# ΕΡΓΑΣΙΑ ΤΕΧΝΟΛΟΓΙΑΣ ΠΟΛΥΜΕΣΩΝ 2021-2022

2022

**ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ**

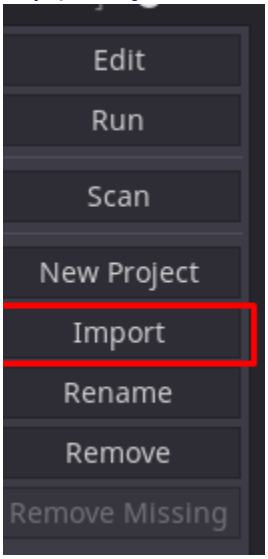
**ΚΩΤΣΗΣ-ΠΑΝΑΚΑΚΗΣ ΒΑΣΙΛΕΙΟΣ-ΕΚΤΩΡ 3180094**

**ΓΟΝΑΤΑ ΑΝΝΑ 3180034**

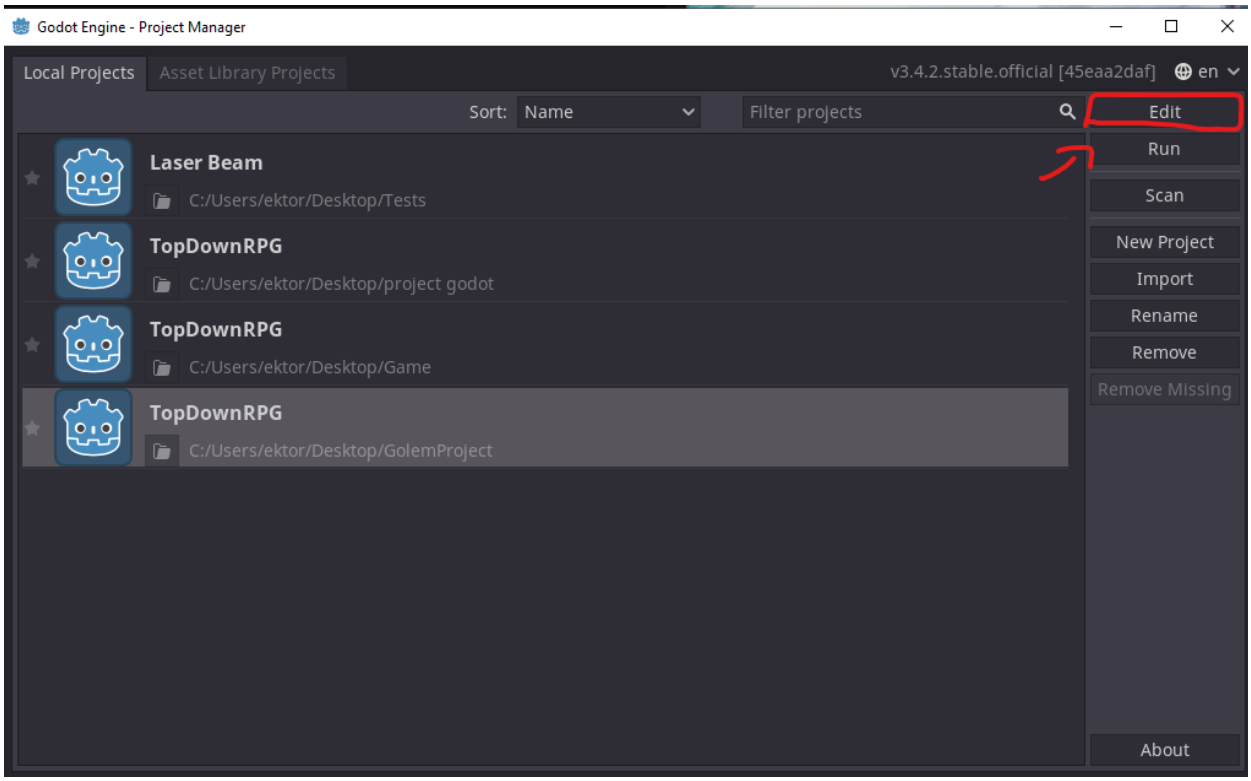
# 1. ΟΔΗΓΙΕΣ ΕΓΚΑΤΑΣΤΑΣΗΣ

## 1. DEBUG έκδοση (Για ανάγνωση ή τροποποίηση του project)

1. Κατεβάστε την πλατφόρμα ψηφιακής διανομής "**Steam**" (**https://store.steampowered.com/**)
2. Ψάξτε στην μπάρα αναζήτησης του **"Καταστήματος"** την μηχανή ανάπτυξης παιχνιδιών "Godot Engine" και κατεβάστε την.
3. Αφότου κατέβει η εφαρμογή, ανοίξτε την εφαρμογή και διαλέξτε το Project σας με την επιλογή "**Import**".




4. Στη συνέχεια πατήστε το κουμπί **"Edit"** για να δείτε τον κώδικα και τις σκηνές του παιχνιδιού.




5. Τώρα μπορείτε να πλοηγείτε το πρότζεκτ. Θα επισημάνουμε κάποια αξιοσημείωτα σημεία στην διεπαφή χρήστη για την πιο εύκολη της χρήση και κατανόηση σε επόμενο κεφάλαιο της αναφοράς

## 2. RELEASE έκδοση (Για εκτέλεση του παιχνιδιού)

Το μόνο αρχείο που χρειάζεστε για να τρέξετε το παιχνίδι είναι το **“Titan Slayer.exe”**

# 1. Η ΕΡΓΑΣΙΑ

## 1. Ο τύπος της εργασίας

Η εργασία μας είναι ένα απλοϊκό δισδιάστατο Top-down ηλεκτρονικό παιχνίδι δράσης με το πρόχειρο όνομα "**Titan Slayer**", με την χρήση της λιγότερο διαδεδομένης μηχανής ανάπτυξης παιχνιδιών, αλλά εξίσου δυνατής με άλλες πιο γνωστές (όπως Unity), Godot (v. 3.4,2). Στο παιχνίδι, ο παίκτης αναλαμβάνει τον ρόλο ενός χαρακτήρα που βασικό του όπλο καθίσταται ένα σπαθί. Σκοπός του παιχνιδιού είναι να ουδετεροποιήσεις όλα τα τέρατα αρχηγούς (ή Bosses) που για απλότητα θα αναφερόμαστε σε αυτούς με το όνομα "Τιτάνες", ο καθένας έχοντας τις δικές του ικανότητες και τεχνάσματα. Μέχρι στιγμής, λόγω της φύσεως της εργασίας και του απαιτούμενου χρόνου που χρειάζεται, το παιχνίδι κατέχει μόνο έναν Τιτάνα και 2 διαφορετικές περιοχές χωρισμένες από ένα loading zone (Τον κόσμο, και την περιοχή που παλεύει ο παίκτης τον Τιτάνα). Ωστόσο, το project είναι φτιαγμένο με τρόπο ώστε να μπορεί να **επεκταθεί** και να είναι πιο εύκολος ο τρόπος δημιουργίας νέων Τιτάνων και περιοχών. Είναι δηλαδή το **template** για την κατασκευή τέτοιου τύπου παιχνιδιού.

## 2. Τα κουμπιά και ο χειρισμός

Σε αυτήν την φάση, ο παίκτης έχει στην κατοχή του τα εξής κουμπιά με τις εξής λειτουργικότητες:

- **Movement Buttons**(Πάνω βελάκι, Κάτω βελάκι, δεξί βελάκι, αριστερό βελάκι) : Τα κουμπιά με το οποίο ο παίκτης κινείται
- **Attack button**(Z): Το κουμπί με το οποίο ο παίκτης μπορεί να βαρέσει. Η επιλογή μιας κατεύθυνσης με τα βελάκια και αυτού του πλήκτρου θα έχουν σαν αποτέλεσμα ο παίκτης να χτυπήσει προς τα εκείνη την κατεύθυνση (συνολικά 4 κατευθύνσεις)
- **Dash button** (X): Το κουμπί με το οποίο ο παίκτης μπορεί να αποκτήσει μια ξαφνική ώθηση προς την κατεύθυνση που κινείται. Αξίζει να αναφερθεί πως όταν ο παίκτης βρίσκεται σε αυτήν την κατάσταση δεν μπορεί τόσο να επιτεθεί, **ούτε όμως να του επιτεθούν**

**Τιτάνες** καθώς είναι στιγμιαία “άτρωτος” (το βαφτίζουμε“invulnerability”)

- **Accept button** (Enter): Το κουμπί με το οποίο ο παίκτης “αποδέχεται” prompts, συγκεκριμένα για το δικό μας project να μπορεί να **πατήσει τις ενδείξεις menu**.
- **Pause button** (ESC): Το κουμπί με το οποίο ο παίκτης μπορεί να **βάλει παύση στο παιχνίδι**. Οταν το ξαναπατήσει, το παιχνίδι συνεχίζει

Να σημειωθεί πως το παιχνίδι είναι συμβατό και με χειριστήριο, με κατάλληλες αντιστοιχίες στα παραπάνω κουμπιά

### 3. Συμβατότητα

Το Release του παιχνιδιού που έχουμε επισυνάψει λειτουργεί μόνο σε Windows OS. Όμως, είναι πολύ εύκολο να γίνει export και να είναι συμβατό με αλλά OS όπως Linux και Mac, αλλά και Android, αν επιθυμείτε (δεν έχει γίνει απαραίτητη λειτουργία και bug testing για android οπότε δεν συνιστάται)

# 2. ΟΙ ΔΗΜΙΟΥΡΓΟΙ

Η εργασία αυτή δομήθηκε από 2 άτομα τον **Κώτση-Πανακάκη(3180094)** και τον **Γονατά (3180034)**. Ο Γονατάς ασχολήθηκε πιο πολύ με τα assets του κόσμου και την δομή του (UI, Animations, Tilesets, κτλ.) ενώ ο Κώτσης με τους μηχανισμούς του παιχνιδιού (Κίνηση, collisions, mechanics κτλ.). Παρόλα αυτά η εργασία ήταν ομαδική και είμαστε και οι δύο γνώριμοι με το project και μπορούμε να δουλέψουμε πάνω σε αυτά που έχουμε φτιάξει ανεξάρτητα της δουλειάς που μας ανατέθηκε.

# 3. ΠΕΡΙΒΑΛΛΟΝ

Σε αυτό το κομμάτι της αναφοράς θα κάνουμε μια ανάλυση του τρόπου με τον οποίο δομήθηκε αυτή η εργασία στα διάφορα υποκεφάλαια, καθένα εκ των

οποίον θα αναφέρεται σε μια από τις 4 θεμελιώδεις οντότητες της Godot, οι οποίες είναι

1. **Assets**
2. **Scenes**
3. **Nodes**
4. **Scripts**

## 1. Assets

Ένα asset αποτελεί ένα αρχείο το οποίο χρησιμοποιείται για την αναπαράστασή, ενός Object (ή και για την ενίσχυση του) στον εικονικό κόσμο του παιχνιδιού. Για παράδειγμα, η παρακάτω εικόνα του παίκτη μπορεί να θεωρηθεί ένα PNG asset που χρησιμοποιείται για την αναπαράσταση του.

Επιπλέον, ο ήχος χτυπήματος του τιτάνα “**golem_hit.wav”** μπορεί να θεωρηθεί ένα asset ήχου που αναπαριστά τον ήχο που ακούγεται όταν προκαλούμαι ζημιά στον Τιτάνα. Γενικά, ένα asset το αξιοποιούμε όταν θέλουμε να οπτικοποιήσουμε ή να ακούσουμε κάτι στο πολυμέσο μας. Συνήθως είναι προσκολλημένα πάνω σε Sprite ή audio streamer κόμβους, ή αλλιώς Nodes.

## 2. Nodes

Στην Godot για να αναπαραστήσουμε ένα Entity (Παίκτης, Τιτάνας, menu button κτλ.) χρειάζεται να χρησιμοποιήσουμε Nodes. Τα nodes απαρτίζουν ένα ενιαίο Entity, το οποίο μπορεί να χρησιμοποιήσει όσα χρειάζεται για την εύρυθμη λειτουργία τους. Ένα node περιέχει δικές του ιδιότητες και μεθόδους που χρησιμοποιούνται για την εκτέλεση λειτουργιών στα οποία ειδικεύονται. Για παράδειγμα, ένα **KinematicObject2D** node περιέχει συναρτήσεις κίνησης και

σύγκρουσης. Ή ένα απλό **Node2D** περιέχει την θέση του σε σχέση με την σκηνή που βρίσκεται. Συνεπώς, υπάρχει μεγάλη απλούστευση των λειτουργιών και του απαραίτητου κώδικα που χρειάζεται για την διεκπεραίωση του project, που είναι κάτι που διαφοροποιεί τις βιβλιοθήκες γραφικών από μία μηχανή ανάπτυξης παιχνιδιών όπως η Godot.
Φυσικά, ένα node μπορεί να κληρονομεί από ένα άλλο (Μονή κληρονομικότητα) και όλα κληρονομούν από το **Object Node**. Ένα node ορίζεται μέσα σε μια σκηνή και μπορεί να έχει αυθαίρετο πλήθος από Child Nodes (Κατά σύμβαση, ένα child node μετακινείται ή απελευθερώνεται κάθε φορά που κάνει το ίδιο και το parent Node).

## 3. Scenes

Μια σκηνή στην Godot αποτελεί την συλλογή ορισμένων Nodes με σκοπό να αρχικοποιηθούν ένα ή παραπάνω entities. Μια σκηνή και όλα της τα nodes μπορούν να τροποποιηθούν καταλλήλως από τον Editor της Godot και να αναλάβουν την θέση τους στον χώρο της σκηνής και να αρχικοποιηθούν οι default ιδιότητες τους. Για παράδειγμα, η παρακάτω είναι η σκηνή του κόσμου που πρωτοεμφανίζεται ο παίκτης.

Όπως βλέπετε, στην σκηνή αυτή είναι ορισμένες στο Scene tab όλα τα Nodes που απαιτούνται για την σωστή αναπαράσταση της, καθώς και οι αρχικές ιδιότητες τους.

Ταυτόχρονα, μια σκηνή μπορεί να είναι και ένα entity που θέλουμε να επαναχρησιμοποιήσουμε ξανά και ξανά, όπως ένα projectile που εμφανίζει ο Τιτάνας για να επιτεθεί, ή ακόμη και ο ίδιος ο παίκτης, προκειμένου να μην είναι αναγκαίος ο επαναπρογραμματισμός τους. Η παρακάτω σκηνή είναι η σκηνή του Τιτάνα, η οποία χρησιμοποιείται σαν external scene στην σκηνή που τον πολεμάει ο παίκτης.


Είναι καλή πρακτική να χρησιμοποιούμε εξωτερικές σκηνές τόσο για entities που θέλουμε να χρησιμοποιούμε πολλαπλές φορές, όσο και για entities που θέλουμε να μπορούμε να τα διαχειριζόμαστε και να τα κάνουμε edit με πιο οργανωμένο τρόπο

## 4. Scripts

Ένα script αποτελεί τον βασικό τρόπο με τον οποίο ελέγχουμε την συμπεριφορά ενός node όταν τρέχει το παιχνίδι μας. Κάθε node μπορεί να έχει από ένα script αρχείο το οποίο υπαγορεύει τους κανόνες και επιπλέον ιδιότητες με τις οποίες θα λειτουργεί. Σε ένα script κάνουμε override τον κώδικα που τρέχει τόσο όταν πρωτοδημιουργείται στον εικονικό κόσμο το Node, όσο και για κάθε στιγμή που περνάει (delta). Αυτές είναι οι συναρτήσεις **_ready()** και **_process(delta)** αντίστοιχα. Έξω από αυτές τις συναρτήσεις μπορούμε να αρχικοποιήσουμε και επιπλέον μεταβλητές ή μεθόδους που θα συμβάλλουν στην εκπλήρωση τον εξειδικευμένων αναγκών που έχουμε για ένα node, όπως φέρ’ ειπείν το διάνυσμα ταχύτητας ενός Kinematic Node **(Vector2)** ,ή τον καταστροφέα ενός node **(συνάρτηση destroy())**.

Να επισημανθεί πως η προγραμματιστική γλώσσα της Godot είναι μοναδική και ονομάζεται GDScript (τα αρχεία έχουν κατάληξη .gd). Είναι μια γλώσσα που θυμίζει σε μεγάλο βαθμό την πιο διαδεδομένη γλώσσα Python 3

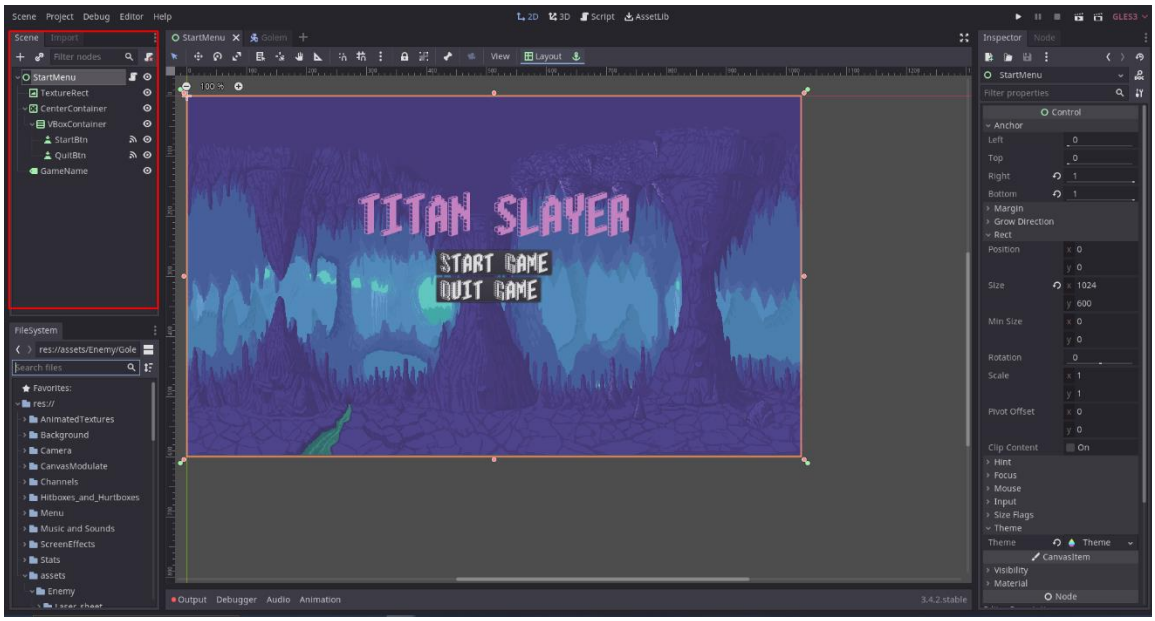
Μια πιο εξειδικευμένη λειτουργία των script της Godot είναι τα λεγόμενα **Signals**, τα οποία είναι "σήματα" τα οποία εκπέμπονται όταν συμβαίνει ένα συγκεκριμένο συμβάν μέσα σε ένα κομμάτι κώδικα. Κάνουμε connect αυτά τα signals σε συναρτήσεις του script μας (με την μέθοδο **Scene.connect(from, to, functionToCall)** και εκπέμπουμε ένα σήμα με την μέθοδο **emit_signal(signal, arg1, arg2, …).** Κάποια βασικά και εύχρηστα signals είναι έτοιμα από τα ίδια τα nodes. Το πιο χαρακτηριστικό παράδειγμα, είναι ένα Signal που εκπέμπεται από ένα **Area2D** node για όταν ένα σώμα εισέρχεται μέσα στο **CollisionShape2D** node του. Όταν συμβαίνει αυτό, εκπέμπεται signal από το Area2D node προς οποιοδήποτε άλλο node που είναι συσχετισμένο μαζί του θέλουμε (ακόμη και τον εαυτό του), οπότε καλείται η συνάρτηση **_on_Area2D_body_entered(),** της οποίας την λειτουργικότητα την ορίζουμε εμείς. Για παράδειγμα, μπορούμε να θέλουμε όταν επιτιθόμαστε στον Τιτάνα, η ζωή του να μειώνεται. Αυτό γίνεται με χρήση signal, και δεν είναι κάτι που κοιτάει συνεχώς η _process() του τιτάνα.

Μπορούμε επίσης κάποιες global μεταβλητές (ή και μεθόδους) να της έχουμε σε καθολικά scripts τα οποία κάνουμε Autoload μέσω της Godot όταν ξεκινάει για πρώτη φορά το παιχνίδι μας. Το πιο απλό παράδειγμα σε αυτό είναι η ζωή του παίκτη, που πρέπει να παραμένει ίδια από σκηνή σε σκηνή (να μην αυξάνεται ή να μειώνεται στην εναλλαγή των σκηνών)

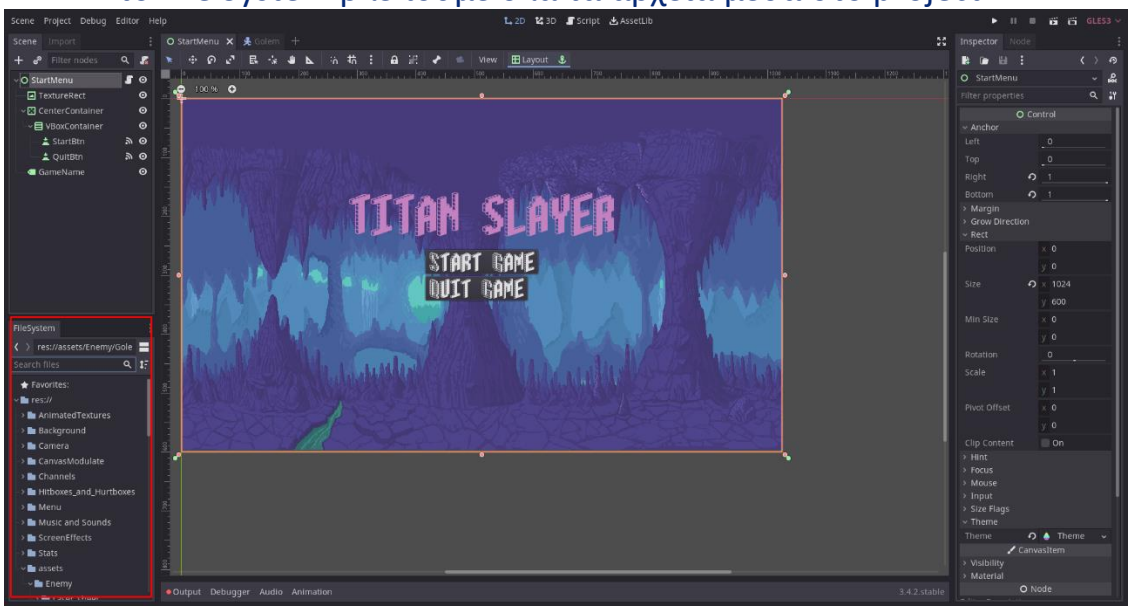
## 5. Editor

Ο editor είναι το περιβάλλον εργασίας της Godot και εκεί που θα κάνουμε edit τις σκηνές.
Θα επισημαίνουμε το βασικό layout.

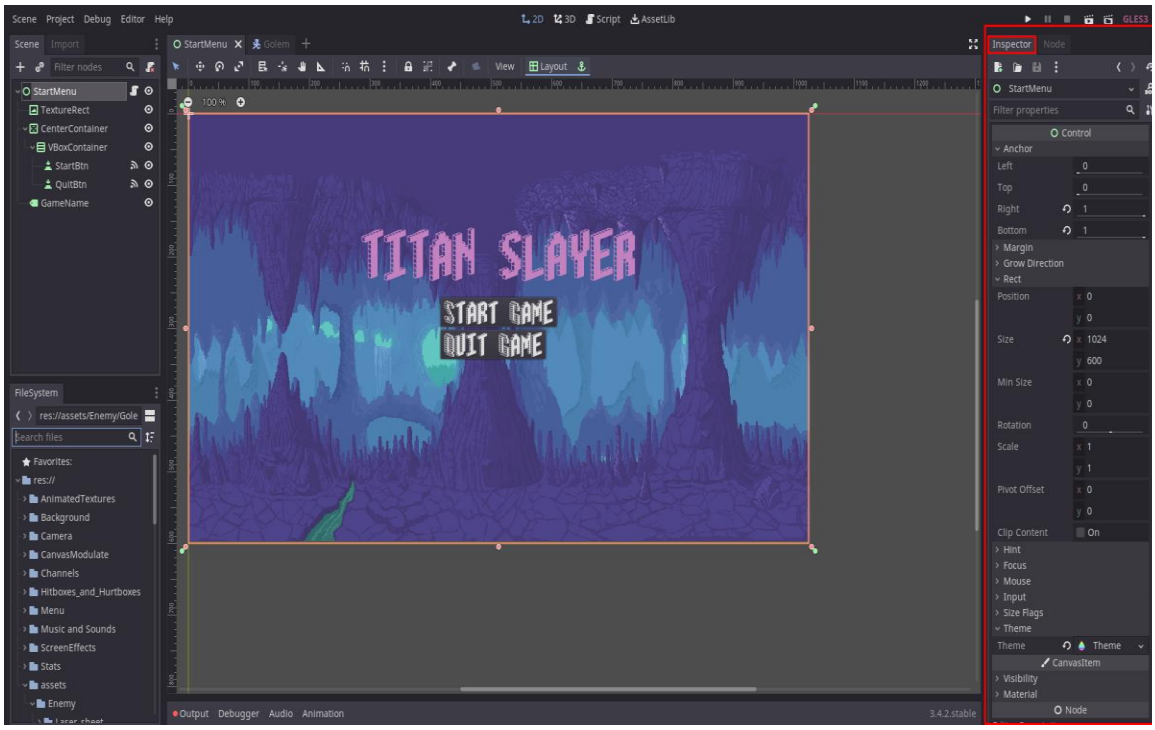
- Στο Scene μπορούμε να φτιάξουμε και να τροποποιήσουμε τα nodes της σκηνης



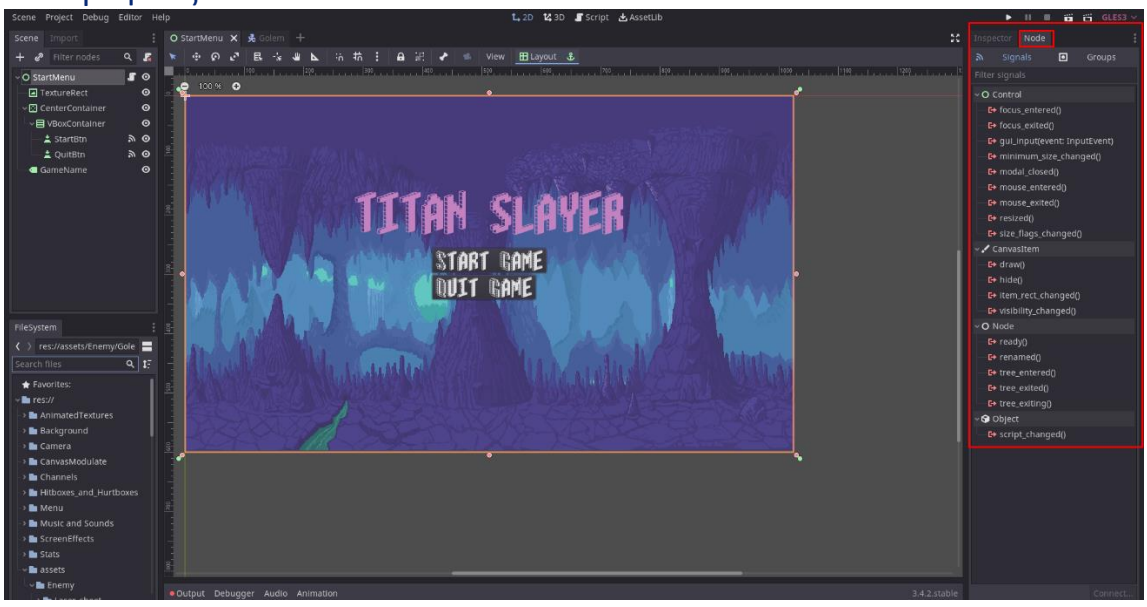

- Στο File System βλέπουμε όλα τα αρχεία μέσα στο project




- Στο inspector μπορούμε να τροποποιήσουμε ελεύθερα τις μεταβλητές του node καθώς και επιπλέον “exportable” μεταβλητές που έχουμε ορίσει σε τυχόν script που έχει προστεθεί πάνω σε αυτό




- Στο Node tab συνδέουμε τα signals του node με τις αντίστοιχες συναρτήσεις



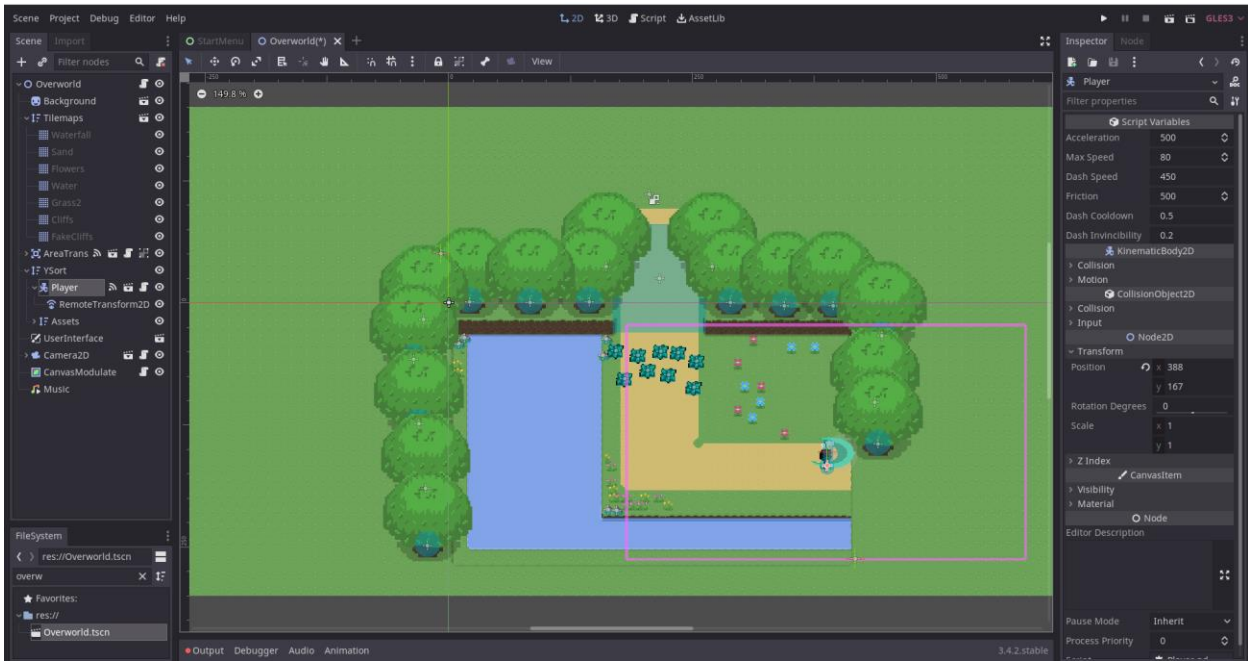

Αποτελεί τον βασικό τρόπο που δουλεύουμε στην Godot για ευκολία τοποθέτησης και οπτικοποίησης της δημιουργίας του παιχνιδιού

# 4. ΑΝΑΛΥΣΗ

Δεν θα αναλύσουμε πως υλοποιήσαμε όλα τα components του project, πάρα θα εστιάσουμε στα πιο βασικά κομμάτια που χαρακτηρίζουν την εργασία μας καθώς και στην ανάλυση κάποιων σημαντικών Node

## 1. Main Scene

Η κύρια μας σκηνή φαίνεται παρακάτω




Όπως φαίνεται, σε αυτήν την σκηνή είναι ορισμένα κάποια nodes, που στην πραγματικότητα είναι απλά εξωτερικές σκηνές (καθώς τα χρειαζόμαστε και σε άλλες σκηνές και όχι μόνο σε αυτήν). Τα σημαντικότερα είναι:

- Τα **tilesets**(Tileset node), τα οποία αποτελούν τα θεμέλια τούβλα του κόσμου, δηλαδή τους τους τοίχους, τις λίμνες, τα λουλούδια κτλ.

- Την **κάμερα** (Camera2D node), η οποία είναι προγραμματισμένη να ακολουθά τον παίκτη καθόλη την διάρκεια του παιχνιδιού και ορίζει το μέγιστο όριο της σκηνής που μπορούμε να δούμε (το ορίζουμε εμείς)

- Τον **παίκτη** (KinematicObject2D node), τον οποίο ελέγχουμε. Το node του του επιτρέπει να συγκρούεται με τοιχώματα και να έχει έτοιμες συναρτήσεις κίνησης

- Το **γρασίδι** (AnimatedSprite2D node), που είναι στόλισμα για τον κόσμο, αλλά και εμπόδιο. Ο παίκτης μπορεί να τα κόψει και θα παίξει κατάλληλο εφέ.

- Την **περιοχή μετάβασης** (Area2D node), η οποία κάνει την μετάβαση από αυτήν την σκηνή στην επόμενη. Οι κανόνες με τους οποίους γίνεται αυτή η μετάβαση είναι γραμμένη στο script της , και χρησιμοποιούνε global μεταβλητές από το **PlayerTransitionState.gd** για να είναι η μετάβαση ομαλή για να αναβαθμίζει την εμπειρία και το immersion του παίκτη. Το που θα μας μεταφέρει αυτή η περιοχή και από ποια μεριά κοιτάει βγαίνοντας ο παίκτης είναι exported μεταβλητές που ορίζουμε για κάθε στιγμιότυπο μιας τέτοιας σκηνής στον Inspector

- Το **User Interface** (CanvasLayer Node) , το οποίο περιέχει το Pause Screen και την ζωή του χρήστη, η οποία αναπαρίσταται με καρδίες οι οποίες αδειάζουν όσο ο παίκτης αποδυναμώνεται (περισσότερες πληροφορίες μετά)

Μεταξύ άλλων, σημαντική δουλεία κάνει και το **Ysort** Node. Οι θυγατρική του κόμβοι ταξινομούνται με βάση το ύψος, ώστε να έχουμε κατάλληλο βάθος για ένα Top-down παιχνίδι (ο παίκτης μπροστά από της πέτρες και τους θάμνους κτλ).

## 2. Ο Παίκτης

Η σκηνή του παίκτη **(Player.tscn)** (μαζί με την σκηνή του Τιτάνα που δουλεύει με παρόμοιο τρόπο) λογικά ήταν η πιο απαιτητική μέσα στο project. Για να γίνει κατανοητή η λειτουργία του θα αναλύσουμε τα πιο βασικά του Nodes που τον αποτελούν. Η τεκμηρίωση του Τιτάνα (Golem.tscn) γίνεται με παρόμοιο τρόπο, με την διαφορά ότι είναι AI και οι κινήσεις του δεν εξαρτώνται από τα input μας

## 2.1 Sprite

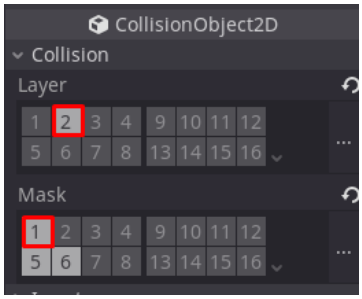
Το sprite του παίκτη είναι ένα μεγάλο **spritesheet** που περιέχει πάνω από 60 frames του, σειριακές ομάδες εκ των οποίων μπορούν να δημιουργήσουν την ψευδαίσθηση κίνησης, γνωστό ως **Animation**. Η default τιμή του frame είναι ίση με 1 όταν τον βλέπουμε στο editor, αλλά στο παιχνίδι όταν κάθεται ακίνητος παίζει το κατάλληλο idle animation

## 2.2 KinematicObject2D

Το βασικό Node της Player.tscn. Όλα τα υπόλοιπα Nodes είναι θυγατρικά αυτού του Node. Επιτρέπει στον παίκτη τόσο να καλέσει συναρτήσεις κίνησης όσο και σύγκρουσης με άλλα entities στο παιχνίδι. Κληρονομεί από το Area2D node, καθώς είναι και αυτό ένα 2D Area Node. Κάθε KinematicObject2D πρέπει απαραίτητα να έχει ένα **CollisionShape2D** node, το οποίο ορίζει το σχήμα της περιοχής που θα ανιχνεύει για τυχών κρούσεις. Όταν το σχήμα ανιχνεύει σύγκρουση, ο παίκτης σταματάει. Αυτό επιτυγχάνεται με την συνάρτηση **move_and_slide(velocity)** που παίρνει σαν όρισμα το διάνυσμα κίνησης που συνεχώς ανανεώνεται στην συνάρτηση _process(delta), οπού delta η χρονική διάρκεια που χρειάζεται για κάθε κλήση της _process (Σταθερή τιμή, ορίζεται από τα FPS του παιχνιδιού). Να σημειωθεί πως στο παιχνίδι το Collision Shape του παίκτη είναι σχήμα κάψουλας, προκειμένου να δημιουργηθεί καλύτερη υφή και τσούλιμα κατά την σύγκρουση

Είναι καλή στιγμή να συζητήσουμε για τον τρόπο που η Godot χειρίζεται τις συγκρούσεις. Επειδή μπορεί να θέλουμε ο παίκτης μας να μην συγκρούεται με κάθε αντικείμενο που έχει Area2D node πάνω του, η Godot επιτρέπει την χρήση Layer σύγκρουσης. Στην παρακάτω φωτογραφία βλέπουμε πως το σώμα του παίκτη ανήκει στο Layer της σύγκρουσης 2 με το όνομα “Player” ενώ έχει μάσκα το Layer 1 με όνομα “World”. Με πιο απλά λόγια, αυτό σημαίνει πως ο παίκτης ανήκει στο Player layer, δηλαδή αυτό είναι το ID του όσον αφορά το πως θα τον βλέπουν αλλά Area2D nodes, ενώ ψάχνει να συγκρουστεί με objects που ανήκουν στο World layer. Στην δικιά μας περίπτωση, αντικείμενα στο World layer αποτελούν οι τοίχοι, οι πέτρες, τα δέντρα, οι πλευρές της λίμνης αλλά όχι τα

λουλούδια και τα κλαδιά. Δηλαδή, ο παίκτης μπορεί να συγκρουστεί με τα πρώτα, αλλά όχι με τα δεύτερα, καθώς αυτά δεν είναι στο World layer. Την ίδια λογική επιστρατεύουμε και πιο μετά όταν θέλουμε να δούμε τί μπορεί να βλάψει τον παίκτη. Δεν πρέπει κάθε σύγκρουση να είναι βλαβερή για την ζωή του παίκτη, οπότε το να έχουμε ξεχωριστά layers για την σύγκρουση του παίκτη και για την ζημιά του είναι πολύ σημαντικό και αναγκαίο.

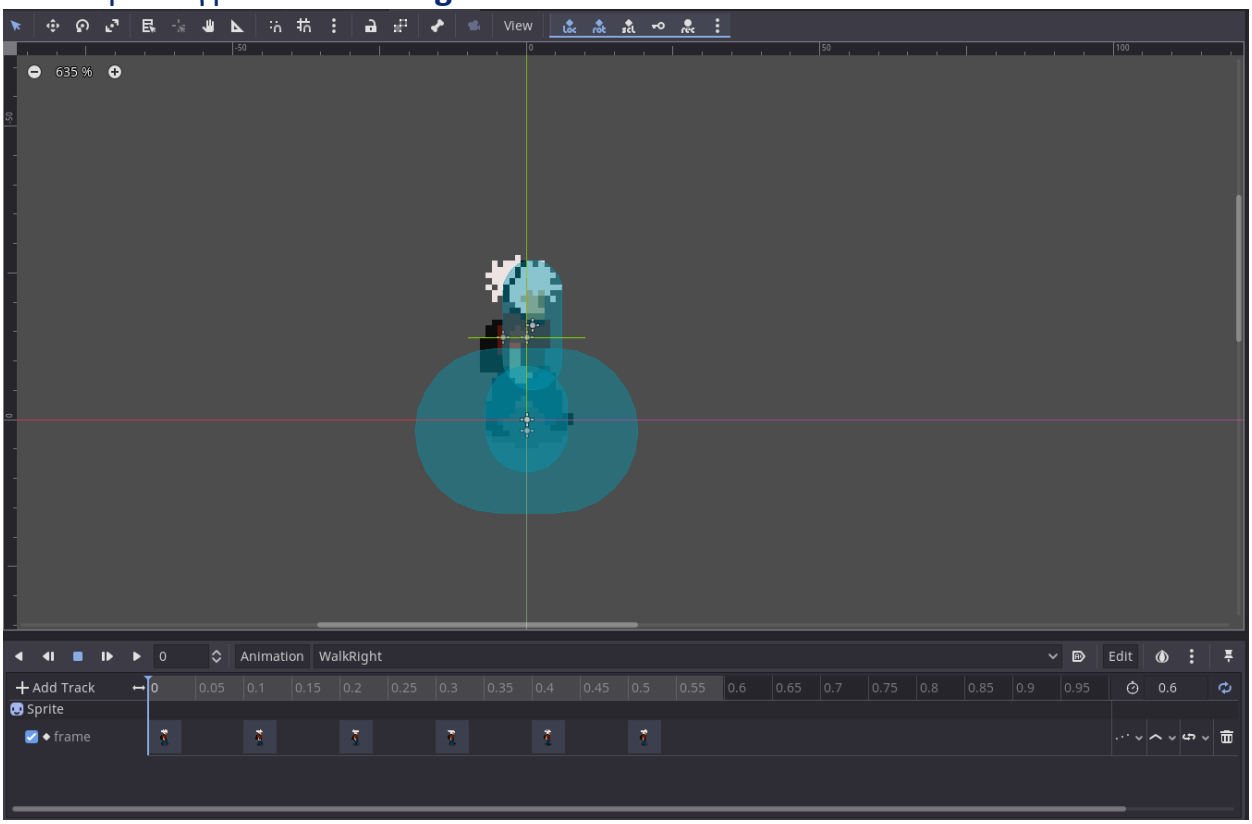



Τέλος, αξίζει να αναφερθεί πως στην κίνηση του παίκτη δεν έχουμε μόνο μια ταχύτητα, παρά μια επιτάχυνση **(ACCELERATION)** η οποία φτάνει σε μια μέγιστη

ταχύτητα **(MAX_SPEED)**, και όταν ο παίκτης σταματάει, μειώνεται με έναν ρυθμό **(FRICTION)**. Με τον τρόπο αυτό επιτυγχάνουμε την ομαλή κίνηση του παίκτη και έχουμε την ψευδαίσθηση εφαρμογής νόμων της φυσικής πάνω σε αυτήν.

## 2.3 AnimationPlayer

Με την χρήση αυτού του node μπορούμε να αποθηκεύσουμε σε προσβάσιμη μορφή όλα τα animation του παίκτη προς όλες τις κατευθύνσεις. Ο παίκτης έχει animations για **IDLE, WALK, ATTACK, DASH** προς όλες τις κατευθύνσεις και ένα για τον θάνατο **DEATH**. Συνολικά δηλαδή 17 διαφορετικά animations. Ακολουθεί ένα παράδειγμα του **WalkRight**

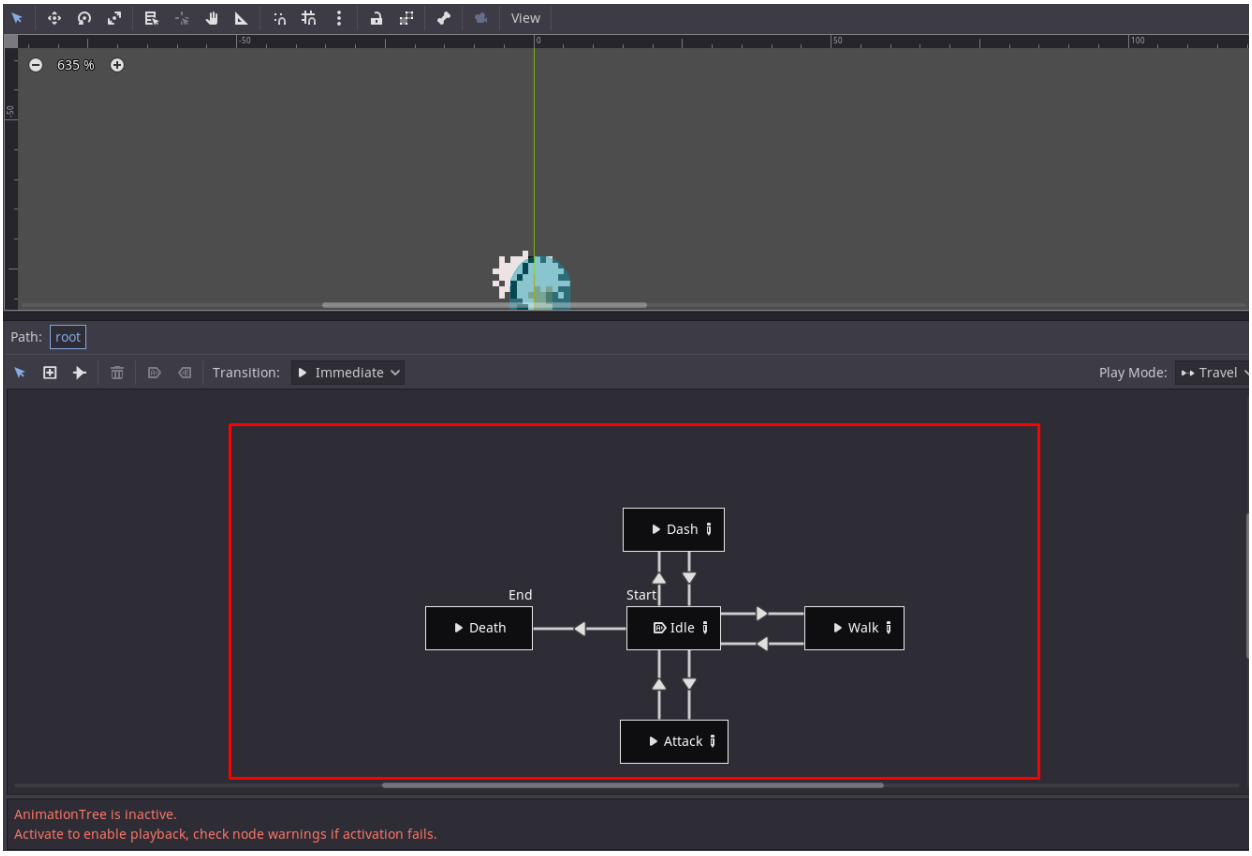



Ουσιαστικά φτιάχνουμε το animation το οποίο καλούμε μέσα στο script όταν έρχεται η ώρα να παίξει, **καθώς και τον χρόνο για τον οποίο θα παίζει**. Για παράδειγμα, αν πατήσουμε το Z, ο παίκτης μαζί με την επίθεση, πρέπει να παίξει και το animation για αυτήν. Ωστόσο, είναι αρκετά δύσκολο και καθόλου αποτελεσματικό να χρειάζεται κώδικας για κάθε κατεύθυνση που θέλουμε να

έχουμε animation. Τι και αν αποφασίζαμε να βάλουμε 8 κατευθύνσεις; Αυτό το πρόβλημα λύνουν ακριβώς τα επόμενα Nodes που θα συζητήσουμε

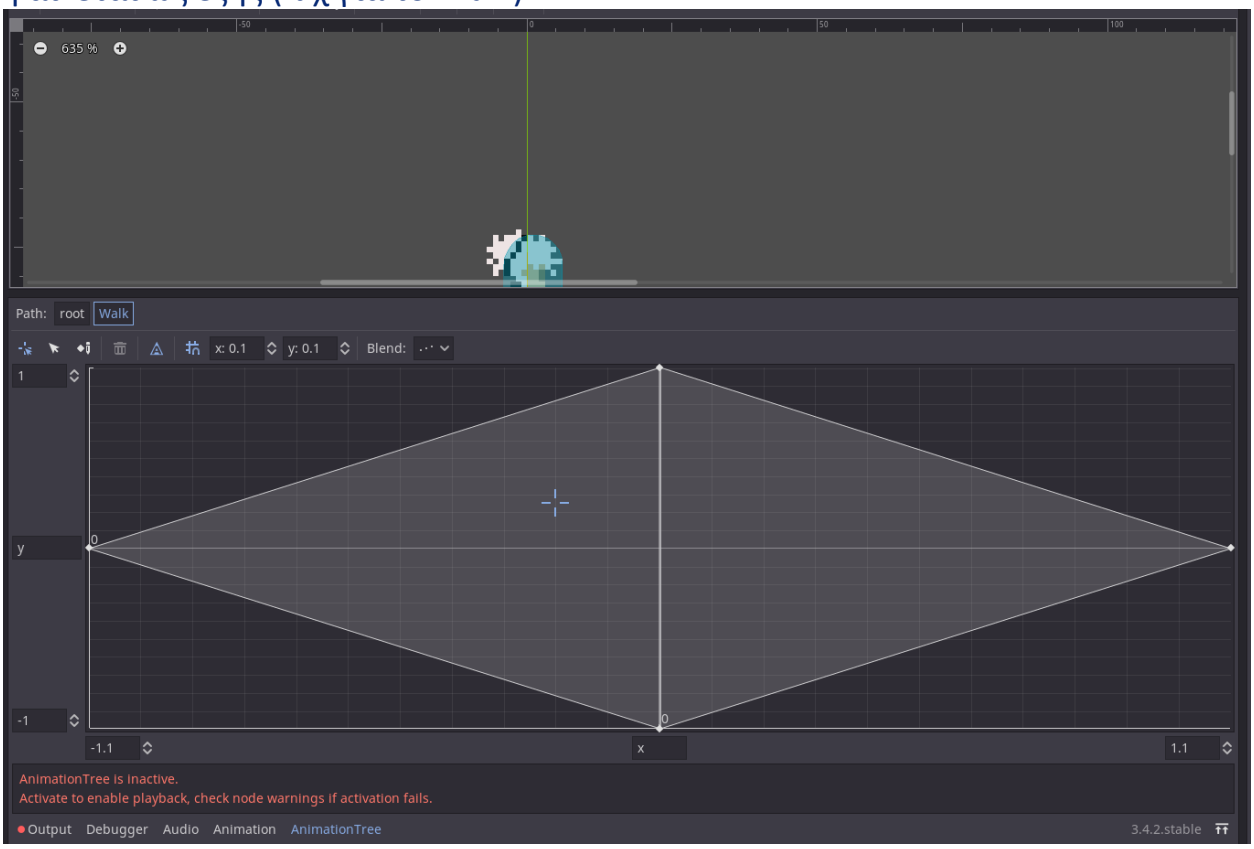
## 2.4 AnimationTree

Ένα AnimationTree Node είναι πολύ χρήσιμο εργαλείο για να μπορούμε εύκολα να μετακινηθούμε από animation σε animation, καθώς μας επιτρέπει να τα αναπαραστήσουμε σαν **πεπερασμένη μηχανή καταστάσεων**. Παρακάτω ένα παράδειγμα του AnimationTree του παίκτη




Βλέπουμε δηλαδή πως ο παίκτης ξεκινάει από την κατάσταση animation IDLE και μπορεί να μεταβεί στις καταστάσεις DASH, WALK, ATTACK μπρος πίσω, και να μεταβεί στην κατάσταση DEATH και να μην επιστρέψει ποτέ. Θυμίζει πολύ ένα απλό DFA. Αυτήν την οπτικοποίηση την αξιοποιούμε με κατάλληλες μεθόδους μέσα στον κώδικα για να κινηθούμε μεταξύ animations.
Μαζί με αυτό όμως, επιτρέπει την αντιστοίχιση κατεύθυνσης στο animation. Οπότε κάθε φορά που πατάμε το πλήκτρο για να αλλάξουμε κατεύθυνση, το animation tree αυτόματα θα το εντοπίσει και θα αλλάξει στο κατάλληλο animation. Η εκχώρηση γίνεται μέσω του BlendSpace2D που κατέχει η Godot και

φαίνεται ως εξής (π.χ για το Walk)




Όπως φαίνεται, κάθε κατεύθυνση είναι αντιστοιχισμένη σε ένα κανονικοποιημένο Vector2 value ((0,1), (1,0), (0,-1),(-1,0)) που δηλώνει την κατεύθυνση κίνησης (η τιμή 1.1 χρησιμοποιείται για να δώσει προτεραιότητα στο «πάνω» ή «κάτω» walk animation σε περίπτωση που επιλεχθεί κάποια διαγώνια κίνηση). Όσον αφορά το πως τα καλούμε στο Player.gd, στην παρακάτω φωτογραφία φαίνεται πως περνάμε το **input_vector** σαν όρισμα, ώστε να ξέρει το animationTree ποια κατεύθυνση πατάμε.

```
animationTree.set("parameters/Idle/blend_position", input_vector)
animationTree.set("parameters/Walk/blend_position", input_vector)
animationTree.set("parameters/Attack/blend_position", input_vector)
animationTree.set("parameters/Dash/blend_position", input_vector)
```

Το animationTree δίνει αφορμή για να μιλήσουμε για το πώς καταλαβαίνουμε την κατάσταση κίνησης που βρίσκεται ο παίκτης κάθε χρονική στιγμή. Όπως αναφέραμε, τα animationTrees είναι απλά FSM με τα οποία ταξιδεύουμε από animation σε animation. Έτσι λοιπόν, είναι λογικό να σκεφτούμε πως το ίδιο θα πρέπει να συμβαίνει με την κατάσταση του παίκτη. Η _process(delta) στον

παίκτη βασίζεται και αυτή στο μοντέλο καταστάσεων. Δηλαδή, όπως φαίνεται στην παρακάτω φωτογραφία, με την βοήθεια ενός Enumerator, ο παίκτης έχει διάφορες καταστάσεις που μπορεί να βρίσκεται κάθε delta χρονική διάρκεια, και ανάλογα με  τίς συνθήκες που βρίσκεται να αλλάζει.

```
func _physics_process(delta):
    match state:
        MOVE:
            move_state(delta)
        ATTACK:
            attack_state()
        DASH:
            dash_state()
        REBOUND:
            rebound_state(delta)
        KNOCKBACK:
            knockback_state(delta)
        DEATH:
            death_state()
        ENTER:
            enter_state()
        WATCH:
            watch_state()
```

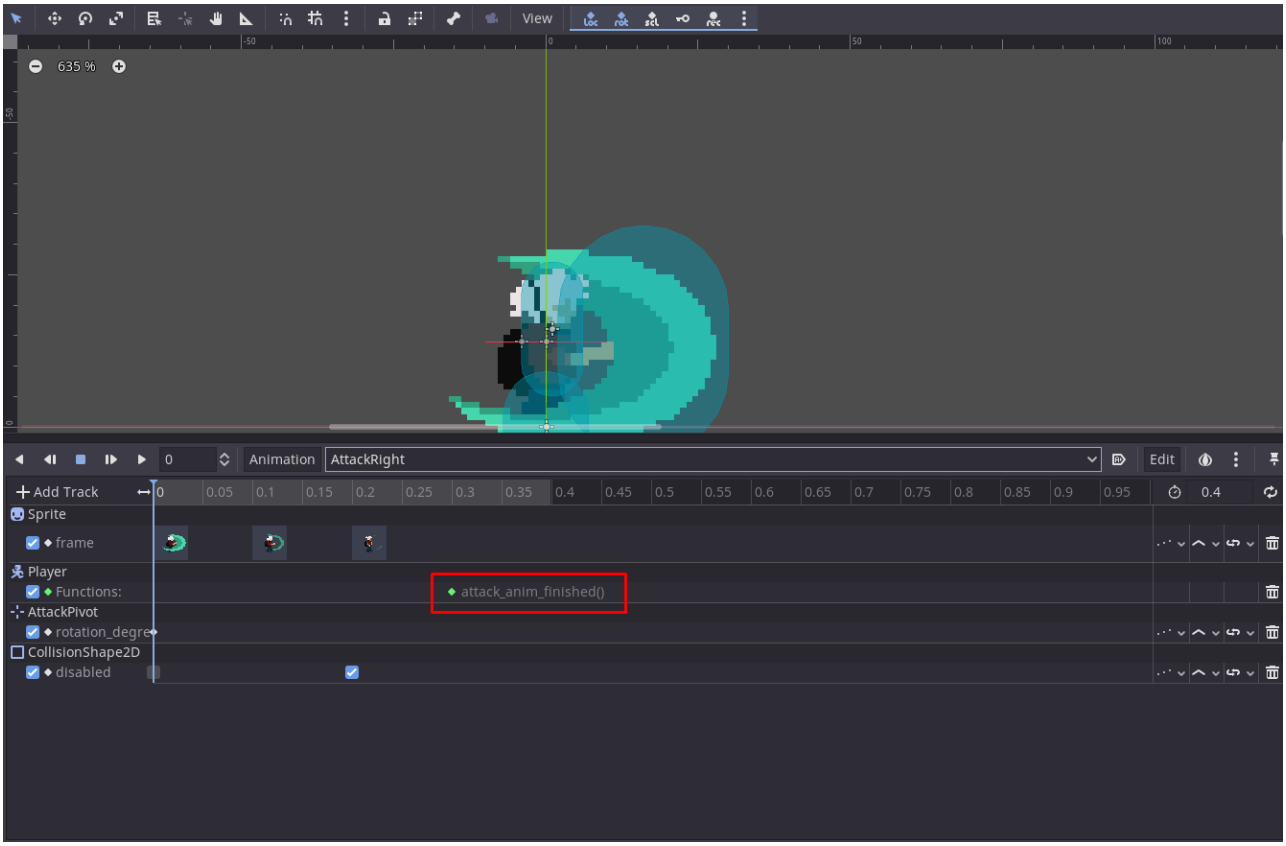
Κάθε delta δευτερόλεπτα, όταν καλείται αυτή η συνάρτηση, συμβαίνει αντιστοίχιση στην τρέχουσα κατάσταση, πράγμα το οποίο γίνεται δυναμικά από τις βοηθητικές μας συναρτήσεις όπως φαίνεται εδώ σε ένα δείγμα

```
15
16 func attack_state(): #used when attacking
17     velocity = Vector2.ZERO
18     animationState.travel("Attack")
19
20 func dash_state(): #Used when dashing
21     velocity = dash_vector * DASH_SPEED
22     animationState.travel("Dash")
23     afterimage()
24     move()
25
26 func rebound_state(delta): #used when hitting the Golem/Boss
27     velocity = velocity.move_toward(Vector2.ZERO, FRICTION * delta)
28     swordHitbox.set_deferred("disabled", true)
29     move()
30
```

Μπορεί να έχετε τον προβληματισμό στο **attack_state()** ότι ο κώδικας είναι πολύ μικρός για να δικαιολογήσει τις αλλαγές που κάνουμε στις καταστάσεις και πως δεν υπάρχουν οι επαρκείς γραμμές κώδικα. Παρόλα αυτά, με τρόπο που θα εξηγήσουμε προσεχώς, αυτό είναι εφικτό καλώντας την συνάρτηση **attack_anim_end()**

```
func attack_anim_finished():
    state = MOVE
```

Αυτή η κλήση γίνεται πολύ εύκολα μέσα από τις λειτουργίες του AnimationPlayer του παίκτη, καθώς μας επιτρέπει όταν το τελειώνει το animation να καλούμε μια συγκεκριμένη συνάρτηση από κάποιο script στην σκηνή μας, πράγμα που κάνει τον συγχρονισμό του animation και της αλλαγής της de facto κατάστασης του παίκτη πιο απλοποιημένο όπως φαίνεται παράδειγμα εδώ στο animation του **AttackRight**




Να σημειώσουμε πως αυτή η διαδικασία λέγεται **keying** και μπορούμε να κάνουμε key ακόμη και ιδιότητες ενός node που ανήκει στον παίκτη (όπως ίσως παρατηρήσατε στην φωτογραφία όπου στρίβουμε την κατεύθυνση του hitbox ή κάνουμε enable το hitbox του). Θυμίζει κάπως τα signals.

## 2.5 Hurtbox και Hitbox

Αυτά τα δύο Nodes αποτελούν 2 από τα 3 Area2D nodes που έχει ο παίκτης. Το πρώτο χρησιμοποιείται για την ανίχνευση ζημιάς πάνω στον παίκτη από οποιαδήποτε πηγή, ενώ το δεύτερο για την πρόκληση ζημιάς σε εξωτερικό στόχο (στην δικιά μας περίπτωση ο Τιτάνας ή το γρασίδι). Έχουμε ορίσει 2 σκηνές **Hurtbox.tscn** και **Hitbox.tscn** με exportable μεταβλητές, ώστε το καθένα που θα ανήκει σε διαφορετικό entity να έχει τις δίκες του ιδιότητες και CollisionShape2D (τα λέηζερ του Τιτάνα έχουν πιο πολύ μεγάλο συντελεστή ζημίας από το σπαθί του παίκτη)

Έχουν και τα δύο κατάλληλα Layers και Masks έτσι ώστε να μπορούν να συγκρούονται μόνο με τα κατάλληλα Area2D nodes και να εκπέμπουν μόνο

επιθυμητά signals (Δηλαδή το Hitbox του παίκτη να ψάχνει Hurtbox εχθρού και το Hitbox εχθρού να ψάχνει μόνο Hurtbox παίκτη). Αυτό μπορείτε να το δείτε στον Inspector του κάθε Node.

Κάθε hurtbox, όταν έρχεται σε επαφή με κατάλληλο hitbox εκπέμπει σήμα το οποίο μειώνει την ζωή του κατόχου του ανάλογα με την τιμή του συντελεστή ζημίας (**area.damage**) και του δίνει προσωρινό invincibility προκειμένου να μην μπορεί να δέχεται συνεχόμενη (και άδικη) επίθεση. Συγκεκριμένα, το hurtbox του παίκτη του δίνει και μια ώθηση προς τα πίσω προς την κατεύθυνση που έγινε το χτύπημα, και ανάλογα και την δύναμη της επίθεσης, οι οποίες είναι ιδιότητες που ορίζονται στο hitbox (τα λέιζερ δεν απομακρύνουν, δεν έχουν μεγάλο συντελεστή οπισθοώθησης)

```
func _on_Hurtbox_area_entered(area):
    for i in area.damage:
        stats.health -= 1
    hurtbox.start_invincibility()

    #var knockback_vector = area.knockbackVector
    var knockback_vector = area.knockbackVector
    var knockback_strength = area.knockback
    velocity = knockback_vector *knockback_strength
    if(state != DEATH):
        state = KNOCKBACK
```

Κάθε hitbox κατά την επαφή του με ένα hurtbox δημιουργεί μια μικρή αντίκρουση που στέλνει τον παίκτη ελαφρώς πίσω ώστε να φαίνεται το βάρος της στιβαρότητας του Τιτάνα (**rebound state**)

## 2.6 Timers

Τα timer nodes βοηθούν αν ποτέ χρειάζεται να κάνουμε καταμέτρηση μέχρις ότου να φτάσουμε σε μια κατάσταση. Για παράδειγμα στον παίκτη, πρέπει να ξέρουμε κάθε πότε μπορεί να κάνει DASH (**DashCooldown)** ή για πόσο μένει άτρωτος κατά την διάρκειά του (**DashInvTimer)**. Εκπέμπουν signals στο timeout τους

### 2.7 AudioStreamPlayer

Χρησιμοποιείται για να παίξουν ένα συγκεκριμένο sound effect που τους έχει ανατεθεί. Παράδειγμα, όταν ο παίκτης επιτίθεται, καλείται η **play()** στο **AttackSFX** node. Στο project χρησιμοποιήσαμε **.wav** αρχεία σαν assets για τα SFX

## 3. Ο Τιτάνας

Όπως αναφέραμε και προηγουμένως, η σκηνή του τιτάνα **Golem.tscn** λειτουργεί με την ίδια λογική όπως του παίκτη. Δηλαδή, οι κινήσεις του Τιτάνα διέπονται από μια μηχανή καταστάσεων ειδικά φτιαγμένη για αυτόν. Η μόνη διαφορά είναι πως δεν είναι δεσμευμένος στα Inputs μας, παρά δρα με βάση διάφορες παραμέτρους, όπως το πόση ζωή έχει, την απόσταση του από τον παίκτη, τυχαιότητα κτλ.

Ωστόσο, αξίζει να σημειωθεί πως υπάρχει μια λειτουργία που δεν εμφανίζεται στο Script του παίχτη, το οποίο είναι η δυναμική δημιουργία στιγμιοτύπων ξένων σκηνών. Ένα παράδειγμα του Τιτάνα είναι το βασικό βλήμα που εμφανίζει από τα χέρια του. Αυτό το entity δεν ανήκει στην σκηνή του Τιτάνα σαν node πάνω του, πάρα γίνεται instance κατ' εντολή του με τον ακόλουθο κώδικα

```
func spawn_projectile():
    var projectile = BULLET_SCENE.instance()
    if(sprite.flip_h):
        projectile.position=self.position + Vector2(-45,-30)
    else:
        projectile.position=self.position + Vector2(45,-30)
    projectile.player=player
    get_parent().add_child(projectile)
```

Έτσι, καλείται στην τρέχουσα σκηνή το βλήμα με τις παραμέτρους που χρειάζεται για να εντοπίσει και να καταδιώξει ελαφρώς τον παίκτη, μέχρις ότου να φύγει μακριά του έξω από τα όρια της σκηνής και εν τέλει να καταστραφεί.

Επιπλέον, υπάρχει και ένα Area2D στο οποίο όταν ο παίκτης μπει μέσα του ξεκινάει η μάχη. Γιατί αλλιώς, όταν θα έμπαινε ο παίκτης στο πεδίο μάχης, ο Τιτάνας θα τον καταδίωκε οπουδήποτε και αν βρισκόταν.

## 4. ΚΑΡΔΙΕΣ

Οι καρδιές είναι το δεύτερο βασικό στοιχείο διεπαφής χρήστη. Είναι η μέθοδος που επιλέξαμε για να οπτικοποιήσουμε την κατάσταση της ζωής τόσο του παίκτη, όσο και του τιτάνα. Εφόσον οι αναφορές για την ζωή των 2 αυτών οντοτήτων είναι global (**PlayerStats.tscn** και **BossStats.tscn)** μπορούμε πολύ εύκολα να τα αντιστοιχίσουμε στην σκηνή **HeartUI** που έχουμε φτιάξει.

Ουσιαστικά, έχουμε 2 διαφορετικά HeartU**I**. Ένα για τον παίκτη και ένα για τον Τιτάνα. Και τα δύο έχουν ένα texture καρδιών που επαναλαμβάνεται συνεχώς, και το μήκος του είναι ανάλογο της ζωής που μένει σε κάθε οντότητα. Άρα, μειώνουμε καταλλήλως το μήκος του ορθογωνίου που έχει κάθε layer καρδιάς προκειμένου να **δημιουργήσουμε την ψευδαίσθηση μείωσης των καρδιών**, ενώ το μόνο που κάνουμε είναι να «αφαιρούμε» το μήκος του κάθε layer.

```
func spawn_projectile():
    var projectile = BULLET_SCENE.instance()
    if(sprite.flip_h):
        projectile.position=self.position + Vector2(-45,-30)
    else:
        projectile.position=self.position + Vector2(45,-30)
    projectile.player=player
    get_parent().add_child(projectile)
```

Περιττό να αναφέρουμε πως όταν η ζωή είτε του παίκτη είτε του τιτάνα φτάσει στο 0, οι καρδιές μηδενίζονται και μπαίνουν σε δικό τους DEATH state μέχρις ότου να απελευθερωθούνε (Τιτάνας)  είτε μέχρις ότου να ξαναφορτώσει η σκηνή (Παίκτης).

# 5. ΠΗΓΕΣ

**Αναγράφουμε πηγές που μας βοήθησαν στην διεκπεραίωση αυτής της εργασίας (από tutorials μέχρι και σε δανεισμό Asset)**

## 1 Tutorials

1.1 https://docs.godotengine.org/en/stable/

1.2 https://www.youtube.com/watch?v=mAbG8Oi-SvQ&list=PL9FzW-m48fn2SlrW0KoLT4n5egNdX-W9a (Πολύ χρήσιμο tutorial για όποιον ενδιαφέρεται να ασχοληθεί με Godot)

## 2 Textures

2.1 **Tileset από SaintJacky** : https://saintjacky.itch.io/adventure-tileset

2.2 **Sprite παίκτη από Szadi Art** : https://szadiart.itch.io/rpg-main-character

2.**3 Sprite Τιτάνα από Kronovi**-: https://darkpixel-kronovi.itch.io/mecha-golem-free

## 3 Audio

3.1 **SFX από το** Legend of Zelda: A Link to the Past (SNES) https://www.101soundboards.com/boards/11034-zelda-a-link-to-the-past-sounds

3.2 **Overworld Κομματι** : Chrono Trigger – Secret of the Forest

3.3 **Κομμάτι Μάχης Τιτάνα**: Bravely Default 2 – Survival Instinct Reveals Fangs

# 6. ΠΡΟΒΛΗΜΑΤΑ

Ένα βασικό θέμα που είχαμε ήταν στο transition του παικτη από την μια σκηνή στην άλλη. Αποφασίσαμε, για λόγους αισθητικής. Όταν μπαίνει μέσα στο AREA2D που κάνει το transition, θα πρέπει το πρόγραμμα να κλέβει τον χειρισμό του παίκτη προσωρινα και να συνεχίζει ευθεία μέχρι να πάει στην επόμενη σκηνή. Ύστερα, θα πρέπει όσο μπαίνει στο επόμενο δωμάτιο, να έχει ακόμα κλεμμένο έλεγχο μέχρι να προχωρήσει ελαφρώς από την μεριά που βγήκε και να πάρει πίσω τον έλεγχο. Ωστόσο, επειδή αυτές οι δυο σκηνές είναι διαφορετικες, δεν μπορούσαμε μόνο με αυτά που είχαμε να κάνουμε την μεταβαση χωρις να χάνουμε πληροφορίες για το από που μπαινει και βγαινει ο παίκτης. Οπότε ορίσαμε ένα global Script με πληροφορίες για το transition του παίκτη που καλείται κάθε φορά που συμβαίνει ένα transition (**PlayerTransitionState.gd**)

Μέχρι στιγμής υπάρχει ένα σπάνιο φαινόμενο που συμβαίνει όταν σε ένα πολύ μικρο χρονικό παράθυρο ο παίκτης χτυπάει τον Τιτάνα ενώ τον χτυπάει και αυτός. Δεν έχει βρεθεί η λύση προς το παρών. Παρόλα αυτά, είναι πολύ σπάνιο και δεν τυχαίνει συχνά προκειμένου να βλάψει εντελώς την εμπειρία του παίκτη. Θα γίνει περαιτέρω απόπειρα εντοπισμού και διόρθωσης του στο μέλλον