Engraginy: Simulació de Sistemes de Transmissió Mecànica

Biel Alavedra Busquet

January 21, 2026

Resum– Engraginy és un videjoc 3D que busca simular mecaniques de transmisió mecanica, fent ús d'engranatges, eixos, i altes mecanismes amb l'objectiu de construïr cadenes de producció i fabriques. El nucli del projecte és el sistema de potencia i transmisió mecànica, basat en grafs que fa ús de calcul vectorial i senyals per actualitzar el sistema quan és requereix. Ademés d'un sistema de transport de materials basat en simulacions físiques. Tots aquests sistemes disenyats i pensats per oferir una experiencia totalment reactiva.
Paraules clau– Godot, Videojoc, Grafs, GDScript, Logística, Simulació 3D.

Abstract– Engraginy is a 3D video game that seeks to simulate mechanical transmission mechanics, utilizing gears, shafts, and other mechanisms with the goal of building production chains and factories. The core of the project is the power and mechanical transmission system, based on graphs that use vector calculus and signals to update the system as required. Additionally, it features a material transport system based on physical simulations. All these systems are designed and intended to offer a fully reactive experience.
Keywords– Godot, Videogame , Graphs, GDScript, Logistics, 3D Simulation.

1 Introducció - Context del treball

El projecte Engraginy neix de la voluntat d'explorar mecàniques de joc més físiques dins del gènere de l'automatització. Mentre que títols referents com *Factorio* i *Satisfactory* utilitzen l'electricitat com un recurs binari o simplificat, aquest treball proposa una xarxa de potència mecànica on cada component (engranatges, eixos, cintes) afecta el rendiment de les cadenes de producció segons les lleis de la cinemàtica bàsica.

L'objectiu principal és la creació d'un sistema modular i extensible en el motor Godot 4 que permeti al jugador construir xarxes de potència complexes. Per aconseguir aquest objectiu s'haurà de disenyar, implementar i testejar tot un videojoc que compleixi amb el seguent llistat de requisits:

- Joc en primera persona del gènere automatització
- Menu de construcció que mostri les opcions possibles
- Sistema de carrega i guardat de partida
- Sistema de construicció basat en graella
- Sistema de simulació de transmisió mecànica
- Sistema de transport d'objectes

• E-mail de contacte: biel.alavedra@gmail.com
• Menció: Computació
• Treball tutoritzat per: Enric Marti Godia
• Curs 2025/26

El repte tecnològic, a afrontar, resideix en la propagació dinàmica de la rotació en un entorn 3D, requerint l'ús de càlculs vectorials per determinar la viabilitat de les connexions, i les direccions de transmisió.

2 Motivació

La meva motivació per desenvolupar aquest projecte és principalment la d'aprendre com funciona el desenvolupament d'un videojoc. Els meus referents a l'hora de desenvolupar el projecte són videojocs com 'Factorio'[5], 'Satisfactory'[6], 'Dyson sphere program'[7], 'Shapez'[8], i 'Minecraft: Create Mod'[9] (una modificació del joc no oficial). D'aquests referents amb els quals tinc més experiència són DSP i Satisfactory, dos videojocs que m'han enganxat molt i m'han mostrat què entretingut i mentalment activador pot arribar a ser el gènere. El DSP és un joc molt més ambiciós, es comença en un petit satèl·lit dins d'un cúmul d'estrelles generat aleatòriament, i acaba quan el jugador construeix una esfera de Dyson i viatja entre sistemes solars. Tot això amb una perspectiva isomètrica que dona molt control i molta visió de tot allò que construeixes. En el Satisfactory l'objectiu és completar un ascensor espacial. Es comença en un planeta (el qual sempre és el mateix, el mapa és prefixat). Aquest joc és en primera persona, i totes les màquines que es construeixen són gegantines. Això impedeix una visió completa d'allò que construeixes. En quantitat de màquines les construccions són sempre més petites i molt més lentes de construir.

Dels dos jocs vull fer una cosa més similar a Satisfactory, ja que aquesta visió en primera persona dona un control

més granular de les construccións a petit nivell, i encaixa molt millor amb les mecaniques que vull incorporar de Minecraft: Create mod, en la majoria de jocs de l'estil una part molt important és la generació d'energia, no serveix de res fer fàbriques més grans si no es tenen els recursos per mantenir-les, tant DSP, Factorio i Satisfactory utilitzen energia elèctrica, sigui solar, eòlica, carbó o nuclear. Tots aquests generadors s'acaben connectant a la xarxa elèctrica per donar energia a totes les màquines. En canvi, el Minecraft: Create mod requereix que tot sigui alimentat per energia cinètica, fent ús d'eixos, engranatges grans i petits, cadenes, corretges de transmissió... Com el meu objectiu és fer un joc amb una ambientació prerevolució industrial, i no del tipus fantàstic, crec que aquesta és la millor forma de fer-ho. També això afegeix totes les mecàniques d'haver de connectar les màquines, no només a una velocitat concreta, doncs algunes seran direccionals, com les cintes per transportar material.

D'entre tots els gèneres de videojocs perquè he escollit automatització? Considero que aquest és un gènere que segueix molt la filosofia d'un programador, o en general la de tots els enginyers: dividir i vèncer. L'inici d'aquests jocs és senzill, hi ha unes poques màquines i cal fer processos simples. Per exemple, amb un extractor de recursos automàtic i cal processar els recursos en brut per tal d'obtenir el recurs processat (de mena de ferro lingots a de ferro). Però això ràpidament canvia, agafant d'exemple el joc DSP. Si vols fer una placa de circuits es necessita un forn de fosa que converteixi el ferro en brut a plaques de ferro, un altre forn de fosa que converteixi coure en brut a plaques de coure i un assemblador que agafi aquests dos materials i els converteixi en la placa de circuits. Això és només un dels primers passos, on cada vegada es disposa de més i més materials diferents, processos diferents i els objectes requerits són cada vegada més complexes. Per força no es poden afrontar tots aquests problemes simultàniament, cal dividir els problemes en mòduls que es puguin replicar cada vegada que calgui per a resoldre aquell problema.

Dins de la comunitat de fans d'aquest gènere de videojoc hi ha molts enginyers i gent que li agrada molt optimitzar processos i fer construccions el màxim d'eficients possibles. Per tot això no només considero el gènere com a entretingut, sinó que també en certa manera és educatiu, doncs força al jugador tant sí com no a organitzar-se, pensar formes eficients de construir coses, com connectar les entrades i sortides de totes les fabriques, calcular què tanta entrada es necessita per la sortida objectiva i construir d'acord amb això. Per completar un joc d'aquests cal ser organitzat, metòdic i pensar en solucions segons el problema que es tingui.

3 DESENVOLUPAMENT I ESTRUCTURA

El programari s'ha estructurat seguint els principis de la programació orientada a objectes (POO) mitjançant el sistema de nodes de Godot. La classe base `Building.gd` defineix els comportaments d'interacció, mentre que `PowerNode.gd` hereta aquesta base per incloure la lògica de ports i connexions. Engranatges, eixos, generadors i maquines hereten d'aquest `PowerNode.gd` per tal de definir més concretament les propietats especifiques de cada cas.

En la figura, Fig. 1, es pot observar un diagrama de moduls amb les clases més importants del projecte, el Jugador controla direcament la càmera, el moviment del model de jugador, i l'iterficie d'usuari, que conte els menus per seleccionar i colocar objectes. Els objectes col·locats per l'usuari, "Buildings" tots anirant acompanyats d'un context component, aquest permet l'interacció (si així o necesita) i la posterior eliminació de l'edifici si així o vol el jugador. Aquests "Buildings" podran ser "PowerNodes", elements de la xarxa de transmisió mecànica que detallare més endevant. Els "PowerNodes" seràn gestionats per el "PowerGridManager" i tots contindran una serie de "PowerNodePort", per gestionar les connexions entre ells.

3.1 Nodes i escenes

Ja que s'ha fet ús de Godot el projecte havia d'estar estructurat internament tal i com ho requereix el motor, és imperatiu fer us del sistema de nodes i escenes per crear tots els objectes i parts del videojoc. Això ens deixa mab dues grans parts, l'estructura del programari, basada en POO i l'estructura interna de tots els objectes i escenes que són un conjunt de nodes predeterminats i nodes amb una implementació propia.

3.2 Nodes i escenes

Abans de parlar de la meva implementació tècnica és important explicar com funciona el sistema de nodes i escenes de Godot. Cada element del joc és un node amb unes propietats i funcions diferents. Hi ha tres famílies de nodes principals, nodes 3D, nodes 2D i nodes de GUI. Dins de la família de nodes 3D tenim exemples com:

- Camera3D
- StaticBody3D
- MeshInstance3D
- CollisionShape3D
- RayCast3D
- etc.

Cada un d'aquests nodes és una classe diferent amb les seves propietats, però totes hereten de la classe Node3D. Els nodes de GUI ens permeten crear interfícies d'usuari on utilitzarem l'estructura jeràrquica per encapsular cada element d'interfície com ara:

- Container
- Label
- TextureRect
- Button
- Panel
- ItemList
- etc.

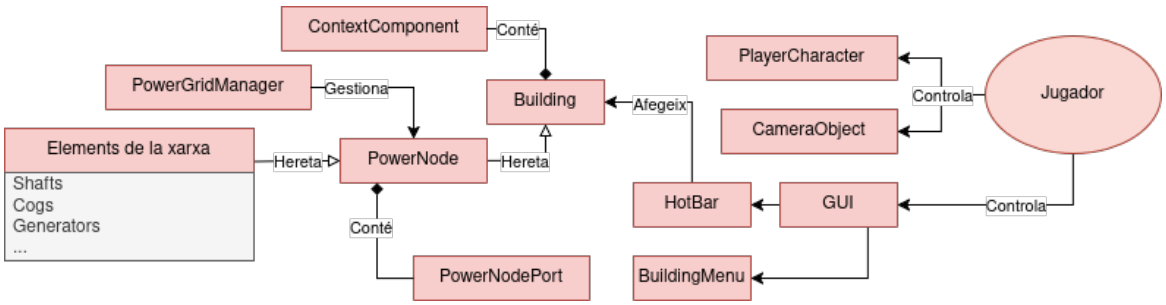

Fig. 1: Diagrama de moduls

Les escenes seran definides per l'usuari i són una col·lecció de nodes de qualsevol tipus, una escena pot ser el jugador, la interfície d'usuari o un nivell sencer d'un joc. Depèn de cada desenvolupador decidir com organitzar. Aquestes escenes es poden afegir a altres escenes per poder així evitar treballar amb grans problemes a la vegada, i tenir-ho tot en mòduls més petits.

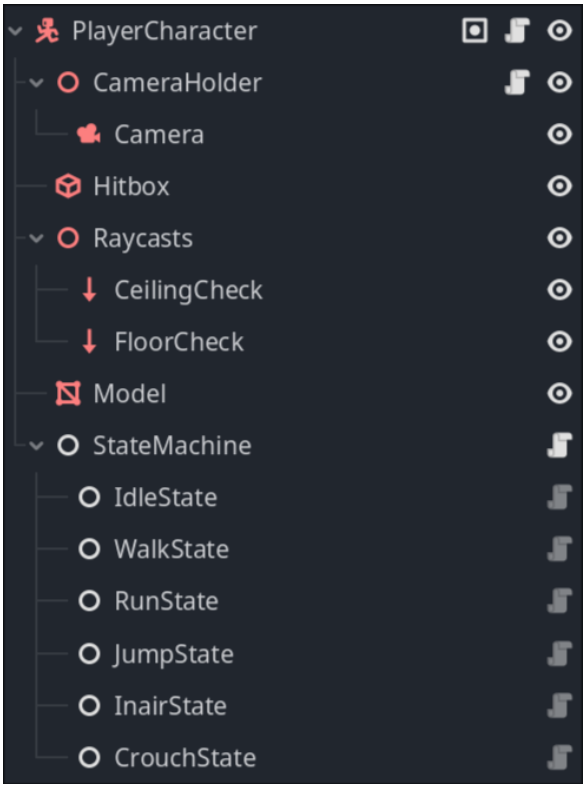

Fig. 2: Escena del jugador

En l'escena que conté el jugador, Fig. 2, tenim com a node pare un "PlayerCharacter" (node propi, hereta de la clase base "CharacterBody3D"), amb 5 fills, el punt d'anclatge de la camera ("CameraHolder"), la caixa de col·lisions del personatge ("Hitbox"), els "RayCast3D" ("Raycasts"), el "Model" i la màquina d'estats encarregada del control del personatge. Aquesta màquina d'estats està iniciada amb nodes buits, aquests nodes són la classe base per a tots els objectes de Godot, i no tenen propietats especials. Podem veure que en el lateral tenim una icona amb un pergamí, això ens indica que hi ha un script vinculat al node, si està en gris vol dir que aquest script és heretat, compartit amb altres nodes de la mateixa classe. Si és blanc ens indica que el node té un script únic.

3.3 Lògica de Transmissió Mecànica

El sistema de transmisió mecànica és una adaptació del sistema existent a Minecraft: Create. Per tant, he volgut simular el seu funcionament, a continuació faré un llistat de les especificacions del sistema, i com s'hauria de comportar sota cada cas específic.

- Tota connexió d'un eix ha de mantenir direcció i velocitat.
- Tota connexió d'un engranatge ha d'invertir la direcció.
- Connectar un engranatge petit a un gran a través de les dents de l'engranatge ha de duplicar velocitat, a l'invers la velocitat es divideix.
- La xarxa no pot superar el límit d'energia subministrada per tot el conjunt de generadors connectats.
- En cas de superar el límit tot el sistema s'ha d'aturar a l'instant.
- En cas de tornar a estar per sota del límit, el sistema ha d'entrar en funcionament a l'instant.
- En cas que un node tingui incoherències en els seus ports connectats s'ha d'eliminar automàticament.

¡FIG ./img/ConnectionsExample.png¿ Figura. Mostra de possibles connexions.

La classe principal d'aquest sistema és "PowerNodes". D'aquesta classe hereten tots els elements de la xarxa que afegirem. Cada "PowerNode" tindrà mínim un "PowerNodePort", aquests ports són cubs ubicats allà on es vol que el node tingui un punt de connexió. Aquest port té quatre propietats molt importants: la direcció de gir respecte al node, el modificador de velocitat, el tipus de port que és, i a quins altres ports es pot connectar.

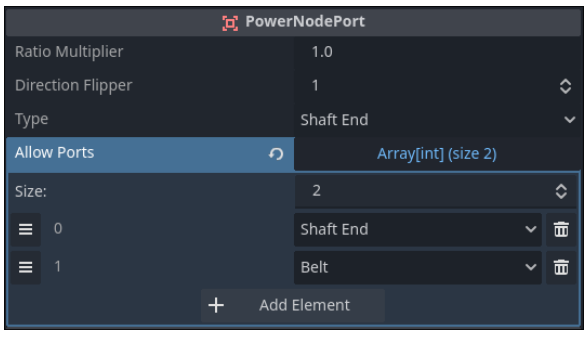

Fig. 3: Exemple de port d'un eix

En la figura 3, podem veure com estan configurats els ports d'un eix bàsic. Podem veure respectivament les propietats: modificador de velocitat ("Ratio Multiplier"), direcció de gir ("Direction Flipper"), tipus de port ("Type"), i connexions permeses ("Allow Ports"). Els tipus de port implementats són: "SHAFT_END", "COG_SMALL", "COG_BIG" i "BELT". En cas d'implementar un element que ens permeti invertir gir hauríem de tenir dos ports, un com el de la figura 3, i l'altre amb el "Direction Flipper" a -1, si volguéssim implementar un element que ens permetis duplicar la velocitat de gir hauríem de modificar el "Ratio Multiplier" a 2.

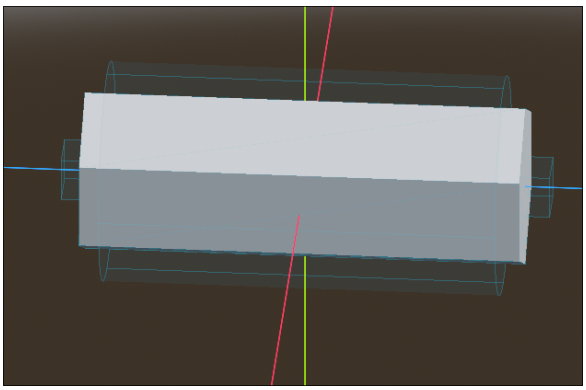

Fig. 4: Exemple de l'escena d'un eix

En la figura 4 veiem l'exemple de l'eix, amb els dos ports col·locats un a cada extrem de l'objecte.

Aquests ports hereten de la classe base de Godot "Area3D". Això és perquè aquesta classe base té ja té implementat un mètode que en cas d'un altre "Area3D" entri en contacte podem connectar una funció que com argument ens doni l'àrea que ha entrat.

Quan detectem aquesta connexió comprovem que l'altra àrea sigui un "PowerNodePort" i que la connexió sigui vàlida. En cas de connexió vàlida llancem el senyal "network_changed", que indica al "PowerGridManager" que hi ha hagut un canvi en la xarxa a la qual pertany el node, i per tant s'ha de recalcular la xarxa.

Aquest procés de càlcul comença amb un algoritme "Breadth-first search" (BFS), es podria utilitzar també un "Depth-first search" (DFS) en aquest cas, per tal de trobar tots els nodes connectats. Quan volem trobar tots els nodes connectats no hi ha diferència entre utilitzar un BFS o un DFS, però més endavant s'ha d'utilitzar forçosament un BFS, per tant, ja tenim la funció implementada. Una vegada tenim tots aquests nodes busquem quins són els generadors que hi ha a la xarxa, i és a partir d'aquests generadors que comencem a propagar tots els canvis. Per propagar els canvis apliquem un altre BFS. En aquest cas ha de ser aquest algoritme, ja que interessa que els canvis es propaguin capa a capa, que es calculin abans els nodes més pròxims per així detectar els possibles conflictes en els extrems. Per cada node definim la seva velocitat com la calculada en el moment d'afegir-lo a la llista de "per visitar", i iterem per les seves connexions per tal de calcular la velocitat que haurien de tenir per no generar conflictes. En cas de no tenir la connexió una velocitat ja assignada li assignem la calculada. Si ja tenia una velocitat assignada ens assegurem que no hi hagi conflictes. En cas de conflicte eliminem la peça i tornem a començar el procés de càlcul de la xarxa. Si no hi ha conflicte afegim la connexió al llistat de "per visitar". Finalment, quan hem acabat amb el càlcul de les velocitats recorrem tota la xarxa per comprovar que la potència és prou per mantenir la xarxa activa. Tots els generadors sumen les seves potències i la resta d'elements resten.

La part més complexa d'aquest procés és el càlcul de velocitats. Els nodes poden estar situats en qualsevol punt del món i en qualsevol direcció. Per exemple un eix sense girar connectat a un de girat 180 graus semblen el mateix, però si es fa una assignació directa de velocitat, el segon eix girarà en sentit contrari. Per solucionar això fem una sèrie de càlculs amb els vectors de direcció de cada element per comprovar com s'ha de comportar aquella connexió. Per fer aquest càlcul seguirem una sèrie de passos, aquests pasos no s'executen sempre tots, depenent dels resultats d'operacions anteriors poden cambiar els pasos que s'executen.

1. $W_{input} = W_{conn} \times R_{conn} \times D_{conn}$
2. $Alg = sgn(\vec{A} \cdot \vec{B})$
3. $Alg = sgn(\vec{S}_{dir} \cdot (1,1,1))$
4. $\vec{V} = (\vec{P}_{conn} + \vec{C}_{conn}) - (\vec{P}_{local} + \vec{C}_{local})$
5. $Alg = sgn((\vec{A} \times \vec{V}) \cdot (-\vec{B} \times \vec{V}))$
6. $W_{res} = \frac{W_{input} \times D_{local} \times Alg}{R_{local}}$

En la primera fórmula obtenim la velocitat d'entrada aparent, sense tenir en compte l'alineació dels dos nodes. Els paràmetres són els següents: Wconn és la velocitat de rotació del node connectat, ≪Rconn≫ és el multiplicador de velocitat del port del node connectat i ≪Dconn≫ és la direcció del port del node connectat.

En la segona calculem l'alineació de l'eix de rotació propi, A, i l'eix de rotació del port connectat, B. L'alineació es calcula utilitzant el producte escalar entre els dos eixos de rotació. A partir d'aquest punt poden passar dues coses: que el valor Alg sigui 0 o 1/-1.

En cas que Alg sigui 1/-1 significarà que els dos ports són paral·lels. Si els dos ports són del tipus "SHAFT_END" la velocitat del port serà la Winput multiplicada pel signe de Alg. En cas que els ports siguin del tipus "COG" la velocitat serà Winput multiplicada pel negatiu de signe de Alg, per tal d'invertir la direcció de gir.

En el cas que Alg sigui 0 voldrà dir que els dos ports són perpendiculars, aquest es dona quan connectem dos engranatges grans en perpendicular, o quan un eix està connectat a una cinta mecànica. En el segon cas per obtenir la direcció correcta apliquem la tercera fórmula, on Sdir és la direcció de gir de l'eix. En el primer cas, dos engranatges grans en perpendicular, apliquem les fórmules 4 i 5. Amb la fórmula 4 obtenim un vector que va des del centre del node connectat al centre del node actual, Pconn/Plocal són les posicions de l'element en l'escena i Cconn/Clocal és un desplaçament per als casos on el centre de gir del node no correspon en la posició en escena. Finalment calculem l'alineació.

Per acabar, aquesta velocitat Winput, que representa la velocitat del port la convertim a la velocitat interna que ha de tenir el node, utilitzant la sisena fórmula desfem el càlcul fet en la primera, però utilitzant els valors del nostre port i

multipliquem per l'alineació final, Alg. Aquest és el valor que retornem perquè el "PowerGridManager" comprovi si hi ha conflicte.

3.4 Logística i Transport d'Ítems

3.5 Interficies d'usuari

3.6 Construcció i interacció amb objectes

3.7 Guardar i carregar escenaris

4 Conclusions

Agraïments

References

[1] Godot

Apèndix