Engraginy: Simulació de Sistemes de Transmissió Mecànica

Biel Alavedra Busquet

February 1, 2026

Resum– Engraginy és un videojoc 3D que busca simular sistemes de transmissió mecànica, fent ús d'engranatges, eixos, i altres mecanismes amb l'objectiu de construir cadenes de producció i fàbriques. El nucli del projecte és el sistema de potència i transmissió mecànica, basat en grafs que fa ús de càlcul vectorial i senyals per actualitzar el sistema quan es requereix. A més d'un sistema de transport de materials basat en simulacions físiques. Tots aquests sistemes dissenyats i pensats per oferir una experiència totalment reactiva. **Paraules clau–** Godot, Videojoc, Grafs, GDScript, Logística, Simulació 3D.

Abstract– Engraginy is a 3D video game that seeks to simulate mechanical transmission systems, 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 calculation 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* [1] i *Satisfactory* [2] 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* [3] que permeti al jugador construir xarxes de potència complexes. Per aconseguir aquest objectiu es dissenya, implementa i testeja un videojoc que compleixi amb el següent llistat de requisits:

- Joc en primera persona del gènere automatització
- Menú de construcció que mostri tots els elements que es poden construir: engranatges, eixos, generadors, etc.
- Sistema de càrrega i guardat de partida
- Sistema de construcció basat en graella
- Sistema de simulació de transmissió mecànica
- Sistema de transport d'objectes

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 transmissió.

La motivació principal per al desenvolupament d'aquest projecte és aprendre com funciona el desenvolupament d'un videojoc. Estudiar el cicle de vida i les diferents fases necessàries per a crear un videojoc. Els referents estudiats a l'hora de desenvolupar el projecte són videojocs com *Factorio*, *Satisfactory*, *Dyson sphere program* (DSP) [4], *Shapez* [5], i *Minecraft: Create Mod* [6] (una modificació del joc no oficial).

D'entre els referents analitzats, cal destacar *DSP* i *Satisfactory*, dos videojocs que sobresurten per la seva capacitat d'immersió i estimulació lògica. Aquests títols exemplifiquen com d'entretingut i mentalment activador pot ser el gènere. El *DSP* és un joc molt més ambiciós, s'inicia en un petit satèl·lit dins d'un cúmul d'estrelles generat aleatòriament, i acaba quan el jugador és capaç de navegar entre sistemes solars i construeix una esfera de Dyson [7]. Tot això amb una perspectiva isomètrica que facilita la supervisió i atorga un gran control de tot allò que es construeix. En la fig. 1 es mostra un exemple de com es veu i es construeix en el joc.

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 es construeix. En quantitat de màquines les construccions són

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

Fig. 1: Exemple de com el jugador veu el joc (vista isomètrica)

sempre més petites i molt més lentes de construir. S'observa en la fig. 2 la gran diferència entre les dues captures, mentre en una es presenta una vista sencera del procés, en la segona hi ha una visió més limitada, però molt més immersiva.

Fig. 2: Exemple de com el jugador veu el joc (primera persona)

L'objectiu és desenvolupar un sistema amb mecàniques més similars a *Satisfactory*, ja que aquesta visió en primera persona dona un control més granular de les construccions a petit nivell, i encaixa molt millor amb les mecàniques a incorporar de *Minecraft: Create mod*. En la majoria dels jocs de l'estil una part molt important és la generació d'energia. És molt ineficient 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, etc. Com l'objectiu és fer un joc amb una ambientació prerevolució industrial, i no del tipus fantàstic, aquesta sembla ser la millor opció per aconseguir-ho. També això afegeix totes les mecàniques d'haver de connectar les màquines, no només a una velocitat concreta, ja que algunes seran direccionals, com les cintes per transportar material.

D'entre tots els gèneres de videojocs perquè s'ha escollit l'automatització? 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, amb unes poques màquines 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 a lingots de ferro). Però això ràpidament canvia, agafant d'exemple el joc *DSP*. Si es vol fer una placa de circuits es necessita un forn de fosa que converteixi el mena de ferro 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 a què agrada molt optimitzar processos i fer construccions el màxim d'eficients possibles. Per tot això el gènere no és només entretingut, sinó que també en certa manera és educatiu, doncs força al jugador a organitzar-se, pensar formes eficients de construir coses, com connectar les entrades i sortides de totes les fàbriques, calcular que tanta entrada es necessita per la sortida objectiva i construir d'acord amb això. Per completar el joc cal ser organitzat, metòdic i pensar en solucions segons el problema que es tingui.

2 Conceptes de Godot: Nodes, Escenes i Senyals

Prèviament a la descripció tècnica del projecte és necessari explicar el funcionament de tres conceptes clau: Nodes, Escenes i Senyals. Aquests termes apareixeran recurrentment al llarg de l'article. Pel fet d'utilitzar Godot l'estructura interna del projecte ha de seguir els requisits del motor. Això implica fer ús del sistema de nodes i escenes per crear tots els objectes, i aprofitar els senyals per simplificar connexions entre objectes.

2.1 Nodes

Cada element del joc és un node amb unes propietats i funcions diferents. Hi ha tres famílies principals: nodes 3D, nodes 2D i nodes de "GUI" (Interfície d'Usuari).

Dins de la família de nodes 3D, que són els més rellevants per la simulació, hi ha exemples com:

- Camera3D
- StaticBody3D
- MeshInstance3D (per als models visuals)
- CollisionShape3D (per a les físiques)
- RayCast3D (per a detectar objectes)

Tots aquests nodes hereten de la classe base `Node3D`. D'aquesta classe obtenen les propietats essencials de qualsevol objecte a l'espai: la posició, la rotació i l'escala.

Els nodes de "GUI" (`Control`) permeten la creació d'interfícies d'usuari on s'utilitza l'estructura jeràrquica per encapsular cada element d'interfície com ara:

- Container

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

Els nodes 2D no són rellevants per al projecte, però molts d'ells tenen equivalents en els nodes 3D.

2.2 Escenes

Les escenes seran definides per l'usuari i són una col·lecció de nodes de qualsevol classe. Una escena pot ser el jugador, la interfície d'usuari o un nivell sencer d'un joc. L'organització i contingut d'aquestes escenes respon a les decisions de disseny i arquitectura del programari. Aquestes escenes es poden instanciar dins d'altres escenes, la qual cosa permet dividir problemes grans en mòduls més petits i manejables.

Fig. 3: Escena del jugador amb la jerarquia de nodes

En l'escena que conté el jugador, fig. 3, hi ha com a node pare un "PlayerCharacter" (node propi, hereta de la classe base "CharacterBody3D"), amb 5 fills, el punt d'ancoratge de la càmera ("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. Al lateral s'observa una icona amb un pergamí, que 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 indica que el node té un script únic.

2.3 Senyals

Els senyals permeten connectar diferents nodes i escenes sense necessitat de conèixer exactament l'estructura de l'escena ni haver de fer referència al node directament. Aquests senyals són missatges que els nodes poden emetre; els altres nodes es poden connectar a aquest senyal i executar una funció quan reben el missatge. Aquests missatges es poden emetre directament per codi, poden respondre a accions del jugador o a canvis en l'estructura de l'escena. Aquest sistema redueix molt les dependències i fa el codi més modular, ja que s'eviten referències directes entre escenes. Per explicar com s'utilitza aquest sistema es pren com a exemple el mecanisme que s'ha desenvolupat per interactuar amb objectes.

En aquest fragment de codi, fig. 4, es defineixen les funcions utilitzades per emetre els tres senyals d'interacció amb l'entorn: `unfocused`, `focused` i `interacted`. La funció `interact_cast()` s'executa en cada fotograma del joc per comprovar si s'està interactuant amb algun objecte. El primer pas és obtenir l'objecte que el jugador està mirant directament. Per fer-ho, s'emet un `RayCast3D` des de l'origen de la càmera fins a una certa distància i es retorna el primer objecte amb el qual col·lideix mitjançant la funció `make_cast_query()`.

Si l'objecte retornat és el mateix que en el fotograma anterior, es fa res. En cas contrari, es verifica si en l'objecte anterior tenia definit el senyal `unfocused`. Si no estigués definit, vol dir que amb aquell objecte no es podia interactuar; si hi és, s'emet el senyal. Tot seguit, si el nou objecte que el jugador està mirant té el senyal `focused`, s'emet per indicar que ha rebut el focus.

D'altra banda, la funció `interact()` només s'executa quan el jugador prem el botó d'interactuar. El procediment és similar: es comprova que l'objecte tingui el senyal `interacted` i, en cas afirmatiu, s'emet.

Per gestionar la resposta a aquests senyals, els objectes interactius disposen d'un node fill anomenat `InteractionComponent`. Aquest conté l'script encarregat de processar els senyals i cridar les funcions adients de l'objecte pare. Aquest component ha de ser fill d'un node `StaticBody3D`, ja que la funció `make_cast_query()` requereix cossos físics per detectar la interacció. Finalment, en l'objecte `InteractionComponent` es busca el node pare i es connecten aquests senyals amb les funcions `in_range`, `not_in_range` i `on_interact` definides en la lògica de l'objecte.

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` defineix els comportaments d'interacció, mentre que `PowerNode` hereta aquesta base per incloure la lògica

```
func interact_cast() -> void:
    var current_cast_result : CollisionObject3D = make_cast_query()
    if current_cast_result != interact_cast_result:
        if interact_cast_result and interact_cast_result.has_user_signal("unfocused"):
            interact_cast_result.emit_signal("unfocused")
        interact_cast_result = current_cast_result
        if interact_cast_result and interact_cast_result.has_user_signal("focused"):
            interact_cast_result.emit_signal("focused")
    pass

func interact() -> void:
    if interact_cast_result and interact_cast_result.has_user_signal("interacted"):
        interact_cast_result.emit_signal("interacted")

    pass
```

Fig. 4: Fragment de codi de l'objecte PlayerCharacter que conté les funcions per emetre senyals

de ports i connexions. Engranatges, eixos, generadors i màquines hereten d'aquest `PowerNode` per tal de definir més concretament les propietats específiques de cada cas.

En la figura, fig. 5, s'observa el diagrama de mòduls del projecte, aquest no és complet, però serveix de resum per a veure les classes més importants i les seves relacions. A la dreta s'observa l'oval que representa al jugador, això no és una classe, però indica que és allò que el jugador controlarà directament.

El jugador controla directament tres components: el `PlayerCharacter`, és l'avatar del jugador, controla el moviment i les col·lisions, el `CameraObject`, s'encarrega del control de càmera i de com el jugador veurà el món, i finalment totes les GUI, depenent del context i amb diferents entrades el jugador podrà controlar aquesta part, per exemple, amb la tecla "TAB" el jugador podrà obrir i tancar el `BuildingMenu`.

Aquest `BuildingMenu` conté un llistat amb tots els possibles objectes que es poden construir, separat en quatre pestanyes (Producció, energia, fonaments i decoració), per a una millor organització. Des d'aquest menú el jugador podrà fer servir el ratolí per arrossegar objectes a la `HotBar`, aquesta barra d'accés directe conté la lògica per a la construcció d'objectes, prement una de les tecles 1-9 es seleccionarà una de les seves caselles i si aquesta conté un objecte el joc mostrarà una previsualització de `Building`, aquesta serà verda si l'objecte està en una posició vàlida, i vermella si està col·lidint amb un altre objecte i no pot ser col·locada.

La classe `Building` és la classe base de qualsevol objecte que el jugador podrà col·locar, conté tota la lògica per gestionar les col·lisions, l'entrada amb escena, la inicialització de variables necessàries i els mètodes necessaris per interactuar i permetre al jugador eliminar l'objecte.

Per gestionar aquesta última part s'utilitza la classe `InteractionComponent`, aquest és un node que s'afegeix a l'escena de tot `Building`, que fent ús de senyals informa l'objecte si el jugador l'està mirant (el cursor del centre de la pantalla està just a sobre l'objecte i el jugador no està massa lluny de l'objecte), l'ha deixat de mirar, o el jugador ha executat alguna entrada especifica com pressionar la tecla "R" per eliminar l'objecte.

La classe `PowerNode` hereta de la classe `Building`, aquesta és la classe que gestiona totes les connexions lògiques per a tots els elements del sistema de potència. Aquests nodes formaran un graf no dirigit on cada node coneixerà únicament els seus veïns, la classe `PowerGridManager` gestionarà la resolució de conflictes dins la xarxa, els canvis de velocitat o l'aturada completa d'aquella xarxa si se supera el límit de potència d'aquella xarxa. Tots aquests canvis seran executats només quan hi hagi un canvi a la xarxa, i només es recalcularà aquella xarxa en concret, així s'evita càrrega computacional innecessària en comprovar canvis on no n'hi ha. Aquests elements de la xarxa seran tot un conjunt de classes més petites que hereten de `PowerNode`, com: `Shaft`, `Cog`, `Machine`, `Generators`, etc. Finalment per gestionar totes aquestes connexions entre `PowerNodes` s'utilitzen els `PowerNodePort`, que s'encarreguen d'avisar al node quan un altre node ha entrat en el rang de connexió i quan un node marxa d'aquest rang.

3.1 Lògica de Transmissió Mecànica

El sistema de transmissió mecànica és una adaptació del sistema existent a Minecraft: Create. Per tant, s'ha optat per simular-ne el comportament. A continuació es farà 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 d'augmentar la velocitat, a l'invers la velocitat es redueix.
- 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.

Fig. 5: Diagrama de mòduls

- 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.

La classe principal d'aquest sistema és `PowerNode`. D'aquesta classe hereten tots els elements de la xarxa que s'afegiran. 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. 7: Exemple de l'escena d'un eix

3.1.1 Construcció de les escenes pels `PowerNode`

Fig. 6: Exemple de port d'un eix

En la figura 6, es presenta com estan configurats els ports d'un eix bàsic. Respectivament apareixen 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 permeti invertir gir s'han d'afegir dos ports, un com el de la figura 6, i l'altre amb el "Direction Flipper" a -1, si es volgués implementar un element que permetés duplicar la velocitat de gir s'hauria de modificar el "Ratio Multiplier" a 2.

En la figura 7 es veu l'exemple de l'eix, amb els dos ports col·locats un a cada extrem de l'objecte, aquests ports són els petits cubs blaus translúcids situats en els extrems. Aquests ports hereten de la classe base de Godot `Area3D`. Ja que aquesta classe base ja té implementats una sèrie de mètodes que permeten la detecció d'altres objectes en entrar o sortir dins la seva àrea, i faciliten cridar altres funcions en el moment que es produeixi un canvi. Així sempre en detectar un traspàs es cridarà la funció "_on_area_entered" i quan es detecti que un objecte surt es cridarà la funció "_on_area_exited".

3.1.2 `PowerGridManager` gestor central de transmissió i càlcul de potència

El `PowerGridManager` entra en funcionament quan es detecta un senyal que informa d'una modificació en la xarxa, aquests senyals són enviats quan el `PowerNode` detecta una connexió o desconnexió vàlida en un dels seus `PowerNodePort`. Aquesta connexió vàlida llança el senyal "network_changed", el `PowerGridManager` recull aquest senyal i comença el procés de recàlcul de la xarxa.

Aquest procés de càlcul comença amb un algoritme "Breadth-first search" ("BFS") [8], es podria utilitzar també un "Depth-first search" ("DFS") [9], per tal de trobar tots els nodes connectats. Per a trobar tots els nodes connectats no hi ha diferència entre l'algorisme "BFS" o el "DFS", però més endavant cal usar forçosament un "BFS", per tant, ja es té la funció implementada. Una vegada s'obtenen tots aquests nodes es comença la busqueda per trobar quins són els generadors de la xarxa, i és a partir d'aquests generadors que es comencen a propagar tots els canvis. Per propagar els canvis s'aplica un altre "BFS". En aquest cas ha de ser aquest algoritme, ja que és preferible que els canvis es propaguin nivell a nivell, que es calculin abans els nodes més pròxims per així detectar els possibles conflictes en els extrems. Per cada node es defineix la seva velocitat com la calculada en el moment d'afegir-lo a la llista de "per visitar", i s'itera 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 se li assigna la calculada. Si ja tenia una velocitat assignada es comprova que aquestes siguin iguals. En cas de conflicte s'elimina la peça i es torna a començar el procés de càlcul de la xarxa. Si no hi ha conflicte s'afegeix la connexió al llistat de "per visitar". Finalment, quan s'ha acabat amb el càlcul de les velocitats es recorre tota la xarxa per comprovar que la potència és suficient per mantenir la xarxa activa. Tots els generadors sumen les seves potències i els altres nodes resten.

3.1.3 Càlcul de velocitats

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 presenten una aparença idèntica, però si es fa una assignació directa de velocitat, el segon eix girarà en sentit contrari. Per solucionar això es fan una sèrie de càlculs amb els vectors de direcció de cada element per comprovar com s'ha de comportar aquesta connexió. Aquest comportament serà representat per la variable Alg, només tindrà dos possibles valors finals, 1 o -1. Per fer aquest càlcul se seguirà una sèrie de passos, aquests passos no s'executen sempre tots, depenent dels resultats d'operacions anteriors poden canviar els passos que s'executen. El funcionament de tot aquest procés es pot seguir visualment en la fig. 8, aquesta figura és el diagrama de flux del procés de càlcul que a continuació s'explicarà.

Fig. 8: Diagrama de flux

$$W_{input} = W_{conn} \times R_{conn} \times D_{conn} \quad (1)$$

$$Alg = sgn(\vec{A} \cdot \vec{B}) \quad (2)$$

$$Alg = sgn(\vec{S}_{dir} \cdot (1,1,1)) \quad (3)$$

$$\vec{V} = (\vec{P}_{conn} + \vec{C}_{conn}) - (\vec{P}_{local} + \vec{C}_{local}) \quad (4)$$

$$Alg = \text{sgn}((\vec{A} \times \vec{V}) \cdot (-\vec{B} \times \vec{V})) \quad (5)$$

$$W_{res} = \frac{W_{input} \times D_{local} \times Alg}{R_{local}} \quad (6)$$

En la fórmula (1) s'aconsegueix la velocitat d'entrada aparent, W_{input}, sense tenir en compte l'alineació dels dos nodes. Els paràmetres són els següents: W_{conn} és la velocitat de rotació del node connectat, R_{conn} és el multiplicador de velocitat del port del node connectat i D_{conn} és la direcció del port del node connectat.

En la fórmula (2) es calcula l'alineació de l'eix de rotació propi, $\vec{A}$, i l'eix de rotació del port connectat, $\vec{B}$. 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 W_{input} multiplicada pel signe de Alg. En cas que els ports siguin del tipus "COG" la velocitat serà W_{input} multiplicada pel negatiu de signe de Alg, per tal d'invertir la direcció de gir (dos engranatges que es toquin sempre giraran en sentits oposats).

En el cas que Alg sigui 0 voldrà dir que els dos ports són perpendiculars, que es dona quan es connecten dos engranatges grans en perpendicular, o quan un eix està connectat a una cinta mecànica. En el segon cas (Alg $==0$) per obtenir la direcció correcta s'utilitza la fórmula (3), on S_{dir} és la direcció de gir de l'eix. En el primer cas (abs(Alg) $==1$), dos engranatges grans en perpendicular, s'apliquen les fórmules (4) i (5). Amb la fórmula (4) s'obté un vector que va des del centre del node connectat al centre del node actual, P_{conn}/P_{local} són les posicions de l'element en l'escena i C_{conn}/C_{local} és un desplaçament per als casos on el centre de gir del node no correspon en la posició en escena. Finalment es calcula l'alineació.

Per acabar, aquesta velocitat W_{input}, que representa la velocitat del port es converteix a la velocitat interna que ha de tenir el node, utilitzant la fórmula (6) s'inverteix el càlcul fet en la primera, però fent servir els valors del port local i es multiplica per l'alineació final, Alg. Aquest és el valor que es retornarà perquè el `PowerGridManager` comprovi si hi ha conflicte.

3.2 Logística i Transport d'Ítems

El sistema de logística és l'encarregat de moure els materials entre les diferents màquines de processament. A diferència de jocs on els inventaris són purament numèrics, en aquest projecte s'ha optat per una representació física de cada ítem en el món.

La peça fonamental d'aquest sistema és la cinta mecànica (`Belt`). A nivell d'arquitectura, la classe `Belt` hereta de `PowerNode`. Això implica que les cintes no funcionen de manera autònoma, sinó que requereixen estar connectades a

la xarxa d'energia mecànica per moure's. La seva velocitat de transport és directament proporcional a la velocitat de rotació de la xarxa; si la xarxa s'atura o s'inverteix el gir, els ítems sobre la cinta reaccionen en conseqüència.

3.2.1 Simulació de moviment i Path3D

Per gestionar el moviment dels ítems al llarg de la cinta, s'ha utilitzat el node `Path3D` de Godot. Quan es col·loca una cinta entre dos punts (siguin eixos o ports de màquines), es genera una corba de Bezier rectilínia que defineix la trajectòria, en la fig. 9, utilitzant la funció de depuració de Godot per veure els `Path3D`, s'observa el camí en verd que va de l'inici al final de la cinta.

Fig. 9: Exemple del camí generat per una cinta

Cada ítem es tracta com un objecte `VisualMaterial` Aquesta classe hereta de la classe base `PathFollow3D`. Aquests nodes han de ser fills d'un `Path3D`, i tenen una propietat especial que indica quin és el seu progrés al llarg del `Path3D`. En cada cicle de processament (`_process`), s'actualitza la propietat `progress` de cada material segons la velocitat actual de la cinta i el temps de delta.

Fig. 10: Cinta mecànica amb objectes en moviment (mig) i aturats(esquerra)

En la fig. 10, es mostra una cinta que actualment es desplaça cap a l'esquerra de la imatge. S'observen tres objectes aturats perquè han arribat al final de la cinta, i un quart objecte que situat al mig que s'està desplaçant cap al tercer. Per aquesta imatge s'ha activat la visualització de caixes de col·lisió, per veure quins són els límits de l'objecte. S'ha implementat un sistema de "contrapressió" (*backpressure*). Cada ítem disposa d'una `Area3D` que detecta si col·lideix amb un altre ítem situat davant seu. Si un ítem arriba al final de la cinta i no pot saltar al següent node (perquè està ple o no hi ha connexió), s'atura. Aquesta aturada es propaga cap enrere: quan l'ítem posterior detecta la col·lisió amb l'ítem aturat, també deté el seu avanç. Això permet crear cues de materials de manera orgànica sense necessitat d'un gestor centralitzat.

3.2.2 Connexions i transferència

La continuïtat del transport es gestiona mitjançant un sistema de ports similar al de la transmissió mecànica, però especialitzat. La classe `Belt` defineix dos ports de detecció: `FrontPort` i `BackPort`.

Quan dues cintes es col·loquen de manera adjacent, o una cinta es connecta a un `MachinePort` (el punt d'entrada/sortida de les màquines), es crea un enllaç lògic anomenat `BeltConnection`.

La transferència d'ítems es produeix quan un material arriba a l'extrem del `Path3D`:

1. El sistema identifica la connexió activa (`ft_conn` o `bk_conn`) segons la direcció del moviment.
2. S'invoca el mètode `try_add_item` del node receptor (una altra cinta o una màquina).
3. Si el receptor accepta l'ítem (té espai lliure), es duplica l'objecte visual al nou contenidor i s'elimina de la cinta original.

Aquest disseny modular permet que les màquines, definides a la classe `Machine`, s'abstraguin del moviment. Aquestes màquines exposen uns ports d'entrada i sortida, i és la lògica de la cinta l'encarregada d'injectar o extreure els materials quan sigui físicament possible.

Fig. 11: Dues cintes connectades entre elles a dues màquines diferents, un extractor de recursos (esquerra) i una construcció que s'utilitza com a destructor (dreta).

Com es pot observar en la figura 11, aquest protocol garanteix que no es perdin ítems durant la transició i que es respectin les capacitats físiques de cada segment de la xarxa logística. En la fig. 11, es mostra una cadena sencera d'esdeveniments: els objectes surten de l'extractor de recursos (esquerra), quan arriben al final de la primera cinta són t ransferits a la segona cinta seguint el protocol esmentat anteriorment, i quan arriben al final de la segona entren al destructor (dreta) per ser eliminats i permetre un flux constant d'objectes.

3.3 Interfícies d'usuari

L'arquitectura de la interfície d'usuari (UI) s'ha dissenyat per ser modular i reactiva, separant clarament la lògica de joc

de la presentació visual. Tota la gestió de finestres i menús es centralitza en un únic gestor, el `MenuManager`, que actua com a controlador principal de l'estat de la interfície.

3.3.1 Gestor de Menús i Estats

La classe `MenuManager` administra la visibilitat i el focus de les diferents pantalles del joc. Aquest sistema permet alternar entre estats de joc actiu (on el ratolí està capturat per la càmera) i estats de menú (on el ratolí interactua amb la UI). L'estructura jeràrquica de la UI s'organitza sota un node `CanvasLayer`, garantint que els elements gràfics es renderitzin sempre per sobre de l'escena 3D.

El sistema distingeix entre tres tipus principals d'interfície:

- **HUD (Heads-Up Display):** Elements persistents com la barra d'accés ràpid (`BottomMenu`), que permet al jugador seleccionar les estructures a construir.
- **Menús de Sistema:** Finestres modals com el menú de construcció (`BuildingMenu`), que mostra el catàleg complet d'objectes disponibles.
- **Interfícies Contextuals:** Menús específics que depenen de l'objecte amb el qual s'interactua, com ara l'inventari d'un forn o la configuració d'una màquina.

Fig. 12: Menú de construcció (part mitjà-superior), barra d'accés ràpid (part inferior)

En la fig. 12, es mostra el menú de construcció, amb les seves quatre pestanyes diferents (part superior) per a organitzar els objectes, i la barra d'accés ràpid (part inferior), omplerta amb una sèrie d'objectes. Per configurar aquesta barra es poden arrossegar les imatges del menú de construcció (part mitjà-superior) fins a la posició de la barra on es vol vincular. Aquesta barra conté nou caselles, cada una vinculada a les tecles 1-9 del teclat.

3.3.2 Components Contextuals

Per vincular la lògica dels objectes 3D amb la interfície 2D sense crear dependències rígides, s'ha implementat el node `InteractionComponent`. Aquest component s'afegeix com a node fill a qualsevol entitat del món (màquines, cofres, etc.) que requereixi una interfície pròpia.

Quan el jugador interactua amb un objecte, el sistema comprova si aquest disposa d'un `InteractionComponent`. En cas afirmatiu, la component instància l'escena de la interfície corresponent (per exemple, `furnace_gui.tscn`) i la injecta al `MenuManager`. Això permet que cada objecte gestioni la seva pròpia lògica visual de forma encapsulada.

3.4 Construcció i interacció amb objectes

El nucli del joc resideix en la capacitat del jugador per modificar l'entorn, col·locar estructures i connectar-les. Aquest sistema es divideix en la lògica de detecció (interacció) i la lògica de col·locació (construcció).

3.4.1 Lògica de detecció

La interacció es basa en el llançament de raigs (*Raycasting*) des de la càmera del jugador. Com s'ha detallat en la secció de senyals, s'utilitza un `InteractionComponent` per estandarditzar el comportament de tots els objectes amb els quals és possible interactuar. Aquest disseny permet diferenciar clarament entre dos tipus d'accions:

1. **Interacció directa:** Accions com obrir un menú o activar un interruptor, gestionades pel senyal `interacted`.
2. **Connexió de ports:** En el cas de les cintes transportadores i eixos, la interacció detecta automàticament els `PowerNodePort` o `MachinePort` propers per facilitar la connexió entre màquines.

3.4.2 Lògica de construcció

El sistema de construcció funciona mitjançant una màquina d'estats implícita gestionada per l'objecte que s'està intentant col·locar. El procés segueix el següent flux:

1. Selecció i Instanciació "Fantasma": Quan el jugador selecciona un objecte del menú, es crea una instància temporal de l'objecte (coneguda com a *ghost* o fantasma). Aquesta instància no té col·lisions físiques actives, però segueix la posició del cursor del jugador, projectada sobre el món 3D.

2. Validació de la posició: Abans de confirmar la construcció, l'objecte executa una funció de validació. Aquesta funció comprova:

- Que no hi hagi col·lisions amb altres objectes existents.
- Que l'objecte estigui recolzat sobre una superfície vàlida (el terra o una altra màquina).
- En el cas d'elements direccionals com les cintes, es calcula la longitud i la rotació necessària entre el punt d'inici i el punt final.

Visualment, aquesta validació es comunica al jugador mitjançant *shaders* que tenyeixen l'objecte de verd (posició vàlida) o vermell (posició invàlida). Tal com es mostra en la fig. 13.

3. Col·locació i Actualització: En confirmar l'acció (clic esquerre), l'objecte passa a ser part permanent de l'escena. En aquest moment:

1. S'activen les seves col·lisions (`CollisionShape3D`).
2. Es registra l'objecte al node pare `Buildings` per garantir-ne la persistència en el guardat.
3. En el cas de nodes de potència (`PowerNode`), es detecten els veïns i es dispara el recàlcul de la xarxa de transmissió mecànica per integrar la nova peça al sistema de grafs.

En la fig. 13 s'observa aquest estat fantasma de l'edifici en els seus dos colors: verd, quan l'objecte pot ser col·locat; i vermell, quan col·lideix amb un altre objecte que n'impedeix la construcció. Finalment, es mostra el resultat un cop ubicat l'edifici.

3.4.3 Construcció Dinàmica: El cas de les Cintes

Un cas especial és la construcció de cintes transportadores (`Belt`). A diferència dels edificis estàtics, les cintes es construeixen definint dos punts. El script de `Belt` gestiona aquest comportament calculant dinàmicament la distància entre els nodes connectats i ajustant l'escala del model 3D i la corba del `Path3D` per adaptar-se a l'espai disponible. Això permet una construcció fluida i flexible, adaptant-se automàticament a la distància entre màquines sense necessitat de col·locar trams individuals peça a peça.

3.5 Guardar i carregar escenaris

Guardar i carregar l'estat del joc és una funcionalitat molt important, rarament es completa un videojoc en una sola sessió, o el jugador no té interès en guardar el seu progrés, fets que fan indispensables un sistema per tal de recuperar l'estat previ. A més per tal de crear nivells de prova i fer la demo jugable era necessari un sistema que permetis guardar i carregar escenaris.

3.5.1 Guardar escenaris

La incorporació d'aquest sistema ha requerit la modificació de l'estructura de persistència de dades dels objectes, canvis al codi executat en el moment d'entrar en escena i en cas de tenir variables que depenen d'altres nodes modificar-les, ja que un node no pot fer referència a un altre en el moment de guardar-lo. Això genera un error de recursivitat infinita per culpa de dependències cícliques.

Tot i tots aquests canvis que s'han hagut de fer el motor Godot proporciona eines molt senzilles i directes per tal de guardar escenes en disc. La primera modificació ha consistit a garantir que tots els objectes a guardar estiguin en una ubicació coneguda, en aquest cas s'ha creat un node anomenat `Building`. Aquest node serveix per a dos propòsits, fa de pare per a tots els nodes del tipus `Building` i manté una llista actualitzada de tots els nodes `PowerNode`.

En prémer el botó de guardar s'executarà una funció recursiva per assegurar-se que tots els nodes (fills, nets, etc.) tinguin la propietat "owner", definida al node `Building`. Aquest pas és imperatiu perquè la funció integrada de Godot "ResourceSaver.save(node, path)", només guarda aquells nodes que el seu "owner" sigui el node passat com a paràmetre.

Aquesta funció permetrà guardar en dos tipus de fitxers diferents, fitxers "tscn" i "scn". El primer és un fitxer de text, similar a "json", que permet una bona llegibilitat i fàcil edició del contingut. És el mateix format que utilitzen les escenes de Godot per defecte, no és un format massa eficient per a l'ordinador però senzill de depurar per al desenvolupador. El format "scn" és un format binari, que aconsegueix temps de guardat i càrrega superiors i una menor mida de fitxer de guardat, per contra, com és un binari és molt complicat de depurar i modificar manualment sense eines externes que l'interpretin.

3.5.2 Carregar escenaris

Per carregar el fitxer de guardat Godot ho fa senzill també, és necessari carregar aquest fitxer com una variable del tipus "PackedScene". Aquesta variable guarda una versió comprimida d'una escena. Una vegada carregat es crea una nova instància de l'escena i s'obté un node `Building` amb tots els elements que contingués en el moment de guardat. Finalment, es substitueix el node anterior pel qual s'acaba de carregar.

3.5.3 Canvis i problemes a l'hora d'implementar el sistema

Explicat el procés de guardat i carrega s'explicaran totes aquelles parts que s'han hagut de modificar per adaptar-les a aquest sistema.

El primer ha sigut marcar amb la propietat "@export"/"@export_storage" totes les variables que es volen guardar. Tota variable que no tingui aquesta propietat serà ignorada per la funció de guardat. La propietat "@export" permet que variables definides en codi siguin persistents i que es puguin modificar des de l'editor del motor, la propietat "@export_storage" permet guardar les variables de forma persistent, però sense que apareguin a l'editor, s'utilitza per variables que són definides en codi i no han de ser mai modificades des de l'editor.

A continuació s'han de modificar totes les variables que guardaven referències directes a nodes, i en el seu lloc guardar un "NodePath". En l'exemple de la fig. 14 s'observa com es mante la propietat "@export" per a modificar aquestes variables des de l'editor i que aquests "paths" es guardin, però en el moment de carregar l'escena s'han de convertir aquests "paths" en les referències directes que guardaran les variables "meshes", "areas" i "collisions".

El codi per executar aquesta conversió s'executarà quan es creï l'objecte i en tot moment que aquest objecte es carregui des de la memòria.

Un altre component important que necessita ser executat en aquests moments és el codi encarregat de connectar els diferents senyals utilitzats. En el moment de guardar i carregar aquests senyals es desconnecten i han de ser reconnectades.

Finalment, tots els `PowerNode` hauran de tornar-se a connectar amb els nodes adjacents i es farà un recàlcul sencer de la xarxa de potència, per tal d'evitar possibles errors que es podrien produir si no es comprova que les velocitats de rotació dels nodes no són les correctes. Durant el procés de càrrega cal considerar que la detecció de col·lisions de les "Area3D" no depenen

Fig. 13: Exemple dels tres estats en el moment de construir un objecte, respectivament: posició vàlida, posició invàlida i posició final.

```
@export_group("Meshes, areas and collistions of the building")
@export var meshes_path : Array[NodePath]
@export var areas_path : Array[NodePath]
@export var collisions_path : Array[NodePath]

var meshes: Array[MeshInstance3D]
var areas : Array[Area3D]
var collisions: Array [StaticBody3D]
```

Fig. 14: Codi actual per guardar les referències a nodes

del fotograma lògic, sinó del cicle de física, per tant, per evitar possibles condicions de carrera, i comportaments no deterministes cal implementar una espera explícita mitjançant la instrucció ”await get_tree().physics_frame”. Això endarrerirà l’execució de les físiques un fotograma sencer, i s’assegura que el fotograma lògic s’hagi completat.

4 Conclusions

Al llarg del projecte s’han desenvolupat molts components diferents que tots han contribuït al producte final. S’ha intentat separar el desenvolupament en mòduls independents que no afectaven els uns als altres, per tal de facilitar la implementació i testatge del videojoc. Per exemple:

- Els sistemes de construcció i el de potència operen de manera independent, interactuant únicament a través d’un senyal puntual.
- La gestió de les propietats dels diferents ítems transportats per cintes és independent a la cinta en si, aquesta només transporta un `PathFollow3D` que resulta que té vinculat un recurs amb les propietats.
- Gràcies a la diferenciació entre `PowerNode` i `PowerNodePort`, es poden afegir els ports necessaris a qualsevol `PowerNode`, amb les propietats que es desitgin (que redueixi o augmenti la velocitat o canviï el sentit), sense modificar la lògica i es poden fer fàcilment objectes com multiplicadors o reductors de velocitat de gir sense escriure una sola línia de codi (això no és completament cert perquè la lògica per a les animacions visuals és diferent per a múltiples elements, però per al càlcul lògic de velocitat i connexió si es compleix).

Això ha facilitat enormement el testatge i l’escalabilitat del codi.

Tots aquests sistemes han tocat parts diferents del motor Godot, i han permès aprendre els fonaments d’aquesta eina, en concret el sistema que més ha requerit aprendre sobre les particularitats del motor ha sigut el guardat i càrrega d’escenaris, ja que s’ha treballat com funciona la persistència de dades, formats dels fitxers de guardat, funcionament de les referències entre nodes, la necessitat de canviar la propietat ”owner” per a guardar l’objecte, en el moment d’entrar en escena quines són les diferències entre les diferents funcions d’inicialització de nodes, en quin moment del cicle lògic es calculen les físiques, reconnectar senyals en el moment d’entrada, etc. La resta de sistemes han sigut més generals, en el sentit que no requereixen coneixement del motor concret, com conceptes com el càlcul vectorial o algoritmes per recórrer i manipular grafs.

Finalment, el projecte ha servit per constatar la complexitat inherent al desenvolupament de videojocs de simulació. La necessitat de coordinar sistemes físics (col·lisions, *backpressure* en cintes) amb estructures de dades lògiques (grafs de potència) ha requerit un estudi exhaustiu de la documentació [10] i una iteració constant. El resultat és una base sòlida, funcional i extensible que compleix amb els requisits d’un joc d’automatització. Com a incidències, cal destacar:

- Haver de modificar pràcticament totes les classes per adaptar-se al sistema de guardat i càrrega.
- Durant la previsualització dels objectes a col·locar desactivar les col·lisions perquè no col·lideixin amb el jugador.
- Afegir els desplaçaments de centre en la fórmula (4) C_{conn}/C_{local}, això és indispensable per al cas de l’extractor de recursos, que l’engranatge gran que l’alimenta no funcionava correctament, ja que el centre de l’objecte i el centre de gir eren diferents, i produïa errors en la direcció.
- La gestió d’incoherències en el càlcul de velocitat dels nodes, eliminar el node que provoca el conflicte.
- El `shader` que controla la textura de la cinta, perquè aparegués correctament en la part superior, inferior, inici i final, sense els costats.

4.1 Línies de treball futur

Tot i que Engraginy estableix una base robusta, el gènere de l'automatització es caracteritza per la seva profunditat i escala. Per aquest motiu, es proposen les següents línies de millora i expansió per a futures iteracions del projecte:

- **Sistema de progressió i recerca:** Implementació d'un arbre de tecnologies que permeti al jugador desbloquejar nous objectes i màquines a mesura que assoleix fites de producció.
- **Expansió de contingut:** Creació d'un mapa més extens amb biomes i distribució de recursos estratègica, així com nous tipus de màquines i recursos per processar.
- **Gestió de l'inventari del jugador:** Desenvolupament d'un sistema d'inventari personal que permeti gestionar recursos manualment, no només a través de les màquines.
- **Mecànica de fluids:** Incorporació de canonades, bombes i tancs per gestionar líquids i gasos, afegint una nova capa de complexitat a la simulació.
- **Optimització gràfica i visual:** Millora dels models 3D i textures, així com optimització del rendiment (LOD, *culling*) per suportar fàbriques massives amb milers d'elements en moviment.
- **Definició d'un "Endgame":** Establiment d'un objectiu final clar (per exemple, la construcció d'una mega-estructura) per donar un sentit de conclusió a la partida.

Agraïments

En primer lloc, vull expressar el meu sincer agraïment al meu tutor, Enric Marti Godia, per la seva guia indispensable en la definició dels objectius i l'estructuració d'aquest treball, així com pels seus consells per millorar la qualitat i claredat de l'escriptura tècnica.

Vull fer una menció especial a la meva parella, Yoeliana, per la seva infinita paciència i per haver-me escoltat sempre que tenia dubtes. La seva capacitat per oferir un punt de vista alternatiu quan em trobava encallat ha estat una ajuda inestimable per tirar endavant.

També vull donar les gràcies al meu amic Xavi, que com a estudiant de matemàtiques m'ha ajudat a polir i validar la part teòrica del sistema de càlcul de velocitats, a més de llegir-se aquest informe i aportar suggeriments de millora essencials.

Finalment, agraeixo als meus pares el seu suport constant i per haver-me escoltat, dia rere dia, quan em queixava que un canvi aparentment inofensiu havia trencat completament el projecte.

References

[1] Wube Software. *Factorio*. [Videojoc]. Disponible a: https://www.factorio.com. [Últim accés: 11/2025].

[2] Coffee Stain Studios. *Satisfactory*. [Videojoc]. Disponible a: https://www.satisfactorygame.com. [Últim accés: 09/2025].

[3] Godot Engine. *Godot Engine - Free and open source 2D and 3D game engine*. [Programari]. Disponible a: https://godotengine.org. [Últim accés: 12/2025].

[4] Youthcat Studio. *Dyson Sphere Program*. [Videojoc]. Disponible a: https://dsp.youthcat.com. [Últim accés: 09/2025].

[5] tobspr Games. *Shapez*. [Videojoc]. Disponible a: https://shapez.io. [Últim accés: 09/2025].

[6] simibubi et al. *Create Mod for Minecraft*. [Modificació de programari]. Disponible a: https://www.curseforge.com/minecraft/mc-mods/create. [Últim accés: 10/2025].

[7] Viquipèdia. *Esfera de Dyson*. [En línia]. Disponible a: https://ca.wikipedia.org/wiki/Esfera_de_Dyson. [Últim accés: 01/2026].

[8] E. W. Weisstein. "Breadth-First Traversal". *MathWorld–A Wolfram Web Resource*. [En línia]. Disponible a: https://mathworld.wolfram.com/Breadth-FirstTraversal.html. [Últim accés: 01/2026].

[9] E. W. Weisstein. "Depth-First Traversal". *MathWorld–A Wolfram Web Resource*. [En línia]. Disponible a: https://mathworld.wolfram.com/Depth-FirstTraversal.html. [Últim accés: 01/2026].

[10] Godot Engine Community. *Godot Engine Stable Branch Documentation*. [Documentació oficial]. Disponible a: https://docs.godotengine.org/en/stable/. [Últim accés: 01/2026].

Apèndix