Universitat de Girona
Escola Politècnica Superior

Grau en Enginyeria Informàtica

PROJECTE FINAL DE GRAU

Videojoc Android estil *Escape Room* de Realitat Augmentada

Autor:
Martí Parnau Fàbregas

Tutors:
Dr.Gustavo Ariel Patow
Dr. Sergio Gonzalo Besuievsky Glikberg

MEMÒRIA

Convocatòria:
Setembre 2023

Departament :
Departament d'Informàtica, Matemàtica Aplicada i Estadística

Projecte: Projecte Final de Grau
Document: Memòria
Títol: Videojoc Android estil *Escape Room* de Realitat Augmentada
Autor: Martí Parnau Fàbregas
Data: Setembre 2023

Estudi:
Grau en Enginyeria Informàtica
Universitat de Girona

Supervisor 1:
Dr.Gustavo Ariel Patow
Universitat de Girona
Email: gustavo.patow@udg.edu
Web: Enllaç

Supervisor 2:
Dr. Sergio Gonzalo Besuievsky Glikberg
Universitat de Girona
Email: gonzalo.besuievsky@udg.edu
Web: Enllaç

Índex

Capítol 1

Introducció, motivacions, propòsit i objectius del projecte

Els hologrames i la interacció amb elements virtuals representats al mon real han estat presents al cinema de ciència ficció al llarg de la història, i ha estat un element fascinant pels humans des que la tecnologia no ho permetia. Avui en dia, existeixen múltiples dispositius que son capaços d'oferir experiències similars a les que estem acostumats a veure en pel·lícules, algunes d'aquestes inassequibles pel ciutadà mitjà, com les **Appple Vision** o les **Hololens** de Microsoft, d'altres ja formen part del dia a dia de quasi tothom, com els dispositius Android.

Tot i això, l'abast d'aquesta tecnologia no està arribant al públic, probablement per la falta d'idees dutes a terme en el sector de la realitat augmentada.

1.1 Motivacions

Aquest projecte ofereix la capacitat d'aportar a la comunitat de la realitat augmentada una idea innovadora, transportant un concepte d'entreteniment ja existent i que agrada a la societat, com és un *EscapeRoom*, a aquesta tecnologia, a la vegada que explorar les possibilitats de desenvolupament amb eines de *AR* amb pocs recursos.

Considero que el projecte és una idea innovadora, amb moltíssim potencial i capacitat, que es desvia del videojoc tradicional però n'hereta la majoria de les coses bones d'un videojoc.

1.1.1 Motivacions personals

La fascinació per les capacitats que ofereix la tecnologia i l'evolució d'aquesta és un dels motius que m'empenya a creure en la realitat augmentada. Veig aquest projecte com una oportunitat per tenir un primer contacte amb el disseny i desenvolupament d'aplicacions d'*AR*, així com l'aprenentatge de les eines de desenvolupament, llenguatges i recursos que hi ha actualment.

A la vegada, és una ocasió per aprendre sobre el disseny i desenvolupament de videojocs, que si bé no és el sector al que em vull dedicar a dia d'avui, és una porció del mon del *software* el qual m'agradaria conèixer abans d'acabar el meu període a la universitat.

1.2 Objectius i Propòsits

En aquesta secció, aprofundim en els propòsits i objectius fonamentals del projecte.

El propòsit general del projecte és la creació d'un prototip d'EscapeRoom de realitat augmentada que ofereixi als jugadors una experiència de joc que fusioni el món virtual i el real,

buscant la màxima interacció amb els elements virtuals a través d'accions al mon real per tal de millorar la sensació d'immersió del jugador.

A més, es vol aconseguir que el jugador senti que la partida passa a la sala on està, i no a en un dispositiu tecnològic.

Els objectius a assolir, son els següents:

- Aprendre a desenvolupar amb Unreal Engine
- Aprendre a utilitzar les funcionalitats imatges augmentades, detecció de plans i "ancoratge" de punts de la llibreria ARCore
- Disseny dels diferents "minijocs" enigmàtics que tinguin la major interacció entre el mon real i virtual. Entre aquests, destacaria:
  - La combinació dels elements presents a la imatge captada per la càmera.
  - L'afectació de la llum entre elements virtuals depenent de la seva col·locació al mon real.
  - La interacció amb els elements virtuals al toc de pantalla del mòbil.
  - La col·lisió entre elements virtuals representats a l'escenari real.
  - La inclinació del pla real on es representa l'element virtual.
  - L'us de la posició del jugador com a interacció amb els elements del mon virtual.
  - Arrodonir i finalitzar la documentació del treball realitzat.

Capítol 2

Estudi de viabilitat

L'estudi de viabilitat del projecte s'ha realitzat sobre els recursos tecnològics. Tal com s'especifica en els objectius, aquest projecte té com a propòsit el disseny i implementació d'un prototip de videojoc funcional, i aquest és l'estudi de viabilitat que s'ha realitzat.

2.1 Recursos Tecnològics

Els recursos de sistema recomanats per desenvolupar amb Unreal Engine són:

- Sistema Operatiu: Windows 10 64-bit version 1909 revision .1350 or higher, or versions 2004 and 20H2 revision .789 or higher.
- Processador: Quad-core Intel or AMD, 2.5 GHz or faster
- Memoria:8 GB RAM
- Targeta gràfica: compatible amb DirectX 11 or 12
- RHI Version:DirectX 11:
  - Latest drivers
  - DirectX 12: Latest drivers
  - Vulkan: AMD (21.11.3+) and NVIDIA (496.76+)

Pel que fa la llibreria de realitat augmentada escollida, ArCore, no hi ha un uns recursos mínims específics, però l'empresa ofereix un llistat de dispositius que han passat un procés de certificació on s'ha comprovat la qualitat de la càmera, dels sensors de moviment i el disseny de l'arquitectura per assegurar que el rendiment serà l'esperat.

En el cas del projecte s'ha utilitzat el dispositiu android `One Plus 9`, present a la llista de dispositius compatibles amb la llibreria ArCore.

2.2 Recursos Humans

En cas que volguéssim convertir el prototip en un producte comercial, caldria involucrar a un conjunt de professionals de diferents sectors. A trets generals, aquests serien:

- Programador especialista en c++ d'Unreal Engine: encarregat d'implementar les mecàniques del videojoc.
- Programador especialista en Blueprints d'Unreal Engine: encarregat de desenvolupar els diferents nivells i característiques de la partida utilitzant les mecàniques disponibles.
- Product Owner: encarregat d'analitzar els requeriments del producte i dissenyar la part creativa del videojoc.
- Dissenyador 3D: encarregat de realitzar la part de disseny visual del videojoc.
- Especialista en producció musical i d'àudio de videojocs: encarregat del so i la música del videojoc.

Els sous mitjans a Espanya segons diferents fonts d'internet son:

- Programador Unreal Engine: 34.000 euros anuals.
- Product Owner: 35.000 euros anuals.
- Dissenyador 3D: 26.500 euros anuals
- Producció Musical i d'àudio: 22.000 euros anuals.

Considerant que el projecte tindria una duració mínima d'un any, el cost mínim seria d'uns 160000 euros anuals només en sou brut a treballadors.

Capítol 3

Metodologia

Un dels entrebancs del projecte ha estat la manca de coneixements en el desenvolupament de videojocs i, en concret, la realitat augmentada. És per això que, malgrat les diferents metodologies de desenvolupament de *software*, com *watterfall*, *Agile*, *Scrum* o *Kanban*, s'ha escollit una metodologia alternativa plantejada pels tutors. Aquesta, consistia en diferents etapes personalitzades pel projecte.

3.1 Recerca i Selecció

La fase inicial implicaria realitzar una investigació de les diferents tecnologies i eines disponibles per crear videojocs de Realitat Augmentada, com motors de joc amb suport a la realitat augmentada, *frameworks*, o *Kits de desenvolupament (SDK)* de *AR* per plataformes android.

Considerant els objectius i requeriments del projecte, es prendria la decisió de quines eines s'utilitzarien.

3.2 Aprenentatge de la tecnologia seleccionada

Amb la tecnologia seleccionada, la següent fase consistiria en dedicar-se a l'aprenentatge realitzant formacions *online*, petits projectes per entendre'n el funcionament, estudiar la documentació i conèixer l'entorn i la comunitat que engloba l'eina.

3.3 Prova de concepte

Realitzar una prova de concepte, és a dir, el *Hello World* del projecte. En el meu cas, aquesta seria implementar una aplicació capaç d'escanejar múltiples imatges i generar-ne un element virtual i visible a través de la pantalla del mòbil.

3.4 Disseny i implementació de mecàniques i estructures bàsiques

Un cop amb les nocions bàsiques d'Unreal Engine i Realitat augmentada, s'implementarien totes les mecàniques i estructures base per satisfer els objectius inicials del projecte, passant per una fase de proves i resultats inicial i pensant en el disseny per ser utilitzades posteriorment en el fil del videojoc.

3.5 Disseny i Implementació del videojoc

Amb les mecàniques i estructures definides i implementades, el següent pas seria incorporar-les en el disseny d'un videojoc, combinant-les en diferents puzzles, definint un fil de partida i construint el videojoc.

Capítol 4

Planificació

En aquest capítol es defineix l'estratègia seguida per arribar als objectius plantejats, descrivint breument el pla de treball i les tasques planificades, amb el temps estimat i els resultats esperats de cada tasca.

Cal tenir en compte que les dates planejades son aproximades.

El projecte s'inicia el desembre de 2022 i amb previsió d'entregar-lo el juny de 2023. Seguint la metodologia proposada en l'apartat anterior, els terminis previstos son els següents:

- **Recerca i Selecció**(20221201 - 20221215) → Durant aquest període s'espera concloure amb una decisió amb fonaments sobre la tecnologia escollida, amb una idea sobre els objectius del projecte i el treball que s'espera realitzar, també les eines que s'utilitzaran: el motor gràfic, les llibreries de AR, el dispositiu pel qual es desenvoluparà, el llenguatge, etc.

- **Aprenentatge de la tecnologia seleccionada i "Hello world"del projecte**(20221215 - 20230201) → L'aprenentatge estarà basat en realitzar un curs genèric de desenvolupament de videojocs amb Unreal Engine en c++ a la plataforma Udemy.[1] Amb aquest curs s'espera tenir prou coneixement per implementar el "Hello World"amb les característiques que es comenten a l'apartat de metodologia. Amb el "Hello World", es busca també tenir tot l'entorn configurat: el dispositiu de desenvolupament amb els llenguatges i compiladors instal·lats, així com els sdk de les llibreries amb les versions corresponents, les versions d'android, l'editor de text configurat per treballar amb l'editor d'Unreal Engine, entre d'altres.

- **Disseny i implementació de mecàniques i estructures bàsiques**(20230201 - 20230315) → Definició i implementació de les mecàniques: interacció amb elements virtuals al clic de la pantalla, moviment de l'element virtual a l'inclinació de la targeta física, sistema de làsers i miralls, etc. Cal dir que durant aquest període no es van implementar totes les mecàniques, ja que amb l'experiència i l'aprenentatge van sorgir noves idees i maneres de fer quelcom, i es va implementar en la següent etapa.

- **Disseny i Implementació del videojoc** (20230315 - 20230415) → Part més creativa, disseny dels diferents nivells en funció de les mecàniques disponibles, definició de les combinacions de mecàniques, eines, seqüència del videojoc..

- **Polit i finalització** (20230415 - 20230501) → En cas de tenir temps, millorar tots aquells detalls que queden pendents durant el projecte o que son millorables.

- **Documentació** (20230501 - 20230601) → Redacció i finalització de la memòria del projecte.

Aquest projecte s'ha dut a terme a la vegada que treballant lluny de casa i en mig d'un canvi de feina que ha requerit la realització de cursos per al nou lloc laboral, pel que tota la planificació s'ha endarrerit lleugerament.

També, per assegurar un bon treball, s'ha prioritzat la completesa d'aquest a la rapidesa en acabar-lo, traslladant l'entrega del juny de 2023 a la del setembre 2023.

Capítol 5

Marc de treball i conceptes previs

En aquesta secció es descriuen els diversos aspectes relacionats amb el desenvolupament general del projecte per tal d'assegurar que el lector pugui entendre els capítols següents, situant-lo en la temàtica que engloba al treball.

5.1 Descripció general AR EscapeRoom

5.1.1 Realitat Augmentada

FIGURA 5.1: Ulleres de Realitat Augmentada

FIGURA 5.2: Realitat augmentada d'Android

La realitat augmentada és una tecnologia que incorpora elements digitals al món real, creant una experiència interactiva i immersiva. Superposa informació generada per dispositius tecnològics com ara imatges, vídeos, sons o models, a l'entorn físic.

A diferència de la realitat virtual, que substitueix el món real per un de simulat, la realitat augmentada combina elements virtuals amb el món real. Normalment s'experimenta a través de dispositius com ara telèfons intel·ligents, tauletes, ulleres intel·ligents o dispositius portàtils. Aquests dispositius utilitzen les càmeres, sensors i potència de processament per capturar l'entorn del món real i augmentar-lo amb contingut virtual en temps real. Els usuaris poden veure i interactuar amb aquests elements virtuals com si existissin a l'espai físic.

En general, la realitat augmentada millora la percepció i la interacció de l'usuari amb el món real combinant el contingut digital perfectament amb l'entorn, obrint noves possibilitats d'entreteniment, educació, comunicació i resolució de problemes.

5.1.2 Escape Room

FIGURA 5.3: Escape Room

Un joc estil *Escape Room* ofereix una experiència d'entreteniment immersiva popular que desafia els participants a resoldre trencaclosques i completar tasques en un període de temps determinat per "escapar"d'una sala o escenari temàtic. Normalment implica un grup de jugadors que treballen junts per descobrir pistes, resoldre enigmes, manipular objectes i desbloquejar compartiments ocults per avançar en el joc.

Els equips que entren a una *Escape Room* solen rebre una història de fons o un objectiu que estableix l'escenari per a la missió. Han de buscar a l'habitació pistes, codis o claus ocults que els ajudin a desbloquejar nous reptes o a obrir portes tancades. Els trencaclosques poden incloure raonament lògic, reconeixement de patrons, càlculs matemàtics, consciència espacial i resolució creativa de problemes. La finalització amb èxit d'un trencaclosques sovint porta al següent, descobrint progressivament la narració i apropant els jugadors al seu objectiu final d'escapar de l'habitació.

5.1.3 AR EscapeRoom

La unió dels dos conceptes previs dona lloc al videojoc del projecte: una *Escape Room* de Realitat Augmentada.

Una **Escape Room de realitat augmentada** és un tipus d'*Escape Room* que superposa els elements virtuals que construeixen una *Escape Room*, en el mon real. D'aquesta manera, la interacció del jugador amb els elements virtuals i per tant tot el fil de la partida estarà passant en el mon real de l'usuari a través del dispositiu tecnològic de realitat augmentada, aconseguint una experiència més interactiva i immersiva.

Generalment, s'utilitzen targetes amb una imatge definida que el dispositiu de realitat augmentada detecta i relaciona a un element virtual per situar-lo en en el mon real. Això permet tenir un element físic amb el que l'usuari pot interactuar afectant característiques de l'element virtual, com la posició, la inclinació, o la rotació.

5.2 Descripció general dels conceptes tècnics del projecte

5.2.1 Visual Studio Code

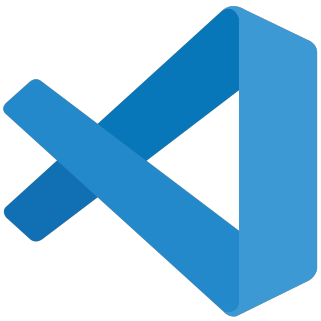

FIGURA 5.4: Logo de Visual Studio Code

Visual Studio code (VS Code) és l'editor de codi *open-source* creat per Microsoft. Implementat amb Typescript i convertit a aplicació d'escriptori amb Electron, s'ha convertit en un dels editors de codi gratuïts més versàtils i complets gràcies a les extensions creades per la gran comunitat que ha aconseguit VS Code, fent-lo capaç d'integrar-se amb qualsevol llenguatge, *framework* o eina de desenvolupament de software a dia d'avui. [2] [3]

5.2.2 Motors Gràfics

FIGURA 5.5: Logo d'Unreal Engine

FIGURA 5.6: Logo de Unity [4]

Un motor gràfic és un *framework* utilitzat per renderitzar escenes gràfiques. S'encarrega dels detalls de renderització a baix nivell, com les textures de les superfícies, el dibuixat de polígons o el calcul dels efectes de llum. A dia d'avui, els més coneguts son Unreal Engine, Unity, i Godot.

5.2.3 Unreal Engine

Unreal Engine[5] és la plataforma de desenvolupament de videojocs creada per Epic Games. Destaca per les seves capacitats avançades de renderització de gràfics, il·luminació realista i textures detallades, les quals permeten la creació de jocs molt realistes i visualment impressionants. El llenguatge de programació utilitzat per desenvolupar amb aquest motor gràfic és c++, però també ofereix els `blueprints`.

Els `blueprints` són una forma de programació visual a partir d'un sistema de nodes de l'estil *drag and drop* que permeten a desenvolupadors amb menys experiència en llenguatges de codi tradicional realitzar funcions i lògiques complexes del videojoc abstraient certa complexitat del codi.

Alguns conceptes importants del framework d'Unreal Engine:

- **Static Mesh** → Peça de geometria que consisteix en un conjunt de polígons que es poden emmagatzemar a la memòria de vídeo i ser renderitzats per la targeta gràfica.
- **Level**→ Col·lecció de volums, llums, plans i molt més que treballen conjuntament per oferir l'experiència desitjada al jugador. Els nivells a UE5 poden variar de mida, des de mons massius basats en el terreny, fins a nivells molt petits que contenen uns quants actors.
- **Actor**→ Qualsevol objecte que pot ser colocat en un nivell, així com una camera, un *Static Mesh*, o un jugador. Els actors suporten transformacions com translació, rotació o escalat, i poden ser generats (*Spawned*) i destruïts mitjançant el codi de la partida.
- **GameMode**→ Classe que gestiona les normes de la partida, així com el nombre de jugadors, com el jugador entra a la partida, la posició del mon on es genera el jugador, si el joc es pot pausar, o aspectes similars sobre l'estat de la partida.
- **PlayerControler**→ Classe que s'utilitza com a interfície entre el Pawn i el jugador humà controlant-lo, enfocat a gestionar l'estat persistent de lo relacionat amb el jugador en sí i les accions que l'humà pot fer. Per defecte, el *PlayerController* té una relació 1 a 1 amb un Pawn.
- **Pawn**→ Classe base de tots els actors que poden ser controlats per jugadors o IA. El **Pawn** és la representació física del jugador o l'entitat IA en el mon. No només determina l'aspecte del jugador, ja que a vegades no té un aspecte visible sinó una posició, rotació, escala, etc, de l'entitat dins el joc, sinó també com aquest interacciona en el mon
- **Widget**→ component que permet mostrar elements 3D d'interficie d'usuari creats a través de l'editor d'Unreal Motion Graphics. El *Widget* en si mateix és una instància ambla que pots interactuar en el mon de la partida.

5.2.4 Hand Tracking

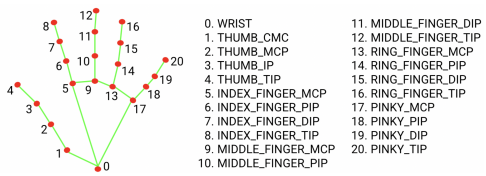

FIGURA 5.7: *Landmarks* del Hand tracking

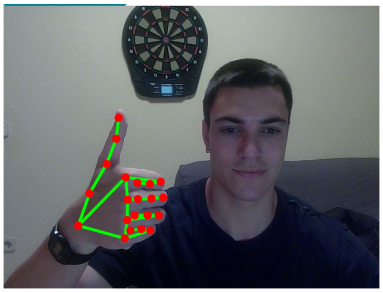

FIGURA 5.8: Exemple de Hand tracking

El *hand tracking* en la realitat augmentada (RA) fa referència a la tecnologia i les tècniques que permeten als sistemes AR reconèixer i fer un seguiment dels moviments i gestos de les mans d'un usuari en temps real, sense necessitat de controladors físics o guants. Permet als usuaris interactuar amb objectes virtuals i interfícies en entorns de realitat augmentada simplement utilitzant les seves mans, fent que les experiències d'AR siguin més intuïtives i immersives. El *hand tracking* normalment implica l'ús de càmeres i algorismes de visió per ordinador per capturar i interpretar els moviments i les posicions de les mans de l'usuari. [6]

5.2.5 HUD

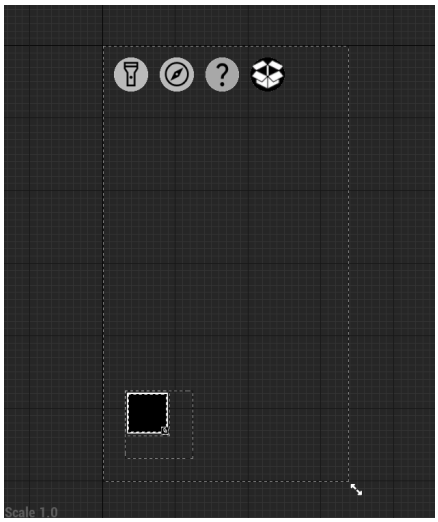

FIGURA 5.9: Exemple de HUD

Als jocs, un HUD (Heads-Up Display) és una interfície en pantalla que mostra informació essencial als jugadors durant el joc. Aquesta informació sovint inclou elements com ara barres de salut, recomptes de municions, mini-mapes, objectius i altres dades relacionades amb el joc. El HUD millora l'experiència de joc proporcionant als jugadors informació crucial, cosa que els permet mantenir-se completament immersos en l'acció.

5.2.6 ArCore

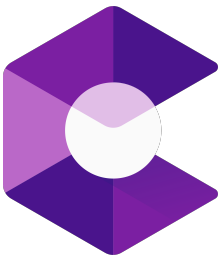

FIGURA 5.10: Logo ArCore

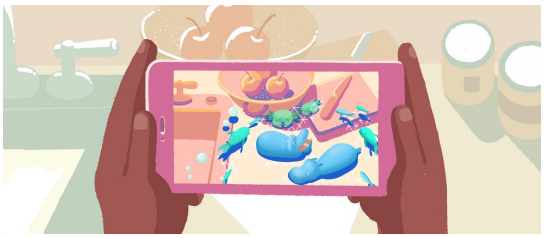

FIGURA 5.11: Dibuix ArCore

ArCore és un *kit de desenvolupament (SDK)* de Google que ofereix API's per crear experiències de realitat augmentada multi-plataforma. Les funcionalitats que s'han considerat de cares a la realització del projecte son les següents: [7] [8]

- **Augmented Images**: L'API d'**Imatges Augmentades** permet detectar imatges 2D a través de la imatge captada per una càmera. Donat un conjunt d'imatges de referència, ArCore utilitza un algorisme de visió per computador per extreure característiques de la informació en escala de grisos de cada imatge i emmagatzema una representació d'aquestes en una base de dades d'Imatge Augmentada.

  En temps d'execució, ArCore busca aquestes característiques en superfícies planes de la imatge captada per la càmera, permetent detectar les imatges en el mon real, estimar la posició, l'orientació i la mida d'aquestes.

- **Depth**: L'api de *Depth* utilitza un algoritme de profunditat a partir del moviment per crear **imatges de profunditat**, que ofereixen una vista en 3D del mon. Cada píxel en

una *imatge de profunditat* està associat a la mesura de l'escena a la càmera. L'algoritme agafa múltiples imatges de diferents angles i les compara per estimar la distància de cada píxel mentre l'usuari mou el dispositiu. També s'aprofita el *hardware* addicional que pugui tenir el dispositiu, com un `Sensor ToF` o un `acceleròmetre`.

- **ARPins**: Els ARPins son posicions fixes del mon real a la realitat augmentada als quals es pot assignar contingut virtual.

Capítol 6

Requisits del sistema

En aquest capítol es descriuen els requisits que ha de complir el sistema, per tal d'establir les bases del que serà desenvolupat tant des del punt de vista del programari com de maquinari.

6.1 Requeriments no funcional

Els requeriments no funcionals són bàsicament les restriccions de qualitat que el sistema ha de satisfer, com per exemple la seguretat, facilitat de manteniment, fiabilitat, escalabilitat, rendiment o capacitat de reutilitzabilitat.

Els requeriments no funcionals definits pel projecte son:

- **Sistema de versions**: per tal de tenir un control del codi durant el desenvolupament, un dels requisits no funcionals ha estat tenir un sistema de control de versions del codi. S'ha escollit Github com a plataforma de repositoris Git, tot i que la metodologia aplicada durant el desenvolupament del projecte no era l'adequada per aprofitar totes les característiques d'un repositori Git.

- **Escalabilitat**: Si bé un *EscapeRoom* és una modalitat de joc on la partida és molt personalitzada i té un fil i una lògica dels *puzzles* molt concreta i definida, un dels requeriments funcionals que s'estableix al projecte és l'escalabilitat. No parlem d'escalabilitat del sistema a nivell de recursos i usuaris, ja que actualment no hi ha cap funcionalitat que requereixi servidors, sinó escalabilitat en el desenvolupament: el disseny del videojoc ha de permetre afegir, treure o modificar nivells o *puzzles*, també funcionalitats de la partida en si.

- **Reutilitzabilitat**: Tot i assenyalar el comentari del punt anterior sobre la personalització que té cada nivell en una *EscapeRoom*, la reutilitzabilitat en el projecte és un requeriment: cal que les mecàniques implementades siguin capaces de ser utilitzades en el nivell que ho necessiti, així com que implementar un nou nivell sigui prou senzill com per només haver d'implementar la lògica del *puzzle*.

- **Facilitat de manteniment**: es busca l'abstracció de certes lògiques en c++, per tal de poder desenvolupar actors que hereden aquestes classes utilitzant Blueprints.

6.2 Requeriments funcionals

Els **Requeriments funcionals** son els les funcionalitats que el producte final determinat pel projecte ha d'incorporar al sistema. Els requeriments funcionals del projecte son els següents:

- **POV**: La partida es juga amb *POV first-person augmented reality*, és a dir, la càmera del mòbil i el que es mostra a la pantalla representa la vista del jugador dins la partida.

- **Virtualització d'elements a través de targetes**: la funcionalitat principal del joc, representar elements digitals en un espai del mon real, per fer-ho, s'utilitza la tecnologia d'***Augmented Images***.
- **Interacció amb els elements digitals**: el jugador pot interactuar amb els elements virtuals de les diferents maneres (sense haver de ser tots aquests compatibles amb totes les maneres d'interacció):
  - **Clic**: utilitzant la pantalla tàctil del dispositiu *android*.
  - **Moviment del mòbil**: mitjançant la posició, rotació o inclinació del dispositiu.
  - **Moviment de la targeta**: mitjançant la posició, rotació o inclinació de la targeta física.
- **HUD**:HUD en pantalla amb els botons per encendre o apagar el flaix i obrir la brúixola, l'inventari, o sol·licitar ajuda. També un espai per visualitzar quin és l'element d'inventari seleccionat.
- **Funcionalitats del HUD**: eines que s'obren en utilitzar els botons descrits anteriorment i que son necessàries per resoldre tots els nivells.
  - **Brúixola**: Brúixola funcional.
  - **Sol·licitar ajuda**: Sistema de pistes que mostra en pantalla una pista en funció del nivell al que estigui la partida o del nivell al qual s'està enfocant.
  - **Sistema d'inventari**: Inventari del jugador amb els articles que conté. En ser desplegat l'inventari, permet visualitzar els articles i seleccionar un dels elements per ser utilitzat.
  - **Llum flaix**: llum situada en la posició del flaix real però que afecta l'espai virtual i que pot ser utilitzada per il·luminar elements virtuals apropant el dispositiu al mon real.
- **Menú d'inici de partida**: un menú que permet a l'usuari iniciar la partida.
- **Pausa**: un botó i pantalla de pausa de la partida.
- **Fi partida**: una pantalla dedicada a informar al jugador que ha completat la partida.

Capítol 7

Estudis i decisions

En aquest capítol es tractaran els diferents aspectes que s'han tingut en compte per prendre les decisions sobre la tecnologia utilitzada per desenvolupar el sistema.

7.1 Dispositiu de Realitat Augmentada

Actualment hi ha múltiples dispositius específicament de realitat augmentada, però un dels objectius del projecte és crear un producte amb caràcter d'entreteniment per tal de fer arribar a més public aquesta tecnologia.

Els dispositius dissenyats per ser de realitat augmentada requereixen un pressupost bastant elevat i no son fàcils d'aconseguir. Alguns dels dispositius més coneguts:

- Microsoft HoloLens 2: 3849 €
- Apple Vision Pro: 3.143 €
- Meta Quest Pro: 1.347 €

No obstant, n'hi ha un compatible amb la realitat augmentada i que pràcticament tothom posseeix: un *Smartphone Android*. A més, actualment quasi qualsevol tecnologia permet desenvolupar per dispositius Android i probablement és pel que més funcions de realitat augmentada s'ha desenvolupat, com els coneguts **filtres d'Instagram** o videojocs com **Pokemon Go**. És per això que el dispositiu escollit és l'***Smartphone Android***.

7.2 Llenguatge de programació

Una de les tecnologies que generalment més fascina de la realitat augmentada és el *Hand Tracking*, ja que ofereix una interacció directe amb l'element virtual sense passar pel contacte amb el dispositiu. És per això que un dels requeriments funcionals del **Treball futur** és la incorporació de *Hand Tracking*. Per tant, es vol fer el prototip de videojoc capaç d'absorbir aquesta tecnologia en un futur, i ***Mediapipe*** és la llibreria que s'utilitzaria.

El llenguatge de programació depèn directament del motor gràfic escollit, i tot i que va ser una decisió que es va prendre en paral·lel, la dependència amb la llibreria *Mediapipe* pel treball futur (entre d'altres característiques comentades en el següent apartat) es va decidir escollir el c++ com a llenguatge de programació del videojoc.

7.3 Motor Gràfic

L'elecció del motor gràfic té una dependència recíproca amb el llenguatge, a l'igual que amb la següent secció *Llibreria de realitat augmentada*. La decisió entre motors gràfics és entre **Unity**[4] i **Unreal engine**.

Els dos motors son capaços de satisfer tots els requeriments funcionals del projecte i els recursos necessaris son similars. És per això que el comentari de l'apartat anterior va ser decisiu a l'hora d'escollir Unreal Engine, tot i que pel projecte en si no era determinant.

També m'agradaria comentar, que a nivell personal i de manera totalment subjectiva, tenia certa atracció i motivació per l'aprenentatge d'Unreal Engine.

7.4 Llibreria/framework de realitat augmentada

De nou, aquesta elecció era entre el framework de Unity, **Vuforia** [9], o la llibreria de Unreal Engine, **ArCore**.

Els dos tenen unes funcionalitats similars i més que suficients pels requeriments del projecte. Sense haver treballat mai amb cap dels dos ni amb cap motor gràfic, vaig tenir la sensació que amb ArCore treballaria a una mica més baix nivell, característica que em motivava per tal d'aprendre'n bé el funcionament.

Aquests detalls en combinació amb els dels apartats anteriors, van concloure en utilitzar la llibreria **ArCore** de Google.

7.5 Editor de text

Unreal engine suporta Microsoft Visual Studio com a Entorn de Desenvolupament Integrat (IDE) per defecte pels projectes en C++ de Windows, però, com es descriu a la secció 5.3.1, **Visual Studio Code** és un dels editors de text de propòsit general més capaç d'integrar-se amb qualsevol tecnologia. Següent aquest gratuït, més lleuger en el consum de recursos, i multi-plataforma, **Visual Studio Code** és l'editor de text escollit per desenvolupar la part de programació de codi del projecte.

Capítol 8

Anàlisi, disseny i implementació del sistema

8.1 Anàlisi del sistema

8.1.1 Diagrama de casos d'us

8.1.1.1 Menú d'inici, Pausa, Fi Partida

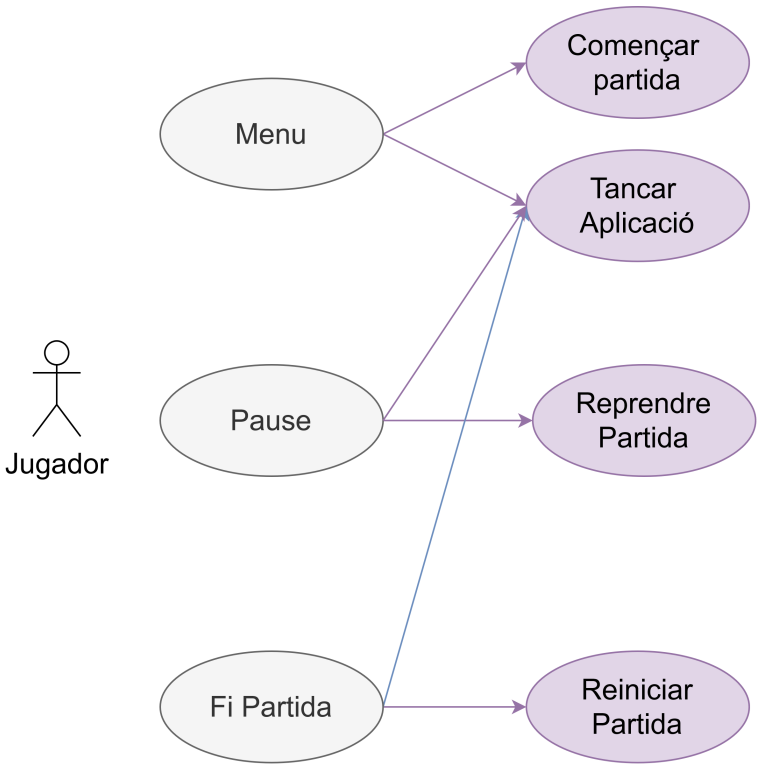

FIGURA 8.1: Diagrama de casos d'us Menú, Pausa i Fi de partida

8.1.1.2 *In Game*

FIGURA 8.2: Diagrama de casos d'us jugador en partida

8.2 Disseny i implementació del sistema

Nota: Els atributs, mètodes, esdeveniments, etc definits en c++ estaran dins una caixa amb ressaltat de sintaxis per a c++ amb un format com el següent:

```
// attribute example
UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Components", meta =
↪  (AllowPrivateAccess = "true"))
typeName varName;

//event example
UFUNCTION(BlueprintCallable, BlueprintImplementableEvent)
void SetWarningLight(bool IsRight);

//method example
bool methodName(bool in, bool out);

```

I les variables definides al Blueprint seguiran el següent format:

VarName(type)→ descripció

Els actors i components d'Unreal Engine compten de dos mètodes presents en totes les classes que n'hereten: **BeginPlay()**, que es crida quan es col·loca una instància d'aquesta classe al nivell i el nivell comença a jugar-se, o bé quan una instància d'aquesta classe es genera mentre es juga el joc, i **Tick()**, que, en cas que la classe el tingui activat, s'executa en cada *Frame* mentre es juga el nivell i des que la instància ha estat generada.

Hi ha classes que s'expliquen a continuació que compten amb aquests mètodes, però no han estat modificats i per tant, no s'explicaran.

8.2.1 Menú d'inici, Pausa i Fi de partida

Menú, Pausa i Fi de partida son les tres pantalles que trobem en el videojoc. Veure Figures 8.3, 8.4 i 8.5 .

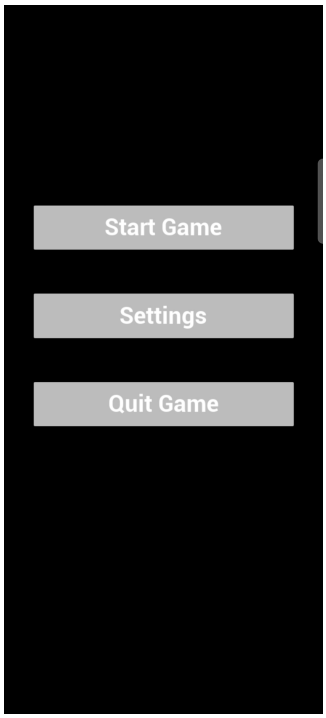

FIGURA 8.3: Menú d'inici

FIGURA 8.4: Menú de pausa

FIGURA 8.5: Menú de fi de partida

8.2.1.1 Menú d'inici

8.2.1.1.1 OnClicked(StartGame)

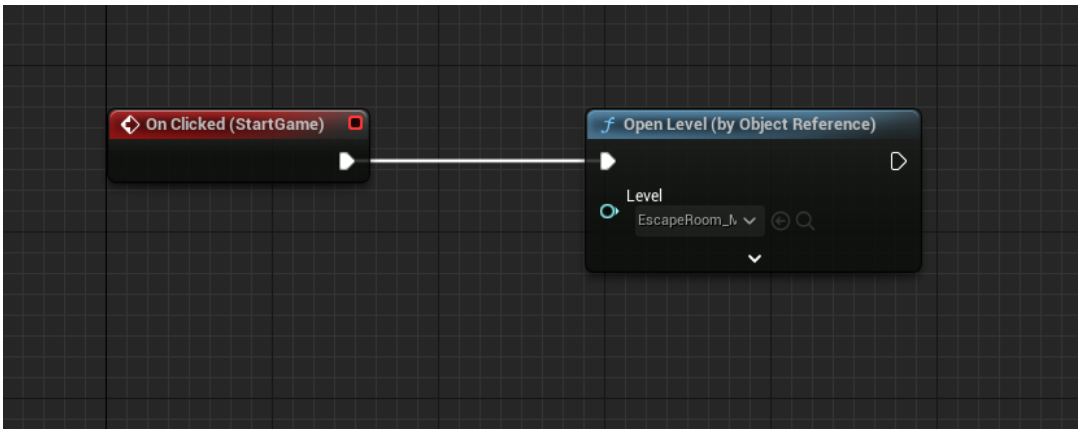

FIGURA 8.6: Implementació del Blueprint OnClicked(StartGame)

Esdeveniment que s'executa quan es clica el botó del menú de començar. Obre el mapa principal **EscapeRoom_Map**. Veure Figura 8.6

8.2.1.1.2 OnClick(QuitButton)

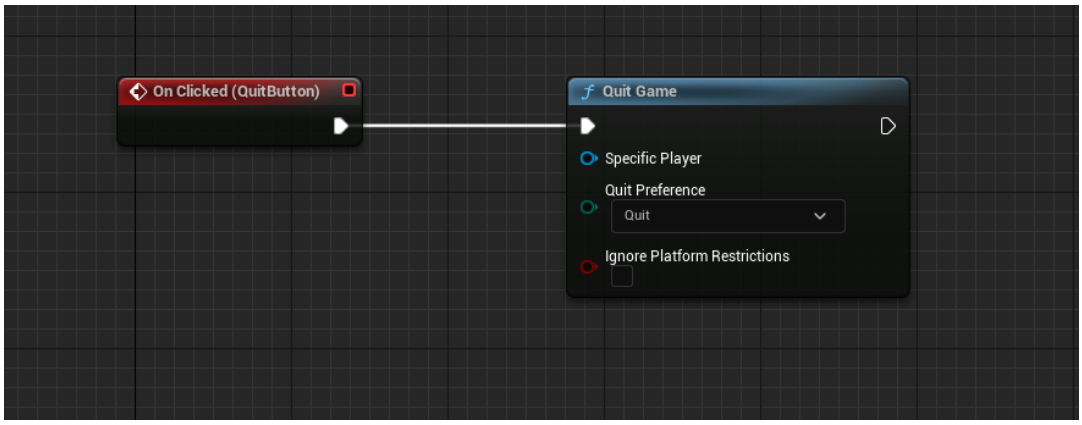

FIGURA 8.7: Implementació del Blueprint OnClicked(StartGame)

Esdeveniment que s'executa quan es clica el botó de sortir de la partida. Tanca l'aplicació amb la funció **Quit Game**. Veure Figura 8.7

8.2.1.2 Pausa

8.2.1.2.1 OnClick(Resume)

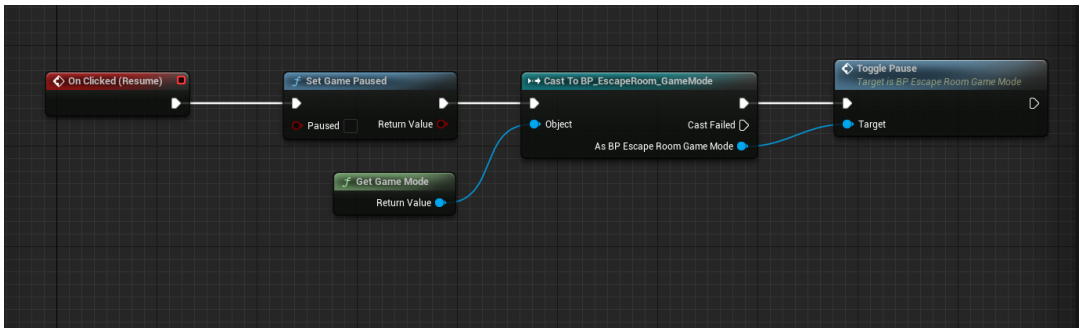

FIGURA 8.8: Implementació del Blueprint OnClick(Resume)

Esdeveniment que s'executa quan es clica el botó de reprendre la partida. Assigna l'estat de pausa del joc a false i crida el mètode del **GameMode Toggle Pause**, que elimina el *Widget* de la pantalla. Veure Figura 8.8

OnClick(QuitButton)

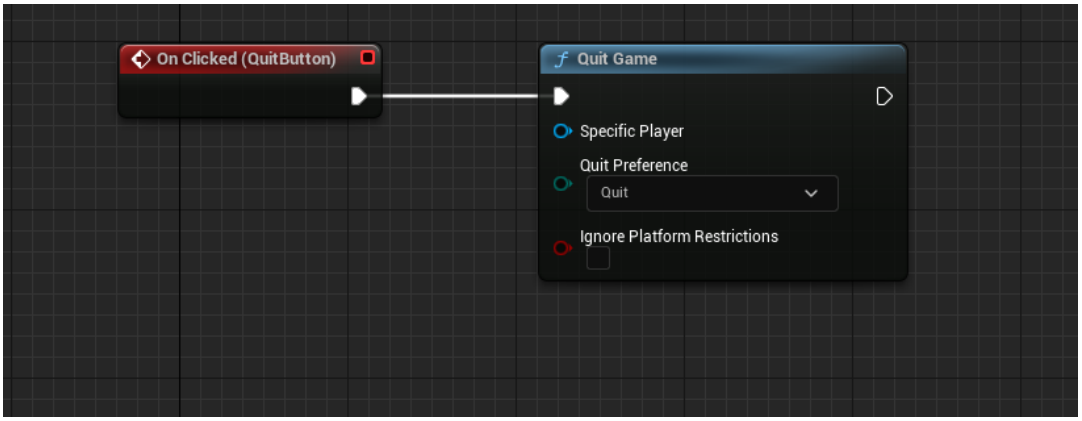

FIGURA 8.9: Implementació del Blueprint OnClick(QuitButton)

Esdeveniment que s'executa quan es clica el botó de sortir de la partida. Tanca l'aplicació amb la funció **Quit Game**. Veure Figura 8.9

8.2.1.3 Fi de partida

OnClick(Restart)

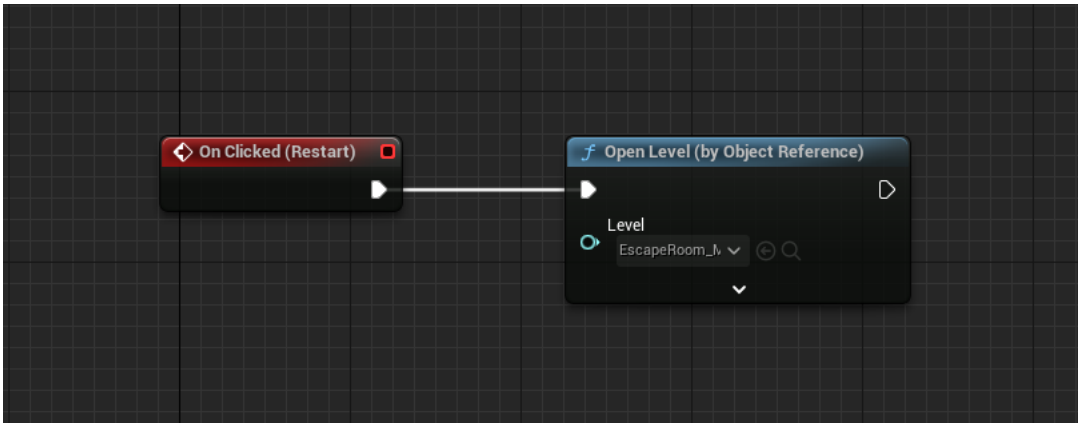

FIGURA 8.10: Implementació Blueprint OnClick(Restart)

Esdeveniment que s'executa quan es clica el botó de reiniciar la partida. Obre el mapa principal **EscapeRoom_Map**. Veure Figura 8.10

8.2.1.3.1 OnClick(QuitButton)

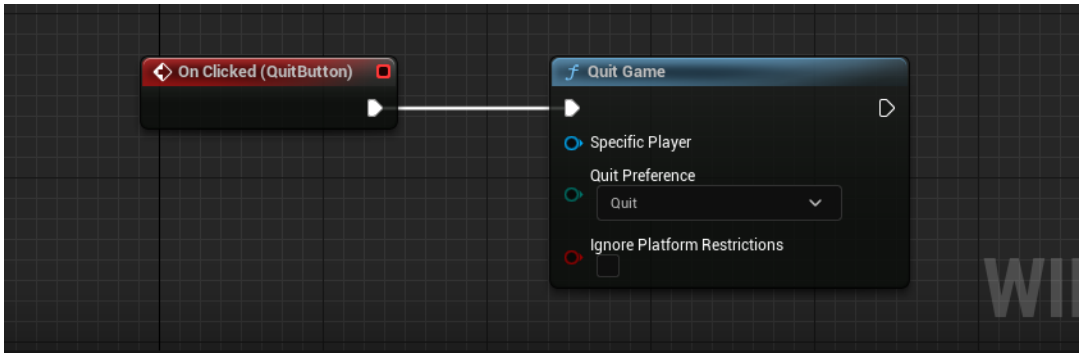

FIGURA 8.11: Implementació del Blueprint OnClick(QuitButton)

Esdeveniment que s'executa quan es clica el botó de sortir de la partida. Tanca l'aplicació amb la funció **Quit Game**. Veure Figura 8.11

8.2.2 HUD

El HUD del videojoc està format pel següent:

- **Botó flaix virtual**: per activar/desactivar el flaix virtual
- **Botó brúixola**: per mostrar la brúixola
- **Botó d'ajuda**: per sol·licitar una pista
- **Botó inventari**: per mostrar l'inventari
- **Element d'inventari**: per mostrar l'element d'inventari seleccionat actual
- **Pausa**: per fer pausa durant la partida

La representació gràfica del HUD es pot veure a la Figura 8.12.

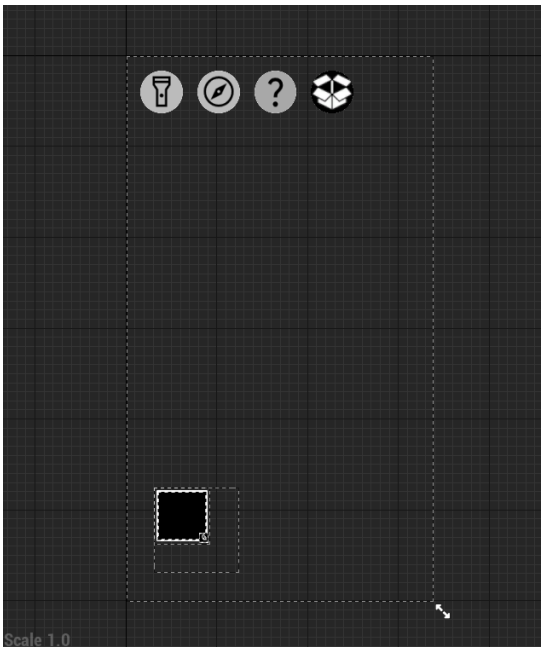

FIGURA 8.12: HUD

8.2.3 Estat de la partida

En el *framework* d'*Unreal Engine* el *Game Mode* és la classe principal per manejar informació sobre la partida que s'està jugant o l'estat d'aquesta.

En aquest projecte, s'implementa la classe ***EscapeRoom_GameMode***. És una classe que hereta de la classe ***AGameModeBase*** i és la classe principal de recepció d'esdeveniment relacionats amb l'estat de la partida, així com de mantenir la informació del nivell actual, l'aparició d'elements virtuals a la detecció d'*Image Candidates* a través de la càmera i el control d'aquests, és propietari de l'inventari, i administrador del HUD.

A més, implementa la interfície **I_EscapeRoom_Dialog**, ja que aquesta classe és també l'encarregada de donar les pistes personalitzades segons l'estat de la partida quan no s'està apuntant a cap nivell.

8.2.3.1 EscapeRoom_GameMode

8.2.3.1.1 Atributs

```
1 UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Config", meta =
  ↪  (AllowPrivateAccess = "true"))
2 class UARSessionConfig* EscapeRoom_SessionConfig;
3
```

Data Asset que defineix la configuració de la sessió de realitat augmentada. És on definirem els *Image Candidates* que detectarà la sessió.

```
1 UPROPERTY(VisibleAnywhere, Category = "Variables")
2 TMap<class UARCandidateImage*, class AEscapeRoom_Spawner*> SpawnedImages;
3
```

Mapa amb un punter als actors que han estat *spawned*, per haver estat detectat en algun moment la imatge que el relaciona amb el mon real. S'utilitza per no duplicar actors i per amagar-los un cop la imatge real no estan a la vista del jugador.

```
1 UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Variables",  meta =
  ↪  (AllowPrivateAccess = "true"))
2 int32 CurrentLevel;
3
```

Atribut per mantenir el nivell (trencaclosques) actual de la partida.

```
1 UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Components", meta =
  ↪  (AllowPrivateAccess = "true"))
2 class UEscapeRoom_CardinalComponent* CompassComponent;
3
```

Punter al component **UEscapeRoom_CardinalComponent**, amb les funcions per obtenir informació dels punts cardinals.

```
UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Components", meta =
 ↪  (AllowPrivateAccess = "true"))
class UEscapeRoom_InventorySystem* InventoryComponent;

```

Punter a l'objecte inventari de la partida.

Un punt a tenir en compte és que, com s'ha comentat, el GameMode és responsable de l'estat de la partida. En cas d'algun dia implementar el mode multi-jugador en el videojoc, caldria derivar aquesta responsabilitat al PlayerController, ja que estaríem parlant d'un "estat"personalitzat a cada jugador, i no de la partida. En aquest cas concret, cada jugador tindria el seu propi inventari.

8.2.3.1.2 Atributs del Blueprint

darkScreen (WBP Dark Screen)→ Referència al *widget* encarregat d'enfosquir la pantalla per simular la llum apagada del mon virtual.

Hints (Array of Strings)→ Llistat de les pistes, amb cada index corresponent al nivell del qual és la pista.

HintUI (WBP Hint)→ Referència al *widget* utilitzat per mostrar per pantalla una pista.

CompassUI (WBP Compass)→ Referència al *widget* encarregat de mostrar la brúixola.

InputCode_1 (WBP Input Code)→ Referència al *widget* encarregat de mostrar la interfície per entrar el codi per superar un dels nivells.

Inventory (W Escape Room Player Menu)→ Referència al *widget* d'Inventari.

HUD UI (WBP UI)→ Referència al HUD del videojoc.

8.2.3.1.3 Mètodes i Esdeveniments

Constructor

```
//Constructor
AEscapeRoom_GameMode();

```

```
AEscapeRoom_GameMode::AEscapeRoom_GameMode()
{
    PrimaryActorTick.bStartWithTickEnabled = true;
    PrimaryActorTick.bCanEverTick = true;
   CurrentLevel = 0;

   CompassComponent = CreateDefaultSubobject<UEscapeRoom_CardinalComponent>(TEXT("Compass
    ↪ Component"));
   InventoryComponent =
    ↪ CreateDefaultSubobject<UEscapeRoom_InventorySystem>(TEXT("Inventory Component"));

}

```

Constructor de la classe. Inicialitza el nivell inicial de la parrtida i crea els subobjectes de classe **UEscapeRoom_CardinalComponent** i **UEscapeRoom_InventorySystem**.

BeginPlay

```
virtual void BeginPlay() override;

```

```
    void AEscapeRoom_GameMode::BeginPlay()
{
    Super::BeginPlay();
    UARBlueprintLibrary::StartARSession(EscapeRoom_SessionConfig);
    UKismetSystemLibrary::ControlScreensaver(false);
}

```

Funció que es crida quan es comença la partida. Inicia la sessió de realitat Augmentada, passant la configuració definida a **EscapeRoom_SessionConfig**.

Tick

```
virtual void Tick(float DeltaTime) override;

```

```cpp
void AEscapeRoom_GameMode::Tick(float DeltaTime)
{

   Super::Tick(DeltaTime);

   TArray<UARTrackedImage*> TrackedImages =
↪  UARBlueprintLibrary::GetAllGeometriesByClass<UARTrackedImage>();
   GEngine->AddOnScreenDebugMessage(
      -1,
      GWorld->DeltaTimeSeconds,
      FColor::Red,
      FString::Printf(TEXT("Collision component INDEX: \%i"), TrackedImages.Num()));

   for (UARTrackedImage* i : TrackedImages) {

      if(i->GetTrackingState() == EARTrackingState::NotTracking){
         AEscapeRoom_Spawner** aux_actor = SpawnedImages.Find(i->GetDetectedImage());
         if(aux_actor != nullptr)
         {
            (*aux_actor)->GetRootComponent()->SetHiddenInGame(true, true);
         }
      }
      else {
         UARBlueprintLibrary::DebugDrawTrackedGeometry(
                i,
                this,
                FColor::Blue,
                20.f,
                0.f
            );

         UARCandidateImage* DetectedImage = i->GetDetectedImage();
         FTransform TrackingTransform = i->GetLocalToTrackingTransform();

         AEscapeRoom_Spawner** ActualActor = SpawnedImages.Find(DetectedImage);

         // Spawnegem un nou actor
         if(!ActualActor)
         {
            AEscapeRoom_Spawner* NewSpawnedActor =
↪  GetWorld()->SpawnActor<AEscapeRoom_Spawner>(SpawnerClass, TrackingTransform);
            if(NewSpawnedActor)
            {
               NewSpawnedActor->SetOwner(this);
               SpawnedImages.Add(DetectedImage, NewSpawnedActor);
               NewSpawnedActor->SetChildActor(DetectedImage);
               NewSpawnedActor->AttachToComponent(
                        RootComponent,
                        FAttachmentTransformRules::KeepRelativeTransform
                    );

               ActualActor = &NewSpawnedActor;
               SetActorLevel(CurrentLevel, NewSpawnedActor->GetSpawnedActor());
            }

         }
         // transformem un actor ja spawnejat
         else {
            AEscapeRoom_Spawner** aux_actor = SpawnedImages.Find(i->GetDetectedImage());
            if(aux_actor != nullptr)
            {
               (*aux_actor)->GetRootComponent()->SetHiddenInGame(false, true);
            }

            if(ActualActor && !((*ActualActor)->ChildActorHasTag("Pin")))
            {
                (*ActualActor)->SetActorTransform(TrackingTransform, true);
            }
         }
      }
   }
}

```

El **Tick** és el mètode que implementa la lògica principal per a la detecció d'imatges i l'*spawn* dels elements virtuals a la partida. En aquesta classe és de les funcions més importants ja que s'encarrega de crear tots els elements de realitat augmentada que s'utilitzen a la partida.

La classe *ARBlueprintLibrary* del modul *AugmentedReality Unreal Engine* compta amb el mètode ***GetAllGeometriesByClass***, que donada una classe de *UARTrackedGeometry*, detecta totes les geometries a través de la càmera. En el cas del projecte, les geometries a detectar són les

targetes: *UARTrackedImage*.

En cada *frame* del videojoc, l'algorisme obté una aquesta llista d'imatges detectades per la càmera i, per cadascuna, en genera l'actor virtual associat a través de la classe *EscapeRoom_Spawner* o li actualitza la posició en cas que ja estigui creat.

Aprofita l'execució per *frame* per detectar i ocultar aquells objectes que prèviament han estat generats però la targeta associada ja no està a la vista del jugador, o per tornar a mostrar objectes virtuals ocults i que tornen a estar a la vista del jugador.

ToggleGlobalLight

```
1 UFUNCTION(BlueprintCallable, BlueprintImplementableEvent)
2 void ToggleGlobalLight();
3
```

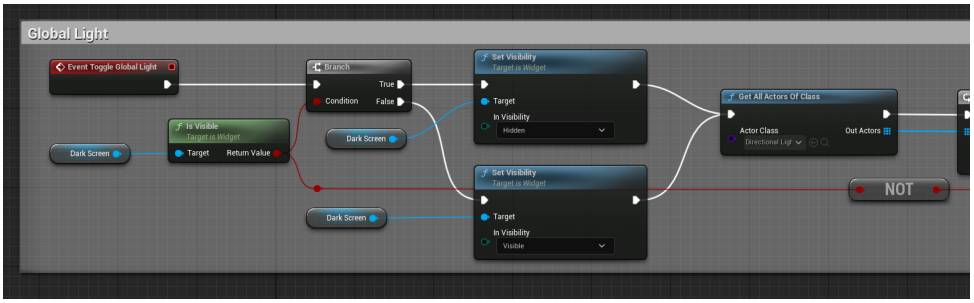

FIGURA 8.13: Implementació del Blueprint ToggleGlobalLight Part 1

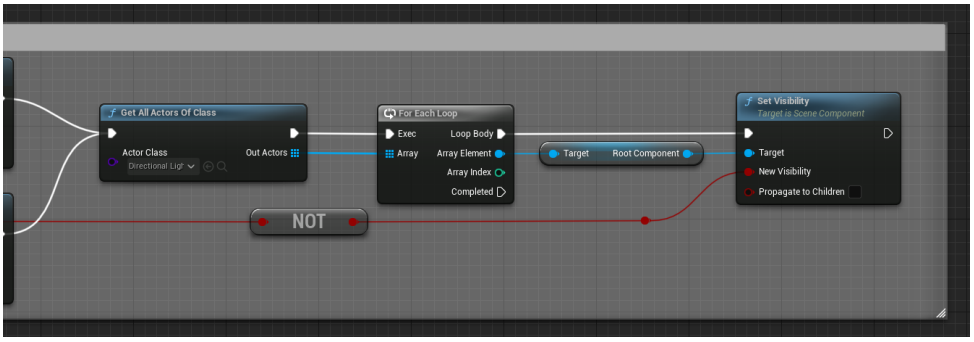

FIGURA 8.14: Implementació del Blueprint ToggleGlobalLight Part 2

Esdeveniment encarregat d'encendre i apagar la llum del mon virtual a la partida alternant la visibilitat dels actors de tipus **DirectionalLight**. També alterna la visibilitat del **WBP_DarkScreen**, que fa un efecte de foscor i claror. Veure Figures 8.13 i 8.14 per implementació. Veure Figures 8.15 i 8.16 per resultats.

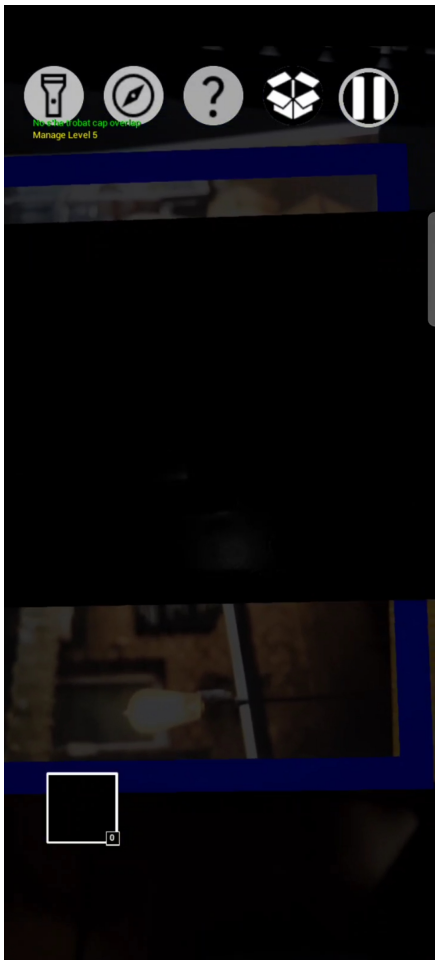

FIGURA 8.15: Efecte llum global apagada

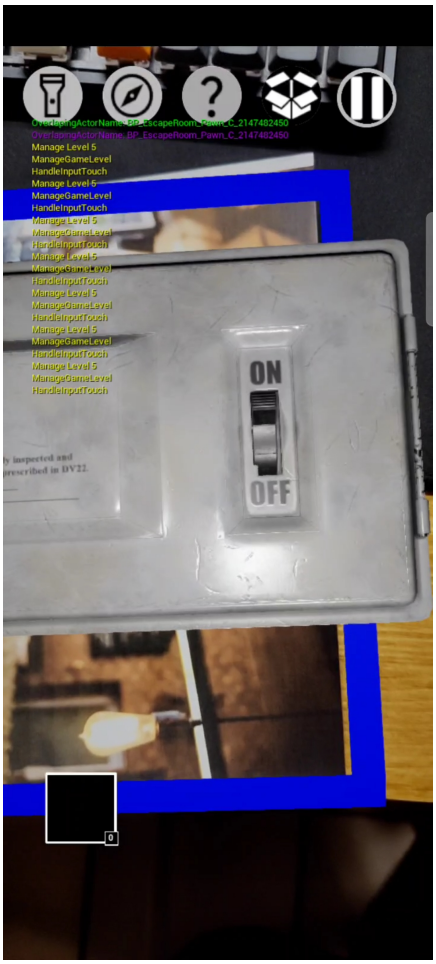

FIGURA 8.16: Efecte llum global encesa

ToggleCompass

```
1 UFUNCTION(BlueprintCallable, BlueprintImplementableEvent)
2 void ToggleCompass();
3
```

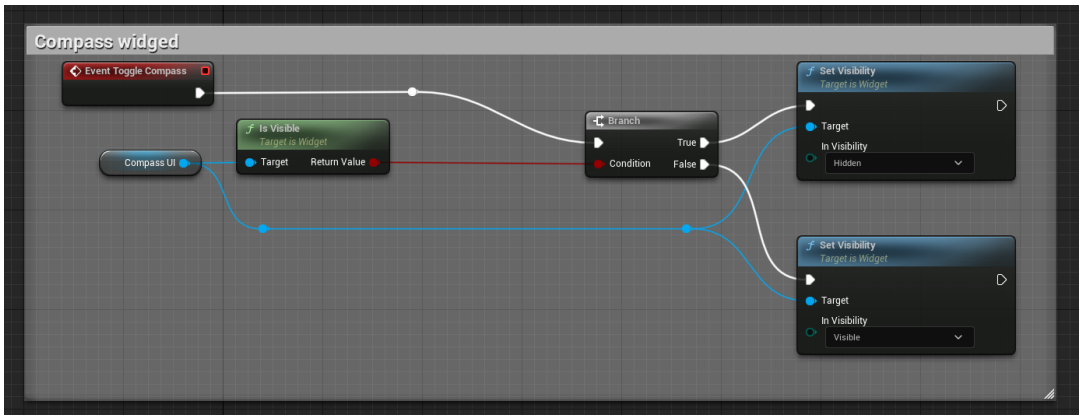

FIGURA 8.17: Implementació del Blueprint ToggleCompas

Esdeveniment encarregat d'alternar la vista amb la brúixola comprovant la visibilitat actual de l'atribut **CompassUI**. Veure Figura 8.17

CloseInputCodeWidget

```
UFUNCTION(BlueprintCallable, BlueprintImplementableEvent)
void OpenInputCodeWitget();

```

```
UFUNCTION(BlueprintCallable, BlueprintImplementableEvent)
void CloseInputCodeWitget();

```

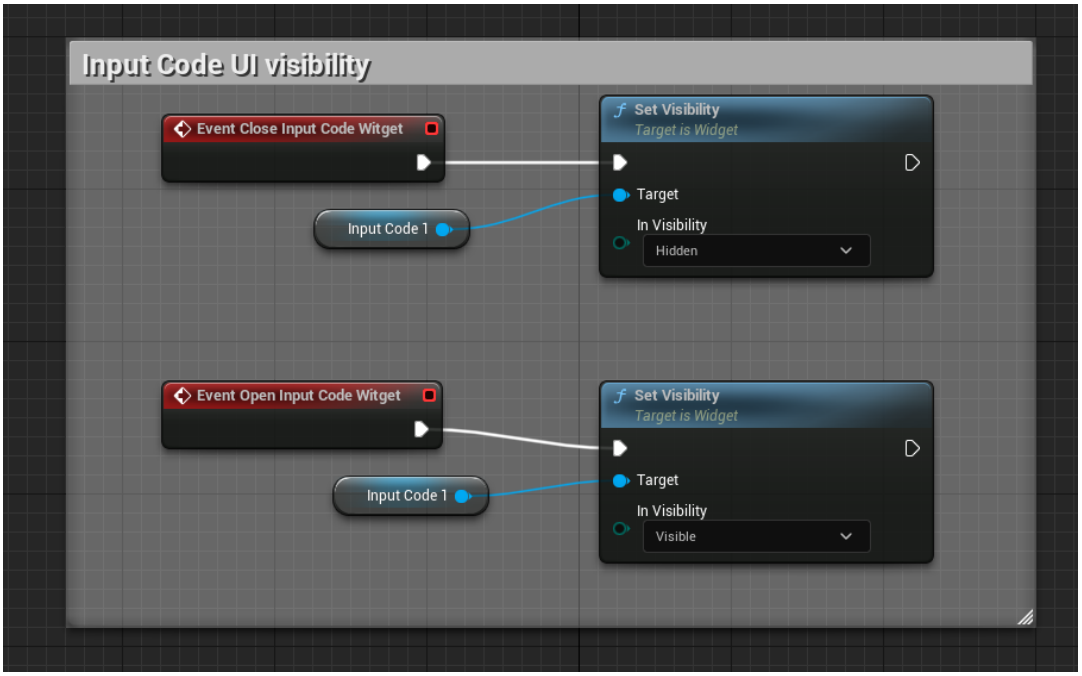

FIGURA 8.18: Implementació dels Blueprints OpenInputCodeWidget() i CloseInputCodeWidget()

Esdeveniments encarregats de de mostrar i ocultar la vista per entrar el codi resolutor del nivell 1. Veure Figura 8.18

CloseInventory/ShowInventory

```
UFUNCTION(BlueprintCallable, BlueprintImplementableEvent)
void CloseInventory();

```

```
UFUNCTION(BlueprintCallable, BlueprintImplementableEvent)
void ShowInventory();

```

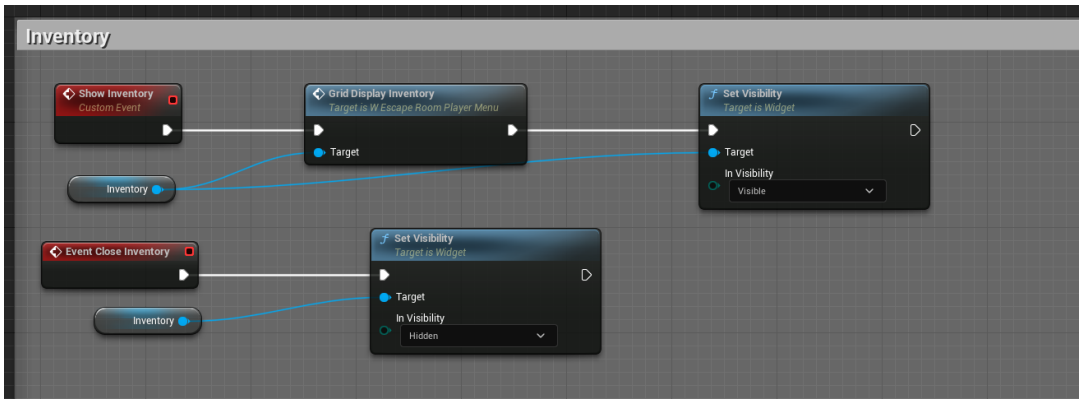

FIGURA 8.19: Implementació dels Blueprints CloseInventory() i ShowInventory()

Esdeveniments encarregats d'obrir i tancar l'inventari assignant la visibilitat a l'atribut **Inventory**. Veure Figura 8.19

AddItemToInventory

```
UFUNCTION(BlueprintCallable)
void AddItemToInventory(FName ItemID, int Quantity);

```

```
void AEscapeRoom_GameMode::AddItemToInventory(FName ItemID, int Quantity)
{
   InventoryComponent->AddToInventory(ItemID, Quantity);
}

```

Mètode que es crida quan es clica un actor de la classe ***EscapeRoom_InventoryItem*** amb nom `ItemID` i que n'afageix `Quantity` elements a l'inventari del qual manté l'estat.

SelectInventoryItem

```
UFUNCTION(BlueprintCallable, BlueprintImplementableEvent)
void SelectInventoryItem(FName ItemID);

```

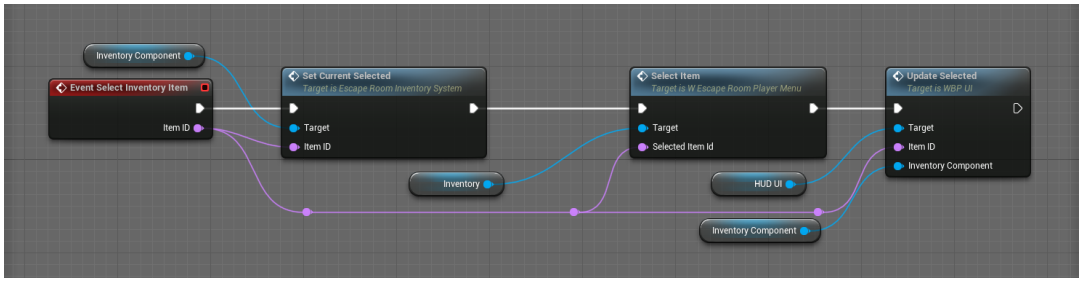

FIGURA 8.20: Implementació del Blueprint SelectInventoryItem(ItemID)

Esdeveniment que marca com a seleccionat l'article d'inventari amb nom `ItemID`. També actualitza l'article que es mostra de manera permanent a la pantalla com a actual seleccionat. Veure Figura 8.20

UseItemInventory

```
UFUNCTION(BlueprintCallable)
void UseItemInventory(FName ItemID);

```

```
void AEscapeRoom_GameMode::UseItemInventory(FName ItemID)
{
   InventoryComponent->RemoveFromInventory(ItemID);
}

```

Esdeveniment que es crida quan l'article seleccionat de l'inventari és utilitzat. Es resta una unitat de l'article a l'inventari.

ManageGameLevel

```
UFUNCTION(BlueprintCallable)
void ManageGameLevel(int32 Level);

```

```
void AEscapeRoom_GameMode::ManageGameLevel(int32 Level)
{
   CurrentLevel = Level;
   for (auto& Elem : SpawnedImages)
   {
      UChildActorComponent* a = Elem.Value->GetSpawnedActor();
      SetActorLevel(CurrentLevel, a);
   }
}

```

Funció que es crida des de la classe *EscapeRoom_GameMode* cada cop que hi ha un canvi de nivell.

La classe *EscapeRoom_GameMode* és coneixedora de tots els actors virtuals que s'han generat a la partida, i cada ***EscapeRoom_Level*** té registrar quin nivell és ell mateix i quan es supera. D'aquesta manera, s'indica a cada element virtual generat a la partida quin és el nou nivell actual.

SetActorLevel

```
bool SetActorLevel(int32 Level, UChildActorComponent* actor);

```

```
bool AEscapeRoom_GameMode::SetActorLevel(int32 Level, UChildActorComponent* actor)
{
   if(actor != nullptr )
   {
      if(AEscapeRoom_Level* actorLevel = Cast<AEscapeRoom_Level>(actor->GetChildActor()))
      {
         actorLevel->ManageLevel(Level);
         return true;
      }
   }
   return false;
}

```

Es fa un cast a EscapeRoom_Level de l'actor al que apunta el punter que s'ha passat per paràmetre, i, si aquest és un EscapeRoom_Level, es crida el seu mètode **ManageLevel**, passant-li per paràmetre el nou nivell de la partida.

8.2.4 Sistema d'*Spawn* d'Actors

La classe ***AEscapeRoom_Spawner*** és l'encarregada de, donada una imatge detectada per l'*EscapeRoom_GameMode*, crear l'element a la partida.

Per una millora en el manteniment i implementació de nous nivells de futures versions, i més flexibilitat en la interacció del jugador amb els actors de la partida, les funcions d'aquesta classe podrien ser implementades en el propi GameMode o en cas de tenir mode multijugador, el PlayerController.

8.2.4.1 EscapeRoom_Spawner

8.2.4.1.1 Atributs

```
UPROPERTY(VisibleAnywhere, BlueprintReadWrite, Category = "Components", meta =
 ↪  (AllowPrivateAccess = "true"))
UChildActorComponent* ChildActor;

```

El ChildActor és l'actor amb el tipus associat a la imatge detectada.

```
UPROPERTY(EditAnywhere, Category = "Variables")
TMap<class UARCandidateImage*,UClass*> SpawnableImages;

```

Mapa amb clau *UARCandidateImage*, valor *UClass*. L'*Image Candidate* fa referència a la imatge detectada, i el *UClass* fa referència al tipus de classe al que es relaciona la imatge i per tant, a quin objecte es generarà segons la imatge detectada.

8.2.4.1.2 Mètodes

SetChildActor

```
void SetChildActor(class UARCandidateImage* DetectedImage);

```

```
void AEscapeRoom_Spawner::SetChildActor(UARCandidateImage* DetectedImage)
{
   UClass** CandidateActorClass = SpawnableImages.Find(DetectedImage);

   if(CandidateActorClass)
   {
      ChildActor->SetChildActorClass((*CandidateActorClass));
   }

}

```

A partir de la imatge detectada `DetectedImage` de tipus UARCandidateImage, crea un objecte del tipus amb que es relaciona l'ImageCandidate en l'atribut TMap<class UARCandidateImage*,UClass*> SpawnableImages.

8.2.5 *Pawn*

El pawn és la representació del jugador en el videojoc. En aquest projecte, el jugador té una representació física al mon real. És per això que el Pawn serà sempre els *ulls* del jugador, en el nostre cas, el dispositiu de realitat augmentada.

El pawn és també l'encarregat d'implementar la lògica de les mecàniques que fan relació amb la posició, com per exemple la il·luminació d'elements virtuals al apropar el flaix o la detecció de col·lisions d'elements virtuals amb la vista del jugador.

8.2.5.1 EscapeRoom_Pawn

8.2.5.1.1 Atributs

```
UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Components", meta =
↪  (AllowPrivateAccess = "true"))
class UCameraComponent* Camera;

```

Representa el punt de vista de la càmera i el camp de visió, i està col·locat al centre, fent que la càmera del mòbil representi els ulls del jugador.

```
UPROPERTY(VisibleAnywhere, Category = "Components", meta = (AllowPrivateAccess = "true"))
class USpotLightComponent* SpotFlashlight;

```

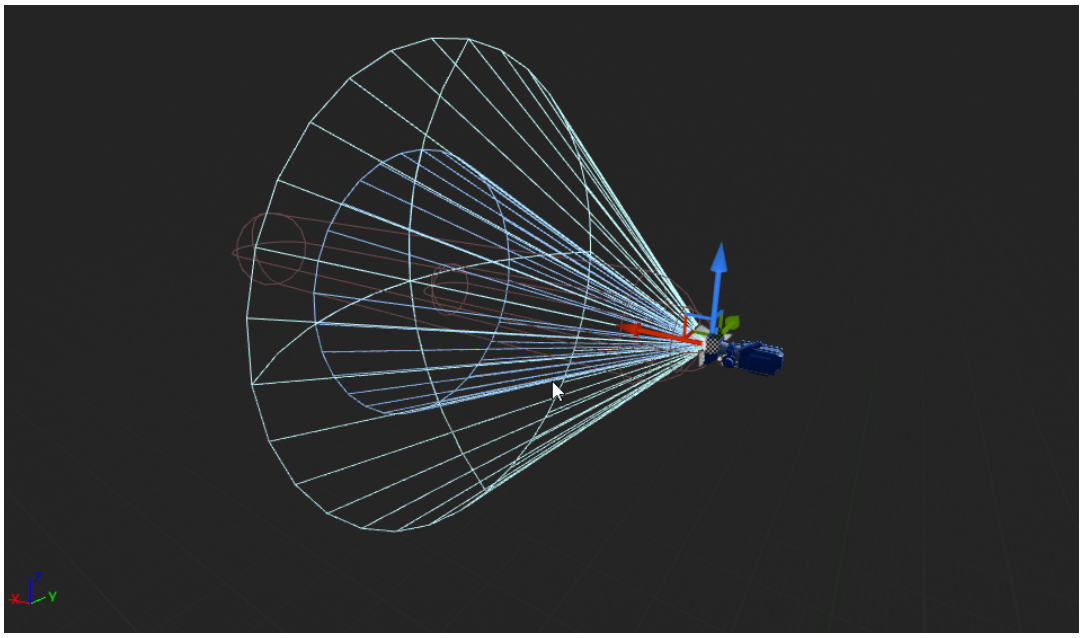

FIGURA 8.21: Component del Blueprint SpotFlashlight

Llum situada a la càmera, conceptualment és un flaix del propi dispositiu amb el que caldrà il·luminar algun objecte virtual durant la partida. Veure Figura 8.21

```
1 UPROPERTY(VisibleAnywhere, Category = "Components", meta = (AllowPrivateAccess = "true"))
2 class UCapsuleComponent* FlashlightCollisionComponent;
3
```

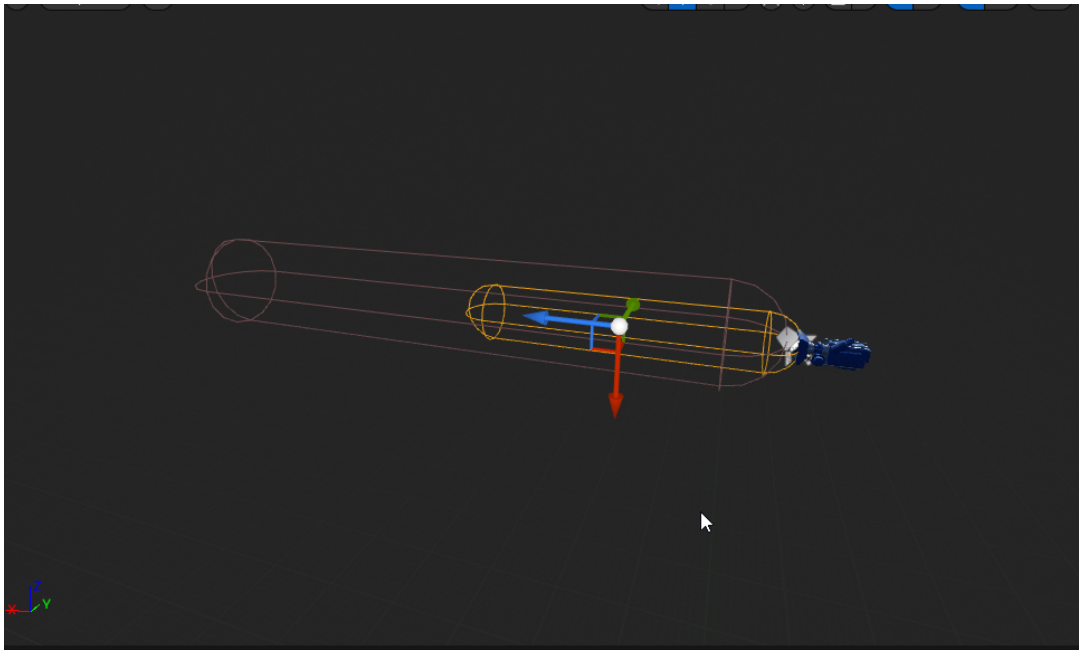

FIGURA 8.22: Component del Blueprint FlashLightCollisionBox

CapsuleComponent per detectar quins objectes de la partida estan essent il·luminats pel flaix. Veure Figura 8.22

```
UPROPERTY(BlueprintReadWrite, VisibleAnywhere, Category = "Components", meta =
↪  (AllowPrivateAccess = "true"))
class UCapsuleComponent* DialogCollisionComponent;

```

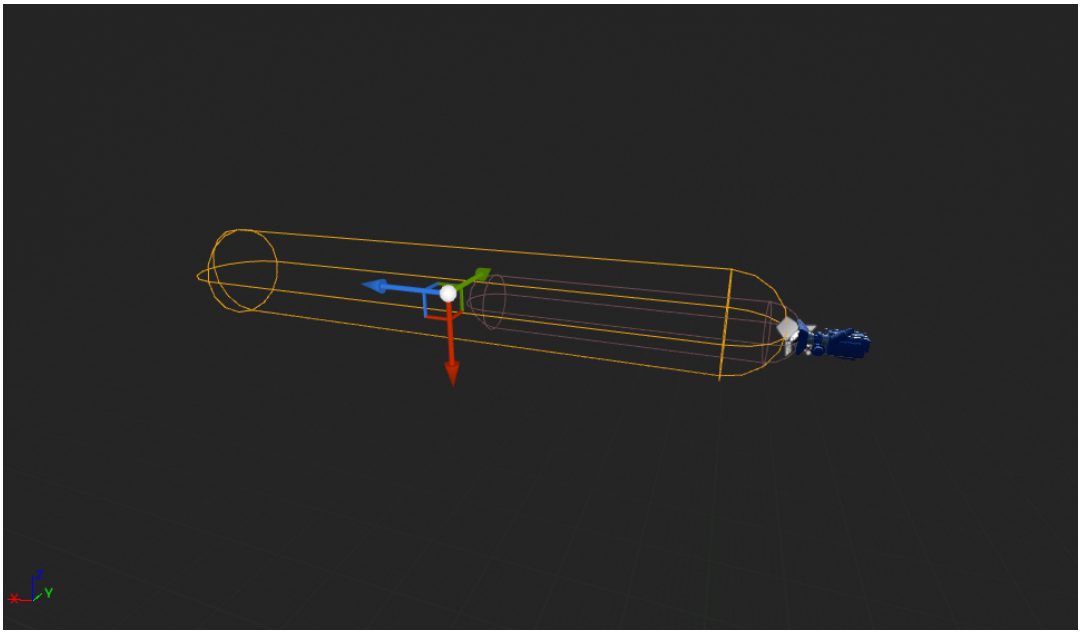

FIGURA 8.23: Component del Blueprint DialogCollisionComponent

CapsuleComponent per detectar quins objectes de la partida estan essent apuntats pel jugador i així poder oferir una pista personalitzada en cas que s'estigui enfocant a un nivell en concret. Veure Figura 8.23.

8.2.5.1.2 Mètodes i Esdeveniment

ToggleFlashLight

```
UFUNCTION(BlueprintCallable, Category="Light")
bool ToggleFlashLight();

```

```
bool AEscapeRoom_Pawn::ToggleFlashLight()
{
   //TODO: sound efects
   GEngine->AddOnScreenDebugMessage(-1, GWorld->DeltaTimeSeconds, FColor::Green,
↪   TEXT("ToggleFlashLight"));

   SpotFlashlight->ToggleVisibility();

   if(SpotFlashlight->GetVisibleFlag())
   {
      FlashlightCollisionComponent->SetCollisionEnabled(ECollisionEnabled::QueryAndPhysics);
   }
   else {
      FlashlightCollisionComponent->SetCollisionEnabled(ECollisionEnabled::NoCollision);
   }

   return  SpotFlashlight->GetVisibleFlag();

}

```

Alterna la visibilitat del flaix virtual, i activa o desactiva la detecció de colisions del `FlashlightCollisionCompon` depenent de la visibilitat del flaix.

OnHelpRequest

```
UFUNCTION(BlueprintCallable, BlueprintImplementableEvent, Category="Light")
void OnHelpRequest();

```

Esdeveniment que es crida quan es sol·licita ajuda. Detecta el primer element amb el que col·lisiona el component `DialogCollisionComponent` i, si aquest implementa la interfície *EscapeRoom_Dialog*, mostra l'ajuda del nivell en qüestió.

8.2.6 Sistema de Nivells

El videojoc està dissenyat de manera que en tot moment de la partida el jugador té els recursos disponibles, com un mon obert. Tot i això, hi ha un fil de partida definit, uns nivells concrets amb un ordre establert i una resolució concreta per cadascun.

Per satisfer aquest sistema, es dissenya la classe *AEscapeRoom_Level*, que implementa un comportament que tots els nivells han de tenir i de la que heretaran tots els nivells, els quals implementaran la lògica concreta del nivell en si.

8.2.6.1 EscapeRoom_Level

8.2.6.1.1 Atributs

```
UPROPERTY(BlueprintReadWrite, EditAnywhere, Category = "Components", meta =
↪  (AllowPrivateAccess = "true"))
bool IsActive;

```

True si el nivell actual és el nivell en qüestió, **false** altrament.

```
UPROPERTY(BlueprintReadWrite, EditAnywhere, Category = "Components", meta =
↪  (AllowPrivateAccess = "true"))
int32 Level;

```

Nivell que representa la propia classe.

```
UPROPERTY(BlueprintReadWrite, EditAnywhere, Category = "Components", meta =
↪  (AllowPrivateAccess = "true"))
TArray<FString> Hints;

```

Llista de pistes del nivell.

```
UPROPERTY(VisibleInstanceOnly, Category = "Widget", meta = (AllowPrivateAccess = "true"))
class UHintWidget* HintWidget;

```

Widget per mostrar les pistes del nivell.

8.2.6.1.2 Mètodes

ManageLevel

```
virtual bool ManageLevel(int32 currentLevel);

```

Mètode encarregat d'implementar la lògica referent al nivell, com activar o desactivar certes mecàniques. És cada classe hereditària qui fa *override* de la funció.

SetLevel

```
virtual void SetLevel(int32 newLevel);

```

Setter del nivell. Hi ha nivells tenen d'atributs objectes que també hereten d'*EscapeRoom_Level*. Per tant un objecte de classe *EscapeRoom_Level* no té perquè tenir nivell propi. Aquest mètode permet al nivell pare, que sí que té nivell propi, indicar-li quin és el nivell al què pertany.

SetHints

```
virtual void SetHints(int32 newLevel);

```

Mètode que, segons el nou nivell `newLevel`, assigna unes pistes o unes altres. Per exemple, si sol·licitem les pistes del nivell 3 però encara no hem solucionat l'1, ens podria retornar pistes del nivell 1 o simplement dir-nos que encara no estem preparats per aquest nivell. Cada classe hereditària implementa aquest mètode.

8.2.7 Nivell 0

La partida comença amb la pantalla més fosca del normal i sense que el jugador pugui interactuar amb res, simulant foscor en el mon virtual. En canvi, sí que es pot utilitzar els elements presents en el HUD, i el que necessitem en concret és el flaix.

El nivell consisteix en apropar el dispositiu a la targeta que genera una caixa de distribució elèctrica i fer clic a l'interruptor. En fer-ho, s'encén la llum de la partida, però caldrà que el flaix virtual estigui il·luminant la caixa elèctrica per poder activar l'interruptor. El nivell 0 es fa implementar prèviament al disseny del sistema de nivells, pel que no manté del tot l'estructura dels següents nivells. Veure Figures 8.24, 8.25 i 8.26.

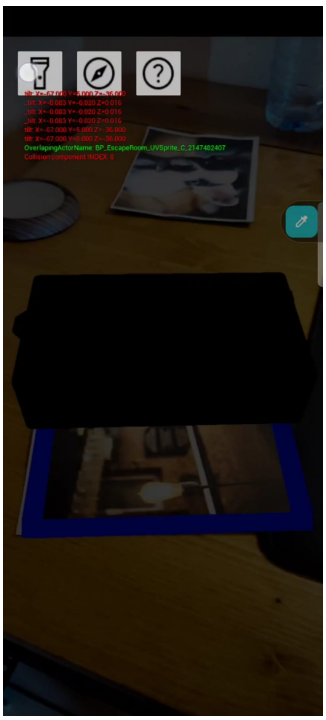

FIGURA 8.24: Nivell 0 amb el flaix apagat

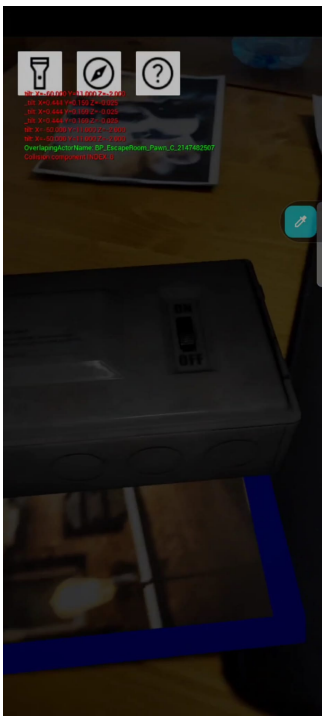

FIGURA 8.25: Nivell 0 amb el flaix encès

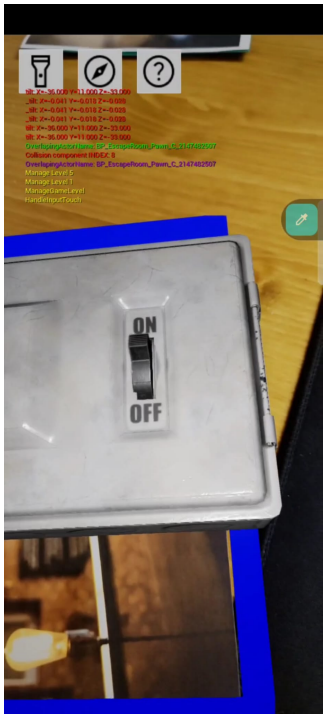

FIGURA 8.26: Nivell 0 amb la llum global encesa

8.2.7.1 Flaix Virtual

El flaix virtual s'implementa en el **EscapeRoom_Pawn** i està format per una capça de col·lisió de la distancia de la llum per tal que la caixa elèctrica detecti que està essent il·luminada, i per una **SpotLight** situada a la càmera del dispositiu.

El propi *Pawn* implementa la lògica per encendre i apagar el flaix, el mètode està explicat en l'apartat d'**EscapeRoom_Pawn**. Veure Figures 8.27 i 8.28

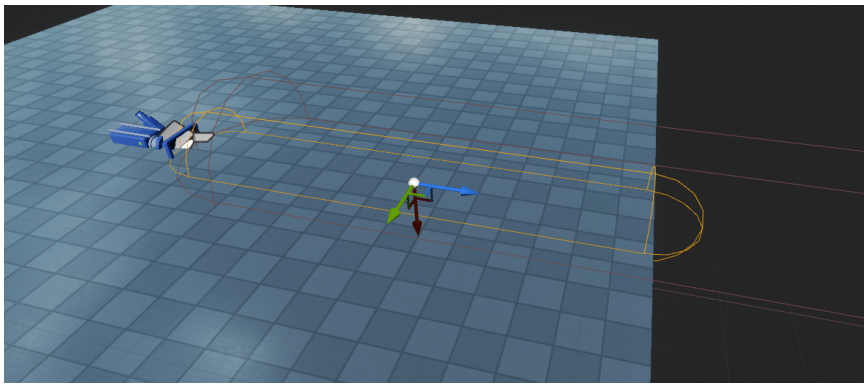

FIGURA 8.27: Capsa de col·lisió del flaix

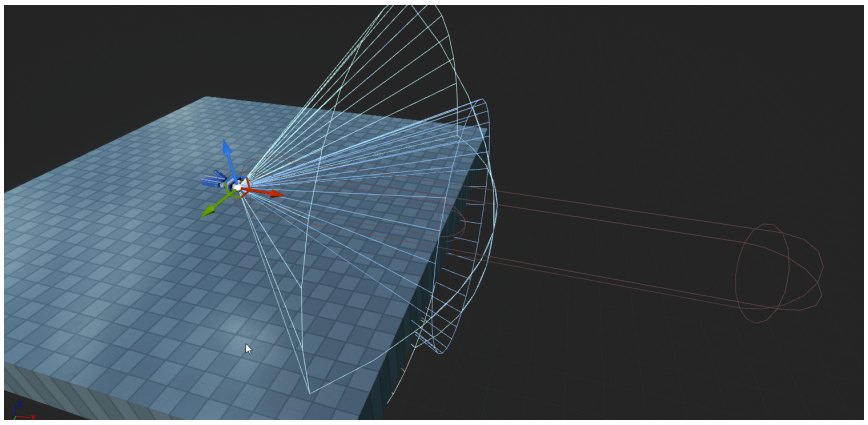

FIGURA 8.28: Llum SpotLight del flaix

8.2.7.2 Caixa elèctrica

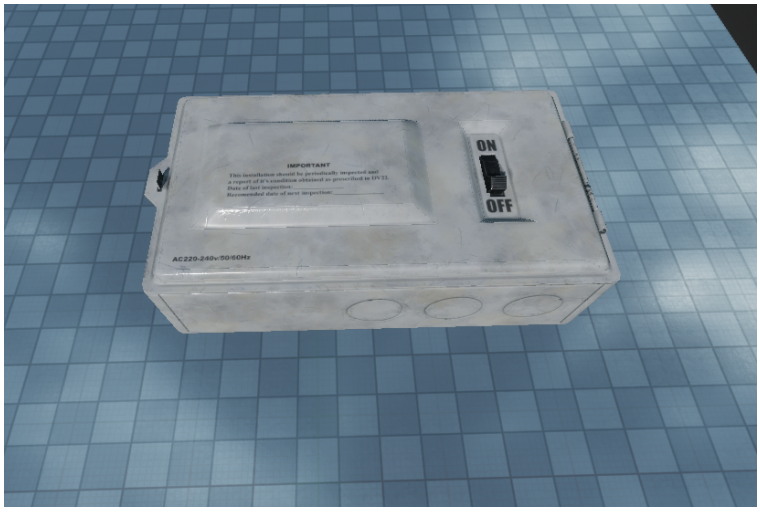

FIGURA 8.29: Caixa elèctrica

La caixa elèctrica està implementada per la classe **EscapeRoom_button**, i el mètode interessant pel nivell és el següent, que es crida quan es detecta el clic a l'interruptor. Veure Figura 8.29

HandleInputTouch

```
UFUNCTION(BlueprintCallable, Category = "Events")
void HandleInputTouch(USceneComponent* ButtonMesh);

```

```
void AEscapeRoom_Button::HandleInputTouch(USceneComponent *t)
{
   AEscapeRoom_GameMode *GameMode =
 ↪  Cast<AEscapeRoom_GameMode>(UGameplayStatics::GetGameMode(this));
   if(GameMode != nullptr)
   {
      GEngine->AddOnScreenDebugMessage(-1, 3, FColor::Yellow,
 ↪  FString::Printf(TEXT("HandleInputTouch")));
      GameMode->ManageGameLevel(1);
   }
    if(lightOn)
   {
      if(GameMode != nullptr)
      {
         GameMode->ToggleGlobalLight();
         lightOn = false;

      }
   }
    else {
      TArray<AActor*> Actors;
      GetOverlappingActors(Actors);
      for(AActor* Actor : Actors)
      {
         FString ActorName = Actor->GetActorNameOrLabel();
         GEngine->AddOnScreenDebugMessage(-1, 3.f, FColor::Purple,
 ↪  FString::Printf(TEXT("OverlapingActorName: \%s"), *ActorName));
         if(t->ComponentHasTag("pressable"))
         {
            if(GameMode != nullptr)
            {
               GameMode->ToggleGlobalLight();
               lightOn = true;
            }
         }
      }
   }

}

```

Aquest mètode guarda l'estat de la llum general en l'atribut **lightOn**(bool). En ser cridat, si la llum està engegada, crida l'esdeveniment del **GameMode** encarregat d'alternar la llum global i canvia l'estat de **lightOn**. En encendre per primer cop la llum, es considera superat el nivell i es notifica al **GameMode**. Si la llum està apagada, obté tots els actors que estan sobreposant-se i crida el mètode de **GameMode** que encendrà la llum.

La llum permet tornar a ser apagada perquè trobo interessant en el treball futur, que algun nivell impliqui tenir la llum apagada.

8.2.8 Nivell 1

El primer nivell consisteix en utilitzar una eina proporcionada per una targeta i que funciona com una llum ultra-violeta per revelar un codi present en un objecte virtual d'una altra targeta, obligant al jugador a moure les targetes, apropant-les i allunyant-les entre elles. Veure Figures 8.30 , 8.31 i 8.32

El jugador haurà de recordar aquest numero i els seus colors per entrar-lo en un *widget* d'entrada de codi que apareix en clicar l'element virtual del nivell 1 i que aquest quedi superat. Veure Figura 8.33.

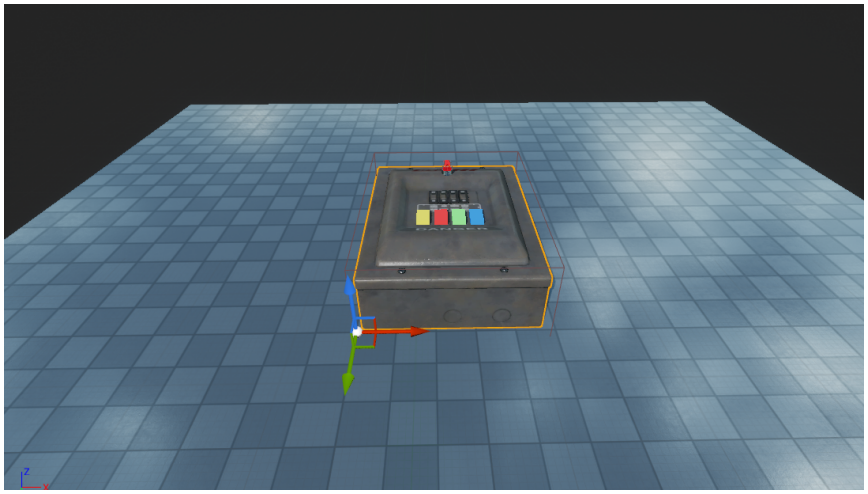

FIGURA 8.30: Figura base del Nivell 1

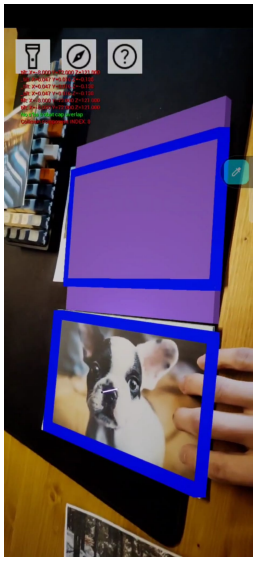

FIGURA 8.31: Codi secret no il·luminat per la llum ultra-violeta

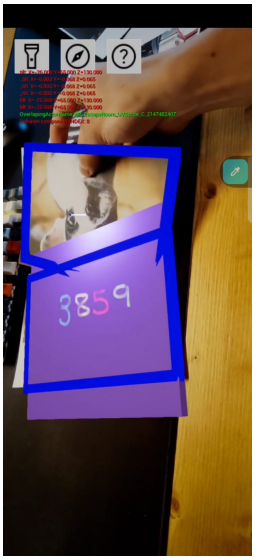

FIGURA 8.32: Codi secret il·luminat per la llum ultra-violeta

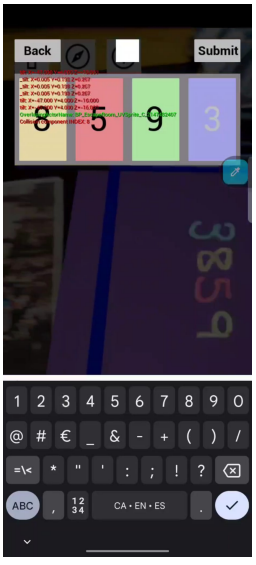

FIGURA 8.33: Entrada del codi secret al WBP_InputCode

8.2.8.1 EscapeRoom_Level_1

8.2.8.1.1 Atributs

```
UPROPERTY(EditDefaultsOnly, Category = "Components", meta = (AllowPrivateAccess = "true"))
UStaticMeshComponent* BaseMesh;

```

Figura principal del nivell.

```
UPROPERTY(EditDefaultsOnly, Category = "Components", meta = (AllowPrivateAccess = "true"))
USceneComponent* Scene;

```

Escena que dona posició al nivell.

8.2.8.1.2 Atributs del Blueprint

Cube_blue, Cube_green, Cube_yellow, Cube_red (StaticMesh)→ Conjunt de cubs de color per donar pistes de que s'ha d'entrar un codi i l'ordre dels números d'aquest per superar el nivell.

Light (StaticMesh)→ Figura en forma de bombeta de color vermell que pren color verd un cop es supera el nivell.

Box(BoxCollision)→ Capsa de col·lisió per detectar el clic al nivell i mostrar el *widget* d'entrada de codi.

8.2.8.1.3 Mètodes i Esdeveniments

Constructor

```
AEscapeRoom_Level_1();

```

```
AEscapeRoom_Level_1::AEscapeRoom_Level_1()
{
   // Set this actor to call Tick() every frame.  You can turn this off to improve
 ↪  performance if you don't need it.
   PrimaryActorTick.bCanEverTick = true;
   Level = 1;

   Scene = CreateDefaultSubobject<USceneComponent>(TEXT("Scene"));
   RootComponent = Scene;

   BaseMesh = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Base Mesh"));
   BaseMesh->SetupAttachment(Scene);

   //SpawnPoint = CreateDefaultSubobject<USceneComponent>(TEXT("Spawn Point"));
   //SpawnPoint->SetupAttachment(RootComponent);

   }

```

Constructor de la classe. Inicialitza l'atribut `Level`, `BaseMesh` i `Scene`.

ManageLevel

```
virtual bool ManageLevel(int32 NewLevel) override;

```

```
bool AEscapeRoom_Level_1::ManageLevel(int32 NewLevel)
{
   IsActive = Level == NewLevel;
   SetWarningLight(Level < NewLevel);
   SetActorTickEnabled(Level > NewLevel);
   SetHints(NewLevel);
   return IsActive;
}


```

Mètode que defineix l'estat del nivell a partir de `NewLevel`.

Fa un efecte visual canviant el color de *Light* en cas que el nou nivell sigui superior a aquest, per indicar que el nivell està superat. Assigna les pistes corresponents segons el nivell, i activa o desactiva el *Ticking* un cop superat el nivell per tal de millorar el rendiment.

SetWarningLight

```

UFUNCTION(BlueprintCallable, BlueprintImplementableEvent)
void SetWarningLight(bool IsRight);

```

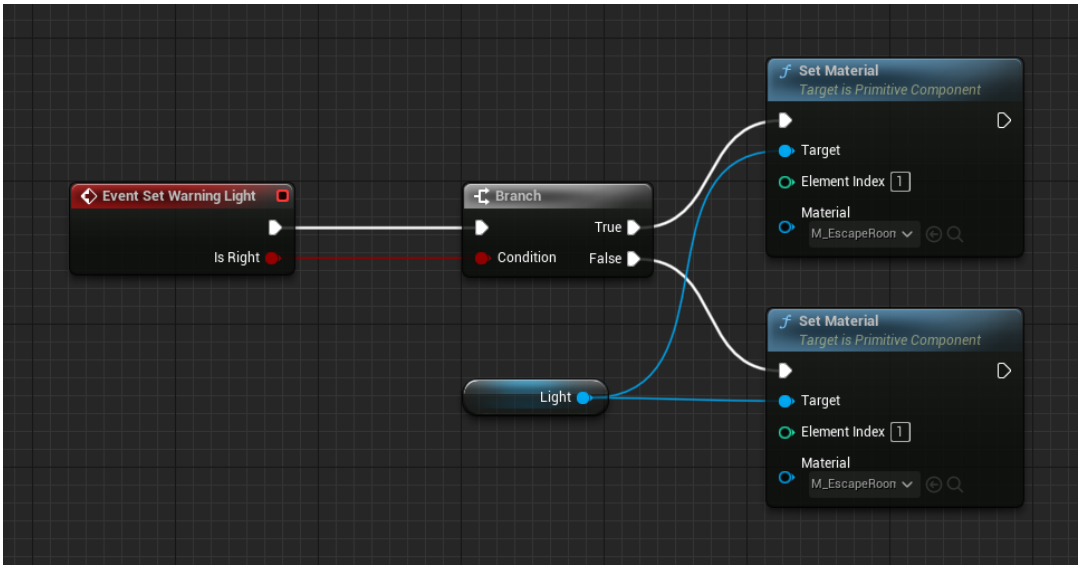

FIGURA 8.34: Implementació del Blueprint SetWarningLight

Mètode que canvia el material de la figura **Light** en funció del paràmetre `IsRight`. Veure Figura 8.34.

SetHints

```
virtual void SetHints(int32 newLevel) override;

```

```
void AEscapeRoom_Level_1::SetHints(int32 newLevel)
{
   Hints.Empty();
   HintIndex = 0;
   if(newLevel < Level)
   {
      Hints.Emplace(TEXT("You are not ready yet."));
   }
   else if(newLevel == Level)
   {
      Hints.Emplace(TEXT("What's hidden can't be seen by sight,\nUse this light to make it
↪  bright."));
      Hints.Emplace(TEXT("Invisible ink lies on this page,\nShine a UV light to reveal the
↪  sage."));
   }
   else if(newLevel > Level)
   {
      Hints.Emplace(TEXT("This level has already been solved."));
   }
}

```

Mètode que, depenent del nivell actual, assigna les pistes que mostrarà el nivell en cas que es sol·liciti la seva pista.

8.2.8.2 Llum ultra-violeta

La llum ultra-violeta és un *plugin* [10] obtingut de la comunitat d'Unreal Engine, i se n'ha utilitzat els actors UVRod i UVSprite, que s'han utilitzat en diferents targetes del nivell 1 per ocultar el codi de números i colors a ser revelat en la proximitat de l'actor l'UVRod per passar el nivell. Veure Figures 8.35 i 8.36.

BP_UVRod i BP_UVSprite

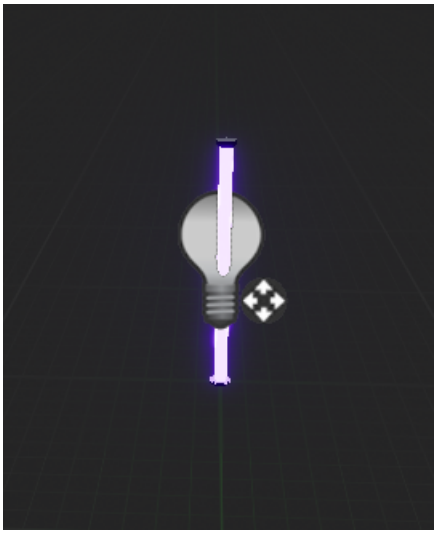

FIGURA 8.35: Vareta de llum Ultra-violeta

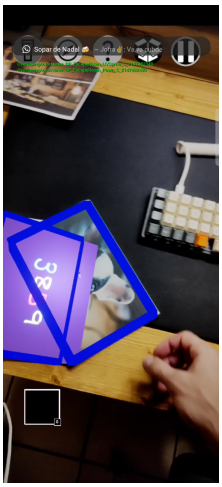

FIGURA 8.36: UVSprite exposat per l'UVRod

8.2.8.3 WBP_InputCode

L'entrada del codi que es revela amb la llum ultra-violeta es fa a través del *widget* **WBP_InputCode**. Aquest *widget* apareix al clicar la capça de col·lisió del nivell 1 té un aspecte que dona una pista al jugador de l'ordre en que ha d'entrar els números en cas que el jugador ja hagi revelat prèviament el codi. Veure Figura 8.37

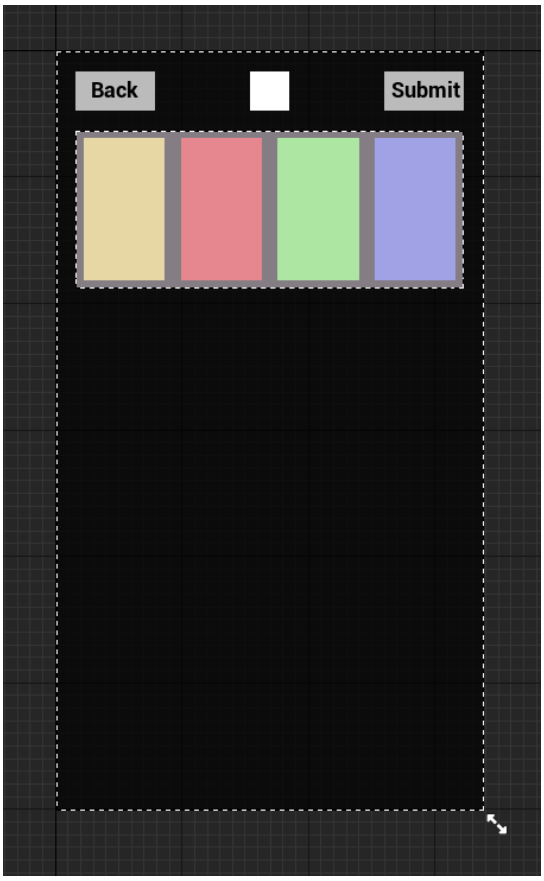

FIGURA 8.37: Disseny del Widget WBP_InputCode

8.2.8.3.1 Atributs

EditableTexBox_{1|2|3|4}(TextBox)→ Text box's per l'entrada del codi. Cada casella s'utilitza per l'entrada d'un i només un digit i és del color que te el número revelat per la llum UV.

Back(Button)→ Botó per sortir del *widget* i continuar amb la partida.

Submit(Button)→ Botó per enviar el codi entrat en les caselles editables.

Image_Rightness(Image)→ Imatge que pren el color vermell o verd depenent de si s'ha enviat un codi correcte o no.

8.2.8.3.2 Mètodes

On Text Changed

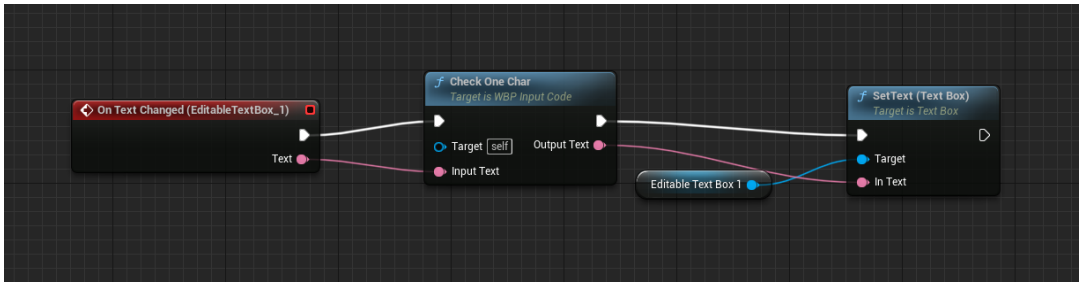

FIGURA 8.38: Implementació del Blueprint OnTextChanged

Implementació de l'OnTextChanged pels quatre *TextBox's* dedicats a l'entrada del codi que s'executa cada vegada que hi ha un canvi en el text en qüestió i assegura que només hi hagi un únic caràcter al *TextBox*. Veure Figura 8.38.

On Click (Back)

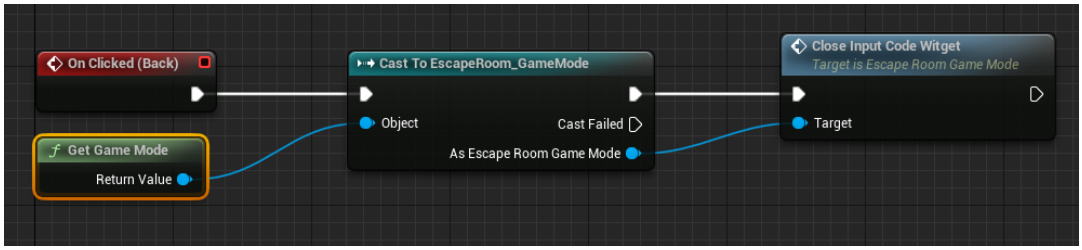

FIGURA 8.39: Implementació del Blueprint OnClick(Back)

Mètode per ocultar el *widget* d'entrada de codi. Veure Figura 8.38.

On Click (Submit)

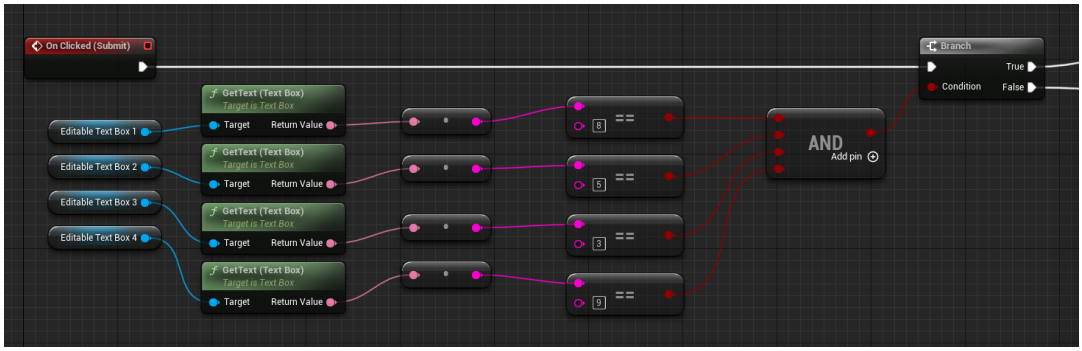

FIGURA 8.40: Implementació del Blueprint OnClick(Submit) Part 1

Primera part on es valida si els dígits del codi s'han entrat correctament i a les caselles adequades. Aquesta comprovació és molt manual i genera una dependència clara amb el

BP_UVSprite, que no permetria reutilitzar la funció en cas de voler utilitzar múltiples vegades aquesta mecànica en la partida.

Per tant, es podria replantejar i que fos cada *BP_UVSprite* qui fes manteniment del seu propi codi, parametritzant el *widget*. Veure Figura 8.40.

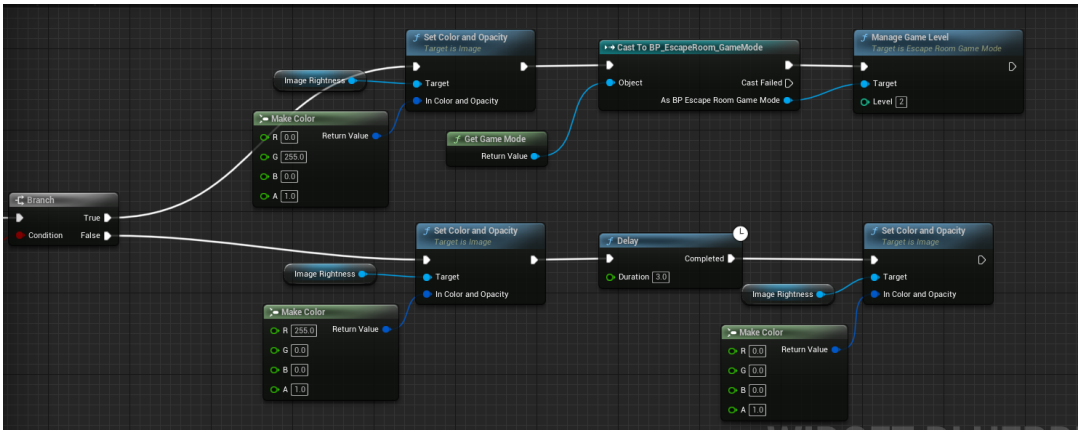

FIGURA 8.41: Implementació del Blueprint OnClick(Submit) Part 2

Es modifica el color de l'**ImageRightness** depenent de la correctesa del codi entrat i es notifica al **GameMode** que s'ha superat el nivell. Veure Figura 8.41.

8.2.9 Nivell 2

El nivell 2 està format per una plataforma amb obstacles i dues boles que es representen en el pla de la targeta. El jugador haurà de fer arribar les boles a dues plataformes per superar en nivell. Per aconseguir-ho, caldrà inclinar les targetes en el mon real, amb què les boles simularan el moviment que tindrien al ser afectades per la gravetat. Un cop aconseguit, les boles queden fixes sobre les plataformes i s'activa el següent nivell. Veure Figura 8.42.

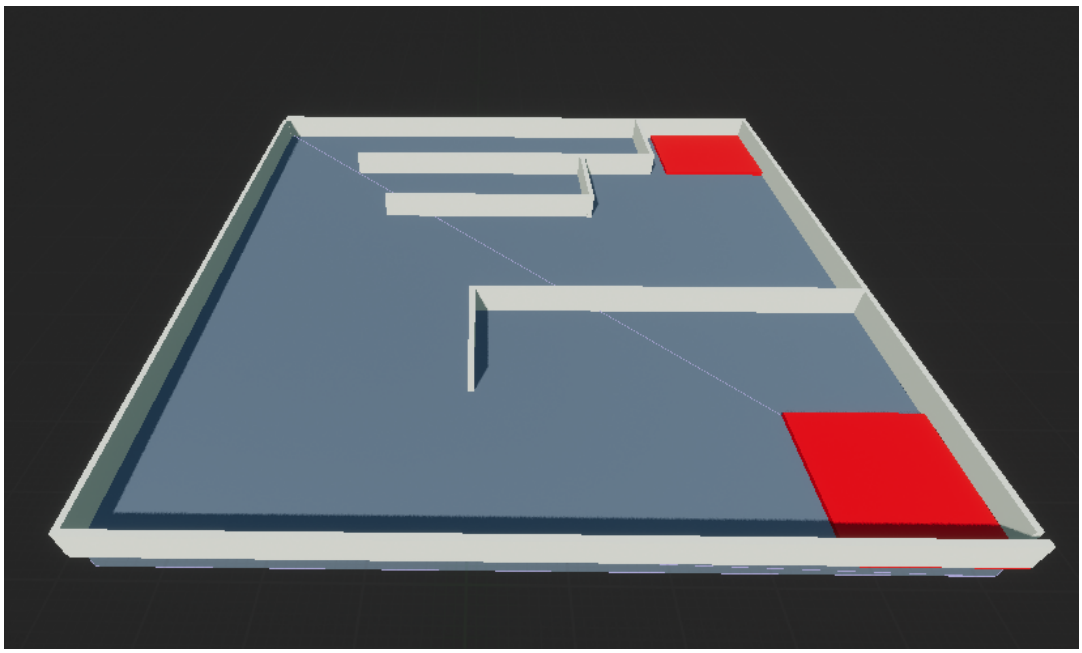

FIGURA 8.42: Disseny del nivell 2

8.2.9.1 EscapeRoom_Level_2

8.2.9.1.1 Atributs

```
UPROPERTY(EditDefaultsOnly, Category = "Components", meta = (AllowPrivateAccess = "true"))
USceneComponent* Scene;
```

Component escena per donar transformació als elements del nivell.

```
UPROPERTY(EditDefaultsOnly, Category = "Components", meta = (AllowPrivateAccess = "true"))
UStaticMeshComponent* BaseMesh;
```

Plataforma base del nivell.

```
UPROPERTY(EditDefaultsOnly, Category = "Components", meta = (AllowPrivateAccess = "true"))
class UEscapeRoom_Trigger* CollisionBox_1;
```

```
UPROPERTY(EditDefaultsOnly, Category = "Components", meta = (AllowPrivateAccess = "true"))
class UEscapeRoom_Trigger* CollisionBox_2;
```

Capces de col·lisió a les quals s'ha de fer arribar les dues boles per superar el nivell.

```
UPROPERTY()
class AEscapeRoom_Balancer* Balancer_1;
```

```
UPROPERTY()
class AEscapeRoom_Balancer* Balancer_2;
```

Actors AEscapeRoom_Balancer Balancer 1 i 2. Implementen la lògica per tal que la inclinació de la targeta els generi un moviment controlat i tenen una representació física en forma de bola.

```
UPROPERTY(EditDefaultsOnly, Category = "Components", meta = (AllowPrivateAccess = "true"))
TSubclassOf<class AEscapeRoom_Balancer> BalancerClass;
```

Atribut per donar tipus *blueprint* als objectes Balancer_1 i Balancer_2.

8.2.9.1.2 Metodes i Esdeveniments

Constructor

```
AEscapeRoom_Level_2();

```

```
AEscapeRoom_Level_2::AEscapeRoom_Level_2()
{
    PrimaryActorTick.bCanEverTick = true;
    Level = 2;
    HintIndex = 0;

    Scene = CreateDefaultSubobject<USceneComponent>(TEXT("Scene"));
    RootComponent = Scene;


    BaseMesh = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Base Mesh"));
     BaseMesh->SetupAttachment(Scene);
     BaseMesh->bDrawMeshCollisionIfSimple = true;
     BaseMesh->bDrawMeshCollisionIfComplex = true;

    CollisionBox_1 = CreateDefaultSubobject<UEscapeRoom_Trigger>(TEXT("Collision Box 1"));
    CollisionBox_1->SetupAttachment(Scene);

    CollisionBox_2 = CreateDefaultSubobject<UEscapeRoom_Trigger>(TEXT("Collision Box 2"));
    CollisionBox_2->SetupAttachment(Scene);

}

```

S'inicialitzen els atributs, components i objectes i s'adjunten a l'escena.

BeginPlay

```
virtual void BeginPlay() override;

```

```
void AEscapeRoom_Level_2::BeginPlay()
{
   Super::BeginPlay();
   SetActorTickEnabled(true);

   FTransform SocketTransform = BaseMesh->GetSocketTransform(TEXT("BalancerSocket_1"));
   Balancer_1 = GetWorld()->SpawnActor<AEscapeRoom_Balancer>(BalancerClass,
    SocketTransform);

   if(Balancer_1 != nullptr)
   {
      Balancer_1->AttachToComponent( BaseMesh ,
    FAttachmentTransformRules::SnapToTargetIncludingScale, TEXT("BalancerSocket_1"));
      Balancer_1->SetOwner(this);
      Balancer_1->SetLevel(Level);
   }
   SocketTransform = BaseMesh->GetSocketTransform(TEXT("BalancerSocket_2"));
   Balancer_2 = GetWorld()->SpawnActor<AEscapeRoom_Balancer>(BalancerClass,
    SocketTransform);

   if(Balancer_2 != nullptr)
   {
      Balancer_2->AttachToComponent( BaseMesh ,
    FAttachmentTransformRules::SnapToTargetIncludingScale, TEXT("BalancerSocket_2"));
      Balancer_2->SetOwner(this);
      Balancer_2->SetLevel(Level);
   }

}

```

Com que els balancers son també actors que hereten de *EscapeRoom_Level*, els hem de generar un cop hem creat l'objecte. En aquest mètode es generen els dos balancers a la posició definida de l'escena i se'ls assigna el nivell al qual pertanyen (ja que poden ser reutilitzats en altres nivells).

Tick

```
virtual void Tick(float DeltaTime) override;

```

```
Super::Tick(DeltaTime);
if(CollisionBox_1->IsPressed() && CollisionBox_2->IsPressed())
{
    AEscapeRoom_GameMode *GameMode =
    Cast<AEscapeRoom_GameMode>(UGameplayStatics::GetGameMode(this));
    if(GameMode != nullptr)
    {
        GameMode->ManageGameLevel(3);
    }

}

```

Comprova si les dues boles estan sobre les plataformes CollisionBox i, en cas afirmatiu, avisa al GameMode que el nivell s'ha superat.

ManageLevel

```
virtual bool ManageLevel(int32 NewLevel) override;

```

```cpp
bool AEscapeRoom_Level_2::ManageLevel(int32 NewLevel)
{

    IsActive = Level == NewLevel;
   SetActorTickEnabled(IsActive);
   SetHints(NewLevel);
   HintIndex = 0;
   if(Balancer_1 != nullptr)
   {
      Balancer_1->ManageLevel(NewLevel);
   }
   if(Balancer_2 != nullptr)
   {
      Balancer_2->ManageLevel(NewLevel);
   }
   return IsActive;
}

```

Activa o desactiva el *Ticking* en funció de si el nivell està actiu per tal de millorar el rendiment i que no es pugui resoldre sense haver resolt abans el nivell previ, assigna les pistes corresponents al nivell actual i actualitza l'estat dels balancers 1 i 2 segons el `NewLevel`.

SetHints

```cpp
virtual void SetHints(int32 newLevel) override;

```

```cpp
void AEscapeRoom_Level_2::SetHints(int32 newLevel)
{
   Hints.Empty();
   HintIndex = 0;
   if(newLevel < Level)
   {
      Hints.Emplace(TEXT("Crack the code before you proceed,\nTo unlock the path you
↪  need."));
   }
   else if(newLevel == Level)
   {
      Hints.Emplace(TEXT("Move with grace, tilt with might,\nActivate the buttons to shed
↪  some light."));
      Hints.Emplace(TEXT("Use the force of motion to make a switch,\nTilt your world and
↪  make them twitch."));
   }
   else if(newLevel > Level)
   {
      Hints.Emplace(TEXT("This level has already been solved."));
   }
}

```

Assigna les pistes en funció del nivell al que es troba la partida.

8.2.9.2 EscapeRoom_Balancer

8.2.9.2.1 Atributs

```
UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Components", meta =
↪  (AllowPrivateAccess = "true"))
USceneComponent* Scene;
```

SceneComponent per donar transformació a l'actor.

```
UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Puzzle",  meta =
↪  (AllowPrivateAccess = "true") )
UStaticMeshComponent* MovingMesh;
```

Static Mesh que realitza el moviment.

```
UPROPERTY(VisibleAnywhere, Category = "Movement")
class UEscapeRoom_BalancerComponent* BalancerComponent;
```

Component que implementa els càlculs del moviment a partir de la inclinació de l'objecte pare.

8.2.9.2.2 Metodes i Esdeveniment

Constructor

```
// Sets default values for this actor's properties
AEscapeRoom_Balancer();

```

```
// Sets default values
AEscapeRoom_Balancer::AEscapeRoom_Balancer()
{
   // Set this actor to call Tick() every frame.  You can turn this off to improve
↪  performance if you don't need it.
   PrimaryActorTick.bCanEverTick = true;

   Scene = CreateDefaultSubobject<USceneComponent>(TEXT("Scene"));
   RootComponent = Scene;

   MovingMesh = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Moving Mesh"));
   MovingMesh->SetupAttachment(Scene);


   BalancerComponent =
↪  CreateDefaultSubobject<UEscapeRoom_BalancerComponent>(TEXT("Balancer Component"));
   BalancerComponent->SetMovingScene(RootComponent);

}

```

Inicialitza el component escena i el l'*StaticMesh* que tindrà moviment, i inicialitza el component encarregat d'aplicar el moviment sobre l'atribut *MovingMesh*.

ManageLevel

```
virtual bool ManageLevel(int32 NewLevel) override;

```

```
bool AEscapeRoom_Balancer::ManageLevel(int32 NewLevel)
{
   IsActive = Level == NewLevel;
   IsActive ? BalancerComponent->Enable() : BalancerComponent->Disable();
   SetActorTickEnabled(IsActive);
   return IsActive;
}

```

Activa o desactiva el *Tick* per millorar el rendiment, i activa o desactiva el component *BalancerComponent* per evitar que les boles es moguin quan el nivell no està actiu.

8.2.9.3 EscapeRoom_BalancerComponent

8.2.9.3.1 Atributs

```
UPROPERTY()
USceneComponent* Scene;

```

Escena sobre la que s'aplica la transformació de moviment en funció de la inclinació del pare.

```
UPROPERTY(EditAnywhere, Category="Movement")
float Speed = 20.f;

```

Factor de velocitat de moviment.

8.2.9.3.2 Mètodes

Constructor

```
UEscapeRoom_BalancerComponent();
```

```
UEscapeRoom_BalancerComponent::UEscapeRoom_BalancerComponent()
{
   // Set this component to be initialized when the game starts, and to be ticked every
   ↪  frame.  You can turn these features
   // off to improve performance if you don't need them.
   PrimaryComponentTick.bCanEverTick = true;

   // ...
}
```

Constructor del component. Activa el *Tick* per defecte.

TickComponent

```
virtual void TickComponent(float DeltaTime, ELevelTick TickType,
↪  FActorComponentTickFunction* ThisTickFunction) override;
```

```cpp
void UEscapeRoom_BalancerComponent::TickComponent(float DeltaTime, ELevelTick TickType,
    FActorComponentTickFunction* ThisTickFunction)
{
    Super::TickComponent(DeltaTime, TickType, ThisTickFunction);
    Move();
    // ...
}
```

Executa la funció **Move** cada *frame*.

SetMovingScene

```cpp
void SetMovingScene(USceneComponent* SceneToMove)
```

```cpp
void UEscapeRoom_BalancerComponent::SetMovingScene(USceneComponent* SceneToMove)
{
    Scene = SceneToMove;
}
```

Scene apunta a l'*SceneComponent* `SceneToMove`, que serà l'escena que rebrà la transformació de posició.

Disable

```cpp
void Disable();
```

```cpp
void UEscapeRoom_BalancerComponent::Disable()
{
    SetComponentTickEnabled(false);
}
```

Desactiva el *Ticking* del component.

Enable

```cpp
void Enable();
```

```cpp
void UEscapeRoom_BalancerComponent::Enable()
{
    SetComponentTickEnabled(true);
}
```

Activa el *Ticking* del component.

Move

```
void Move();
```

```
void UEscapeRoom_BalancerComponent::Move()
{
   AActor* owner = GetOwner();
   if( owner != nullptr)
   {
      FRotator RootRotation = owner->GetRootComponent()->GetComponentRotation();

      GEngine->AddOnScreenDebugMessage(-1, GWorld->DeltaTimeSeconds, FColor::Yellow,
 ↪  FString::Printf(TEXT("moving")) );


      FVector DeltaLocation = FVector::ZeroVector;
      DeltaLocation.X = -RootRotation.Pitch * Speed *
 ↪  UGameplayStatics::GetWorldDeltaSeconds(this);
      DeltaLocation.Y = RootRotation.Roll * Speed *
 ↪  UGameplayStatics::GetWorldDeltaSeconds(this);
      FHitResult* HitResult = nullptr;
      Scene->AddLocalOffset(DeltaLocation, true, HitResult);
   }

}
```

Obté la rotació del propietari del component i, a partir d'aquesta i el factor de velocitat **Speed**, n'obté la següent posició de l'escena que rep el moviment i l'aplica sobre **Scene**.

8.2.10 Nivell 3

El nivell 3 consta d'una plataforma amb un emissor laser, un sensor receptor laser, i una figura que fa efecte mirall, tots representats en una mateixa targeta. Els tres elements estan sotmesos a una mecànica que els fa rotar sobre si mateixos en funció de la rotació del dispositiu de realitat augmentada. L'objectiu per superar el nivell és alinear el laser amb el sensor, però la intensió és distribuir les 3 figures presents a la plataforma de manera que només amb la targeta principal no sigui possible accedir al sensor laser. D'aquesta manera, caldria una segona targeta que generés una figura de tipus mirall per tal de rebotar el laser i així accedir al sensor i superar el nivell. Veure Figures 8.43 i 8.44.

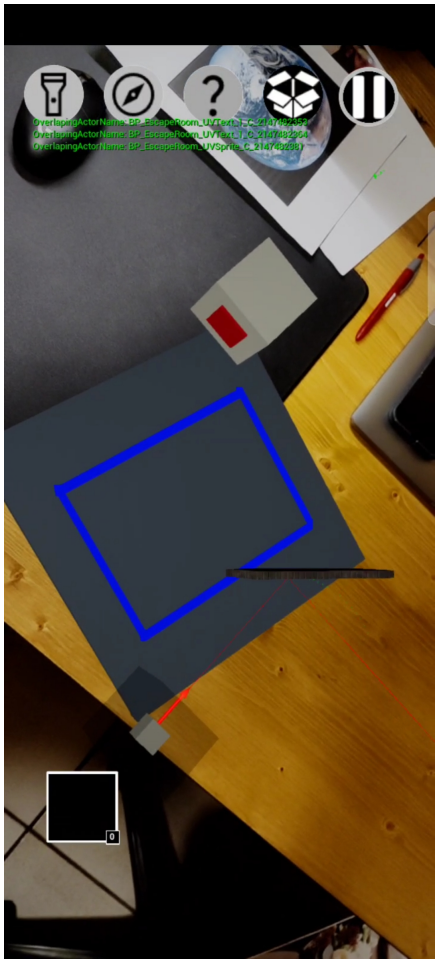

FIGURA 8.43: Disseny del nivell 3

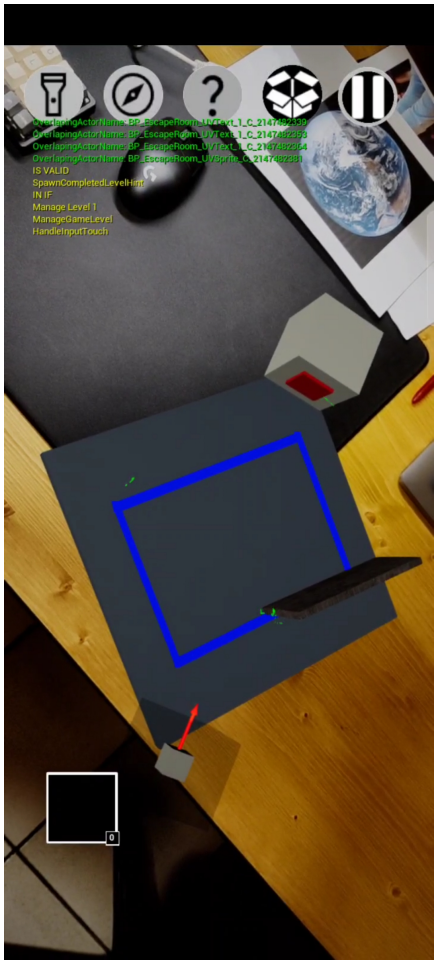

FIGURA 8.44: Nivell 3 un cop el laser te contacte amb el sensor

8.2.10.1 EscapeRoom_Level_3

8.2.10.1.1 Atributs

```
1 UPROPERTY(EditDefaultsOnly, Category = "Components", meta = (AllowPrivateAccess = "true"))
2 USceneComponent* Scene;
3
```

Escena que dona lloc al nivell.

```
    UPROPERTY(EditDefaultsOnly, Category = "Components", meta = (AllowPrivateAccess =
↪   "true"))
   UStaticMeshComponent* BaseMesh;

```

Plataforma base del nivell.

```
UPROPERTY()
class AEscapeRoom_LaserEmitter* LaserEmitter;

```

Actor emissor del laser.

```
UPROPERTY(EditDefaultsOnly, Category = "Components", meta = (AllowPrivateAccess = "true"))
TSubclassOf<class AEscapeRoom_LaserEmitter> LaserEmitterClass;

```

Classe d'emissor de laser per associar el *LaserEmitter* l'hereditari en Blueprint. LaserEmitterClass, prendrà valor en els defaults del Blueprint, d'una classe hereditària de AEscapeRoom_LaserEmitter. D'aquesta manera des de la classe en c++ es pot fer *spawn* d'una classe Blueprint. Es pot veure'n l'us a la funció **BeginPlay** d'aquest apartat.

```
UPROPERTY()
class AEscapeRoom_LaserSensor* LaserSensor;

```

Actor sensor de laser.

```
UPROPERTY(EditDefaultsOnly, Category = "Components", meta = (AllowPrivateAccess = "true"))
TSubclassOf<class AEscapeRoom_LaserSensor> LaserSensorClass;

```

Classe de sensor de laser per associar el *LaserSensor* l'hereditari en Blueprint.

```
UPROPERTY(EditDefaultsOnly, Category = "Components", meta = (AllowPrivateAccess = "true"))
class UStaticMeshComponent* Mirror_1;

```

Mesh prendrà un material reflectant i actuarà com a mirall pel laser.

```
UPROPERTY(VisibleAnywhere, Category = "Movement")
class UEscapeRoom_InputMotionComponent* LaserSensorInputMotionComp;

```

```
UPROPERTY(VisibleAnywhere, Category = "Movement")
class UEscapeRoom_InputMotionComponent* LaserEmitterInputMotionComp;

```

```
UPROPERTY(VisibleAnywhere, Category = "Movement")
class UEscapeRoom_InputMotionComponent* MirrorInputMotionComp;

```

Components encarregats d'efectuar la rotació dels objectes a partir de la rotació del dispositiu als que s'associen els tres elements del nivell: Mirror_1, LaserEmitter i LaserSensor.

8.2.10.1.2 Mètodes i Esdeveniment

Constructor

```
AEscapeRoom_Level_3();
```

```
AEscapeRoom_Level_3::AEscapeRoom_Level_3()
{
   // Set this actor to call Tick() every frame.  You can turn this off to improve performance if you don't need it.
   PrimaryActorTick.bCanEverTick = true;
   Level = 3;


   Scene = CreateDefaultSubobject<USceneComponent>(TEXT("Scene"));
   RootComponent = Scene;

   BaseMesh = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Base Mesh"));
   BaseMesh->SetupAttachment(Scene);


   Mirror_1 = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Mirror 1"));
   Mirror_1->SetupAttachment(Scene);


   LaserEmitterInputMotionComp = CreateDefaultSubobject<UEscapeRoom_InputMotionComponent>(TEXT("Laser Emitter Input Motion Component"));
   LaserSensorInputMotionComp = CreateDefaultSubobject<UEscapeRoom_InputMotionComponent>(TEXT("Laser Sensor Input Motion Component"));
   MirrorInputMotionComp = CreateDefaultSubobject<UEscapeRoom_InputMotionComponent>(TEXT("Mirror Input Motion Component"));

}
```

Inicialitza el nivell, crea les figures corresponents i les vincula a l'escena. També inicialitza els components que posteriorment seran vinculats als objectes sobre els que han d'aplicar el moviment.

BeginPlay

```
virtual void BeginPlay() override;
```

```cpp
// Called when the game starts or when spawned
void AEscapeRoom_Level_3::BeginPlay()
{
   Super::BeginPlay();
   // SetActorTickEnabled(false);


   FTransform SocketTransform = BaseMesh->GetSocketTransform(TEXT("LaserEmitterSocket"));
   LaserEmitter = GetWorld()->SpawnActor<AEscapeRoom_LaserEmitter>(LaserEmitterClass,
    SocketTransform);


   if(LaserEmitter != nullptr)
   {
      LaserEmitterInputMotionComp->SetMovingScene(LaserEmitter->GetRootComponent());
      LaserEmitter->AttachToComponent( BaseMesh ,
    FAttachmentTransformRules::SnapToTargetIncludingScale, TEXT("LaserEmitterSocket"));
      LaserEmitter->SetOwner(this);
      LaserEmitter->SetLevel(Level);
   }

   SocketTransform = BaseMesh->GetSocketTransform(TEXT("LaserSensorSocket"));
   LaserSensor = GetWorld()->SpawnActor<AEscapeRoom_LaserSensor>(LaserSensorClass,
    SocketTransform);

   if(LaserSensor != nullptr)
   {
      LaserSensorInputMotionComp->SetMovingScene(LaserSensor->GetRootComponent());
      LaserSensor->AttachToComponent( BaseMesh ,
    FAttachmentTransformRules::SnapToTargetIncludingScale, TEXT("LaserSensorSocket"));
      LaserSensor->SetOwner(this);
   }

   if(Mirror_1 != nullptr)
   {
      MirrorInputMotionComp->SetMovingScene(Mirror_1);
   }
}
```

Genera i col·loca sobre la plataforma els actors **LaserEmitter** i **LaserSensor** i n'inicialitza el nivell de cadascun, que també hereten de *EscapeRoom_Level*.

Vincula ambdós objectes i el **Mirror_1** amb el component encarregat d'aplicar-hi moviment a la rotació del dispositiu.

Tick

```cpp
virtual void Tick(float DeltaTime) override;
```

```cpp
void AEscapeRoom_Level_3::Tick(float DeltaTime)
{
   Super::Tick(DeltaTime);


   if(LaserEmitter->HasActiveSensor())
   {

      AEscapeRoom_GameMode *GameMode =
    Cast<AEscapeRoom_GameMode>(UGameplayStatics::GetGameMode(this));
      if(GameMode != nullptr)
      {
         GameMode->ManageGameLevel(5);
      }
      LaserEmitterInputMotionComp->Disable();
      LaserSensorInputMotionComp->Disable();
      MirrorInputMotionComp->Disable();

   }
}
```

Valida en cada *Frame* si el **LaserEmitter** està rebotant en algun sensor. En cas afirmatiu, notifica al GameLevel que s'ha superat el nivell i desactiva els components. La tasca de desactivar els components hauria de recaure sobre el **ManageLevel**, però en alguna situació

el *delay* era prou gran com perquè el LaserEmitter realitzés moviment i quedés desalineat el laser i el sensor.

ManageLevel

```
virtual bool ManageLevel(int32 NewLevel) override;
```

```
bool AEscapeRoom_Level_3::ManageLevel(int32 NewLevel)
{
   IsActive = Level == NewLevel;
   SetActorTickEnabled(IsActive);
   SetHints(NewLevel);
   LaserEmitter->ManageLevel(NewLevel);
   return IsActive;
}
```

Desactiva el *ticking* en cas de que el nivell actual sigui diferent al propi, assigna les pistes corresponents i administra el nivell de l'emisor de laser.

SetHints

```
virtual void SetHints(int32 newLevel) override;
```

```
void AEscapeRoom_Level_3::SetHints(int32 newLevel)
{
   Hints.Empty();
   HintIndex = 0;
   if(newLevel < Level)
   {
      Hints.Emplace(TEXT("To fire the laser and light up the way,\nPress the buttons in
↪  the correct array."));
      Hints.Emplace(TEXT("Buttons must be pushed with care,\nTo activate the laser beam in
↪  the air"));

   }
   else if(newLevel == Level)
   {
      Hints.Emplace(TEXT("The laser beam needs a guiding hand,\nBounce it off mirrors to
↪  hit the sensor stand."));
      Hints.Emplace(TEXT("The sensor waits for the laser's shine,\nUse mirrors to guide it
↪  and align"));
   }
   else if(newLevel > Level)
   {
      Hints.Emplace(TEXT("This level has already been solved."));
   }
}
```

Assigna les pistes depenent del nivell al que estigui la partida.

8.2.10.2 EscapeRoom_LaserEmitter

EscapeRoom_LaserEmitter és la classe que implementa l'emissor laser utilitzat en el nivell 3. Conceptualment és una figura física que emet un raig laser visible en una direcció. Aquest raig laser té una distància màxima, i es susceptible a ser reflectit en establir contacte amb cert material reflectant, a la vegada que detectar l'impacte en una figura definida com a sensor. **EscapeRoom_LaserEmitter** hereta de la classe EscapeRoom_Level. Veure Figura 8.45

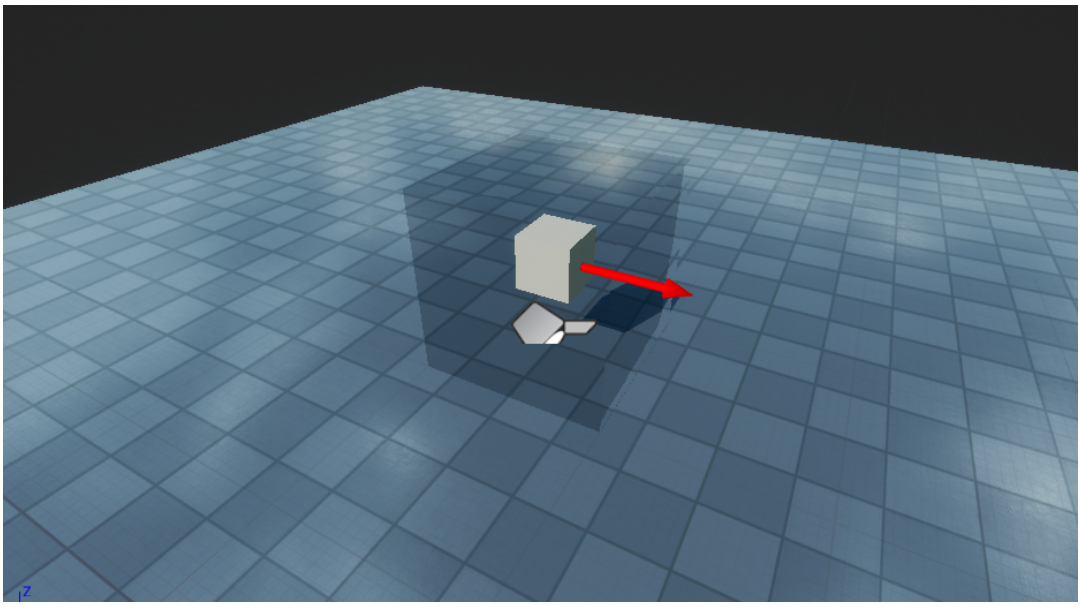

FIGURA 8.45: Figura d'EscapeRoom_LaserEmitter

8.2.10.2.1 Atributs

```
UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Components", meta =
↪  (AllowPrivateAccess = "true"))
UStaticMeshComponent* EmitterRoot;

```

```
UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Components", meta =
↪  (AllowPrivateAccess = "true"))
UStaticMeshComponent* Emitter;

```

```
UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Components", meta =
↪  (AllowPrivateAccess = "true"))
class UArrowComponent* Arrow;

```

Per donar un aspecte interessant a l'emissor laser sense haver de tenir coneixements de disseny, es va utilitzar el següent:

- EmitterRoot: *UStaticMeshComponent* de material **M_Glass**, fent un efecte de vidre i que podem relacionar amb la lent d'un laser.
- Emitter: *UStaticMeshComponent* bàsic que representa l'origen de la llum.
- Arrow: *UArrowComponent* que ens dona una idea clara de la direcció i color que tindrà la llum. De l'*UArrowComponent* també se n'obté el vector que dona direcció a la llum.

```
UPROPERTY(EditDefaultsOnly, Category = "Components", meta = (AllowPrivateAccess = "true"))
UMaterialInterface* MirrorMaterial;

```

Material que és detectat com a reflectint i causarà el rebot del làser.

```
UPROPERTY(EditDefaultsOnly, Category = "Components", meta = (AllowPrivateAccess = "true"))
UMaterialInterface* SensorMaterial;

```

Material que és detectat com a sensor per activar un *flag* de sensor actiu.

```
UPROPERTY(EditDefaultsOnly, Category = "Components", meta = (AllowPrivateAccess = "true"))
class AEscapeRoom_LaserSensor* ActivatedSensor;

```

Punter a la un **AEscapeRoom_LaserSensor** i que s'utilitza com a referència al sensor actiu en cas que el laser estigui en contacte amb un sensor.

```
UPROPERTY(EditAnywhere)
float Distance = 10000.f;

```

Màxima distància que s'emet el laser.

8.2.10.2.2 Metodes i Esdeveniment

Constructor

```
AEscapeRoom_LaserEmitter();

```

```
AEscapeRoom_LaserEmitter::AEscapeRoom_LaserEmitter()
{
   // Set this actor to call Tick() every frame.  You can turn this off to improve
 ↪  performance if you don't need it.
   PrimaryActorTick.bCanEverTick = true;


   EmitterRoot = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("EmitterRoot"));
   RootComponent = EmitterRoot;

   Emitter = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Emitter"));
   Emitter->SetupAttachment(EmitterRoot);

   Arrow = CreateDefaultSubobject<UArrowComponent>(TEXT("Arrow"));
   Arrow->SetupAttachment(Emitter);

   Level = 3;

}

```

S'inicialitzen els diferents components i es vinculen a l'escena. S'activa el permís d'executar la funció **Tick**. També es dóna valor al nivell, però hauria de ser el nivell pare qui assignés aquest valor ja que és un objecte que no implementa un nivell per si mateix sinó per ser inclòs en un nivell, en aquest cas, **EscapeRoom_Level_3**.

BeginPlay

```
virtual void BeginPlay() override;

```

```
void AEscapeRoom_LaserEmitter::BeginPlay()
{
   Super::BeginPlay();
   SetActorTickEnabled(false);

}

```

Es desactiva el **Tick** en ser creat l'objecte per millorar rendiment i ja que el laser no s'activa fins que el nivell actual sigui al que pertany l'objecte.

Tick

```
virtual void Tick(float DeltaTime) override;

```

```
// Called every frame
void AEscapeRoom_LaserEmitter::Tick(float DeltaTime)
{
   Super::Tick(DeltaTime);
   // TODO: afegir un delay amb els timers de c++ perquè fer aquesta funció a cada tick és
   ↪  molt costós

   CastLight(Arrow->GetComponentLocation(),Arrow->GetForwardVector(), Distance );

}

```

En cada *frame* del videojoc, es crida la funció encarregada de calcular el raig de llum. És passa per paràmetre l'origen del raig i la direcció, que s'obtenen a partir de l'atribut **Arrow** explicat anteriorment, i la distancia del raig.

La funció podria ser optimitzada afegint un *delay* en l'execució del Tick. Com s'explica a continuació, la funció **CastLight** realitza una crida recursiva en el calcul de la direcció del laser, i sobrecarregar crides recursives podria afectar el rendiment del videojoc.

El casos on més impacte tindria és si es realitzessin molts rebots en miralls. Aquest no és el cas en l'estat del projecte, així que no és un factor que afecti a la jugabilitat.

CastLight

```
void CastLight(FVector CastOrigin, FVector CastDirection, float CastDistance);

```

```
void AEscapeRoom_LaserEmitter::CastLight(FVector CastOrigin, FVector CastDirection, float
↪  CastDistance)
{
   FVector CastEnd = CastDirection*CastDistance + CastOrigin;


    //Debug
   FCollisionShape Sphere = FCollisionShape::MakeSphere(0.5);
   FHitResult HitResult;
   FCollisionQueryParams TraceParams = FCollisionQueryParams();
   TraceParams.bReturnFaceIndex=true;
   TraceParams.bTraceComplex =true;

   bool HasHit = GetWorld()->LineTraceSingleByChannel(
      HitResult,
      CastOrigin,CastEnd,
      ECC_GameTraceChannel1,
      TraceParams
   );

   if(HasHit){
      DrawDebugLine(GetWorld(),CastOrigin,HitResult.ImpactPoint,FColor::Red);
      DrawDebugSphere(GetWorld(), HitResult.ImpactPoint, 0.5, 10, FColor::Green, false,
↪  5);
      //
↪  HitResult.GetComponent()->GetMaterialFromCollisionFaceIndex(HitResult.FaceIndex,HitResult.SectionIndex);

      int32 SectionIndex;
      UMaterialInterface* CollisionMaterial =
↪  HitResult.GetComponent()->GetMaterialFromCollisionFaceIndex(HitResult.FaceIndex,
↪  SectionIndex);
      // UMaterialInterface* CollisionMaterial =
↪  HitResult.GetComponent()->GetMaterial(HitResult.FaceIndex);

      if(CollisionMaterial != nullptr && MirrorMaterial != nullptr)
      {
         if(CollisionMaterial == MirrorMaterial)
         {
            CastLight(
               HitResult.ImpactPoint,
               UKismetMathLibrary::MirrorVectorByNormal(CastDirection,
↪  HitResult.ImpactNormal),
               CastDistance-(FVector::Distance(CastOrigin, HitResult.ImpactPoint))
            );
         }
         else if (CollisionMaterial == SensorMaterial)
         {
            GEngine->AddOnScreenDebugMessage(-1, GWorld->DeltaTimeSeconds, FColor::Yellow,
↪  (TEXT("HIT SENSOR")));
            ActivatedSensor = Cast<AEscapeRoom_LaserSensor>(HitResult.GetActor());

         }
         else
         {
            GEngine->AddOnScreenDebugMessage(-1, GWorld->DeltaTimeSeconds, FColor::Yellow,
↪  (TEXT("HIT NO MIRROR or SENSOR")));
         }

      }
   }
   else {
      ActivatedSensor = nullptr;
      DrawDebugLine(GetWorld(),CastOrigin,CastEnd,FColor::Red);
   }

}

```

A partir d'un punt origen `FVector CastOrigin`, un vector de direcció FVector `CastDirection` i la distància `float CastDistance`, és calcula el punt destí **CastEnd** i és llança un raig per detectar si hi ha hagut alguna col·lisió. En cas negatiu, es pinta el raig i es para la recursivitat. En cas afirmatiu, es pinta el laser fins al punt de la col·lisió i es consulta quin és el material amb que s'ha col·lisionat. Si és el material que hem indicat com a reflectint, es realitza una crida recursiva a la mateixa funció, passant el punt de col·lisió com a nou origen del raig, calculant la nova direcció a partir de la normal de la superfície impactada i restant la distància que el raig ja ha viatjat. Si el material és d'un sensor, es guarda la referència a l'actor que té l'objecte amb aquest material.

ManageLevel

```
virtual bool ManageLevel(int32 level) override;

```

```

bool AEscapeRoom_LaserEmitter::ManageLevel(int32 currentLevel)
{
   IsActive = Level == currentLevel;
   SetActorTickEnabled(IsActive);
   return IsActive;
}

```

Activa o desactiva el **Tick** en funció de si el nivell està actiu o no.

HasActiveSensor

```
bool HasActiveSensor();

```

```
bool AEscapeRoom_LaserEmitter::HasActiveSensor()
{
   return (ActivatedSensor != nullptr);
}

```

Retorna **true** si el hi ha un sensor actiu, false altrament.

8.2.10.3 EscapeRoom_InputMotionComponent

EscapeRoom_InputMotionComponent és la classe encarregada de la rotació d'un component d'escena a partir de la rotació del dispositiu de realitat augmentada. D'aquesta manera, tot i que el dispositiu android representi la visió del jugador, es busca crear un efecte o sensació d'estar rotant l'element virtual des del contacte del mon real, ja que el moviment que s'aplica amb les pròpies mans del jugador és el que s'acaba traslladant a l'element virtual.

8.2.10.3.1 Atributs

```
UPROPERTY()
USceneComponent* Scene;

```

Component d'escena sobre el que s'aplicaran les transformacions.

```

APlayerController* PlayerControllerRef;

```

Punter al *PlayerController* de la partida, utilitzat per obtenir l'estat del dispositiu d'entrada del jugador.

```
UPROPERTY(EditAnywhere, Category="Movement")
float Speed = 10.f;

```

Factor de velocitat del moviment que afecta l'atribut **Scene**.

8.2.10.3.2 Metodes

Constructors

```
UEscapeRoom_InputMotionComponent();
```

```
UEscapeRoom_InputMotionComponent::UEscapeRoom_InputMotionComponent()
{
   // Set this component to be initialized when the game starts, and to be ticked every
    frame.  You can turn these features
   // off to improve performance if you don't need them.
   PrimaryComponentTick.bCanEverTick = true;

   // ...
}
```

Constructor per defecte, permet el *Ticking* del component

```
UEscapeRoom_InputMotionComponent(USceneComponent * SceneToMove);
```

```
UEscapeRoom_InputMotionComponent::UEscapeRoom_InputMotionComponent(USceneComponent *
    SceneToMove)
{
   // Set this component to be initialized when the game starts, and to be ticked every
    frame.  You can turn these features
   // off to improve performance if you don't need them.
   PrimaryComponentTick.bCanEverTick = true;
   Scene = SceneToMove;
   // ...
}
```

Constructor que rep per paràmetre el puntera al component d'escena que rebrà la transformació `SceneToMove`. Assigna l'atribut **Scene** i permet el *Ticking* del component.

BeginPlay

```
virtual void BeginPlay() override;
```

```cpp
// Called when the game starts
void UEscapeRoom_InputMotionComponent::BeginPlay()
{
   Super::BeginPlay();
   PlayerControllerRef = UGameplayStatics::GetPlayerController(GetWorld(), 0);

   // ...

}
```

Assigna l'atribut **PlayerControllerRef** al *PlayerController* de la partida, que en el cas del projecte, és únic.

TickComponent

```cpp
virtual void TickComponent(float DeltaTime, ELevelTick TickType,
    FActorComponentTickFunction* ThisTickFunction) override;
```

```cpp
// Called every frame
void UEscapeRoom_InputMotionComponent::TickComponent(float DeltaTime, ELevelTick TickType,
    FActorComponentTickFunction* ThisTickFunction)
{
   Super::TickComponent(DeltaTime, TickType, ThisTickFunction);
   MoveMesh();
   // ...
}
```

Crida en cada *frame* la funció encarregada d'aplicar la lògica de transformació sobre l'objecte.

MoveMesh

```cpp
void MoveMesh() const;
```

```cpp
void UEscapeRoom_InputMotionComponent::MoveMesh() const
{
   if(Scene != nullptr)
   {
      if(PlayerControllerRef)
      {
         FVector _tilt;
         FVector _rotation;
         FVector _gravity;
         FVector _acceleration;

         PlayerControllerRef->GetInputMotionState(_tilt, _rotation, _gravity,
    _acceleration);

         FRotator DeltaRotation = FRotator::ZeroRotator;
         DeltaRotation.Yaw = _rotation.Z * -Speed *
    UGameplayStatics::GetWorldDeltaSeconds(this);
         Scene->AddLocalRotation(DeltaRotation, true);

      }
      else
      {
         GEngine->AddOnScreenDebugMessage(-1, GWorld->DeltaTimeSeconds, FColor::Blue,
    FString::Printf(TEXT("No playerControllerRef")));
      }
   }
   else
   {
         GEngine->AddOnScreenDebugMessage(-1, GWorld->DeltaTimeSeconds, FColor::Blue,
    FString::Printf(TEXT("No Scene to move")));
   }
}
```

Obté els diferents valors de l'*input motion* del dispositiu a través del **PlayerControllerRef**. S'utilitza l'eix **Z** de la rotació del dispositiu per calcular el nou *Yaw* (del sistema de moviment tridimencional Yaw, Pitch and Roll) i aquest s'afegeix a la rotació de l'escena, provocant un moviment rotatori sobre l'eix Z de l'objecte que rep la transformació.

SetMovingScene

```
void SetMovingScene(USceneComponent* SceneToMove);
```

```

void UEscapeRoom_InputMotionComponent::SetMovingScene(USceneComponent* SceneToMove)
{
   Scene = SceneToMove;
}
```

Assigna el component escena sobre la que s'aplica la transformació.

Enable/Disable

```
void Enable();
```

```
void UEscapeRoom_InputMotionComponent::Enable()
{
   SetComponentTickEnabled(true);
}
```

```
void Disable();
```

```

void UEscapeRoom_InputMotionComponent::Disable()
{
   SetComponentTickEnabled(false);
}
```

Activa el *Ticking* del component en cas de l'**Enable**, el desactiva en el **Disable**.

8.2.11 Nivell 4

Nota: a la classe del quart nivell se'l va anomenar EscapeRoom_Level_5, però en realitat és el quart.

El quart nivell busca la utilització de factors del mon real a la partida. En aquest cas es juga amb els punts cardinals, obligant al jugador a modificar la seva posició en el mon real per tal de satisfer les condicions per superar el nivell.

El nivell consisteix en col·locar la carta enfocant al nord i, tenint sel·leccionat l'*item* corresponent de l'inventari (una clau que has d'haver recollit prèviament), fer clic al pany de la porta. Sabrem que cal anar cap al nord gràcies a les pistes, i es disposa de l'eina **Compass** que mostra on estan els diferents punts cardinals. En cas que s'estigui enfocant el nord, l'element de la targeta no serà visible i per tant tampoc es podrà clicar el pany.

Aquest és l'últim nivell, un cop obres la porta estàs lliure de la sala en la qual s'està tancat en mon virtual. Veure Figures **??** i 8.47.

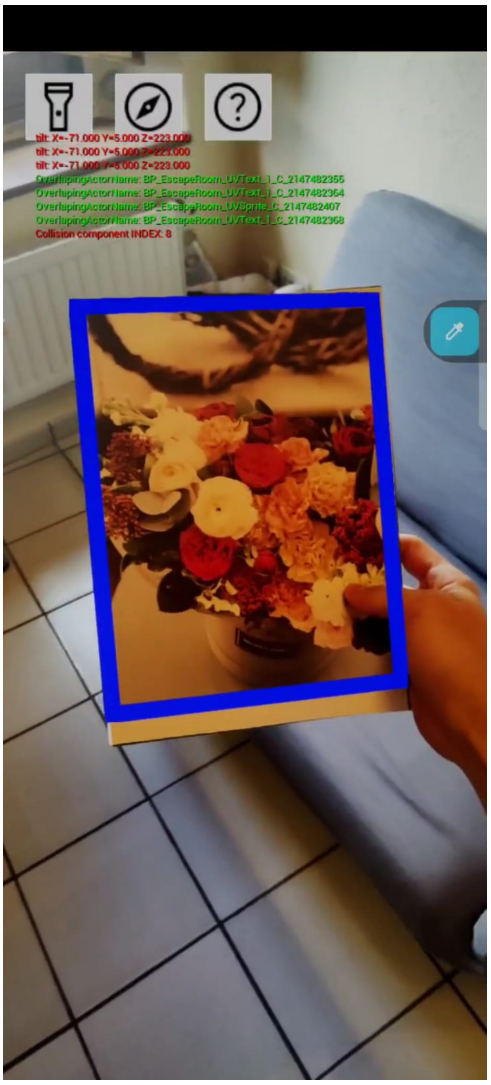

FIGURA 8.46: Escaneig sense apuntar al nord

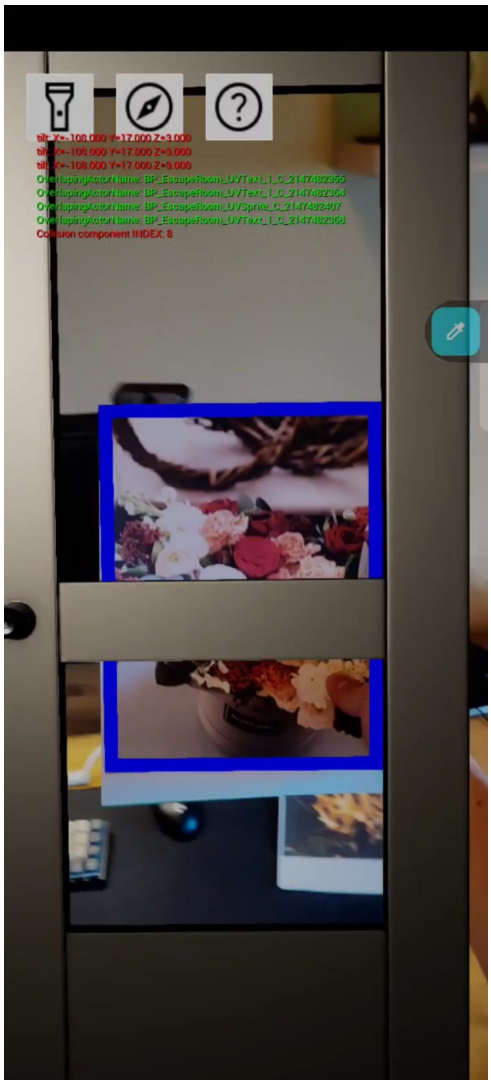

FIGURA 8.47: Escaneig apuntant al nord

8.2.11.1 EscapeRoom_Level_5

8.2.11.1.1 Atributs

```
UPROPERTY(EditDefaultsOnly, Category = "Components", meta = (AllowPrivateAccess = "true"))
UStaticMeshComponent* BaseMesh;
```

Figura principal i que adquireix la forma de porta.

```
UPROPERTY(EditDefaultsOnly, Category = "Components", meta = (AllowPrivateAccess = "true"))
USceneComponent* Scene;
```

Escena a la que vinculem els diferents components.

```
UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Components", meta =
↪  (AllowPrivateAccess = "true"))
FName InventoryItemNeeded;
```

Nom de l'article d'inventari que serà necessari tenir seleccionat per tal de superar el nivell en clicar el pom de la porta.

```
UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Components", meta =
↪  (AllowPrivateAccess = "true"))
class UEscapeRoom_CardinalComponent* CompassComponent;
```

Component de punts cardinals i que s'utilitza per saber si la carta està apuntant al nord.

8.2.11.1.2 Atributs del Blueprint

Box(BoxCollision)→ Capsa de col·lisió situada pom de la porta per detectar el clic i executar un esdeveniment.

8.2.11.1.3 Metodes i Esdeveniment

Constructor

```
AEscapeRoom_Level_5();
```

```
AEscapeRoom_Level_5::AEscapeRoom_Level_5()
{
   // Set this actor to call Tick() every frame.  You can turn this off to improve
    performance if you don't need it.
   PrimaryActorTick.bCanEverTick = true;
   Level = 5;

   Scene = CreateDefaultSubobject<USceneComponent>(TEXT("Scene"));
   RootComponent = Scene;

   BaseMesh = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Base Mesh"));
   BaseMesh->SetupAttachment(Scene);
    // BaseMesh->SetVisibility(ESlateVisibility::Hidden);

   CompassComponent = CreateDefaultSubobject<UEscapeRoom_CardinalComponent>(TEXT("Compass
    Component"));

}
```

Constructor de la classe. Inicialitza l'escena com a *RootComponent* i inicialitza i vingula a l'escena la figura principal, i crea el component de brúixola.

BeginPlay

```
virtual void BeginPlay() override;
```

```
void AEscapeRoom_Level_5::BeginPlay()
{
   Super::BeginPlay();
   SetActorTickEnabled(true);

}
```

Activa el *Ticking* de la classe.

Tick

```
virtual void Tick(float DeltaTime) override;
```

```
void AEscapeRoom_Level_5::Tick(float DeltaTime)
{
   Super::Tick(DeltaTime);

    BaseMesh->SetVisibility(CompassComponent->IsNorth());

}

```

En cada *Frame*, s'assigna la visibilitat de la figura principal en funció de si el dispositiu està apuntant al nord o no.

ManageLevel

```
virtual bool ManageLevel(int32 NewLevel) override;
```

```cpp

bool AEscapeRoom_Level_5::ManageLevel(int32 NewLevel)
{

   if(NewLevel > Level)
   {
      GEngine->AddOnScreenDebugMessage(-1, 10, FColor::Green, FString::Printf(TEXT("WON
↪  GAME!")));
   }
   IsActive = Level == NewLevel;
   SetActorTickEnabled(true);
    SetHints(NewLevel);
   return IsActive;
}
```

Mostra un missatge de partida guanyada si `NewLevel` és superior al nivell propi i assigna les pistes corresponents al nivell actual de la partida. En aquesta funció es podria deshabilitar el propi *Ticking* i el de **CompasComponent** en funció de si el nivell està actiu o no per millorar el rendiment.

SetHints

```cpp
virtual void SetHints(int32 newLevel) override;
```

```cpp

void AEscapeRoom_Level_5::SetHints(int32 newLevel)
{
   Hints.Empty();
   HintIndex = 0;
   if(newLevel < Level)
   {
      Hints.Emplace(TEXT("You are not ready yet."));
   }
   else if(newLevel == Level)
   {
      Hints.Emplace(TEXT("A northern path you seek to find,\nAn object pointing north may
↪  help you align"));
   }
   else if(newLevel > Level)
   {
      Hints.Emplace(TEXT("This level has already been solved."));
   }
}
```

Assigna les pistes en funció del nivell propi textbfLevel i el nou nivell rebut per paràmetre `newLevel`.

8.2.11.1.4 Esdeveniments del Blueprint

On Input Touch End (Box)

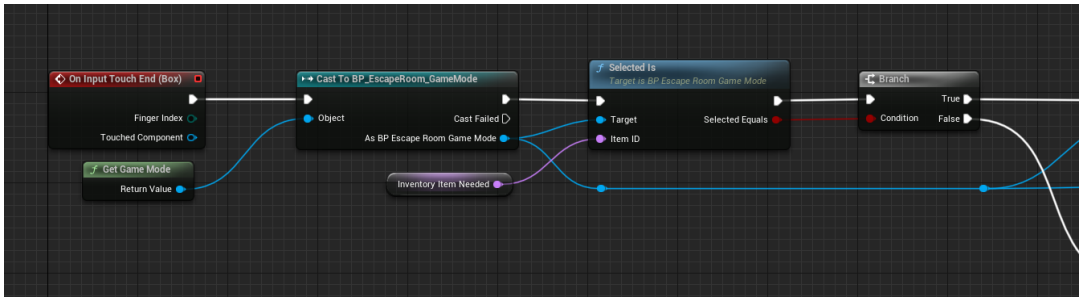

FIGURA 8.48: Implementació del Blueprint On Input Touch End (Box) Part 1

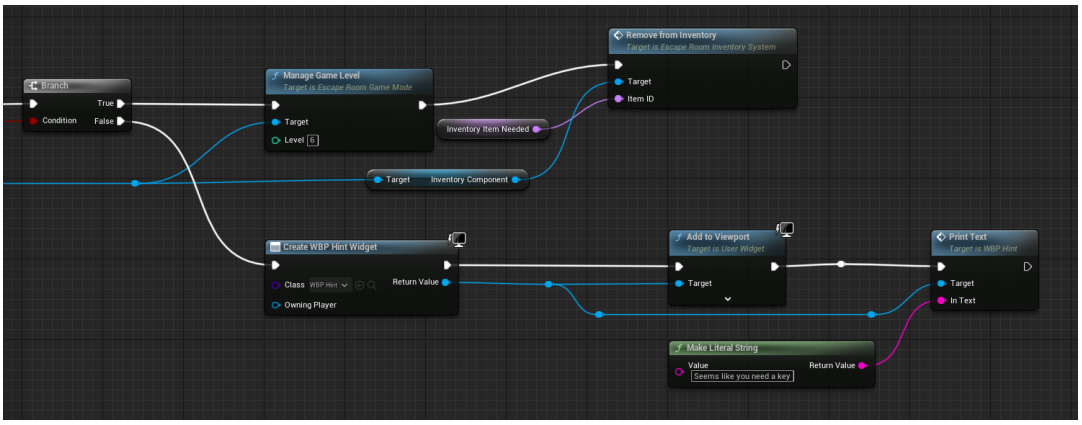

FIGURA 8.49: Implementació del Blueprint On Input Touch End (Box) Part 2

S'obté l'element d'inventari actual seleccionat a través del *GameMode*, que és responsable de l'inventari, i es comprova si l'article és el necessari per a realitzar la operació d'obrir la porta. Veure Figura 8.48.

En cas **afirmatiu**, es notifica al *GameMode* que s'ha superat el nivell i s'elimina l'element de l'inventari, que ja ha sigut utilitzat. En cas **negatiu**, es genera una nova pista que informa al jugador que necessita una clau per a realitzar l'acció. Veure Figura 8.49

8.2.11.2 EscapeRoom_CardinalComponent

El sistema de brúixola està compost de dues parts: el component **EscapeRoom_CardinalComponent** que implementa la lògica d'obtenció del nord i que és un component reutilitzable en els actors que necessitin aquesta informació, i el *Widget* **WBP_Compas**, que mostra en pantalla la **Rosa dels vents** enfocant sempre al nord independentment de la posició del dispositiu. Veure Figures 8.50 i 8.51

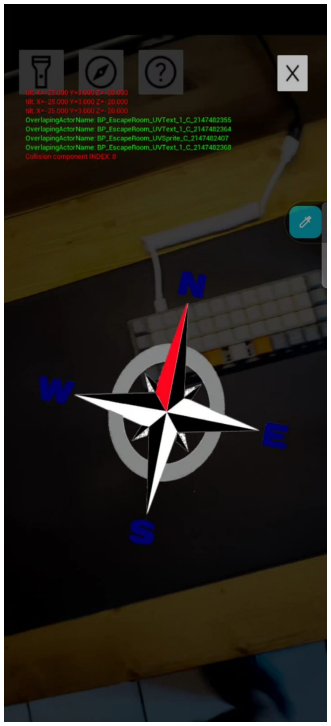

FIGURA 8.50: Visualització de la brúixola amb el mobil apuntant a l'Est

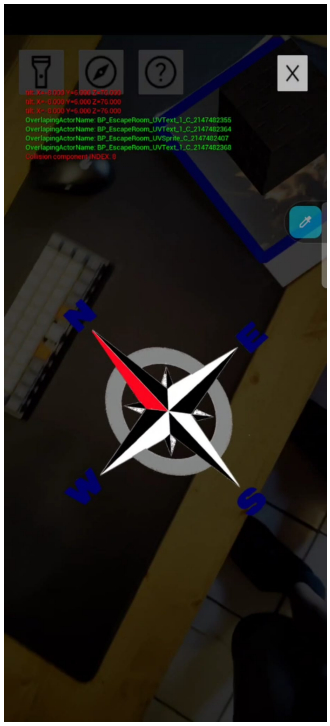

FIGURA 8.51: Visualització de la brúixola amb el mobil apuntant al Nord

8.2.11.2.1 Atributs

```
FVector _tilt;
```

Vector on es guarda la inclinació del dispositiu i que s'utilitza per fer el càlcul corresponent i obtenir el nord.

```
UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Variables", meta =
↪  (AllowPrivateAccess = "true"))
int northAngleLimitUp = 30;
```

```
UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Variables", meta =
↪  (AllowPrivateAccess = "true"))
int northAngleLimitDown = -30;
```

Marges d'error en graus que s'accepta pel nord. Per exemple, com que el nord en graus representat al dispositiu és 0º, es considerarà que apunta al nord des de -30 fins 30º.

```
APlayerController* PlayerControllerRef;
```

Referència al *PlayerController* de la partida. És l'objecte que proporciona les dades necessàries per calcular el nord.

8.2.11.2.2 Mètodes

Constructor

```
UEscapeRoom_CardinalComponent();

```

```
UEscapeRoom_CardinalComponent::UEscapeRoom_CardinalComponent()
{
   // Set this component to be initialized when the game starts, and to be ticked every
↪   frame.  You can turn these features
   // off to improve performance if you don't need them.
   PrimaryComponentTick.bCanEverTick = true;

   // ...
}

```

Constructor del component. S'assigna la permissió d'executar el *Tick*.

BeginPlay

```
virtual void BeginPlay() override;

```

```
void UEscapeRoom_CardinalComponent::BeginPlay()
{
   Super::BeginPlay();
   PlayerControllerRef =  UGameplayStatics::GetPlayerController(GetWorld(), 0);
}

```

S'obté el punter al *PlayerController* de la partida.

IsNorth

```
bool IsNorth(IsNorth);

```

```
bool UEscapeRoom_CardinalComponent::IsNorth()
{
   int northAngle = GetNorthInDegrees();
   return northAngle > northAngleLimitDown && northAngle < northAngleLimitUp;
}

```

Retorna **true** si el nord en graus està entre els límits preestablerts, **false** altrament.

GetNorth

```
UFUNCTION(BlueprintCallable)
int GetNorth();

```

```
int UEscapeRoom_CardinalComponent::GetNorth()
{
   PlayerControllerRef->GetInputMotionState(_tilt, _rotation, _gravity, _acceleration);
    return (int)(_tilt.Z * 100);
}

```

Obté els valors dels sensors d'*InputMotion* i retorna el valor de l'eix Z de la inclinació del dispositiu, guardat en l'atribut **_tilt**. El multiplica per 100 per convertir-lo a *integer* amb una precisió acceptable.

GetNorthInDegrees

```
UFUNCTION(BlueprintCallable)
int GetNorthInDegrees();

```

```
int UEscapeRoom_CardinalComponent::GetNorthInDegrees()
{
   return GetNorth() * -180 / 313;
}

```

El valor de l'eix Z de la inclinació del dispositiu va de 0 a 313 (en els graus 0 a -180) i de 0 a -313 (en els graus de 0 a 180). Amb aquesta operació convertim aquest valor a graus de 0 a 180 i de 0 a -180. D'aquesta manera la podem utilitzar com a graus en un sistema 2d.

8.2.11.3 WBP_Compass

El **WBP_Compass** és el *widget* encarregat de mostrar en pantalla la brúixola, formada per una **Rosa dels vents** que apunta al nord en tot moment. Veure Figura 8.52.

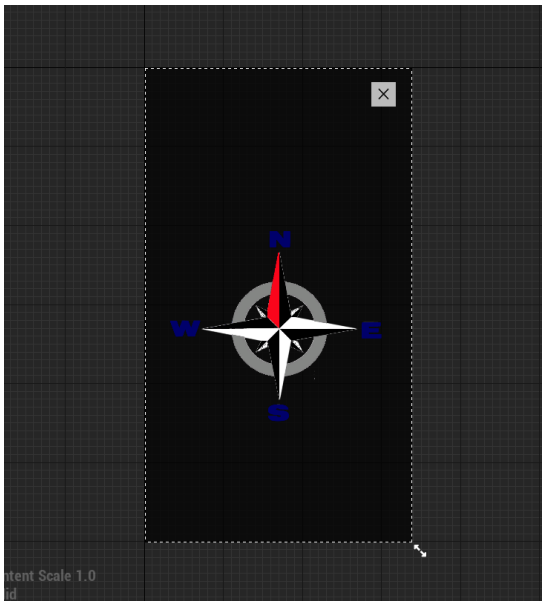

FIGURA 8.52: Disseny del widget WBP_Compass

8.2.11.3.1 Mètodes

Tick

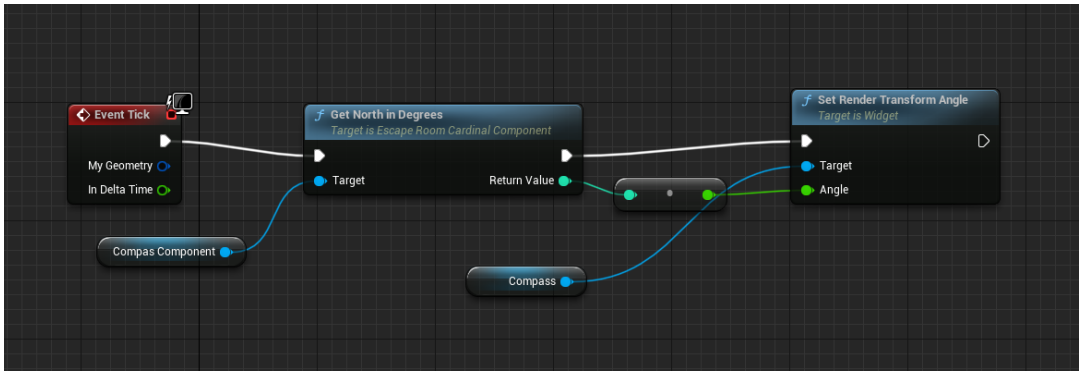

FIGURA 8.53: Implementació del Blueprint Tick

En cada *Frame* consulta al component **UEscapeRoom_CardinalComponent** el nord en graus i aplica una transformació sobre la imatge de la rosa dels vents. Veure Figura 8.53.

OnClicked(Close)

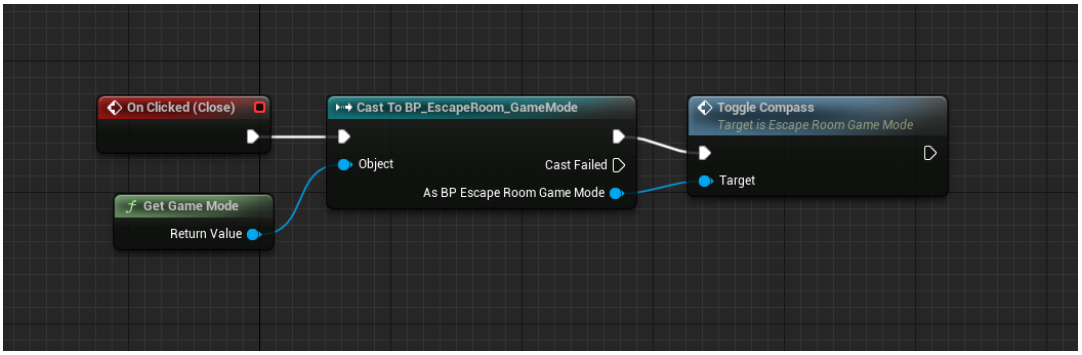

FIGURA 8.54: Implementació del Blueprint OnClicked(Close)

Crida la funció del **GameMode** encarregada d'activar/desactivar la visibilitat del *Widget*. Com que aquest esdeveniment només es pot saltar quan està obert (ja que prové del clic a un botó del propi *widget*), sabem que aquest sempre s'ocultarà. Veure Figura 8.54.

8.2.12 Inventari

8.2.12.1 EscapeRoom_InventorySystem

8.2.12.1.1 Atributs

```
1 UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Components", meta =
  ↪  (AllowPrivateAccess = "true"))
2 int Size;
```

Quantitat d'elements que pot tenir l'inventari i mida del *Grid* que es mostra.

```
1 UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Components", meta =
  ↪  (AllowPrivateAccess = "true"))
2 TMap<FName,int32> Content;
```

Mapa que guarda el contingut de l'inventari amb una relació de nom (key,Fname) → Quantitat(value,int32).

```
1 UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Components", meta =
  ↪  (AllowPrivateAccess = "true"))
2 FName CurrentSelected;
```

Clau de l'element de l'inventari seleccionat.

8.2.12.1.2 Mètodes

Constructor

```
UEscapeRoom_InventorySystem();

```

```
   UEscapeRoom_InventorySystem::UEscapeRoom_InventorySystem()
{
   // Set this component to be initialized when the game starts, and to be ticked every
    frame.  You can turn these features
   // off to improve performance if you don't need them.
   PrimaryComponentTick.bCanEverTick = true;
   Size = 10;

}

```

Es permet el *ticking* i s'inicialitza la mida de l'inventari

AddToInventory

```
UFUNCTION(BlueprintCallable, BlueprintNativeEvent, Category="Inventory")
void AddToInventory(FName ItemID, int Quantity);

```

```
void UEscapeRoom_InventorySystem::AddToInventory_Implementation(FName ItemID, int32
    Quantity)
{

   if(Content.Contains(ItemID))
   {
      *Content.Find(ItemID) += Quantity;
   }
   else
   {
      Content.Add(ItemID, Quantity);
   }
}

```

Implementació de l'esdeveniment que afageix o en suma la quantitat d'un article amb clau `ItemID` i quantitat `Quantity`.

RemoveFromInventory

```
UFUNCTION(BlueprintCallable, BlueprintNativeEvent, Category="Inventory")
void RemoveFromInventory(FName ItemID);

```

```
void UEscapeRoom_InventorySystem::RemoveFromInventory_Implementation(FName ItemID)
{
   if(Content.Contains(ItemID))
   {
      *Content.Find(ItemID) -= 1;
      if(*Content.Find(ItemID) <= 0)
      {
         Content.Remove(ItemID);
      }
   }
}
```

Implementació de l'esdeveniment que esborra un article amb identificador `ItemID` de l'inventari. En cas que la nove quantitat sigui inferior o igual a 0, l'elimina del mapa **Content**.

SetCurrentSelected

```
UFUNCTION(BlueprintCallable, BlueprintNativeEvent, Category="Inventory")
void SetCurrentSelected(FName ItemID);
```

```
void UEscapeRoom_InventorySystem::SetCurrentSelected_Implementation(FName ItemID)
{
   CurrentSelected = ItemID;
}
```

S'assigna l'identificador `ItemID` en el **CurrentSelected**, atribut que guarda l'identificador de l'article seleccionat actual.

GetCurrentSelected

```
UFUNCTION(BlueprintCallable, Category="Inventory")
FName GetCurrentSelected() const;
```

```
FName UEscapeRoom_InventorySystem::GetCurrentSelected() const
{
   return CurrentSelected;
}
```

Funció per obtenir l'identificador d'article seleccionat actual.

LogInventory

```
UFUNCTION(BlueprintCallable, Category="Inventory")
void LogInventory() const;
```

```
void UEscapeRoom_InventorySystem::LogInventory() const
{
   for (const TPair<FName, int32>& pair : Content)
   {
      GEngine->AddOnScreenDebugMessage(-1, 10.0f, FColor::Purple,
↪  FString::Printf(TEXT("INVENTORY CONTENT KEY: %s"), *(pair.Key.ToString())));
      GEngine->AddOnScreenDebugMessage(-1, 10.0f, FColor::Purple,
↪  FString::Printf(TEXT("INVENTORY CONTENT VALUE: %i"), pair.Value));

   }
}
```

Funció per mostrar per pantalla el contingut de l'inventari en mode *debug*.

8.2.12.2 EscapeRoom_InventoryItem

Classe principal que dona representació física al videojoc d'un element de l'inventari

8.2.12.2.1 Atributs

```
UPROPERTY(EditDefaultsOnly, Category = "Components", meta = (AllowPrivateAccess = "true"))
USceneComponent* Scene;
```

Component d'escena per donar posició i moviment a l'actor.

```
UPROPERTY(BlueprintReadWrite, Category = "Components", meta = (AllowPrivateAccess =
↪  "true"))
UStaticMeshComponent* BaseMesh;
```

Figura principal que dona aspecte físic a l'actor.

```
UPROPERTY(EditDefaultsOnly, BlueprintReadWrite, Category = "Components", meta =
↪  (AllowPrivateAccess = "true"))
FDataTableRowHandle ItemID;
```

Identificador a una fila de la taula on emmagatzem els articles d'inventari i que assignem en els defaults de la classe des de l'editor d'Unreal Engine.

8.2.12.2.2 Mètodes

Constructor

```
AEscapeRoom_InventoryItem();
```

```
AEscapeRoom_InventoryItem::AEscapeRoom_InventoryItem()
{
   // Set this actor to call Tick() every frame.  You can turn this off to improve
   ↪ performance if you don't need it.
   PrimaryActorTick.bCanEverTick = true;

   Scene = CreateDefaultSubobject<USceneComponent>(TEXT("Scene"));
   RootComponent = Scene;

   BaseMesh = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Base Mesh"));
    BaseMesh->SetupAttachment(Scene);
    //BaseMesh->bDrawMeshCollisionIfSimple = true;
    //BaseMesh->bDrawMeshCollisionIfComplex = true;
}
```

Assignem l'escena com a component arrel i hi vinculem la figura principal.

8.2.12.3 ItemStruct

Structure d'Unreal Engine que definim en l'editor d'Unreal i que serà un registre d'una taula on es guardaran tots els possibles articles de l'inventari.

Els camps que té l'struct son:

- **Name** (FName) → Nom identificador de l'article.
- **Description** (Text) → Descripció de l'article.
- **Thumbnail** (Texture 2D) → Icona que es mostra en visualitzar l'inventari.
- **ItemClass** (Actor) → Actor al que fa referencia l'element d'inventari
- **StackSize** (Integer) → Quantitat màxima d'articles acumulables.

Aquesta estructura està preparada pel treball futur, per poder generar a la partida l'actor al que està relacionat l'article. És per això que que hi ha el camp **ItemClass**. Veure Figura 8.55.

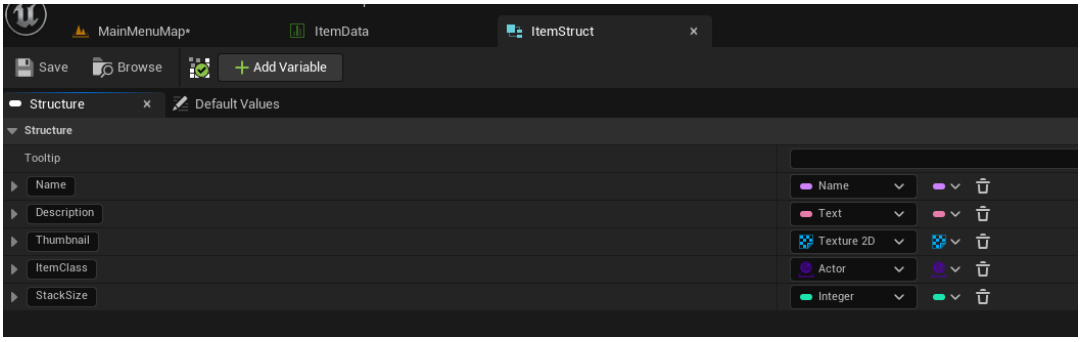

FIGURA 8.55: Estructura ItemStruct

8.2.12.4 ItemData

Data Table, estructura d'Enreal Engine, on cada registre de la taula és una **Structure** de **ItemStruct**. A aquesta taula és on definirem tots els elements d'inventari i els assignarem l'icona que els representarà. Veure Figura 8.56.

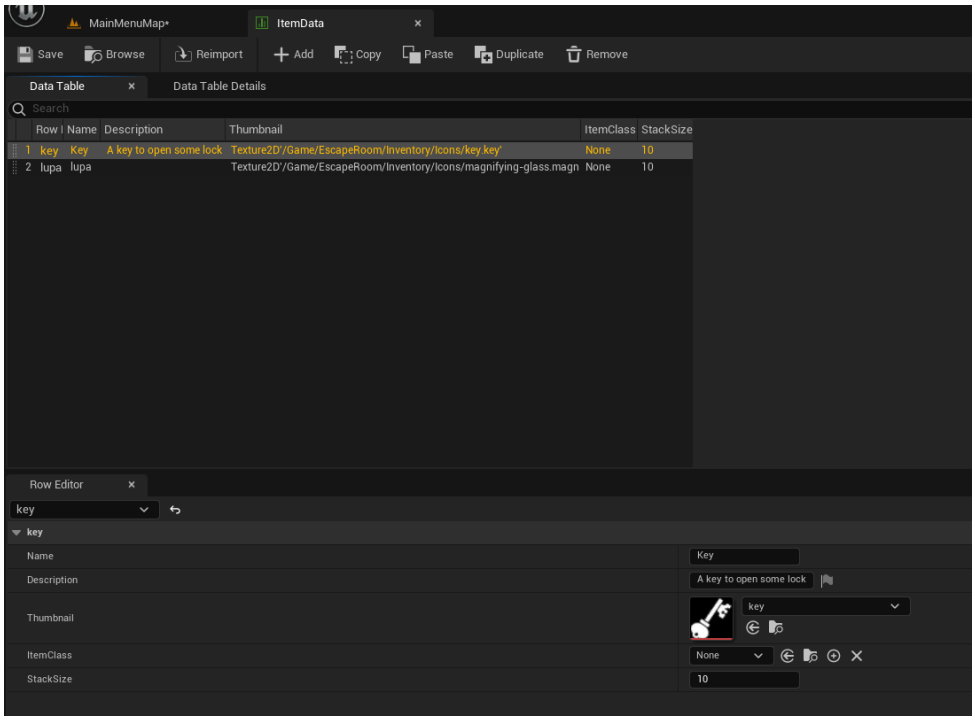

FIGURA 8.56: Visualització del configurador de ItemData

8.2.12.5 Inventory UI

8.2.12.5.1 InventoryMenu

Widget que conté un **InventoryGrid** (explicat en el següent apartat). Reserva l'espai per aquest i inclou el botó de tancar l'inventari. Veure Figura 8.57.

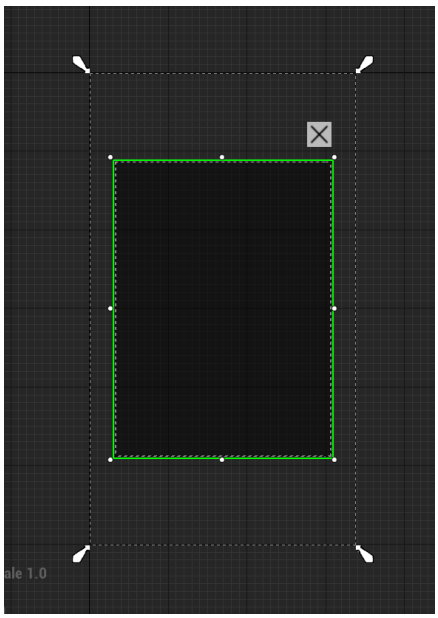

FIGURA 8.57: Disseny del Widget InventoryMenu

És el receptor d'esdeveniments que requereixen actualitzar la visualització de l'inventari. Aquests son quan:

- Es selecciona un article de l'inventari.

- Cal actualitzar quin article es veu seleccionat.
- Es sol·licita la visualització de l'inventari complet.

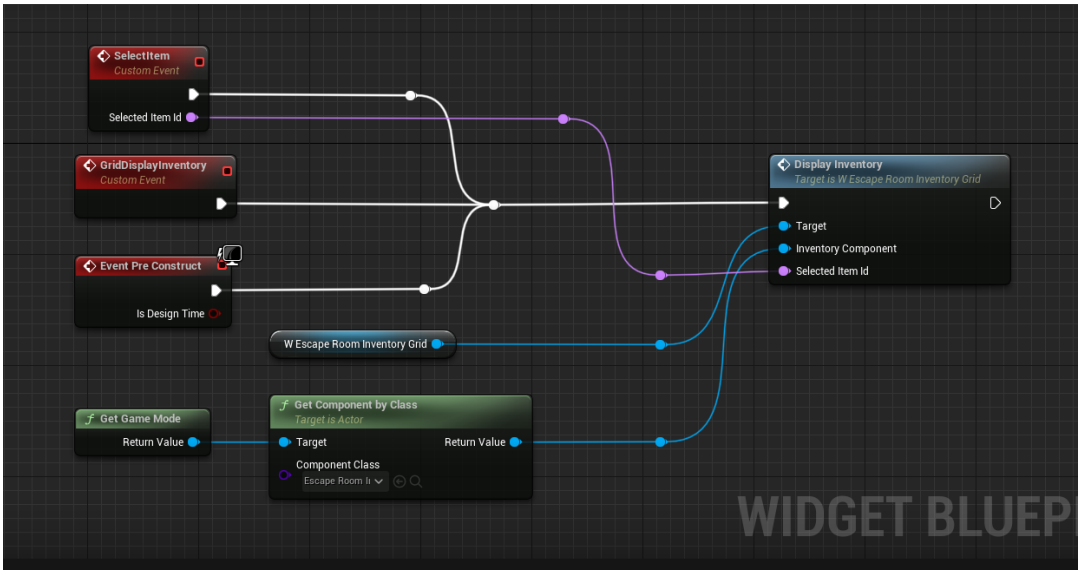

FIGURA 8.58: Implementació dels Blueprints SelectItem, GridDisplayINventory i EventPreConstruct

Es crida l'esdeveniment encarregat de renderitzar el *grid* amb els elements d'inventari. Veure Figura 8.58.

8.2.12.5.2 InventoryGrid

Widget que mostra, element a element de l'inventari, en un Grid. Cada element està representat en un **InventorySlot** (explicat en el següent apartat).

L'**InventoryGrid** s'encarrega de recorre cada element de l'atribut **Content** de l'**EscapeRoom_InventorySystem** que conté el **GameMode**, en genera un InventorySlot, utilitzat per visualitzar un únic article d'inventari, i l'afageix en una estructura de tipus WrapBox.

L'inventari s'ha de mantenir actualitzat, per fer-ho, és crea de nou tot el *grid* cada cop que el volem visualitzar o que rep un canvi (com per exemple, seleccionar un element). Això es realitza amb el mètode **DisplayInventory**.

Les variables que trobem en el widget son:

- **InventoryComponent** (EscapeRoom_InventorySystem) → Referència al component inventari que assignarem amb un punter a l'inventari del GameMode.
- **LocalSelectedItem** (FName) → Nom identificador de l'article d'inventari.
- **BOX_Grid** (Wrap Box) → Capça on s'afageixen els *Slots* i es distribueixen en forma de *grid*.

El mètode que implementa la construcció del *grid* mostra a la figura 8.59:

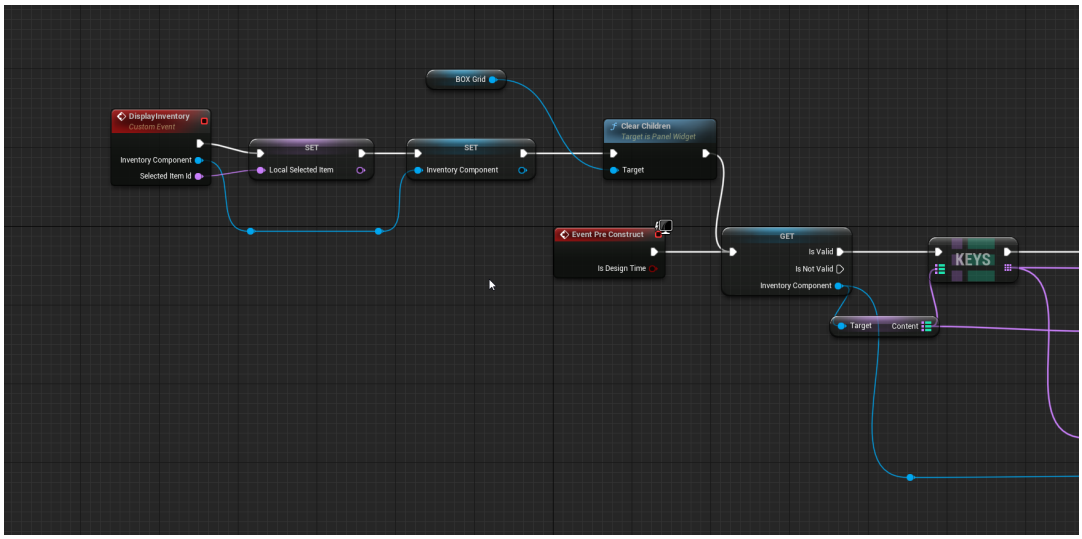

FIGURA 8.59: Implementació del Blueprint DisplayInventory Part 1

En la primera part de la implementació es veu l'assignació de la variabla **LocalSelectedItem** i el component d'inventari **InventoryComponent**. Seguidament s'aplica el mètode **ClearChildren** del **BoxGrid**, que esborra tots els fills (en aquest cas, *Slots* d'un element d'invenari) del *Wrap Box*. Es veu també l'obtenció en forma de llista de les claus de l'atribut **Content** de l'InventoryComponent. Aquesta llista conté totes les claus dels elements presents en l'inventari. Veure Figura 8.59.

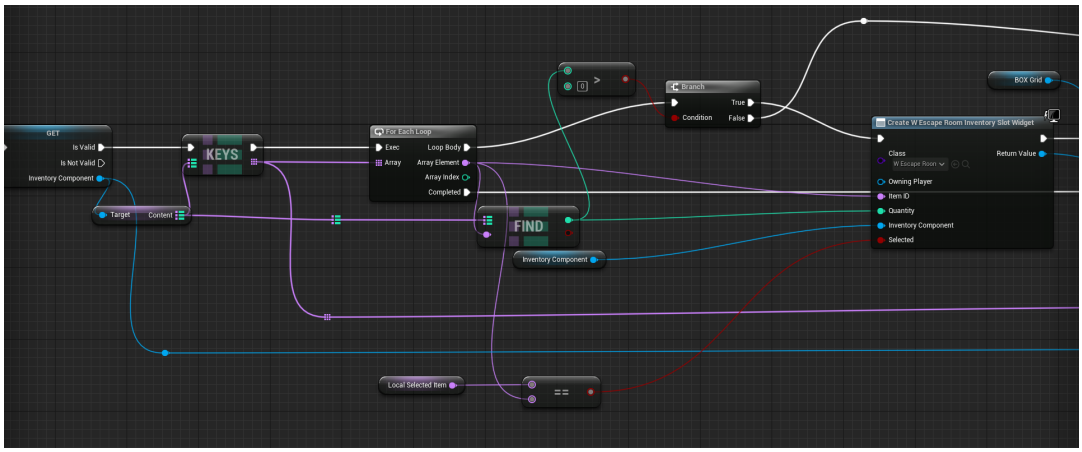

FIGURA 8.60: Implementació del Blueprint DisplayInventory Part 2

Per cada element de la llista, s'obté la seva quantitat accedint al mapa i, en cas que sigui superior a 0, es crea un widget de classe **W_EscapeRoom_InventorySlot**, que serà l'encarregat de mostrar un element amb la seva quantitat. Per crear-lo requereix el nom identificador **ItemID**, la quantitat **Quantity** i si és o no l'element seleccionat. Veure Figura 8.60.

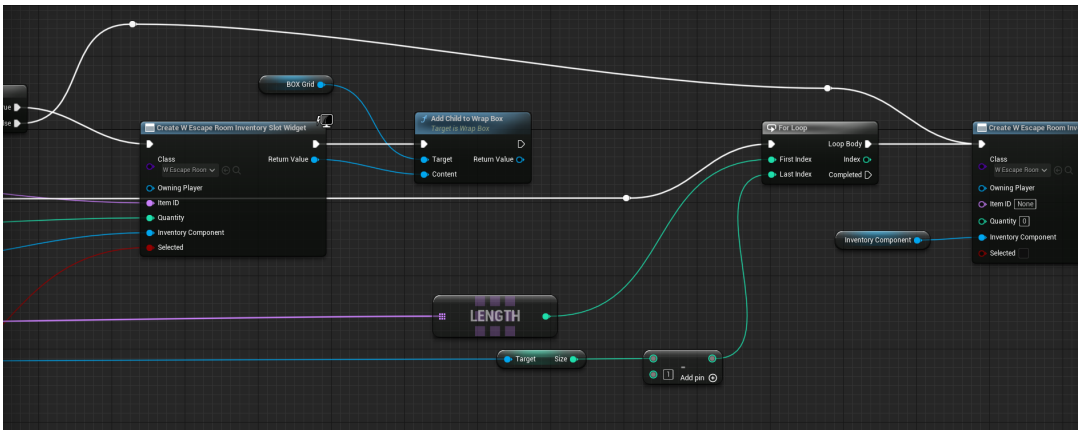

FIGURA 8.61: Implementació del Blueprint DisplayInventory Part 3

I finalment s'afageix el *widget* creat dins el **BOXGrid** utilitzant el mètode del Wrap Box: **AddChildToWrapBox**. Veure Figura 8.61.

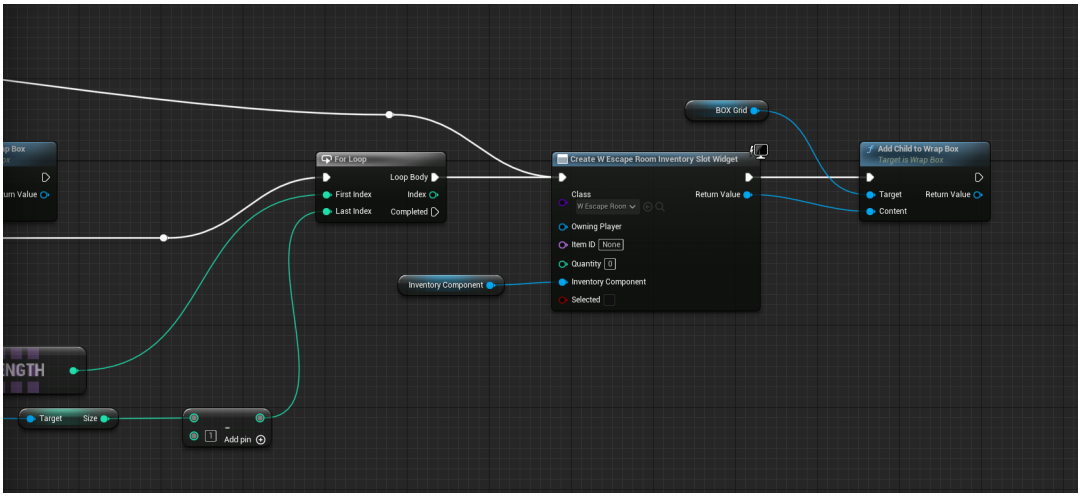

FIGURA 8.62: Implementació del Blueprint DisplayInventory Part 4

La resta de la funció s'encarrega de completar l'inventari des de N elements fins a Size per tal de que sempre es visualitzin totes les caselles d'aquest, sigui buit o ple. Veure Figura 8.62.

8.2.12.5.3 InventorySlot

Finalment en el sistema d'inventari tenim l'InventorySlot. És el widget encarregat de mostrar la icona associada a un ItemID, mostrar-ne la quantitat, i crear l'efecte de seleccionat. També detecta el clic de l'article i ho notifica al GameMode. Veure Figura **??**.

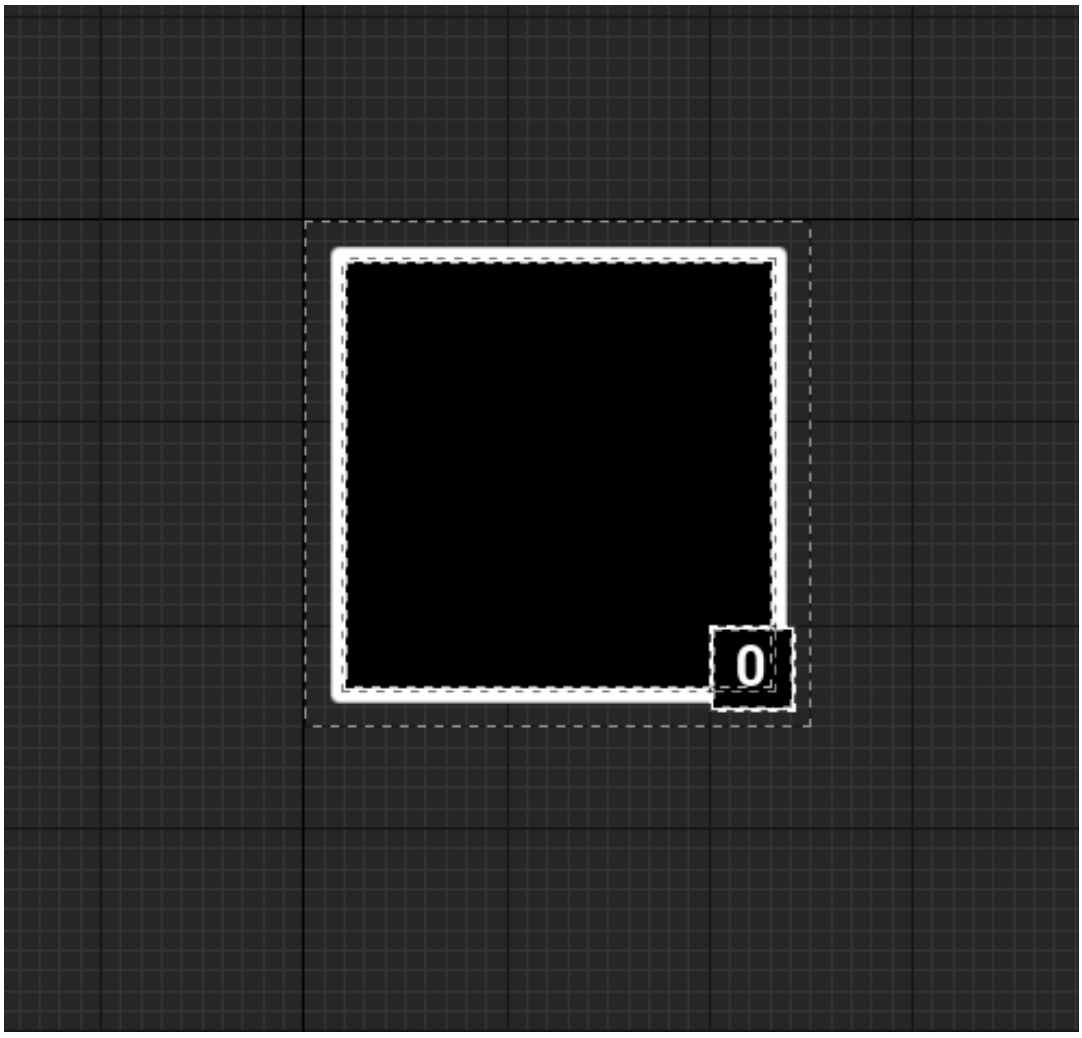

FIGURA 8.63: Disseny del Widget W_EscapeRoom_InventorySlot

Consta de les següents variables, que s'inicialitzen en la creació:

- **ItemID** (FName) → Per obtenir la icona associada.
- **Quantity** (Integer) → Quantitat a mostrar.
- **Selected** (Boolean) → True si està seleccionat, false altrament.

Els mètodes que implementa són:

OnClickEvent

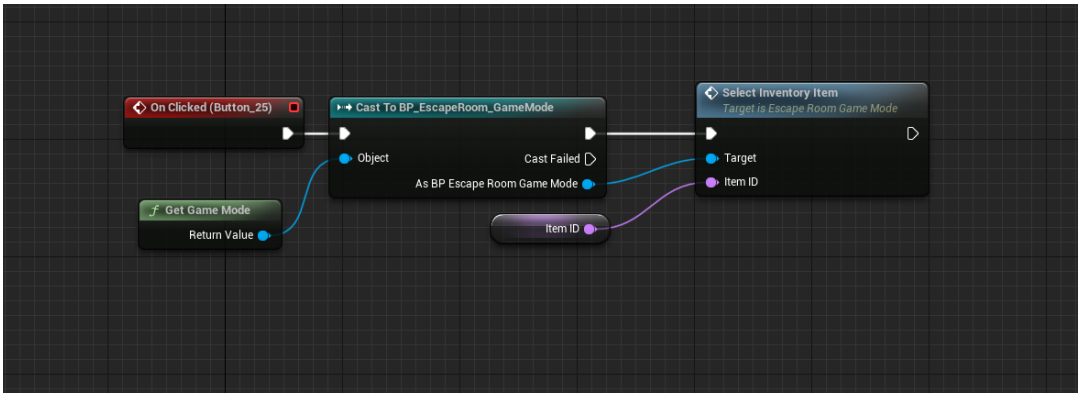

FIGURA 8.64: Implementació del Blueprint OnClickEvent(Button)

Esdeveniment en clicar un slot. Es crida la funció del GameMode **SelectInventoryItem** i es passa l'**ItemID** de l'Slot en qüestió com a paràmetre. Veure Figura 8.64.

EventPreConstruct

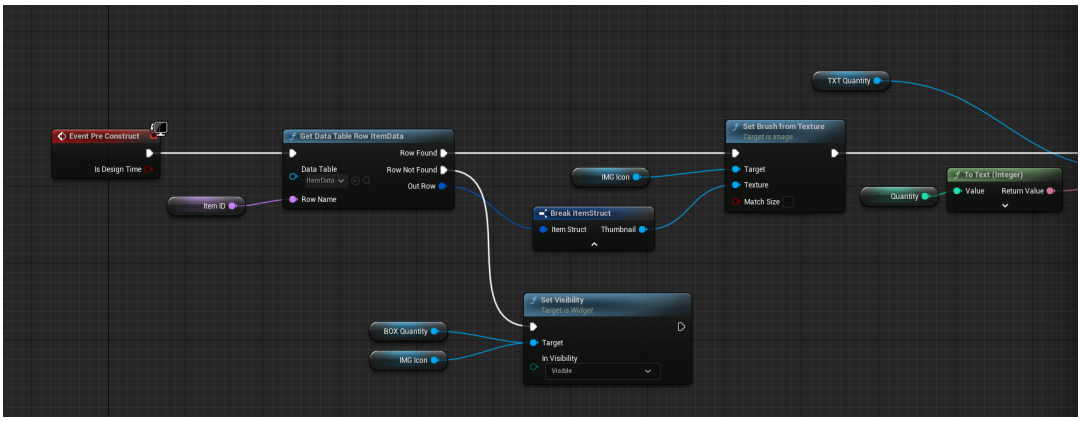

FIGURA 8.65: Implementació del Blueprint EventPreConstruct Part 1

En la primera part del mètode, es veu l'obtenció de la icona amb el camp de l'*ItemStruct* **Thumbnail** utilitzant el mètode **GetDataTableRow** de la taula explicada anteriorment **ItemData** i assignació d'aquesta icona a la variable de widget **IMG icon**. En cas que no es trobi l'article en la taula **ItemData**, es crea un Slot buit. Veure Figura 8.65.

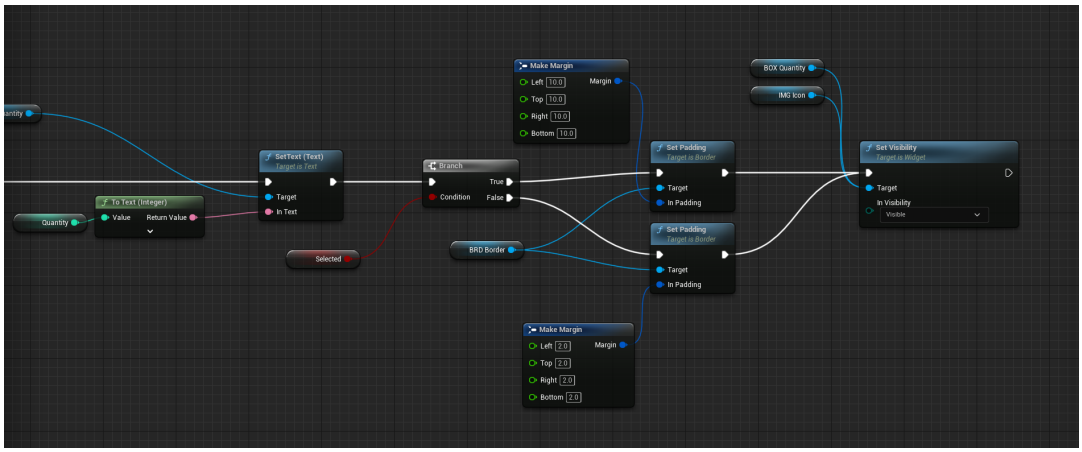

FIGURA 8.66: Implementació del Blueprint EventPreConstruct Part 2

En la segona part del mètode, s'assigna la quantitat com a text a la variable de widget **TXT Quantity**, i s'assigna dinàmicament el gruix d'un border en funció de la variable **Selected** per fer l'afecte d'article seleccionat. Veure Figura 8.66 per implementació. Figures 8.67 i 8.68 per resultats.

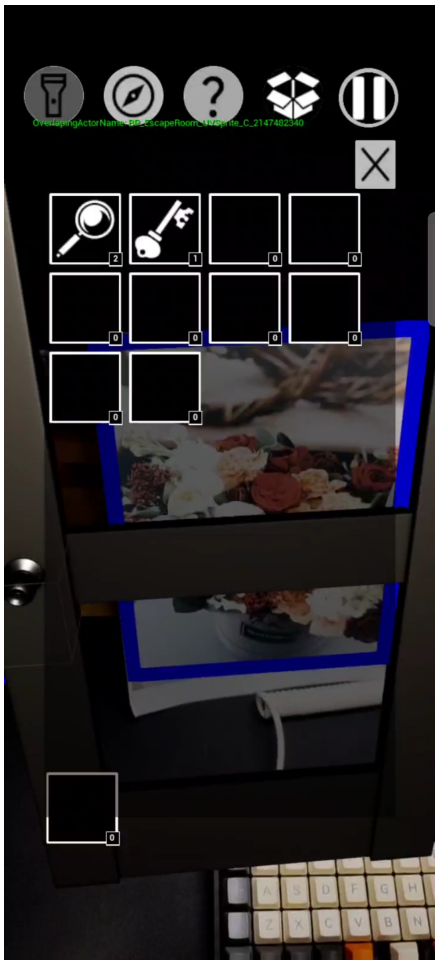

FIGURA 8.67: Inventari

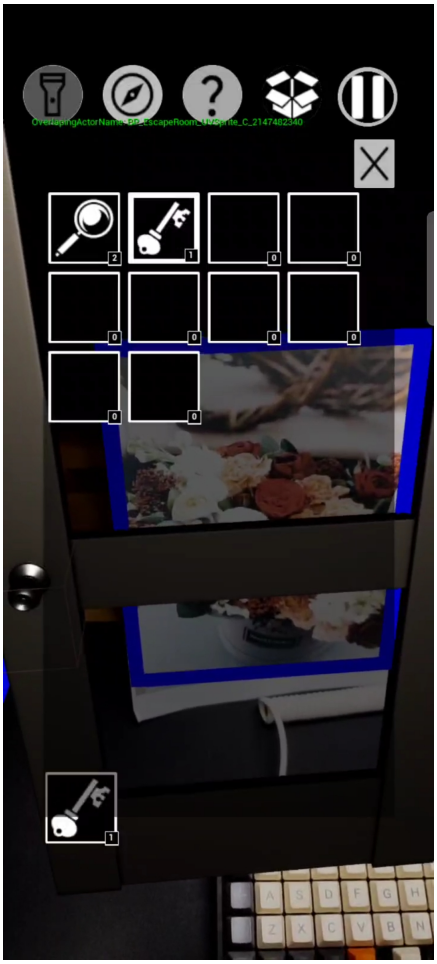

FIGURA 8.68: Inventari amb un *Item* seleccionat

8.2.13 Sistema de pistes

El sistema de pistes és un sistema dissenyat per tal que el jugador pugui sol·licitar ajuda en tot moment de la partida i aquesta estigui personalitzada en funció del nivell que està o del l'objecte virtual al que està enfocant. Aquesta ajuda ve donada en forma de rodolí, que es mostra en pantalla com un dialeg que es va escrivint lletra a lletra. Veure Figures 8.69, 8.70 i 8.71.

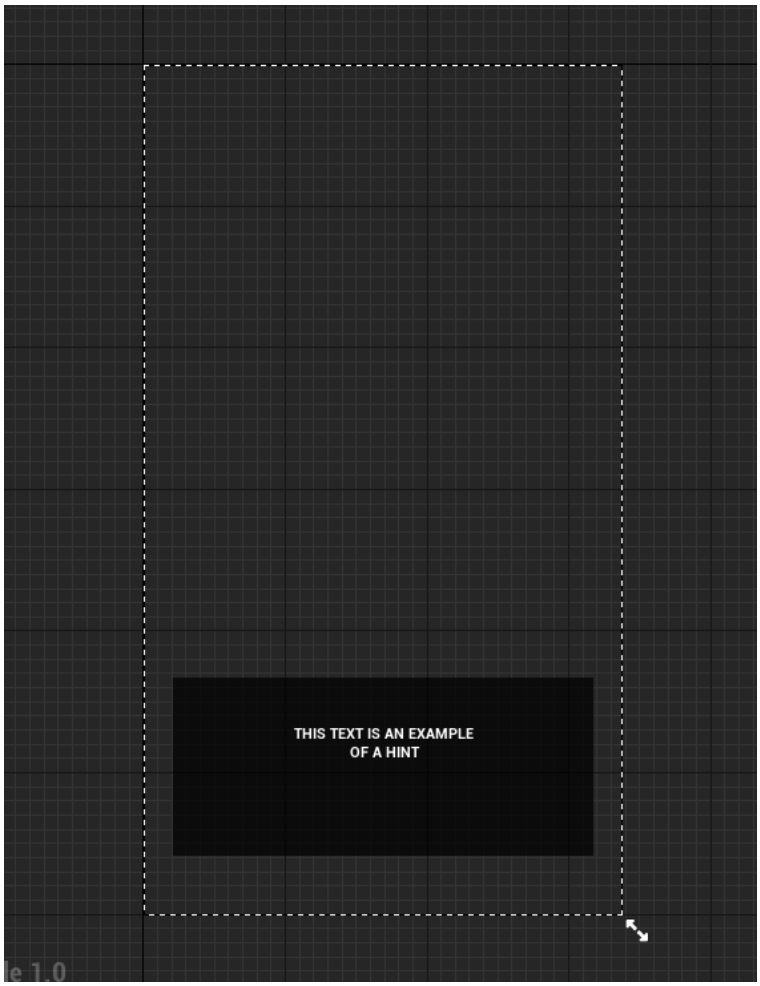

FIGURA 8.69: Disseny del Widget WBP_Hint

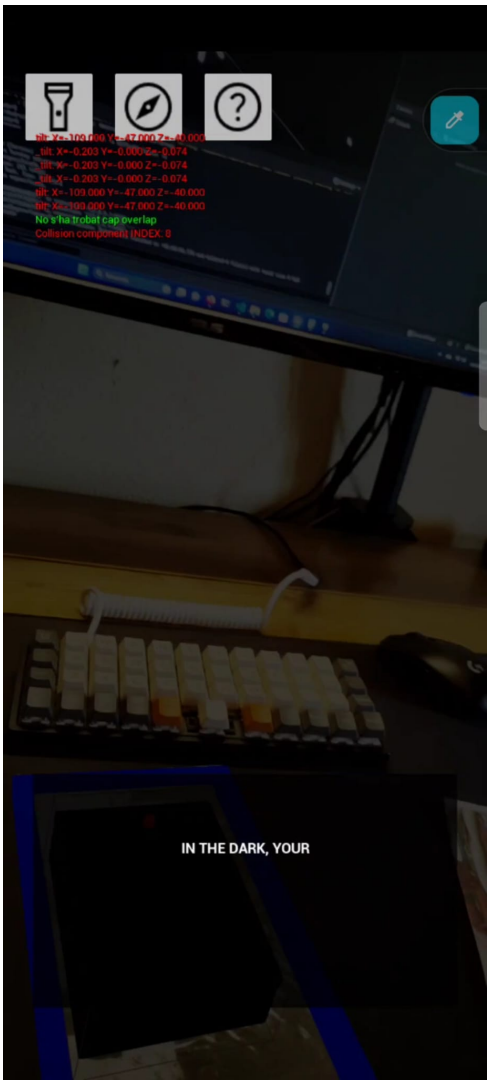

FIGURA 8.70: Visualització pista escrivint-se

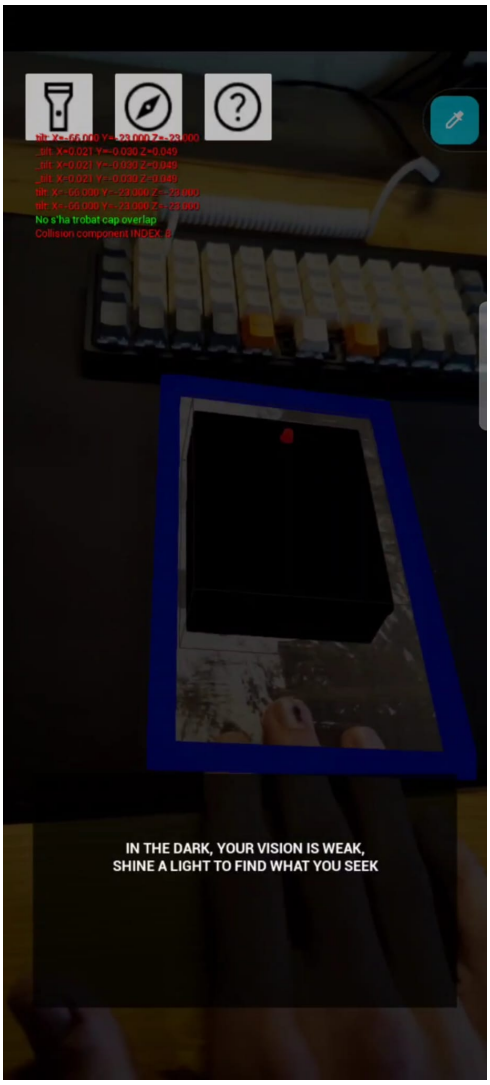

FIGURA 8.71: Visualització pista completa

Per fer-ho, es va dissenyar una interfície anomenada **I_EscapeRoom_Dialog**, les classes les quals l'implementin hauran de desenvolupar la lògica.

En el projecte, les classes que implementen **I_EscapeRoom_Dialog** son les següents:

- **EscapeRoom_GameMode** → és l'engarregat de proveir una pista personalitzada pel nivell que s'està intentant superar, en cas que no s'estigui enfocant a cap objecte que implementi **I_EscapeRoom_Dialog**.
- **EscapeRoom_Level** → per tal que cada nivell pugui tenir les seves pròpies pistes. D'aquesta manera, la implementació de les funcions a implementar de **I_EscapeRoom_Dialog** que es poden reutilitzar ja s'hereten en les classes especifiques de nivells concrets.

8.2.13.1 I_EscapeRoom_Dialog

Interfície que implementen aquells actors als quals es pot sol·licitar una pista. Els següents mètodes son els que cal que les classes que implementen **I_EscapeRoom_Dialog** continguin. D'aquesta manera, quan el GameMode rep la sol·licitud d'ajuda, si l'objecte sobre el que es demana ajuda implementa la interfície, pot cridar aquests mètodes amb seguretat.

8.2.13.1.1 Metodes

ShowHint

```
UFUNCTION(BlueprintCallable, BlueprintNativeEvent, Category="Dialog")
void ShowHint();

```

HideHint

```
UFUNCTION(BlueprintCallable, BlueprintNativeEvent, Category="Dialog")
void HideHint();

```

8.2.13.1.2 Metodes

ShowHint - EscapeRoom_GameMode

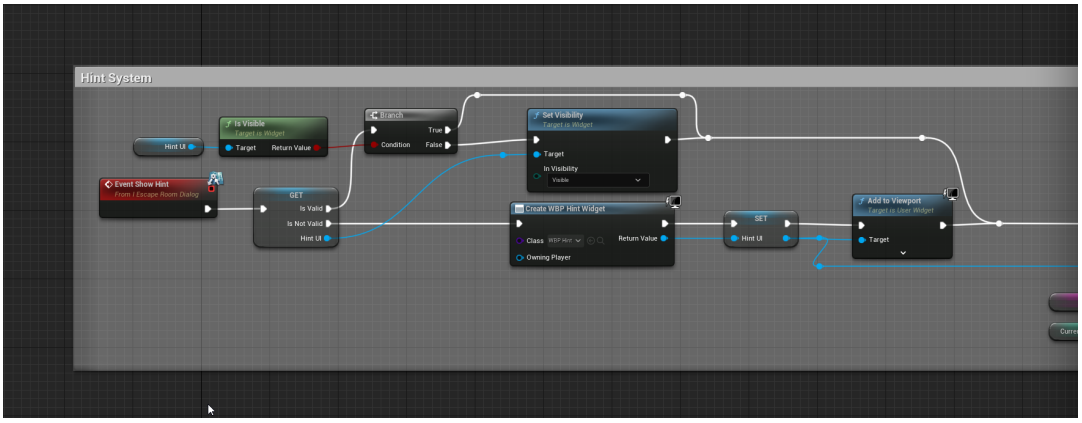

FIGURA 8.72: Implementació del Blueprint ShowHint Part 1

Creem el widget WBP_Hint i en guarden la referencia. Veure Figura 8.72.

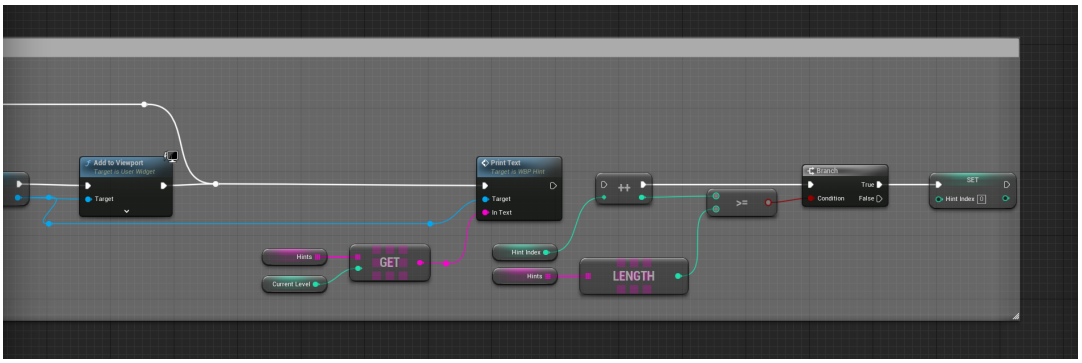

FIGURA 8.73: Implementació del Blueprint ShowHint Part 2

Es crida el mètode del **PrintText** del WBP_Hint, que escriu el missatge en l'espai reservat, passant per paràmetre el text amb la pista corresponent al nivell. Veure Figura 8.73.

ShowHint - EscapeRoom_Level

```
void AEscapeRoom_Level::ShowHint_Implementation()
{
   //GEngine->AddOnScreenDebugMessage(-1, GWorld->DeltaTimeSeconds, FColor::Yellow,
 ↪  FString::Printf(TEXT("SHOWING LEVEL 2 HINT")) );
   if(IsValid(HintWidgetClass))
   {
      if(HintWidget == nullptr)
      {
         HintWidget = Cast<UHintWidget>(CreateWidget(GetWorld(), HintWidgetClass));
         if(HintWidget != nullptr)
         {
            HintWidget->AddToViewport();
            HintWidget->PrintText(Hints[HintIndex]);
            HintIndex++;
            if(HintIndex >= Hints.Num())
            {
               HintIndex = 0;
            }
         }
      }
      else{
         HintWidget->SetVisibility(ESlateVisibility::Visible);
         HintWidget->PrintText(Hints[HintIndex]);
         if(HintIndex >= Hints.Num())
         {
            HintIndex = 0;
         }
      }
   }

}

```

Es crea un *Widget* de tipus **HintWidgetClass** (assignat en l'editor des dels defaults de la classe), l'afegim en pantalla i cridem el mètode del **PrintText** del WBP_Hint, que escriu el missatge en l'espai reservat, passant per paràmetre el text amb la pista corresponent al nivell. En aquest cas sí que es manté una variable amb l'índex de la pista mostrada fins que arriba a la mida de la llista de pistes, que l'assignem a 0 de nou per tal de tenir múltiples pistes per nivell i que, en acabar-se, torni a mostrar des de la primera.

8.2.13.2 WBP_Hints

Widget encarregat de mostrar per pantalla una pista, fent un efecte d'escriure's al moment.

Les variables del widget són les següents, tenint en compte que la funció que mostra el text es crida recursivament fins a mostrar tota la seqüencia de caràcters:

- **HintText** (Text) → Variable de Widget que mostra un text per pantalla.
- **LetterCount** (Integer) → Index de caràcter que s'està tractant.
- **InText** (String) → String amb la pista a mostrar.
- **CurrentText** (String) → Text que s'ha mostrat.
- **NextLetter** (String) → Següent lletra a tractar.

PrintText

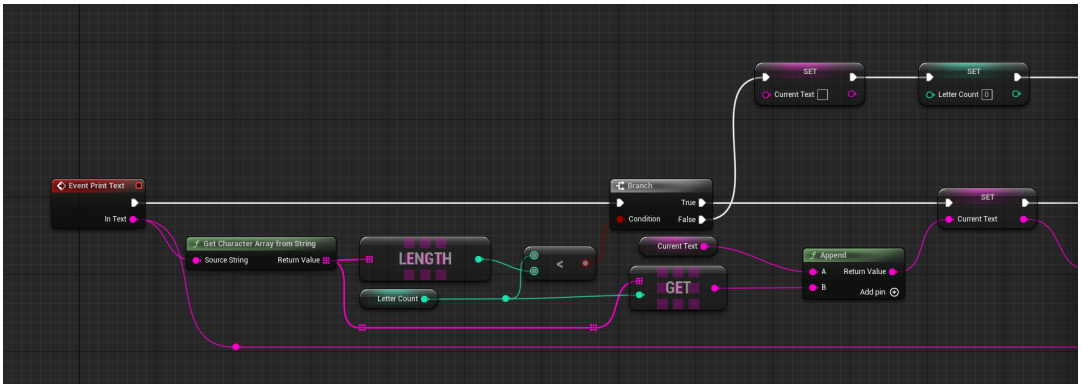

FIGURA 8.74: Implementació del Blueprint PrintText WBP_Hint Part 1

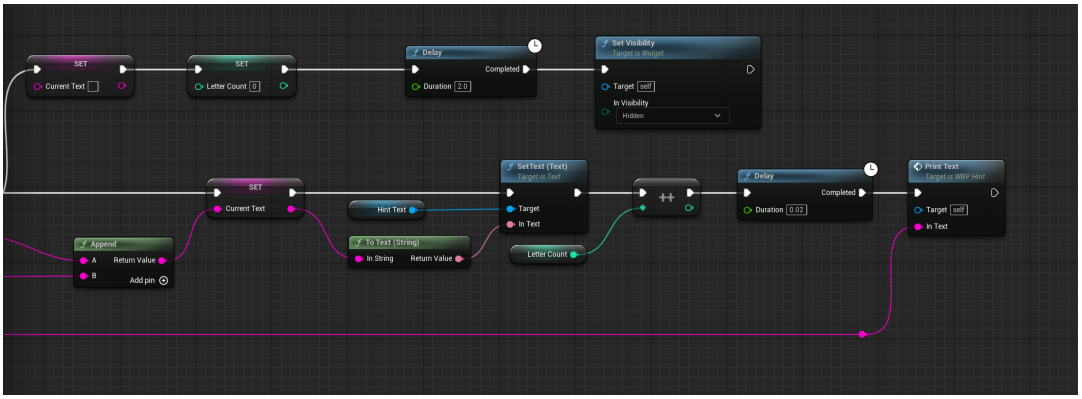

FIGURA 8.75: Implementació del Blueprint WBP_Hint Part 2

Crida recursiva on cada iteració s'obté una nova lletra de la pista fins recórrer-la tota. Aquesta lletra s'afageix al la variable de widget **HintText** i un cop mostrada s'aplica un *Delay* per crear un afecte on s'escriu el missatge lletra a lletra. Veure Figures 8.74 i 8.75.

8.2.14 Sol·licitud de pistes

Una de les parts interessants del sistema de pistes és que, depenent de l'objecte virtual s'enfoqui, pot mostrar pistes personalitzades a ell. En un joc tradicional, l'objecte que ofereix la pista tindria una capsa de col·lisió i el personatge de la partida detectaria la proximitat amb l'objecte a través d'aquesta capsa. En el projecte, s'ha fet d'aquesta mateixa manera, amb la diferencia que el jugador no és un personatge virtual, i per tant la proximitat i l'enfocament ha de ser el del dispositiu real. Per fer-ho, el *Pawn* té una capsa de col·lisió enfocada en la direcció que enfoca la càmera amb origen al propi dispositiu.

El següent esdeveniment del *Pawn* n'implementa la lògica, veure Figures 8.76, 8.77 i 8.78.

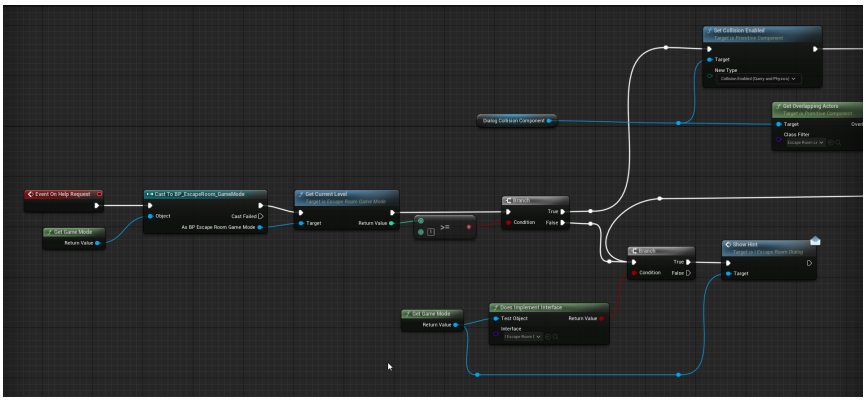

FIGURA 8.76: Implementació del Blueprint OnHelpRequest Part 1

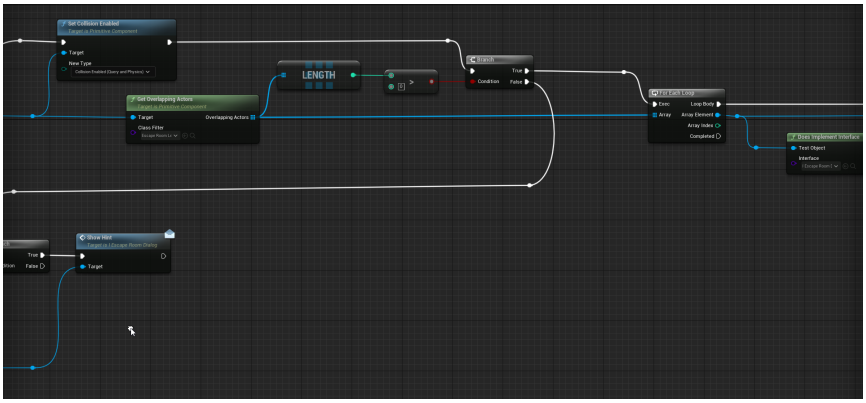

FIGURA 8.77: Implementació del Blueprint OnHelpRequest Part 2

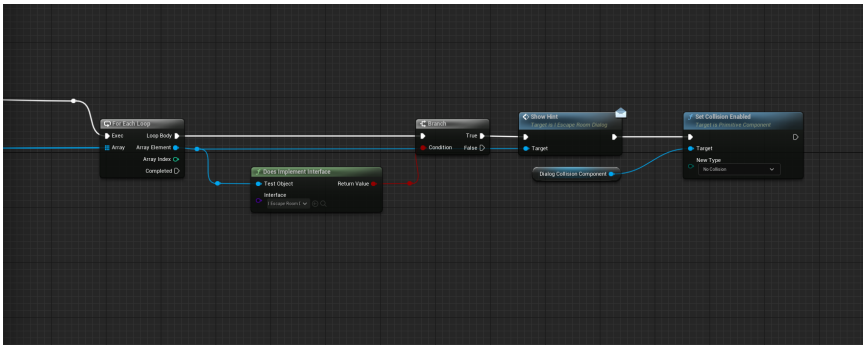

FIGURA 8.78: Implementació del Blueprint OnHelpRequest Part 3

Obté el nivell actual de la partida, i, en cas que s'estigui al nivell 0, mostra la pista del GameMode ja que el Nivell 0 no implementa l'interface Dialog. Altrament, obté els actors que està col·lisionant el component de col·lisió col·locat a la càmera i que detecta quin nivell s'està enfocant. En cas que no es detecti cap col·lisió, es mostra la pista del GameMode pel nivell actual. Si hi ha col·lisió i l'actor implementa la interfície I_EscapeRoom_Dialog, es mostra la pista pròpia de l'actor. Veure Figures 8.76, 8.77 i 8.78.

Capítol 9

Proves i Resultats

9.1 Proves Realitzades

Les proves s'han realitzat a un dispositiu Android, concretament el One Plus 9, ja que és l'únic entorn de desenvolupament que permetia l'execució de l'aplicació. Durant tot el període d'implementació de les mecàniques es van realitzar proves per comprovar el correcte funcionament, amb l'ajuda de funcions de *debug* d'Unreal Engine com:

- **GEngine->AddOnScreenDebugMessage(...);** per visualitzar missatges en partida.
- **DrawDebugSphere(...);** per visualitzar el punt d'impacte en col·lisions.
- **DrawDebugLine();** en casos on s'ha realitzat *LineTrace*.
- **DebugDrawTrackedGeometry(...)** per les geometries d'ArCore detectades.

Algunes d'aquestes ajudes encara son presents al projecte i es poden veure en les imatges de la partida. Veure Figura 9.1.

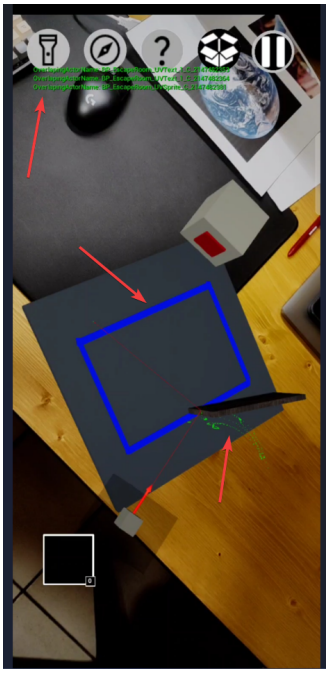

FIGURA 9.1: Exemple d'eines de debug utilitzades

9.2 Resultats

FIGURA 9.2: Inici de l'aplicació

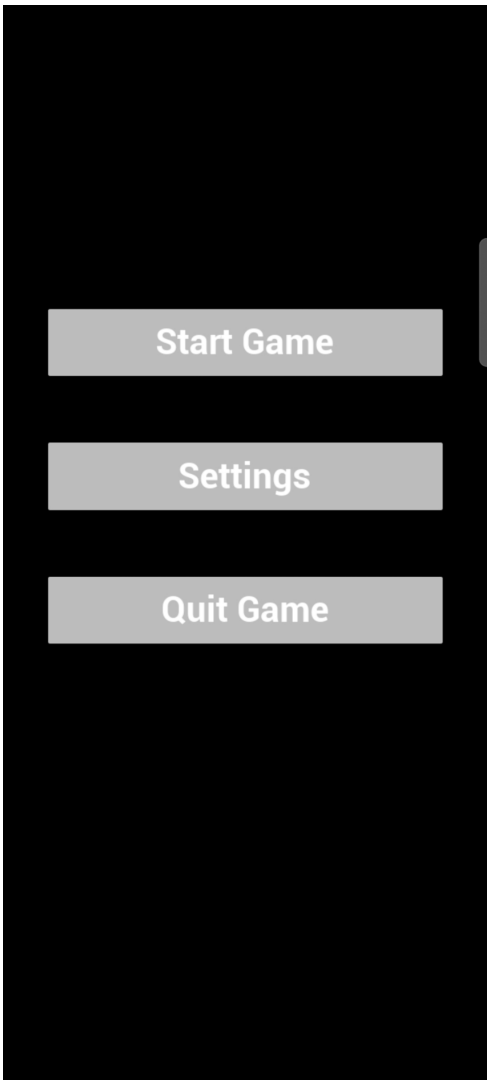

FIGURA 9.3: Menú d'inici

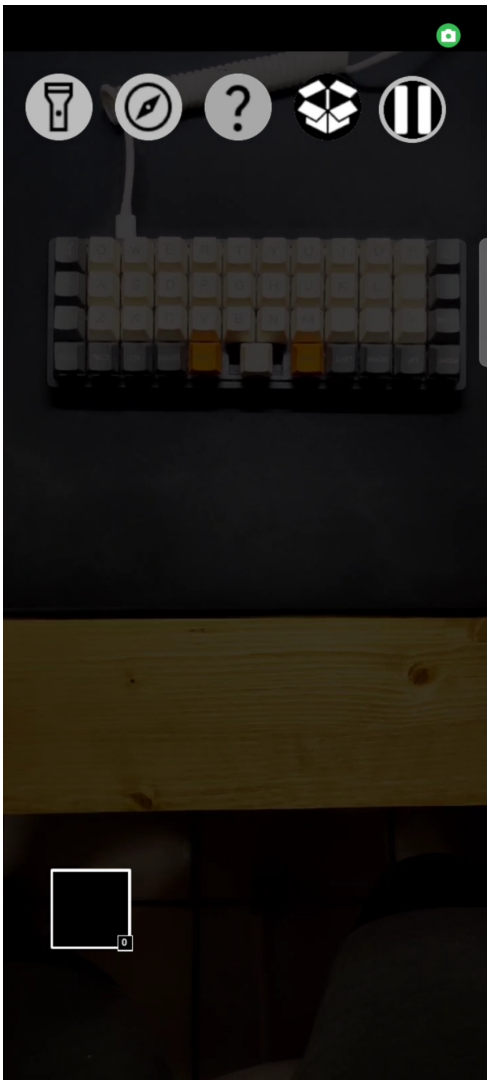

FIGURA 9.4: Inici de partida amb la llum apagada

FIGURA 9.5: Menú de Pausa

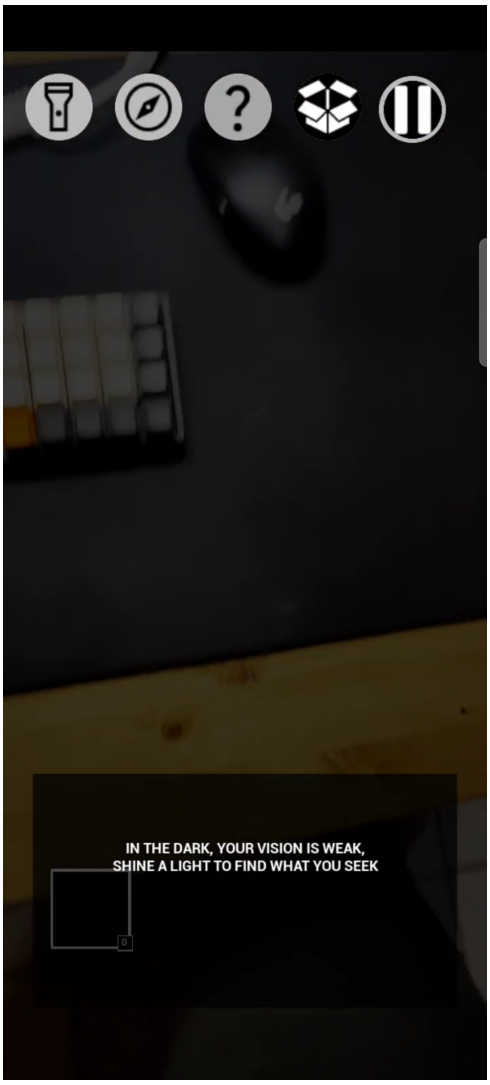

FIGURA 9.6: Sol·licitud d'ajuda

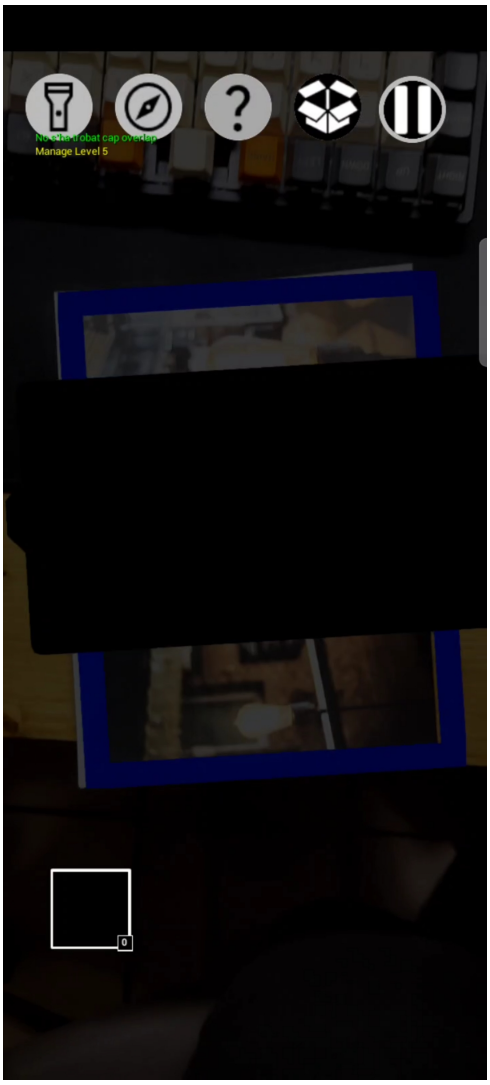

FIGURA 9.7: Escaneig targeta nivell 0

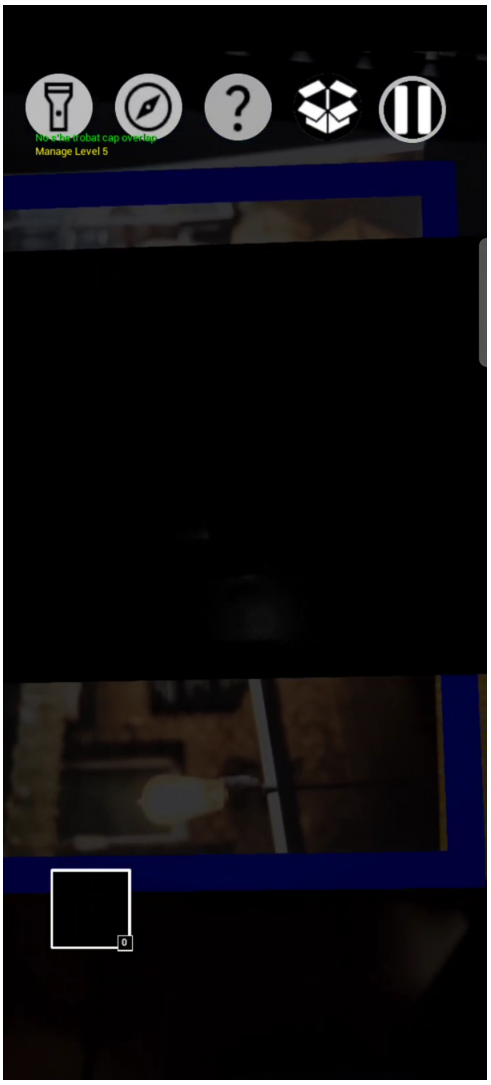

FIGURA 9.8: Nivell 0 amb el flaix apagat

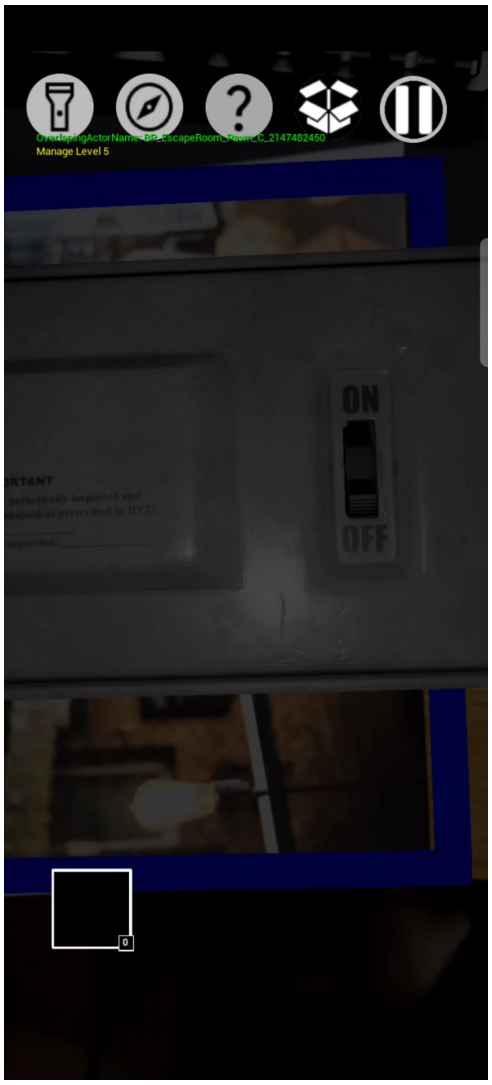

FIGURA 9.9: Nivell 0 amb el flaix encès

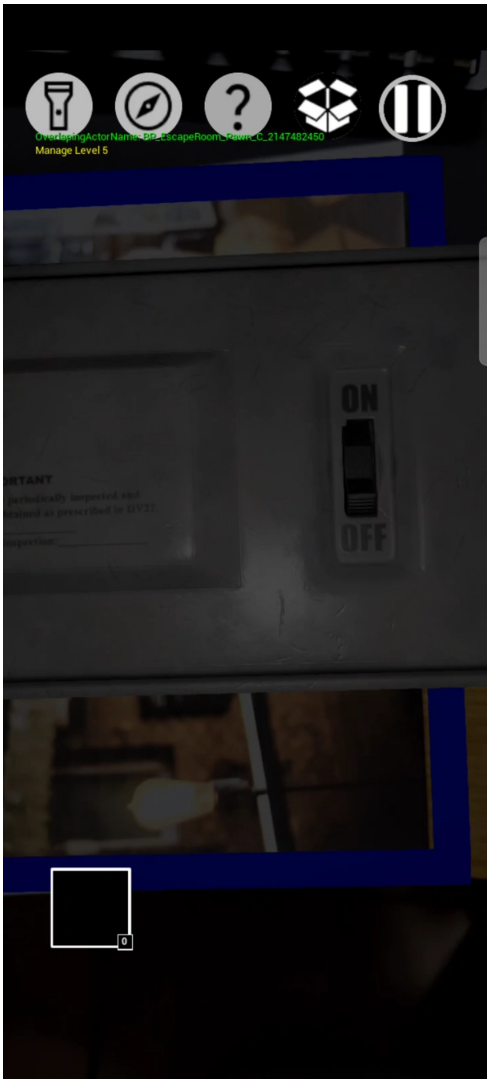

FIGURA 9.10: Nivell 0 amb el flaix encès

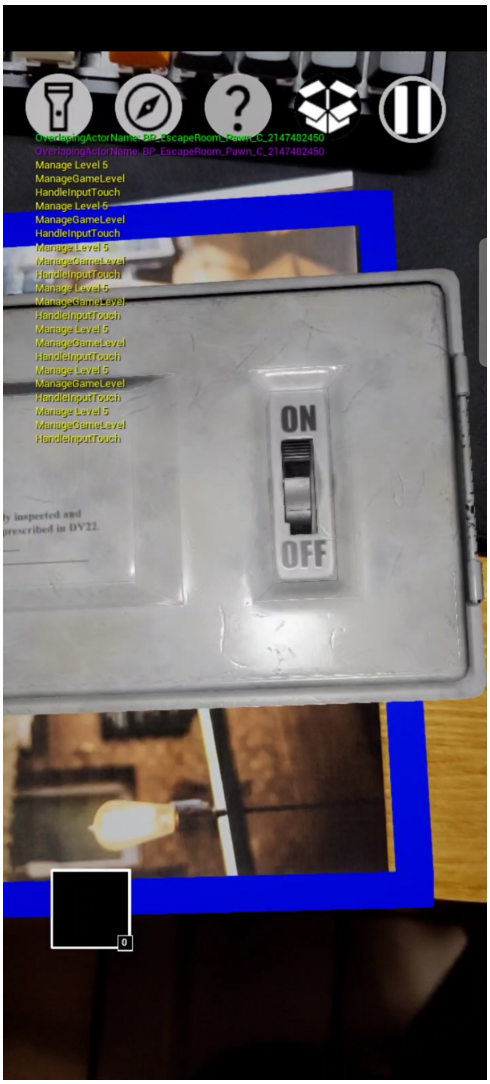

FIGURA 9.11: Nivell 0 amb la llum global encesa

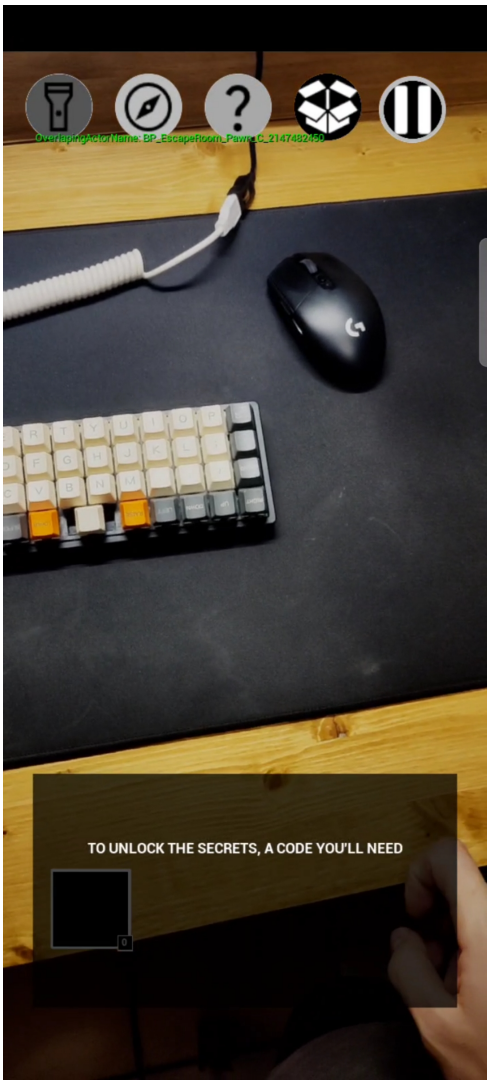

FIGURA 9.12: Visualització de pista en construcció

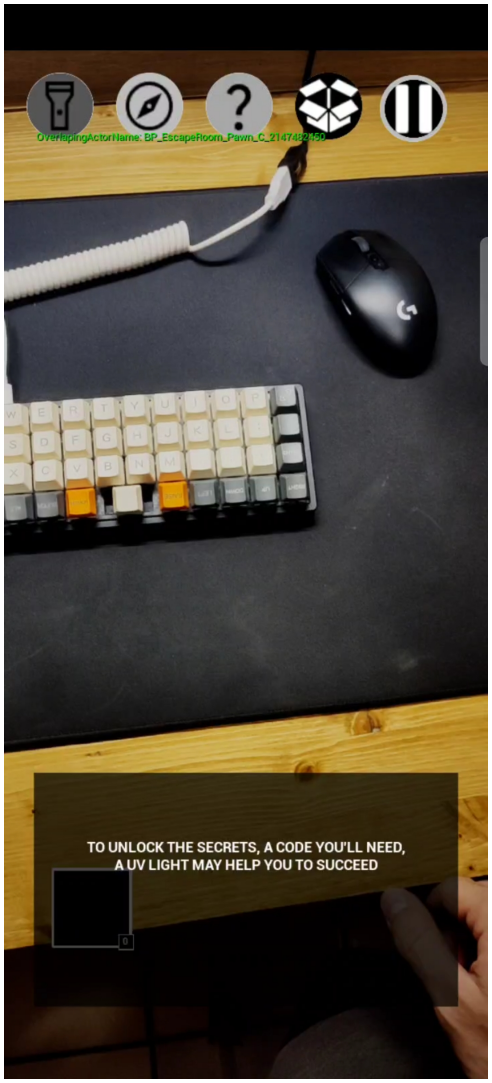

FIGURA 9.13: Visualització de pista completa

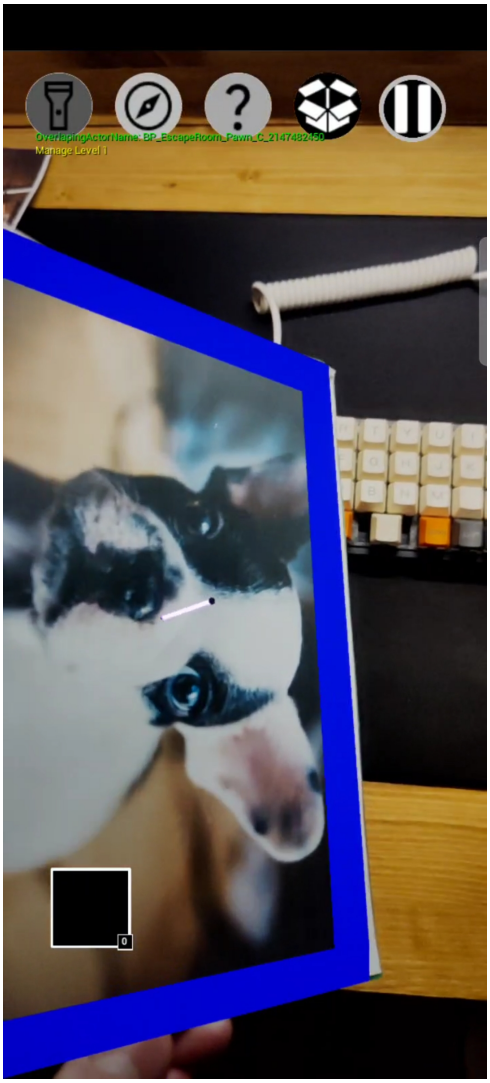

FIGURA 9.14: Visualització UVRod

FIGURA 9.15: Visualització UVRod il·luminant una figura

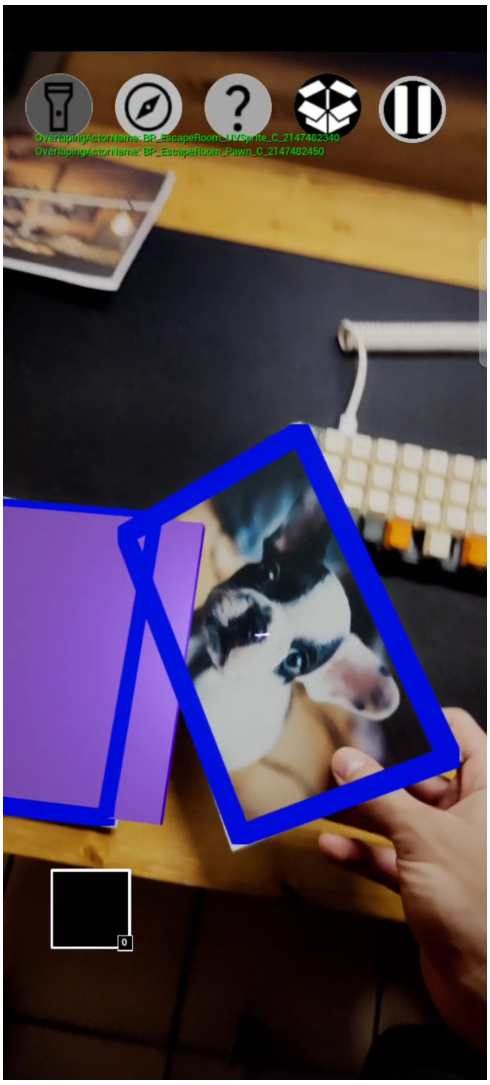

FIGURA 9.16: UVRod apropant-se al codi a revelar

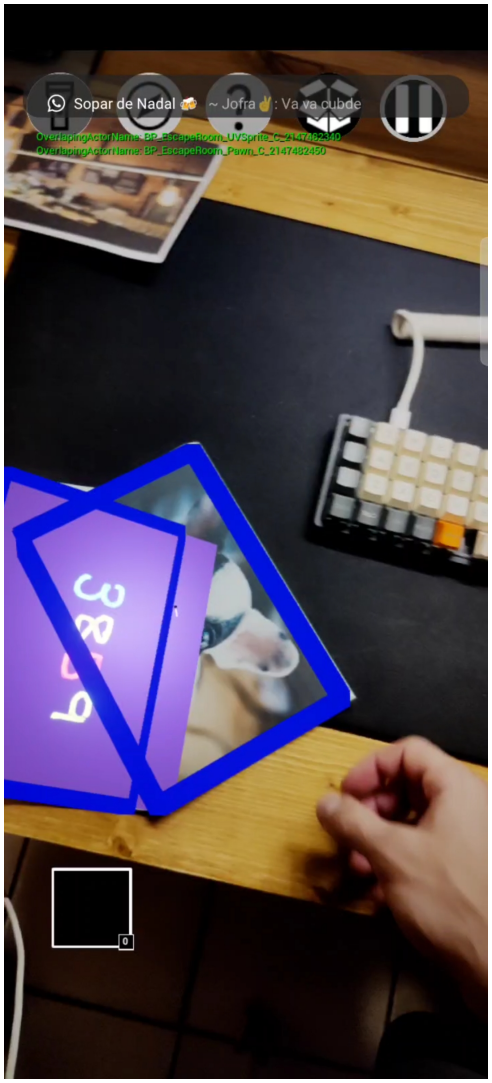

FIGURA 9.17: LLum ultravioleta revelant el codi

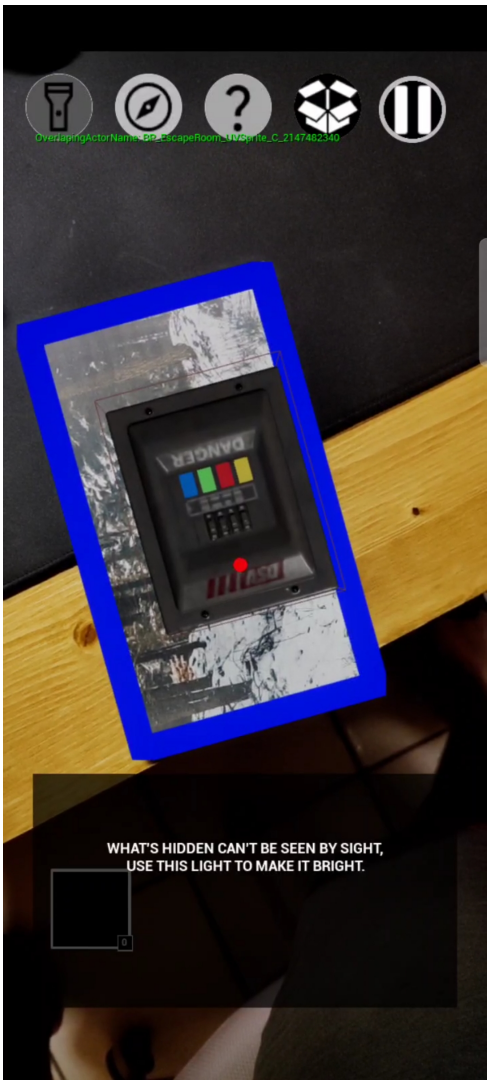

FIGURA 9.18: Pista del nivell 1

FIGURA 9.19: Widget d'entrada del codi

FIGURA 9.20: Entrada de codi amb el codi revelat a sota

FIGURA 9.21: Desplegat del teclat per entrar el codi

FIGURA 9.22: Submit de codi incorrecte

FIGURA 9.23: Submit de codi correcte

FIGURA 9.24: Pista del nivell 2

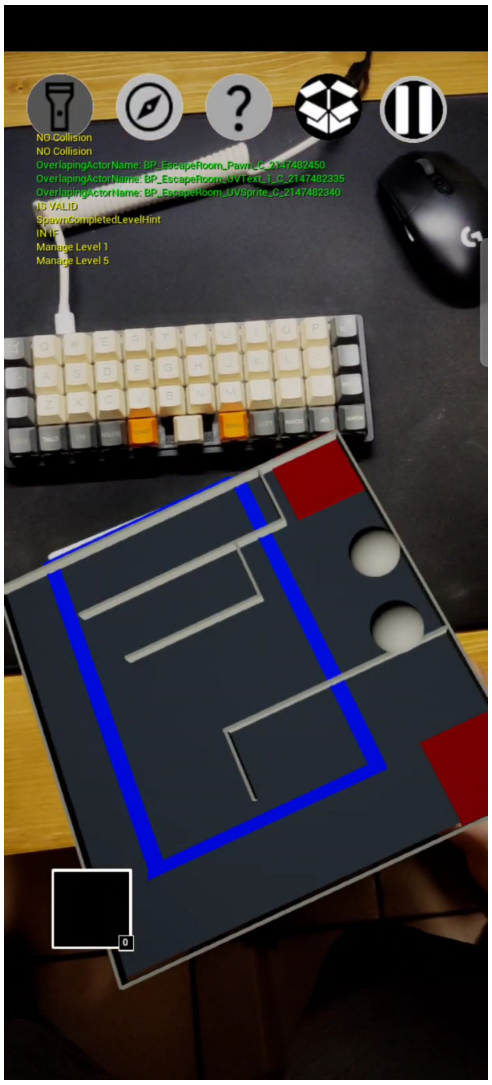

FIGURA 9.25: Visualització inicial del nivell 2

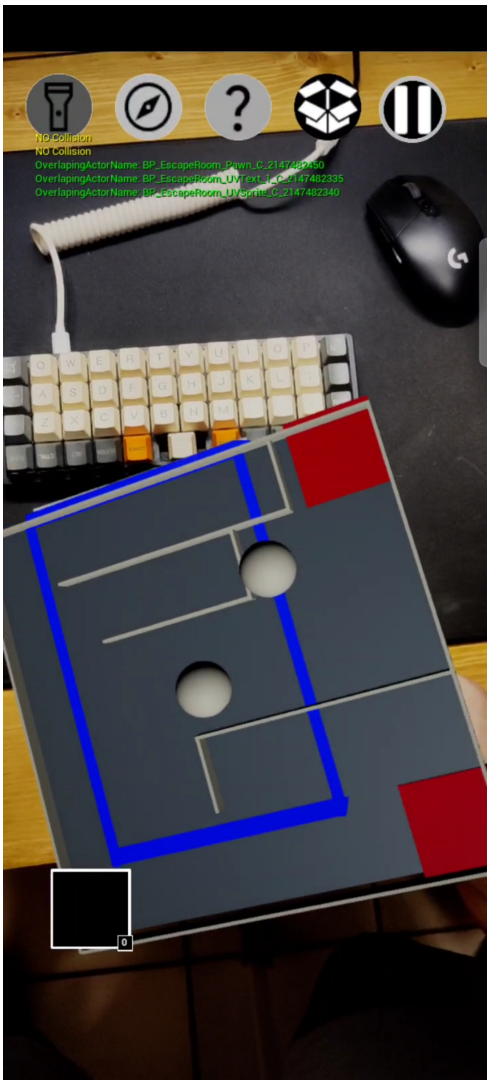

FIGURA 9.26: Moviment de les boles a la inclinació de la targeta

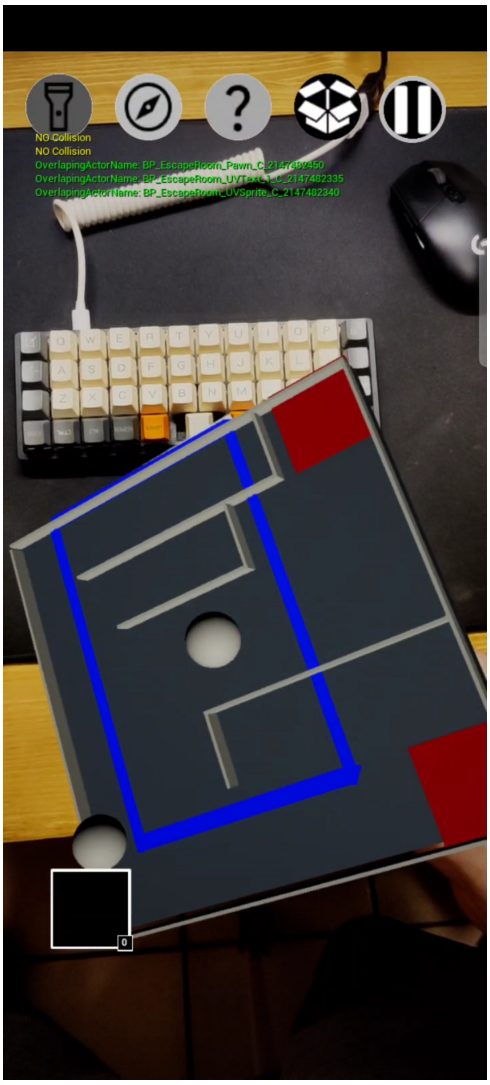

FIGURA 9.27: Moviment de les boles a la inclinació de la targeta

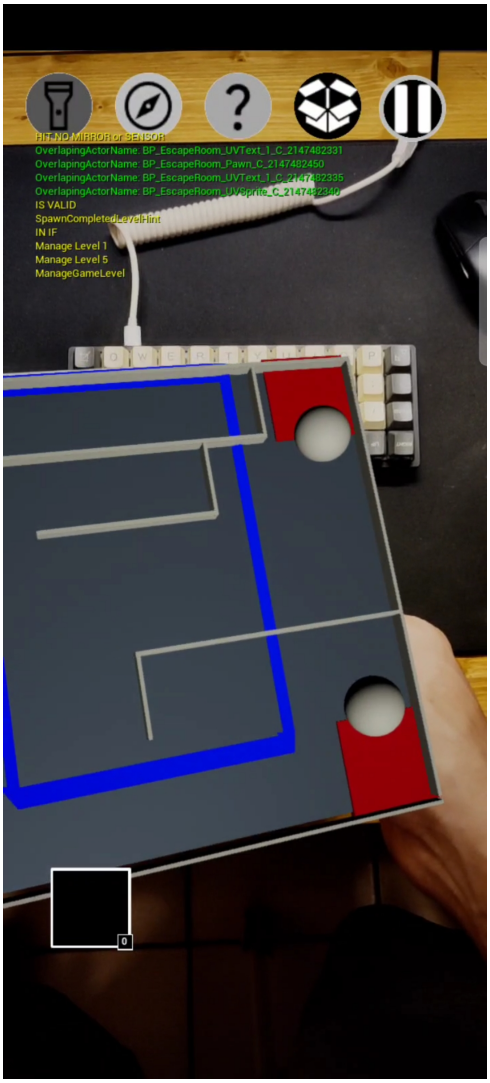

FIGURA 9.28: Nivell amb les boles sobre les plaques

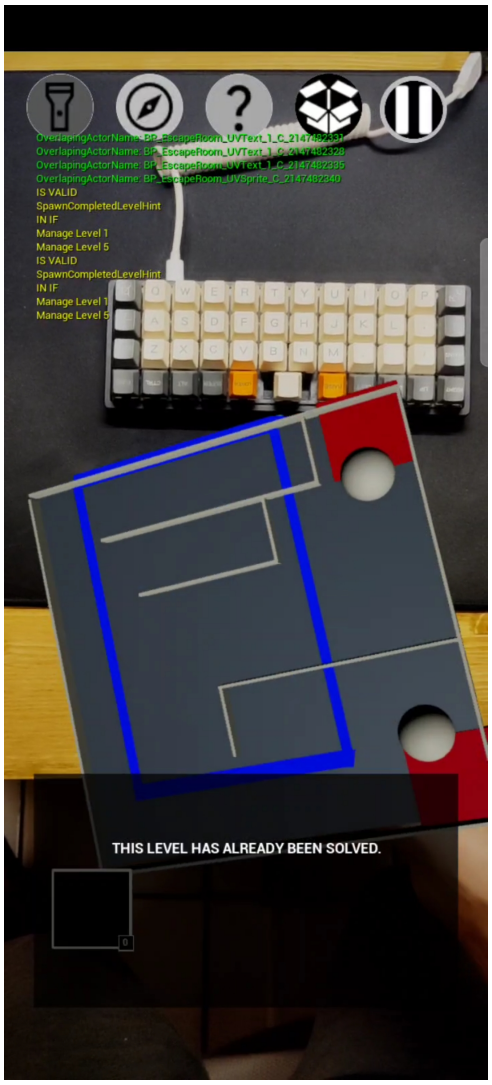

FIGURA 9.29: Sol·licitud de pista sobre un nivell resolt

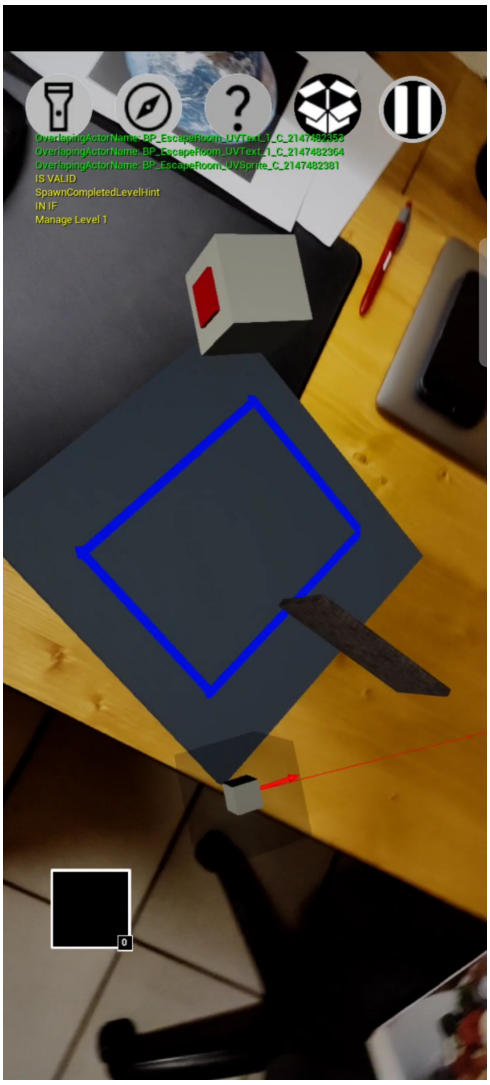

FIGURA 9.30: Nivell 3 amb el laser sense ser reflectit

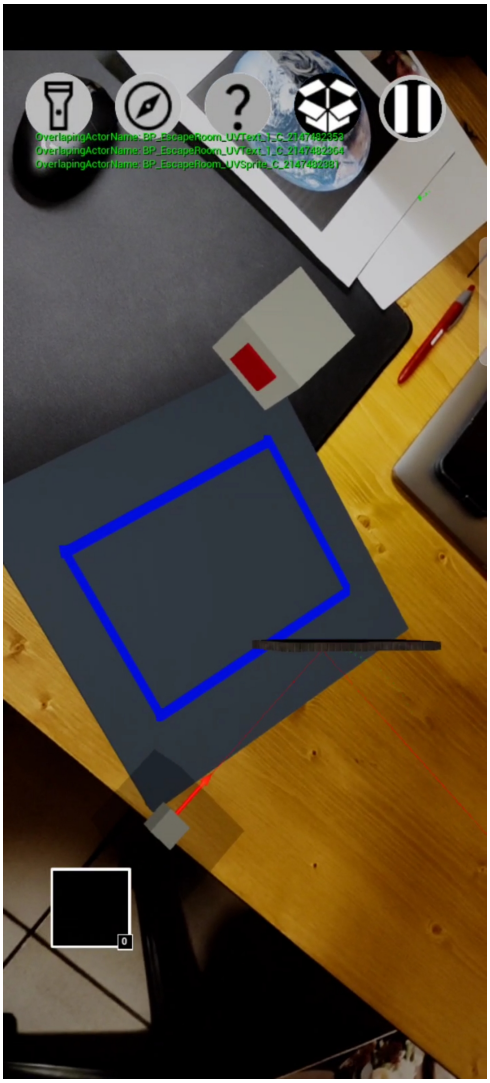

FIGURA 9.31: Nivell 3 amb el laser reflectit

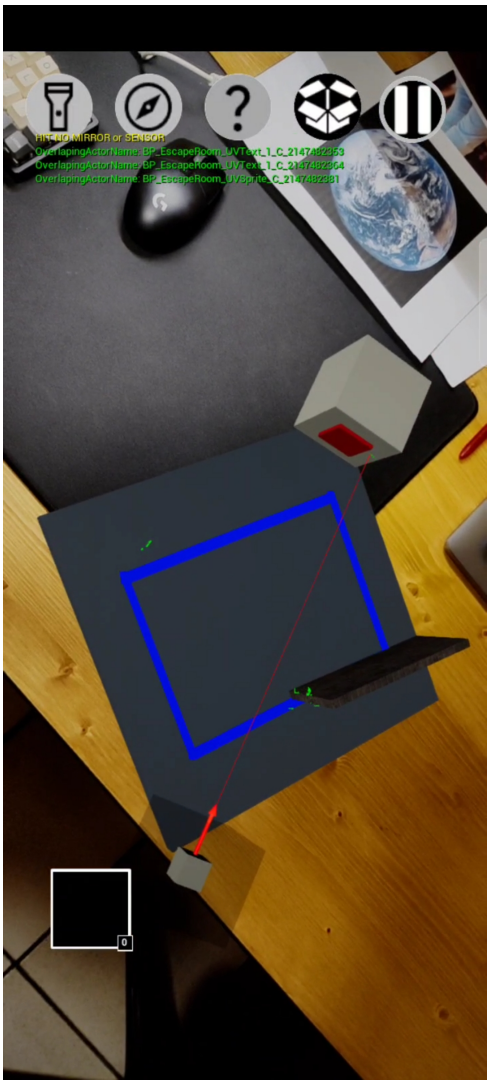

FIGURA 9.32: Laser collisionant a un material que no és de sensor

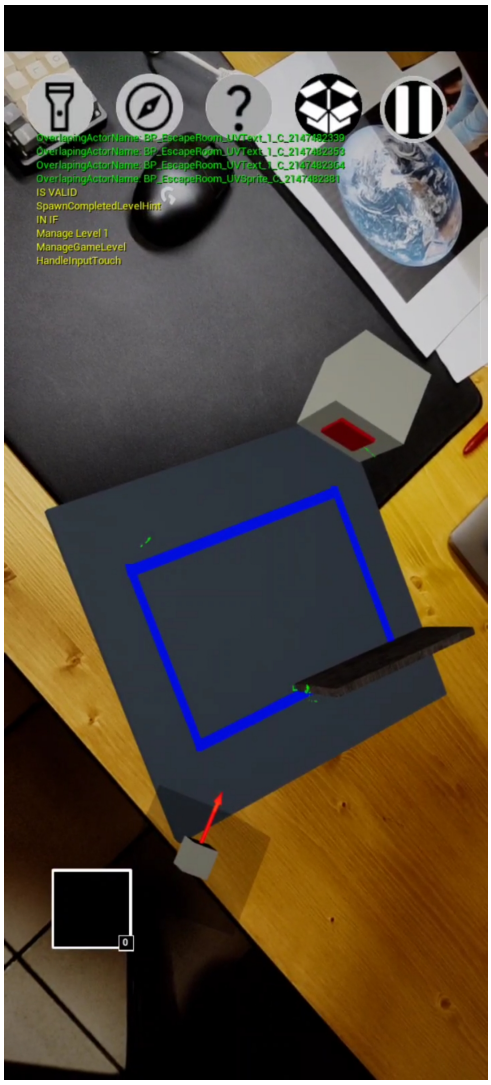

FIGURA 9.33: Laser un cop col·lisionat amb el sensor

FIGURA 9.34: Brúixola amb el dispositiu enfocant l'Est

FIGURA 9.35: Brúixola amb el dispositiu enfocant al Nord

FIGURA 9.36: Carta del nivell 5 enfocant en direcció diferent al Nord

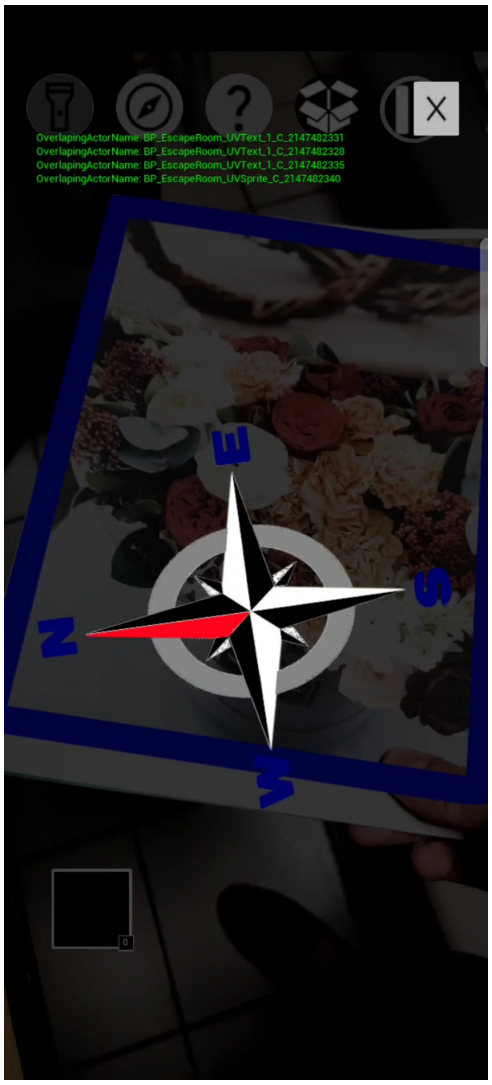

FIGURA 9.37: Carta del Nivel 5 juntament amb la brúixula

FIGURA 9.38: Carta del nivell 5 essent visible un cop enfocat el Nord

FIGURA 9.39: Carta del nivell 5 no visible al desenfocar el Nord

FIGURA 9.40: Nivell 5

FIGURA 9.41: Missatge al fer clic al pom de la porta

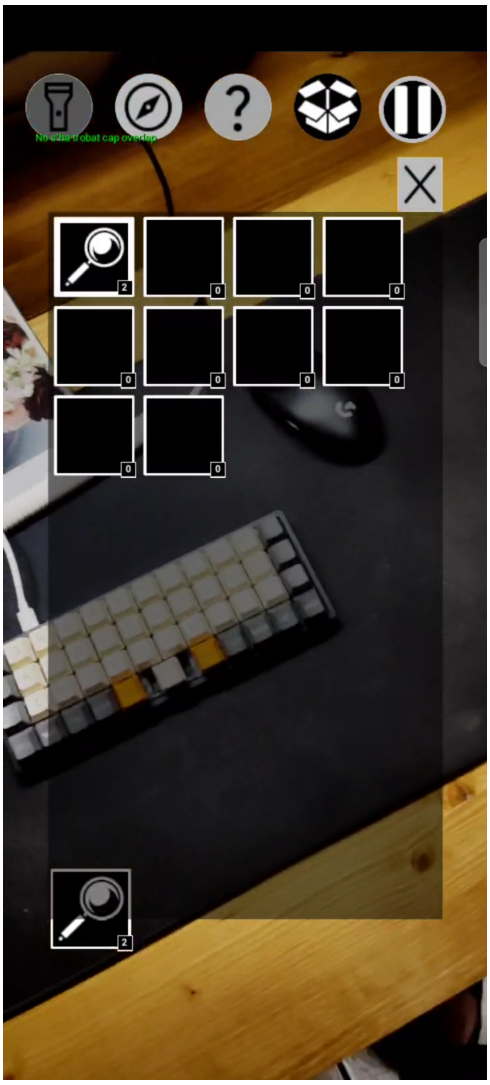

FIGURA 9.42: Selecció d'un element de l'inventari que no és una clau

FIGURA 9.43: Missatge al fer clic al pom de la porta amb l'element seleccionat

FIGURA 9.44: Targeta que genera un element d'inventari seleccionable

FIGURA 9.45: Inventari un cop agafada la clau

FIGURA 9.46: Nivell 5 amb la clau seleccionada apunt per clicar el pom

FIGURA 9.47: Menú de partida guanyada

Capítol 10

Conclusions

L'objectiu del projecte era crear un prototip funcional de videojoc d'estil *EscapeRoom* de realitat augmentada i considero s'ha aconseguit, juntament amb tots els propòsits que es van determinar al principi del projecte. Considero el prototip final un èxit, amb un bon enfoc del que es pot aconseguir amb pocs recursos i començant des de 0 en una aplicació de realitat augmentada.

Per tal de desenvolupar un videojoc d'estil poc comú, ha sigut necessari aprendre bé el funcionament d'Unreal Engine. També el disseny i l'estructura d'un videojoc i les diferents parts que el formen. He aconseguit sense dubte tenir el contacte que desitjava amb el desenvolupament de videojocs i que no havia vist a la carrera de GEINF, havent pogut aplicar molt del coneixement dels anys de carrera. He sortit de la zona de confort en treballar amb un contingut molt diferent al que havia vist fins el moment, però sento que he assolit un bon nivell de coneixement sobre el desenvolupament de videojocs i la realitat augmentada que s'ha vist reflectit a mesura que avançava el projecte, que m'ha permès implementar funcionalitats no previstes en els objectius inicials, com el sistema de pistes, l'inventari, o la brúixola.

Ara bé, no és més que això, un prototip inicial i amb moltíssima capacitat creixement, que combina petites característiques per demostrar el que es pot fer, però que deixa molt marge d'ampliació i millora en cadascuna. Això és un dels aspectes que més em motiva de cares al futur i que més m'incentiva en confiar en la realitat augmentada.

Tot i això, el transcurs del projecte ha estat ple d'entrebancs i no ha estat gens fàcil. Per començar, vaig haver de posar molt esforç i hores prèvies al començament del projecte per assolir el coneixement mínim d'Unreal Engine per poder començar a dissenyar el projecte. A més, la falta d'us de la realitat augmentada també repercuteix negativament a l'hora de desenvolupar: per un costat, m'he trobat que la documentació i exemples oficials son molt millorables, en concret la d'Unreal Engine, fins i tot la pròpia plantilla de realitat augmentada d'Unreal Engine es inservible per un *bug* identificat per ells però que no arreglen. Per altra costat, no hi ha comunitat, no hi ha un consens establert pels desenvolupadors de dissenys estàndard d'aquelles coses presents en tot projecte de realitat augmentada, i tot desenvolupador sap lo important que és una bona comunitat activa, crítica i compartint coneixement en el mon del *Software*.

De totes maneres, m'he agafat els obstacles com a reptes i oportunitats de contribuir a aquesta comunitat, i així els he superat.

La planificació, s'ha desviat. En un principi vaig pensar que tindria més temps del que realment he disposat al combinar el treball amb la feina i tots els períodes descrits en l'apartat de planificació s'han allargat, sobretot la part final d'implementació i la documentació, havent d'ajornar l'entrega la següent convocatòria. També em va sortir una oferta de feina que va tallar el desenvolupament del projecte al haver de fer cursos i obtenir certificats per entrar i ser competent a la nova feina.

Capítol 11

Treball Futur

En aquest apartat es comenten les possibles millores i ampliacions que es poden realitzar.

11.1 Disseny gràfic

Un aspecte a tenir en compte per millorar l'experiència del jugador és el disseny gràfic del prototip. Per tal de fer mes complet el videojoc en versions futures, m'agradaria realitzar un bon disseny de les figures.

11.2 Só

Pràcticament qualsevol videojoc té efectes de so i banda sonora. En el projecte no s'ha pogut afegir aquesta característica però és un element que cal incloure en el videojoc i, per la naturalesa dels enigmes, seria molt interessant que fos part de la solució d'algun d'aquests.

11.3 Mecàniques

11.3.1 Hand tracking

Com he comentat en algun dels apartats de la memòria, crec que el *Hand tracking* és de les qualitats més fascinants de la realitat augmentada. A mi és un dels treballs futurs que més em motiven, tot i que no és un requeriment funcional necessari per tal que el producte sigui complet.

11.3.2 Detecció d'objectes

La IA està prenent un paper cada cop més important al mon del desenvolupament del software. Com s'ha comentat des del principi, un dels propòsits més importants del projecte és que hi hagi la màxima interacció entre el mon real i el virtual durant la partida. Incorporar la detecció d'objectes per ser utilitzat en la resolució d'algun dels nivells entra dins dels objectius futurs.

11.3.3 SLAM

En el videojoc, sempre es depèn de les targetes, i ArCore té mecanismes per a geo-localitzar el dispositiu en l'espai real i situar-lo en el mon virtual. M'encantaria incorporar SLAM (Simultaneous Localization and Mapping) en el videojoc, ja que podria canviar radicalment l'experiència del jugador, podent aprofitar realment l'espai físic de la sala sense haver de dependre de l'escaneig de targetes en tot moment.

11.3.4 *Grab* Objectes

Una mecànica que aportaria molta versatilitat a l'hora de dissenyar nous nivells és la capacitat d'agafar objectes en clicar-los a través de la pantalla del dispositiu i arrastrar-los mentre es mantingui el contacte amb el dit, és per això que aquesta característica estaria inclosa en el plantejament del desenvolupament de futures versions.

11.3.5 Nivells amb la llum apagada

Una de les funcionalitats ja existents al videojoc és la d'apagar la llum global. Aquesta mecànica ens permetria fer nivells que requerissin tenir la llum apagada, per exemple, un que per ser resolt necessitis un codi fluorescent que brilla en la foscor i que només és visible amb la llum apagada.

11.4 Mode Multi-jugador

Aquesta funcionalitat és molt ambiciosa però tecnològicament viable. ArCore té una funcionalitat anomenada **Point Cloud** que permet tenir elements virtuals compartits al núvol i Unreal Engine ja té gestió multi-jugador. Penso que seria molt enriquidor poder fer una modalitat multi-jugador on es requerís la cooperació dels múltiples jugadors en una mateixa sala per resoldre els diferents nivells.

11.5 Pàgina Web del producte

Un dels requeriments per poder jugar al videojoc és tenir les targetes físiques. M'agradaria tenir una web on descarregar les imatges, juntament amb les instruccions del videojoc, una possible explicació de la temàtica de la partida, etc, per tal de poder fer arribar el joc a la gent.

Capítol 12

Annexos

Com a Annex, s'adjunta el projecte sencer d'Unreal Engine a la carpeta del projecte per tal que el corrector el pugui explorar amb llibertat. Podem trobar el codi c++ a la carpeta **/Source/EscapeRoomV2** i tots els assets a **/Content/EscapeRoom**.

Capítol 13

Manual d'usuari i/o instal·lació

Per tal d'utilitzar l'aplicació el primer que cal fer és comprovar si el dispositiu és compatible amb ArCore. Ho podeu fer consultant el llistat del següent enllaç: .

Seguidament, caldrà instal·lar else **Serveis Google Play per a RA** al Play Store:

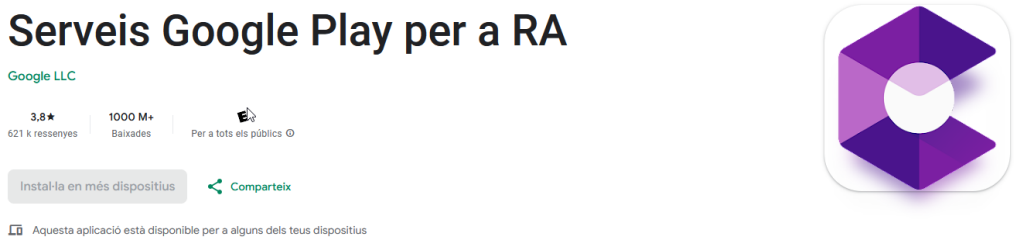

FIGURA 13.1: Servei Google Play per a RA

A la carpeta del projecte hi ha un fitxer TFG_MartiParnau_Descarregues.txt amb un enllaç a una carpeta compartida de drive, d'on es pot descarregar el fitxer .apk que conté l'aplicació instal·lable. Tots els dispositius són diferents, però deixo un exemple. Veure Figures 13.2 i 13.3.

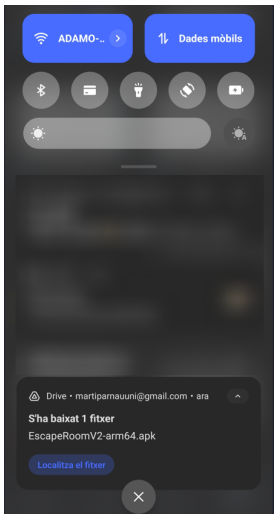

FIGURA 13.2

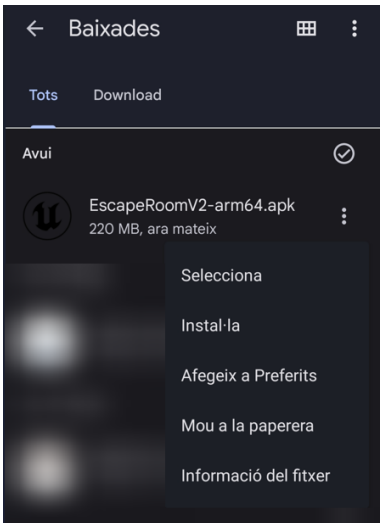

FIGURA 13.3

Bibliografia

[1] Stephen Ulibarri Ben Tristem GameDev.tv Team. *Unreal Engine 5 C++ Developer: Learn C++ and Make Video Games*. URL: https://www.udemy.com/course/unrealcourse/ (cons. 04-09-2023).

[2] Microsoft. *Visual Studio Code*. URL: https://code.visualstudio.com/ (cons. 04-09-2023).

[3] Microsoft. *Repository Github Visual Studio Code*. URL: https://github.com/microsoft/vscode (cons. 04-09-2023).

[4] Unity Tecnologies. *Unity Documentation*. URL: https://docs.unity.com/ (cons. 04-09-2023).

[5] Inc. Epic Games. *Unreal Engine 5.0 Documentation*. URL: https://docs.unrealengine.com/5.0/en-US/ (cons. 04-09-2023).

[6] Google. *Mediapipe Handtracking Solution*. URL: https://developers.google.com/mediapipe/solutions/vision/hand_landmarker (cons. 04-09-2023).

[7] Google. *ArCore Documentation*. URL: https://developers.google.com/ar/develop (cons. 04-09-2023).

[8] Epic Games. *Developing for ARCore*. URL: https://developers.google.com/ar/develop (cons. 04-09-2023).

[9] PTC Inc. *Vuforia Library*. URL: https://library.vuforia.com/ (cons. 04-09-2023).

[10] Gabeee. *UV Light Unreal Engine Plugin*. URL: https://www.unrealengine.com/marketplace/en-US/product/uv-light (cons. 04-09-2023).