Mòdul de visió per computador per a videojoc 3D

David Martínez Pérez

Resum–
L'objectiu d'aquest projecte és el desenvolupament d'un mòdul de visió per computador per a un videojoc en 3D multiplataforma però molt enfocat a plataformes mòbils, fent servir les càmeres que disposen avui en dia aquests típus de dispositius. Aquest projecte està enfocat principalment en el desenvolupament d'una intel·ligència artificial (IA) en un entorn Unity que pugui ser capaç de fer deteccions i seguiments facials per tal d'aportar més realisme al personatge principal del videojoc. Pel desenvolupament d'aquest mòdul, s'ha utilitzat OpenCV, una llibreria de suport de visió per computador. El resultat final d'aquest projecte és una aplicació, tant com per a OSX com per a Android, d'una una escena 3D amb el personatge principal ja dotat amb aquesta intel·ligència. Això permetrà determinar quin efecte té, cap a l'usuari, tenir la sensació de ser observat.

Paraules clau– Mòdul, Unity 3D, Videojoc, Visió, multiplataforma, mòbil, Intel·ligència artificial, OpenCV.

Abstract– The aim of this project is to develop a computer vision module for 3D multi-platform game but focused on mobile platforms, using the cameras that nowadays are available in these devices. This project is focused mainly on the development of an artificial intelligence (AI) in a Unity environment, which is able to detection and face tracking in order to bring more realism to the main character of the game. For the development of this module, it has been used OpenCV, a library of computer vision support. The final result of this project is an app, for both OSX and Android, that contains a 3D scene with the main character endowed with this intelligence. This will determine what effect, to the user, having the feeling of being watched.

Keywords– Module , Unity 3D , Video-game , Vision, multi-platform, mobile, artificial intelligence , OpenCV

1 Introducció

ELSvideojocs cada vegada juguen més amb la realitat virtual i ja comença a no ser insòlit la utilització de diferents dispositius mòbils per a involucrar més al jugador en el món virtual.

L'empresa Big Imagination Games m'ofereix realitzar un mòdul d'intel·ligència artificial basada en la visió, que busca dotar a un personatge virtual en 3D de la capacitat de detectar cares i fer-ne un seguiment, identificar-les i reconèixer certes expressions facials.

La detecció i el seguiment de cares és un problema resolt i utilitzat en diferents àrees però molt innovador en el món dels videojocs. El que volem és ser capaços de dotar a un personatge d'intel·ligència visual per tal que pugui reconèixer cares i fer un seguiment amb el moviment del cap, simulant que el personatge ens contempla i és capaç de veure'ns.

2 Objectius

Coneixent les limitacions de temps i el gruix del mòdul a realitzar, en aquest TFG es realitzarà només la detecció i seguiment de cares.

El mòdul haurà de ser desenvolupat amb Unity 3D [1] per poder recrear el moviment tridimensional del personatge.

Ja que el videojoc estarà disponible per diferents plataformes, i aprofitant la capacitat de Unity, és necessari que sigui possible l'execució en diferents sistemes operatius i es farà especial èmfasi en les opcions per a dispositius mòbils més utilitzades que són iOS i Android.

Abans de la integració dins del projecte, es crearà una es-

• E-mail de contacte: David.Martinezpere@e-campus.uab.cat
• Menció realitzada: Computació
• Treball tutoritzat per: Francesc Xavier Roca Marva (CVC)
• Curs 2015/16

cena de proves per poder testejar a fons les possibilitats del nostre sistema. Així, s'optimitzarà el sistema per a garantir una experiència gratificant per a l'usuari.

L'algorisme no ha de trigar mes de 16 ms en renderitzar un frame (60 fps) en un telèfon mòbil d'alta gamma.

3 ESTAT DE L'ART

3.1 Introducció històrica

Els videojocs:

Els videojocs tenen el seu origen en la dècada dels 40, després de la segona guerra mundial, les potències vencedores van construir les primeres supercomputadores programables com el ENIAC[2] (1946). No van trigar en aparèixer els primers intents de crear software lúdic i es van anar repetint durant les següents dècades.

A la dècada dels 60 apareixen els primers videojocs moderns establint així un món que no ha deixat de créixer i desenvolupar-se amb els límits imposats per la creativitat dels seus desenvolupadors i l'evolució tecnològica.

Al igual que va ocórrer amb el cinema i la televisió, el videojoc ha aconseguit, amb només mig segle d'història, l'estatus de mitjà artístic. En molt poc temps exerceix un impacte en les noves generacions que el veuen com un nou mitjà audiovisual que permet protagonitzar les teves pròpies aventures.

Els dispositius:

Des del mateix naixement dels videojocs, la interacció humà-màquina ha quedat limitada a l'ús de comandaments. Al llarg de la història, aquests comandaments formats per joysticks i botons, han anat adaptant diferents formes per aconseguir una millor experiència de joc però casi sempre ha quedat limitada a l'ús de les mans degut a que ha sigut la interacció més eficient i fiable.

3.2 Els nous sistemes i les noves formes d'interacció

En l'actualitat, i des de fa pocs anys, aquesta visió clàssica dels videojocs, està sent radicalment canviada amb l'aparició dels dispositius tàctils i sobretot amb l'arribada del smartphone, els nous terminals de telefonia mòbil intel·ligents. El poder computacional d'un smartphone és equiparable al d'un ordinador domèstic, això permet als desenvolupadors crear aplicacions accessibles a un gran gruix d'usuaris amb una alta qualitat gràfica que abans estava restringida pels ordinadors d'alta gama.

L'aparició dels sistemes de visió i captació de moviment per a consoles domèstiques també han aconseguit revolucionar la manera d'interactuar amb certs tipus de videojocs.

Sony, al 2003, amb l' Eyetoy[3] per a la Playstation2 i Playstation Portable, va esser la primera en començar a explorar les possibilitats d'aquests sistemes creant així una sèrie de jocs basats en la capacitat d'una càmera. Però va esser Nintendo, a l'any 2006 amb la wii, la primera en crear una consola basada en un sistema de captació de moviment a través de comandaments sense fils connectats per infraroig. Al 2010, Microsoft influenciat pels antecedents, decideix crear i apostar per un sistema altament sofisticat e innovador, el Kinect[4], que amb una càmera potent permet a l'usuari tenir una interacció molt alta per moviment i veu. Sony havia decidit abandonar l'Eyetoy per a la nova generació de

Playstation, la Playstation 3, però, influenciat pel wiimote[5] i el kinect, també decideix treure al mercat el seu propi sistema de captació de moviment i de visió, el Playstation Move [6] juntament amb el Playstation eye [7], i més endevant, per a Playstation 4, el seu sistema de visió i reconeixement de veu més sofisticat i a l'altura del kinect, la Playstation Camera [8].

3.3 OpenCV i Unity3D

3.3.1 La llibreria OpenCV[9]

OpenCV (Open Source Computer Vision Library) és una llibreria de visió i aprenentatge per computador en temps real de codi obert. Va esser desenvolupada per la divisió russa d'Intel. L'ús és gratuït sota la llicencia open source BSD. La llibreria té més de 2500 algorismes optimitzats. Aquests algorismes poden ser utilitzats per al reconeixement facial, la identificació d'objectes, el seguiment a través de càmeres, l'extracció de models 3D, etc.

La comunitat d'OpenCV té poc menys de 50 milers d'usuaris i l'estimació de descarregues es de 7 milions. És utilitzada en companyies empresarials, en grups de recerca i en diferents cossos governamentals.

És una llibreria escrita nativament en C++ i també té interfície per a C, Python, Java i MATLAB. És una llibreria multiplataforma per tant suporta Windows, Linux, Android i MacOS.

3.3.2 Unity3D

Unity, creada per Unity Technologies, és una plataforma de desenvolupament flexible i molt poderosa per a desenvolupar jocs i experiències interactives 3D i 2D multiplataforma. Permet crear jocs per a Windows, OS X, Linux, Xbox 360, Xbox ONE, PlayStation 3, PlayStation 4, Playstation Vita, Wii, Wii U, iPad, iPhone, Android i Windows Phone. A més, gràcies al plugin web de Unity, també és possible desenvolupar videojocs de navegador per a Windows y Mac.

3.3.3 OpenCV i Unity3D

A simple vista sembla que OpenCV és la millor opció pel desenvolupament d'aquest projecte però ja d'entrada ens trobem amb el primer problema.

Unity té basat l'Script en Mono[10], la implementació de codi obert de .NET Framework. Per tant, permet l'execució i el desenvolupament en Javascript, Boo i C#. Com ja hem vist amb anterioritat, OpenCV no té una interfície cap a C# i és per això que caldrà buscar un wrapper que permeti fer de pont entre Unity i OpenCV.

Per al desenvolupament d'aquest projecte,he trobat diferents opcions bastant conegudes en la comunitat de Unity.

EmguCV[11] i OpenCVSharp[12] són dos wrappers que tenen molt suport documentatiu però la llicencia no compleix amb els requeriments de l'empresa ja que requereix

de la publicació del codi que es realitzi amb la seva utilització. Els requeriments de la empresa no em permeten realitzar codi destinat a la comunitat Open Source.

Una altre solució, el wrapper OpenCV.NET [13]. No és un wrapper massa estès però la seva construcció és realitzada sent molt fidel a la llibreria OpenCV. Això facilita molt la solució del problemes per la manca de documentació. Requereix de les llibreries d'OpenCV per a totes les plataformes i les de Android no estan proveides de compatibilitat amb Unity. Es a dir, no es poden incloure dins d'una Build.

Després d'haber intentat provar diferents possibles solucions sense bons resultats (com per exemple utilitzar java com a pont [24]), es va optar per usar un package de Unity que ja donés una integració directe tot i no tenir el control absolut de com s'havia realitzat.

3.4 La detecció de cares

La detecció de cares dirigida per computador busca identificar automàticament rostres en una imatge o un vídeo. Aquest procés és molt senzill per al SVH (Sistema Visual Humà) però complex per a un computador.

3.4.1 Els mètodes

Actualment es poden distingir quatre categories de mètodes diferents.

Mètodes basats en el coneixement

Són aquelles tècniques que es basen en unes regles prèvies definides per el desenvolupador sobre les característiques més importants sobre les cares a detectar. Quan més generals siguin les regles, més fàcil és que hi hagin falsos positius. Si pel contrari, són més estrictes, possiblement no es detectin totes les cares desitjades.

Mètodes basats en caràcters invariants

Aquests mètodes utilitzen com a referència el color de la pell i la textura. Són susceptibles a tenir problemes amb els canvis d'il·luminació o al soroll en la imatge. Amb el to de la pell, els algoritmes que utilitzen tota la gama de colors presenten millors resultats que els que utilitzen una escala de grisos.

Mètodes basats en plantilles

Aquests mètodes utilitzen plantilles i s'avalua la correspondència entre la cara i la plantilla. Es necessita modelar geomètricament l'objecte a analitzar.

Mètodes basats en l'aparença

No es necessita un coneixement explícit sobre les característiques de la cara que es vol detectar. Necessiten d'algorismes d'aprenentatge per extreure les característiques necessàries.

3.4.2 La detecció de cares utilitzant Haar Cascades: Viola-Jones [14]

Viola-Jones és un algorisme de detecció d'objectes proposat per en Paul Viola i Michael Jones en el seu article "Rapid Object Detection using a Boosted Cascade of Simple Features" en el 2001. És un algorisme d'aprenentatge computacional basat en una funció cascada que ha d'esser entrenat amb un conjunt d'imatges positives, aquelles imatges en les que hi apareix l'objecte a identificar (en el nostre cas, imatges amb cares), i negatives, aquelles que no hi apareix.

En la fase d'entrenament, aquest algorisme, fa una recol·lecció de característiques que són determinants alhora de la detecció. Per fer això utilitzem les característiques Haar (Fig.1), dos basades en dos rectangles, una basada en tres rectangles i un altre basada en quatre rectangles, com un nucli de convolució.

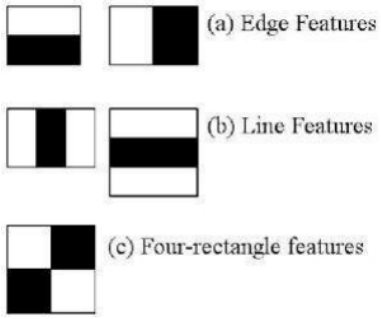

Fig. 1: Representació de les característiques Haar

Les característiques que extraiem són un valor simple obtingut de la luminància dels píxels en la regió blanca restat a la mateixa suma a la regió obscura. Tot i així, per simplificar els càlculs, Viola-Jones utilitza una representació intermèdia de la imatge, anomenada imatge integral.

La imatge integral ens permet calcular ràpidament la suma dels píxels dins d'un àrea determinada de la imatge ja que aquesta suma s'obté a partir de quatre referències (Fig.2).

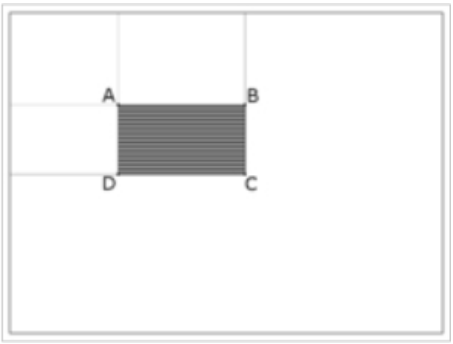

Fig. 2: Les quatre referències per realitzar la suma de píxels sobre una imatge intergral

D'aquesta forma, cada àrea es calcularà com:

$$\sum_{\substack{A(x)<x'\leq C(x)\\ A(y)<y'\leq C(y)}} i(x,y) = sum(A)+sum(C)-sum(B)-sum(D)$$

Per tal de seleccionar les característiques Haar que millor classifiquen per posteriorment fer l'entrenament, Viola-Jones utilitza Adaboost[15].

Adaboost combina una sèrie de classificadors dèbils, classificadors en que el seu encert no és molt millor que l'aleatorietat, als que els hi assigna diferents pesos per a, finalment, formar un classificador fort. D'aquesta forma

aconsegueix passar de més de 160.000 característiques, que impossibilita la detecció en temps real, a avaluar "només" 6.000 aproximadament.

Tot i així segueix sent ineficient el fet d'haver-hi de mirar, per exemple en una finestra de 24x24, les 6.000 característiques per finalment decidir si és una cara o no. Segurament ja sabíem d'abans que podia ser descartable.

Aquí s'inclou el concepte de Classificador en Cascada. Viola-Jones, en comptes de comprovar les 6.000 característiques directament, les agrupa en nivells (fent que els primers nivells tinguin menys característiques). Si una finestra no passa un nivell, ja no es comproven els següents i es descarta, si els passa tots es considera que és una cara. Llavors, el pitjor dels casos que ens podem trobar és que una finestra caigui just en l'últim nivell, tot i així, com a molt tindrà el mateix cost que mirar les 6.000 una per una.

3.5 El seguiment facial

Un cop detectada la cara objectiu, s'iniciarà un procés de seguiment facial de la mateixa, intentant evitar el número màxim de distorsions, com una nova cara a l'escena o un canvi d'il·luminació, i buscant la forma més optimitzada de fer-ho.

El cost del seguiment facial ha de ser menor que el d'una nova detecció per a que així sigui justificable aplicar-hi un algorisme diferent en comptes de fer deteccions repetidament.

3.5.1 CAMShift [16]

CAMShift (Continuously Adaptive MeanShift), és una ampliació del Mean-Shift [17] i resol el problema de que la finestra sigui estàtica ajustant-la (finestra) segons la proximitat o la llunyania de l'objecte. Està basat en quatre passos, la creació del histograma, la probabilitat facial de cada píxel, el desplaçament del rectangle que conté la cara i el càlcul de la mida.

L'histograma de color només serà calculat pel rectangle que contingui la cara ja detectada i el desplaçament que es vulgui seguir.

CAMShift té el problema que, a l'estar basat en el color de la imatge, és molt susceptible a canvis d'il·luminació, un canvi brusc podria distorsionar completament el seguiment de la cara. De la mateix manera, es veu molt afectat en fons molt semblants al to de la pell humana.

També, si la cara en seguiment surt de l'escena i en el seu lloc hi posem una mà, l'algorisme es confondrà, ja que la tonalitat de pell és la mateixa o molt semblant, i començarà a fer el seguiment d'aquesta, i així podria passar amb qualsevol part del cos nuu. Per solucionar aquest problema haurem de relacionar aquest algorisme amb un altre de detecció de formes però si fem això, el cost computacional augmentarà perdent així la principal avantatge de l'algorisme.

3.5.2 Seguiment amb informació temporal

Aprofitant el potencial que té l'algorisme Viola-Jones, ja que funciona a temps real, podem utilitzar-lo per fer el seguiment sempre buscant la manera d'optimitzar-lo. Amb aquest algoritme el que es pretén es aprofitar la informació temporal relacionada als videos. Es a dir, un frame no variarà molt del anterior, el moviment produït serà poc. Sabent això podrem reduir l'àrea de recerca de cares, no caldrà que mirem en tot el nou frame on hi ha una cara.

D'aquesta forma, en els pitjors del casos, l'algorisme tindrà el mateix cost que realitzar l'algoritme Viola-Jones directament per cada frame. Com ja hem dit, Viola-Jones, funciona en temps real.

4 METODOLOGIA

Scrum[18] és la metodologia emprada en aquest projecte, la mateixa que és usada a l'empresa, per així estar en concordança amb l'equip del projecte.

També seré partícep de tots aquells daylies en els que els horaris em deixin assistir i, de totes formes, informaré diàriament al meu tutor de l'empresa, que a la vegada em farà de product owner juntament amb el meu tutor de treball, de tota la feina feta i de tot el que es realitzarà.

4.1 La metodologia Scrum en aquest projecte

Cal dir, ja que el projecte no s'ha relitzat en equip, que no totes les característiques d'aquesta metodològia han sigut emprades però si les més essencials i adients per aquest projecte.

Scrum és una metodologia àgil molt adient en el desenvolupament de software en general i també en el sector dels videojocs. Actualment és una pràctica molt utilitzada.

Cada participant del projecte assumeix un rol. En el cas d'aquest projecte, alguns rols han quedat difosos entre vàris participants. Sobretot hi ha tres rols essencials i coverts. L'Scrum Master, estarà covert tant com pel meu tutor de TFG com el meu tutor de pràctiques d'empresa. El Product Owner, aquest sobretot vindrà marcat pel meu tutor de pràctiques i pel productor del joc. També jo mateix he aportat alguns punts importants d'aquest rol. L'equip de desenvolupament, que com ja he explicat amb anterioritat, en aquest cas només serà format per mi.

Tot i que en el transcurs d'aquest projecte s'han anat modificant, els objectius definits han sigut dividits par tasques assignant-lis una aproximació del temps necessàri per a la seva realització.

Un cop definides aquestes tasques, el Product Owner, en aquest cas el meu tutor de pràctiques amb la meva col·laboració, les ordena per ordre d'importància i preferències, tot creant el que es coneix com a Product Backlog, el conjunt de requeriments a alt nivell prioritzats que defineixen la feina a fer en un període determinat.

4.2 Planificació

Per atacar els objectius del projecte i tal com demana la metodologia Scrum, s'ha dividit el projecte en tasques i subtasques. En aquest apartat s'explicarà quines són i com han sigut planificades aquestes tasques per ordre de preferència:

- L'integració d'OpenCV a Unity. És una tasca molt important i crítica pel desenvolupament del projecte. Cal que estigui disponible com a mínim per a OSX i Android.
- L'accés a la càmera del dispositiu contemplan les diferents opcions. Tenim dos formes de connectar-nos a la càmera, amb la llibreria OpenCV o directament amb

Unity. En aquesta tasca s'ha de implementar i decidir quina es la millor manera.

- La detecció facial amb els mètodes proporcionats per OpenCV. Bàsicament, realitzar la implementació del codi utilitzan l'algorisme Viola-Jones d'OpenCV i realitzar proves de costos i funcionament.
- La creació de l'escena de proves. Una tasca senzilla que aportarà un gran valor a l'hora de extreure conclusions del algorisme.
- El seguiment facial. Implementar l'algorisme de seguiment facial amb informació temporal e integrar-lo a l'escena de proves per tal de fer una valoració de rendiment.
- Finalment la integració al projecte principal. Aquesta tasca vol que el mòdul s'integri dins del personatge principal i que s'extreguin conclusions de sensacions i també de rendiment.

5 DESENVOLUPAMENT DEL PROJECTE

El projecte té tres blocs en termes generals. El seguiment facial (el bloc relacionat amb la part d'implementació de la IA), el moviment del personatge principal per tal de crear un moviment realista i la interficie d'usuari (escenes i eines de test visuals). En aquest apartat s'explicarà com han sigut dissenyats aquests blocs per tal de realitzar aquest projecte.

5.1 El seguiment facial

5.1.1 La implentació del algorisme de Seguiment

Per aquest projecte s'ha optat per utilitzar l'algorisme del seguiment amb informació temporal tot i que en un futur, si no es compleixen les espectatives, aquest podria ser substituït o reinventat per algún altre mètode que pogués ser més eficaç.

L'algorisme ha sigut dissenyat e implementat tal i com mostra el diagrama de flux de la Fig.3.

Amb aquest algorisme no caldrà que analitzem tota la imatge de cada frame proporcionat per la càmera, sinò només, on podem preveure que hi serà la següent cara.

5.1.2 Arquitectura de la solució

En aquest apartat es presenta l'arquitectura proposada per solucionar la part més important del mòdul, l'algorisme de detecció i seguiment facial. Aquest mòdul ha de ser capaç de connectarse a la càmera del dispositiu, fer captures facials a partir d'una imatge i realitzar un seguiment d'aquesta cara capturada.

El nostre disseny contempla la utilització de la classe abstracte MonoBehaviour [21], una classe proporcionada per Unity que ens ajuda a manegar els objectes que interactuen en una escena. Tots els Scripts que heredin de MonoBehaviour podràn ser afegits com a components de l'escena. Per tant, tal i com es mostra en la Fig.4, en l'escena tindrem un gameobject que actuarà com a CameraController i l'Script FaceTracking serà afegit com a component al gameobject que volem que faci el seguiment, en el nostre cas el personatge.

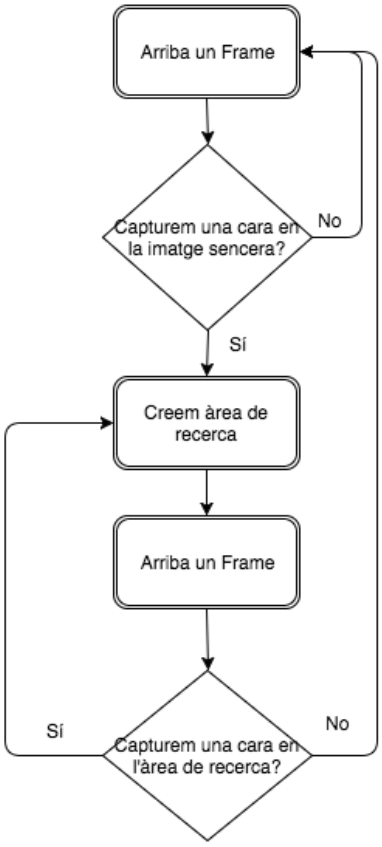

Fig. 3: Diagrama de flux de l'algoritme

Per poder assolir els requeriments demanats, s'han diferenciat quatre funcionalitats necessàries que han sigut resoltes de la següent manera:

- El CameraController: Necessitem una entitat capaç d'establir una connexió i manegar la càmera del dispositiu. Aquesta classe és l'encarregada de demanar l'autorització d'ús de la càmera al dispositiu i de subministrar els frames al algorisme de seguiment.

  Aqusta classe tindrà un mètode públic, TakePhoto, encarregat de retornar un frame de la càmera ja passat a escala de grissos ja que, per als algorismes implementats, no necessitarem el color de la imatge. A més, mirarà si cal retornar la imatge en vertical o horitzontal depenent si es tracta de la versió en OSX o Android.

- El FaceDetector: Aquesta classe s'encarrega de fer les deteccions de cares sobre un frame de forma diferent depenent de l'estat en el que ens trobem de l'algorisme de seguiment (s'explicarà amb més detall més endevant).

  També s'encarrega de carregar l'entrenament necessari per a la detecció. Té tots els paràmetres necessàris per aplicar l'algorisme Viola-Jones. Té dos atributs importants que delimiten a quina distància es deixa o es comença a detectar una cara, la finestra mínima i màxima per a aplicar a l'algorisme.

- El SearchRect: Serveix per representar l'àrea de recerca. Hem optat per crear una classe que realitzi i contingui tots els càlculs e informació necessària per representar un àrea de recerca donat el rectangle que

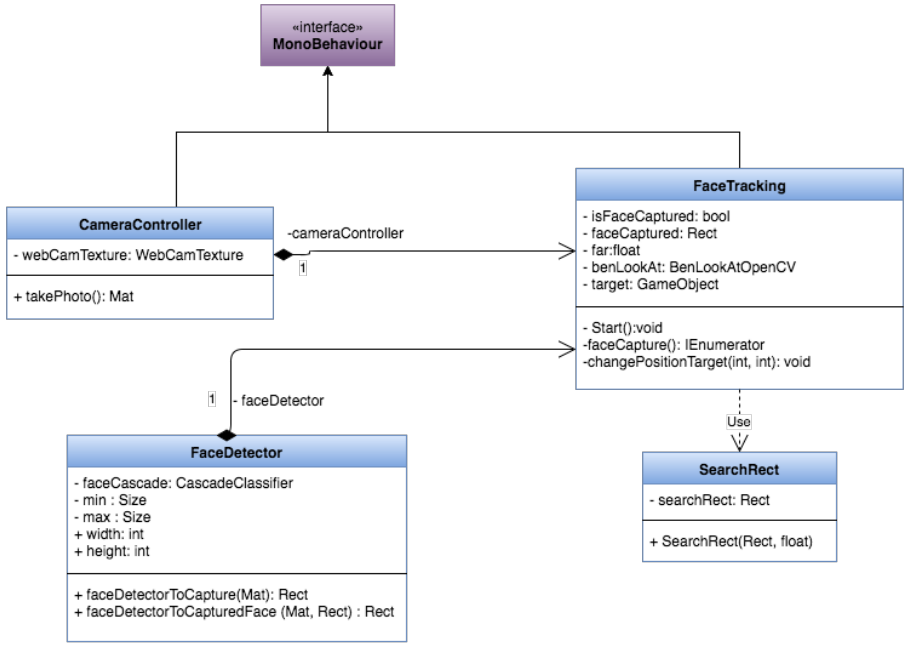

Fig. 4: Diagrama de classes

representa una detecció facial i un factor multiplicatiu que s'aplicarà per dimensionar aquest espai de recerca.

Aquesta classe serà utilitzada pel FaceTracking, que calcurà un àrea de recerca (un nou SearchRect) cada cop que es detecti una nova cara.

- El FaceTracking, serà la classe encarregada d'aplicar e implementar l'algorisme de seguiment facial que utilitzarem en la nostra aplicació, es a dir, la part de IA més gran del mòdul.

  Per controlar els diferents estats s'ha realitzat mitjançant una màquina d'estats[28]. Les màquines d'estats són molt utilitzades en el món del videojocs i permeten controlar diferents estats i transicions d'una IA. En el nostre cas tindrem dos estats amb dos transicions d'anada i tornada.

  - L'estat de no detecció, quan no tenim una cara detectada i busquem en tota la imatge per trobar-ne una.

  - L'estat de cara detectada (o de detecció), es aquell estat en el que ja hi tenim una cara detectada i l'àrea on cerquem una cara ha quedat reduida.

  Si ens trobem en l'estat de no detecció i detectem una cara transitarem cap al estat de detecció, de la mateixa manera, si perdem la cara pasarem un altre cop al estat de no detecció. Amb aquesta arquitectura podrem afegir més estats si en el futur són necessaris, com per exemple un estat de reconeixement d'expressió i fer que el personatge tingui algun típus de reacció emocional.

5.2 Moviment del Personatge

Un cop provat que el mòdul de seguiment funciona com es demana, volem assolir un moviment realista del personatge principal del joc cap a un punt donat pel FaceTracking.

Per poder articular el personatge donat un punt, s'ha optat per utilitzar un Plugin d'Inverse Kinematics[25], concretament el Final IK[26]. Aquest Plugin permet donada una posició, fer el càlcul dels angles corresponents per a conseguir, en el nostre cas, que acabi mirant aquest punt (LookAtIK). Es a dir, en aquest TFG no s'ha realitzat res que tingui a veure amb el càlcul d'angles per tal d'assolir aquest objectiu.

Per a configurar tot això, s'ha realitzat un altre script que permet controlar i configurar l'observació del personatge al objectiu, el BenLookAtOpenCV. Aquesta classe també hereda de MonoBehaviour i s'afegirà com a component al personatge (Fig. 5).

Caldrà identificar quin es l'objectiu (target) en tot moment si n'hi ha algún (si hi ha cara detectada o no). D'això s'encarregarà la classe FaceTracking que cridarà al mètode i modificarà el target cridant al mètode SetTarget cada cop que tinguem una nova posició. Aquesta relació es pot observar en la Fig. 5.

El setTarget s'encarregarà de cridar als mètodes del LookAtIK necessàris per fer el càlcul d'angles de la rotació.

Per tal de crear un moviment molt més realista, s'ha decidit que el personatge principal no realitzi només un moviment de cap. S'han definit tres típus de moviments diferents amb pesos d'importància diferents, entenén com a importància quin serà el grau d'enfocament cap al objectiu.

- El primer moviment serà del cos. Aquest començarà a enfocar-se en la direcció del objectiu però no necessàriament arribarà a estar enfocat del tot, es a dir, té una importància baixa (valor de 0 a 0.5).

- Tot seguit començarà a desplaçar-se el cap arribant a quasi una enfocació total. Té una importància mitjaalta (valor de 0.5 a 0.8).

- Per últim seràn els ulls que acabaràn enfocant-se just cap a la posició del objectiu. Aquestos són els que tenen una importànica més alta (valor 1).

Aquest script està realitzat per a que, els paràmetres esmentats ara mateix, siguin modificables desde el editor de Unity.

Cal destacar, que amb el moviment del cap o del ulls ja tindriem una sensació d'observació alta, però amb la combinació d'aquestes tres, la sensació de realisme que es crea és molt elevada.

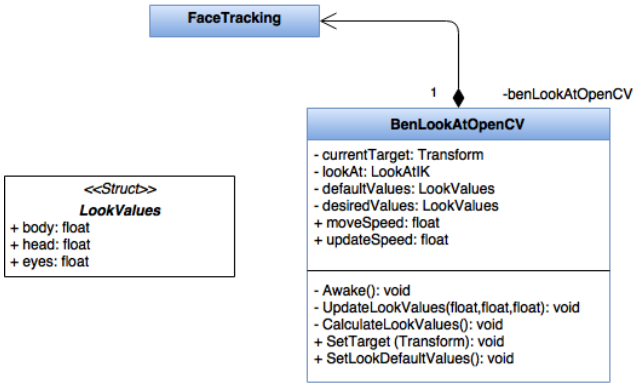

Fig. 5: Arquitectura de la classe BenLookAtOpenCV

5.3 La interfície d'usuari

En aquest bloc s'explica tot el desenvolupat relacionat amb la interfície i visió de l'usuari, tant com les escenes que han sigut creades com eines per poder realitzar un testeig més còmodament.

5.3.1 Escenes

Durant el transcurs del projecte s'han realitzat dues escenes. En aquest apartat volem ensenyar com son i quina informació volen representar.

Una ens servirà per a extreure conclusions sobre rendiment i l'altre per poder veure el funcionament i l'efecte sorgit de proporcionar aquest mòdul al personatge principal.

- L'escena de proves. És aquella que ens ha de servir per poder testejar el nostre algorisme sense necessitat d'integrar-lo a l'escena real del joc i que servirà, sobretot, per extreure conclusions de rendiment.

  Ha de tenir unes característiques extrapolables a les del joc. L'escena es composada per un dau de sis cares on la cara número 1 representarà la part frontal del personatge principal. És aquesta cara la que ens ha de seguir.

  En aquesta escena podrem treure conclusions més precises sobre el rendiment del mòdul ja que es troba menys contaminada d'altres funcionalitats que no siguin exclusivament necessàries pel mòdul. Es per això, i com es mostra en la Fig.6, que podem desactivar la sortida de la càmera, ja que això fa que l'algorisme consumeixi temps extra.

- L'escena amb el personatge real. Aquesta escena és una simulació a una escena real del joc, en la que disposem ja d'un personatge principal animat amb el que haurem d'interactuar.

  Per tant, el personatge ja tindrà integrat el mòdul i els scripts de moviment, també desenvolupats en aquest treball i ja explicats amb anterioritat, per poder realitzar el moviment cap a la posició desitjada.

Fig. 6: Escena de proves amb la sortida de càmera desactivada.

En la Fig.7 veiem com queda l'escena en la qual el personatge es troba en estat de detecció. Podem observar que ens trobem en aquest estat per el missatge mostrat en la UI. Se'ns mostra la sortida de la imatge de la càmera, quina és l'àrea de la cara detectada i com queda l'àrea de recerca per a la següent captura.

Fig. 7: Escena amb el personatge amb estat de detecció.

Per tal de poder usar les dos escenes, s'ha creat, per a cada una, un executable per a OSX i un altre per a Android. Per usar-les només caldrà iniciar l'aplicació i donar els permisos d'utilització de la càmera si es el primer cop que s'inicia en aquest dispositiu. Un cop realitzat això, l'aplicació ja estarà connectada amb la càmera i es mostrarà el personatge que ja haurà començat a executar l'algorisme.

5.3.2 Eines de Test

Alhora d'ajudar-nos més i poder realitzar test més comodament, s'han dissenyat dos eines:

- El UI Controller (Fig. 8), per tal de mostrar i representar la informació que genera l'algorisme. Els mètodes implementats seràn cridats pel FaceTracking.

  En les dues escenes es mostrarà informació a l'usuari per poder interpretar com esta funcionant l'algorisme. Se'ns mostrarà en tot moment si hi ha una cara capturada o, si pel contrari, l'algorisme es troba en un estat de no detecció. També en l'escena de proves, i com s'ha explicat amb anterioritat, es mostrarà el percentatge d'àrea que està sent analitzada en tot moment. Sempre que hi hagi una cara detectada, mostrarem com es veu afectat això a la posició del objecte, es a dir, a quin punt estarà mirant el nostre personatge.

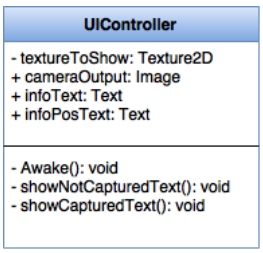

Fig. 8: Arquitectura de la classe UIController

Per últim, i com a eina més visual sobre el funcionament del nostre algorisme, la nostra UI ens mostrarà la imatge que està capturant la càmera en aquell moment i, un cop processada per l'algorisme, amb un requadre es marcarà la zona on es troba la cara detectada. A més, l'algorisme de seguiment també ens mostrarà en quina àrea s'està cercant la pròxima cara tal i com es mostra a la Fig.9.

Fig. 9: L'àrea de cara i on la buscarà al següent frame.

- La consola de desenvolupament. S'ha desenvolupat una consola que mostri els registres de debug necessàris i els errors o missatges interns de Unity ja que Unity no dona aquesta eina per a builds de dispositius mòbils. L'arquitectura d'aquesta eina es mostra en la Fig. 10.

  La consola es controlada per un ConsoleManager que, en el mètode OnEnable, permet que els missatges de Unity vagin dirigits al HandleLog. En el HandleLog es genera un objecte de la classe Log amb l'informació del missatge que ha arribat i es guardarà en una llista. Un cop fet això, es cridarà al paintConsole() que aquest printarà la llista de Logs desats.

6 RESULTATS

Per extreure els resultats ens basem en el rendiment que té el mòdul i en el funcionament i l'efecte de l'algorisme un cop ha sigut integrat al personatge principal.

6.1 Rendiment

Volem saber quina es la millora de rendiment que ofereix utilitzar l'algorisme de seguiment facial. Analitzarem la reducció d'espai en píxels que estalviem contra fer una detecció de la imatge sencera cada cop i també, voldrem saber

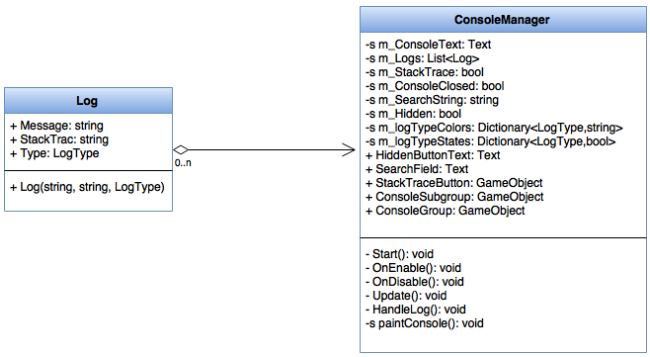

Fig. 10: Arquitectura de la consola de desenvolupament

la reducció d'ús de CPU que això suposa, en especial, el temps que triga un frame.

El test es realitzarà amb l'escena de proves i per a plataformes OSX i Android. D'aquesta manera podrem veure més clarament que es el que triga l'algorisme sense cap distorsió dels temps per altres algorismes.

Per poder veure millor els resultats, s'ha afegit un camp a la UI per veure quin es el percentatge sobre el total que s'analitza de la imatge.

Per veure el consum de CPU utilitzarem l'eina Profiler[27] proporcionada per Unity que ens permet veure el consum de CPU en percentatge i el que triga un frame en renderitzarse.

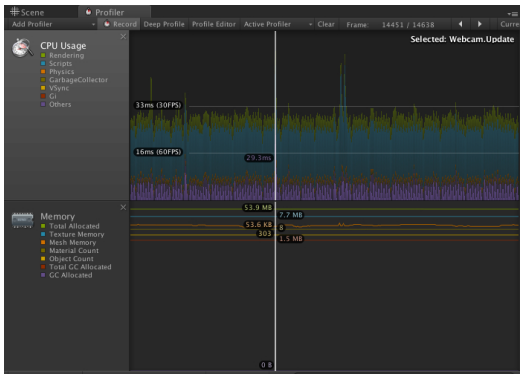

Fig. 11: Profiler de Unity

Tal i com està marcat en els objectius, el temps que triga un frame en un mòbil d'alta gama com en el que realitzem el test, no pot ser superior a 16 ms (60 fps). Si fos així, s'hauria de buscar una nova optimització o inclús, ja fora d'aquest projecte, una nova forma de realitzar-lo.

L'algorisme triga aproximadament (depenent del tamany de l'àrea de recerca) uns 13 ms per frame, és a dir, compliria amb els requisits. Tot i que, quan s'ha de realitzar Viola-Jones a la imatge sencera, l'algorisme presenta un pic d'aproximadament 30 ms.

6.2 Funcionament i efecte

Per a dir que el mòdul funciona, ha de ser capaç de detectar una cara i fer un seguiment exclusiu d'aquesta.

Amb aquestes simulacions, podrem determinar si els requisits de funcionament i d'efecte, que sorgeix de dotar d'aquesta intel·ligència al nostre personatge, s'han complert. A més, ens servirà per acabar d'ajustar paràmetres i intentar crear una sensació de realisme més gran.

El seguiment ha de funcionar en vertical i en horitzontal, es a dir, si el moviment es d'esquerra a dreta o de dalt a baix. Sempre que no hi hagin adversitats massa grans de lluminositat com pot ser un contrallum fort o poca il·luminació, l'algorisme continuarà funcionant. Recordem que aquest algorisme es basa en la forma dels objectes i no en el color.

Podem observar el funcionament i l'efecte que es crea amb el moviment horitzonal en la Fig. 12 i el vertical en la Fig. 13.

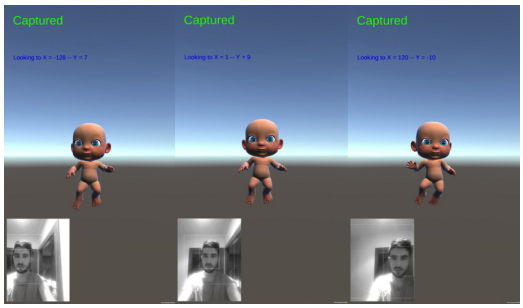

Fig. 12: Moviment horitzontal

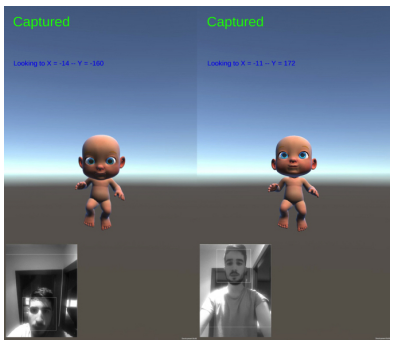

Fig. 13: Moviment vertical

El tracking només funciona amb cares frontals. Si la cara que es vol capturar és troba girada o inclinada, no aconseguirem capturar-la. Per tant, aqui hi ha un punt feble del algorisme, en el moment que l'usuari no miri cap al front, la cara es perdrà.

També, i depenent del rectangle mínim i màxim, si la cara es troba molt lluny o molt a prop, aquesta no serà detectada. Com són paràmetres adjustables, no té gaire importància e inclús es una aventatge per no detectar cares que no interessin.

El seguiment ha de ser intolerable quan apareguin noves cares. Es pot ajustar l'àrea de recerca per tal de que sigui suficientment petita com per no detectar un altre cara que es trobi a prop com es veu en la Fig. 14. D'aquesta forma, utilitzem l'algorisme de seguiment amb dos objectius, estalviar processament (rendiment) i no detectar altres cares (funcionament). També, per cada cara detectada (si n'hi hagués més d'una), es calcula quin és la cara que queda més a prop per determinar quina és la cara objectiu del seguiment.

D'aquesta forma l'algorisme només detectarà aquesta cara.

7 Conclusions

L'algorisme de Viola-Jones és capaç de funcionar en temps real. Tot i així, és bona idea aplicar un algorisme de seguiment diferent al de aplicar el Viola-Jones inicial repetidament, ja que estem desaprofitant la informació que ens ha ofert el frame anterior i podem reduïr el temps en que triga un frame fins a més de la meitat.

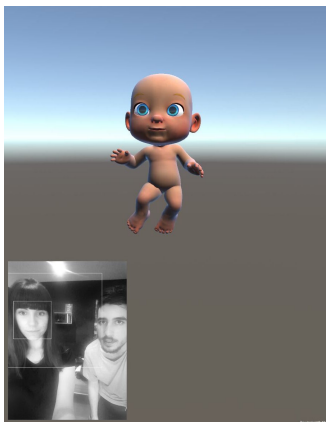

Fig. 14: Detecció de cara amb més d'una possibilitat

L'algorisme, en general i amb uns paràmetres ajustats manualment, triga aproximadament uns 13 m, complint amb l'objectiu de temps, tot i que presenta pics de 30 ms per frame quan s'ha de buscar de nou en tota la imatge. Això farà que, quan l'escena real estigui ben montada, s'hagi d'analitzar ben bé si serà viable o si, fora d'aquest TFG, s'haurà de buscar alguna alternativa. Per tant, de moment podem dir que produeix una bona sensació, que compleix amb els requisits però que quedarà en suspens fins que es puguin fer proves més exhaustives. De moment, no podem extreure més conclusions en aquest àmbit.

L'integració d'OpenCV a Unity3D no és un problema de fàcil solució. Els Wrappers estan orientats a la integració d'OpenCV a un sistema .Net però no al propi Unity3D. Això dificulta molt la tasca per la política estricte que presenta Unity3D. És per això que ja hi han paquets preparats a la Asset Store [23] que acostumen a ser de pagament.

El mòdul té una sèrie de paràmetres que encara no están provistos d'especificacions i que han sigut ajustats manualment per a la realització d'aquest projecte. Per a que el mòdul sigui més genèric, caldrà que, o s'adjustin uns paràmetres fixes o que es facin dinàmics o depenents del dispositiu que utilitzi l'aplicació. Alguns d'aquests paràmetres són:

- L'atribut far del gameobject objectiu. Aquest atrbitu es troba a la classe FaceTracking. S'ha de calibrar aquest paràmetre tenint en compte la distància a la que es posarà l'usuari davant de la càmera. És a dir, els usuaris, quan utilitzen una web cam en un ordinador, aquesta està posada a més distància que quan s'utilitza la càmera del mòbil. Això afecta directament a la posició on a de mirar l'objecte per crear una sensació real d'observació. Quan més a prop estigui l'usuari de la càmera, el gameobject objectiu ha de estar més aprop del personatje de l'escena.

- El factor multiplicatiu per crear l'àrea de recerca. Aquest factor representa com ha de ser de gran l'àrea de recerca respecte a la cara detectada. El tamany de l'àrea de recerca té una doble importància:

  - Per una banda, te relació amb el rendiment. Quan

més gran, permetrà que l'algorisme sigui més tolerant al moviment però més costos computacionalment. S'-ha de buscar l'equilibri, sempre tenint en compte que no pot ser massa sensible al moviment e intentar que estigui per sota dels 16 ms per frame.

- També afecta al funcionament i a l'efecte proporcionat ja que quan més petita sigui l'àrea de recerca, menys probabilitat hi ha de que una segona cara sigui detectada per l'algorisme.

Aquest factor es passat per paràmetre a l'hora de crear un Objecte SearchRect.

- La grandaria mínima i màxima del rectangle del algorisme de Viola-Jones. Aquests límits especifiquen el tamany mínim i màxim que pot tenir una cara detectada. Va relacionat amb la distància en la que l'usuari es posarà davant de la càmera. Com ja ha sigut explicat amb anterioritat, les distàncies a les que l'usuari es posa davant de la càmera son diferents per cada plataforma, per això serà bona idea fer aquests límits depenents d'on s'estigui executant l'aplicació. Amb aquests límits, podem determinar a quina distància es deixa de detectar la cara tant si es troba massa a prop, com si es troba massa lluny.

7.1 Línies de continuació

Un cop finalitzat el seguiment de cares i aprofitant que tenim OpenCV integrat, aquest mòdul ha de continuar oferint noves funcionalitats. Fins ara, les establertes per a continuar investigant són:

- El reconeixement facial. Dotar al personatje de la intel·ligència suficient per a que sigui capaç d'identificar algunes persones ja presentades amb anterioritat.

- El reconeixement d'expressions. Aquesta, cada vegada més, està guanyant importància i pes. El que es pretén és que el personatge reaccioni amb diferents emocions a certes expressions concretes.

AGRAÏMENTS

A tot l'equip de BIGGames per la col·laboració en aquest projecte, l'aportació de noves idees i per donar-me la oportunitat de realitzar-lo amb ells.

A vosaltres tres, ja sabeu qui sóu. Per la vostra companyia i ajuda quan més ho he necessitat i, sobretot, per la vostra gran amistat.

A la familia, sempre esteu allà quan cal i on cal.

REFERÈNCIES

[1] Unity3D, pàgina oficial `http://unity3d.com`

[2] ENIAC, entrada de la viquipèdia `https://ca.wikipedia.org/wiki/ENIAC`

[3] Eyetoy, entrada de la viquipèdia `https://ca.wikipedia.org/wiki/EyeToy`

[4] Kinect, entrada en la viquipèdia `https://ca.wikipedia.org/wiki/Kinect`

[5] Wiimote o wiiremote, entrada en la viquipèdia `https://ca.wikipedia.org/wiki/Wii_Remote`

[6] Playstation Move, entrada en la viquipèdia `https://en.wikipedia.org/wiki/PlayStation_Move`

[7] Playstation eye, entrada en la viquipèdia `https://en.wikipedia.org/wiki/PlayStation_Eye`

[8] Playstation Camera, entrada en la viquipèdia `https://en.wikipedia.org/wiki/PlayStation_Camera`

[9] OpenCV, pàgina oficial `http://opencv.org/`

[10] Mono, pàgina oficial `http://www.mono-project.com/`

[11] EmguCV, pàgina oficial `http://www.emgu.com/wiki/index.php/Main_Page`

[12] OpenCVSharp, repositori associat al wrapper `https://github.com/shimat/opencvsharp`

[13] Opencv .NET, repositori associat al wrapper `https://bitbucket.org/horizongir/opencv.net`

[14] Viola-Jones, Rapid Object Detection using a Boosted Cascade of Simple Features, 2001 `https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf`

[15] Adaboost, entrada en la viquipèdia `https://en.wikipedia.org/wiki/AdaBoost`

[16] CAMShift, Object Tracking Using CamShift Algorithm and Multiple Quantized Feature Spaces, John G. Allen, Richard Y. D. Xu, Jesse S. Jin, 2006, `http://crpit.com/confpapers/CRPITV36Allen.pdf`

[17] Meanshitf, entrada en la viquipèdia `https://ca.wikipedia.org/wiki/Mean-Shift`

[18] Scrum, entrada en la viquipèdia `https://ca.wikipedia.org/wiki/Scrum`

[19] Plugins, Manual de Unity3D, Plugins, `http://docs.unity3d.com/Manual/Plugins.html`

[20] Mat, Documentació OpenCV, Estructures bàsiques, `http://docs.opencv.org/2.4/modules/core/doc/basic_structures.html`

[21] MonoBehaviour, Documentació Unity, Script Reference, MonoBehaviour,
http://docs.unity3d.com/ScriptReference/MonoBehaviour.html

[22] HaarClassifiersCascade, Documentació Opencv, Cascade Classification,
http://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html

[23] Asset Store, tenda de paquets per a Unity3D
https://www.assetstore.unity3d.com/en/

[24] Buildings Plugins for Android, Android Library Projects, Documentació Unity,
http://docs.unity3d.com/Manual/PluginsForAndroid.html

[25] Robot Inverse Kinematics,Learn about robots, autor desconegut,
http://www.learnaboutrobots.com/inverseKinematics.htm

[26] Final IK, entrada a l'Asset store,
https://www.assetstore.unity3d.com/en/#!/content/14290

[27] The Profiler Window, Documentació Unity,
http://docs.unity3d.com/Manual/Profiler.html

[28] Fernando Bevilacqua, 2013, Finite-State Machines: Theory and Implementation,
http://gamedevelopment.tutsplus.com/tutorials/finite-state-machines-theory-and-\implementation--gamedev-11867