}

tmx::TileLayer::Tile getTile(int tileX, int tileY)
{
    sf::Vector2u chunkLocale;
    const auto& selectedChunk = getChunkAndTransform(tileX, tileY,
        chunkLocale);
    return selectedChunk->getTile(chunkLocale.x, chunkLocale.y);
}
void setColor(int tileX, int tileY, sf::Color color, bool refresh =
    true)
{
    sf::Vector2u chunkLocale;
    const auto& selectedChunk = getChunkAndTransform(tileX, tileY,
        chunkLocale);
    selectedChunk->setColor(chunkLocale.x, chunkLocale.y, color,
        refresh);
}

sf::Color getColor(int tileX, int tileY)
{
    sf::Vector2u chunkLocale;
    const auto& selectedChunk = getChunkAndTransform(tileX, tileY,
        chunkLocale);
    return selectedChunk->getColor(chunkLocale.x, chunkLocale.y);
}

void update(sf::Time elapsed)
{
    for (auto& c : m_visibleChunks)
    {
        for (AnimationState& as : c->getActiveAnimations())
        {
            as.currentTime += elapsed;
```

```
                tmx::TileLayer::Tile tile;
                tile.ID = as.animTile.animation.frames[0].tileID;
                tile.flipFlags = 0; // TODO: get flipFlags from original
                    tmx::TileLayer::Tile

                std::uint32_t animTime = 0;
                for (const auto& frame : as.animTile.animation.frames)
                {
                    animTime += frame.duration;
                    if (as.currentTime.asMilliseconds() >= animTime)
                    {
                        tile.ID = frame.tileID;
                        if (frame == as.animTile.animation.frames.back())
                        {
                            as.currentTime = sf::milliseconds(0);
                        }
                    }
                }

                setTile(as.tileCords.x, as.tileCords.y, tile);
            }
        }
    }

private:
    //increasing m_chunkSize by 4; fixes render problems when mapsize !=
        chunksize
    //sf::Vector2f m_chunkSize = sf::Vector2f(1024.f, 1024.f);
    sf::Vector2f m_chunkSize = sf::Vector2f(512.f, 512.f);
    sf::Vector2u m_chunkCount;
    sf::Vector2u m_MapTileSize;   // general Tilesize of Map
    sf::FloatRect m_globalBounds;

    using TextureResource = std::map<std::string, std::unique_ptr<sf::
        Texture>>;
    TextureResource m_textureResource;

    struct AnimationState
    {
        sf::Vector2u tileCords;
        sf::Time startTime;
        sf::Time currentTime;
        tmx::Tileset::Tile animTile;
        std::uint8_t flipFlags;
    };

    class Chunk final : public sf::Transformable, public sf::Drawable
    {
    public:
        using Ptr = std::unique_ptr<Chunk>;

        // the Android OpenGL driver isn't capable of rendering quads,
```

```
        // so we need to use two triangles per tile instead
#ifdef __ANDROID__
        using Tile = std::array<sf::Vertex, 6u>;
#endif
#ifndef __ANDROID__
        using Tile = std::array<sf::Vertex, 4u>;
#endif
        Chunk(const tmx::TileLayer& layer, std::vector<const tmx::Tileset*>
             tilesets,
            const sf::Vector2f& position, const sf::Vector2f& tileCount,
                const sf::Vector2u& tileSize,
            std::size_t rowSize,  TextureResource& tr, const std::map<std::
                uint32_t, tmx::Tileset::Tile>& animTiles)
            : m_animTiles(animTiles)
        {
            setPosition(position);
            layerOpacity = static_cast<sf::Uint8>(layer.getOpacity() /  1.f
                 * 255.f);
            sf::Color vertColour = sf::Color(200 ,200, 200, layerOpacity);
            auto offset = layer.getOffset();
            layerOffset.x = offset.x;
            layerOffset.x = offset.y;
            chunkTileCount.x = tileCount.x;
            chunkTileCount.y = tileCount.y;
            mapTileSize = tileSize;
            const auto& tileIDs = layer.getTiles();

            //go through the tiles and create all arrays (for latter
                manipulation)
            for (const auto& ts : tilesets)
            {
                if(ts->getImagePath().empty())
                {
                    tmx::Logger::log("This example does not support
                        Collection of Images tilesets", tmx::Logger::Type::
                        Info);
                    tmx::Logger::log("Chunks using " + ts->getName() + "
                        will not be created", tmx::Logger::Type::Info);
                    continue;
                }
                m_chunkArrays.emplace_back(std::make_unique<ChunkArray>(*tr
                    .find(ts->getImagePath())->second, *ts));
            }
            std::size_t xPos = static_cast<std::size_t>(position.x /
                tileSize.x);
            std::size_t yPos = static_cast<std::size_t>(position.y /
                tileSize.y);
            for (auto y = yPos; y < yPos + tileCount.y; ++y)
            {
                for (auto x = xPos; x < xPos + tileCount.x; ++x)
                {
                    auto idx = (y * rowSize + x);
```

```
                m_chunkTileIDs.emplace_back(tileIDs[idx]);
                m_chunkColors.emplace_back(vertColour);
            }
        }
        generateTiles(true);
    }

    void generateTiles(bool registerAnimation = false)
    {
        if (registerAnimation)
        {
            m_activeAnimations.clear();
        }
        for (const auto& ca : m_chunkArrays)
        {
            sf::Uint32 idx = 0;
            std::size_t xPos = static_cast<std::size_t>(getPosition().x
                / mapTileSize.x);
            std::size_t yPos = static_cast<std::size_t>(getPosition().y
                / mapTileSize.y);
            for (auto y = yPos; y < yPos + chunkTileCount.y; ++y)
            {
                for (auto x = xPos; x < xPos + chunkTileCount.x; ++x)
                {
                    if (idx < m_chunkTileIDs.size() && m_chunkTileIDs[
                        idx].ID >= ca->m_firstGID
                        && m_chunkTileIDs[idx].ID <= ca->m_lastGID)
                    {
                        if (registerAnimation && m_animTiles.find(
                            m_chunkTileIDs[idx].ID) != m_animTiles.end()
                            )
                        {
                            AnimationState as;
                            as.animTile = m_animTiles[m_chunkTileIDs[
                                idx].ID];
                            as.startTime = sf::milliseconds(0);
                            as.tileCords = sf::Vector2u(x,y);
                            m_activeAnimations.push_back(as);
                        }

                        sf::Vector2f tileOffset(x * mapTileSize.x, (
                            float)y * mapTileSize.y + mapTileSize.y - ca
                            ->tileSetSize.y);

                        auto idIndex = m_chunkTileIDs[idx].ID - ca->
                            m_firstGID;
                        sf::Vector2f tileIndex(idIndex % ca->
                            tsTileCount.x, idIndex / ca->tsTileCount.x);
                        tileIndex.x *= ca->tileSetSize.x;
                        tileIndex.y *= ca->tileSetSize.y;
                        Tile tile =
                        {
```

```
#ifndef __ANDROID__
                                sf::Vertex(tileOffset - getPosition(),
                                    m_chunkColors[idx], tileIndex),
                                sf::Vertex(tileOffset - getPosition() + sf
                                    ::Vector2f(ca->tileSetSize.x, 0.f),
                                    m_chunkColors[idx], tileIndex + sf::
                                    Vector2f(ca->tileSetSize.x, 0.f)),
                                sf::Vertex(tileOffset - getPosition() + sf
                                    ::Vector2f(ca->tileSetSize.x, ca->
                                    tileSetSize.y), m_chunkColors[idx],
                                    tileIndex + sf::Vector2f(ca->tileSetSize
                                    .x, ca->tileSetSize.y)),
                                sf::Vertex(tileOffset - getPosition() + sf
                                    ::Vector2f(0.f, ca->tileSetSize.y),
                                    m_chunkColors[idx], tileIndex + sf::
                                    Vector2f(0.f, ca->tileSetSize.y))
#endif
#ifdef __ANDROID__
                                sf::Vertex(tileOffset - getPosition(),
                                    m_chunkColors[idx], tileIndex),
                                sf::Vertex(tileOffset - getPosition() + sf
                                    ::Vector2f(ca->tileSetSize.x, 0.f),
                                    m_chunkColors[idx], tileIndex + sf::
                                    Vector2f(ca->tileSetSize.x, 0.f)),
                                sf::Vertex(tileOffset - getPosition() + sf
                                    ::Vector2f(ca->tileSetSize.x, ca->
                                    tileSetSize.y), m_chunkColors[idx],
                                    tileIndex + sf::Vector2f(ca->tileSetSize
                                    .x, ca->tileSetSize.y)),
                                sf::Vertex(tileOffset - getPosition(),
                                    m_chunkColors[idx], tileIndex),
                                sf::Vertex(tileOffset - getPosition() + sf
                                    ::Vector2f(0.f, ca->tileSetSize.y),
                                    m_chunkColors[idx], tileIndex + sf::
                                    Vector2f(0.f, ca->tileSetSize.y)),
                                sf::Vertex(tileOffset - getPosition() + sf
                                    ::Vector2f(ca->tileSetSize.x, ca->
                                    tileSetSize.y), m_chunkColors[idx],
                                    tileIndex + sf::Vector2f(ca->tileSetSize
                                    .x, ca->tileSetSize.y))
#endif
                            };
                            doFlips(m_chunkTileIDs[idx].flipFlags,&tile[0].
                                texCoords,&tile[1].texCoords,&tile[2].
                                texCoords,&tile[3].texCoords);
                            ca->addTile(tile);
                        }
                        idx++;
                    }
                }
            }
        }
```

```
        ~Chunk() = default;
        Chunk(const Chunk&) = delete;
        Chunk& operator = (const Chunk&) = delete;
        std::vector<AnimationState>& getActiveAnimations() { return
            m_activeAnimations; }
        tmx::TileLayer::Tile getTile(int x, int y) const
        {
            return m_chunkTileIDs[calcIndexFrom(x,y)];
        }
        void setTile(int x, int y, tmx::TileLayer::Tile tile, bool refresh)
        {
            m_chunkTileIDs[calcIndexFrom(x,y)] = tile;
            maybeRegenerate(refresh);
        }
        sf::Color getColor(int x, int y) const
        {
            return m_chunkColors[calcIndexFrom(x,y)];
        }
        void setColor(int x, int y, sf::Color color, bool refresh)
        {
            m_chunkColors[calcIndexFrom(x,y)] = color;
            maybeRegenerate(refresh);
        }
        void maybeRegenerate(bool refresh)
        {
            if (refresh)
            {
                for (const auto& ca : m_chunkArrays)
                {
                    ca->reset();
                }
                generateTiles();
            }
        }
        int calcIndexFrom(int x, int y) const
        {
            return x + y * chunkTileCount.x;
        }
        bool empty() const { return m_chunkArrays.empty(); }
        void flipY(sf::Vector2f *v0, sf::Vector2f *v1, sf::Vector2f *v2, sf
            ::Vector2f *v3)
        {
            //Flip Y
            sf::Vector2f tmp = *v0;
            v0->y = v2->y;
            v1->y = v2->y;
            v2->y = tmp.y ;
            v3->y = v2->y  ;
        }

        void flipX(sf::Vector2f *v0, sf::Vector2f *v1, sf::Vector2f *v2, sf
            ::Vector2f *v3)
```

```
        {
            //Flip X
            sf::Vector2f tmp = *v0;
            v0->x = v1->x;
            v1->x = tmp.x;
            v2->x = v3->x;
            v3->x = v0->x ;
        }

        void flipD(sf::Vector2f *v0, sf::Vector2f *v1, sf::Vector2f *v2, sf
            ::Vector2f *v3)
        {
            //Diagonal flip
            sf::Vector2f tmp = *v1;
            v1->x = v3->x;
            v1->y = v3->y;
            v3->x = tmp.x;
            v3->y = tmp.y;
        }

        void doFlips(std::uint8_t bits, sf::Vector2f *v0, sf::Vector2f *v1,
             sf::Vector2f *v2, sf::Vector2f *v3)
        {
            //0000 = no change
            //0100 = vertical = swap y axis
            //1000 = horizontal = swap x axis
            //1100 = horiz + vert = swap both axes = horiz+vert = rotate
                180 degrees
            //0010 = diag = rotate 90 degrees right and swap x axis
            //0110 = diag+vert = rotate 270 degrees right
            //1010 = horiz+diag = rotate 90 degrees right
            //1110 = horiz+vert+diag = rotate 90 degrees right and swap y
                axis
            if(!(bits & tmx::TileLayer::FlipFlag::Horizontal) &&
               !(bits & tmx::TileLayer::FlipFlag::Vertical) &&
               !(bits & tmx::TileLayer::FlipFlag::Diagonal) )
            {
                //Shortcircuit tests for nothing to do
                return;
            }
            else if(!(bits & tmx::TileLayer::FlipFlag::Horizontal) &&
                (bits & tmx::TileLayer::FlipFlag::Vertical) &&
               !(bits & tmx::TileLayer::FlipFlag::Diagonal) )
            {
                //0100
                flipY(v0,v1,v2,v3);
            }
            else if((bits & tmx::TileLayer::FlipFlag::Horizontal) &&
               !(bits & tmx::TileLayer::FlipFlag::Vertical) &&
               !(bits & tmx::TileLayer::FlipFlag::Diagonal) )
            {
                //1000
```

```
                flipX(v0,v1,v2,v3);
            }
            else if((bits & tmx::TileLayer::FlipFlag::Horizontal) &&
               (bits & tmx::TileLayer::FlipFlag::Vertical) &&
               !(bits & tmx::TileLayer::FlipFlag::Diagonal) )
            {
                //1100
                flipY(v0,v1,v2,v3);
                flipX(v0,v1,v2,v3);
            }
            else if(!(bits & tmx::TileLayer::FlipFlag::Horizontal) &&
               !(bits & tmx::TileLayer::FlipFlag::Vertical) &&
               (bits & tmx::TileLayer::FlipFlag::Diagonal) )
            {
                //0010
                flipD(v0,v1,v2,v3);
            }
            else if(!(bits & tmx::TileLayer::FlipFlag::Horizontal) &&
                (bits & tmx::TileLayer::FlipFlag::Vertical) &&
                (bits & tmx::TileLayer::FlipFlag::Diagonal) )
            {
                //0110
                flipX(v0,v1,v2,v3);
                flipD(v0,v1,v2,v3);
            }
            else if((bits & tmx::TileLayer::FlipFlag::Horizontal) &&
                !(bits & tmx::TileLayer::FlipFlag::Vertical) &&
                (bits & tmx::TileLayer::FlipFlag::Diagonal) )
            {
                //1010
                flipY(v0,v1,v2,v3);
                flipD(v0,v1,v2,v3);
           }
            else if((bits & tmx::TileLayer::FlipFlag::Horizontal) &&
               (bits & tmx::TileLayer::FlipFlag::Vertical) &&
               (bits & tmx::TileLayer::FlipFlag::Diagonal) )
            {
                //1110
                flipY(v0,v1,v2,v3);
                flipX(v0,v1,v2,v3);
                flipD(v0,v1,v2,v3);
            }
        }

    private:
        class ChunkArray final : public sf::Drawable
        {
        public:
            using Ptr = std::unique_ptr<ChunkArray>;
            tmx::Vector2u tileSetSize;
            sf::Vector2u tsTileCount;
            std::uint32_t m_firstGID, m_lastGID;
```

```
            explicit ChunkArray(const sf::Texture& t, const tmx::Tileset&
                ts)
                : m_texture(t)
            {
                auto texSize = getTextureSize();
                tileSetSize = ts.getTileSize();
                tsTileCount.x = texSize.x / tileSetSize.x;
                tsTileCount.y = texSize.y / tileSetSize.y;
                m_firstGID = ts.getFirstGID();
                m_lastGID = ts.getLastGID();
            }

            ~ChunkArray() = default;
            ChunkArray(const ChunkArray&) = delete;
            ChunkArray& operator = (const ChunkArray&) = delete;

            void reset()
            {
                m_vertices.clear();
            }
            void addTile(const Chunk::Tile& tile)
            {
                for (const auto& v : tile)
                {
                    m_vertices.push_back(v);
                }
            }
            sf::Vector2u getTextureSize() const { return m_texture.getSize
                (); }

        private:
            const sf::Texture& m_texture;
            std::vector<sf::Vertex> m_vertices;
            void draw(sf::RenderTarget& rt, sf::RenderStates states) const
                override
            {
                states.texture = &m_texture;
#ifndef __ANDROID__
                rt.draw(m_vertices.data(), m_vertices.size(), sf::Quads,
                    states);
#endif
#ifdef __ANDROID__
                rt.draw(m_vertices.data(), m_vertices.size(), sf::Triangles
                    , states);
#endif
            }
        };

        sf::Uint8 layerOpacity;     // opacity of the layer
        sf::Vector2f layerOffset;   // Layer offset
        sf::Vector2u mapTileSize;   // general Tilesize of Map
        sf::Vector2f chunkTileCount;   // chunk tilecount
```

```
        std::vector<tmx::TileLayer::Tile> m_chunkTileIDs; // stores all
            tiles in this chunk for later manipulation
        std::vector<sf::Color> m_chunkColors; // stores colors for extended
             color effects
        std::map<std::uint32_t, tmx::Tileset::Tile> m_animTiles;    //
            animation catalogue
        std::vector<AnimationState> m_activeAnimations;     // Animations
            to be done in this chunk
        std::vector<ChunkArray::Ptr> m_chunkArrays;
        void draw(sf::RenderTarget& rt, sf::RenderStates states) const
            override
        {
            states.transform *= getTransform();
            for (const auto& a : m_chunkArrays)
            {
                rt.draw(*a, states);
            }
        }
    };

    std::vector<Chunk::Ptr> m_chunks;
    mutable std::vector<Chunk*> m_visibleChunks;
    Chunk::Ptr& getChunkAndTransform(int x, int y, sf::Vector2u&
        chunkRelative)
    {
        uint32_t chunkX = floor((x * m_MapTileSize.x) / m_chunkSize.x);
        uint32_t chunkY = floor((y * m_MapTileSize.y) / m_chunkSize.y);
        chunkRelative.x = ((x * m_MapTileSize.x) - chunkX * m_chunkSize.x )
             / m_MapTileSize.x ;
        chunkRelative.y = ((y * m_MapTileSize.y) - chunkY * m_chunkSize.y )
             / m_MapTileSize.y ;
        return  m_chunks[chunkX + chunkY * m_chunkCount.x];
    }
    void createChunks(const tmx::Map& map, const tmx::TileLayer& layer)
    {
        //look up all the tile sets and load the textures
        const auto& tileSets = map.getTilesets();
        const auto& layerIDs = layer.getTiles();
        std::uint32_t maxID = std::numeric_limits<std::uint32_t>::max();
        std::vector<const tmx::Tileset*> usedTileSets;

        for (auto i = tileSets.rbegin(); i != tileSets.rend(); ++i)
        {
            for (const auto& tile : layerIDs)
            {
                if (tile.ID >= i->getFirstGID() && tile.ID < maxID)
                {
                    usedTileSets.push_back(&(*i));
                    break;
                }
            }
            maxID = i->getFirstGID();
```

```
        }

        sf::Image fallback;
        fallback.create(2, 2, sf::Color::Magenta);
        for (const auto& ts : usedTileSets)
        {
            const auto& path = ts->getImagePath();
            //std::unique_ptr<sf::Texture> newTexture = std::make_unique<sf
                ::Texture>();
            std::unique_ptr<sf::Texture> newTexture = std::make_unique<sf::
                Texture>();
            sf::Image img;
            if (!img.loadFromFile(path))
            {
                newTexture->loadFromImage(fallback);
            }
            else
            {
                if (ts->hasTransparency())
                {
                    auto transparency = ts->getTransparencyColour();
                    img.createMaskFromColor({ transparency.r, transparency.
                        g, transparency.b, transparency.a });
                }
                newTexture->loadFromImage(img);
            }
            m_textureResource.insert(std::make_pair(path, std::move(
                newTexture)));
        }

        //calculate the number of chunks in the layer
        //and create each one
        const auto bounds = map.getBounds();
        m_chunkCount.x = static_cast<sf::Uint32>(std::ceil(bounds.width /
            m_chunkSize.x));
        m_chunkCount.y = static_cast<sf::Uint32>(std::ceil(bounds.height /
            m_chunkSize.y));

        sf::Vector2u tileSize(map.getTileSize().x, map.getTileSize().y);

        for (auto y = 0u; y < m_chunkCount.y; ++y)
        {
            sf::Vector2f tileCount(m_chunkSize.x / tileSize.x, m_chunkSize.
                y / tileSize.y);
            for (auto x = 0u; x < m_chunkCount.x; ++x)
            {
                // calculate size of each Chunk (clip against map)
                if ((x + 1) * m_chunkSize.x > bounds.width)
                {
                    tileCount.x = (bounds.width - x * m_chunkSize.x) /  map
                        .getTileSize().x;
                }
```

```
            if ((y + 1) * m_chunkSize.y > bounds.height)
            {
                tileCount.y = (bounds.height - y * m_chunkSize.y) /
                    map.getTileSize().y;
            }
            //m_chunks.emplace_back(std::make_unique<Chunk>(layer,
                usedTileSets,
            //    sf::Vector2f(x * m_chunkSize.x, y * m_chunkSize.y),
                tileCount, map.getTileCount().x, m_textureResource));
            m_chunks.emplace_back(std::make_unique<Chunk>(layer,
                usedTileSets,
                sf::Vector2f(x * m_chunkSize.x, y * m_chunkSize.y),
                    tileCount, tileSize, map.getTileCount().x,
                    m_textureResource, map.getAnimatedTiles()));
        }
    }
}

void updateVisibility(const sf::View& view) const
{
    sf::Vector2f viewCorner = view.getCenter();
    viewCorner -= view.getSize() / 2.f;

    int posX = static_cast<int>(std::floor(viewCorner.x / m_chunkSize.x
        ));
    int posY = static_cast<int>(std::floor(viewCorner.y / m_chunkSize.y
        ));
    int posX2 = static_cast<int>(std::ceil((viewCorner.x + view.getSize
        ().x) / m_chunkSize.x));
    int posY2 = static_cast<int>(std::ceil((viewCorner.y + view.getSize
        ().x)/ m_chunkSize.y));

    std::vector<Chunk*> visible;
    for (auto y = posY; y < posY2; ++y)
    {
        for (auto x = posX; x < posX2; ++x)
        {
            std::size_t idx = y * int(m_chunkCount.x) + x;
            if (idx >= 0u && idx < m_chunks.size() && !m_chunks[idx]->
                empty())
            {
                visible.push_back(m_chunks[idx].get());
            }
        }
    }

    std::swap(m_visibleChunks, visible);
}

void draw(sf::RenderTarget& rt, sf::RenderStates states) const override
{
    //calc view coverage and draw nearest chunks
```

```
        updateVisibility(rt.getView());
        for (const auto& c : m_visibleChunks)
        {
            rt.draw(*c, states);
        }
    }
};

#endif //SFML_ORTHO_HPP_
```

Βιβλιογραφία

[1] Artur Moreira, Henrik Vogelius Hansson, and Jan Haller. ***SFML Game Development***. 1st ed. Livery Place, 35 Livery Street, Birmingham B3 2PB, UK: Packt, 2013.

[2] Maxime Barbier. ***SFML Blueprints***. 1st ed. Livery Place, 35 Livery Street, Birmingham B3 2PB, UK: Packt, 2015.

[3] Raimondas Pupius. ***SFML Game Development By Example***. 1st ed. Livery Place, 35 Livery Street, Birmingham B3 2PB: Packt, 2015.

[4] Raimondas Pupius. ***Mastering SFML Game Development***. 1st ed. Livery Place, 35 Livery Street, Birmingham B3 2PB: Packt, 2017.

[5] Milcho G. Milchev. ***SFML Essentials***. 1st ed. Livery Place, 35 Livery Street, Birmingham B3 2PB: Packt, 2015.

[6] Steve Rabin. ***Game AI Pro 3***. 1st ed. 6000 Broken Sound Parkway NW, Suite 300, Boca Raton FL 33487-2742: CRC Press, 2017.

[7] Fletcher Dunn and Ian Parberry. ***3D Math Primer for Graphics andGame Development***. 2nd ed. 6000 Broken Sound Parkway NW, Suite 300, Boca Raton FL 33487-2742: CRC Press, 2011.

[8] Robert Nystrom. ***Game Programming Patterns***. 1st ed. 1 Principal Place, Worship Street, London, EC2A 2FA United Kingdom: Genever Benning, 2014.

[9] William Stallings. ***Computer organization and architecture : designing for performance***. 10th ed. 221 River Street, Hoboken, NJ 07030: Pearson Education, Inc., 2015.

[10] *Cristiano Ferreira and Mike Geig.* ***Get Started with the Unity* Entity Component System (ECS), C# Job System, and Burst Compiler***. `https://intel.ly/3cC0J7a`. Ημερομηνία πρόσβασης: 31-3-2021.

[11] *Cybercomputing.* ***CPU Cache***. `https://www.cybercomputing.co.uk/Languages/Hardware/cacheCPU.html`. Ημερομηνία πρόσβασης: 1-4-2021.

[12] *Drew Campbell.* ***Understanding Delta Time***. `https://bit.ly/31FsOnW`. Ημερομηνία πρόσβασης: 1-4-2021.

[13] *Erin Catto.* ***Box2D C++ tutorials - Introduction***. `https://www.iforce2d.net/b2dtut/`. Ημερομηνία πρόσβασης: 9-4-2021.

[14] *Glenn Fiedler.* ***Fix Your Timestep!*** `https://gafferongames.com/post/fix_your_timestep/`. Ημερομηνία πρόσβασης: 1-4-2021.

[15] *Intel.* ***How to Manipulate Data Structure to Optimize Memory Use on 32-Bit Intel® Architecture***. https://software.intel.com/content/www/us/en/develop/articles/how-to-manipulate-data-structure-to-optimize-memory-use-on-32-bit-intel-architecture.html. Ημερομηνία πρόσβασης: 1-4-2021.

[16] *Laurent Gomila.* ***Controlling the 2D camera with views***. https://bit.ly/2OAbWvI. Ημερομηνία πρόσβασης: 10-4-2021.

[17] *Mike Acton.* ***CppCon 2014: Mike Acton "Data-Oriented Design and C++"***. https://www.youtube.com/watch?v=rX0ItVEVjHc. Ημερομηνία πρόσβασης: 16-4-2021.

[18] *Noel Llopis.* ***Data-Oriented Design (Or Why You Might Be Shooting Yourself in The Foot With OOP)***. https://gamesfromwithin.com/data-oriented-design. Ημερομηνία πρόσβασης: 16-4-2021.

[19] *Richard Fabian.* ***Data-Oriented Design***. https://www.dataorienteddesign.com/dodbook/. Ημερομηνία πρόσβασης: 31-3-2021.

[20] *Scott Meyers.* ***code::dive conference 2014 - Scott Meyers: Cpu Caches and Why You Care"***. https://www.youtube.com/watch?v=WDIkqP4JbkE. Ημερομηνία πρόσβασης: 16-4-2021.

[21] *Wilmer Lin.* ***Entity Component System for Unity: Getting Started***. https://www.raywenderlich.com/7630142-entity-component-system-for-unity-getting-started. Ημερομηνία πρόσβασης: 1-4-2021.