ESCOLA POLITÈCNICA SUPERIOR
UNIVERSITAT DE LES ILLES BALEARS

PROJECTE FINAL DE CARRERA

Estudis:

ENGINYERIA INFORMÀTICA

Títol

REHABILITACIÓ COMPLEMENTÀRIA BASADA EN VIDEOJOCS UTILITZANT LEAP MOTION

Autor: Marc Ferrer Fontirroig
Directora: Cristina S. Manresa Yee

Data: 9 de juny de 2017

Índex

1 Introducció **7**
- 1.1 Motivació . . . 7
- 1.2 Objectius . . . 8
- 1.3 Estructura del document . . . 9

2 Context **9**
- 2.1 Interfícies basades en visió . . . 9
- 2.2 Rehabilitació de la mà . . . 12
- 2.3 Interfícies basades en visió per a rehabilitació . . . 14

3 Anàlisi i disseny **15**
- 3.1 Introducció . . . 15
- 3.2 Leap Motion . . . 15
  - 3.2.1 Hardware: Leap motion . . . 15
  - 3.2.2 Arquitectura del Leap Motion . . . 16
  - 3.2.3 Leap Motion API . . . 19
- 3.3 Requeriments . . . 23
- 3.4 Arquitectura del sistema . . . 24
- 3.5 Disseny dels jocs . . . 25
  - 3.5.1 Runner boy - Exercici d'extensió del canell . . . 25
  - 3.5.2 Cubes road - Exercici d'abducció i adducció del canell . 27
  - 3.5.3 Catch stars - Exercici d'abducció i adducció del dits . . 29
- 3.6 Aplicació de monitoratge . . . 30

4 Implementació **32**
- 4.1 Tecnologies utilitzades . . . 32
- 4.2 Detalls d'implementació . . . 34
  - 4.2.1 Servidor web . . . 34
  - 4.2.2 Servidor socket . . . 35
  - 4.2.3 Runner Boy . . . 38
  - 4.2.4 Cubes Road . . . 41
  - 4.2.5 Catch Stars . . . 46
  - 4.2.6 Network plugin . . . 52
  - 4.2.7 Aplicació de monitoratge . . . 55

5 Futur **58**

6 Conclusions **59**

7 Referències **59**

Índex de figures

1 Exercicis de flexo-extensió de la mà. 12
2 Exercicis d'extensió dels dits. 13
3 Exercicis d'oposició del polze. 13
4 Exercicis de mobilitat lateral de la mà. 14
5 Representació del dispositiu i el seu camp de visió [2]. 16
6 Esquema d'aplicacions utilitzant la interfície nativa. 17
7 Esquema d'aplicacions utilitzant la interfície de web socket. . . 18
8 Sistema de coordenades del Leap Motion. 20
9 Representació d'una mà i els vectors palmNormal i direction. . 21
10 Representació d'una mà i els vectors de direcció dels dits. . . . 22
11 Representació dels diferents ossos de la classe Hand. 22
12 Imatge del jugador saltant per capturar una estrella. 26
13 Imatge del jugador esquivant un obstacle durant la partida. . . 27
14 Situació normal del joc. 28
15 Imatge del jugador després d'haver col·lisionat amb varis enemics. 29
16 Imatge que mostra els diferents components del joc. 30
17 Arquitectura de l'aplicació de monitorització. 31
18 Visor de l'aplicació de monitoratge. 32
19 Codi bàsic d'un servidor Express.js. 35
20 Estructura del fitxer de sessió de joc. 35
21 Sales del servidor de socket 36
22 Funció principal del servidor de socket 37
23 funció que atén les dades de seguiment 38
24 Event de desconnexió 38
25 Inici del gest. 40
26 Fi del gest. 40
27 Bucle principal del joc. 42
28 Inicialització de l'objecte *picker* 44
29 Funció que atén les col·lisions dels cubs. 45
30 Mètode d'inici del joc. 47
31 Bucle principal 48
32 Funció d'actualització principal. 49
33 Funció d'actualització principal. 50
34 Mètode *update* de la classe *Group* 52
35 *Network plugin* 53
36 Estructura de dades del *plugin*. 55
37 Configuració dels *plugins* per a l'aplicació. 56

38 Actualització de la barra de reproducció. 58

Cristina S. Manresa Yee, faig constar que he dirigit el Projecte de Final de Carrera titulat “Rehabilitació complementària basada en videojocs utilitzant Leap Motion” realitzat per l’alumne Marc Ferrer Fontirroig i que el treball està acabat i preparat per a la seva presentació pública.

Palma, dia de mes i any

Signat: Cristina S. Manresa Yee

Resum

El projecte que es presenta descriu el desenvolupament d'una prova de concepte per a la utilització d'interfícies basades en visió per a la realització d'exercicis de rehabilitació física, que pugui servir com a teràpia complementària a la teràpia convencional amb un professional com pugui ser un fisioterapeuta.

El projecte consisteix en el desenvolupament de tres videojocs web que permetin, mitjançant el dispositiu *Leap Motion Controller* la realització de diversos exercicis de les articulacions de la mà i els dits. També es presenta una aplicació per tal que els responsables de la teràpia puguin monitorar els exercicis realitzats pels usuaris.

Durant la realització del projecte s'ha tengut molt present en tot moment que l'objectiu és una correcte realització dels exercicis que permeti als usuaris millorar amb la seva rehabilitació.

1 Introducció

Aquest projecte final de carrera s'emmarca en el projecte de cooperació universitària al desenvolupament (OCDS-CUD2016/13) 'Disseny d'experiències interactives dirigides al benestar de persones amb necessitats especials' finançat per la Oficina de Cooperació al Desenvolupament i Solidaritat de la Universitat de les Illes Balears. Entre els objectius del projecte, hi ha el disseny i desenvolupament d'experiències interactives per salut per a persones amb discapacitat com per exemple, aplicacions per a rehabilitació física.

En aquest aspecte, pot esser molt important la participació activa d'aquestes persones en els exercicis de rehabilitació. La utilització d'exercicis basats en videojocs pot tenir un efecte motivant sobre aquestes persones. A més un bon disseny d'aquests videojocs, fa que no sigui necessària la supervisió constant d'un professional a l'hora de la realització dels exercicis. Per això però, no ens podem recolzar en les interfícies dels sistemes interactius tradicionals com són el teclat, el ratolí o les pantalles tàctils. Es necessiten interfícies que permetin un grau de llibertat major en els moviments dels usuaris. És aquí on cobren importància les interfícies basades en visió, sobretot amb l'aparició de sistemes comercials de molt bona qualitat.

Una interfície basada en visió (VBI de les inicials en anglès) és una interfície que utilitza la informació visual com entrada al sistema interactiu en un context d'Interacció Persona Ordinador. Les VBI perceben a l'usuari i les seves accions a través d'una càmera, i l'anàlisi i el reconeixement del moviment humà i els gestos en temps real, pot ser molt útil en una àmplia gamma d'aplicacions, des d'interacció amb videojocs fins a la navegació en mons virtuals. Avui en dia, l'aparició de càmeres integrades a dispositius mòbils o ordinadors, i la gran potència i el baix cost de sistemes comercials que utilitzen gestos del cos per interactuar (especialment amb videojocs) com són *Microsoft Kinect*, *Leap Motion*, *Sony Move* o *Nintendo Wii*, fan que aquests sistemes adrecin nous camps com per exemple la rehabilitació.

1.1 Motivació

Com s'ha comentat, aquest projecte s'emmarca en el projecte de cooperació universitària al desenvolupament (OCDS-CUD2016/13), i té com a objectius la realització d'una sèrie d'experiències interactives per salut que permetin als usuaris la realització d'exercicis de rehabilitació física.

La motivació del projecte és la realització d'unes aplicacions que permetin als usuaris de teràpies de rehabilitació complementar els exercicis de la teràpia amb la realització d'exercicis de manera autònoma.

A títol personal el projecte m'aporta la motivació de poder treballar amb

una interfície basada en visió, amb diverses tecnologies per al desenvolupament de videojocs, i també, el poder experimentar amb tecnologies que permeten la realització d'aplicacions web amb temps real.

1.2 Objectius

L'objectiu principal d'aquest projecte fi de carrera, és realitzar una prova de concepte d'exercicis de rehabilitació de lesions de les articulacions de la mà (canell i dits) utilitzant interfícies basades en visió comercials.

La utilització d'exercicis basats en videojocs, aporta un context motivant extra que pot ajudar a augmentar la participació dels pacients en la seva teràpia de rehabilitació, tant en temps invertit, com quant a l'atenció a aquests exercicis. Això és un aspecte fonamental per a l'èxit de la rehabilitació.

La teràpia remota basada en videojocs presenta un desavantatge, i és la falta de supervisió per part del responsable de la rehabilitació, típicament un fisioterapeuta.

Per tal de suplir aquest desavantatge, un dels objectius d'aquesta prova de concepte és que el responsable de la teràpia sigui capaç de veure una reproducció dels moviments realitzats per l'usuari. D'aquesta manera es pot fer un seguiment de l'evolució de l'usuari de manera més ràpida. De la mateixa manera el responsable de la teràpia és capaç de detectar possibles errors en la realització dels exercicis de rehabilitació, que d'una altra manera podrien tenir un efecte perjudicial per l'usuari, i aquests es poden corregir de manera gairebé immediata.

En concret, es pretén utilitzar el controlador *Leap Motion*. El *Leap Motion*, és un dispositiu capaç de detectar i capturar les dades de les mans i els dits dins el seu camp de visió, sense necessitat d'utilitzar cap tipus de marca de referència. Tot i que no està específicament dissenyat per a rehabilitació, el seu preu reduït i les seves dimensions en comparació a altres dispositius de visió, fan que aquest sigui un dispositiu idoni per els tipus d'exercicis proposats.

Els subobjetius del projecte que es persegueixen per aconseguir l'objectiu principal són:

- Conèixer el funcionament de Leap Motion.
- Desenvolupar tres videojocs amb diferents tecnologies que treballin exercicis de rehabilitació de la mà i dits.
- Desenvolupar una aplicació de monitoratge perquè el terapeuta pugui controlar els moviments realitzats pel pacient.

1.3 Estructura del document

La memòria d'aquest projecte està estructurat de la següent forma:

Context en aquesta secció es descriuen els conceptes bàsics per contextualitzar el projecte: interfícies basades en visió, rehabilitació de la mà i exemples de sistemes interactius on s'ajunten ambdós conceptes.

Anàlisi i disseny en aquesta secció es descriu quin ha estat el sistema i les aplicacions desenvolupades. Es descriuen quines són les característiques del dispositiu Leap Motion, es fa una anàlisi dels requeriments del projecte i finalment es descriu detalladament el disseny de cada aplicació.

Implementació es presenten els detalls d'implementació del sistema i les aplicacions desenvolupades. Es fa un repàs de les tecnologies utilitzades en la realització del projecte i s'analitzen les causes de la seva elecció. Al final de la secció, es descriuen els detalls d'implementació de cada aplicació.

Futur en aquesta secció es descriuen una sèrie d'accions de futur una vegada acabat el projecte.

Conclusions s'exposa l'experiència obtinguda a títol personal i les conclusions obtingudes després de realitzar el projecte.

Bibliografia en aquesta secció s'exposa la documentació utilitzada per realitzar el projecte.

2 Context

2.1 Interfícies basades en visió

Una interfície basada en visió (VBI de les inicials en anglès) és una interfície que utilitza la informació visual com entrada al sistema interactiu en un context d'Interacció Persona Ordinador.

Les VBI perceben a l'usuari i les seves accions a través d'una càmera, i l'anàlisi i el reconeixement del moviment humà i els gestos en temps real, pot ser molt útil en una àmplia gamma d'aplicacions, des d'interacció amb videojocs fins a la navegació en mons virtuals. Avui en dia, l'aparició de càmeres integrades a dispositius mòbils o ordinadors, i la gran potència i el baix cost de sistemes comercials que utilitzen gestos del cos per interactuar (especialment amb videojocs) com són *Microsoft Kinect*, *Leap Motion*, *Sony Move* o *Nintendo Wii*, fan que aquests sistemes adrecin nous camps com

per exemple la rehabilitació. Quan s'empra la visió per computador per interactuar amb sistemes interactius, diferents dificultats poden aparèixer a causa de la il·luminació, la variabilitat de l'aparença humana o fons amb molts d'objectes o objectes en moviment que estan fora del nostre estudi.

És molt important extreure només la informació rellevant d'una informació visual sobrecarregada, tal com faria l'ull humà, per tal de concentrar tot l'esforç computacional en obtenir els resultats correctes de l'anàlisi.

Quan s'utilitzen els gestos o el moviment del cos per interactuar, les VBI, es centren en una sèrie de tasques, que tenen com a objectiu la detecció i seguiment o la detecció d'una part del cos, ja sigui la cara, les mans, els ulls o qualsevol part del cos. Definim primer aquests conceptes. La detecció d'una part del cos implica determinar una sortida binaria, que indica si la part del cos està present o no. Normalment la localització de la part del cos a la imatge també es dóna. El seguiment significa localitzar objectes i informar dels seus canvis de posició en el temps. El seguiment també pot ser entès com la detecció d'un objecte fotograma a fotograma. El reconeixement o la identificació implica comparar una imatge d'entrada amb un conjunt de models en una base de dades, que resulta en confiança, puntuacions i probabilitats, que defineixen en quina mesura encaixen les dades de la imatge en cada model. La detecció també s'anomena a vegades reconeixement, perquè si hi ha diferents objectes a una imatge, un d'ells ha de ser reconegut. Un cas especial de reconeixement, és la verificació o autenticació, que avalua si les dades d'entrada pertanyen a una identitat en concret, amb un nivell de confiança molt alt [13].

Es poden trobar VBI que necessiten comandaments (p.e. *Sony Move*, *Nintendo Wii*) i d'altres que funcionen directament amb el moviment del cos (p.e. *Microsoft Kinect*, *Asus Xtion* o *Leap Motion*). A continuació descrivim aquests sistemes de forma abreujada, ja que més endavant ens centrarem en la tecnologia utilitzada i les seves característiques.

Microsoft Kinect *Microsoft Kinect* és un dispositiu detector de moviment, desenvolupat per *Microsoft* per a la videoconsola *Xbox 360* i adaptat després a ordinadors. Permet als usuaris controlar interactuar amb la seva videoconsola o ordinador sense necessitat de cap controlador físic. Aquest dispositiu és capaç de detectar gestos, i comandes de veu gràcies al micròfon que incorpora. Té com a objectiu principal augmentar la base de jugadors de la videoconsola *Xbox* [4].

La primera versió del dispositiu consistia en una barra horitzontal d'aproximadament 23cm, que estava dissenyat per ser col·locat damunt la videoconsola. El dispositiu comptava amb una càmera RGB, un sensor

de profunditat i un micròfon de múltiples matrius. D'aquesta manera el dispositiu és capaç de oferir un model complet del cos en 3D, reconeixement facial i reconeixement de veu.

Asus Xtion *Asus xtion* és un sensor de moviment exclusiu per a ordinador, desenvolupat per l'empresa ASUS per tal d'oferir als desenvolupadors un perifèric de detecció de moviment i unes llibreries pel desenvolupament d'aplicacions.

El sensor consisteix en una càmera i un sensor de profunditat que li permeten detectar en temps real, diversos gestos i el seguiment del cos complet de l'usuari [1].

Leap Motion *Leap Motion* és el dispositiu utilitzat en aquest projecte. Creat per una empresa amb la que comparteix nom. Està centrat en capturar els moviments i gestos de les mans i els dits, utilitza dues càmeres i tres LEDs infraroigs per fer un seguiment dels moviments. Més endavant es descriurà en detall aquest dispositiu [6].

Nintendo Wii La *Nintento Wii* és una consola llançada per *Nintendo* a l'any 2006. *Nintendo* es va posar l'objectiu d'atreure al món de les videoconsoles al gran públic no aficionat als videojocs tradicionals [15]. Per tal d'aconseguir-ho, la principal característica d'aquesta videoconsola era la inclusió del controlador *Wii Remote*, capaç de detectar el moviment de l'usuari que l'utilitza. La videoconsola també és capaç de detectar cap a quin lloc de la pantalla apunta el controlador.

Per la detecció de moviment el controlador fa servir acceleròmetres, i per detectar cap on apunta, disposa d'un sensor òptic d'infraroigs i una barra de LEDs infraroigs que es col·loca típicament damunt la pantalla a la que es connecta la videoconsola.

Sony Move *Sony Move* és un sistema de control de videojocs per moviment. desenvolupat per *Sony Interactive Entertainment* per a ser utilitzat amb la seva videoconsola PS3, i posteriorment PS4.

Conceptualment similar al *Wii remote* de *Nintendo*, es basa en dos dispositius, una càmera (*PlayStation Eye* o *PlayStation Camera*), que és capaç de capturar els moviments d'un controlador en forma de vareta. Aquesta vareta té una esfera en un dels seus extrems, que s'il·lumina i s'utilitza com a marca per a detectar el moviment de l'usuari.

La càmera és capaç de posicionar l'usuari en tres dimensions utilitzant la mida de l'esfera lluminosa.

2.2 Rehabilitació de la mà

La rehabilitació és un conjunt de tècniques i mètodes que serveixen per recuperar una funció o activitat del cos que ha disminuït o s'ha perdut per alguna causa, com per exemple un accident o una malaltia.

En el cas de les mans, entre les patologies que poden requerir teràpia de la mà es poden trobar [3]:

- Fractures, lesions en els tendons, lesions per aixafament i amputació.
- Artritis.
- Post quirúrgiques en síndrome del túnel carpià, artroplàsties. Tenorràfia o transposicions tendinoses, exèresi de tumors i reconstrucció de defectes congènits.
- Lesions relacionades amb el treball (Epicondilitis, Tendinitis).
- Deformacions congènites, lesions neuropatològiques, neuropatia diabètica, lesions del plexe braquial, miopatia primària i distrofia muscular.

A causa de les immobilitzacions, patologies i traumatismes de la mà, els pacients en teràpia han de realitzar exercicis repetitius destinats a recuperar la mobilitat de la mà, canell i dits. Aquests exercicis es fan al centre de rehabilitació, però freqüentment es recomana la seva execució i repetició fora del centre.

Els exercicis poden incloure [5]:

Flexoextensió del canell assegut amb l'avantbraç recolzat damunt una superfície plana, i deixant la mà fora, dur la mà cap amunt tot el que es pugui i mantenir la posició 5-10 segons. Repetir el moviment invers, portant la mà cap avall.

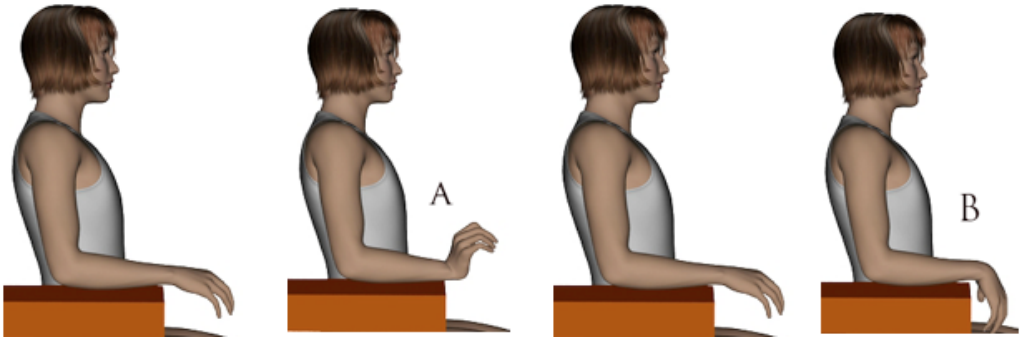

Figura 1: Exercicis de flexo-extensió de la mà.

Mobilitat dels dits amb la mà oberta, separar els dits en forma de ventall tot el possible i mantenir la posició 5-10 segons i tornar a la posició inicial de repòs.

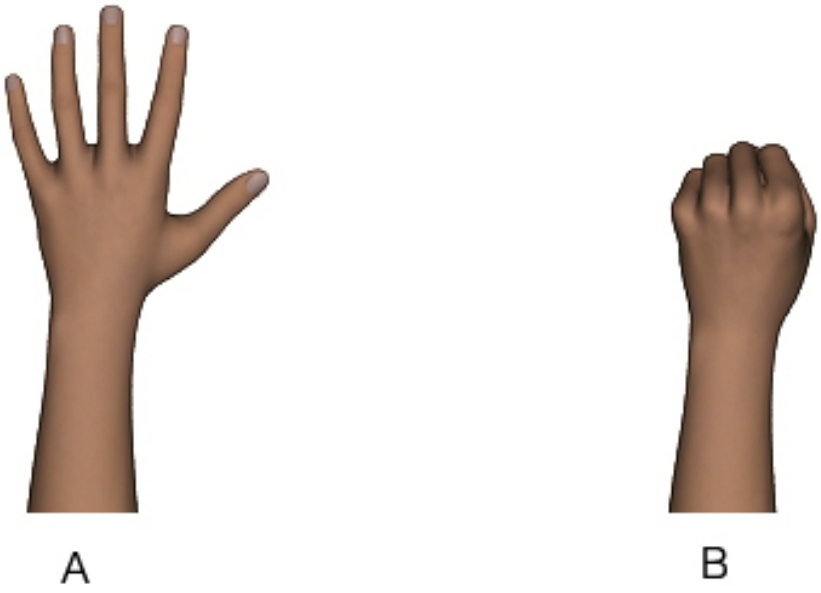

Figura 2: Exercicis d'extensió dels dits.

Oposició del polze dur el palpís del dit polze a la base de cada un dels altres dits començant per l'índex i acabant pel dit petit.

Figura 3: Exercicis d'oposició del polze.

Mobilitat lateral del canell amb la mà oberta, i els dits estirats, realitzar moviments amb el canell dirigint la mà primer cap a fora, mantenir 5-10 segons per a posteriorment relaxar tornant a la posició de repòs.

Continuar movent la mà cap a dins, mantenir uns altres 5-10 segons i tornar a la posició inicial.

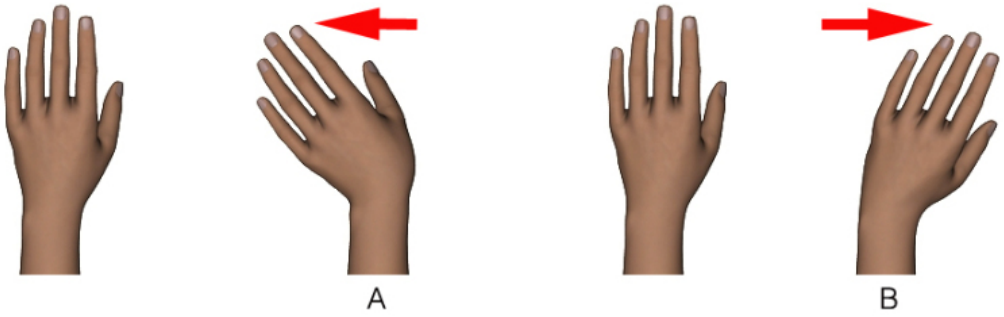

Figura 4: Exercicis de mobilitat lateral de la mà.

2.3 Interfícies basades en visió per a rehabilitació

En aquesta subsecció ens centrarem en el treball relacionat amb VBI que empren directament el moviment del cos sense la necessitat d'emprar comandaments. L'ús de *Microsoft Kinect* (o sistemes parescuts com l'*Asus Xtion*) s'han emprat per la rehabilitació de pacients amb accidents vasculars cerebrals, treball d'equilibri o que simplement necessiten recuperació física.

En concret, *Kinect*, presenta un grau d'utilització molt gran en aquests àmbits. *Webster i Celik* [17], presenten un recull de treballs que utilitzen *Kinect* per al cuidat de persones majors, per a la rehabilitació de pacients amb accidents vasculars cerebrals i exercicis basats en videojocs. Amb aplicacions tan diverses que van des de la detecció de caigudes en el cas de persones majors, fins aplicacions destinades a ajudar a que els exercicis de rehabilitació repetitius siguin més entretenguts. En el cas dels exercicis basats en videojocs, un dels treballs estudiats presenta un joc per treballar exercicis d'equilibri. En aquest joc, en el que jugador està representat per una figura humana virtual, uns contorns de figures humanes es movien cap al jugador virtual, que havia de imitar els contorns per tal de passar a través d'ells sense tocar-los.

En referència a la rehabilitació de la mà, Leap motion s'empra gràcies a la informació que ens pot donar dels dits i de la posició de la mà. Emprant aquest tecnologia, Liu et al (2015) [19] presenten un sistema on el metge pot prescriure al pacient certs moviments per imitar i aquest pot obtenir retroalimentació automàtica en forma de puntuació, d'acord amb la similitud. També es troba la utilització de videojocs per complementar la teràpia convencional (Iosa et al., 2015) [12]. Matos et al (2014) [14] presenten un joc per entrenar el rang d'obertura de la mà, l'usuari ha d'obrir la mà per

recollir pomes i transportar-les a una cistella mantenint l'obertura de la mà. Khademi et al. (2014) juguen al Fruit Ninja emprant la mà a rehabilitar per pacients amb accidents vascular cerebrals [8].

3 Anàlisi i disseny

3.1 Introducció

Dins aquest apartat s'analitza i descriu quin ha estat el sistema desenvolupat per tal de satisfer els objectius plantejats anteriorment.

En primer lloc es presenta una descripció de l'arquitectura i les característiques del controlador Leap Motion, que serà la interfície basada en visió que s'emprarà.

Posteriorment es fa una anàlisi de requisits del sistema a desenvolupar tant des del punt de vista funcional com tecnològic, la qual cosa permet tenir una descripció detallada del sistema.

Com a darrera secció es descriu el disseny de l'aplicació amb tots els seus subsistemes i els diferents jocs desenvolupats.

3.2 Leap Motion

3.2.1 Hardware: Leap motion

El *Leap motion* és un petit dispositiu que es connecta mitjançant USB i que és capaç de capturar els moviments de mans i dits, així com gestos o moviments de llapis (stylus) a través de dues càmeres i tres LEDs infraroigs. El dispositiu és capaç de capturar les dades de les mans fins a una distància d'uns 60 cm, tant a la part superior com als laterals del dispositiu, gràcies al gran camp de visió de les càmeres (uns 150°) [2].

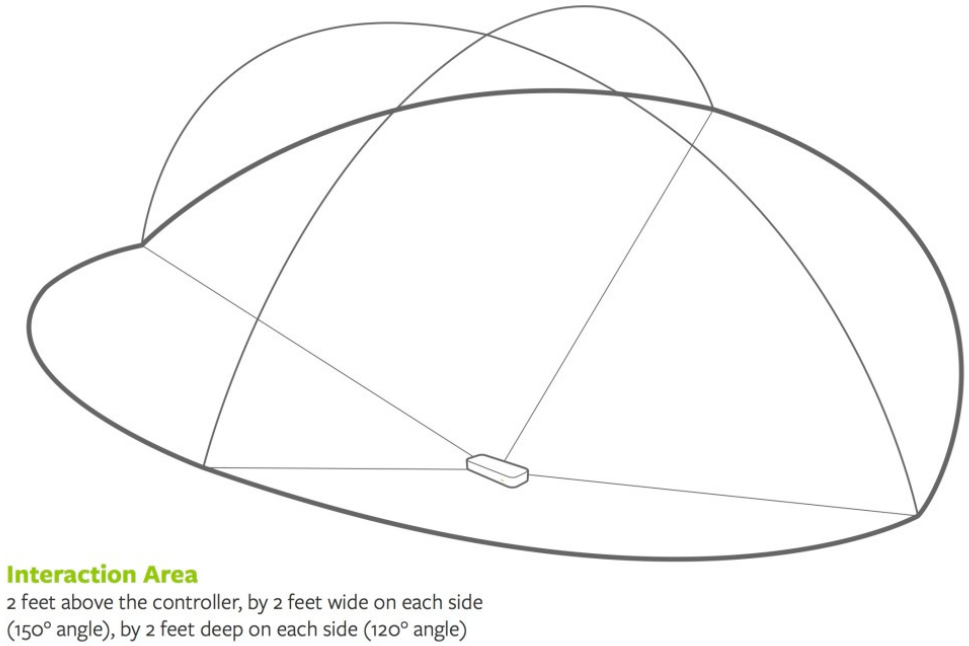

Figura 5: Representació del dispositiu i el seu camp de visió [2].

Les dades capturades per aquestes càmeres són enviades per USB a l'ordinador al qual es connecta el controlador Leap, on el software de Leap Motion, que s'executa com un servei, s'encarrega de processar-les i deixar-les disponibles perquè puguin ser obtingudes mitjançant APIs.

El servei del *Leap Motion* proporciona la informació a les APIs en forma d'una sèrie de fotogrames que contenen tota la informació de seguiment capturada. Al final és competència d'aquestes APIs proporcionar a l'usuari les dades en forma d'estructures orientades a objectes, així com també proporcionar una sèrie de funcions que permeten operar amb aquestes dades.

3.2.2 Arquitectura del Leap Motion

El *Leap Motion* és capaç de funcionar en múltiples sistemes operatius i les seves dades poden ser obtingudes de dues formes: mitjançant una interfície nativa o a través d'un servidor de *Web socket*. D'aquesta manera, les possibilitats per als desenvolupadors són molt diverses i inclouen molts dels llenguatges de programació més utilitzats actualment [9].

Interfície nativa La interfície nativa es proporciona a través d'una llibreria que connecta el servei del *Leap Motion*. Aquest servei després proveeix les

dades de seguiment obtingudes a les aplicacions desenvolupades. La llibreria es pot utilitzar directament en el cas d'aplicacions desenvolupades amb C++ i Objective C, o es poden utilitzar una de les APIs per Java, C# o *Python* [9].

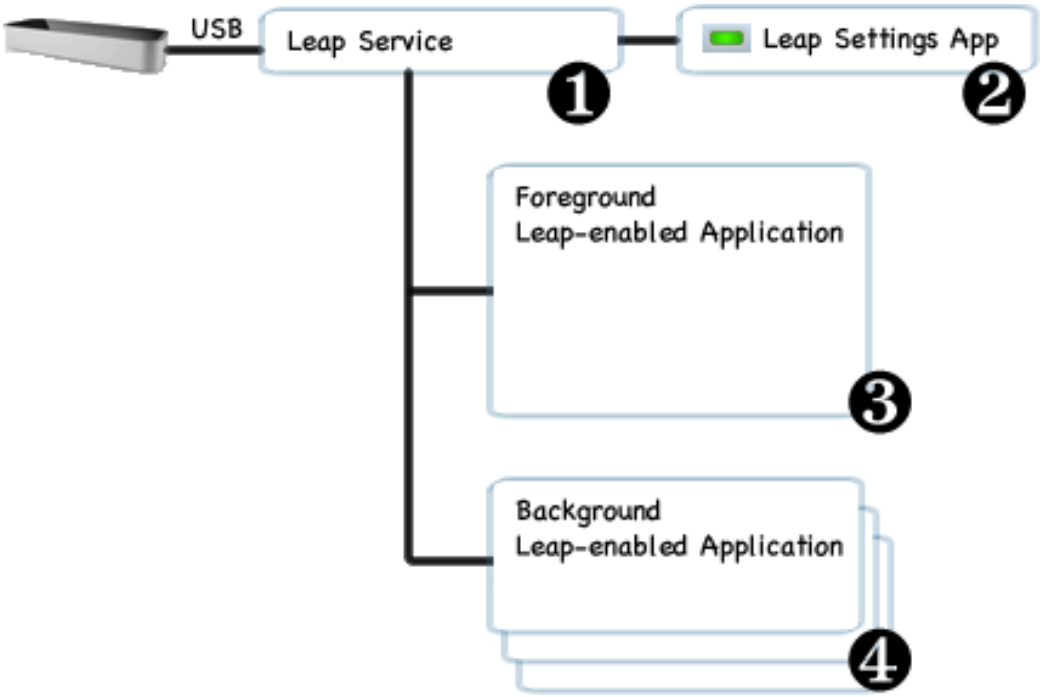

Figura 6: Esquema d'aplicacions utilitzant la interfície nativa.

A continuació es descriuen les diferents parts que intervenen en la connexió del dispositiu mitjançant la interfície nativa representada a la figura 6.

1. El servei de *Leap Motion* rep les dades del controlador a través d'una connexió USB, processa les dades rebudes i les envia a les aplicacions. Per defecte el servei només envia informació a les aplicacions en primer pla, però les aplicacions poden demanar rebre dades també en segon pla.

2. L'aplicació de *Leap Motion* és independent del servei i permet als usuaris configurar la seva instal·lació de *Leap Motion*.

3. Una aplicació en primer pla rep dades de seguiment del servei de *Leap Motion*.

4. Quan una aplicació perd el focus del sistema operatiu, el servei de *Leap Motion* deixa d'enviar-li dades. Les aplicacions dissenyades per funcionar en segon pla poden demanar al servei que els proveeixi dades de seguiment fins i tot en segon pla.

Interfície web socket El servei de Leap Motion executa un servidor de *web socket* local en el domini local (*localhost*) a través del port 6437. Aquesta interfície de web socket proveeix dades de seguiment en format JSON. Després un client *JavaScript* desenvolupat per la mateixa empresa de Leap Motion, s'encarrega de capturar aquestes dades i presentar-les com a objectes de *JavaScript* estàndard.

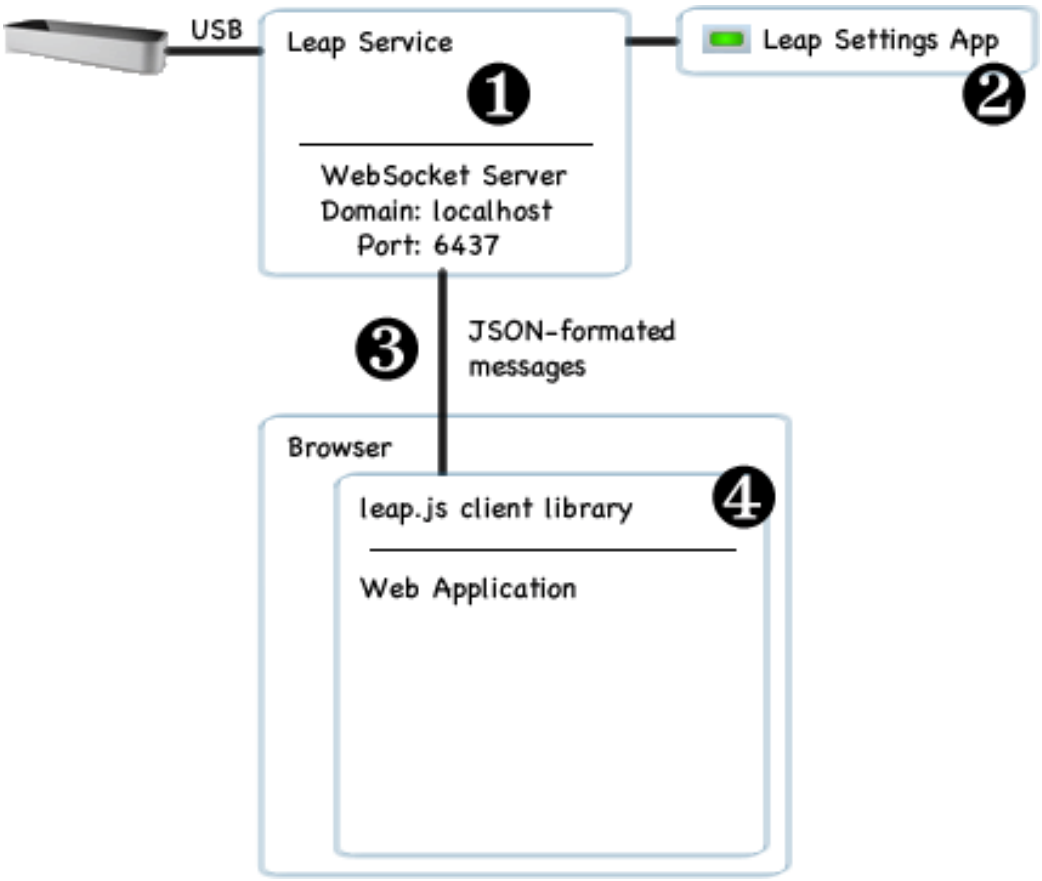

Figura 7: Esquema d'aplicacions utilitzant la interfície de web socket.

Tot seguit es descriuen les diferents parts que intervenen en la connexió del dispositiu mitjançant *web socket*, representat a la figura 7

1. El servei de *Leap Motion* proveeix un servidor de *web socket* a través de la direcció *http://127.0.0.1:6437*.

2. L'aplicació de configuració del *Leap Motion* permet habilitar o deshabilitar el servidor de *web socket*.

3. El servidor envia les dades de seguiment en format JSON. Les aplicacions també poden enviar missatges de configuració al servidor.

4. La mateixa empresa de *Leap Motion* ens proveeix d'una llibreria *JavaScript* que pot ser utilitzada per les aplicacions web per accedir a les dades de seguiment del *Leap Motion*.

Aquesta interfície està dissenyada per ser utilitzada per aplicacions web, tot i que qualsevol aplicació pot establir una connexió amb el servidor de web socket. Aquest servidor segueix l'estàndard de web socket RFC6455.

3.2.3 Leap Motion API

El sistema *Leap Motion* realitza un seguiment de les mans i els dits. El dispositiu ofereix una gran precisió i una alta taxa de fotogrames per segon, i retorna valors discrets de posició i moviment.

Els sensors es dirigeixen cap a l'eix Y, (cap amunt, quan el controlador es situa en la seva posició de funcionament estàndard) i com s'ha comentat anteriorment, ofereix un camp de visió d'uns 150°.

Sistema de coordenades El sistema *Leap Motion* utilitza un sistema de coordenades cartesianes. L'origen d'aquest sistema de coordenades es situa a la part superior del dispositiu. Els eixos X i Z es situen en el pla horitzontal, amb l'eix X paral·lel a l'eix més llarg del dispositiu. L'eix Y és vertical amb valors més positius cap amunt. L'eix Z té valors positius en direcció a l'usuari i valors negatius del dispositiu en direcció contraria a l'usuari.

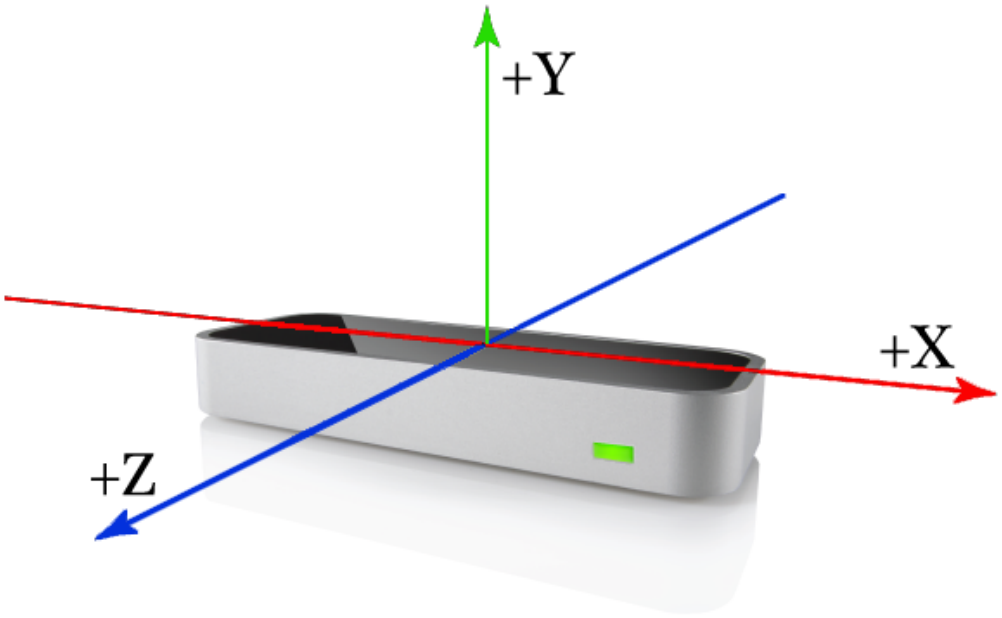

Figura 8: Sistema de coordenades del Leap Motion.

La API mesura les quantitats físiques en les següents unitats.

Distància	mil·límetres
Temps	microsegons
Velocitat	mil·límetres / segon
Angles	Radians

Mentre el dispositiu fa un seguiment de les mans i dits en el seu camp de visió, proveeix actualitzacions de les dades com un conjunt de fotogrames (*frames*) de dades. Cada objecte *frame* conté qualsevol mà detectada, detallant les seves propietats en un únic instant de temps. L'objecte *Frame* és essencialment, l'origen del model de dades retornat per la API. Aquesta posa a disposició dels desenvolupadors les següents classes.

Hands Les mans es representen mitjançant la classe *Hand* i proveeix informació sobre la identitat, posició i altres característiques de la mà detectada. Llista també tots els dits, així com també el braç al qual està lligada la mà.

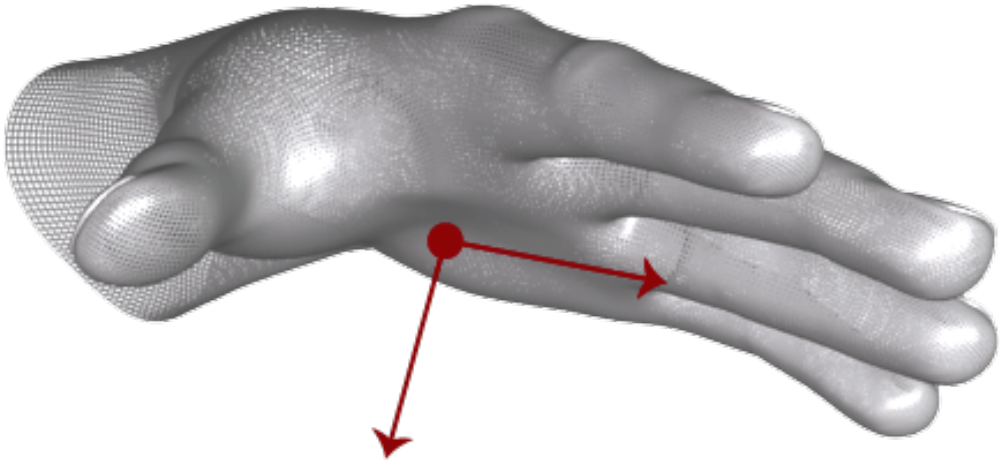

Figura 9: Representació d'una mà i els vectors palmNormal i direction.

El software del *Leap Motion* utilitza un model intern de la mà per tal d'oferir un seguiment predictiu, fins i tot quan parts de la mà no són visibles per els sensors. La classe *Hand* ofereix una propietat *Hand.confidence*, que indica en quin nivell d'exactitud encaixa en el model intern, la mà detectada.

Tot i que el dispositiu és capaç de detectar més de dues mans si més d'una persona està dins el camp de visió. Es recomana limitar les mans a dues per maximitzar la qualitat de les dades de seguiment.

Arms *Hand.arm* és un objecte que proporciona informació sobre l'orientació, longitud, amplada i punts finals d'un braç. Quan el colze està fora del camp de visió el sistema estima la posició del braç basant-se en observacions passades, així com amb les proporcions normals d'un braç humà.

Fingers Proporciona informació de cada dit de la mà. En cas que part d'un dit, o fins i tot tot el dit, no sigui visible, les característiques s'estimen en base a observacions recents i del model intern de la mà. Aquests dits s'identifiquen amb el seu nom en anglès (*thumb*, *index*, *middle*, *ring* i *pinky*).

Figura 10: Representació d'una mà i els vectors de direcció dels dits.

Els objectes del tipus *Finger* conté objectes de la classe *Bone* que descriuen la posició i orientació de cada os del dit. Tots els dits contenen 4 ossos, ordenats de la base a la punta dels dits.

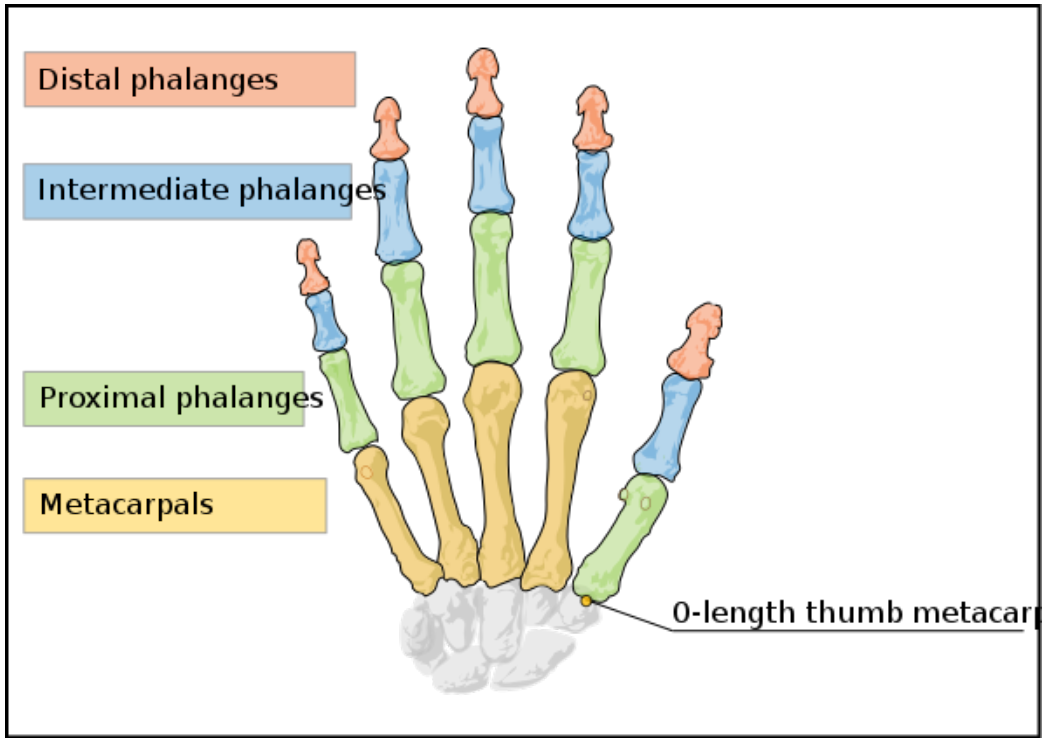

Figura 11: Representació dels diferents ossos de la classe Hand.

Els ossos s'identifiquen de la següent manera:

Metacarpal el *metacarpal* (metacarpià) és l'os situat dins la mà, que connecta el dit al canell (excepte el polze).

Proximal Phalanx *proximal phalanx* (primera falange) és l'os de la base del dit, connectat al palmell.

Intermediate Phalanx l'*intermediate phalanx* (segona falange) és l'os intermig del dit, situat entre la punta i la base del dit.

Distal Phalanx el *distal phalanx* (tercera falange) és el darrer os del dit, situat a la punta del dit.

Per mantenir una coherència, el model del polze no reflecteix el model anatòmic real. Un polze real té un os menys que la resta de dits, en canvi el model retornat per la API conté un metacarpià de longitud 0 per tal de tenir el mateix número d'ossos que la resta de dits.

3.3 Requeriments

Tot i que l'objectiu principal del projecte és realitzar una prova de concepte per confirmar que és possible utilitzar el controlador *Leap Motion* per complementar i monitorar teràpies de rehabilitació, és important establir una sèrie de requeriments tècnics i funcionals del projecte per tal de definir bé el seu abast.

Requeriments d'usuari

RU1 Un usuari ha de poder seleccionar un exercici a realitzar d'una llista.

RU2 Un usuari ha de poder realitzar exercicis d'extensió del canell.

RU3 Un usuari ha de poder realitzar exercicis d'abducció i adducció del canell.

RU4 Un usuari ha de poder realitzar exercicis d'abducció i adducció dels dits.

RU5 Un responsable de la rehabilitació d'un usuari ha de ser capaç de reproduir les sessions d'exercicis dels usuaris.

Requeriments funcionals dels jocs

RF1 Els jocs han de ser el més intuïtius possibles.

RF2 Els jocs han d'engrescar als usuaris a realitzar els exercicis.

RF3 Els jocs han d'enviar les dades de seguiment dels moviments.

RF4 Cada joc ha de permetre als usuaris realitzar un sol exercici.

Requeriments no funcionals

RNF1 Els jocs han de poder ser executats a qualsevol sistema operatiu.

RNF2 Els usuaris no han d'instal·lar cap aplicació addicional llevat dels controladors del dispositiu *Leap Motion*.

3.4 Arquitectura del sistema

Una vegada definits els requeriments del projecte, a continuació es presenta una descripció de l'arquitectura dissenyada per tal de satisfer aquests requeriments.

Un dels objectius ha estat des del primer moment que l'accés a l'aplicació fos el més fàcil possible per part dels usuaris. Per això es va decidir que el millor era que les aplicacions s'executassin en un entorn web.

Una aplicació web té l'avantatge de ser multiplataforma, és independent del sistema operatiu que tingui l'usuari el qual facilita l'accés a l'aplicació.

En el nostre cas s'utilitza una arquitectura web tradicional que consta d'un servidor web que és l'encarregat de servir les aplicacions de rehabilitació als usuaris. Aquestes aplicacions contenen la major part de la lògica de l'aplicació. L'arquitectura presenta una peculiaritat, fora del que és una aplicació web tradicional, que és la utilització d'un servidor de *web socket* que s'encarrega de rebre les dades del dispositiu *Leap Motion* dels usuaris i guardar aquestes dades en fitxers per a que puguin ser reproduïts posteriorment pels responsables de les teràpies de rehabilitació.

// grafic de l'arquitectura de l'aplicació

Aplicacions web de rehabilitació aquestes aplicacions són les encarregades de connectar-se al dispositiu *Leap Motion* dels usuaris per tal que aquests puguin realitzar els exercicis.

Servidor web un servidor web tradicional encarregat de servir les pàgines web als usuaris. També ha de ser l'encarregat de gestionar totes les dades dels usuaris per a que els responsables de la teràpia puguin fer un seguiment de les sessions de rehabilitació dels usuaris.

Servidor web socket és un servidor que habilita una comunicació bidireccional en temps real amb els usuaris. Aquest servidor s'encarrega de guardar les dades enviades pels dispositius *Leap Motion* dels usuaris per tal que aquestes puguin ser monitorades posteriorment.

3.5 Disseny dels jocs

Per tal de satisfer els requeriments plantejats anteriorment, s'han dissenyat tres jocs diferents.

Cada joc està dissenyat per treballar un tipus d'exercici determinat. Això implica que la complexitat del joc no pot ser molt elevada, ja que el control que pot arribar a tenir l'usuari sobre el joc, realitzant una mateixa acció repetidament és baix. A més, no s'ha d'oblidar que l'objectiu principal dels jocs no és el simple entreteniment de l'usuari, sinó que el més important és que l'usuari es senti motivat per realitzar els exercicis de rehabilitació i els realitzi de manera correcta.

A més s'ha desenvolupat una aplicació de monitoratge, que podria ser utilitzada pels responsables de de la rehabilitació per dur a terme un seguiment de l'evolució dels usuaris.

3.5.1 Runner boy - Exercici d'extensió del canell

Aquest joc, està específicament dissenyat per treballar el moviment d'extensió del canell. És un joc de desplaçament lateral infinit on l'usuari agafa el control d'una persona o jugador, que va corrent per l'escenari del joc com es mostra a la figura 12.

Figura 12: Imatge del jugador saltant per capturar una estrella.

L'objectiu és capturar unes estrelles que van apareixent en direcció contraria a l'usuari per obtenir punts. Per tal d'afegir complexitat al joc també apareixen uns obstacles que l'usuari ha d'esquivar per no perdre la partida. A més, a mesura que la puntuació de l'usuari es va incrementant, també ho fa la velocitat a la que es desplaça el joc, tant els obstacles com les estrelles de puntuació.

L'usuari controla el joc per mitjà del dispositiu Leap Motion. El primer té la capacitat de fer botar el jugador, per tal d'obtenir les estrelles que s'acosten per l'aire o d'esquivar els obstacles que apareixen per terra. El jugador realitza el moviment de salt quan l'usuari estén el canell, elevant els dits de la mà per sobre del canell tant com li sigui possible.13.

Figura 13: Imatge del jugador esquivant un obstacle durant la partida.

El jugador realitza el moviment de salt quan l'usuari estén el canell, elevant els dits de la mà per sobre del canell tant com li sigui possible.

3.5.2 Cubes road - Exercici d'abducció i adducció del canell

En aquest joc, els usuaris treballen els moviments d'abducció i adducció del canell de la mà. Es tracte d'un joc en 3 dimensions, en el que uns cubs es mouen a través d'una carretera en direcció a l'usuari, aquest ha d'intentar capturar-los desplaçant lateralment un altre cub.

L'objectiu del joc és capturar la major quantitat de cubs possibles, incrementant així la puntuació de l'usuari. A mesura que la puntuació de l'usuari augmenta, també ho fa la velocitat a la que es desplacen els objectes cap a ell augmentant així la dificultat del joc.

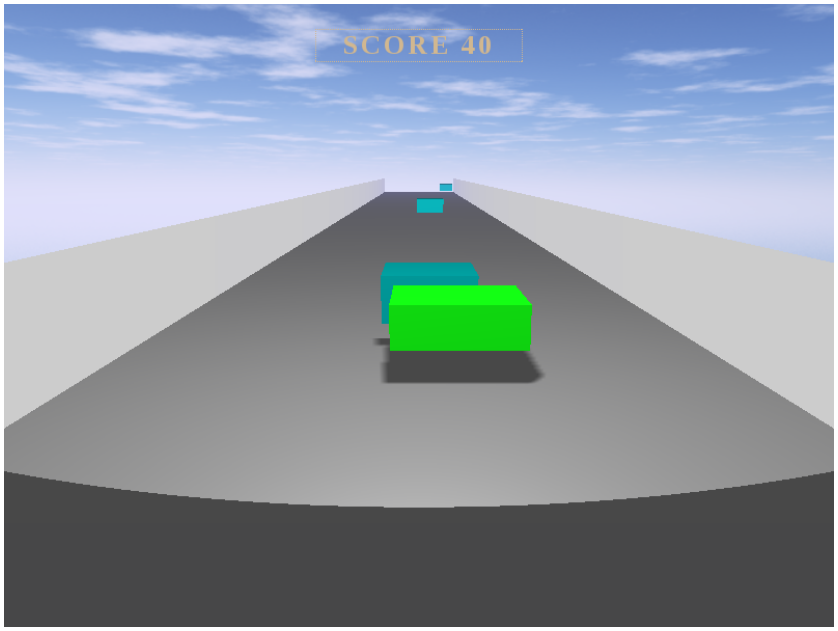

Figura 14: Situació normal del joc.

Amb l'objectiu d'augmentar la complexitat del joc i la motivació dels usuaris, periòdicament van apareixent uns cubs d'un color diferent que l'usuari haurà d'esquivar. En cas que l'usuari sigui incapaç d'esquivar algun d'aquests cubs que podríem denominar cubs enemics, la mida del cub que utilitza l'usuari per capturar els objectes es va reduint, d'aquesta manera com més errors comet l'usuari més difícil és per ell seguir amb la partida.

La figura següent mostra l'estat del jugador després d'haver capturat per error varis cubs enemics, així com l'aspecte d'un d'aquests cubs enemics.

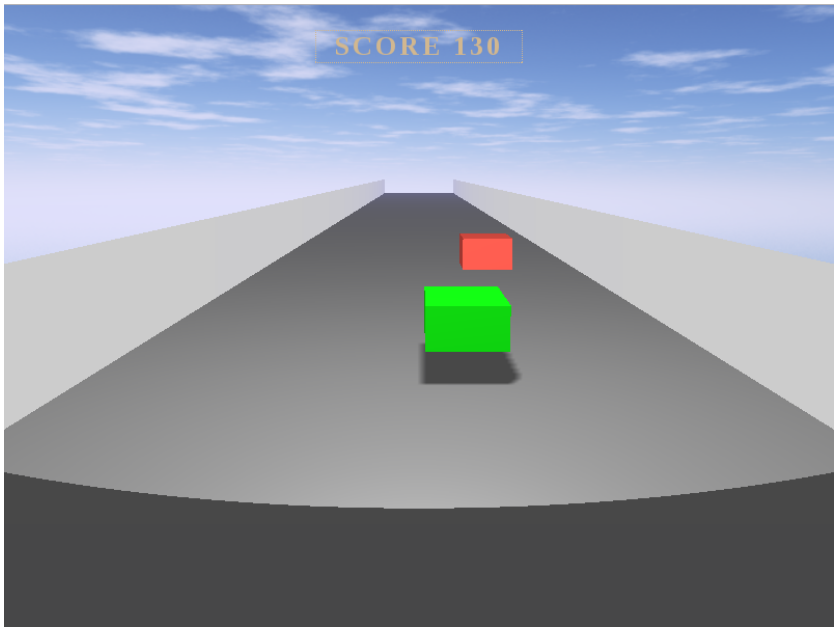

Figura 15: Imatge del jugador després d'haver col·lisionat amb varis enemics.

El moviment horitzontal del cub capturador (verd) es duu a terme mitjançant moviments d'abducció i adducció de la mà.

3.5.3 Catch stars - Exercici d'abducció i adducció del dits

Aquest darrer joc, ha estat dissenyat per treballar els exercicis d'abducció i adducció dels dits cor i anular. El moviment consisteix en separar els dits un de l'altre tant com sigui possible i tornar-los a ajuntar.

Es tracta d'un joc en 2D de desplaçament vertical, on l'usuari ha de capturar unes estrelles, que van caient, per augmentar la seva puntuació. Per capturar les estrelles, els usuaris han de desplaçar horitzontalment uns cubs vermells, que hi ha a la part inferior de la pantalla, per tal de fer-los col·lisionar amb aquestes estrelles.

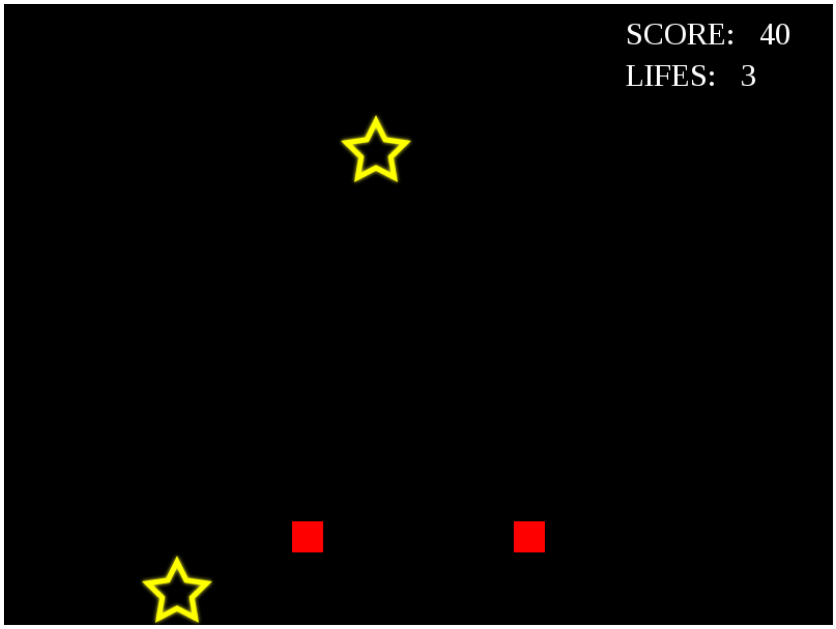

Figura 16: Imatge que mostra els diferents components del joc.

3.6 Aplicació de monitoratge

Aquesta aplicació permet als responsables de les teràpies de rehabilitació monitorar les sessions d'exercicis que fan els seus pacients. D'aquesta manera un responsable pot avaluar la millora d'un pacient. De la mateixa manera es poden detectar errors en la realització dels exercicis.

L'aplicació de monitoratge consta de 3 parts fonamentals, un *plugin* del controlador *Leap Motion*, un servidor de *web socket* i una aplicació de reproducció dels moviments capturats pel controlador Leap del pacient. La següent figura mostra un esquema de l'arquitectura de l'aplicació de monitoratge.

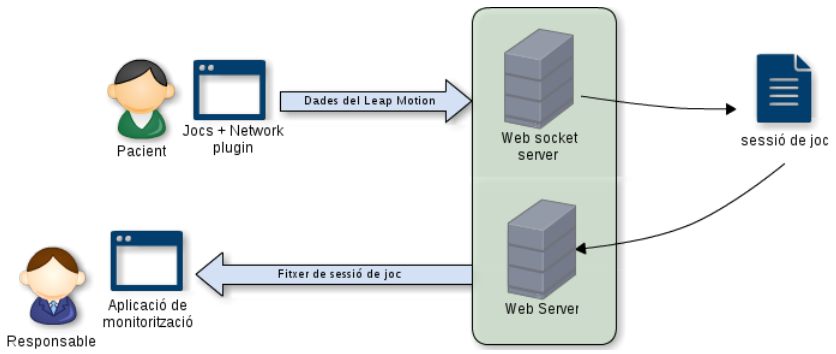

Figura 17: Arquitectura de l'aplicació de monitorització.

Així, un pacient es connecta a un dels jocs definits anteriorment, i mentre l'usuari juga, un plugin del Leap Motion va enviant les dades capturades a un servidor de web socket utilitzant una estructura determinada, a la vegada, aquest servidor s'encarrega de guardar les dades rebudes a un fitxer. Després en un altre moment, un dels responsables de la teràpia d'aquest pacient, es connecta a l'aplicació de monitoratge per tal de reproduir la sessió de joc del seu pacient. En aquest cas el servidor web l'únic que fa es servir les aplicacions als usuaris i el fitxer de la sessió de joc cap a l'aplicació de monitoratge.

Network plugin aquest és un *plugin* que segueix la mateixa filosofia que els plugins ja disponibles pel *Leap Motion* [10]. El primer que fa el *plugin* és connectar-se a un servidor de web socket. Després quan el controlador Leap Motion comença a enviar dades, aquest plugin les captura i les envia al servidor de *socket* utilitzant una estructura determinada.

Web socket un servidor de *web socket* tradicional que accepta connexions del plugin del *Leap Motion* i escolta els esdeveniments d'aquestes connexions. Quan rep una connexió nova, crea un fitxer de text i una vegada creat el fitxer de text, va escrivint les dades rebudes a través del *socket* al fitxer de text en format JSON. Una vegada que el plugin tanca la connexió amb el servidor, aquest darrer rep un esdeveniment de desconnexió i deixa el fitxer disponible per a ser reproduït per l'aplicació de monitoratge.

Visor de monitoratge és una aplicació web que es connecta al controlador *Leap Motion*, i utilitza les dades del fitxer, creat pel servidor de socket, per reproduir els moviments de la mà que ha realitzat l'usuari del joc.

Com si es tractàs d'un fitxer de vídeo, aquesta aplicació permet pausar la reproducció dels moviments, així com moure's endavant i enrere. Per

tal de facilitar la revisió dels moviments, els usuaris d'aquesta aplicació també tenen la capacitat de rotar la mà que es veu a la reproducció i modificar el zoom de la imatge.

La següent imatge mostra com veuen els responsables de les teràpies, la reproducció dels exercicis realitzats pels seus pacients.

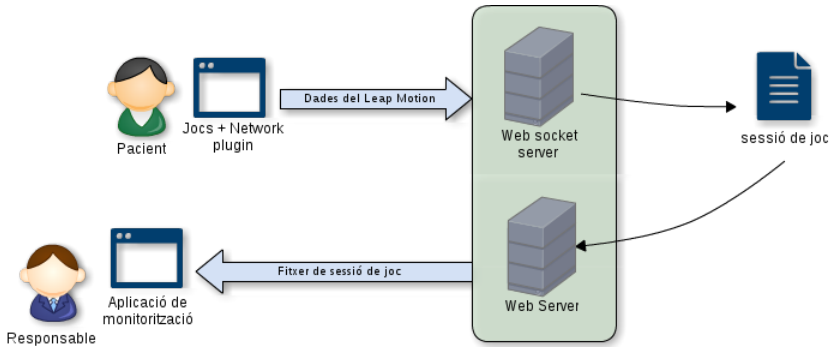

Figura 18: Visor de l'aplicació de monitoratge.

4 Implementació

En aquest apartat es descriuen els detalls d'implementació de les diferents aplicacions i sistemes desenvolupats.

Primer, es presenten les tecnologies utilitzades per desenvolupar les aplicacions i les raons per les quals s'han triat aquestes tecnologies, i després es detallen els aspectes més interessants de la implementació de les diferents aplicacions.

4.1 Tecnologies utilitzades

Durant la implementació de les aplicacions i sistemes dissenyats s'han utilitzat les següents tecnologies i llenguatges de programació.

HTML, CSS són la base de l'estructura i el disseny de qualsevol aplicació web.

JavaScript és el llenguatge de programació utilitzat pels navegadors web.

Node.js és un entorn multiplataforma que permet executar codi JavaScript a la banda del servidor. Presenta una arquitectura basada en esdeveniments i que permet l'execució d'operacions d'entrada i sortida de

manera asíncrona. Aquestes prestacions tenen l'objectiu de maximitzar la productivitat d'aplicacions amb múltiples operacions d'entrada i sortida, així com facilitar la implementació d'aplicacions web de temps real [18].
Aquestes característiques fan que aquesta tecnologia sigui una de les més adequades per a la implementació del servidor de web socket, ja que aquesta rebrà moltes peticions i haurà de realitzar operacions d'escriptura per a cada petició.

Per altra banda s'han utilitzat diverses llibreries que han ajudat a implementar les diferents aplicacions, en especial a l'hora la programació dels videojocs.

Express.js és un *framework* per al desenvolupament de servidors web amb *Node.js*.

Socket.io és una llibreria per a *JavaScript* per a la implementació d'aplicacions web de temps real. Consta de dues parts, una llibreria client que s'executa al navegador i una llibreria per al servidor (*Node.js*). Proporciona un gran ventall de prestacions al voltant de *web socket*.

Phaser.io és una llibreria per al desenvolupament de videojocs *HTML5* en 2d.

Three.js és una llibreria *JavaScript* que permet la creació de gràfics en 3 dimensions per a navegadors web.

Physijs és un *plugin* per a *Three.js* que facilita la gestió de càlculs físics. Afegeix prestacions com gravetat, gestió de col·lisions, rebots, etc, a una escena de *Three.js*.

TweenMax és una llibreria per a la gestió d'animacions complexes en *Javascript*.

Leapjs Plugin és un *plugin* per al *software* de *Leap Motion* per a *JavaScript* que permet la reproducció dels fotogrames capturats pel controlador *Leap Motion*. Utilitzat en la implementació de l'aplicació de monitoratge.

Webpack és un agregador de mòduls per a *JavaScript*. La seva funció és recopilar els mòduls que utilitza una aplicació i generar un sol fitxer estàtic que representa aquells mòduls. Això ens permet mantenir un enfocament modular a l'hora de desenvolupar aplicacions web.

4.2 Detalls d'implementació

Per al desenvolupament del projecte i la consecució dels objectius prevists, s'han implementat els següents sistemes i aplicacions.

Servidor web encarregat de servir les aplicacions web i els fitxers estàtics necessaris per al funcionament d'aquestes.

Servidor web socket Utilitzat per guardar les dades de seguiment dels dispositius *Leap Motion* dels usuaris, per tal que puguin ser reproduïts posteriorment quan es desitgi.

Runner boy videojoc *HTML5* desenvolupat utilitzant la llibreria *Phaser.io*. Està dedicat a exercitar el moviment de flexió del canell.

Cubes Road videojoc *HTML5* en 3D desenvolupat utilitzant la llibreria *Three.js*. Té com a objectiu exercitar els moviments d'abducció i adducció del canell.

Catch Stars videojoc *HTML5* desenvolupat únicament amb l'element *HTML5 canvas*. L'objectiu principal d'aquest joc és exercitar els moviments d'abducció i adducció dels dits cor i anular. D'altra banda un altre objectiu d'implementar el joc utilitzant només la API dels elements canvas, ha estat comparar com és la creació d'un videojoc utilizant APIs bàsiques com les de *canvas*, en vers a la utilització de llibreries com *Phaser.io* i *Three.js* que faciliten molts processos.

LeapJS Network plugin plugin propi desenvolupat seguint la filosofia dels *plugins* ja disponibles per *Leapjs*. La seva funció és enviar les dades de seguiment del controlador *Leap Motion*, a un servidor de *web socket*.

Aplicació de monitoratge una aplicació web que utilitza el *plugin leapjs-playback* per reproduir les dades de seguiment del *Leap Motion*. Per tal de reproduir els moviments, utilitza el fitxer guardat pel servidor de *web socket*.

A continuació es descriuen de manera detallada els aspectes més interessants de la implementació d'aquests sistemes i aplicacions.

4.2.1 Servidor web

És tracta d'un servidor web bàsic, implementa les funcionalitats necessàries per a donar servei als diferents videojocs desenvolupats.

S'ha desenvolupat utilitzant el *framework Express.js* ja que aquest simplifica moltes de les configuracions que necessita una servidor web, és pot implementar un servidor web senzill amb un parell de línies de codi.

```
1   const express = require('express');
2   const app = express();
3
4   const PORT = 3000;
5   app.get('/greeting', (req, res) => {
6     res.send('Hello World!');
7   })
8
9   app.use(
10    express.static(path.resolve(__dirname, 'games')));
11
12  app.listen(PORT, () => {
13    console.log(`App listening on port ${PORT}!`);
14  });
```

Figura 19: Codi bàsic d'un servidor Express.js.

4.2.2 Servidor socket

Aquest servidor, desenvolupat mitjançant la llibreria *Socket.io*, s'encarrega de rebre les connexions del *plugin* de *Leap Motion* i generar els fitxers de les sessions de joc.

El seu paper més important és generar el fitxer de les sessions de joc utilitzant l'estructura adecuada, que després podrà ser reproduïda utilitzant el plugin de reproducció *leapjs-playback*. A continuació es mostra quina és l'estructura de dades que ha de guardar el servidor de *web socket* en el fitxer.

```
1   {
2     metadata: {
3       formatVersion: 2,
4       generatedBy: 'Socket.io saver',
5       frames: 0,
6       protocolVersion: 6,
7       frameRate: '1.1e+2',
8       modified: new Date()
9     },
10    frames: [packingStructure]
11  }
```

Figura 20: Estructura del fitxer de sessió de joc.

Les meta dades vénen determinades pel *plugin* de reproducció y la majoria són constants o són regenerades a l'hora de reproducció del *plugin*, com és el cas del *frameRate* (ràtio de fotogrames).

Les dades de seguiment del Leap es guarden en forma de *Array* de fotogrames a la propietat *frames*, amb l'única peculiaritat que el primer element de la *Array* descriu l'estructura amb què es guarda cada fotograma. Aquest estructura amb què es guarda cada fotograma serà descrita a l'apartat de *Network Plugin*, ja que és el plugin desenvolupat el que s'encarrega de generar les dades amb l'estructura adequada i enviar-les.

El Servidor utilitza el que *Socket.io* defineix com a *namespace* per tal d'atendre les connexions de cada un dels jocs. Un "namespace" es pot entendre com una sala en la qual els *sockets* es poden comunicar entre si, d'aquesta manera un servidor pot enviar missatges a tots els *sockets* connectats a una sola sala i només als d'aquella sala. Es defineix una sala per a cada videojoc, d'aquesta manera:

```
1  io.of('/catch-stars')
2    .on('connection', leapRecorder);
3  io.of('/runner-boy')
4    .on('connection', leapRecorder);
5  io.of('cubes-road')
6    .on('connection', leapRecorder);
```

Figura 21: Sales del servidor de socket

Com es pot veure totes les connexions a les diferents sales, són manegades per la mateixa funció *leaprecorder*. Tot i que pot parèixer innecessari crear una sala per a cada videojoc, d'aquesta manera és més fàcil generar fitxers amb noms diferents per a cada videojoc, a més el codi queda més estructurat de cara al futur. La funció *leapRecorder* rep com a paràmetre un objecte *Socket* i serà a través d'aquest que es rebran els esdeveniments enviats pel *plugin*.

```
function leapRecorder(socket){
  let fileName = socket.id.substring(socket.nsp.name.length
      + 1);
  fileName = socket.nsp.name.substring(1) + '-' + fileName
     + '.json';
  const ws = fs.createWriteStream(fileName);
  const fileData = {
    metadata: {
      formatVersion: 2,
      generatedBy: 'Socket.io saver',
      frames: 0,
      protocolVersion: 6,
      frameRate: '1.1e+2',
      modified: new Date()
    },
    frames: [packingStructure]
  }
  const str = JSON.stringify(fileData);
  ws.write(str.substring(0, str.length - 2));

  socket.on('disconnect', () => {
    ws.write(']}');
    ws.close();
  });

  socket.on('frameBuffer', data => {
    let buffer = JSON.stringify(data[0]);
    for (let i = 1; i < data.length; i++) {
      buffer = buffer + ',' + JSON.stringify(data[i]);
    }
    ws.write(',' + buffer);
  });
}
```

Figura 22: Funció principal del servidor de socket

Val la pena descriure breument el funcionament d'aquesta funció. Quan un client es connecta al servidor, es crea un fitxer amb l'estructura descrita anteriorment. El fitxer es guarda en format JSON (*JavaScript Object Notation*, en anglès). En concret, del text resultant de convertir l'objecte a JSON, es guarda tot llevat dels dos darrers caràcters.

```
ws.write(str.substring(0, str.length - 2));
```

Això serà el que permetrà anar afegint les dades rebudes periòdicament dins de l'array *frames*, sense haver de regenerar cada vegada l'objecte complet. Així, quan el *plugin* envia un missatge de dades del *Leap Motion*, aquestes són afegides directament al final del fitxer en format JSON.

```
1  socket.on('frameBuffer', data => {
2    let buffer = JSON.stringify(data[0]);
3    for (let i = 1; i < data.length; i++) {
4      buffer = buffer + ',' + JSON.stringify(data[i]);
5    }
6    ws.write(',' + buffer);
7  });
```

Figura 23: funció que atén les dades de seguiment

Com es pot veure, dins del missatge de dades del *plugin*, no arriba només un fotograma capturat pel *Leap Motion*, sinó que arriben múltiples fotogrames. Això s'ha fet així per tal d'optimitzar la utilització de xarxa del *plugin*, ja que el ràtio de fotogrames per segon produït pel controlador Leap Motion, pot ser molt gran.

Finalment, una vegada l'usuari tanca el joc, el servidor detecta la desconnexió de l'usuari i tanca el fitxer. Per tal que el fitxer contingui un JSON ben format, s'han d'afegir els dos caràcters que s'han omès a l'inici de la connexió.

```
1  socket.on('disconnect', () => {
2    ws.write(']}');
3    ws.close();
4  });
```

Figura 24: Event de desconnexió

Una vegada tancat el fitxer, aquest ja està disponible per a ser reproduït per l'aplicació de monitoratge.

4.2.3 Runner Boy

Per a la implementació d'aquest joc s'ha fet servir el *framework Phaser.io*. Aquest *framework*, basa l'execució dels jocs en vàries funcions principals, que podem anomenar funcions d'estat. Un objecte de la classe *Phaser.Game*, pot tenir diversos estats i cada estat executa totes aquestes funcions en cas d'estar definides.

init és la primera funció a ser cridada. S'utilitza per inicialitzar les variables inicials del joc. En el nostre cas s'utilitza per definir les variables de la mida del joc, velocitat dels objectes, gravetat, així com moltes variables auxiliars que necessitarà el joc posteriorment.

preload : aquesta funció és cridada una vegada al principi de l'execució del joc, després de la funció *init*. Aquí carregam les imatges que utilitzarà el nostre joc, per a que el *framework* sigui capaç d'introduir-les a l'escena del joc més ràpidament.

create es crida just després que la funció *preload* s'hagi completat. Aquí s'afegeixen els objectes principals del joc, el fons, el terra, el jugador, etc. També és el moment en què s'inicialitzen els intervals d'inserció d'obstacles i estrelles que aniran apareixent periòdicament.

update aquest és el bucle principal del joc. S'executarà periòdicament i és on s'han d'introduir les operacions principals d'actualització del joc, per exemple, actualitzacions de posicions d'objectes en base als algoritmes del joc. En el nostre cas com que gran part de les accions del joc estan basades en esdeveniments, ens ha quedat una funció molt senzilla, que només gestiona el moviment horitzontal del jugador.

render la majoria d'objectes es renderitzen automàticament quan utilitzam el *framework Phaser.io*, i per tant, en cas de no necessitar cap efecte de postprocessament, no fa falta s'implementi aquesta funció. En el nostre cas no ens ha calgut implementar-la.

Pot ser, la part més interessant de la implementació és la de control mitjançant el controlador *Leap Motion*.

Com s'ha descrit anteriorment, l'usuari té la capacitat de fer saltar el jugador realitzant moviments d'extensió del canell. Aquest és un gest que no és detectat automàticament pel servei del *Leap Motion*.

Per tal de detectar el gest d'extensió del canell i actuar en conseqüència sobre el joc, s'ha implementat una funció de detecció d'aquest gest. Un gest es pot definir com un moviment que realitza l'usuari amb la mà, i que consta d'un estat inicial que indica l'inici del gest, una sèrie de moviments de transició, i un estat final que indica la finalització del gest.

En el nostre cas, l'usuari es posiciona amb la mà i el braç paral·lels al pla horitzontal, estant la mà sobre el controlador *Leap Motion*. El gest comença quan l'usuari eleva els dits de la mà per damunt el braç.

Figura 25: Inici del gest.

Per detectar aquest moviment es calcula el producte escalar entre el vector direccional del braç i el vector normal del palmell de la mà. Per calcular aquest producte s'utilitzen les components Y i Z dels vectors en tres dimensions que obtenim de la API del *Leap Motion*. En la posició inicial on la mà i el braç tenen la mateixa direcció (en components y, z), el vector normal del palmell i el vector direccional del braç formen un angle de 90°, i per tant, el producte escalar és 0.

En la realitat l'usuari no manté una posició tan estable i per tant el producte escalar no arriba a tenir un valor exacte a 0. Per això definim un valor llindar a partir del qual podem determinar amb seguretat que l'usuari ha començat a realitzar el moviment d'extensió del canell.

A partir d'aquí consideram el gest com a iniciat. Mentre l'usuari mantingui el moviment, el producte escalar seguirà augmentant i per tant podem considerar que l'usuari segueix realitzant el gest.

Quan el producte vectorial deixa d'augmentar consideram que l'usuari ha arribat al màxim del moviment d'extensió i per tant podem donar el gest com a acabat.

Figura 26: Fi del gest.

4.2.4 Cubes Road

Per implementar aquest joc s'ha utilitzat la llibreria *Three.js* que permet la manipulació d'objectes 3d en navegadors web.

Quan s'utilitza la llibreria *Three.js* s'han de tenir en compte tres elements bàsics. Una escena, una càmera i un *renderer*.

Scene l'escena és l'objecte que utilitza *Three.js* per mostrar els objectes. Una vegada creat qualsevol objecte, si volem que es mostri per pantalla, s'ha d'afegir a aquesta escena.

Camera és l'objecte a través del qual veiem l'escena en 3 dimensions, *Three.js* disposa de vàries càmeres diferents, l'estudi de les quals estaria fora de l'abast d'aquest projecte.

Renderer l'objecte *renderer* és el que utilitza *Three.js* per pintar els elements per pantalla. S'ha utilitzat *WebGL* per aquest propòsit, Three.js disposa d'altres "renderers" que generalment s'utilitzen en navegadors que no disposen de la tecnologia *WebGL*.

La implementació es basa en 4 classes principals per gestionar el joc. La classe principal, *Game*, és la classe que representa el joc i conté la major part de les funcions. La classe *Cube*, és la classe que representa els cubes que ha de capturar l'usuari, a més, aquesta serveix de base per les altres dues classes principals, *EnemyCube* que representa els cubs enemics que l'usuari ha d'esquivar, i *Picker* que representa el jugador dintre del joc. A diferència del joc anterior, on la llibreria gestionava el bucle principal del joc, en aquest cas, s'ha hagut de gestionar manualment.

La següent figura mostra el bucle principal del joc:

```
1  animate(tFrame) {
2    if (this.state === 'Playing') {
3      requestAnimationFrame(this.animate.bind(this));
4    }
5    const pickerXPos = getPosition(
6      (-roadWidth / 2) + 75, (roadWidth / 2) - 75);
7    if (pickerXPos) {
8      this.picker.position.x = pickerXPos;
9      this.picker.needsUpdate();
10   }
11   if (tFrame - this.tCube > this.cubesInterval){
12     if (tFrame - this.tEnemy > this.enemiesInterval) {
13       this.tEnemy = tFrame;
14       this.tCube = tFrame;
15       this.addEnemy();
16     }else {
17       this.tCube = tFrame;
18       this.addCube();
19     }
20   }
21   this.handleMisses();
22   this.scene.simulate();
23   this.renderer.render(this.scene, this.camera);
24 }
```

Figura 27: Bucle principal del joc.

El bucle està controlat per la funció *requestAnimationFrame* [16], aquesta funció avisa al navegador de que es vol realitzar una animació i fa que aquest cridi a una funció just abans del pròxim repintat de pantalla.

Com es pot veure és un bucle molt senzill. A continuació es descriu breument.

El primer que es controla és l'estat del joc, si el joc està en estat *Playing* el bucle reserva la pròxima iteració amb la funció *requestAnimationFrame*. L'estat del joc pot canviar quan el jugador perd la partida.

Després es crida a la funció *getPosition*, aquesta funció rep com a paràmetres un valor mínim i un màxim, la funció retorna un valor que representa la posició de la mà entre aquests dos valors. Més endavant es presenten els detalls d'implementació de la funció.

A continuació, si la funció *getPosition* ha retornat un valor, s'actualitza la posició a l'eix X del jugador. Recordem que el jugador només es pot moure en l'eix X per mirar de capturar els cubs. Podria ser que la funció no retornàs cap valor, perquè el controlador *Leap Motion* no està connectat, i per tant no és capaç d'obtenir els valors de la mà de l'usuari.

El pas següent és comprovar si cal afegir un nou cub a l'escena, aquesta comprovació consisteix en calcular el temps que ha passat des del darrer cop que es va afegir un cub al joc, si fa més temps que un determinat interval, s'afegeix un cub, sinó no. Aquest interval s'anirà fent més petit a mesura que la puntuació del jugador augmenti i per tant apareixeran cubs més sovint. Per explicar com es calcula el temps que ha passat des de la darrera vegada que es va afegir un cub, cal mencionar que la funció *requestAnimationFrame*, crida a la funció que se li passa com paràmetre afegint un paràmetre addicional. Aquest paràmetre és una referència del temps que ha passat des del temps en que la navegació a la pàgina web actual va començar amb una precisió de cinc microsegons [16]. Per si sol, aquest paràmetre no té gaire valor, però donada la precisió que ofereix pot ser utilitzat per guardar temps de referència dins el joc. Així cada vegada que s'afegeix un cub, es guarda el temps en el qual s'ha afegit utilitzant aquest valor.

En el cas que s'hagi d'afegir un cub, el joc després determina quin tipus de cub s'ha d'afegir, un cub per a que el jugador obtingui més puntuació, o un cub enemic. Aquesta comprovació és fa seguint la mateixa filosofia que l'anterior, si ha passat més d'un determinat temps des de la darrera vegada que es va afegir un cub enemic, s'afegeix un cub enemic, sinó s'afegeix un cub bo, o cub de premi. Aquest interval de cubs enemics sempre serà més gran que el de cubs bons, de tal manera que sempre apareguin més cubs bons que cubs enemics.

La funció *handleMisses* gestiona els cubs que l'usuari no ha capturat, bé perquè aquest cub era un cub enemic i per tant l'usuari l'ha ignorat deliberadament, o bé perquè ha estat incapaç de capturar-lo. Es considera que l'usuari ha estat incapaç de capturar el cub quan la posició respecte a l'eix Z del cub és major que la posició respecte a l'eix Z de la càmera. Ja que la càmera enfoca en direcció al jugador en aquest moment el cub ja no és visible a l'escena.

Si el cub és un cub enemic aquest simplement s'elimina de l'escena. En canvi si el cub és un cub de premi, el jugador l'hauria d'haver capturat i per tant es considera que el jugador ha fallat un cub, si el jugador falla 5 o més cubs perd la partida.

Per acabar es criden dues funcions que no són d'implementació pròpia. Com hem pogut comprovar, el bucle no gestiona el moviment dels cubs, llevat del cub que representa el jugador, el moviment d'aquests es gestionat per la llibreria *Physi.js*. Aquesta llibreria també és l'encarregada de detectar les col·lisions entre objectes de l'escena i avisar-nos mitjançant esdeveniments. El mètode *simulate* de l'objecte *scene* s'encarrega de demanar a la llibreria que realitzi els càlculs necessaris per determinar les noves posicions de tots els cubs de l'escena. En darrera instància, el mètode *render* de l'objecte

renderer notifica a *Three.js* que ha de repintar l'escena.

Com s'ha mencionat en aquest apartat, la llibreria *Physi.js* s'encarrega de detectar les col·lisions entre objectes per nosaltres. No obstant, no n'hi ha prou amb detectar que dos objectes han col·lisionat, sinó que s'ha d'actuar en conseqüència, i per això la llibreria permet atendre les col·lisions mitjançant esdeveniments de la mateixa manera que es fa amb els esdeveniments clàssics de JavaScript [7].

D'aquesta manera, l'objecte joc dóna d'alta el que es coneix com a funció *handler* o *callback*, que és una funció que atén un esdeveniment o que és cridada per retornar la resposta a una funció, per atendre els esdeveniments de col·lisió que origina l'objecte *Picker*.

```
1  initPicker() {
2    this.picker = new Picker(this.scene);
3    this.picker.position.y = CUBES_HEIGHT;
4    this.picker.position.z = PICKER_Z_POSITION;
5    this.picker.addToScene();
6    this.picker.addEventListener('collision', this.
       onCubePicked.bind(this));
7  }
```

Figura 28: Inicialització de l'objecte *picker*

Així, cada vegada que l'objecte *picker* col·lisiona amb un altre objecte dins l'escena, la funció *onCubePicked* és cridada.

```
1  onCubePicked(cubeMesh, relVelocity, relRotation,
       contactNormal){
2    this.particlesHolder.spawnParticles(
3    cubeMesh.position.clone(), 10, cubeMesh.material.color,
         .8);
4    this.scene.remove(cubeMesh);
5    if (this.cubesMap.get(cubeMesh) instanceof EnemyCube) {
6      this.picker.shrink();
7    }else {
8      this.score += 10;
9      this.scoreText.innerHTML = `score ${this.score}`;
10     this.level = Math.floor(this.score / 50);
11     if (this.score % 50 === 0) {
12       this.cubesInterval = Math.max(
13       this.cubesInterval - this.cubesIntervalStep, this.
             maxCubesInterval);
14       this.velocity = Math.min(
15       this.velocity + this.velocityStep, this.maxVelocity);
16     }
17     if (this.score % 100 === 0) {
18       this.shrinkCubes();
19     }
20   }
21   this.cubesMap.delete(cubeMesh);
22 }
```

Figura 29: Funció que atén les col·lisions dels cubs.

La funció rep varis paràmetres que donen informació sobre la col·lisió entre els objectes, l'únic que s'utilitza és el primer, que és una referència a l'objecte amb el qual ha col·lisionat el jugador. Cal recordar, que aquesta funció és un mètode de la classe principal Game i per tant, l'objecte *this* fa referència a l'objecte de la classe Game. El primer que es fa quan es detecta una col·lisió, és generar un efecte visual que crea múltiples partícules en el lloc de la col·lisió i genera un efecte similar a una explosió, d'això se n'encarrega l'objecte *particlesHolder*. No s'entrarà en més detalls per no tractar-se d'una funcionalitat del joc, sinó d'un simple efecte visual per remarcar que el jugador ha capturat un cub.

Acte seguit, s'elimina el cub amb el qual ha col·lisionat l'usuari, ja que aquest cub ha estat capturat.

A partir d'aquí tenim dues opcions, que el cub sigui un cub enemic, o un cub bo o de premi.

Si el cub era un cub enemic: el que es fa és fer tornar l'objecte *picker* una mica més petit, per tal que a l'usuari li sigui més difícil capturar els pròxims cubs.

En cas que sigui un cub de premi, s'augmenta la puntuació de l'usuari en 10 punts. Llavors, es prenen una sèrie de mesures per augmentar la dificultat de la partida. Cada 50 punts l'interval entre els cubs es redueix i augmenta la velocitat a la que es mouen els cubs. A més cada 100 punts els cubs disminueixen la seva mida. El darrer pas és eliminar la referència al cub capturat del mapa que el joc utilitza per tenir una referència a tots els cubs que hi ha a l'escena.

D'aquesta manera, el joc continua fins que l'usuari perd la partida o decideix deixar de jugar.

4.2.5 Catch Stars

Aquest joc, s'ha implementat sense utilitzar cap llibreria externa per poder experimentar i comparar com seria la creació d'un joc utilitzant només APIs de baix nivell, ja disponibles en el navegador. Es pot comparar així, les facilitats que donen les diverses llibreries i *frameworks* que hi ha disponibles per a desenvolupar videojocs per navegador, en vers la utilització d'una API bàsica com és la que proporciona l'element *canvas* HTML5 i *CanvasRenderingContext2D*.

D'altra banda, implementar un joc sense utilitzar cap *framework* ni llibreria extra pot donar una visió més real i acurada del que realment estan fent aquestes llibreries pel desenvolupador. És el cas per exemple, de la detecció de col·lisions, que la majoria de llibreries de desenvolupament de videojocs en HTML5, gestionen automàticament, sense requerir gairebé cap acció al desenvolupador. En la implementació d'aquest videojoc en canvi, s'ha hagut de programar aquesta detecció de col·lisions.

La implementació consta de diverses classes:

CatchstarGame és la classe principal del joc, conté entre d'altres, les funcions de inicialització, actualització, renderització, així com el bucle principal del joc.

Component és la classe de la qual estenen els altres components visuals del joc. Conté els atributs bàsics que necessita un objecte del joc (posició, color, mida, velocitat, etc), així com les funcions d'actualització de posició, pintar, i calcular col·lisions amb un altre objecte, aquestes dues darreres, només per a objectes quadrats.

Star Aquesta classe estén de la classe *Component* i a part d'afegir alguns atributs, sobreescriu les funcions de pintar, per tal de pintar una estrella en lloc d'un simple cub, i calcular col·lisions.

Picker és la classe que representa els dits del jugador dins el joc i s'utilitza per capturar les estrelles que cauen. també estén de la classe *Component* i sobreescriu la funció d'actualització, ja que l'actualització d'aquest objecte està controlada per l'usuari a través del dispositiu *Leap Motion*.

Group Permet agrupar un conjunt de components i poder gestionar-los alhora. S'utilitza per agrupar les estrelles que van apareixent i disposa de mètodes per tractar tots els objectes del grup a la vegada, com és el cas de les funcions *draw* i *update*, que pinten i actualitzen respectivament, tots els objectes de dins el grup.

Després de construir una instància de la classe *CatchStarGame*, el joc comença amb el mètode *start* de la mateixa classe.

```
1  start() {
2    this.reset();
3    this.clear();
4    this.stars = new Group(this);
5
6    const mid = Math.floor(this.width / 2);
7    this.lPicker = new Picker(
8    this, 30, 30, 'red', mid - 90, this.height - 100, 'left')
         ;
9    this.rPicker = new Picker(
10   this, 30, 30, 'red', mid + 60, this.height - 100, 'right'
         );
11
12   this.scoreComponent = new Component(
13   this, '30px', 'Consolas', 'white', this.width - 200, 40,
         'text');
14   this.scoreComponent.text = 'SCORE:   0';
15   this.lifesComponent = new Component(
16   this, '30px', 'Consolas', 'white', this.width - 200, 80,
         'text');
17   this.lifesComponent.text = 'LIFES:   ' + this.lifes;
18   window.requestAnimationFrame(this.animate.bind(this));
19 }
```

Figura 30: Mètode d'inici del joc.

El primer que fa la funció és cridar a la funció *reset*, aquesta funció configura els valors inicials del joc, com per exemple, puntuació 0, velocitat inicial de les estrelles, etc. Després, crida a la funció *clear*, aquesta funció esborra tot el que hi hagi pintat dins l'element *canvas* pintant un quadrat

negre de la mida exacte del *canvas*, aquest serà el fons del joc. Si el joc no ha començat encara, no hi ha res a esborrar, però com es veurà més endavant, aquesta mateixa funció serà cridada en el bucle principal del joc abans de repintar els components.

Tot seguit inicialitza el grup d'estrelles, com es pot veure per això s'utilitza la classe Group descrita anteriorment.

A continuació, s'inicialitzen els dos components *picker*, que són els que utilitza el jugador per capturar les estrelles. També s'inicialitzen els textos de puntuació i de les vides del jugador.

Per acabar, la funció inicia el bucle principal del joc cridant a *requestAnimationFrame*, la mateixa que utilitza el joc descrit anteriorment.

El bucle principal del joc és el següent:

```
1  animate(tFrame) {
2    this.animationId = window.requestAnimationFrame(this.
       animate.bind(this));
3    this.clear();
4
5    this.update(tFrame);
6    this.render();
7    this.lastRender = tFrame;
8  }
```

Figura 31: Bucle principal

El primer que fa és reservar la pròxima iteració del joc per utilitzant una altra vegada la funció *requestAnimationFrame*.

Després crida a la funció clear per esborrar els components pintants a la pantalla, ja que en aquest bucle s'actualitzaran les seves posicions i per tant han de ser pintats en un altre lloc. Després es crida la funció update, que s'encarregarà d'actualitzar les posicions de tots els components que hi hagi en aquest moment en el joc. També s'encarrega de determinar si fa falta afegir una nova estrella utilitzant la mateixa tècnica que utilitza el joc *Runner Boy*, de comprovar el temps que ha passat des de la darrera vegada que es va introduir una estrella. La funció render, s'encarrega únicament de cridar a les funcions de pintat (*draw*) dels diferents components del joc, el grup d'estrelles, els objectes *picker*, i els textos.

Com s'ha comentat, la funció update, crida a les funcions d'actualització dels diferents components.

```
1  update(tFrame) {
2    if (tFrame - this.lastStar > this.starsInterval) {
3      this.addStar(tFrame);
4    }
5
6    this.lPicker.update();
7    this.rPicker.update();
8
9    this.stars.update();
10 }
```

Figura 32: Funció d'actualització principal.

Tant *lPicker* com *rPicker* són els components que controla l'usuari mitjançant el controlador *Leap Motion*. D'aquesta manera, el mètode *update* de la classe *Picker* ha d'obtenir les dades de posició del Leap Motion per actualitzar la posició del component. Per això s'ha hagut d'implementar una funció que calculi la distància entre dos dits a partir de les dades de seguiment del *Leap Motion*.

```
function getFingersDistance(f1, f2) {
    let output = {};
    const frame = controller.frame();
    if (frame.hands && frame.hands.length === 1) {
        const capturedHand = frame.hands[0];
        const fingerA = capturedHand[f1];
        const fingerB = capturedHand[f2];
        const middleVector = [fingerA.direction[0], fingerA.
           direction[2]];
        const ringVector = [fingerB.direction[0], fingerB.
           direction[2]];

        const dotProduct = Leap.glMatrix.vec2.dot(
           middleVector, ringVector);
        const lengths = Leap.glMatrix.vec2.len(middleVector)
           * Leap.glMatrix.vec2.len(ringVector);
        const cosinus = dotProduct / lengths;
        output.angle = Math.acos(cosinus) * 180 / Math.PI;

        const distance = Math.abs(fingerA.tipPosition[0] -
           fingerB.tipPosition[0]);
        if (distance > maxDistance) {
            maxDistance = distance;
        }
        if (distance < minDistance) {
            minDistance = distance;
        }
        output.distance = distance;
        output.max = maxDistance;
        output.min = minDistance;
    }
    return output;
}
```

Figura 33: Funció d'actualització principal.

Aquesta funció agafa les dades de dos dits, els noms dels quals arriben per paràmetre, i calcula dades com la distància entre les puntes dels dos dits, l'angle que hi ha entre ells així com quina és la distància mínima i màxima de separació que ha realitzat l'usuari amb aquells dos dits. En funció de la distància actual i les distàncies mínimes i màximes, el mètode *update* de la classe *Picker* calcula quina és la seva nova posició.

Com s'ha comentat anteriorment, ja que no s'han utilitzat llibreries addicionals per implementar aquest joc, la detecció de col·lisions s'ha hagut de programar de manera explícita. D'això se'n encarrega el mètode *update* de la classe *Group*, ja que aquest mètode itera sobre totes les estrelles que hi

ha al joc i comprova si cada una d'elles ha col·lisionat amb algun dels dos objectes de la classe *Picker*.

La detecció de col·lisions pot semblar un problema senzill, però a mesura que el nombre d'objectes i la complexitat d'aquests augmenta, el càlcul d'aquestes col·lisions pot arribar a suposar un repte computacional per a qualsevol aplicació en temps real, com per exemple els videojocs.

En el nostre cas s'ha utilitzat un algoritme molt senzill que és basa en el càlcul de la col·lisió de dues esferes.

Es defineixen dues esferes que contenen els objectes dels quals volem calcular la col·lisió. Una vegada tenim les dues esferes definides, si la distància entre els seus centres és major a la suma dels seus radis, les dues esferes estan massa lluny una de l'altra com per col·lisionar. En canvi si la distància és menor a la suma dels seus radis, les dues esferes intersequen o una d'elles conté completament l'altra i per tant es pot determinar que els dos objectes han col·lisionat.

Aquest procés procés és calculat per el mètode *crashWith* de la classe *Star*. Com s'ha comentat és el mètode *update* de la classe *Group* el que s'encarrega de detectar si una estrella ha col·lisionat amb un *picker* i ho fa utilitzant aquest mètode *crashWith*.

```
update() {
    for (let [id, c] of this.components) { // eslint-disable-
        line no-unused-vars
        c.update();
        if (c.type === 'star' && ! c.collided) {
            if (c.crashWith(this.game.lPicker)) {
                c.collided = true;
                this.game.canvas.dispatchEvent(new Event('
                    collision'));
            }
            if (c.crashWith(this.game.rPicker)) {
                c.collided = true;
                this.game.canvas.dispatchEvent(new Event('
                    collision'));
            }
        }

        if (c.y > this.game.height || c.collided) {
            this.remove(c);
            if (!c.collided) {
                this.game.canvas.dispatchEvent(new Event('
                    starMiss'));
            }
        }
    }
}
```

Figura 34: Mètode *update* de la classe *Group*

En cas de detectar-se una col·lisió es dispara un esdeveniment de tal manera que, l'objecte de la classe *CatchStarsGame* que està escoltant aquests esdeveniments actui en conseqüència. També es pot veure en aquesta funció com si es detecta que una estrella s'ha desplaçat més enllà de la part inferior del *canvas*, es dispara un altre esdeveniment *starMiss*.

Quan l'objecte joc rep un esdeveniment de tipus *collision* augmenta la puntuació de l'usuari, en canvi si rep un esdeveniment de tipus *starMiss*, vol dir que l'usuari no ha pogut capturar una estrella i per tant perd una vida. La partida acaba quan l'usuari ha perdut totes les vides.

4.2.6 Network plugin

Com s'ha descrit a l'apartat de disseny, aquest és un *plugin* per al controlador *Leap Motion* que es connecta a un servidor de *web socket* i envia periòdicament les dades de seguiment capturades.

La funció "Leap.plugin(name, function)" registra un *plugin* per al controlador Leap Motion de tal manera que el desenvolupador és capaç de dir al la API del *Leap Motion* que utilitzi un *plugin* prèviament registrat, utilitzant la funció "Leap.use(name, options)"

La següent figura mostra les parts més rellevants del registre del *plugin*.

```
1  Leap.plugin('socket-networking', function(scope) {
2      const controller = this;
3      scope.framePacker = framePacker = new FramePacker();
4          if (!scope.bufferWindow && scope.bufferWindow !== 0)
               {
5          scope.bufferWindow = 30;
6      }
7      scope.connectionEstablished = function() {
8          scope.sendFrames = true;
9      };
10     controller.on('streamingStarted', function() {
11         scope.connect();
12     });
13     scope.sendFramesBuffer = function() {
14         scope.connection.emit('frameBuffer', framesBuffer);
15     }
16     return {
17         beforeFrameCreated: function(frameData) {
18             if (!scope.sendFrames) {
19                 return;
20             }
21             if (frameData.hands && frameData.hands.length >
                   0) {
22                 framesBuffer.push(framePacker.pack(frameData)
                       );
23                 if (framesBuffer.length >= scope.bufferWindow
                       ) {
24                     scope.sendFramesBuffer();
25                     framesBuffer = [];
26                 }
27             }
28         }
29     };
30 });
```

Figura 35: *Network plugin*

S'han eliminat les parts menys rellevants per l'estudi.

En termes generals, la funció que *Leap.plugin* rep com a segon paràmetre, defineix una sèrie de funcions internes i retorna un objecte. Aquest objecte

és utilitzat per l'API del Leap Motion quan es crida a la funció "Leap.use".

En el nostre cas, l'objecte retornat pel *plugin* només defineix una propietat *beforeFrameCreated*, la qual representa una funció que serà cridada amb les dades del pròxim *frame* (fotograma), abans que aquest sigui retornat per cap mètode de la API. Aquí és on el nostre *plugin* agafa les dades de seguiment, les empaqueta utilitzant una estructura determinada i les envia al servidor de *web socket*.

A causa del gran nombre de fotogrames que és capaç d'enviar el controlador *Leap Motion*, el *plugin* no envia les dades al servidor per a cada fotograma, sinó que va acumulant les dades ja empaquetades i quan ha empaquetat una quantitat de fotogrames, els envia al servidor. Aquesta quantitat de fotogrames que acumula el *plugin* és configurable a través de la propietat *bufferWindow* (finestra de dades).

La utilització d'una finestra de dades com aquesta pot ser beneficiós de cara als requeriments computacionals i de xarxa del *plugin*, però per altra banda, fa que les dades ja no arribin al servidor en temps real, per tant si l'aplicació que utilitza aquest *plugin* necessita una connexió en temps real amb el servidor, sempre es pot definir aquest *bufferWindow* amb el valor 0.

L'estructura que s'utilitza per guardar les dades és la mateixa que utilitza el plugin *leapjs-playback* [11] per a reproduir les dades del Leap Motion. Així quan les dades es guarden a un fitxer, ja estan disponibles per a ser reproduïdes utilitzant el *plugin* de *playback*. La següent figura mostra quines són les dades enviades al servidor així com també l'estructura utilitzada per enviar-les.

```
const packingStructure = [
    'id',
    'timestamp',
    {hands: [[
        'id', 'type', 'direction', 'palmNormal',
        'palmPosition', 'palmVelocity',
        'stabilizedPalmPosition', 'pinchStrength',
        'grabStrength', 'confidence',
        'armBasis', 'armWidth', 'elbow', 'wrist'
    ]]},
    {pointables: [[
        'id', 'direction', 'handId', 'length',
        'stabilizedTipPosition', 'tipPosition',
        'tipVelocity', 'tool', 'carpPosition', 'mcpPosition',
        'pipPosition', 'dipPosition', 'btipPosition',
        'bases', 'type'
    ]]},
    {interactionBox: [
        'center', 'size'
    ]}
];
```

Figura 36: Estructura de dades del *plugin*.

4.2.7 Aplicació de monitoratge

És una aplicació web que utilitza la llibreria *Three.js* i els plugins de Leap Motion *leapjs-playback* i *leapjs-rigged-hand*, per crear una escena en tres dimensions on es visualitza una mà realitzant els mateixos moviments que ha realitzat l'usuari durant la seva sessió de joc.

A part de la inicialització de l'escena 3D, la major part de la implementació d'aquesta aplicació ha estat configuració dels *plugins* mencionats.

```
controller
    .use('transform', {
        position: new THREE.Vector3(0, -80, 0)
    })
    .use('playback', {
        recording: './leap-output.json',
        pauseOnHand: false,
        loop: false
    })
    .use('riggedHand', {
        parent: scene,
        helper: true,
        renderFn: function() {
        renderer.render(scene, camera);
        controls.update();
    },
    materialOptions: {
        color: new THREE.Color(0x333333)
    },
    camera: camera
    }).connect();
```

Figura 37: Configuració dels *plugins* per a l'aplicació.

L'aplicació utilitza varis plugins ja disponibles per a *leapjs*, *transform*, *playback* i *riggedHand*.

El plugin *transform* s'utilitza per alterar les posicions de la mà dins l'escena. Utilitzant aquest *plugin* es poden traslladar les coordenades que retorna la API del *Leap Motion*. S'ha configurat per desplaçar les dades del *Leap Motion* 80 punts cap avall en l'eix Y, facilitant la visualització de l'aplicació en pantalles petites.

Playback, és el *plugin* que permet reproduir els moviments dels usuaris a partir d'un fitxer. Aquest *plugin* també ofereix la possibilitat de guardar els moviments dels usuaris en fitxers per després reproduir-los. Analitzant les opcions de configuració, podem veure tres opcions, *recording*, *pauseOnHand* i *loop*.

recording indica quin fitxer de sessió de joc es reproduirà. Aquest fitxer serà servit per el servidor web, el *plugin playback* està preparat per descarregar els fitxers de reproducció via *AJAX*.

pauseOnHand configura si el *plugin* ha de pausar la reproducció en cas que el dispositiu *Leap Motion* detecti una mà en el seu camp de visió. Aquí es desactiva aquesta opció ja que l'usuari de reproducció de les sessions no utilitza el dispositiu *Leap Motion*.

loop indica si el *plugin* ha de reproduir el fitxer en bucle o no. També està desactivat, perquè es vol que l'usuari tengui el control de quan es pausa i s'executa la reproducció.

RiggedHand és un *plugin* que facilita la creació de mans en base a les dades de seguiment del *Leap Motion*. Així no s'ha hagut de crear un model en 3D d'una mà per introduir-lo a l'escena.

Per tal de pintar el model de la mà en 3D, utilitza també la llibreria *Three.js*, i per tant necessita una càmera, una escena, i un render, tal com passa amb el videojoc implementat *Cubes Road*. Tal com els altres *plugins*, permet una extensa configuració mitjançant un objecte d'opcions.

parent un objecte de *THREE.scene* en el qual s'afegirà la mà creada pel *plugin*.

helper aquesta opció afegeix uns vectors a l'interior de la mà que representen els diferents ossos de cada un dels dits. D'aquesta manera és més fàcil identificar la posició exacte de cada dit i detectar si l'exercici es realitza correctament.

renderFn és una funció que és cridada periòdicament i s'utilitza per indicar a l'objecte renderer de Three.js que ha de pintar els objectes a pantalla. A més aprofitam per actualitzar els controls que permeten a l'usuari rotar l'escena amb el ratolí.

materialOptions aquestes opcions es passen directament als objectes creats pel *plugin* amb *Three.js*, accepta qualsevol paràmetre que sigui acceptat per els materials de *Three.js*. Aquí s'utilitza per canviar el color dels materials.

camera l'objecte càmera creat amb *Three.js*.

L'aplicació de monitoratge disposa també d'una barra de reproducció com la que utilitzen els reproductors de vídeo. Desafortunadament el *plugin* no disposa de resposta de progrés de la reproducció i per tant, s'ha hagut d'implementar l'actualització de la barra de reproducció. Es consulta periòdicament quin és el fotograma actual de la reproducció, i en funció d'aquest i de la quantitat total de fotogrames, es calcula la posició del punter de la barra de reproducció.

```
function initSlider(min, max) {
    playSlider.max = max;
    playSlider.min = min;
    setInterval(() => {
        if (player.state === 'playing') {
            playSlider.value = player.recording.frameIndex;
        }
    }, 5);
    playSlider.oninput = (event) => {
        player.setFrameIndex(parseInt(event.target.value) -
            1);
    }
}
```

Figura 38: Actualització de la barra de reproducció.

Utilitzant un interval de 5ms, s'actualitza el punter de la barra de reproducció. 5ms ens dóna una taxa de refresc de 200 vegades per segon, més que suficient per no perdre cap fotograma produït pel *plugin* de reproducció i oferir un moviment fluït del punter.

5 Futur

Una vegada acabat el projecte i coberts tots els requeriments plantejats, es poden analitzar diferents vies de millora del projecte, ja siguin noves funcionalitats o millores de les prestacions ja existents. Algunes de les possibles serien les següents:

- Integrar els jocs en una plataforma web que requereixi un registre, en la qual les sessions de joc dels usuaris es guardin per tipus de joc i per temps. La mateixa plataforma pot tenir un accés pels responsables de les teràpies, que podrien per exemple tenir un llistat dels seus pacients amb les seves sessions de joc.

- Analitzar les dades de les sessions de joc per extreure informació estadística. Les dades de les sessions de joc que es guarden, només s'utilitzen per a reproduir les sessions. Aquestes dades es podrien analitzar per extreure dades del moviment de cada pacient, rangs de moviment, màxima extensió d'un moviment, etc. Així els responsables podrien tenir accés a representacions gràfiques de les dades.

- Millorar l'apartat gràfic dels jocs. No s'ha d'oblidar que un dels objectius principals d'utilitzar videojocs per realitzar exercicis de rehabilitació, és aconseguir una motivació extra de cara als pacients d'aquesta

rehabilitació. L'apartat gràfic d'un joc pot influir molt a l'hora d'incentivar el joc, per tant, per tal d'augmentar aquesta motivació extra es pot treballar en millorar els gràfics dels jocs desenvolupats.

6 Conclusions

Durant el desenvolupament d'aquest projecte, s'ha aconseguit complir amb els objectius plantejats a l'inici del mateix.

S'ha descrit la implementació com a prova de concepte, d'un sistema de suport a la rehabilitació de pacients amb lesions a les articulacions de la mà, oferint la possibilitat que aquestes sessions de rehabilitació siguin monitorades i revisades pel responsable de la teràpia. Revisant els requeriments plantejats inicialment, es pot afirmar que el projecte s'ha completat amb èxit, tot i quedar obertes vàries opcions de millora.

A títol personal, la realització d'aquest projecte m'ha permès experimentar amb tecnologies de desenvolupament de videojocs i interfícies basades en visió, fins ara desconegudes per a mi, però amb les quals tenia moltes ganes de treballar. Aquest fet ha llastrat possiblement, l'evolució del projecte en les seves fases inicials, però res que no s'hagi pogut suplir amb la diversitat de recursos que es poden trobar actualment a la xarxa.

En la mesura del possible s'han utilitzat tecnologies conegudes per al desenvolupament d'algunes parts o sistemes del projecte. Per altra banda, una de les etapes més costoses de desenvolupament del projecte, ha estat treballar amb l'apartat gràfic dels videojocs. Aquest apartat és on el projecte té més recorregut per endavant, sinó el que més. El desenvolupament del projecte, m'ha permès adquirir coneixements en relació al desenvolupament de videojocs i descobrir la gran complexitat que hi pot haver darrera parts aparentment senzilles del seu desenvolupament.

També he pogut descobrir la utilitat que poden arribar a tenir les experiències interactives per salut, basades en videojocs, experiències que des del meu punt de vista, actualment, ja podrien tenir una major implantació.

7 Referències

[1] *Asus Xtion.* URL: `https://www.asus.com/3D-Sensor/Xtion/`.

[2] Alex Colgan. *How Does the Leap Motion Controller Work?* URL: `http://blog.leapmotion.com/hardware-to-software-how-does-the-leap-motion-controller-work/`.

[3] J Delisa. *Rehabilitation Medicine Principles and Practice*. 1998. Cap. 68, pàg. 1717 - 1732.

[4] *E3: Microsoft shows off gesture control technology for Xbox 360*. URL: http://latimesblogs.latimes.com/technology/2009/06/microsofte3.html.

[5] *Ejercicios de mano y muñeca*. URL: https://traumatologiahellin.wordpress.com/ejercicios/ejercicios-de-mano-y-muneca/.

[6] *How Does the Leap Motion Controller Work?* URL: http://blog.leapmotion.com/hardware-to-software-how-does-the-leap-motion-controller-work/.

[7] *Javascript Event*. URL: https://developer.mozilla.org/en-US/docs/Web/API/Event.

[8] Maryam Khademi et al. "Free-hand interaction with leap motion controller for stroke rehabilitation". A: (2014), pàg. 1663 - 1668. DOI: http://dx.doi.org/10.1145/2559206.2581203.

[9] *Leap Motion SDK and Plugin Documentation*. URL: https://developer.leapmotion.com/documentation/index.html.

[10] *LeapJS and Plugins*. URL: https://developer-archive.leapmotion.com/javascript#plugins.

[11] *LeapJS Playback*. URL: https://github.com/leapmotion/leapjs-playback.

[12] Iosa M et al. "Leap motion controlled videogame-based therapy for rehabilitation of elderly patients with subacute stroke: a feasibility pilot study". A: *Topics in Stroke Rehabilitation* 22.4 (2015), pàg. 306 - 316. DOI: 10.1179/1074935714Z.0000000036.

[13] Cristina Manresa-Yee. *Advanced and natural interaction system for motion-impared users*. 2009.

[14] Nuno Matos, António Santos i Ana Vasconcelos. "Kinteract: a multi-sensor physical rehabilitation solution based on interactive games". A: (2014), pàg. 350 - 353. DOI: http://dx.doi.org/10.4108/icst.pervasivehealth.2014.255325.

[15] *Nintendo hopes Wii spells wiinner*. URL: https://usatoday30.usatoday.com/tech/gaming/2006-08-14-nintendo-qa_x.htm.

[16] *requestAnimationFrame*. URL: https://developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame.

[17] Webster i Celik. "Systematic review of Kinect applications in elderly care and stroke rehabilitation". A: *Journal of NeuroEngineering and Rehabilitation* 11 (108 2014). DOI: 10.1186/1743-0003-11-108.

[18] *What You Need To Know About Node.js*. URL: http://readwrite.com/2013/11/07/what-you-need-to-know-about-nodejs.

[19] Liu Z. et al. "Leap-Motion Based Online Interactive System for Hand Rehabilitation". A: (2015).