PFC Visió per computador

Jugant amb la realitat

Alumne: Jorge Arnal Montoya

Consultor: David Masip

Titulació: Enginyeria Informàtica

Resum

Aquest projecte s'ha desenvolupat dins de l'àrea de Visió per computadors, mitjançant el reconeixement d'un patró podem definir tres eixos que conformen un espai tridimensional on hem implementat un videojoc de combats entre robots a sobre d'un entorn real.

Els videojocs és actualment la forma d'entreteniment que factura més diners a nivell mundial, per sobre d'altres indústries com la música o el cinema. En els últims anys hem vist productes com Eyetoy de Sony, Invizimals de Novarama Studios, You're in the movies de Microsoft que fan servir la tecnologia de la visió per computador per crear productes d'entreteniment on el món real del jugador s'integra dins de l'entorn virtual.

El desenvolupament d'aquest projecte es pot dividir en dues parts. La primera part és la detecció de la marca i la creació de l'espai 3D a partir de la marca; això s'ha aconseguit a través de la llibreria Vuforia. La segona part és la implementació d'un petit videojoc de combats on el player pot interactuar amb el personatge que s'integra sobre l'entorn real a través de la pantalla.

Per desenvolupar aquest projecte hem fet servir el motor de videojocs Unity3D i la llibreria Vuforia de Qualcomm.

Agraïments

Com autor d'aquest projecte voldria agrair les persones que m'han ajudat i donat suport durant el desenvolupament d'aquest projecte així com el temps emprat durant el període de formació dins de la UOC.

En primer ordre vull agrair a la meva companya pel suport inestimable en els dies que costava més posar-se davant de l'ordinador. Agrair també el meu soci pel seu enteniment en els dies que havia de robar-li hores a la feina per poder preparar material per la universitat.Com no agrair a la meva família perquè sense ells no som el que som.

Per últim agrair l'ajuda de Francesc Rodríguez que ha realitzat la GUI del videojoc així com el logotip en 3D del videojoc. Sense oblidar-me dels professors de la UOC que ens ajuden en els dubtes i en especial al tutor d'aquest projecte David Masip pel seu suport.

Índex

1. Introducció

1.1 Motivació

El projecte a desenvolupar dins de l'assignatura de PFC –Visió per computador consistirà en una aplicació interactiva on l'usuari tindrà la possibilitat mitjançant una marca i una càmera simular baralles entre personatges en un entorn 3D dins del seu dispositiu mòbil.

En els últims anys ha hagut un gran canvi en la societat on gairebé tothom disposa d'un telèfon intel·ligent que no utilitzem únicament per fer trucades, si no que disposem d'una plataforma on poder fer moltes altres tasques com consultar el correu electrònic, navegar per la web, utilitzar l'aplicació de mapes, … Ens trobem doncs aplicacions tant de tipus productiu com d'entreteniment, ja sigui per veure vídeos, comunicar-nos als nostres amics mitjançant xarxes socials o utilitzar-ho per a jugar a videojocs de més o menys simplicitat tecnològica.

Com a conseqüència de l'augment de la potència dels mòbils i que disposen de càmeres suficientment avançades, aquest projecte consistirà en desenvolupar un videojoc per a plataforma mòbil on s'utilitzarà la visió per computador per fer un estudi de l'entorn del jugador i simular un entorn 3D augmentant el món real.

La part de jugabilitat del videojoc consistirà en un entorn 3D relativament senzill on varis personatges simularan un baralla virtual.

Pel desenvolupament del projecte utilitzarem llibreries de visió per computador que facilitin la implementació de les funcionalitats específiques.

Per altra part, utilitzarem el motor 3D de videojocs Unity, un motor multi-plataforma i actualment el més utilitzat per empreses desenvolupadores de videojocs independents degut al seu baix cost i les seves altes prestacions.

1.2 Objectius

Els objectius principals del projecte són els següents:

- Utilitzant marcadors trobarem el centre de coordenades del nostre entorn virtual, un exemple de marcadors seria el següent.

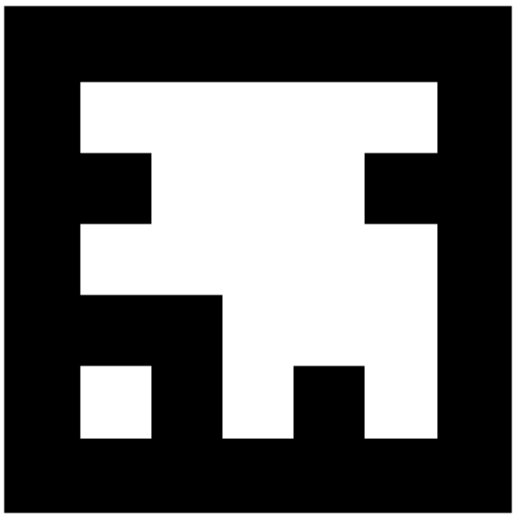

Fig. 1 Marcador

- Aquests marcadors els utilitzarem d'una forma física impresos en paper i els utilitzarem en el món real. Mitjançant visió per computador serem capaços de detectar el marcador i crear uns eixos de coordenades virtual on es renderitzarà el nostre entorn de videojoc.

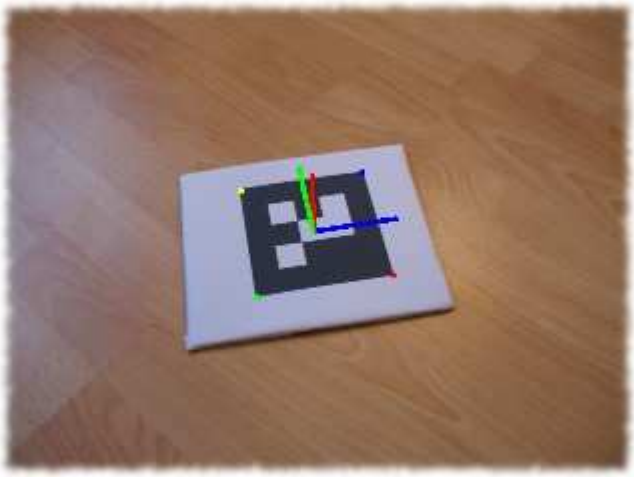

Fig. 2 Marcador en entorn real

- Una vegada tenim aquest centre de coordenades utilitzarem realitat augmentada per simular les nostres batalles virtuals dins del propi entorn real augmentat.

Fig. 3 Imatge del joc Reality Fighters

El resultat del projecte serà un petit videojoc de baralles que s'executarà en plataforma mòbil. L'usuari podrà interactuar mitjançant el seu dispositiu per controlar la càmera, a través de la pantalla tàctil executarà comandaments sobre el seu avatar dins del videojoc, el personatge rival executarà una intel·ligència artificial senzilla.

Com es pot apreciar, el projecte es basa en productes que ja existeixen com a productes comercials, és per això que creiem que és totalment viable segons els recursos dels que disposarem per la realització del projecte, evidentment el resultat final distarà molt d'un producte comercial on treballen desenes de persones tant de la branca informàtica com de la branca artística.

1.3 Planificació

El projecte el dividirem en les següents tasques i amb una estimació aproximada de cadascuna d'elles.

- Tasca 1: estudi de les diferents llibreries que afavoreixen el desenvolupament de realitat augmentada sota el Motor de videojocs Unity3D.
- Tasca 2: detecció de marca mitjançant llibreria de realitat augmentada.
- Tasca 3: generació d'entorn virtual sobre l'entorn real.
- Tasca 4: generació de mini joc de baralles en l'entorn virtual.
- Tasca 5: memòria.

1.3.1 Cronograma

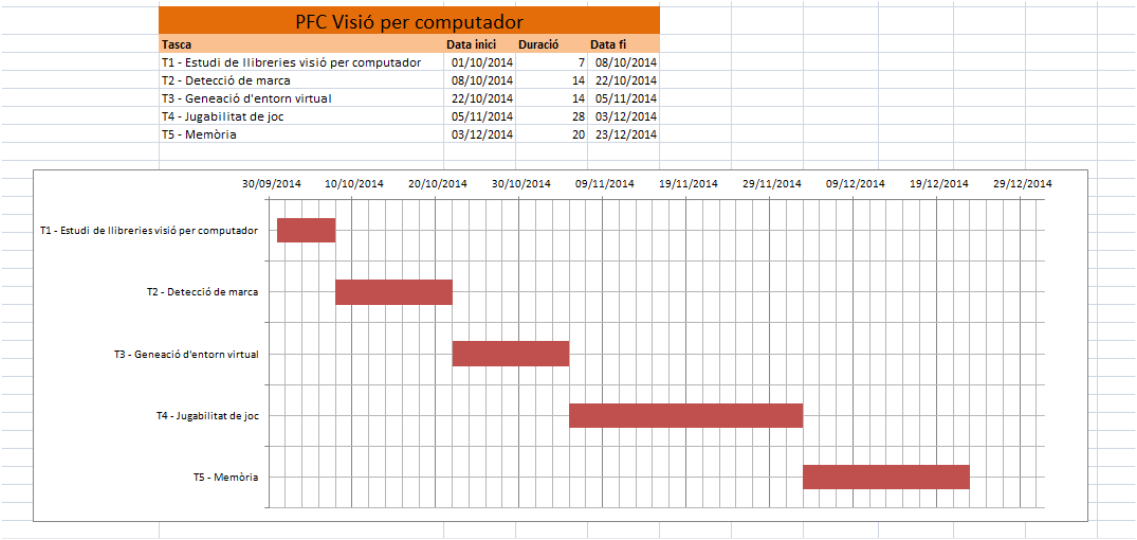

Fig. 4 Cronograma del desenvolupament del projecte

1.4 Productes obtinguts

Durant la creació del projecte s'han obtingut els següents productes:

- Un videojoc jugable per plataforma Android
- Memòria del projecte
- Annex instal·lació de llibreria Vuforia de Qualcomm.
- Annex codi font

1.5 Entorn de desenvolupament

Les eines utilitzades pel desenvolupament del projecte han sigut:

- Un PC amb processador Intel i5 @ 2.4Ghz i 4GB de memòria RAM
- Sistema operatiu Windows 7 64 bits
- Motor Unity3D v.4.5.5f1
- Microsoft Visual Studio 2010
- Vuforia SDK 3.0.9
- Tauleta Android Samsung Tab 4.0 10.1

1.6 Descripció dels capítols

Aquesta memòria està dividida en 5 capítols:

- Capítol 1, trobem una introducció al projecte i al desenvolupament del mateix.
- Capítol 2, trobarem l'estat de l'art en el camp de la Visió per computador aplicada als videojocs.
- Capítol 3, explicarem el desenvolupament del videojoc.
- Capítol 4, veurem els resultats aconseguits del projecte.
- Capítol 5, donarem conclusions i millores que es poden fer en el producte.

2. Estat de l'art

Degut a l'augment de població usuària de videojocs, actualment la indústria del videojoc té una facturació superior a la resta d'indústries d'entreteniment com el Cinema, Música i DVD. Podem trobar estudis al respecte a la web de AEVI (Asociación Española de Videojuegos) i més concretament al "Balance económico de la industria del videojuego 2009" (Enllaç 1).

Actualment a Espanya tenim l'empresa de desenvolupament de videojocs Novarama Studios situada a Barcelona com un dels referents en videojocs amb Realitat Augmentada amb videojocs com Reality Fighters o la franquícia Invizimals que ha transcendit dels videojocs a un producte totalment *transmedia* com sèrie de tv, cromos i altres tipus de *merchandising*.

Fig. 5 Imatge del joc Invizimals

També empreses com Nintendo han apostat per la realitat augmentada en la seva plataforma Nintendo 3DS, consola que porta incorporada una càmera i inclou targes de tipus marcadors amb petits mini jocs.

Fig. 6 Visió per computador amb Nintendo 3DS

La pròpia Sony amb la seva consola de mà PSVita també incorpora marcadors i novament mini jocs amb els que jugar utilitzant aquest tipus de tecnologia.

Fig. 7 Visió per computador amb PS Vita

Com podem apreciar aquesta tecnologia està totalment provada i té un ús més que significatiu a nivell comercial, és per això que considerem aquest projecte molt interessant tant a nivell formatiu com de futur.

Tal com s'ha comentat prèviament, utilitzarem el motor de videojocs Unity 3D, aquest motor ens permetrà desenvolupar la part del videojoc de forma ràpida sense haver de preocupar-nos en crear la tecnologia per poder fer el pintat dels elements del propi videojoc.

Per poder realitzar la tasca de visió per computador utilitzarem plugins que funcionin sota Unity i que siguin compatibles amb les plataformes mòbils més importants actualment (iOS, Android).

2.1 Llibreries de visió per computador sota motor Unity

Pel desenvolupament del projecte hem estudiat diferents llibreries que ens permetran utilitzar la visió per computador per introduir realitat augmentada dins del nostre projecte sota el motor de videojocs Unity 3D.

Aquestes llibreries són:

- ARToolKit (Enllaç 2)
    - Pros
        - Integració amb Unity senzilla.
        - Suporta Windows, Mac, iOS i Android.
    - Cons
        - Necessita Unity Pro.
        - El desplegament a Android té diferents passos que no es fan directament des de l'editor de Unity.

- ARPA (Enllaç 3)
    - Pros
        - Integració amb Unity.
        - Suporta Windows, iOS i Android.

- Cons
  - No suporta Mac segons la web.
  - Té dues versions diferents una de pagament i una altra gratuïta.

- UART (Enllaç 4)
  - Pros
    - Integració amb Unity.
  - Cons
    - Web amb poca informació.
    - No mostra clarament les plataformes suportades.

- Vuforia (Enllaç 5)
  - Pros
    - Integració amb Unity.
    - Suporta iOS i Android.
    - Desplegament a Android directa i senzilla.
    - Web amb gran varietat de recursos i documentació.
    - No té versió de pagament i funciona tant amb Unity Pro com la versió gratuïta.
  - Cons
    - No suporta Windows ni Mac sota Unity 3D

- Multi-Platform Detection Plugin (Enllaç 6)
  - Pros
    - Suporta PC, Android i iOS.
  - Cons
    - Segons la web no suporta versió de Mac, tot i que sembla que el plugin de Unity si que ho suporta.
    - Tot i tenir una versió gratuïta per Unity, té dues versions de pagament superiors.
    - Documentació poc detallada a la web.

Després d'estudiar les diferents alternatives i valorar els pros i contres, ens trobem que la llibreria Vuforia és una llibreria prou robusta, compleix els nostres requisits de poder fer desplegats del projecte a plataformes mòbils, amés ens trobem que la seva web és molt completa i té una gran varietat de recursos amb els que aprendre a fer funcionar la llibreria.

Per últim confirmar que la integració amb Unity de la llibreria ha sigut molt neta i senzilla.

3. Solucions algorísmiques

En aquest apartat explicarem tot el referent al codi utilitzat en el desenvolupament del projecte.

3.1 Jerarquia utilitzada a Unity 3D

Abans de començar a parlar del codi que hem implementat, parlarem de l'estructura de dades que ens trobarem en la jerarquia de Unity en el moment d'obrir el projecte.

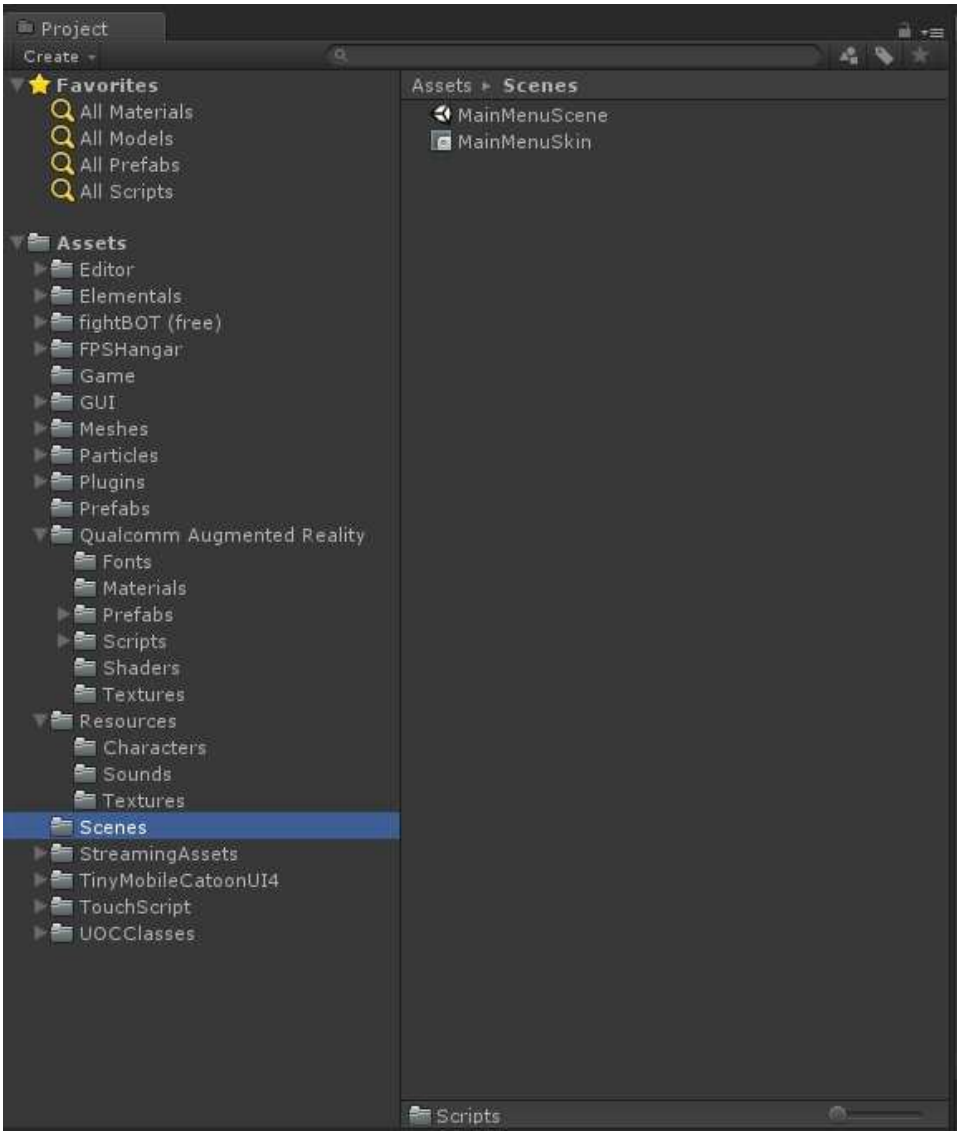

Fig. 8 Pestanya Project en Motor Unity

Dins d'aquesta arquitectura hem de destacar les següents carpetes on trobarem diferents elements que ens interessarà pel desenvolupament del projecte.

- Editor, en aquesta carpeta trobarem tot els scripts per a poder integrar el detector de marca amb la llibreria Vuforia de Qualcomm a nivell de l'editor.
- Elementals, en aquesta carpeta podem trobar diferents sistemes de partícules o efectes.
- fightBot (free), aquí trobem el model del nostre personatge així com les seves animacions.
- FPSHangar, novament un model 3D, en aquest cas ens trobem amb l'escenari utilitzat de la part del joc.
- Game, en aquesta carpeta trobarem tot el codi utilitzat pel desenvolupament del videojoc, tant la intel·ligència artificial del nostre enemic, com el controlador del player, o la GUI del videojoc.
- GUI, en aquesta carpeta ens trobem amb les textures que utilitzem per presentar la GUI del videojoc.
- Prefabs, en aquesta carpeta trobem els prefabs que ens permetran generar els personatges del joc.
- Qualcomm Augmented Reality, en aquesta carpeta ens trobem amb els elements necessaris per la utilització de la llibreria Vuforia de Qualcomm.
- Resources, en aquesta carpeta trobem diferents recursos que utilitzem en el joc com la música i algunes textures.
- Scenes, aquesta carpeta conté les dues escenes del videojoc que fem servir com són la del Menú principal (MainMenuScene) o la del joc (GameScene).
- UOCClasses, aquí trobem scripts que ens han ajudat al desenvolupament del videojoc con són GUIController, UOCTournament, ... Aquests scripts els explicarem amb més detall a continuació.

3.1 Diagrama del videojoc

El funcionament del videojoc el podem explicar amb el següent diagrama de flux.

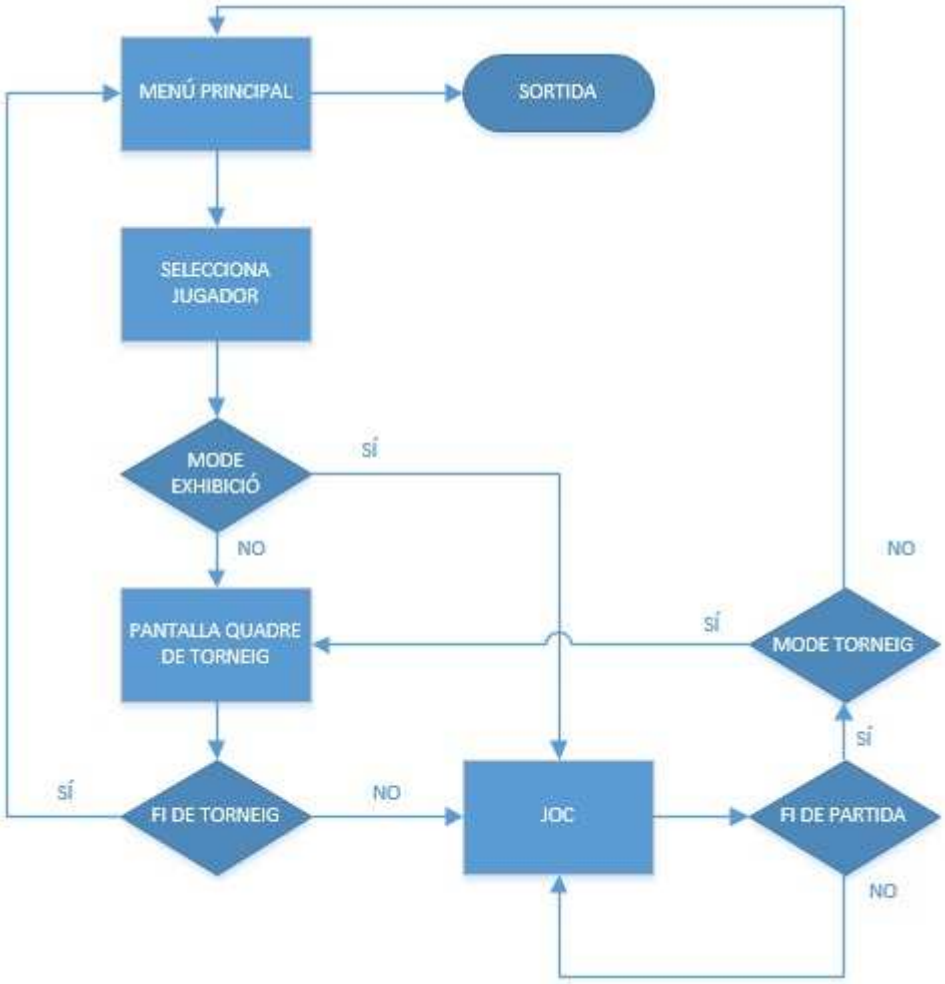

Fig. 9 Diagrama de flux del nostre videojoc

El joc comença en el menú principal depenent de si el player prem sobre el botó d'exhibició o torneig passarà a la pantalla de selecció de jugador, tenim un tercer botó que és el botó de sortida que tancarà l'aplicació.

Una vegada a la pantalla de selecció de jugador el jugador ha d'escollir un dels personatges, una vegada escollit si hem entrat en mode exhibició se'ns assignarà directament un rival i passarem a la pantalla de joc, en cas contrari anirem a la pantalla de quadre de torneig.

En la pantalla de quadre de torneig mentre no hi hagi un guanyador passarem a la pantalla de joc.

En la pantalla de joc es mantindrà fins que hi hagi un guanyador de la partida, en aquest moment en cas d'estar en una partida de mode Exhibició passem directament a la pantalla de menú principal en cas contrari tornarem a la pantalla de quadre de torneig.

3.1 Escena MainMenuScene

Hem implementat un menú principal on el jugador pot escollir entre els dos modes de joc que es trobarà en el joc. Aquest modes són exhibició i el mode torneig.

A continuació podem veure unes imatge que representa com és el menú principal del joc.

Fig. 10 Menú principal del nostre videojoc

En aquesta pantalla trobem que tenim tres botons un de sortida del joc, un botó per entrar en el mode exhibició del joc i un tercer botó pel mode torneig.

Si premem sobre el botó de exhibició o sobre el botó de torneig passem a la pantalla de selecció de personatge que podem veure a continuació.

Fig. 11 Imatge del selecció de jugador

En aquesta pantalla escollim el nostre personatge, en cas d'haver seleccionat en el menú principal la opció d'exhibició se'ns assignarà un personatge rival de forma aleatòria i anirem a la pantalla de joc. En cas contrari passarem a la pantalla de torneig que podem veure a continuació.

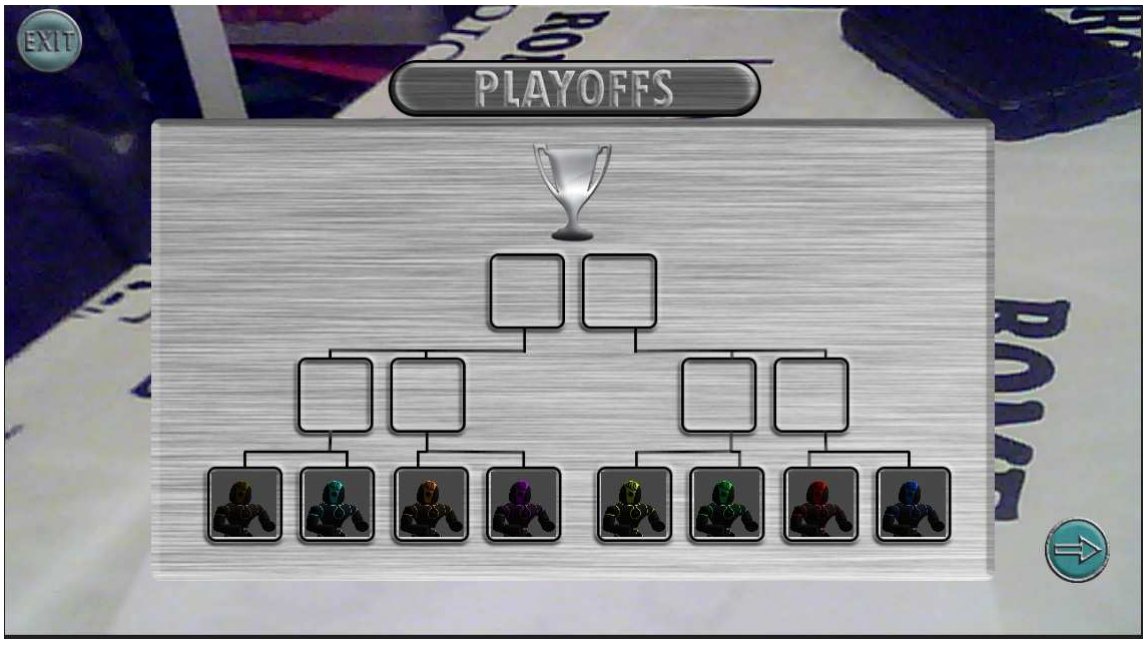

Fig. 12 Imatge de torneig

3.1.1 Classe MainMenuScreen

Aquesta classe és la encarregada de presentar el menú principal del videojoc per pantalla.

La seva lògica està implementada mitjançant un màquina d'estats on trobem els següents estats definits en un enumerat:

- MAIN_MENU, aquest estat mostra la pantalla de botons d'exhibició i torneig. Segons premem el botó d'exhibició o torneig passarem a les pantalles associades al botó.
- SELECT_PLAYER, aquest estat mostra la pantalla de selecció de personatge. En cas d'haver premut el botó d'exhibició a l'escollir un personatge se'ns assignarà un rival de forma aleatòria i passarà a la pantalla de joc; en cas contrari passarem directament a la pantalla d'eliminatòries.
- SHOW_PLAYOFF, aquest estat mostra la pantalla del quadre d'eliminatòries del torneig, haurem de polsar continuar fins arribar al nostre aparellament que és quan passarem a l'escena de joc.

3.2 Escena GameScene

L'escena GameScene és l'escena principal del joc, en aquesta escena s'executarà tota l'acció. A la figura següent podem veure una imatge del joc ingame.

Fig. 13 Imatge de la baralla en el joc

L'escena GameScene està composada per la jerarquia que podem veure en la següent imatge:

Fig. 14 Imatge de la jerarquia de l'escena de joc

D'aquesta jerarquia destaquem els objectes:

- ARCamera y FrameMarker0, són els objectes necessaris per integrar la visió per computador que ens permetrà detectar la nostra marca per poder generar l'espai tridimensional.

- World, conté l'escenari 3D que es mostra augmentant l'entorn real del jugador.
- Directional light, ens il·lumina l'escena per tenir una millor integració dels elements 3D.
- GameScreen, conté la classe que controla l'escena de joc.
- VirtualJoystick, Punch, Kick, Special, conté els botons de control del personatge.

Per portar a terme tot aquest codi hem utilitzat les diferents classes que veurem a continuació.

3.3 Classe GameScreen

La classe GameScreen és la responsable de presentar per pantalla la GUI del videojoc així com de preparar l'escena per la baralla entre els personatges del joc.

Quan s'inicia l'objecte que conté el component GameScreen, inicia els personatges de joc segons el que el jugador hagi escollit en el menú principal. Per això crida al mètode InitPlayers on es creen de forma dinàmica un personatge que controla el jugador i un segon personatge que està controlat per la intel·ligència artificial. Aquest personatge controlat per la IA té diferents paràmetres definits segons el personatge escollit així com un material que mostrarà una estètica diferent per cada personatge. Una vegada creats els personatges del joc passem a l'estat de començament de combat.

Destaquem que aquesta classe novament funciona amb una màquina d'estats on es controla l'estat de joc segons els diferents estats. Els estats que ens trobem són:

- IN_GAME, és l'estat més típic dins del videojoc i mostra la barra de vida dels personatges durant la partida.
- SHOW_ROUND, aquest estat mostra el començament del round i dona temps al jugador a preparar-se pel combat.
- END_ROUND, mostra la informació de fi de partida i guanyador.

- SHOW_PLAYOFFS, mostra el quadre d'eliminatòries del joc des de l'escena de joc.

3.4 Classe CFighterPlayer

La classe CFighterPlayer conté la informació dels diferents personatges del nostre joc, en aquest cas ens trobem amb 8 personatges diferents. Aquesta classe conté els següents atributs que detallem a continuació perquè s'utilitzen:

- ProfileTexture, conté la textura 2D que s'utilitza per representar el personatge en la selecció de personatges.
- CharacterName, conté el nom del personatge.
- Human, és una variable booleana que defineix si aquest personatge està controlat o no per un humà.
- Id, conté un identificador del personatge.
- MinDecissionTime, conté el valor mínim de temps que el personatge trigarà en prendre una decisió.
- RandomDecissionTime, conté un temps aleatori que sumat al MinDecissionTime ens donarà el temps que el personatge trigarà en prendre una decisió.
- AggressivePct, conté un valor entre 0 i 1 que ens diu el grau d'agressivitat del personatge quan és controlat per la intel·ligència artificial. Quan més proper a 1 és el valor més agressiu serà el personatge i ens atacarà més directament en cas contrari el personatge intentarà fugir després de cada cop. Aquest valor també s'ha utilitzat per al percentatge de vida que treu un personatge, com més alt és aquest valor més vida treu en el moment de donar un cop amb èxit.
- DeffensivePct, aquesta variable contindrà un valor entre 0 i 1 que ens donarà la probabilitat d'èxit de poder defensar-se un personatge controlat per la intel·ligència artificial davant un cop.
- KickPct i PunchPct, aquestes dues variables han de tenir un valor que sumats doni 1.0, ens dirà la probabilitat de donar un cop de peu o un cop de puny.
- SimpleAttackPct, aquesta variable tindrà un valor entre 0 i 1, quan el personatge controlat per la IA decideix que vol colpejar al jugador genera un valor aleatori entre 0 i 1, si aquest valor és per sota del valor

d'aquesta variable el jugador controlat per la IA executarà un cop simple, en cas contrari executarà un combo (conjunt de cops continuats).

Per últim destacar d'aquesta classe que el fet d'haver implementat les intel·ligències artificials segons un valors de variables ens permet generar diferents comportaments dels enemics.

3.5 Classe CFighterGame

La classe CFighterGame és una classe de tipus singleton que podem accedir des de qualsevol part del nostre codi. Aquesta classe serà l'encarregada de crear els diferents tipus de partides que tindrem en el nostre joc.

A continuació explicarem alguns dels mètodes en que consisteix la classe:

- InitPlayers, inicialitza el llistat de lluitadors que tenim en el videojoc, la inicialització dels personatges defineix les característiques de com lluiten en el joc.
- InitTournament, inicialitza una partida de joc de tipus torneig, inicialitzant la variable tournament que utilitzarem durant la partida.
- InitExhibitionGame, inicialitza una partida de tipus exhibició, on es barallaran dos personatges i una vegada acabada la partida tornarem al menú principal de joc.
- InitTournamentGame, inicialitza una baralla dins d'un torneig.
- ShowGUIDraw, mostra una eliminatòria a nivell de GUI, aquest mètode és estàtic i l'utilitzem tant quan mostrem les eliminatòries del torneig des de la pantalla de menú principal com des del joc.
- ShowGUIDrawWinners, mostra les eliminatòries que encara no tenen els dos lluitadors basant-se en els guanyadors de les eliminatòries prèvies.
- ShowGUICharacter, mostra el personatge que lluitarà en una eliminatòria.
- ShowGUITournament, mostra per pantalla el torneig complert, pintant totes les eliminatòries que conté el torneig.

3.6 Classe UOCHitInfo

Aquesta classe és una classe molt simple que únicament conté el propietari d'un cop. La utilitzem per establir-la com a component de les esferes que utilitzem per comprovar les col·lisions dels personatges.

Quan una esfera que conté aquest component topa contra un personatge comprovem el seu propietari, en cas de que el propietari sigui diferent del que ha detectat la col·lisió llavors notifiquem tant el propietari com el que rep el cop de l'èxit del cop.

3.7 Classe RobotPlayer

Per la implementació del control del personatge tant humà com de la intel·ligència artificial hem utilitzat una màquina finita d'estats.

El nostre personatge té tres tipus de cops de puny i tres tipus diferents de cop de peu que el jugador humà podrà executar segons la combinació de botons que faci servir.

La màquina d'estats està composada pels estats següents:

- IDLE, serà l'estat inicial i des d'on prendrem totes les decisions inicials.
- PUNCH, aquest estat defineix que el personatge està donant un cop de puny.
- KICK, aquest estat defineix que el personatge està donant un cop de peu.
- HIT, aquest estat s'executarà mentre el personatge sigui colpejat.
- CINEMATICS, aquest estat s'utilitzarà per deixar el personatge en un estat de repòs mentre mostrem informació d'inici i fi de partida.
- DEFFENSE, aquest estat s'executarà mentre el personatge estigui en posició de defensa.
- DIE, aquest estat defineix que el personatge està en animació de morir, previ a fi de partida.

- KEEP_RANGED, aquest estat l'utilitzem per fer endarrerir el personatge del seu rival.
- APPROACH_AND_ATTACK_SIMPLE, aquest estat estableix el personatge en l'estat d'apropar-se a l'oponent i una vegada suficientment prop executa un atac senzill.
- APPROACH_AND_COMBO_ATTACK, aquest estat estableix el personatge en l'estat d'apropar-se a l'oponent i una vegada suficientment prop executa un atac de tipus combo on barreja diferents atacs senzills.
- ATTACK_SIMPLE, aquest estat genera un atac senzill entre els diferents tipus d'atacs que té el personatge.
- ATTACK_COMBO aquest estat genera un atac senzill segons el tipus de combo que estigui executant.

Remarcar també que tenim diferents variables que permet, modificant el seus valors, tenir diferents comportaments per part dels enemics. Aquestes variables són:

- m_AggressivePct, segons el valor d'aquesta variable el nostre personatge serà més o menys agressiu.
- m_DeffensivePct, segons el valor d'aquesta variable el nostre personatge tindrà més o menys precisió a l'hora d'executar moviments defensius.
- m_KickPct, segons el valor d'aquesta variable podem fer que el personatge executi més cops de peu.
- m_PunchPct, segons el valor d'aquesta variable podem fer que el personatge executi més cops de puny.
- m_SimpleAttackPct, segons el valor d'aquesta variable la intel·ligència artificial realitzarà un atac de tipus senzill o de tipus combo.

El diagrama de la màquina d'estats quan el personatge és controlat per la intel·ligència artificial és el següent:

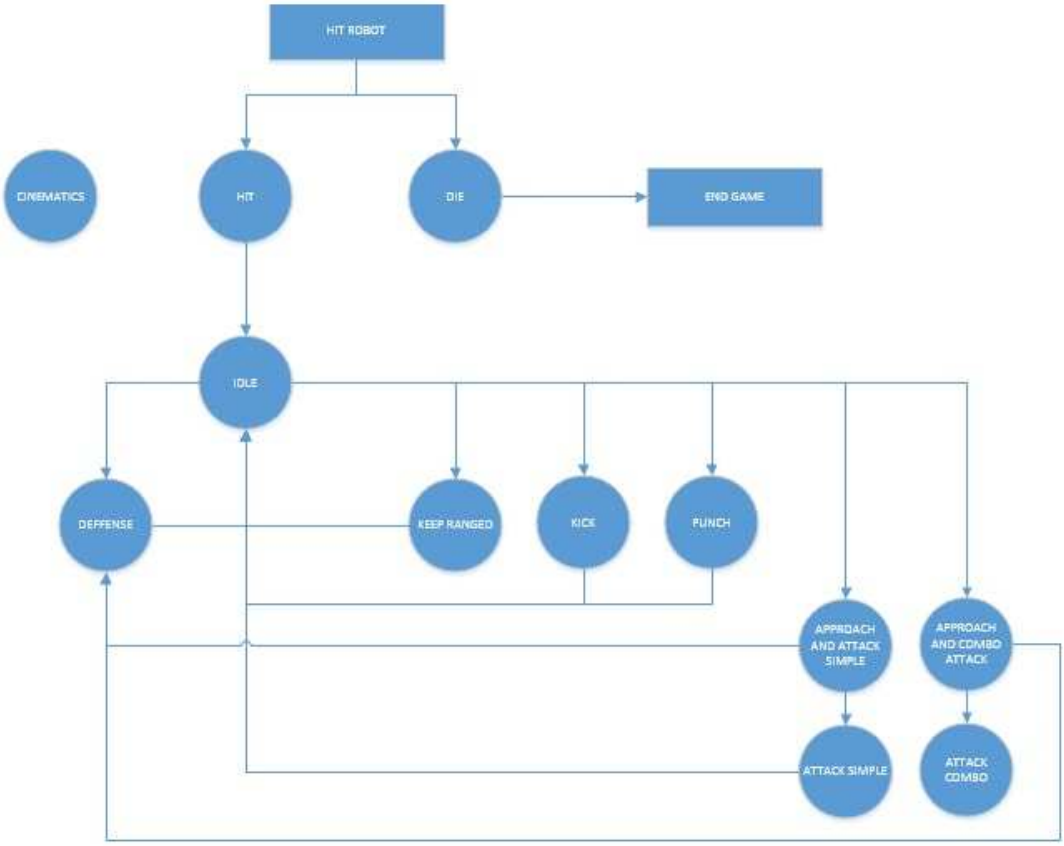

Fig. 15 Diagrama de flux de la IA

La màquina d'estats de la intel·ligència artificial la podem dividir en tres grans blocs:

- CINEMATICS, el personatge estarà en aquest estat mentre es presenta la partida o quan s'ha acabat la partida.
- HIT, DIE, quan el personatge rep un cop segons la vida que li resti passarà a l'estat de HIT si encara li resta vida o a l'estat de DIE si se li ha acabat la vida.
- IDLE, l'estat de IDLE l'utilitzem com a concentrador de pressa de decisions. Segons la les variables de la intel·ligència artificial el personatge passarà als diferents estats d'atac ja siguin senzills o de tipus combo, apropar-se i atacar o un moviment de fugida com és l'estat de KEEP_RANGED.

3.8 Classe UOCTournament

Per la implementació del torneig de joc, hem creat una classe templatitzada Torneig que passant-li una llista de jugadors i un mètode que simula un combat ens permet introduir tornejos.

Aquesta classe també utilitza una classe templatitzada CDraw que representa totes les eliminatòries i a la vegada aquesta classe utilitza una classe templatitzada CMatch que ens permet definir combats en el nostre videojoc.

3.9 Classe GUIController

Aquesta classe ens permet treballar amb la GUI del videojoc de forma més còmode sense haver-nos de preocupar de la resolució del dispositiu en que treballem.

En aquesta classe ens trobem amb els següents mètodes estàtics.

- GetRectangleGUI, a partir de la posició x, y, width, height en valors normalitzats ens retorna les coordenades de pantalla segons la resolució del joc actual.
- GetRectangleGUICentered, ens retorna el rectangle en coordenades de pantalla centrat a partir de la posició i mida donats en valors normalitzats.
- GetNormalizedPosition, ens retorna una posició normalitzada a partir d'un punt en coordenades de pantalla.
- GetNormalizedMousePosition, ens retorna la posició en coordenades normalitzades del ratolí.
- GetNormalizedTouchPosition, ens retorna la posició en coordenades normalitzades del cop de dit sobre la pantalla tàctil.

3.10 Classe UOCUtils

Aquesta classe la utilitzem com a suport en la implementació del projecte ens trobem només dos mètodes:

- ShuffleList, ens barreja un conjunt d'elements segons el número de vegades que li diem en el paràmetre count del mètode.

- WrapAngle, aquest mètode estableix un angle de tipus radians entre 0 i 2 PI Radians.

3.11 Classe UOCVirtualButton

Aquesta classe ens permet generar botons virtuals que funcionin tant en una plataforma com Windows o MAC, com en una pantalla multi tàctil com seria Android o iOS.

3.12 Classe UOCVirtualJoystick

Si la classe anterior la fèiem servir per controlar botons virtuals, ara ens trobem en una classe que gestionarà un joystick virtual que pot funcionar tant en Windows o MAC mitjançant tecles, com en una pantalla multi tàctil com seria Android o iOS.

3.13 Ombres sobre pla transparent

Comentar que gràcies al fet d'utilitzar un motor gràfic com Unity 3D on actualment hi ha tants desenvolupadors realitzant videojocs comercials podem trobar infinitat de recursos gratuïts per a utilitzar en el nostre videojoc.

Hem utilitzat un shader per poder projectar ombres sobre l'escenari real dels models 3D virtuals. La informació d'aquest shader l'hem trobat a l'enllaç 7.

4. Resultats

4.1 Pacs anteriors

La primera PAC vam definir el projecte a nivell de que volíem fer i com.

La segona PAC vam desenvolupar una demo sota plataforma Android utilitzant la llibreria de visió per computador Vuforia de Qualcomm.

Aquesta demo detectava la marca que ens generava l'espai tridimensional de l'entorn virtual sobre l'entorn real i mostrava models 3D sobre aquest espai.

En la següent imatge podem veure l'entorn real amb la marca.

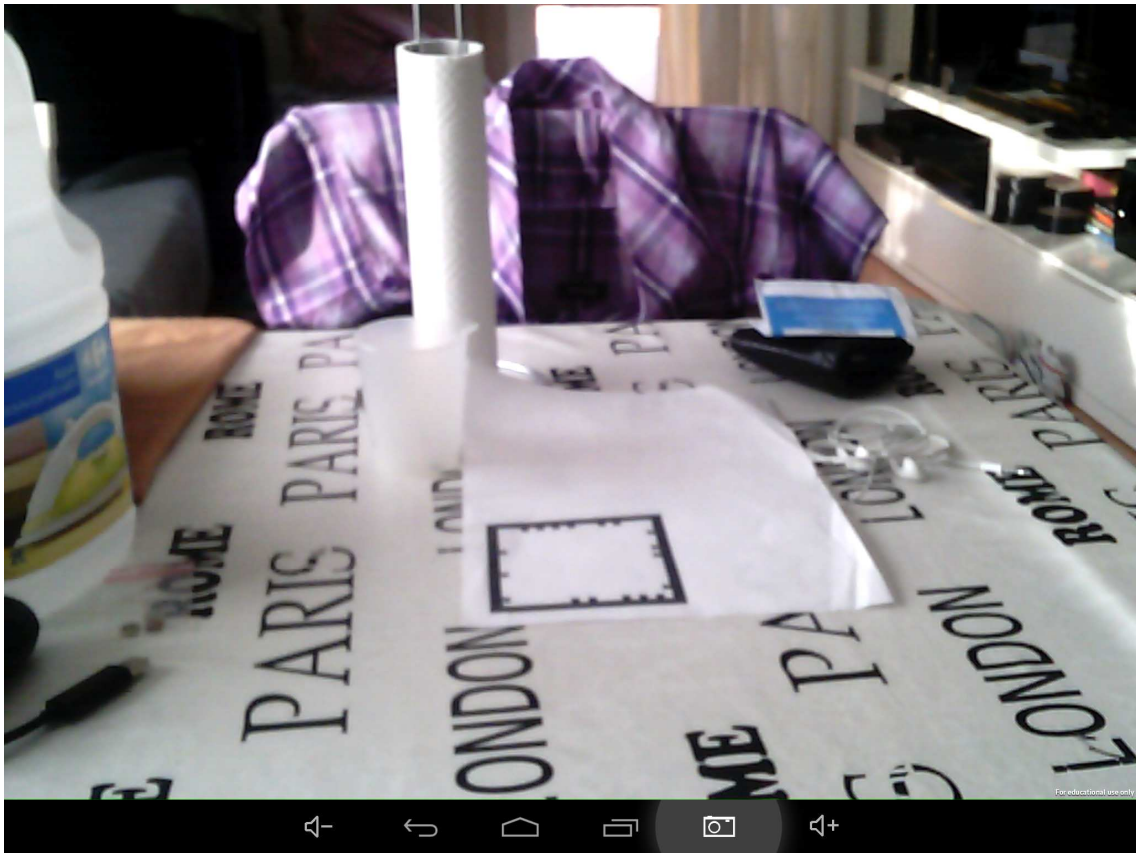

Fig. 16 Marca sobre entorn real

I a continuació podem veure una segona imatge amb l'entorn virtual augmentant l'entorn real.

Fig. 17 Imatge del joc de l'entrega 2

En la PAC 3 vam implementar la intel·ligència artificial del personatge rival així com una primera versió de la GUI del videojoc.

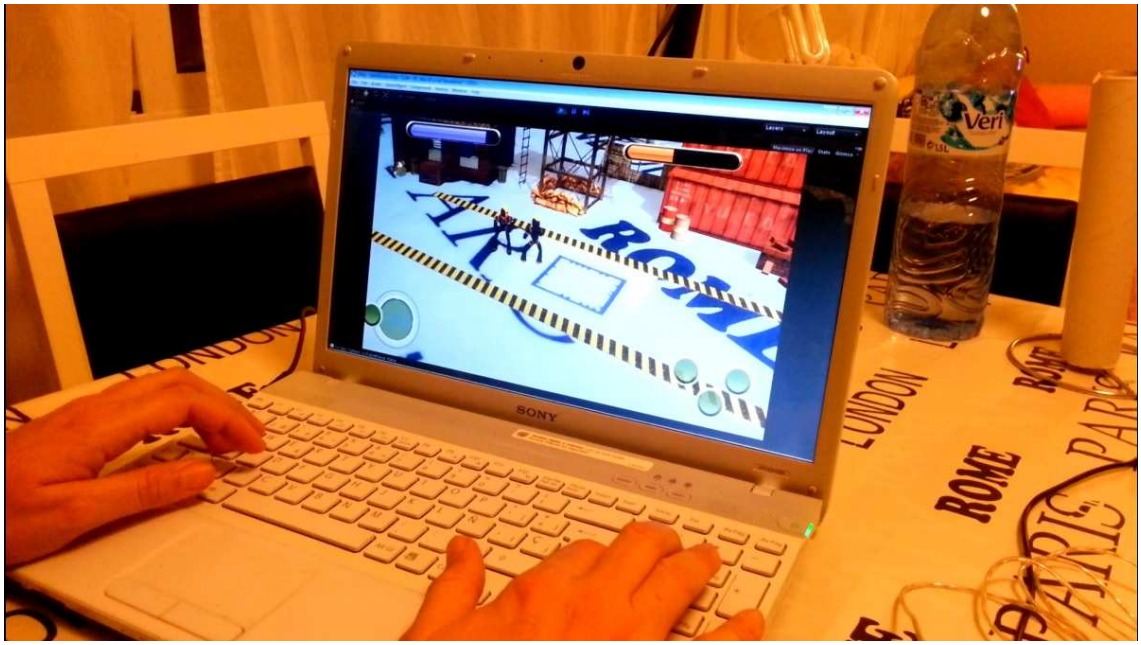

Fig. 18 Imatge del joc en l'entrega 3

Podem veure un vídeo del resultat de la PAC 3 en l'enllaç 8.

4.2 Projecte final

El resultat final del projecte és un videojoc amb dos modes de joc que pot escollir el jugador com es veu a la següent figura.

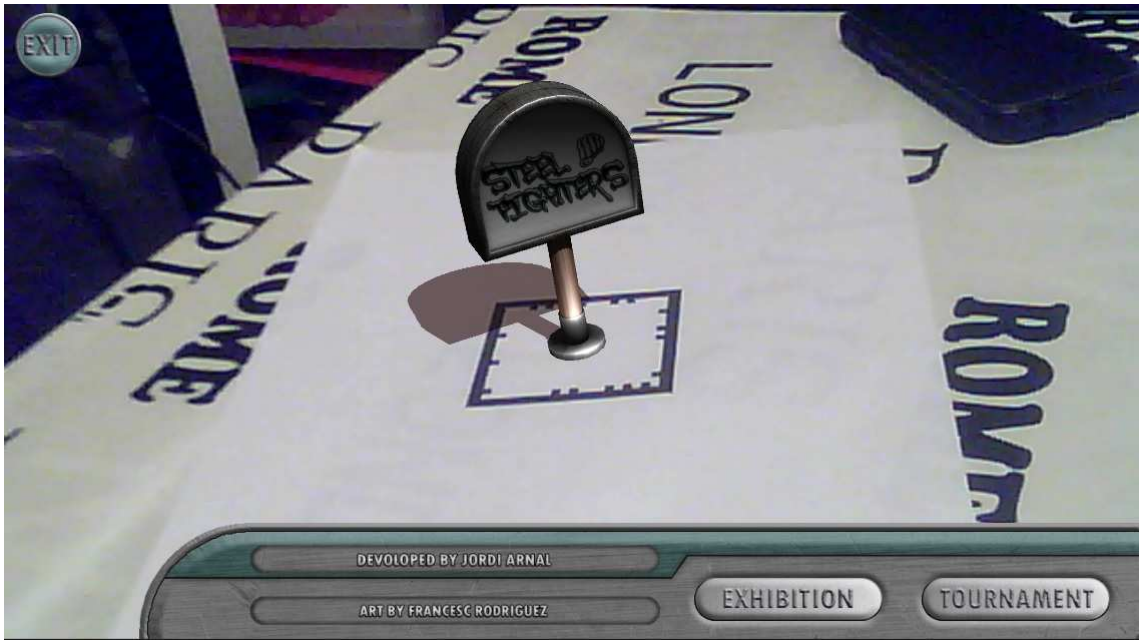

Fig. 19 Imatge del menú principal

Una vegada el jugador ha escollit qualsevol dels dos modes passa a la pantalla de selecció de personatge, tal com es veu en la següent figura.

Fig. 20 Imatge de selecció de personatge

En cas d'haver escollit el mode torneig passaríem a la pantalla on el jugador podrà veure les eliminatòries del torneig.

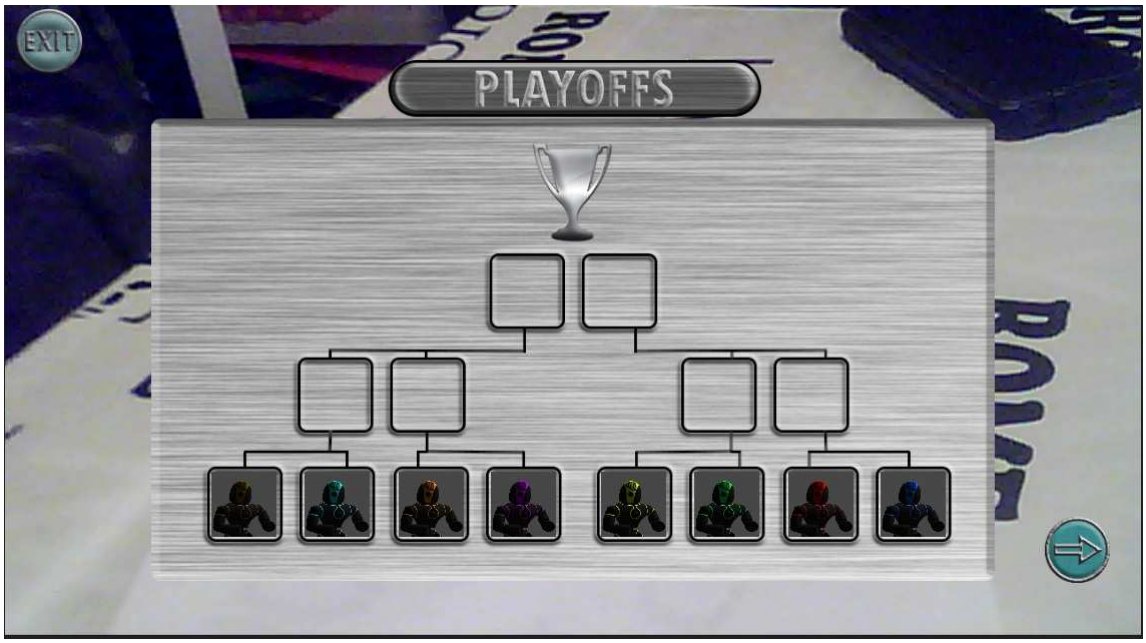

Fig. 21 Imatge de la pantalla de torneig

Per últim passarem a la pantalla de joc, on es desenvoluparà la batalla entre els dos personatges tal com podem veure en la següent imatge.

Fig. 22 Imatge del joc

5. Conclusions

Com a conclusió d'aquest projecte podem dir que hem assolit els reptes amb els que vam començar el desenvolupament.

La idea principal del projecte era desenvolupar un petit videojoc on s'utilitzés la visió per computador per detectar una marca i integrar un entorn virtual dins del món real.

Per altra banda, volíem crear un videojoc de combats on el jugador humà pogués lluitar contra intel·ligències artificials controlades pel dispositiu. Punt que hem assolit mitjançant la integració de vuit intel·ligències artificials de combat diferents que poden lluitar contra el jugador humà, així com dos modes de joc tant exhibició com Torneig.

Per últim, volíem fer el desplegament del projecte a tauletes tàctils amb sistema operatiu Android. Punt que també hem assolit gràcies a l'ús del motor de videojocs Unity 3D.

5.1 Millores

Com a comentari final voldria afegir un conjunt de millores que es podria afegir dins del projecte, però que molt probablement s'allunyaria de l'abast d'un projecte de fi de carrera com aquest.

- Millorar la classe GUIController, ara mateix aquesta classe no es preocupa de l'aspect ratio del dispositiu on s'executa el videojoc i això afecta a que si el dispositiu no és 16:9 les imatges es veuen deformades.
- El control del personatge es podria ampliar amb nous moviments i cops especials segons el personatge escollit. Especialment hem trobat que ens ha faltat animacions de cops tant ajupits com en salt que hagués donat una profunditat molt més gran al control i a la intel·ligència artificial.

- Ens hagués agradat millorar la intel·ligència artificial i basant-nos en el concepte del joc Pedra-Paper-Tisores on tenim tres cops que cap d'ells és superior als altres si no que un guanya amb un segon, però perd amb un tercer, la intel·ligència artificial de videojocs clàssics de lluita com Street Fighter es basava en aquest concepte.
- Es podria haver ampliat el modes de joc, introduint un mode història o un mode de jugabilitat en línia per a diferents jugadors humans.
- Altre element que es podria millorar és la utilització de cinemàtiques d'entrada dels personatges en l'arena que faria el joc molt més espectacular.
- La última millora que comentaré seria la d'haver introduït diferents escenaris, normalment en els videojocs de lluita cada personatge té el seu propi escenari amb el que interactua.

Annex Instal·lació Qualcomm Vuforia

La instal·lació de l'extensió de Unity de Vufori que ens permet integrar visió per computador en el projecte ha sigut molt senzilla.

En l'enllaç 9 podem trobar la informació per descarregar i introduir l'extensió.

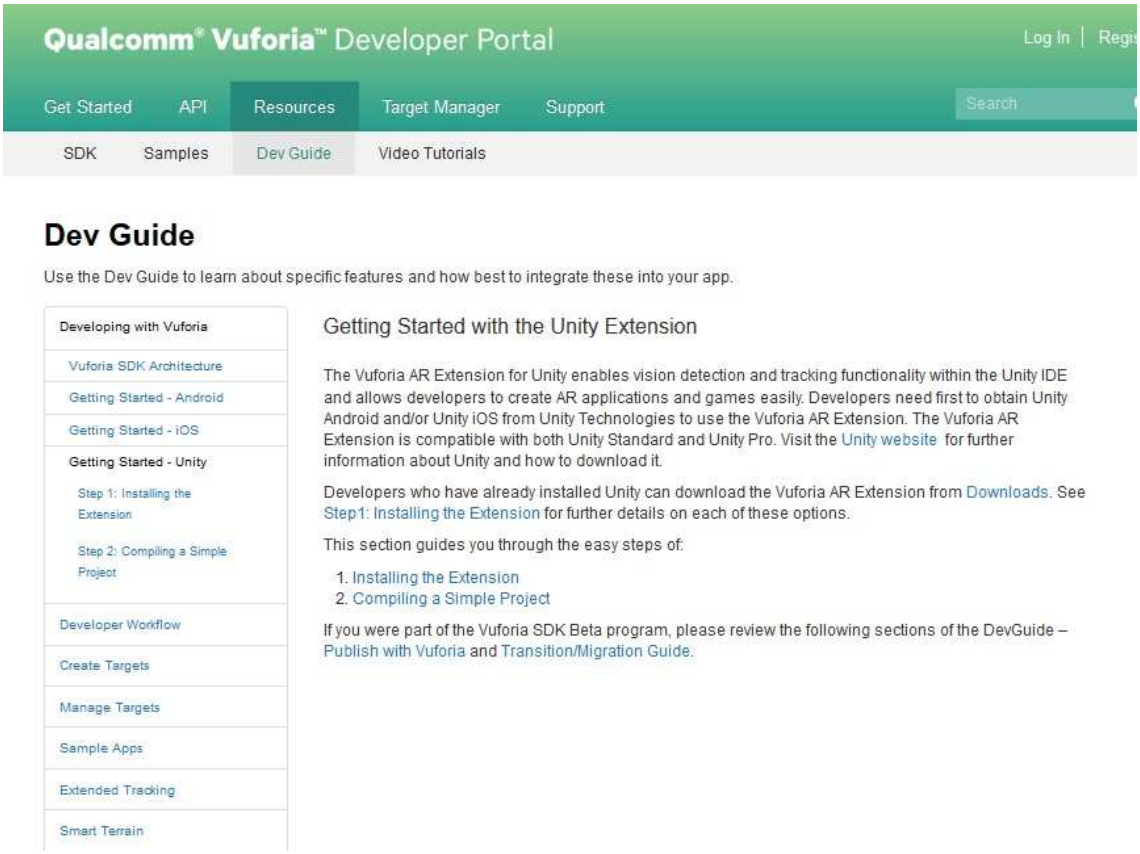

Fig. 23 Web de Vuforia

Una vegada descarregat l'extensió hem d'introduir-la dins del nostre projecte, per fer això anirem al menú Assets i polsarem sobre l'opció de Import Package/Custom packages com es veu en la següent imatge.

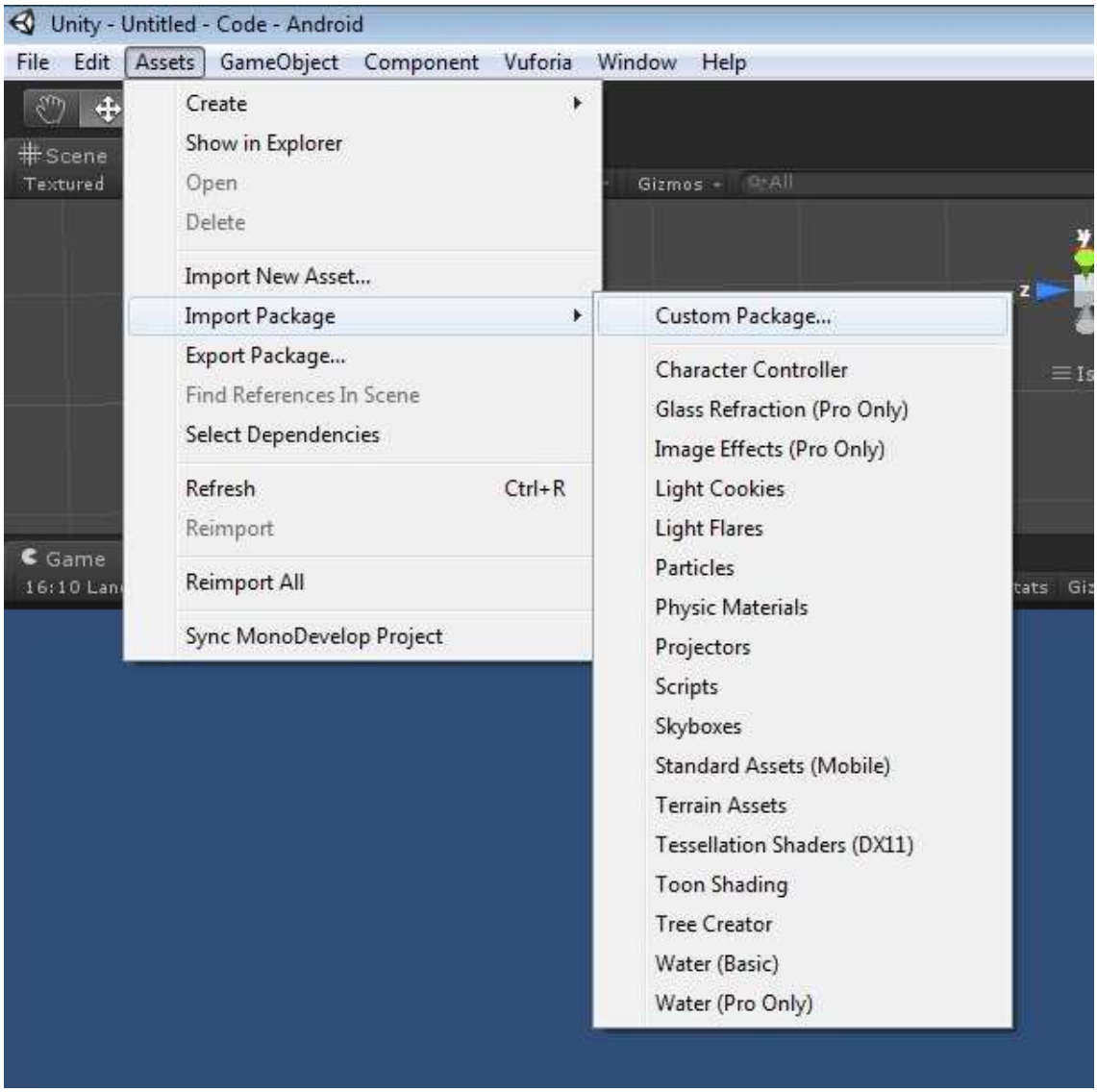

Fig. 24 Importació d'un custom package a Unity

Al polsar sobre l'opció se'ns demanarà que escollim l'arxiu de tipus package que volem afegir al nostre projecte. Seleccionem el fitxer que hem descarregat de la pàgina de Vuforia i ens sortirà una finestra com la següent on simplement li direm que importi els fitxers que per defecte surten seleccionats.

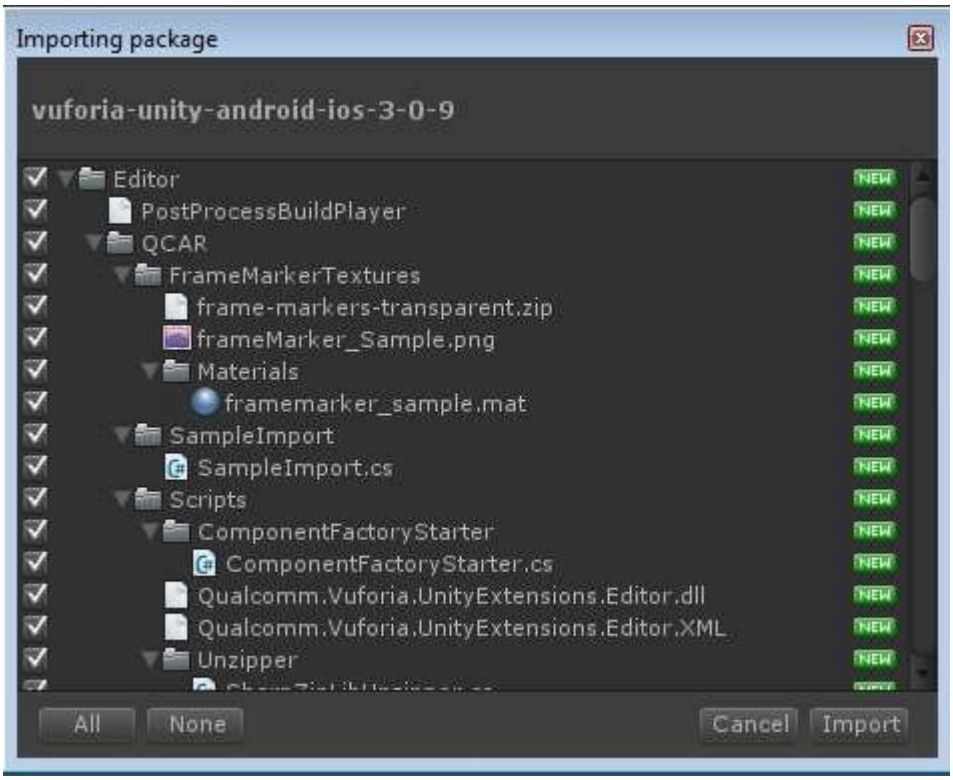

Fig. 25 Importació d'un custom package a Unity segona part

Una vegada tenim importada l'extensió de Vuforia passem a introduir-la en la nostra escena, per això escollim el prefab ARCamera que es troba dins de la carpeta del projecte Qualcomm/Prefabs i l'arrastrem a la jerarquia la nostra escena.

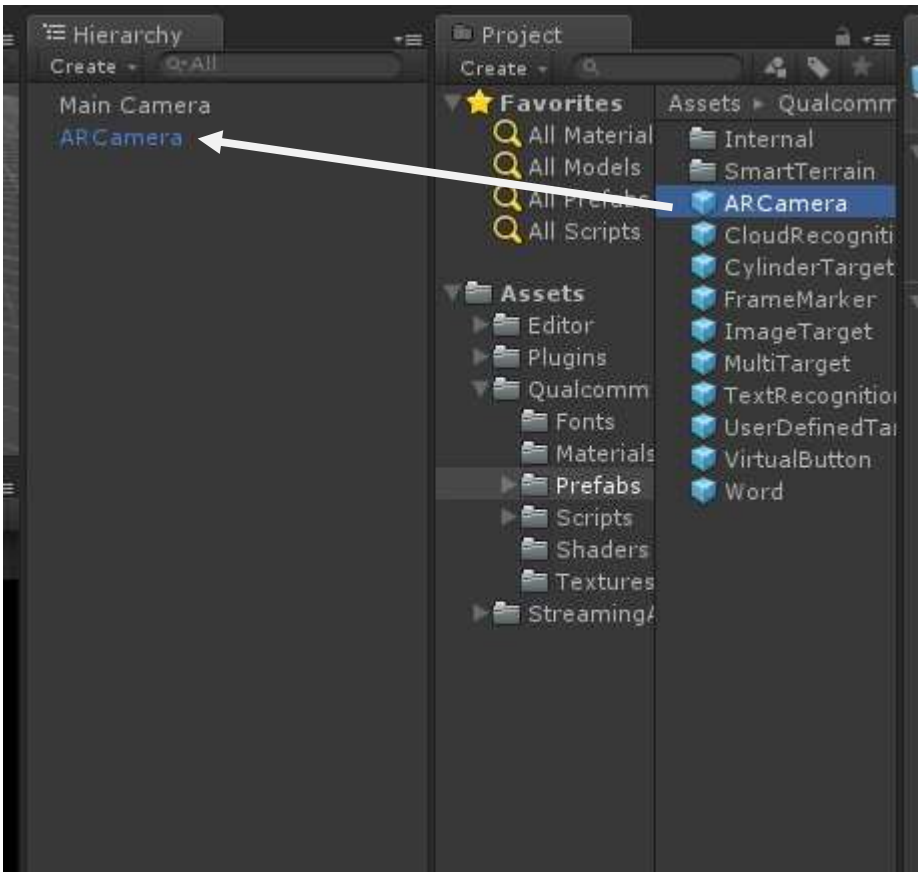

Fig. 26 Inserció d'una càmera AR a la nostra jerarquia d'escena

Ara esborrem la càmera que se'ns ha creat per defecte en la nostra escena.

Fig. 27 Main camera en Hierarchy de la nostra escena

A continuació inserim el prefab FrameMarker que es troba en la mateixa carpeta Qualcom/Prefabs i canviem l'escala en l'inspector a 1.

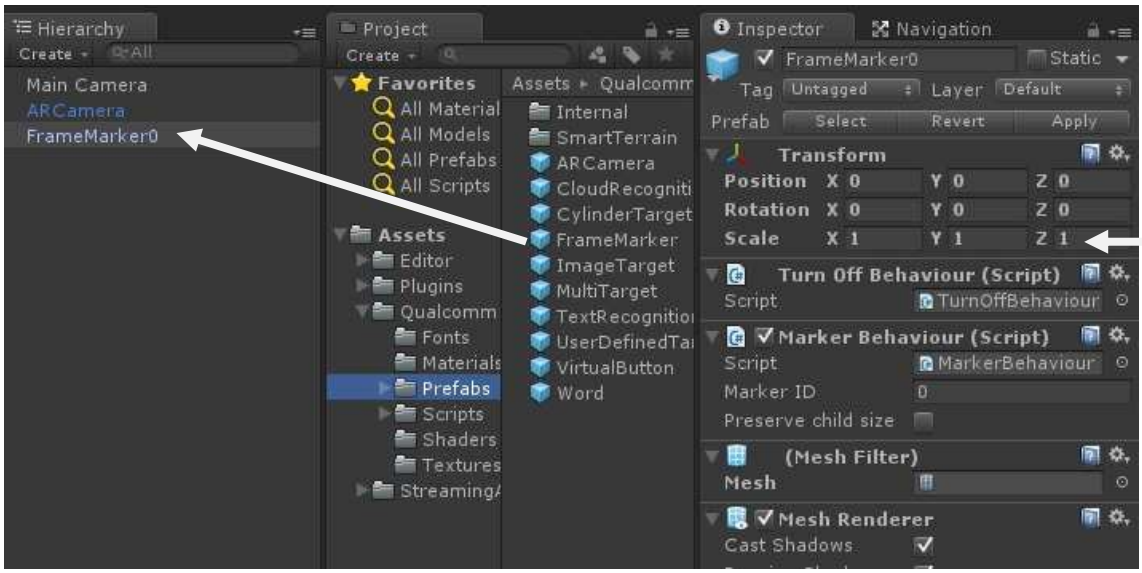

Fig. 28 Inserció del marcador a la nostra escena

Per últim només ens faltarà generar un entorn 3D que s'incrustarà a sobre del món real. Per això hem introduït una llum, una caixa i una esfera dins de l'escena i aquest és el resultat sense el món virtual.

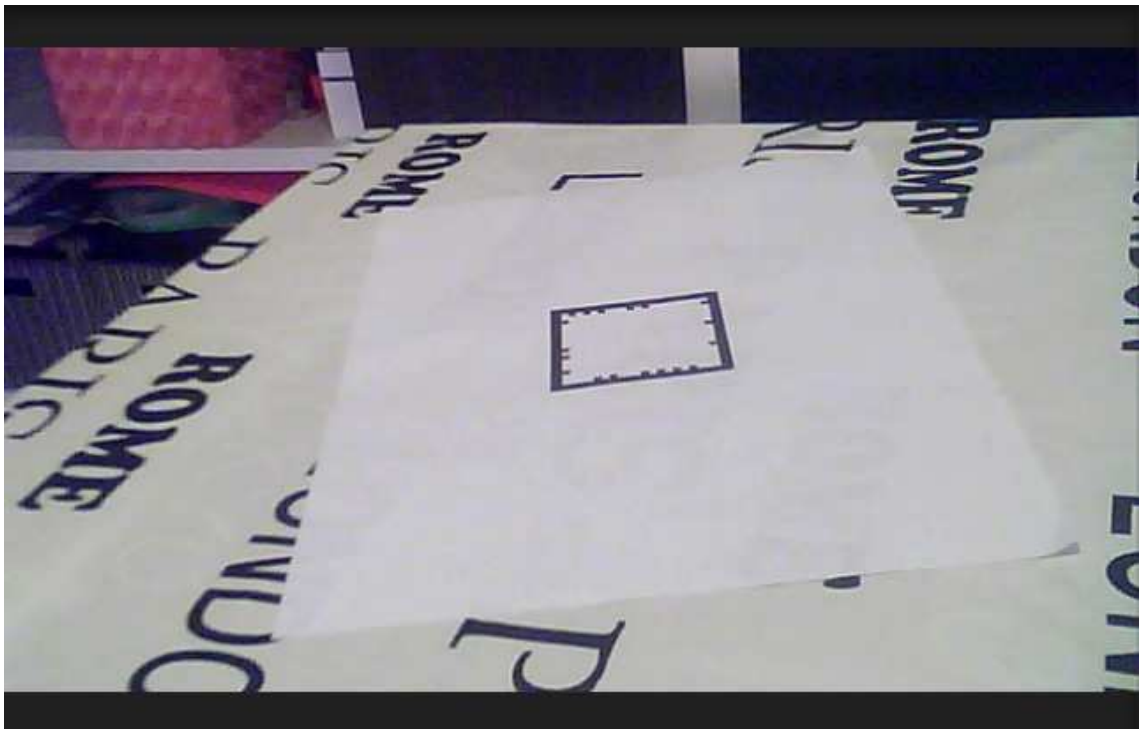

Fig. 29 Escena de test amb marca en món real

La mateixa escena amb l'entorn virtual augmentant el món real.

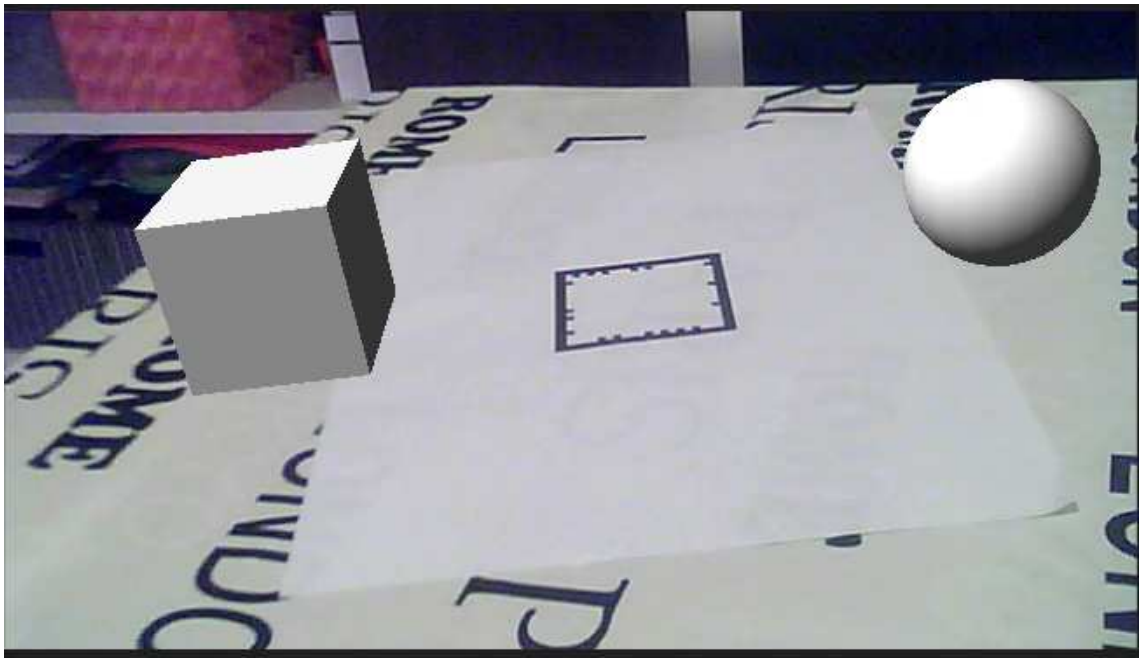

Fig. 30 Escena de test amb món real augmentat amb món virtual

Annex codi font

FightersGameControl.cs

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using UOCUtils;
using UOCTournament;

namespace UOCFightersGame
{
	public class CFighterPlayer
	{
		public Texture2D m_ProfileTexture;
		public string m_CharacterName;
	 public bool m_Human;
	 public int m_Id;

	 public float m_MinDecissionTime;
	 public float m_RandomDecissionTime;
	    public float m_AggressivePct;
	    public float m_DeffensivePct;
	    public float m_KickPct;
	    public float m_PunchPct;
	    public float m_SimpleAttackPct;

	 public CFighterPlayer(string CharacterName, Texture2D ProfileTexture,
bool Human, int Id, float MinDecissionTime, float RandomDecissionTime, float
AggressivePct, float DeffensivePct, float KickPct, float PunchPct, float
SimpleAttackPct)
		{
			m_CharacterName=CharacterName;
			m_ProfileTexture=ProfileTexture;
		m_Human=Human;
		m_Id=Id;

		m_MinDecissionTime=MinDecissionTime;
		m_RandomDecissionTime=RandomDecissionTime;
		   m_AggressivePct=AggressivePct;
		   m_DeffensivePct=DeffensivePct;
		   m_KickPct=KickPct;
		   m_PunchPct=PunchPct;
		   m_SimpleAttackPct=SimpleAttackPct;
		  }
		  public static CFighterPlayer SimulateFight(CFighterPlayer Player1,
CFighterPlayer Player2)
		{
			float l_MaxValues=100.0f;
			float l_Winner=UnityEngine.Random.value*l_MaxValues*2.0f;
			if(l_Winner>l_MaxValues)
				return Player1;
			else
				return Player2;
		}
	}
	public class CFighterGame
	{
		public enum TGameType
	 {
	     EXHIBITION,
```

```
            TOURNAMENT
        }

        static private CFighterGame m_FighterGame=new CFighterGame();
              private CTournament<CFighterPlayer> m_Tournament=null;
              private bool m_OnTournament=false;
        private CMatch<CFighterPlayer> m_CurrentMatch=null;
        List<CFighterPlayer> m_Players;
        private TGameType m_GameType=TGameType.EXHIBITION;
        static List<List<Vector2>> m_DrawPositions;

              static public CFighterGame GetFighterGame()
              {
                    return m_FighterGame;
              }
              private CFighterGame()
              {
              }
        public void InitPlayers()
        {
            m_Players=new List<CFighterPlayer>();

            m_Players.Add(new CFighterPlayer("FRANKIE",
Resources.Load<Texture2D>("Characters/Player1"), false, 0, 0.4f, 0.5f, 0.75f,
0.65f, 0.4f, 0.6f, 0.7f));
            m_Players.Add(new CFighterPlayer("TINNY",
Resources.Load<Texture2D>("Characters/Player2"), false, 1, 0.6f, 0.7f, 0.6f,
0.75f, 0.2f, 0.8f, 0.6f));
            m_Players.Add(new CFighterPlayer("ERIC",
Resources.Load<Texture2D>("Characters/Player3"), false, 2, 0.4f, 0.5f, 0.65f,
0.65f, 0.5f, 0.5f, 0.65f));
            m_Players.Add(new CFighterPlayer("KEGEE",
Resources.Load<Texture2D>("Characters/Player4"), false, 3, 0.6f, 0.7f, 0.5f,
0.8f, 0.7f, 0.3f, 0.7f));

            m_Players.Add(new CFighterPlayer("EWOK",
Resources.Load<Texture2D>("Characters/Player5"), false, 4, 0.2f, 0.2f, 0.9f,
0.55f, 0.4f, 0.6f, 0.3f));
            m_Players.Add(new CFighterPlayer("FERROID",
Resources.Load<Texture2D>("Characters/Player6"), false, 5, 0.2f, 0.3f, 0.85f,
0.5f, 0.2f, 0.8f, 0.35f));
            m_Players.Add(new CFighterPlayer("D'ARIO",
Resources.Load<Texture2D>("Characters/Player7"), false, 6, 0.1f, 0.3f, 0.9f,
0.45f, 0.5f, 0.5f, 0.25f));
            m_Players.Add(new CFighterPlayer("LACOYA",
Resources.Load<Texture2D>("Characters/Player8"), false, 7, 0.1f, 0.2f, 0.95f,
0.4f, 0.7f, 0.3f, 0.2f));
        }
        public List<CFighterPlayer> GetPlayers()
        {
            if(m_Players==null)
                InitPlayers();
            return m_Players;
        }
              public void InitTournament(int IdPlayer)
              {
            InitPlayers();
            m_Players[IdPlayer].m_Human=true;
            m_Tournament=new CTournament<CFighterPlayer>(m_Players, new
CTournament<CFighterPlayer>.SimulateMatch(CFighterPlayer.SimulateFight), true);
              }
#if UNITY_EDITOR
```

```
            public void CheckTournament()
            {
                if(m_Tournament==null)
                    InitTournament(0);
            }
#endif
            public CTournament<CFighterPlayer> GetTournament()
            {
                return m_Tournament;
            }
            public void SetOnTournament(bool OnTournament)
            {
                m_OnTournament=OnTournament;
            }
            public bool IsOnTournament()
            {
                return m_OnTournament;
            }
        public void SetGameType(TGameType GameType)
        {
            m_GameType=GameType;
        }
        public TGameType GetGameType()
        {
            return m_GameType;
        }
        public void InitExhibitionGame(int HumanPlayerId, int OpponentPlayerId)
        {
            List<CFighterPlayer> l_Players=GetPlayers();
            SetGameType(TGameType.EXHIBITION);
            l_Players[HumanPlayerId].m_Human=true;
            l_Players[OpponentPlayerId].m_Human=false;
            m_CurrentMatch=new CMatch<CFighterPlayer>(l_Players[HumanPlayerId],
l_Players[OpponentPlayerId], null, null);
            Debug.Log("Exhibition match "+HumanPlayerId+" "+OpponentPlayerId);
        }
        public void InitTournamentGame()
        {
            SetGameType(TGameType.TOURNAMENT);
            GetPlayers();
            m_CurrentMatch=m_Tournament.GetCurrentMatch();
        }
        public CMatch<CFighterPlayer> GetCurrentMatch()
        {
            return m_CurrentMatch;
        }
        static void ShowGUIDraw(List<Vector2> Positions, CDraw<CFighterPlayer>
Draw)
        {
            List<CMatch<CFighterPlayer>> l_Matches=Draw.GetMatches();
            List<CFighterPlayer> l_Players=new List<CFighterPlayer>();
            for(int i=0;i<l_Matches.Count;++i)
            {
                l_Players.Add(l_Matches[i].GetPlayer1());
                l_Players.Add(l_Matches[i].GetPlayer2());
            }
            ShowGUIDraw(Positions, l_Players);
        }
        static void ShowGUIDrawWinners(List<Vector2> Positions,
CDraw<CFighterPlayer> Draw)
        {
            List<CMatch<CFighterPlayer>> l_Matches=Draw.GetMatches();
```

```
            List<CFighterPlayer> l_Players=new List<CFighterPlayer>();
            for(int i=0;i<l_Matches.Count;++i)
            {
                CFighterPlayer l_Winner=l_Matches[i].GetWinner();
                if(l_Winner!=null)
                    l_Players.Add(l_Winner);
            }
            ShowGUIDraw(Positions, l_Players);
        }
        static void ShowGUIDraw(List<Vector2> Positions, List<CFighterPlayer>
Players)
        {
            for(int i=0;i<Players.Count;++i)
                ShowGUICharacter(Positions[i], Players[i].m_ProfileTexture);
        }
        static void ShowGUICharacter(Vector2 Position, Texture2D
_CharacterTexture)
        {
            float l_OffsetTexture=0.1f;
            Vector2 l_Size=new Vector2(0.07734375f, 0.1375f)*0.92f;
            Rect l_GUITextureRect=GUIController.GetRectangleGUI(Position.x,
Position.y, l_Size.x-l_OffsetTexture*l_Size.x*2.0f, l_Size.y-
l_OffsetTexture*l_Size.y*2.0f);
            GUI.DrawTexture(l_GUITextureRect, _CharacterTexture);
        }
        public static void ShowGUITournament(Texture2D PlayoffBackgroundTexture)
        {
            GUI.DrawTexture(GUIController.GetRectangleGUI(0.12890625f, 0.1f,
0.7421875f, 0.8152777777777777777777777777778f), PlayoffBackgroundTexture);

            CDraw<CFighterPlayer>
l_Draw=CFighterGame.GetFighterGame().GetTournament().GetMainDraw();
            int l_IdCurrentDraw=0;
            while(l_Draw!=null)
            {
                ShowGUIDraw(m_DrawPositions[l_IdCurrentDraw], l_Draw);
                CDraw<CFighterPlayer> l_PreviousDraw=l_Draw;
                l_Draw=l_Draw.GetNextDraw();
                ++l_IdCurrentDraw;
                if(l_Draw==null && l_IdCurrentDraw<3)
                    ShowGUIDrawWinners(m_DrawPositions[l_IdCurrentDraw],
l_PreviousDraw);
            }
        }
        static void InitTournamentPositions()
        {
            m_DrawPositions=new List<List<Vector2>>();
            //Quarter final
            m_DrawPositions.Add(new List<Vector2>());
            m_DrawPositions[0].Add(new Vector2(0.182f, 0.745f));
            m_DrawPositions[0].Add(new Vector2(0.264f, 0.745f));
            m_DrawPositions[0].Add(new Vector2(0.347f, 0.745f));
            m_DrawPositions[0].Add(new Vector2(0.428f, 0.745f));

            m_DrawPositions[0].Add(new Vector2(0.525f, 0.745f));
            m_DrawPositions[0].Add(new Vector2(0.607f, 0.745f));
            m_DrawPositions[0].Add(new Vector2(0.690f, 0.745f));
            m_DrawPositions[0].Add(new Vector2(0.771f, 0.745f));
            //Semi final
            m_DrawPositions.Add(new List<Vector2>());
            m_DrawPositions[1].Add(new Vector2(0.262f, 0.574f));
            m_DrawPositions[1].Add(new Vector2(0.343f, 0.574f));
```

```
            m_DrawPositions[1].Add(new Vector2(0.5998f, 0.574f));
            m_DrawPositions[1].Add(new Vector2(0.681f, 0.574f));

            //Final
            m_DrawPositions.Add(new List<Vector2>());
            m_DrawPositions[2].Add(new Vector2(0.431f, 0.412f));
            m_DrawPositions[2].Add(new Vector2(0.512f, 0.412f));
        }
        public static void CheckDrawPositions()
        {
            if(m_DrawPositions==null)
                InitTournamentPositions();
        }
       }
}
```

GameScreen.cs

```
//#define UOC_CREATE_HOT_TOURNAMENT

using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using UOCUtils;
using UOCFightersGame;
using UOCTournament;

public class GameScreen : MonoBehaviour
{
    public GUISkin m_Skin;
    public RobotPlayer m_Player1;
    public RobotPlayer m_Player2;

    public Texture2D m_BarBackground;
    public Texture2D m_BarFillPlayer1;
    public Texture2D m_BarFillPlayer2;

    public Texture2D m_FightTexture;

    public GameObject m_RobotPrefab;
    public Transform m_Robot1Transform;
    public Transform m_Robot2Transform;

    public UOCVirtualJoystick m_Joystick;
    public UOCVirtualButton m_Kick;
    public UOCVirtualButton m_Punch;
    public UOCVirtualButton m_Deffense;
    public Texture2D m_YouWinTexture;
    public Texture2D m_YouLoseTexture;
    public AudioClip m_FightSound;
    public AudioClip m_YouWinSound;
    public AudioClip m_YouLoseSound;

    public float m_CurrentTime=0.0f;

    public List<Material> m_Materials;

    public bool m_YouWin;

    CFadeIn m_FadeIn;
    CFadeOut m_FadeOut;
```

```
    public float m_FadeTime=0.8f;

    public enum TGameState
    {
        SHOW_ROUND,
        IN_GAME,
        END_ROUND,
        SHOW_PLAYOFFS,
        GOTO_MAIN_MENU,
        GOTO_GAME_FROM_PLAYOFFS,
    }
    public TGameState m_GameState;
    public TGameState m_NextGameState;

    public Texture2D m_PlayoffBackgroundTexture;
    public AudioClip m_ClickSound;
    bool m_FadeInVolume=true;

    void Start()
    {
        audio.volume=0.0f;
        audio.Play();
        m_FadeIn=new CFadeIn();
        m_FadeOut= new CFadeOut();
        m_FadeInVolume=true;

#if UOC_CREATE_HOT_TOURNAMENT
        InitHotTournament();
        CFighterGame.CheckDrawPositions();
        //CFighterGame.GetFighterGame().CheckTournament();
#endif

        InitPlayers();

        SetShowRoundGameState();
      }
    void InitHotTournament()
    {
        CFighterGame.GetFighterGame().InitTournament(0);
        CFighterGame.GetFighterGame().InitTournamentGame();
    }
#if UNITY_EDITOR
    void CheckHotValues()
    {
        if(m_FadeIn==null)
            m_FadeIn=new CFadeIn();
        if(m_FadeOut==null)
            m_FadeOut= new CFadeOut();

        CFighterGame.CheckDrawPositions();
#if UOC_CREATE_HOT_TOURNAMENT
        CFighterGame.GetFighterGame().CheckTournament();
#endif
    }
#endif
      void Update()
    {
#if UNITY_EDITOR
        CheckHotValues();
#endif
      }
    void OnGUI()
```

```
    {
        if(Application.isLoadingLevel)
        {
            GUI.DrawTexture(GUIController.GetRectangleGUI(0.0f, 0.0f, 1.0f,
1.0f), CFadeOut.m_BlackTexture);
            return;
        }
        GUI.skin=m_Skin;
#if UNITY_EDITOR
        CheckHotValues();
#endif
        switch(m_GameState)
        {
            case TGameState.IN_GAME:
                ShowInGameScreen();
                break;
            case TGameState.SHOW_ROUND:
                ShowShowRoundScreen();
                break;
            case TGameState.END_ROUND:
                ShowShowEndRoundScreen();
                break;
            case TGameState.SHOW_PLAYOFFS:
                ShowPlayoffsScreen();
                break;
        }
    }
    CMatch<CFighterPlayer> GetCurrentMatch()
    {
        CFighterGame l_FighterGame=CFighterGame.GetFighterGame();
        if(l_FighterGame.GetGameType()==CFighterGame.TGameType.EXHIBITION)
            return l_FighterGame.GetCurrentMatch();
        else
            return l_FighterGame.GetTournament().GetCurrentMatch();
    }
    void ShowInGameScreen()
    {
        float l_Scale=1.0f;
        float l_ScaleProfile=0.15f;
        CMatch<CFighterPlayer> l_CurrentMatch=GetCurrentMatch();
        CFighterPlayer l_Player1=l_CurrentMatch.GetPlayer1();
        CFighterPlayer l_Player2=l_CurrentMatch.GetPlayer2();
        if(!l_Player1.m_Human)
        {
            CFighterPlayer l_Aux=l_Player1;
            l_Player1=l_Player2;
            l_Player2=l_Aux;
        }

        GUI.Label(GUIController.GetRectangleGUI(0.02f, 0.08f, 0.3609375f*l_Scale,
0.15f), l_Player1.m_CharacterName, "RightNameStyle");
        GUI.DrawTexture(GUIController.GetRectangleGUI(0.4f, 0.03f,
0.390625f*l_ScaleProfile, 0.8375f*l_ScaleProfile), l_Player1.m_ProfileTexture);

        GUI.DrawTexture(GUIController.GetRectangleGUICentered(0.2f, 0.05f,
0.3609375f*l_Scale, 0.044444444444444444444444444444444f*l_Scale),
m_BarBackground);
        Rect l_Player1Rect=GUIController.GetRectangleGUICentered(0.2f, 0.05f,
0.35390625f*l_Scale, 0.036111111111111111111111111111111f*l_Scale);
        float l_Pct1=m_Player1.GetCurrentLife();
```

```
        GUI.DrawTextureWithTexCoords(new Rect(l_Player1Rect.x, l_Player1Rect.y,
l_Player1Rect.width*l_Pct1, l_Player1Rect.height), m_BarFillPlayer1, new
Rect(0.0f, 0.0f, l_Pct1, 1.0f));

        GUI.Label(GUIController.GetRectangleGUI(0.62f, 0.08f, 0.3609375f*l_Scale,
0.15f), l_Player2.m_CharacterName, "LeftNameStyle");
        GUI.DrawTexture(GUIController.GetRectangleGUI(0.54140625f, 0.03f,
0.390625f*l_ScaleProfile, 0.8375f*l_ScaleProfile), l_Player2.m_ProfileTexture);

        GUI.DrawTexture(GUIController.GetRectangleGUICentered(0.8f, 0.05f,
0.3609375f*l_Scale, 0.044444444444444444444444444444444f*l_Scale),
m_BarBackground);
        Rect l_Player2Rect=GUIController.GetRectangleGUICentered(0.8f, 0.05f,
0.35390625f*l_Scale, 0.036111111111111111111111111111111f*l_Scale);
        float l_Pct2=m_Player2.GetCurrentLife();
        GUI.DrawTextureWithTexCoords(new Rect(l_Player2Rect.x, l_Player2Rect.y,
l_Player2Rect.width*l_Pct2, l_Player2Rect.height), m_BarFillPlayer2, new
Rect(0.0f, 0.0f, l_Pct2, 1.0f));
    }
    void ShowShowRoundScreen()
    {
        float l_IntroTime=1.5f;
        float l_ExtroStartTime=4.5f;
        float l_ExtroTime=1.5f;
        float l_IntroOffset=1.0f;
        float l_ExtroOffset=1.0f;

        m_CurrentTime+=Time.deltaTime;
        float l_IntroOffsetStart=l_IntroOffset*Mathf.Cos((1.0f-Mathf.Max(0.0f,
(l_IntroTime-m_CurrentTime)/l_IntroTime))*Mathf.PI*0.5f);
        float l_ExtroOffsetStart=-l_ExtroOffset*Mathf.Cos((1.0f-
Mathf.Clamp((m_CurrentTime-l_ExtroStartTime)/l_ExtroTime, 0.0f,
1.0f))*Mathf.PI*0.5f);

GUI.DrawTexture(GUIController.GetRectangleGUICentered(0.5f+l_IntroOffsetStart+l_E
xtroOffsetStart, 0.3f, 0.2390625f, 0.13888888888888888888888888888889f),
m_FightTexture);
        if(m_CurrentTime>=(l_ExtroStartTime+l_ExtroTime))
            SetIngameGameState();
        if(m_FadeIn.IsExecuting())
        {
            m_FadeIn.OnGUI(m_FadeTime);
            if(m_FadeInVolume)
                audio.volume=m_FadeIn.GetPercentage(m_FadeTime);
        }
        else
            audio.volume=1.0f;
    }
    void ShowShowEndRoundScreen()
    {
        float l_IntroTime=1.5f;
        float l_ExtroStartTime=4.5f;
        float l_ExtroTime=1.5f;
        float l_IntroOffset=1.0f;
        float l_ExtroOffset=1.0f;

        m_CurrentTime+=Time.deltaTime;
        float l_IntroOffsetStart=l_IntroOffset*Mathf.Cos((1.0f-Mathf.Max(0.0f,
(l_IntroTime-m_CurrentTime)/l_IntroTime))*Mathf.PI*0.5f);
        float l_ExtroOffsetStart=-l_ExtroOffset*Mathf.Cos((1.0f-
Mathf.Clamp((m_CurrentTime-l_ExtroStartTime)/l_ExtroTime, 0.0f,
1.0f))*Mathf.PI*0.5f);
```

```
        if(m_YouWin)

GUI.DrawTexture(GUIController.GetRectangleGUICentered(0.5f+l_IntroOffsetStart+l_E
xtroOffsetStart, 0.3f, 0.3890625f, 0.13888888888888888888888888888889f),
m_YouWinTexture);
        else

GUI.DrawTexture(GUIController.GetRectangleGUICentered(0.5f+l_IntroOffsetStart+l_E
xtroOffsetStart, 0.3f, 0.4015625f, 0.13888888888888888888888888888889f),
m_YouLoseTexture);

        if(m_CurrentTime>=(l_ExtroStartTime+l_ExtroTime))
        {

if(CFighterGame.GetFighterGame().GetGameType()==CFighterGame.TGameType.EXHIBITION
|| !m_YouWin)
            {
                if(!m_FadeOut.IsExecuting())
                    m_FadeOut.Execute();
                else
                {
                    bool l_Done=m_FadeOut.OnGUI(m_FadeTime);
                    audio.volume=1.0f-m_FadeOut.GetPercentage(m_FadeTime);
                    if(l_Done)
                       Application.LoadLevel("MainMenuScene");
                }
            }
            else
            {
                if(m_YouWin)
                {
                    CMatch<CFighterPlayer>
l_Match=CFighterGame.GetFighterGame().GetTournament().GetCurrentMatch();
                    CFighterPlayer l_Player1=l_Match.GetPlayer1();
                    CFighterPlayer l_Player2=l_Match.GetPlayer2();
                    l_Match.SetWinner(l_Player1.m_Human ? l_Player1 : l_Player2);
                    l_Match.GetDraw().NextMatch();
                    SetShowPlayoffsGameState();
                }
            }
        }
    }
    void SetIngameGameState()
    {
        m_Player1.SetActiveHumanControls(true);
        m_Player2.SetActiveHumanControls(true);
        m_GameState=TGameState.IN_GAME;
        m_Player1.SetIdleState();
        m_Player2.SetIdleState();
    }
    public void OnEndGame(bool YouWin)
    {
        if(YouWin)
            audio.PlayOneShot(m_YouWinSound);
        else
            audio.PlayOneShot(m_YouLoseSound);
        m_YouWin=YouWin;
        m_GameState=TGameState.END_ROUND;
        m_CurrentTime=0.0f;
        m_Player1.SetCinematicsState();
        m_Player2.SetCinematicsState();
        m_Player1.SetActiveHumanControls(false);
```

```
        m_Player2.SetActiveHumanControls(false);
    }
    IEnumerator PlaySoundDelayed(AudioClip Sound, float DelayedTime)
    {
        yield return new WaitForSeconds(DelayedTime);
        audio.PlayOneShot(m_FightSound);
    }
    void SetShowRoundGameState()
    {
        m_FadeIn.Execute();
        StartCoroutine(PlaySoundDelayed(m_FightSound, 1.0f));
        m_GameState=TGameState.SHOW_ROUND;
        m_CurrentTime=0.0f;
        m_Player1.SetCinematicsState();
        m_Player2.SetCinematicsState();
    }
    void InitPlayers()
    {
        if(m_Player1==null && m_Player2==null)
        {

m_Player1=(RobotPlayer)((GameObject)GameObject.Instantiate(m_RobotPrefab)).GetCom
ponent<RobotPlayer>();

m_Player2=(RobotPlayer)((GameObject)GameObject.Instantiate(m_RobotPrefab)).GetCom
ponent<RobotPlayer>();


            CFighterPlayer l_Player1=null;
            CFighterPlayer l_Player2=null;

            CFighterGame l_FighterGame=CFighterGame.GetFighterGame();
            int l_IdMaterialHuman=0;
            int l_IdMaterialOpponent=1;
            CMatch<CFighterPlayer> l_Match=null;
            if(l_FighterGame.GetGameType()==CFighterGame.TGameType.EXHIBITION)
            {
                l_Match=l_FighterGame.GetCurrentMatch();
                l_Player1=l_Match.GetPlayer1();
                l_Player2=l_Match.GetPlayer2();
                l_IdMaterialHuman=l_Player1.m_Id;
                l_IdMaterialOpponent=l_Player2.m_Id;
            }
            else
            {
                l_Match=l_FighterGame.GetTournament().GetCurrentMatch();
                l_Player1=l_Match.GetPlayer1();
                l_Player2=l_Match.GetPlayer2();
                if(!l_Player1.m_Human)
                {
                    CFighterPlayer l_Aux=l_Player2;
                    l_Player2=l_Player1;
                    l_Player1=l_Aux;
                }
                l_IdMaterialHuman=l_Player1.m_Id;
                l_IdMaterialOpponent=l_Player2.m_Id;
            }
            InitPlayer(m_Player1, true, m_Player2, m_Robot1Transform,
"RobotPlayer", m_Materials[l_IdMaterialHuman], l_Player1);
            InitPlayer(m_Player2, false, m_Player1, m_Robot2Transform, "RobotAI",
m_Materials[l_IdMaterialOpponent], l_Player2);
```

```
            m_Player1.SetActiveHumanControls(false);
            m_Player2.SetActiveHumanControls(false);
        }
    }
    void InitPlayer(RobotPlayer Player, bool Human, RobotPlayer PlayerOpponent,
Transform _Transform, string Name, Material _Material, CFighterPlayer
FighterPlayer)
    {
        Player.gameObject.name=Name;
        Player.transform.position=_Transform.position;
        Player.transform.rotation=_Transform.rotation;
        Player.m_Opponent=PlayerOpponent;
        Player.m_Human=Human;
        Player.m_GameScreen=this;

        Player.m_MinDecissionTime=FighterPlayer.m_MinDecissionTime;
        Player.m_RandomDecissionTime=FighterPlayer.m_RandomDecissionTime;
           Player.m_AggressivePct=FighterPlayer.m_AggressivePct;
           Player.m_DeffensivePct=FighterPlayer.m_DeffensivePct;
           Player.m_KickPct=FighterPlayer.m_KickPct;
           Player.m_PunchPct=FighterPlayer.m_PunchPct;
           Player.m_SimpleAttackPct=FighterPlayer.m_SimpleAttackPct;

        if(Human)
        {
            Player.m_Joystick=m_Joystick;
            Player.m_Kick=m_Kick;
            Player.m_Punch=m_Punch;
            Player.m_Deffense=m_Deffense;
        }
        foreach(SkinnedMeshRenderer l_SkinnedMeshRenderer in
Player.GetComponentsInChildren<SkinnedMeshRenderer>())

l_SkinnedMeshRenderer.material=(Material)Material.Instantiate(_Material);
    }
    void SetShowPlayoffsGameState()
    {
        GameObject.Destroy(m_Player1.gameObject);
        GameObject.Destroy(m_Player2.gameObject);

        m_Joystick.enabled=false;
        m_Kick.enabled=false;
        m_Punch.enabled=false;
        m_Deffense.enabled=false;

        m_GameState=TGameState.SHOW_PLAYOFFS;

    }
    void ShowPlayoffsScreen()
    {
        bool l_ScreenActive=!m_FadeOut.IsExecuting();
        if(GUI.Button(GUIController.GetRectangleGUI(0.01f, 0.02f, 0.0578125f,
0.10277777777777777777777777777778f), "", "ExitButton"))
        {
            if(l_ScreenActive)
            {
                m_NextGameState=TGameState.GOTO_MAIN_MENU;
                m_FadeOut.Execute();
                audio.PlayOneShot(m_ClickSound);
            }
        }
```

```
        if(GUI.Button(GUIController.GetRectangleGUI(0.915f, 0.815f, 0.0578125f,
0.10277777777777777777777777777778f), "", "NextButton"))
        {
            CTournament<CFighterPlayer>
l_Tournament=CFighterGame.GetFighterGame().GetTournament();
            audio.PlayOneShot(m_ClickSound);

            if(l_Tournament.GetWinner()!=null)
            {
                m_NextGameState=TGameState.GOTO_MAIN_MENU;
                m_FadeOut.Execute();
                //Application.LoadLevel("MainMenuScene");
            }
            else
            {
                //Debug.Log("Has winner
"+(l_Tournament.GetCurrentMatch().GetWinner()==null ? "NO" : "YES"));
                if(l_Tournament.GetCurrentMatch().GetPlayer1().m_Human ||
l_Tournament.GetCurrentMatch().GetPlayer2().m_Human)
                {
                    //NewMatchAfterPlayoffsScreen();
                    m_NextGameState=TGameState.GOTO_GAME_FROM_PLAYOFFS;
                    m_FadeOut.Execute();
                }
                else
                    l_Tournament.Simulate();
            }
        }

        CFighterGame.ShowGUITournament(m_PlayoffBackgroundTexture);

        if(m_FadeOut.IsExecuting())
        {
            bool l_Done=m_FadeOut.OnGUI(m_FadeTime);
            if(m_NextGameState==TGameState.GOTO_MAIN_MENU)
            {
                audio.volume=1.0f-m_FadeOut.GetPercentage(m_FadeTime);
                if(l_Done)
                    Application.LoadLevel("MainMenuScene");
            }
            else if(m_NextGameState==TGameState.GOTO_GAME_FROM_PLAYOFFS)
            {
                if(l_Done)
                    NewMatchAfterPlayoffsScreen();
            }
        }
    }
    void NewMatchAfterPlayoffsScreen()
    {
        m_Joystick.enabled=true;
        m_Kick.enabled=true;
        m_Punch.enabled=true;
        m_Deffense.enabled=true;
        m_Player1=null;
        m_Player2=null;
        m_FadeInVolume=false;
        InitPlayers();
        SetShowRoundGameState();
    }
}
```

MainMenuScreen.cs

```
using UnityEngine;
using System.Collections;
using System;
using UOCUtils;
using System.Collections.Generic;
using UOCTournament;
using UOCFightersGame;

public class MainMenuScreen : MonoBehaviour
{
    public enum TMainMenuScreenState
    {
        PRODUCER,
        DEVELOPER,
        MAIN_MENU,
        SELECT_PLAYER,
        SHOW_PLAYOFF
    }
    public GUISkin m_Skin;
    public TMainMenuScreenState m_MainMenuScreenState;
    public List<Texture2D> m_CharactersTextures;
    public List<Material> m_CharactersMaterials;
    public GameObject m_Logo;
    public Texture2D m_MainMenuBackgroundTexture;
    public Texture2D m_SelectPlayerBackgroundTexture;
    public GameObject m_Player1;
    public GameObject m_Player2;
    int m_SelectedPlayer=-1;
    int m_SelectedOpponent=-1;
    float m_CurrentTime=0.0f;
    const float m_SelectionPlayerTime=3.0f;
    public Texture2D m_PlayoffBackgroundTexture;
    public AudioClip m_ClickSound;
    public Texture2D m_UOCTexture;
    public Texture2D m_UOCDeveloper;
    CFadeIn m_FadeIn;
    CFadeOut m_FadeOut;
    public float m_FadeTime=0.8f;
    public float m_DeveloperProducerScreenTime=5.0f;
    public delegate void DeveloperProducerNextState();
    static public bool m_ShowDeveloperScreen=true;

       void Start()
    {
        audio.volume=0.0f;
        m_FadeIn=new CFadeIn();
        m_FadeOut= new CFadeOut();
#if UNITY_EDITOR
        //SetProducerState();
        SetMainMenuState();
        //SetTournamentState();
        //InitTournament();
#else
        if(m_ShowDeveloperScreen)
            SetProducerState();
        else
            SetMainMenuStateWithFadeIn();
#endif
        m_ShowDeveloperScreen=false;
#if UNITY_EDITOR
        CheckHotValues();
```

```
#endif
    }
    void SetProducerState()
    {
        m_CurrentTime=0.0f;
        m_FadeIn.Execute();
        m_MainMenuScreenState=TMainMenuScreenState.PRODUCER;
    }
    void SetDeveloperState()
    {
        m_CurrentTime=0.0f;
        m_FadeIn.Execute();
        m_MainMenuScreenState=TMainMenuScreenState.DEVELOPER;
    }
       void Update()
    {
#if UNITY_EDITOR
        CheckHotValues();
#endif
      }
    void InitTournament()
    {
#if UNITY_EDITOR
        if(m_SelectedPlayer==-1)
            m_SelectedPlayer=0;
#endif
        CFighterGame.GetFighterGame().InitTournament(m_SelectedPlayer);
        CFighterGame.CheckDrawPositions();
    }
#if UNITY_EDITOR
    void CheckHotValues()
    {
        if(m_FadeIn==null)
            m_FadeIn=new CFadeIn();
        if(m_FadeOut==null)
            m_FadeOut= new CFadeOut();

        CFighterGame.CheckDrawPositions();
        CFighterGame.GetFighterGame().CheckTournament();
    }
#endif
    void OnGUI()
    {
#if UNITY_EDITOR
        CheckHotValues();
#endif
        if(Application.isLoadingLevel)
        {
            GUI.DrawTexture(GUIController.GetRectangleGUI(0.0f, 0.0f, 1.0f,
1.0f), CFadeOut.m_BlackTexture);
            return;
        }
        GUI.skin=m_Skin;
        switch(m_MainMenuScreenState)
        {
            case TMainMenuScreenState.PRODUCER:
                ShowProducerDeveloperScreen(m_UOCTexture, new
DeveloperProducerNextState(SetDeveloperState));
                break;
            case TMainMenuScreenState.DEVELOPER:
                ShowProducerDeveloperScreen(m_UOCDeveloper, new
DeveloperProducerNextState(SetMainMenuStateWithFadeIn));
```

```
                break;
            case TMainMenuScreenState.MAIN_MENU:
                OnGUIOnMainMenuScreen();
                break;
            case TMainMenuScreenState.SELECT_PLAYER:
                OnGUIOnSelectPlayerScreen();
                break;
            case TMainMenuScreenState.SHOW_PLAYOFF:
                OnGUIOnShowPlayoffScreen();
                break;
        }
    }
    void OnGUIOnMainMenuScreen()
    {
        m_Logo.transform.Rotate(new Vector3(0.0f, 15.0f*Time.deltaTime, 0.0f),
Space.World);

        GUI.DrawTexture(GUIController.GetRectangleGUI(0.11875f,
0.818055555555555555555555555555556f, 0.88125f,
0.181944444444444444444444444444444f), m_MainMenuBackgroundTexture);

        if(GUI.Button(GUIController.GetRectangleGUI(0.01f, 0.02f, 0.0578125f,
0.102777777777777777777777777777778f), "", "ExitButton"))
        {
            audio.PlayOneShot(m_ClickSound);
            Application.Quit();
        }

        if(GUI.Button(GUIController.GetRectangleGUI(0.6f, 0.89f, 0.18125f,
0.095833333333333333333333333333333f), "", "ExhibitionButton"))
        {
            audio.PlayOneShot(m_ClickSound);
            SetExhibitionState();
        }
        if(GUI.Button(GUIController.GetRectangleGUI(0.8f, 0.89f, 0.18125f,
0.095833333333333333333333333333333f), "", "TournamentButton"))
        {
            audio.PlayOneShot(m_ClickSound);
            SetTournamentState();
        }
        if(m_FadeIn.IsExecuting())
        {
            m_FadeIn.OnGUI(m_FadeTime);
            float l_Pct=m_FadeIn.GetPercentage(m_FadeTime);
            audio.volume=l_Pct;
        }
        else
            audio.volume=1.0f;
    }
    void SetMainMenuStateWithFadeIn()
    {
        SetMainMenuState();
        m_FadeIn.Execute();
    }
    void SetMainMenuState()
    {
        m_MainMenuScreenState=TMainMenuScreenState.MAIN_MENU;
        m_Logo.SetActive(true);
        SetPlayersEnabled(false);
       }
    void SetExhibitionState()
    {
```

```
        InitSelectPlayerScreen(CFighterGame.TGameType.EXHIBITION);
    }
    void SetTournamentState()
    {
        InitSelectPlayerScreen(CFighterGame.TGameType.TOURNAMENT);
    }
    void InitSelectPlayerScreen(CFighterGame.TGameType GameType)
    {
        m_MainMenuScreenState=TMainMenuScreenState.SELECT_PLAYER;
        CFighterGame.GetFighterGame().SetGameType(GameType);
        m_Logo.SetActive(false);
        SetPlayersEnabled(false);
        m_SelectedPlayer=-1;
        m_SelectedOpponent=-1;
    }
    void OnGUIOnSelectPlayerScreen()
    {
        m_CurrentTime+=Time.deltaTime;
        float l_OffsetX=0.001f;
        Vector2 l_Size=new Vector2(0.07734375f, 0.1375f)*0.92f;
        float l_Start=0.41f;

        GUI.DrawTexture(GUIController.GetRectangleGUI(0.11875f,
0.818055555555555555555555555555556f, 0.88125f,
0.181944444444444444444444444444444f), m_SelectPlayerBackgroundTexture);

        if(m_SelectedPlayer==-1)
        {
            if(GUI.Button(GUIController.GetRectangleGUI(0.01f, 0.02f, 0.0578125f,
0.102777777777777777777777777777778f), "", "ExitButton"))
                SetMainMenuState();
        }

        for(int i=0;i<m_CharactersTextures.Count;++i)
        {
            float l_OffsetTexture=0.1f;
            Rect
l_GUIRect=GUIController.GetRectangleGUI(l_Start+(l_OffsetX+l_Size.x)*i, 0.86f,
l_Size.x, l_Size.y);
            Rect
l_GUITextureRect=GUIController.GetRectangleGUI(l_Start+(l_OffsetX+l_Size.x)*i+l_O
ffsetTexture*l_Size.x, 0.86f+l_OffsetTexture*l_Size.y, l_Size.x-
l_OffsetTexture*l_Size.x*2.0f, l_Size.y-l_OffsetTexture*l_Size.y*2.0f);
            if(m_SelectedPlayer==-1)
            {
                m_CurrentTime=0.0f;
                GUI.DrawTexture(l_GUITextureRect, m_CharactersTextures[i]);
                if(GUI.Button(l_GUIRect, "", "SelectPlayerButton"))
                {
                    audio.PlayOneShot(m_ClickSound);
                    m_SelectedPlayer=i;
                    SelectPlayer(m_Player1, i);
                }
            }
            else
            {
                GUI.DrawTexture(l_GUITextureRect, m_CharactersTextures[i]);
                GUI.DrawTexture(l_GUIRect,  (i==m_SelectedPlayer ||
i==m_SelectedOpponent) ? m_Skin.GetStyle("SelectPlayerButton").active.background
: m_Skin.GetStyle("SelectPlayerButton").normal.background);
            }
        }
```

```
        if(m_SelectedPlayer!=-1)
        {
            if(m_CurrentTime>=m_SelectionPlayerTime)
            {


if(CFighterGame.GetFighterGame().GetGameType()==CFighterGame.TGameType.EXHIBITION
)
                        {
                            if(m_SelectedOpponent==-1)
                            {
                                m_SelectedOpponent=UnityEngine.Random.Range(0,
m_CharactersTextures.Count-1);
                                if(m_SelectedOpponent==m_SelectedPlayer)

m_SelectedOpponent=(m_SelectedOpponent+1)%m_CharactersMaterials.Count;
                                SelectPlayer(m_Player2, m_SelectedOpponent);
                            }
                            if(m_CurrentTime>=(2.0f*m_SelectionPlayerTime))
                            {
                                if(m_FadeOut.IsExecuting())
                                {
                                    bool l_Done=m_FadeOut.OnGUI(m_FadeTime);
                                    audio.volume=1.0f-
m_FadeOut.GetPercentage(m_FadeTime);
                                    if(l_Done)
                                        RunGame();
                                }
                                else
                                    m_FadeOut.Execute();
                            }
                        }
                        else
                        {
                            InitTournament();
                            SetShowDrawState();
                        }
            }
        }
    }
    void SetPlayersEnabled(bool Enabled)
    {
        m_Player1.SetActive(Enabled);
        m_Player2.SetActive(Enabled);
    }
    void SetMaterialToPlayer(GameObject Player, Material _Material)
    {
        foreach(SkinnedMeshRenderer l_SkinnedMeshRenderer in
Player.GetComponentsInChildren<SkinnedMeshRenderer>())
            l_SkinnedMeshRenderer.material=_Material;
    }
    void SelectPlayer(GameObject Player, int IdPlayer)
    {
        Player.SetActive(true);
        SetMaterialToPlayer(Player, m_CharactersMaterials[IdPlayer]);
    }
    void RunGame()
    {
        CFighterGame l_FighterGame=CFighterGame.GetFighterGame();
```

```
        if(l_FighterGame.GetGameType()==CFighterGame.TGameType.EXHIBITION)
            l_FighterGame.InitExhibitionGame(m_SelectedPlayer,
m_SelectedOpponent);
        else
            l_FighterGame.InitTournamentGame();
        Application.LoadLevel("GameScene");
    }
    void SetShowDrawState()
    {
        m_MainMenuScreenState=TMainMenuScreenState.SHOW_PLAYOFF;
        SetPlayersEnabled(false);
    }
    void OnGUIOnShowPlayoffScreen()
    {
        bool l_ScreenActive=!m_FadeOut.IsExecuting();
        CFighterGame.ShowGUITournament(m_PlayoffBackgroundTexture);
        if(GUI.Button(GUIController.GetRectangleGUI(0.01f, 0.02f, 0.0578125f,
0.10277777777777777777777777777778f), "", "ExitButton"))
        {
            if(l_ScreenActive)
            {
                audio.PlayOneShot(m_ClickSound);
                SetMainMenuState();
            }
        }

        if(GUI.Button(GUIController.GetRectangleGUI(0.915f, 0.815f, 0.0578125f,
0.10277777777777777777777777777778f), "", "NextButton"))
        {
            if(l_ScreenActive)
            {
                CTournament<CFighterPlayer>
l_Tournament=CFighterGame.GetFighterGame().GetTournament();
                audio.PlayOneShot(m_ClickSound);

                if(l_Tournament.GetCurrentMatch().GetPlayer1().m_Human ||
l_Tournament.GetCurrentMatch().GetPlayer2().m_Human)
                    m_FadeOut.Execute();
                else
                {
                    l_Tournament.Simulate();
                    if(l_Tournament.GetWinner()!=null)
                        SetMainMenuState();
                }
            }
        }
        if(m_FadeOut.IsExecuting())
        {
            bool l_Done=m_FadeOut.OnGUI(m_FadeTime);
            audio.volume=1.0f-m_FadeOut.GetPercentage(m_FadeTime);
            if(l_Done)
                RunGame();
        }
    }
    void ShowProducerDeveloperScreen(Texture2D _Texture2D, Delegate FnNextState)
    {
        GUI.DrawTexture(GUIController.GetRectangleGUI(0.0f, 0.0f, 1.0f, 1.0f),
_Texture2D);

        if(m_FadeIn.IsExecuting())
            m_FadeIn.OnGUI(m_FadeTime);
        else
```

```
        {
            if(m_FadeOut.IsExecuting())
            {
                bool l_Done=m_FadeOut.OnGUI(m_FadeTime);
                if(l_Done)
                    FnNextState.DynamicInvoke();
            }
            else
            {
                m_CurrentTime+=Time.deltaTime;
                if(m_CurrentTime>=m_DeveloperProducerScreenTime)
                    m_FadeOut.Execute();
            }
        }
    }
}
```

RobotPlayer.cs

```
//#define UOC_NOT_UPDATE_AI_DECISSION
//#define UOC_FAST_KO

using UnityEngine;
using System.Collections;
using System.Collections.Generic;

public class RobotPlayer : MonoBehaviour
{
    public enum TState
    {
        IDLE,
        PUNCH,
        KICK,
        HIT,
        CINEMATICS,
        DEFFENSE,
        DIE,
        KEEP_RANGED,
        APPROACH_AND_ATTACK_SIMPLE,
        APPROACH_AND_COMBO_ATTACK,
        ATTACK_SIMPLE,
        ATTACK_COMBO
    }
    public enum TAttackType
    {
        SOFT_KICK,
        MEDIUM_KICK,
        HARD_KICK,
        SOFT_PUNCH,
        MEDIUM_PUNCH,
        HARD_PUNCH,
        RANGED
    }
    public bool m_Human;
    public UOCVirtualJoystick m_Joystick;
    public UOCVirtualButton m_Kick;
    public UOCVirtualButton m_Punch;
    public UOCVirtualButton m_Deffense;
    public float m_Speed=5.0f;
    public float m_HitSpeed=0.75f;
    public float m_HittingSpeed=0.3f;
```

```
private CharacterController m_CC;
public float m_VerticalSpeed;
public TState m_State=TState.IDLE;
public GameObject m_Explosion;
public GameObject m_LeftFist;
public GameObject m_RightFist;
public GameObject m_RightFoot;
private string m_CurrentAnimationName;
public GameObject m_CurrentPhysicsHit;
private float m_Life;
private float m_CurrentLife;
private float m_StartCurrentLife=1.0f;
private float m_CurrentLifeTime=0.0f;
public float m_CurrentHitLife;
public float m_StrongPunchLife=-0.15f;
public float m_MediumPunchLife=-0.1f;
public float m_WeakPunchLife=-0.05f;
public float m_StrongKickLife=-0.21f;
public float m_MediumKickLife=-0.14f;
public float m_WeakKickLife=-0.07f;
public float m_DeffenseDistance=0.5f;
public float m_AttackDistance=0.4f;
public RobotPlayer m_Opponent;
private bool m_CanDeffense=true;
private const float m_ChangeLifeTime=0.4f;

public GameScreen m_GameScreen;

public TrailRenderer m_TrailRenderer;

//AI parameters
public float m_MinDecissionTime=0.4f;
public float m_RandomDecissionTime=0.5f;
public float m_CurrentDecissionTime=1.0f;
public float m_AggressivePct=0.75f;
public float m_DeffensivePct=0.6f;
public float m_KickPct=0.8f;
public float m_PunchPct=0.2f;
public float m_DistanceKeepRangedMoved=0.8f;
public float m_CurrentDistanceKeepRangedMoved=0.0f;
public float m_SimpleAttackPct=0.7f;
public int m_CurrentComboAttackId=0;
public List<TAttackType> m_CurrentComboAttack=null;

List<List<TAttackType>> m_KickCombos;
List<List<TAttackType>> m_PunchCombos;
Vector3 m_StartPosition;
Quaternion m_StartRotation;

public List<AudioClip> m_HitSounds;
public List<AudioClip> m_MissSounds;
public GameObject  m_HitParticles;
public GameObject  m_MissParticles;

void Start()
{
    DisableHitters();

    m_CurrentHitLife=0.0f;
    m_Life=1.0f;
    m_CurrentLife=1.0f;
    m_StartCurrentLife=1.0f;
```

```
        m_CurrentLifeTime=0.0f;

        m_CC=GetComponent<CharacterController>();
        m_PunchCombos=new List<List<TAttackType>>();
        m_KickCombos=new List<List<TAttackType>>();
        Add3Combo(ref m_PunchCombos, TAttackType.SOFT_PUNCH,
TAttackType.SOFT_PUNCH, TAttackType.HARD_KICK);
        Add4Combo(ref m_PunchCombos, TAttackType.SOFT_PUNCH,
TAttackType.SOFT_PUNCH, TAttackType.SOFT_KICK, TAttackType.HARD_KICK);
        Add2Combo(ref m_PunchCombos, TAttackType.SOFT_PUNCH,
TAttackType.HARD_KICK);
        Add2Combo(ref m_PunchCombos, TAttackType.MEDIUM_PUNCH,
TAttackType.HARD_KICK);

        Add3Combo(ref m_KickCombos, TAttackType.SOFT_KICK, TAttackType.SOFT_KICK,
TAttackType.HARD_PUNCH);
        Add2Combo(ref m_KickCombos, TAttackType.SOFT_KICK,
TAttackType.HARD_KICK);

        m_StartPosition=transform.position;
        m_StartRotation=transform.rotation;
       }
    void DisableHitters()
    {
        m_LeftFist.SetActive(false);
        m_RightFist.SetActive(false);
        m_RightFoot.SetActive(false);
    }
    void Add2Combo(ref List<List<TAttackType>> Combos, TAttackType Attack1,
TAttackType Attack2)
    {
        List<TAttackType> l_AttackCombo=new List<TAttackType>();
        l_AttackCombo.Add(Attack1);
        l_AttackCombo.Add(Attack2);
        Combos.Add(l_AttackCombo);
    }
    void Add3Combo(ref List<List<TAttackType>> Combos, TAttackType Attack1,
TAttackType Attack2, TAttackType Attack3)
    {
        List<TAttackType> l_AttackCombo=new List<TAttackType>();
        l_AttackCombo.Add(Attack1);
        l_AttackCombo.Add(Attack2);
        l_AttackCombo.Add(Attack3);
        Combos.Add(l_AttackCombo);
    }
    void Add4Combo(ref List<List<TAttackType>> Combos, TAttackType Attack1,
TAttackType Attack2, TAttackType Attack3, TAttackType Attack4)
    {
        List<TAttackType> l_AttackCombo=new List<TAttackType>();
        l_AttackCombo.Add(Attack1);
        l_AttackCombo.Add(Attack2);
        l_AttackCombo.Add(Attack3);
        l_AttackCombo.Add(Attack4);
        Combos.Add(l_AttackCombo);
    }
    void SetDeffenseState()
    {
        m_CanDeffense=false;
        animation.CrossFade("def_head");
        animation["def_head"].speed=1.8f;
        m_State=TState.DEFFENSE;
    }
```

```
    void FixedUpdate()
    {
        bool l_LateralMovement=false;
        Vector3 l_Movement=Vector3.zero;
        switch(m_State)
        {
            case TState.CINEMATICS:
                break;
            case TState.IDLE:
                if(m_Human)
                    l_Movement=UpdateHumanMovement(ref l_LateralMovement,
m_Speed);
                if(l_LateralMovement)
                {
                    if(animation.name!="loop_walk_funny")
                        animation.CrossFade("loop_walk_funny");
                }
                else
                {
                    if(animation.name!="loop_idle")
                        animation.CrossFade("loop_idle");
                }
                if(m_Human)
                {
                    if(m_Punch.m_Pressed)
                    {
                        if(m_Joystick.m_Active && m_Joystick.m_Direction.y<-0.5f)
                        {
                            SetHardPunch();
                        }
                        else if(m_Joystick.m_Active &&
m_Joystick.m_Direction.y>0.5f)
                        {
                            SetSoftPunch();
                        }
                        else
                            SetMediumPunch();

                        m_State=TState.PUNCH;
                    }
                    if(m_Kick.m_Pressed)
                    {
                        if(m_Joystick.m_Active && m_Joystick.m_Direction.y<-0.5f)
                        {
                            SetHardKick();
                        }
                        else if(m_Joystick.m_Active &&
m_Joystick.m_Direction.y>0.5f)
                        {
                            SetSoftKick();
                        }
                        else
                            SetMediumKick();

                        m_State=TState.KICK;
                    }
                    if(m_Deffense.m_Pressed)
                    {
                        SetDeffenseState();
                    }
                }
                else
```

```
                {
                    m_CurrentDecissionTime-=Time.deltaTime;
                    if(MustAIDecide())
                        AIDecideState();
                    if(MustDeffense())
                        SetDeffenseState();
                }
                break;
            case TState.DEFFENSE:
                if(!animation.isPlaying)
                {
                    m_CanDeffense=true;
                    SetIdleState();
                }
                break;
            case TState.HIT:
                l_Movement=transform.right;
                l_Movement.Normalize();
                l_Movement*=m_HittingSpeed;
                if(!animation.isPlaying)
                {
                    SetIdleState();
                }
                break;
            case TState.DIE:
                if(!animation.isPlaying)
                    OnEndGame();
                break;
            case TState.KICK:
            case TState.ATTACK_COMBO:
            case TState.ATTACK_SIMPLE:
            case TState.PUNCH:
                if(m_Human)
                    l_Movement=UpdateHumanMovement(ref l_LateralMovement,
m_HittingSpeed);
                AnimationState
l_AnimationState=animation[m_CurrentAnimationName];
                if(l_AnimationState!=null)
                    m_CurrentPhysicsHit.SetActive(IsHitEnabled());
                if(!animation.isPlaying)
                {
                    if(m_State==TState.ATTACK_COMBO)
                        SetNextComboAttack();
                    else
                        SetIdleState();
                    m_CurrentPhysicsHit=null;
                }
                break;
            case TState.KEEP_RANGED:
                l_Movement=transform.right*m_Speed;
                m_CurrentDistanceKeepRangedMoved-
=(l_Movement*Time.deltaTime).magnitude;
                if(m_CurrentDistanceKeepRangedMoved<0.0f)
                {
                    InitDecissionTime();
                    SetIdleState();
                }
                break;
            case TState.APPROACH_AND_ATTACK_SIMPLE:
                if((transform.position-
m_Opponent.transform.position).magnitude<m_AttackDistance)
                    SetAttackSimple();
```

```
                else
                    l_Movement=-transform.right*m_Speed;

                if(MustDeffense())
                    SetDeffenseState();
                break;
            case TState.APPROACH_AND_COMBO_ATTACK:
                if((transform.position-
m_Opponent.transform.position).magnitude<m_AttackDistance)
                    SetComboAttack();
                else
                    l_Movement=-transform.right*m_Speed;

                if(MustDeffense())
                    SetDeffenseState();
                break;
        }
        UpdateCurrentLife();
        UpdateVerticalMovement(ref l_Movement);

        CollisionFlags l_CollisionFlags=m_CC.Move(l_Movement*Time.deltaTime);
        if((l_CollisionFlags & CollisionFlags.Below)!=0)
            m_VerticalSpeed=0.0f;

        if(m_CurrentPhysicsHit!=null)

m_TrailRenderer.transform.position=m_CurrentPhysicsHit.transform.position;
        m_TrailRenderer.enabled=m_CurrentPhysicsHit!=null;
    }
    void UpdateCurrentLife()
    {
        m_CurrentLifeTime-=Time.deltaTime;
        m_CurrentLife=Mathf.Max(0.0f, m_StartCurrentLife+(Mathf.Min(1.0f,
(m_ChangeLifeTime-m_CurrentLifeTime)/m_ChangeLifeTime))*(m_Life-
m_StartCurrentLife));
    }
    bool MustAIDecide()
    {
#if UOC_NOT_UPDATE_AI_DECISSION
        return false;
#endif
        return m_CurrentDecissionTime<=0.0f || (m_Opponent.transform.position-
transform.position).magnitude<m_DeffenseDistance;
    }
    void SetHardPunch()
    {
        m_CurrentAnimationName="punch_hi_right";
        m_CurrentPhysicsHit=m_RightFist;
        m_CurrentHitLife=m_StrongPunchLife;
        animation.CrossFade(m_CurrentAnimationName);
        animation[m_CurrentAnimationName].speed=1.0f;
    }
    void SetHardKick()
    {
        m_CurrentHitLife=m_StrongKickLife;
        m_CurrentPhysicsHit=m_RightFoot;
        m_CurrentAnimationName="kick_jump_right";
        animation.CrossFade(m_CurrentAnimationName);
        animation[m_CurrentAnimationName].speed=1.0f;
    }
    void SetMediumKick()
    {
```

```
        m_CurrentHitLife=m_MediumKickLife;
        m_CurrentPhysicsHit=m_RightFoot;
        m_CurrentAnimationName="kick_lo_right";
        animation.CrossFade(m_CurrentAnimationName);
        animation[m_CurrentAnimationName].speed=1.0f;
    }
    void SetSoftKick()
    {
        m_CurrentHitLife=m_WeakKickLife;
        m_CurrentPhysicsHit=m_RightFoot;
        m_CurrentAnimationName="kick_lo_right";
        animation.CrossFade(m_CurrentAnimationName);
        animation[m_CurrentAnimationName].speed=2.0f;
    }
    void SetSoftPunch()
    {
        m_CurrentAnimationName="punch_hi_left";
        m_CurrentPhysicsHit=m_LeftFist;
        m_CurrentHitLife=m_WeakPunchLife;
        animation.CrossFade(m_CurrentAnimationName);
        animation[m_CurrentAnimationName].speed=2.0f;
    }
    void SetMediumPunch()
    {
        m_CurrentHitLife=m_StrongPunchLife;
        m_CurrentAnimationName="punch_hi_left";
        m_CurrentPhysicsHit=m_LeftFist;
        m_CurrentHitLife=m_MediumPunchLife;
        animation.CrossFade(m_CurrentAnimationName);
        animation[m_CurrentAnimationName].speed=1.0f;
    }
    void SetAttackSimple()
    {
        float l_Pct=Random.value*(m_KickPct+m_PunchPct);
        float l_Pct2=Random.value;
        if(l_Pct<m_KickPct)
        {
            if(l_Pct2<0.33f)
                SetHardKick();
            else if(l_Pct2<0.66f)
                SetMediumKick();
            else
                SetSoftKick();
        }
        else
        {
            if(l_Pct2<0.33f)
                SetHardPunch();
            else if(l_Pct2<0.66f)
                SetMediumPunch();
            else
                SetSoftPunch();
        }
        InitDecissionTime();
        m_State=TState.ATTACK_SIMPLE;
    }
    void SetKeepRangedStage()
    {
        m_State=TState.KEEP_RANGED;
        m_CurrentDistanceKeepRangedMoved=m_DistanceKeepRangedMoved;
        animation.CrossFade("loop_walk_funny");
    }
```

```
    void InitDecissionTime()
    {

m_CurrentDecissionTime=m_MinDecissionTime+Random.value*m_RandomDecissionTime;
    }
    void AIDecideState()
    {
        float l_Pct=Random.value;

        if(l_Pct<=m_AggressivePct)
        {
            l_Pct=Random.value;
            if(l_Pct<m_SimpleAttackPct)
                SetApproachAndSimpleAttack();
            else
                SetApproachAndComboAttack(true);
        }
        else
        {
            if((m_Opponent.transform.position-
transform.position).magnitude<0.75*m_DistanceKeepRangedMoved)
                SetKeepRangedStage();
            else
                InitDecissionTime();
        }
    }
    void SetApproachAndSimpleAttack()
    {
        m_State=TState.APPROACH_AND_ATTACK_SIMPLE;
        animation.CrossFade("loop_walk_funny");
    }
    void SetComboAttack()
    {
        switch(m_CurrentComboAttack[m_CurrentComboAttackId])
        {
            case TAttackType.SOFT_KICK:
                SetSoftKick();
                break;
            case TAttackType.SOFT_PUNCH:
                SetSoftPunch();
                break;
            case TAttackType.MEDIUM_KICK:
                SetMediumKick();
                break;
            case TAttackType.MEDIUM_PUNCH:
                SetMediumPunch();
                break;
            case TAttackType.HARD_KICK:
                SetHardKick();
                break;
            case TAttackType.HARD_PUNCH:
                SetHardPunch();
                break;
        }
        m_State=TState.ATTACK_COMBO;
    }
    void SetNextComboAttack()
    {
        ++m_CurrentComboAttackId;
        if(m_CurrentComboAttackId<m_CurrentComboAttack.Count)
            SetApproachAndComboAttack(false);
        else
```

```
            SetKeepRangedStage();
    }
    void SetApproachAndComboAttack(bool Init)
    {
        m_State=TState.APPROACH_AND_COMBO_ATTACK;
        animation.CrossFade("loop_walk_funny");
        if(Init)
        {
            float l_Pct=Random.value*(m_KickPct+m_PunchPct);
            if(l_Pct<m_KickPct)
            {
                int
l_CurrentComboAttackId=(int)Mathf.Min((float)m_KickCombos.Count-1,
m_KickCombos.Count*Random.value);
                m_CurrentComboAttack=m_KickCombos[l_CurrentComboAttackId];
            }
            else
            {
                int
l_CurrentComboAttackId=(int)Mathf.Min((float)m_PunchCombos.Count-1,
m_PunchCombos.Count*Random.value);
                m_CurrentComboAttack=m_PunchCombos[l_CurrentComboAttackId];
            }
            m_CurrentComboAttackId=0;
        }
    }
    bool IsHitEnabled()
    {
        AnimationState l_AnimationState=animation[m_CurrentAnimationName];
        if(l_AnimationState!=null)
            return l_AnimationState.normalizedTime>0.3f &&
l_AnimationState.normalizedTime<0.7f;
        return false;
    }
    bool IsTryingToHit()
    {
        return (m_State==TState.PUNCH || m_State==TState.KICK) && IsHitEnabled();
    }
    bool MustDeffense()
    {
        bool l_HitsPlayer=m_Opponent.IsTryingToHit() &&
(m_Opponent.transform.position-transform.position).magnitude<m_DeffenseDistance;
        if(l_HitsPlayer && m_CanDeffense)
        {
            m_CanDeffense=false;
            float l_Pct=Random.value;
            return l_Pct<=m_DeffensivePct;
        }
        return false;
    }
    public void SetCinematicsState()
    {
        m_State=TState.CINEMATICS;
    }
    public void SetIdleState()
    {
        animation.CrossFade("loop_idle");
        animation["loop_idle"].speed=1.0f;
        m_State=TState.IDLE;
    }
    void CreateExplosion(Vector3 Position)
    {
```

```
        GameObject l_GO=(GameObject)GameObject.Instantiate(m_Explosion, Position,
Quaternion.Euler(0.0f, 0.0f, 0.0f));
        GameObject.Destroy(l_GO,  5.0f);
    }
    Vector3 UpdateHumanMovement(ref bool LateralMovement, float Speed)
    {
        Vector3 l_Movement=Vector3.zero;
        if(m_Joystick.m_Active)
        {
            Vector2 l_Direction=m_Joystick.m_Direction;
            l_Direction.Normalize();

            l_Movement=transform.right*-l_Direction.x;
            LateralMovement=l_Movement.magnitude>0.15f;
            l_Movement.Normalize();
            l_Movement*=Speed*m_Joystick.m_Direction.magnitude;
        }
        return l_Movement;
    }
    void UpdateVerticalMovement(ref Vector3 Movement)
    {
        m_VerticalSpeed+=Physics.gravity.y*Time.deltaTime;
        Movement.y=m_VerticalSpeed;
    }
    void Hit(float HitLife)
    {
        DisableHitters();

        m_CanDeffense=true;
        AddLife(HitLife);
#if UOC_FAST_KO
        Debug.Log("DEBUG FAST KO");
        if(m_Life>0 && m_Human)
#else
        if(m_Life>0)
#endif
        {
            m_State=TState.HIT;
            animation.CrossFade("xhit_body");
        }
        else
        {
            animation.CrossFade("final_head");
            m_State=TState.DIE;
        }
    }
    bool IsDeffending()
    {
        return m_State==TState.DEFFENSE;
    }
    void OnTriggerEnter(Collider other)
    {
        if(other.gameObject.tag=="HitCollider")
        {
            UOCHitInfo l_HitInfo=other.gameObject.GetComponent<UOCHitInfo>();
            GameObject l_Particles=null;
            if(l_HitInfo.m_Owner==m_Opponent)
            {
                if(m_State!=TState.HIT && !IsDeffending())
                {

l_Particles=(GameObject)GameObject.Instantiate(m_HitParticles);
```

```
                    PlayHitSound();
                    Hit(l_HitInfo.m_Owner.m_CurrentHitLife);
                }
                else
                {

l_Particles=(GameObject)GameObject.Instantiate(m_MissParticles);
                    PlayMissSound();
                }
                l_Particles.transform.position=other.transform.position;
                GameObject.Destroy(l_Particles, 3.5f);
            }
        }
    }
    void OnEndGame()
    {
        m_GameScreen.OnEndGame(!m_Human);
    }
    public void RestartGame()
    {
        m_Life=1.0f;
        m_CurrentLife=1.0f;
        m_StartCurrentLife=1.0f;
        m_CurrentLifeTime=0.0f;
        animation.Play("loop_idle");
        transform.position=m_StartPosition;
        transform.rotation=m_StartRotation;
        m_State=TState.IDLE;
        InitDecissionTime();
    }
    void PlayHitSound()
    {
              PlayRandomSoundFromList(m_HitSounds);
    }
    void PlayMissSound()
    {
              PlayRandomSoundFromList(m_MissSounds);
    }
    void PlayRandomSoundFromList(List<AudioClip> Sounds)
    {
              int l_IdSound=Random.Range(0, Sounds.Count-1);
              audio.PlayOneShot(Sounds[l_IdSound]);
       }
    void AddLife(float Life)
    {
        m_CurrentLifeTime=m_ChangeLifeTime;
        m_StartCurrentLife=m_CurrentLife;
        m_Life+=Life;
    }
    public float GetCurrentLife()
    {
        return m_CurrentLife;
    }
    public float GetHitLifeByAggressivity()
    {
        return m_CurrentHitLife*(1.0f+Mathf.Max(0.0f, m_AggressivePct-
0.5f)*0.5f);
    }
    public void SetActiveHumanControls(bool Active)
    {
        if(m_Human)
        {
```

```
            m_Kick.gameObject.SetActive(Active);
            m_Punch.gameObject.SetActive(Active);
            m_Deffense.gameObject.SetActive(Active);
            m_Joystick.gameObject.SetActive(Active);
        }
    }
}
```

UOCHitInfo.cs

```
using UnityEngine;
using System.Collections;

public class UOCHitInfo : MonoBehaviour
{
    public RobotPlayer m_Owner;
}
```

GUIController.cs

```
using UnityEngine;
using System.Collections;

namespace UOCUtils
{
       public class GUIController
    {
         public static Rect GetRectangleGUI(float x, float y, float Width, float
Height)
        {
            return new Rect(Screen.width*x, Screen.height*y, Screen.width*Width,
Screen.height*Height);
        }
        public static Rect GetRectangleGUICentered(float x, float y, float Width,
float Height)
        {
            Rect l_Rect=GetRectangleGUI(x, y, Width, Height);
            l_Rect.x-=l_Rect.width*0.5f;
            l_Rect.y-=l_Rect.height*0.5f;
            return l_Rect;
        }
        public static Vector2 GetNormalizedPosition(Vector2 Position)
        {
            return new Vector2(Position.x/Screen.width,
Position.y/Screen.height);
        }
        public static Vector2 GetNormalizedMousePosition()
        {
            Vector3 l_MousePosition=Input.mousePosition;
            l_MousePosition.y=Screen.height-l_MousePosition.y;
            return GUIController.GetNormalizedPosition(new
Vector2(l_MousePosition.x, l_MousePosition.y));
        }
        public static Vector2 GetNormalizedTouchPosition(Touch _Touch)
        {
            Vector3 l_TouchPosition=_Touch.position;
            l_TouchPosition.y=Screen.height-_Touch.position.y;
            return GUIController.GetNormalizedPosition(new
Vector2(l_TouchPosition.x, l_TouchPosition.y));
        }
```

```
    }
}
```

TransparentAdvancedFullShadows.shader

```
Shader "FX/Matte Shadow" {
Properties {
    _Color ("Shadow Color", Color) = (1,1,1,1)
    _ShadowInt ("Shadow Intensity", Range(0,1)) = 1.0
    _Cutoff ("Alpha cutoff", Range(0,1)) = 0.5
}


SubShader {
    Tags {
        "Queue"="AlphaTest"
        "IgnoreProjector"="True"
        "RenderType"="TransparentCutout"
    }
    LOD 200
    ZWrite off
    Blend zero SrcColor

CGPROGRAM
#pragma surface surf ShadowOnly alphatest:_Cutoff

fixed4 _Color;
float _ShadowInt;

struct Input {
    float2 uv_MainTex;
};

inline fixed4 LightingShadowOnly (SurfaceOutput s, fixed3 lightDir, fixed atten)
{
    fixed4 c;
    c.rgb = lerp(s.Albedo, float3(1.0,1.0,1.0), atten);
    c.a = 1.0-atten;
    return c;
}


void surf (Input IN, inout SurfaceOutput o) {
    o.Albedo = lerp(float3(1.0,1.0,1.0), _Color.rgb, _ShadowInt);
    o.Alpha = 1.0;
}
ENDCG
}

Fallback "Transparent/Cutout/VertexLit"
}
```

UOCTournament.cs

```
using System;
using UnityEngine;
using System.Collections.Generic;
using UOCUtils;

namespace UOCTournament
```

```
{
	public class CMatch<T>
	{
		private T m_Player1, m_Player2;
		private T m_Winner;
		private Delegate m_FnSimulate;
		private CDraw<T> m_Draw;

		public CMatch(T Player1, T Player2, Delegate FnSimulate, CDraw<T>
Draw)
		{
			m_Draw=Draw;
			m_Player1=Player1;
			m_Player2=Player2;
			m_Winner=default(T);
			m_FnSimulate=FnSimulate;
		}
		public void Simulate()
		{
			m_Winner=(T)m_FnSimulate.DynamicInvoke(m_Player1, m_Player2);
		}
		public T GetWinner()
		{
			return m_Winner;
		}
		public void SetWinner(T Winner)
		{
			m_Winner=Winner;
		}
		public T GetPlayer1()
		{
			return m_Player1;
		}
		public T GetPlayer2()
		{
			return m_Player2;
		}
		public CDraw<T> GetDraw()
		{
			return m_Draw;
		}
	}
	public class CDraw<T>
	{
		private List<CMatch<T>> m_Matches;
		private CDraw<T> m_NextDraw;
		private int m_CurrentMatch;
		private Delegate m_MatchFnSimulate;
		private int m_CurrentDraw;

		public CDraw(List<T> Players, Delegate MatchFnSimulate, int
CurrentDraw)
		{
			m_CurrentDraw=CurrentDraw;
			m_MatchFnSimulate=MatchFnSimulate;
			m_Matches=new List<CMatch<T>>();

			if((Players.Count%2)!=0)
				Debug.LogError("Trying to create a draw on tournament
with odd players '"+Players.Count+"'");

			for(int i=0;i<Players.Count;i+=2)
```

```
                        {
                                m_Matches.Add(new CMatch<T>(Players[i], Players[i+1],
m_MatchFnSimulate, this));
                        }
                        m_CurrentMatch=0;
                        m_NextDraw=null;
                }
                public bool IsFinished()
                {
                        return m_CurrentMatch==m_Matches.Count;
                }
                public void Simulate()
                {
                        if(IsFinished())
                        {
                                if(m_Matches.Count==1)
                                        return;
                                m_NextDraw.Simulate();
                                return;
                        }
                        m_Matches[m_CurrentMatch].Simulate();
                        NextMatch();
                }
                public void NextMatch()
                {
                        ++m_CurrentMatch;
                        if(IsFinished())
                                CreateNextDraw();
                }
                private void CreateNextDraw()
                {
                        if(m_NextDraw==null)
                        {
                                if(m_Matches.Count==1)
                                        return;
                                List<T> l_PlayersToNextDraw=new List<T>();
                                for(int i=0;i<m_Matches.Count;++i)

        l_PlayersToNextDraw.Add(m_Matches[i].GetWinner());
                                m_NextDraw=new CDraw<T>(l_PlayersToNextDraw,
m_MatchFnSimulate, m_CurrentDraw+1);
                                return;
                        }
                }
                public CDraw<T> GetLastDraw()
                {
                        if(m_NextDraw==null)
                                return this;
                        return m_NextDraw.GetLastDraw();
                }
                public List<CMatch<T>> GetMatches()
                {
                        return m_Matches;
                }
                public CDraw<T> GetNextDraw()
                {
                        return m_NextDraw;
                }
                public int GetCurrentMatch()
                {
                        return m_CurrentMatch;
                }
```

```
		public int GetCurrentDraw()
		{
			return m_CurrentDraw;
		}
	}
	public class CTournament<T>
	{
		private CDraw<T> m_MainDraw=null;
		public delegate T SimulateMatch(T Player1, T Player2);

		public CTournament(List<T> Players, Delegate MatchFnSimulate, bool
ShufflePlayers)
		{
			m_MainDraw=new CDraw<T>(ShufflePlayers ?
CUOCUtils.ShuffleList<T>(Players) : Players, MatchFnSimulate, 0);
		}
		public void Simulate()
		{
			m_MainDraw.Simulate();
		}
		public CDraw<T> GetLastDraw()
		{
			return m_MainDraw.GetLastDraw();
		}
		public bool IsFinished()
		{
			CDraw<T> l_LastDraw=GetLastDraw();
			List<CMatch<T>> l_Matches=l_LastDraw.GetMatches();
			return l_LastDraw.IsFinished() && l_Matches.Count==1;
		}
		public T GetWinner()
		{
			CDraw<T> l_LastDraw=GetLastDraw();
			List<CMatch<T>> l_Matches=l_LastDraw.GetMatches();
			if(l_LastDraw.IsFinished() && l_Matches.Count==1)
				return l_Matches[0].GetWinner();
			return default(T);
		}
		public CDraw<T> GetMainDraw()
		{
			return m_MainDraw;
		}
		public CMatch<T> GetCurrentMatch()
		{
			CDraw<T> l_LastDraw=GetLastDraw();
			return l_LastDraw.GetMatches()[l_LastDraw.GetCurrentMatch()];
		}
	}
}
```

UOCUtils.cs

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

namespace UOCUtils
{
    public class CUOCUtils
    {
        public static List<T> ShuffleList<T>(List<T> Elements, int Count=5)
```

```
            {
                  List<T> l_Elements=new List<T>();
                  for(int i=0;i<Elements.Count;++i)
                        l_Elements.Add(Elements[i]);
                  for(int i=0;i<l_Elements.Count*Count;++i)
                  {
                        int
l_Id1=((int)(UnityEngine.Random.value*l_Elements.Count))%l_Elements.Count;
                        int
l_Id2=((int)(UnityEngine.Random.value*l_Elements.Count))%l_Elements.Count;
                        if(l_Id1!=l_Id2)
                        {
                              T l_Aux=l_Elements[l_Id1];
                              l_Elements[l_Id1]=l_Elements[l_Id2];
                              l_Elements[l_Id2]=l_Aux;
                        }
                  }
                  return l_Elements;
            }
        public static float WrapAngle(float Angle)
        {
            if(Angle<0.0f)
                return Angle+2.0f*Mathf.PI;
            if(Angle>2.0f*Mathf.PI)
                return Angle-2.0f*Mathf.PI;
            return Angle;
        }
    }
}
```

UOCVirtualButton.cs

```
using UnityEngine;
using System.Collections;
using UOCUtils;

public class UOCVirtualButton : MonoBehaviour
{
    public Vector2 m_Position;
    public KeyCode m_KeyboardCode;
    public Texture2D m_ButtonTexture;
    public float m_TimePressed=0.3f;
    private float m_CurrentTimePressed;
    public Color m_PressedColor;
    public Color m_UnpressedColor;
    private Vector2 m_ButtonSizeNormalized;
    public bool m_Pressed;

    void Start()
    {
        m_CurrentTimePressed=m_TimePressed;
        m_ButtonSizeNormalized=new Vector2(m_ButtonTexture.width/1280.0f,
m_ButtonTexture.height/720.0f);
       }
       void Update()
    {
        m_Pressed=false;
#if UNITY_EDITOR || UNITY_STANDALONE
        Vector3 l_MousePosition=Input.mousePosition;
        l_MousePosition.y=Screen.height-l_MousePosition.y;
```

```
        if((Input.GetMouseButtonDown(0) &&
GUIController.GetRectangleGUI(m_Position.x, m_Position.y,
m_ButtonSizeNormalized.x, m_ButtonSizeNormalized.y).Contains(l_MousePosition))
            || Input.GetKeyDown(m_KeyboardCode))
        {
            m_Pressed=true;
            m_CurrentTimePressed=0.0f;
        }
#else
        if(Input.multiTouchEnabled)
             {
                    foreach(Touch l_Touch in Input.touches)
            {
                Vector3 l_TouchPosition=l_Touch.position;
                l_TouchPosition.y=Screen.height-l_TouchPosition.y;
                Vector2
l_NormalizedTouchPosition=GUIController.GetNormalizedTouchPosition(l_Touch);

                if((l_Touch.phase==TouchPhase.Began &&
GUIController.GetRectangleGUI(m_Position.x, m_Position.y,
m_ButtonSizeNormalized.x, m_ButtonSizeNormalized.y).Contains(l_TouchPosition)))
                {
                    m_Pressed=true;
                    m_CurrentTimePressed=0.0f;
                }
            }
        }
#endif
        m_CurrentTimePressed+=Time.deltaTime;
       }
    void OnGUI()
    {
        Color l_CurrentColor=GUI.color;
        GUI.color=m_CurrentTimePressed<m_TimePressed ? m_PressedColor :
m_UnpressedColor;
        GUI.DrawTexture(GUIController.GetRectangleGUI(m_Position.x, m_Position.y,
m_ButtonSizeNormalized.x, m_ButtonSizeNormalized.y), m_ButtonTexture);
        GUI.color=l_CurrentColor;
    }
}
```

UOCVirtualJoystick.cs

```
using UnityEngine;
using System.Collections;
using UOCUtils;

public class UOCVirtualJoystick : MonoBehaviour
{
    public Texture2D m_BackgroundTexture;
    public Texture2D m_StickTexture;
    private Vector2 m_BackgroundSize, m_BackgroundSizeNormalized;
    private Vector2 m_StickSize, m_StickSizeNormalized;
    public Vector2 m_Position;
    public Vector2 m_Direction;
    public float m_Angle;
    public bool m_Active;
    public float m_DeadZone=0.35f;
#if !UNITY_EDITOR && !UNITY_STANDALONE
    private int m_FingerId;
#endif
```

```
    void Start()
    {
        m_StickSize=new Vector2(m_StickTexture.width, m_StickTexture.height);
        m_StickSizeNormalized=new Vector2(m_StickSize.x/1280.0f,
m_StickSize.y/720.0f);
        m_BackgroundSize=new Vector2(m_BackgroundTexture.width,
m_BackgroundTexture.height);
        m_BackgroundSizeNormalized=new Vector2(m_BackgroundSize.x/1280.0f,
m_BackgroundSize.y/720.0f);
#if !UNITY_EDITOR && !UNITY_STANDALONE
        m_FingerId=-1;
#endif
       }
       void Update()
    {
        m_Direction=Vector2.zero;
#if UNITY_EDITOR || UNITY_STANDALONE
        if(Input.GetKey(KeyCode.RightArrow))
            m_Direction.x=1.0f;
        if(Input.GetKey(KeyCode.LeftArrow))
            m_Direction.x=-1.0f;
        if(Input.GetKey(KeyCode.DownArrow))
            m_Direction.y=1.0f;
        if(Input.GetKey(KeyCode.UpArrow))
            m_Direction.y=-1.0f;

        m_Direction.Normalize();
        Vector3 l_MousePosition=Input.mousePosition;
        l_MousePosition.y=Screen.height-l_MousePosition.y;
        Vector2
l_NormalizedMousePosition=GUIController.GetNormalizedMousePosition();

        if(Input.GetMouseButtonDown(0) &&
GUIController.GetRectangleGUI(m_Position.x, m_Position.y,
m_BackgroundSizeNormalized.x,
m_BackgroundSizeNormalized.y).Contains(l_MousePosition))
            m_Active=true;

        if(Input.GetMouseButton(0) && m_Active )
        {
            m_Direction=(l_NormalizedMousePosition-
(m_Position+m_BackgroundSizeNormalized*0.5f));
            m_Direction.x/=m_BackgroundSizeNormalized.x*0.5f;
            m_Direction.y/=m_BackgroundSizeNormalized.y*0.5f;
        }
        else
            m_Active=m_Direction.magnitude>m_DeadZone;
#else
        if(Input.multiTouchEnabled)
        {
            foreach(Touch l_Touch in Input.touches)
            {
                Vector3 l_TouchPosition=l_Touch.position;
                l_TouchPosition.y=Screen.height-l_TouchPosition.y;
                Vector2
l_NormalizedTouchPosition=GUIController.GetNormalizedTouchPosition(l_Touch);
                switch(l_Touch.phase)
                {
                    case TouchPhase.Began:
                        if(m_FingerId==-1)
                        {
```

```
                            if(GUIController.GetRectangleGUI(m_Position.x,
m_Position.y, m_BackgroundSizeNormalized.x,
m_BackgroundSizeNormalized.y).Contains(l_TouchPosition))
                            {
                                m_Active=true;
                                m_FingerId=l_Touch.fingerId;
                            }
                        }
                        break;
                    case TouchPhase.Stationary:
                    case TouchPhase.Moved:
                        if(m_FingerId==l_Touch.fingerId)
                        {
                            m_Direction=(l_NormalizedTouchPosition-
(m_Position+m_BackgroundSizeNormalized*0.5f));
                            m_Direction.x/=m_BackgroundSizeNormalized.x*0.5f;
                            m_Direction.y/=m_BackgroundSizeNormalized.y*0.5f;
                        }
                        break;
                    case TouchPhase.Canceled:
                    case TouchPhase.Ended:
                        if(m_FingerId==l_Touch.fingerId)
                        {
                            m_Active=false;
                            m_FingerId=-1;
                        }
                        break;
                }
            }
        }
#endif
        m_Active=m_Active && m_Direction.magnitude<=2.5f;
        if(m_Active)
        {
            Vector2 l_Direction=m_Direction;
            l_Direction.Normalize();
            m_Angle=Mathf.Atan2(l_Direction.x,l_Direction.y)-Mathf.PI/2.0f;
        }
#if !UNITY_EDITOR && !UNITY_STANDALONE
        else
            m_FingerId=-1;
#endif
       }
    void OnGUI()
    {
        GUI.DrawTexture(GUIController.GetRectangleGUI(m_Position.x, m_Position.y,
m_BackgroundSizeNormalized.x, m_BackgroundSizeNormalized.y),
m_BackgroundTexture);
        if(m_Active)

GUI.DrawTexture(GUIController.GetRectangleGUI(m_Position.x+((m_Direction.x+1.0f)/
2.0f)*m_BackgroundSizeNormalized.x-m_StickSizeNormalized.x*0.5f,
m_Position.y+((m_Direction.y+1.0f)/2.0f)*m_BackgroundSizeNormalized.y-
m_StickSizeNormalized.y*0.5f, m_StickSizeNormalized.x, m_StickSizeNormalized.y),
m_StickTexture);
    }
}
```

Figures

Glossari

Classe singleton, patró de disseny que garanteix que una classe només tinguin una instància única i proporciona un punt d'accés global a ella.

Combo, en un joc de lluita és un conjunt d'atacs consecutius.

GUI, veure Graphics User Interface.

Graphics User Interface, és la representació gràfica dels elements a nivell d'usuari amb els que el jugador pot interactuar.

IA, veure Intel·ligència artificial.

Intel·ligència artificial, és una intel·ligència creada mitjançant codi perquè un element reaccioni de forma raonada.

Prefab, en Unity és un objecte que ens permet ser utilitzat com una plantilla per crear objectes a partir d'aquest.

Bibliografia

Enllaç 1 http://www.aevi.org.es/index.php?option=com_mtree&task=att_download&link_id=11&cf_id=30, enllaç amb “Balance económico de la industria del videojuego 2009”.

Enllaç 2 http://www.artoolworks.com/support/library/ARToolKit_for_Unity, enllaç a la pàgina de la llibreria de visió per computador ARToolKit per Unity.

Enllaç 3 http://www.arpa-solutions.net/es/ARPA_Plugin_Unity, enllaç a la pàgina de la llibreria de visió per computador ARPA Augmented Reality.

Enllaç 4 https://research.cc.gatech.edu/uart/, enllaç a la pàgina de la llibreria de visió per computador UART.

Enllaç 5 https://developer.vuforia.com/, enllaç al portal de desenvolupament de la llibreria de visió per computador Vuforia de Qualcomm.

Enllaç 6 https://www.assetstore.unity3d.com/en/#!/content/12728, enllaç a la descàrrega del paquet de Unity de visió per computador Multi-Platform Marker Detection Plugin.

Enllaç 7 http://forum.unity3d.com/threads/matte-shadow.14438/, enllaç al fòrum de Unity d’on hem tret el shader per a integrar ombres en plans transparents.

Enllaç 8 https://www.youtube.com/watch?v=cmclsWGSLAs, enllaç al vídeo presentació de la PAC 3 del projecte.

Enllaç 9 https://developer.vuforia.com/resources/dev-guide/getting-started-unity-extension, enllaç a la pàgina on explica com començar amb la llibreria de visió per computador Vuforia de Qualcomm.