EPS
Escola Politècnica Superior
UdG

# Projecte/Treball Fi de Carrera

**Estudi:** Eng. Tècn. Informàtica de Sistemes. Pla 2001

**Títol:** MODELAT, DISSENY I DESENVOLUPAMENT D'UNA APLICACIÓ MULTIJUGADOR PER ANDROID

**Document:** Memòria

**Alumne:** DAVID RUBIO PELEGRINA

**Director/Tutor:** Ignacio Martin
**Departament:** Informàtica i Matemàtica Aplicada
**Àrea:** Coneixament de llengues i sistemes informàtics

**Convocatòria** (mes/any): 09/2013

# TREBALL DE FI DE CARRERA

## MEMÓRIA DEL PROJECTE

# ÍNDEX DE CONTINGUTS

# 1. INTRODUCCIÓ

## 1.1 BREU HISTORIA DE LAS APLICACIONS MÒBILS

El món de la telefonia mòbil ha sofert un canvi important durant els últims anys, revolucionant el tradicional concepte inicial, comunicar-se. Els dispositius mòbils, anteriorment equipats exclusivament per satisfer las necessitats dels seus usuaris inexperts, també han anat sofrint mutacions, oferint actualment a part de la funcionalitat de *trucades*, moltes altres, les quals s'han convertit en essencials en l'acte de comunicació quotidiana.

Les múltiples transformacions que han anat esdevenint en els últims anys han estat conseqüència, en gran part, de l'evolució dels Sistemes Operatius per a telèfons mòbils, marcant els diferents camins en aquest sector. Des de els "inicis" de l'aparicó d'iOS, fins a Android, Windows Phone, Blackeberry OS i Symbian entre altres, els sistemes operatius han anat obrint el camí de la revolució tecnològica, aportant noves idees i hardware, canviant així la manera d'entendre els dispositius mòbils generant noves experiències als seus usuaris finals.

Tot i els diferents canvis en la tecnologia mòbil, la qual neix amb la II Guerra Mundial[1], la comunicació continua jugant un paper primordial, sent la seva primera funcionalitat. No obstant, les noves tecnologies, han anat acostant els dispositius als ordinadors, tant en capacitat com en funcionalitats, gràcies als seus potents microcontroladors, amb els quals s'han obert nous camins per tal d'explotar totes les seves possibilitats. D'aquestes funcionalitats, la més rellevant i amb un creixement més significatiu en la ultima dècada, ha estat, sens dubte, el jocs virtual.

D'aquesta manera, actualment es poden trobar molt tipus de jocs, els quals ofereixen una gran varietat de gèneres, aptes i segmentats en tot el tipus de *target*, sent integrats, cada cop amb més normalitat, en la vessant comunicativa dels dispositius.

Tots aquests esdeveniments han provocat que aquest mercat hagi anat creixent i que el seu suport sigui una de les peces claus dels requeriments de qualsevol Sistema Operatiu mòbil.

[1] http://www.abadiadigital.com/dynatac-8000x-el-primer-movil-de-la-historia/

### 1.3 CORRELACIÓ ENTRE LES APLICACIONS I EL PROJECTE

El meu Projecte Final de Carrera es tracta d'un projecte d'investigació i de gestió, en el qual s'intentaran explotar totes les possibilitats per un d'aquests Sistemes Operatius mòbils, mitjançant l'estudi, desenvolupament i implementació d'un joc multi-jugador per dispositius Android. En ell s'analitzaran les necessitats i funcionalitats que ha d'oferir la meva aplicació/joc i s'implementaran mitjançant diverses eines i/o llibreries per tal de resoldre-les. Durant el mateix, es realitzarà una recerca i anàlisi d'aquestes eines, dels avantatges i inconvenients subjacents, per tal de prendre la decisió més optima enquadrada les necessitats i objectius del Projecte, tenint en compte també seu disseny de la nostra aplicació i finalment proper tal de procedir la seva implementació, sempre regint-me per les possibilitats comunicatives que gaudeixen aquest tipus de dispositius.

### 1.4 MOTIVACIONS

La principal motivació en la selecció d'aquest tema com a P/TFC, és l'interès personal, que he adquirit al llarg de la meva formació acadèmica per aquest sector, que està creixent dramàticament durant els últims anys, fet que es veu plasmat les nombroses possibilitats que pot oferir el mercat laboral, sobretot en les àrees de les noves tecnologies i eines relacionades amb aplicacions per Tablets i Smartphones.

Crec certament que aquest és un sector que encara està molt verd, ja que té pocs anys de vida (com la informàtica en general) i que deixa moltes possibilitats en nínxols i sectors de mercat encara per explotar, sent cada cop més les empreses que decideixen invertir-hi i aportar-hi els seus propis esforços i inversions.

A més, també hi ha jugat un paper important en l'elecció d'aquesta temàtica, la possibilitat d'afrontar el repte de realitzar un projecte d'aquesta magnitud, començant des de zero, estructurant totes i cada una de les seves etapes d'aquest, prenent les decisions corresponents durant totes elles de les quals es composa tant el desenvolupament com el disseny.

Per últim, m'agradaria emfatitzar, també, la motivació de ser capaç d'afrontar i analitzar quines són les millors solucions possibles per tal de poder respondre a les especificacions concretes que haurà de complir la meva aplicació i saber justificar amb certesa el perquè d'aquesta elecció.

Així doncs, considero també com a part importantíssima de l'elecció l'experiència que em pot aportar aquest treball i els coneixements adquirits amb ell de cara al meu perfil i carrera professional.

### 1.5 PROPÒSIT

El propòsit d'aquest treball és el d'aconseguir desenvolupar un joc/aplicació en Android que es composi de diversos elements i funcionalitats útils, integrant-les correctament en el sistema i d'aquesta manera aconseguir obtenir una aplicació completa i funcional de cara a l'usuari final.

L'objectiu primordial és desenvolupar-la completament amb eines i llibreries OpenSource, demostrant que qualsevol idea (o quasi totes) que tinguem per plataformes mòbils, es poden generar sense la necessitat de pagar cap tipus de llicència software o de desenvolupament.

Per últim, demostrar que per el sistema operatiu Android hi ha una gran oferta d'aquestes eines i tecnologies, per les quals és important el saber realitzar un estudi de viabilitat i utilitat, tenint com a referència l'aplicació en qüestió, realitzant un anàlisi de les avantatges i inconvenients que em comporten en el meu projecte.

### 1.6 OBJECTIUS (EL QUE ES VOL ACONSEGUIR)

L'Objectiu final és el d'implementar un joc multi-jugador *un-contra-un* per al Sistema Operatiu Android, integrant diverses tecnologies OpenSource disponibles per a la plataforma.

Per aconseguir això es pretén dissenyar una arquitectura de comunicació, entre els diversos jugadors mitjançant els serveis de xarxa i tecnologies de comunicació disponibles.

L'arquitectura de xarxa s'haurà de dissenyar i implementar per tal de permetre la òptima comunicació entre els dos jugadors. Tanmateix s'estudiarà quin és el model de comunicació que més s'adapta al meu objectiu final, duent-lo a terme de forma eficaç, és a dir, aconseguint que aquest pugui integrar-se en l'acte comunicatiu i alhora, emmagatzemar dades i estadístiques de l'aplicació.

Així doncs, per la meva aplicació serà necessari la implementació d'una part client i de una part servidor, sent la part client evidentment el dispositiu mòbil i la part servidor la que faci de pont entre els clients, oferint-los-hi els serveis corresponents.

Al tractar-se de un joc, també serà important treballar amb un dels diferents motors gràfics disponibles per la plataforma Android, per tal de visualitzar l'evolució de la partida i permetre als dispositius interactuar entre ells mitjançant una mecànica de joc.

També serà primordial crear i desenvolupar una mecànica de joc que s'adapti a les necessitats del nostre joc en concret, permetent un sistema dinàmic de torns de 3 segons que aprofiti les funcionalitats del hardware.

Un dels altres objectius, és poder realitzar la gestió de les partides als usuaris, permetent-los-hi crear les seves pròpies partides o bé buscar les diverses partides disponibles en un determinat moment.

Per tal de facilitar la comunicació entre els diversos jugadors, s'implementarà també un sistema de missatgeria intern, donant una importància central a les xarxes socials, permetent el compartir, convidar i agregar entre altres opcions.

# 2. ESTUDI DE VIABILITAT

Per tal de poder realitzar aquest projecte, els paràmetres principals que s'han agut de tenir en compte han estat, principalment, l'elecció de quines tecnologies, eines, *frameworks* i llibreries són necessàries per la resoldre totes les especificacions que tindria que complir el meu projecte. D'aquesta manera, ha calgut també realitzar un estudi previ d'aquestes eines i de les seves respectives alternatives, i fer una recerca dels seus pros i contres per tal de veure quina és la que més s'ajusta a les necessitats finals.

Per altre banda, he agut de definir el meu criteri d'elecció de cada tecnologia, seguint una sèrie de passos i/o premisses que resultaven importants per respectar les especificacions anteriorment definides.

La tecnologia seleccionada ha agut de complir les següents condicions descrites en ordre d'importància:

1. La eina havia de ser OpenSource, permetent disposar de la seva versió full sense la necessitat de pagar cap tipus de llicència.
2. Tenia que ser possible la seva integració amb l'entorn de desenvolupament, més concretament l'SDK d'Android.
3. Era important també que aquestes oferissin també una àmplia documentació o disposessin d'una forta comunitat que facilités la correcta formació en la tecnologia per tal de resoldre els possibles problemes que es poguessin anar trobant durant el seu desenvolupament.

La avantatge d'utilitzar eines OpenSource no es basa únicament en la vessant econòmica, sinó que tradicionalment ofereix altres característiques que identifiquen aquest tipus de software. Ens ofereixen la possibilitat d'utilitzar-lo per qualsevol propòsit que ens interessi, sense cap tipus de limitació més que l'abast del mateix. També ens permet estudiar el seu funcionament en profunditat, transparentant les seves desavantatges i els seus punts forts, el que ens permet determinar la seva utilitat per tal de complir amb els meus objectius i d'aquesta manera poder-lo adaptar a les nostres pròpies necessitats ampliant-lo o modificant-lo segons escaigui.

Un altre gran avantatge és que el seu funcionament i desenvolupament, el qual està suportat per unes comunitats compromeses, que estant constantment realitzant tasques de *feedback* a darrera i oferint la seva visió personal i modificacions diverses,

convertint-les en eines amb una forta escalabilitat, fàcil manteniment i de intuïtiva modificació. Aquest aspecte aporta seguretat i estabilitat al software, impulsant els forts d'aquest tipus de distribucions.

Per dur a terme el desenvolupament del joc, he fet ús del propi SDK d'Android els qual ja aporta un emulador integrat, l'Android Virtual Device Manager (AVDM), que permet emular la nostra aplicació en un nombre variat de models de smartphones amb diferents resolucions i tipus de processadors i memòries. No obstant, tot i les seves complertes funcionalitats, ha fet falta disposar de dispositius mòbils amb sistema operatiu Anrdroid per poder testejar, de manera més tangible, l'aplicació, ja que l'AVDM resulta lent en algunes ocasions, depenent de les tecnologies utilitzades, no ofereix un correcte rendiment i funcionament..

Per els testejos i les proves he utilitzat un Samsung Galaxy II i un Sony Xperia P, dispositius que funcionen sota diferent versions Android i que disposen d'una resolució diferent. En aquest sentit, ha estat molt important des de un principi, la inversió d'un temps important en la investigació, recerca i formació, estudiant cada una de les tecnologies/eines concretes amb les que s'han agut de treballat, que amb permetessin complir amb els objectius i implementar cada una de les funcionalitats descrites anteriorment.

**ESTUDI DEL PRESSUPOST**

En referència al pressupost necessari que he agut de requerir per el projecte, aquest s'ha basat sobretot en els següents costos fixes:

| | |
|---|---|
| **Mòbil Sony Xperia P** | 250€ |
| **Samsung Galaxi SII** | 392€ |
| **PC equipat amb 2 pantalles** | 1500€ |
| **120 dies de treball/ 8 hores el dia** | 20€/h= 19.200€ |
| **TOTAL** | **21.342€** |

Als anteriors s'hi afegeixen d'altres variables, com per exemple, el lloguer del despatx, las factures de llum, les contribucions autàrquiques... que farien pujar el preu de venta final de l'aplicació, valorada en 21.342€, a la qual encara li afegiríem l'IVA del 21%.

# 3. METEDOLOGIA

Durant el transcurs d'aquest projecte, ha sigut necessari utilitzar una metodologia de desenvolupament que permetés definir i estructurar cada una de les tasques a realitzar, de manera que fos possible seguir una estructura lògica que facilités el seguiment per part del meu tutor. Aquesta havia de permetre alhora certa flexibilitat alhora de definir aquestes tasques davant dels possibles problemes, modificacions i imprevistos que ens poguéssim anar trobant durant aquest període de desenvolupament i d'aquesta manera vàrem decidir utilitzar la SCRUM.

## 3.1. QUÈ ÉS SCRUM?

Ideada el 1986 per Hirotaka Takeuchi i Ikujiro Nonaka, va ser pensada com una metodologia de desenvolupament amb l'objectiu principal de incrementar la velocitat i la flexibilitat. Va sorgir per la necessitat de solucionar alguns errors comuns en el procés de desenvolupament normal, com per exemple:

- **Caos/Desordre degut als canvis en els requeriments:** Els requeriments inicials canvien de forma dràstica des de el moment en que el projecte és dissenyat fins que es finalitza. En la gran majoria de les metodologies de desenvolupament, tot el disseny es realitzava a l'inici del projecte i no estava permès realitzar canvis en el mateix quan els requeriments canviaven durant el període de desevolupament.

- **Estimacions poc realistes de temps, cost i qualitat del producte:** El cap del projecte i els desenvolupadors tendeixen a subestimar la quantitat de temps i recursos que pot portar la realització d'un determinat estudi i quin nombre de funcionalitats es poden arribar a produir dintre aquestes etapes. En l'actualitat, aquests aspectes no poden ser predits amb exactitud en la seva fase inicial.

- **Els desenvolupadors estan forçats a mentir sobre el progres d'un projecte:** Quan el cap de projecte subestima el temps i el cost necessari per arribar a un cert nivell de qualitat, els desenvolupadors tenen per tant que mentir sobre el progrés realitzat davant del client o enfrontar-se amb la indignació del cap del projecte.

Partint d'aquests errors, la metodologia SCRUM es defineix com una alternativa per tal d'evitar-los i permetre una percepció més honesta de les etapes de les que es composa un projecte i de les tasques a realitzar.

En SCRUM presenta entregues parcials i regulars durant tot el projecte, donant sempre prioritat a les tasques que aporten més benefici al desenvolupador i al receptor del producte. Normalment, s'utilitza en entorns on existeix la necessitat d'obtenir resultats de forma àgil i on els requisits són canviants o estan poc definits, donant d'aquesta manera importància a altres aspectes com la innovació, flexibilitat, competitivitat i productivitat.

També s'utilitza per resoldre situacions en que no s'està entregant al client el que necessita, quan les entregues s'allarguen massa , els costos es disparen, la qualitat no és acceptable, quan es necessita la capacitat de reacció davant de la competència, quan és necessari identificar i solucionar ineficiències sistemàticament o quan es vol treballar utilitzant un procés especialitzat en el desenvolupament de producte.

## 3.2 ARQUITECTURA DE SCRUM

SCRUM és un model de referencia que defineix un conjunt de pràctiques i rols, els quals es poden designar com a punt de partida per definir el procés de desenvolupament. Es composen per a diferents rols que representen i dibuixen cada una de les figures que participen en el projecte, de manera que, des de un principi queda pautat quin és el paper de cada un en el mateix.

### a) ROLS

Els rols principals serien:

**Product Owner:** Representa la veu client o propietari, i és l'encarregat d'escriure les *Històries,* les prioritza i les col·loca *al Product Backlog.*

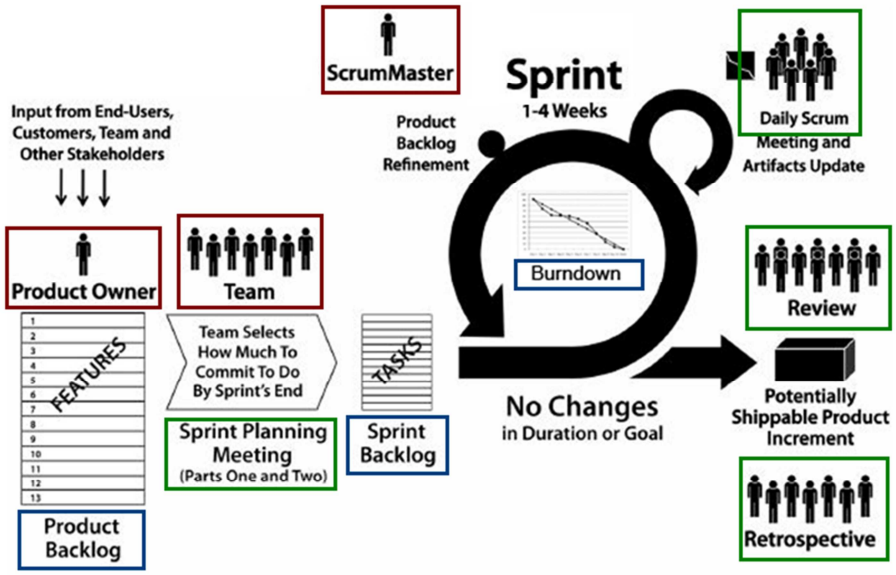
**Scrum Master:** Juga el paper de coordinador de l'Scrum, amb l'objectiu que les regles i limitacions es compleixin.

**Equip de desenvolupament:** És l'encarregat de les etapes de anàlisi, disseny, desenvolupament, festeig, etc... I el responsable de l'entrega del producte final.

### b) *SPRINTS*

SCRUM defineix blocs temporals petits i fixes durant el projecte, normalment espais de temps que poden variar d'entre dues setmanes o un mes, els quals són, en aquest últim cas, anomenats *Sprints*. Al final de cada una d'aquestes iteracions l'objectiu és el d'oferir un resultat complet i funcional de forma que es pugui entregar al client com una part acabada del projecte.

Les característiques que formen cada un dels *Sprints*, venen donades per el *Product Backlog,* que defineix el treball a realitzar durant cada etapa. Els elements que composaran el *Product Backlog,* es determinen durant la reunió de *Sprint Planning*, on el *Product Owner* identifica els elements que el composaran i que hauran d'estar finalitzats al final de cada *Sprint*. Durant la duració d'un *Sprint*, els requisits no podran ser modificats i per tant qualsevol canvi haurà d'esperar a ser introduït en els *Sprints* posteriors.




**Imatge 1 - Esquema de funcionament dels Sprints.**

### c) *PRODUCT BACKLOG*

El *Product Backlog,* conté les descripcions genèriques de tots els requeriments que ha de complir el projecte, de les funcionalitats que haurà d'incloure... sent per tant necessari obtenir una llista dels requisits/objectius prioritaris per ordre d'importància, que d'alguna manera s'utilitza com a pla del projecte. Així doncs, el *Product Backlog* conté tot el que al final haurà d'estar desenvolupat i es pot anar modificant durant el transcurs del mateix, sempre i quan, no s'enfronti a les necessitats i especificacions definides pel client.

A nivell professional, normalment aquesta priorització és realitzada amb l'ajuda del client que, sent aquest el que marca tots els objectius en relació al valor, respecte el cost amb l'objectiu, finalment, de dividir-los en entregues o *Sprints*. El client durà un seguiment regular del desenvolupament del producte de manera que en tot moment té una visió del que s'està desenvolupant i en cas de que ho vulgui, pot replantejar objectius i modificar el que cregui necessari en iteracions futures. Això és molt útil, ja que és habitual que durant el projecte que el client tingui la necessitat de canviar la idea del que vol i del que necessita. De fet, aquest és un dels principis clau de SCRUM, reconèixer i tindre en compte que durant la duració del projecte els clients o compradors poden canviar la idea del que volen i el que necessiten, i que aquests canvis imprevisibles no poden ser adreçats fàcilment en una perspectiva tradicional o d'una forma planificada. Per tant SCRUM es centra en maximitzar l'habilitat de l'equip per respondre de forma ràpida i reaccionar davant dels esdeveniments emergents que puguin sorgir.

### d) ETAPES D'UN *SPRINT*

L'*Sprint* es pot dividir en vàries etapes:

1. **Planificació de l'*Sprint*:** A l'inici de cada *Sprint*, s'ha de seleccionar quines tasques o feina es duran a terme, caldrà estimar el temps que es trigarà en realitzar aquestes tasques, també serà important identificar en quina quantitat del projecte es realitzarà durant aquest *Sprint*. És important també fixar-se un temps com a límit.
2. **Revisió de l'*Sprint*:** Un cop es finalitza un *Sprint*, caldrà realitzar un anàlisis del propi *Sprint*, analitzant la feina que s'ha pogut realitzar, la que no i quins han set els seus motius. Durant aquesta etapa és habitual presentar el treball realitzat fins llavors als interessats.
3. **Retrospectiva de l'*Sprint*:** Després de cada *Sprint*, es du a terme una retrospectiva de l' *Sprint*, en el qual tots els membres del projecte o de l'equip deixaran les seves impressions sobre l' *Sprint* recent superat. El propòsit principal de la retrospectiva és el de realitzar una millora continua del procés.

### e) SCRUM EN EL MEU PROJECTE

La premissa sobre la que parteix SCRUM de que durant la duració del projecte les funcionalitats i/o especificacions poden canviar segons necessitats o imprevistos és el que ha marcat la meva decisió final, conduint-me a utilitzar aquesta metodologia.

Durant les especificacions inicials del projecte, sabia que hi haurien etapes en que potser seria necessari enfrontar-se a certs imprevistos o canvis deguts a causes externes o al replantejament d'alguna de les tasques definides al principi. Al tindre que utilitzar durant el transcurs de desenvolupament llibreries o tecnologies, que desconeixia com s'integrarien a la plataforma Android, era necessari poder introduir, en cas de problemes o modificacions, els canvis corresponents de la manera més àgil possible.

Un altre dels factors determinants en la seva elecció, és el fet que es presenti com una metodologia senzilla i efectiva, fàcil d'aprendre i que es pot començar a utilitzar quasi de forma immediata.

Una de les seves principals avantatges que vaig tenir l'oportunitat de tangibilitzar és que un cop vaig tenir definides les especificacions i un llistat de totes les funcionalitats i tasques a definir, vaig poder marcar-me les fites específiques a desenvolupar primer i els objectius i dates per presentar-les, els quals em van ajudar a centrar-me en cada una d'aquestes etapes, tenint en compte que cada una d'elles tenien que oferir una part funcional del meu projecte.

# 4. PLANIFICACIÓ

La planificació del meu projecte pot ser dividida en diferents etapes que recullen tots els esforços realitzats durant el mateix, des de el seu plantejament inicial i proposta fins la redacció d'aquesta memòria. Així doncs es podria dividir en 5 etapes:

1) Elecció i proposta del P/TFC
2) Anàlisi de l'entorn de desenvolupament Android i de les llibreries
3) Formació en cada un dels anteriors elements
4) Desenvolupament de l'aplicació
5) Redacció de la Memòria i anàlisi dels resultats.

## 4.1 ELECCIÓ I PROPOSTA DEL P/TFC

Durant aquest primera etapa es va escollir la temàtica del meu projecte. Un cop determinat que aquesta tractaria sobre el desenvolupament d'un joc per plataformes mòbils o tablets, va ser necessari de idear els requeriments funcionals i no funcionals que aquesta hauria d'incloure.

Sabent de quin tipus de funcionalitats disposaria la meva aplicació, i definits els seus elements principals va ser necessari de buscar un tutor que m'orientés i m'aconsellés amb certes decisions que s'haurien de prendre.

Al final d'aquesta primera etapa, es va tenir clar quin seria l'abast del projecte i els objectius que es volien assolir amb el mateix.

La seva duració va ser de dues setmanes, del 13 al 27 d'Abril.

## 4.2 ANÀLISI DE L'ENTORN DE DESENVOLUPAMENT ANDROID I DE LES LLIBRERIES I TECNOLOGIES UTILITZADES

Un cop definit el projecte i els seus objectius, la següent etapa va constar d'un període de formació i de recerca.

Sabent per quina plataforma es desenvoluparia, va ser necessari documentar-se i formar-se en la creació d'aplicacions per Android, estudiant la seva arquitectura i funcionament.

Es va realitzar una recerca dels diferents motors gràfics disponibles per la plataforma, posant especial ímpetu en els que suportessin partides multi-jugador i tinguessin una bona quantitat de documentació disponible per el seu posterior estudi. També va ser necessari l'estudi de diversos aspectes que em serien d'utilitat en la fase de disseny del meu sistema, com per exemple, el sistema de bases de dades, model de comunicació entre usuaris, etc...

Un cop realitzada aquesta recerca i estudi, ja em vaig poder fer a la idea de com dissenyaria el meu projecte i quins elements el compondrien.

Aquesta etapa va tenir una duració de dues setmanes també, des de el 29 d'abril fins al 13 de maig.

### 4.3 FORMACIÓ EN CADA UN DELS ANTERIORS ELEMENTS

En aquesta etapa ja es tenia clar quin seria el disseny i l'arquitectura del meu projecte, i per tant també sabia per quines tecnologies em decantaria. Es basava doncs en l'estudi de totes aquestes i la en la integració entre elles.

Aquesta etapa es va iniciar directament junt amb la de desenvolupament de l'aplicació.

### 4.4 DESENVOLUPAMENT DE L'APLICACIÓ

Com ja s'ha comentat durant el període de desenvolupament, es va utilitzar la metodologia SCRUM per tal d'organitzar les tasques a organitzar. El primer que es va fer va ser llistar totes les tasques a realitzar per a continuació distribuir-les en *Sprints* de dos setmanes:

| 1er *Sprint* (Del 15 al 29 de Maig) | Tipus | Temps estimat |
|---|---|---|
| **Implementació del servidor amb *nodejs* i socket.io** | Tasca | 4h |
| **Instal·lació de la base de dades amb *MongoDB* i *Mongoose*** | Tasca | 1h |
| **Estructura de l'aplicació Android** | Tasca | 2h |
| **Activitat *Introduction*** | Tasca | 2h |
| **Activitat *Main Menu*** | Tasca | 2h |
| **Implementar *websockets* amb android mitjançant la llibreria *java IOSocket*** | Tasca | 4h |

**Objectiu de l'*Sprint*:** L'objectiu d'aquest primer *Sprint* era tenir la nostra aplicació Android comunicant-se amb el servidor *nodejs*.

| 2on *Sprint* (Del 30 de Maig al 13 de Juny) | Tipus | Temps estimat |
|---|---|---|
| **Dissenyar i implementar els models de dades de *users i matches* amb *Mongoose*** | Tasca | 2h |
| **Activitat de *LoginForm*** | Tasca | 8h |
| **Activitat de *RegistratioForm*** | Tasca | 8h |
| **Implementar el recordatori del compte d'usuari amb *sendmailer*** | Tasca | 3h |
| **Integrar un projecte Libgdx amb l'SDK d'Android** | Tasca | 4h |

**Objectiu de l'*Sprint*:** Al final d'aquest *Sprint*, ja podíem donar d'alta un usuari guardant les seves dades a *MongoDB*, i permetre-li accedir a l'aplicació amb el seu compte.

| 3er *Sprint* (Del 14 al 28 de Juny) | Tipus | Temps estimat |
|---|---|---|
| **Implementar les *Screens* del joc** | Tasca | 8h |
| **Permetre encriptar la contrasenya dels nostres usuaris** | *Issue* | 2h |
| **Afegir els actors necessaris dintre el joc** | Tasca | 8h |
| **Animar les diferents textures** | Tasca | 8h |
| **Comunicació de *libgdx* amb el servidor** | Tasca | 6h |

**Objectiu de l'*Sprint*:** Al final d'aquest *Sprint*, teníem definits els actors i *screens* del nostre joc i el podíem animar els actors segons determinats esdeveniments. No es va tenir temps d'implementar la comunicació de *libgdx* amb el servidor, amb le qual la tasca es va passar al següent *sprint*.

| 4rt *Sprint* (Del 29 al 13 de Juliol) | Tipus | Temps estimat |
|---|---|---|
| **Comunicació de *libgdx* amb el servidor** | Tasca | 6h |
| **Implementar les activitats de *PersonalOptions*** | Tasca | 12h |
| **Implementar la detecció de moviments amb *libgdx*** | Tasca | 10h |
| **Implementar les activitats de *FriendsList*** | Tasca | 12h |
| **Moure els actors mitjançant interaccions amb la pantalla** | Tasca | 8h |

**Objectiu de l'*Sprint*:** Al final d'aquesta etapa l'usuari podia veure i modificar les seves dades personals, es podien consultar i agregar amics. Ara ja també podíem moure i animar els actors interactuant amb la pantalla.

| 6è *Sprint* (Del 14 al 28 de Juliol) | Tipus | Temps estimat |
|---|---|---|
| **Canviar la llibreria *IOSocket* per *SocketIO-Client*.** | *Bugfixing* | 8h |
| **Permetre realitzar partida entre dos jugadors** | Tasca | 12h |
| **Implementar la mecànica de joc** | Tasca | 5h |
| **Implementar les activitats de *FightRoom*** | Tasca | 12h |
| **Permetre mostrar missatges dels *events* al joc** | Tasca | 8h |

**Objectiu de l'*Sprint*:** Era poder crear i buscar partides creades pels usuaris i poder-se comunicar entre ells mitjançant la sala de chat. Pel que fa al joc ara es mostraven els missatges dels esdeveniment*s* que es duen a terme durant la partida. També es va tindre que solucionar un problema amb la llibreria *IOSocket*, que no era compatible amb les versions més antigues d'Android i es va canviar per *SocketIO-Client*.

| 7è *Sprint* (Del 29 de Juliol al 12 d'Agost) | Tipus | Temps estimat |
|---|---|---|
| **Implementar un servei per gestionar la connexió de les activitats amb el servidor.** | Tasca | 6h |
| ***Submission events* en el joc** | Tasca | 10h |
| **Implementar *Timer*** | Tasca | 4h |
| **Afegir sistema de puntuació** | Tasca | 4h |
| **Estructura d'estats dels actors del joc** | Tasca | 20h |

**Objectiu de l'*Sprint*:** Al final d'aquest S*print l*es activitats de l'aplicació es comunicaven amb el servidor per mitjà d'un servei. Ara la partida disposava d'un *timer* que controlava el temps de la partida, es contaven els punts segons les accions i s'havia dissenyat la estructura d'estats dels actors.

| 8è *Sprint* (Del 13 al 27 d'Agost) | Tipus | Temps estimat |
|---|---|---|
| **Afegir animacions als actors** | Tasca | 24h |
| ***Events* de finalització de partides** | Tasca | 10h |
| **Guardar els resultats de la partida** | Tasca | 4h |
| **Integrar projecte *libgdx* amb l'aplicació Android** | Tasca | 2h |

**Objectiu de l'*Sprint*:** Al final d'aquest *Sprint* el joc estava finalitzat i es guardaven els resultats de les partides al base de dades. Finalment es va integrar amb l'SDK d'Android.

### 4.5 REDACCIÓ DE LA MEMÒRIA I ANÀLISI DELS RESULTATS

La redacció de la memòria es va dur a terme durant tot el procés de realització del projecte en paral·lel amb totes les altres etapes esmentades anteriorment, de manera que va tenir una durada de 4 mesos i mig, des de Abril fins a finals d'Agost.

# 5. MARC DE TREBALL I CONCEPTES PREVIS

En aquest capítol es presenten el conceptes previs que cal conèixer per tal de poder comprendre plenament el desenvolupament del projecte i les tasques realitzades.

A continuació s'ofereix una descripció dels principals conceptes de les tecnologies utilitzades, des de l'arquitectura d'Android, el funcionament dels motors gràfics per aplicacions mòbils (en especial Libgdx), introducció a *nodejs i a mongodb*, etc...

## 5.1 LA PLATAFORMA ANDROID

Android és un entorn de software creat per dispositius mòbils, que inclou un sistema operatiu basat en Linux, una completa interfase d'usuari, aplicacions, biblioteques de codi, estructures per aplicacions, compatibilitat multimèdia i moltes altres coses més.

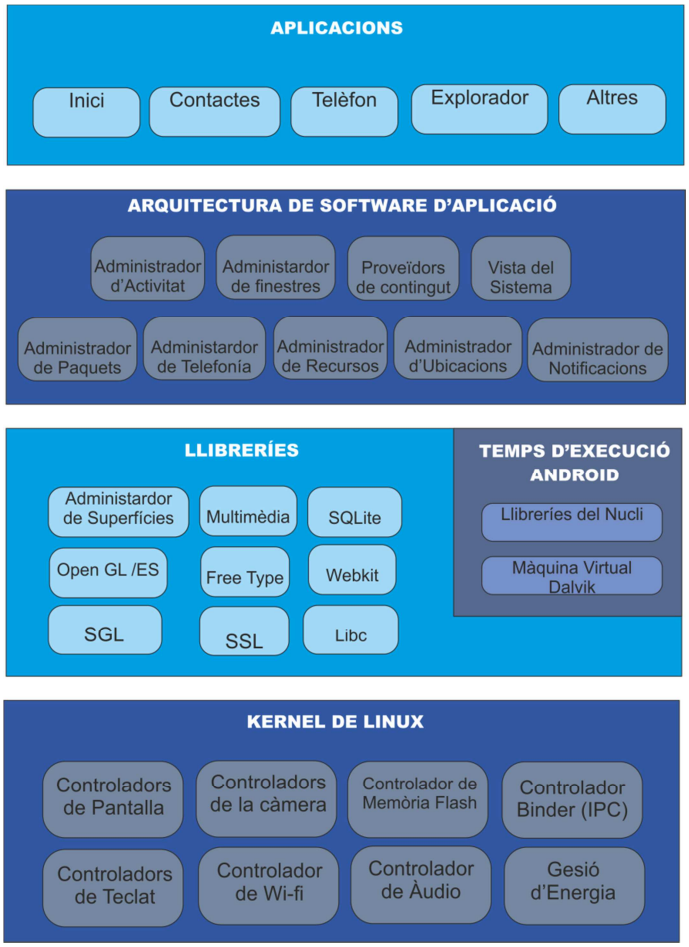
Tot i que els components del Sistema Operatiu subjacent s'escriuen en C o C++, les aplicacions per Android s'escriuen en Java. Inclús les aplicacions incorporades en el Sistema Operatiu són de Java.

Per tant no existeixen diferències entre les aplicacions incorporades i les creades amb l'SDK d'android, el que significa que es poden crear completes aplicacions per aprofitar els recursos disponibles en el dispositiu.

### 5.1.1. Arquitectura d'Android

L'arquitectura d'Android està formada per una pila de capes software que interactuen entre elles. En termes de desenvolupament, aquesta divisió en capes facilita al desenvolupador la creació d'aplicacions, ja que d'aquesta manera tot està perfectament estructurat per que es pugui accedir a les capes més baixes utilitzant les llibreries disponibles per això, evitant així tenir que programar a baix nivell les funcionalitats necessàries per el desenvolupament de una aplicació.

El Sistema Operatiu d'Android es divideix en 5 seccions, de 4 nivells principals:




**Imatge 2 -Arquitectura del Sistema Operatiu Android.**

Aquests són:

### a) El Kernel de Linux

El *Kernel* o nucli de Linux es pot definir com el cor del sistema operatiu, i és l'encarregat de que el hardware i el software del dispositiu puguin comunicar-se i treballar junts.

Algunes de les seves funcions més importants són:

- L'administració de memòria per tots els programes i processos en execució.
- L'administració del temps del processador per els programes i processos en execució.
- És l'encarregat de permetre l'accés als perifèrics i elements del nostre dispositiu d'una manera còmoda.

La capa més baixa de l'arquitectura de tot Sistema Operatiu és el nucli del sistema. En el cas d'Android aquest s'ajuda del *Kernel* de Linux (versió 2.6.x). En aquesta capa de l'arquitectura, Android té accés a la gestió de memòria i dels processos , la pila de xarxa i el model de *drivers*.

El nucli actua com una capa d'abstracció entre el hardware i la resta de les capes de l'arquitectura. El desenvolupador no accedeix directament a aquesta capa , en canvi té que utilitzar les llibreries disponibles en les capes superiors.

### b) Llibreries

Les llibreries que pot utilitzar el desenvolupador en les capes superiors estan escrites en C/C++. Les funcionalitats que ofereixen aquestes llibreries són accessibles des de el *framework* d'aplicacions. És a dir, un API de Java, que és el llenguatge que s'utilitza per programar aplicacions per Android.

Les més destacades són:

- **Administrador de superfícies:** És una llibreria encarregada de composar els diferents elements de navegació de pantalla. Gestiona també les finestres que pertanyen a les diferents aplicacions actives en cada moment.
- ***OpenGL/ES i SGL*:** Representen les llibreries gràfiques i per tant sustenten la capacitat gràfica d'Android. *OpenGL/ES* maneja gràfics en 3D i permet utilitzar, en cas de que estigui disponible en el propi dispositiu mòbil, el hardware encarregat de proporcionar gràfics 3D. Per l'altre banda, SGL proporciona gràfics en 2D, per el que serà la llibreria més habitualment utilitzada per la majoria de les aplicacions. Una característica important de la capacitat gràfica d'Android és que és possible desenvolupar aplicacions que combinin gràfics en 3D i 2D.
- **Llibreria multimèdia:** Aquesta llibreria ens proporciona tots els còdecs necessaris per el contingut multimèdia suportat en Android (vídeo, àudio, imatges estàtiques i animades ,etc...)
- ***Free Type*:** Permet treballar de forma ràpida i senzilla amb diferents tipus de fonts.
- **Llibreria SSL:** Possibilita la utilització del protocol SSL per tal d'establir connexions segures.
- ***SQLite*:** Permet la creació i gestió de bases de dades relacionals.
- ***Webkit*:** Proporciona un motor per les aplicacions de tipus navegador inclòs per defecte en la plataforma Android.

- **Llibreria *libc*:** Inclou totes les capçaleres i funcions segons l'estàndard del llenguatge C. Totes les altres llibreries es defineixen en aquest llenguatge.

#### c) Temps d'execució de Android (O entron d'execució d'android)

El sistema inclou una màquina virtual de Java (JVM), que s'anomena *Dalvik*, que ha set creada per Google per dispositius amb poca memòria i poca capacitat de procés. En Android, cada aplicació corre en el seu propi procés i té la seva pròpia instància en la màquina virtual *Dalvik*.

*Dalvik* executa *arxius .dex* en comptes dels clàssics *.class* de la màquina virtual de Java d'escriptori. Aquests estan més optimitzats per els dispositius mòbils i són més compactes. En aquesta màquina virtual no disposen de tota la API de *JavaSE* o JavaME, sino que podem utilitzar un subconjunt anomenat *Core Libraries*.

### 5.1.2. Anatomia de les aplicacions

Una aplicació Android corre dintre del seu propi procés Linux, per tant, una característica fonamental d'Android és que el temps y cicle de vida d'una aplicació no està controlat per la mateixa aplicació sinó que ho determina el sistema a partir

Es pot diferenciar cinc elements principals que constitueixen les aplicacions d'Android:

- Les Activitats
- Els Intents
- El *Broadcast Intent Reciever*
- El Servei
- L'*Intent Provider*

Una aplicació en Android té que declarar totes les seves activitats, els punt d'entrada, la comunicació, les capes, els permisos i els intents a través de l'arxiu AndroidManifest.xml. És molt important tenir en consideració com aquests components impacten en el temps de vida del procés associat amb una aplicació, perquè si no són usats de forma apropiada, el sistema detindrà el procés de l'aplicació encara que s'estigui fent alguna cosa important.

#### a) Activitats

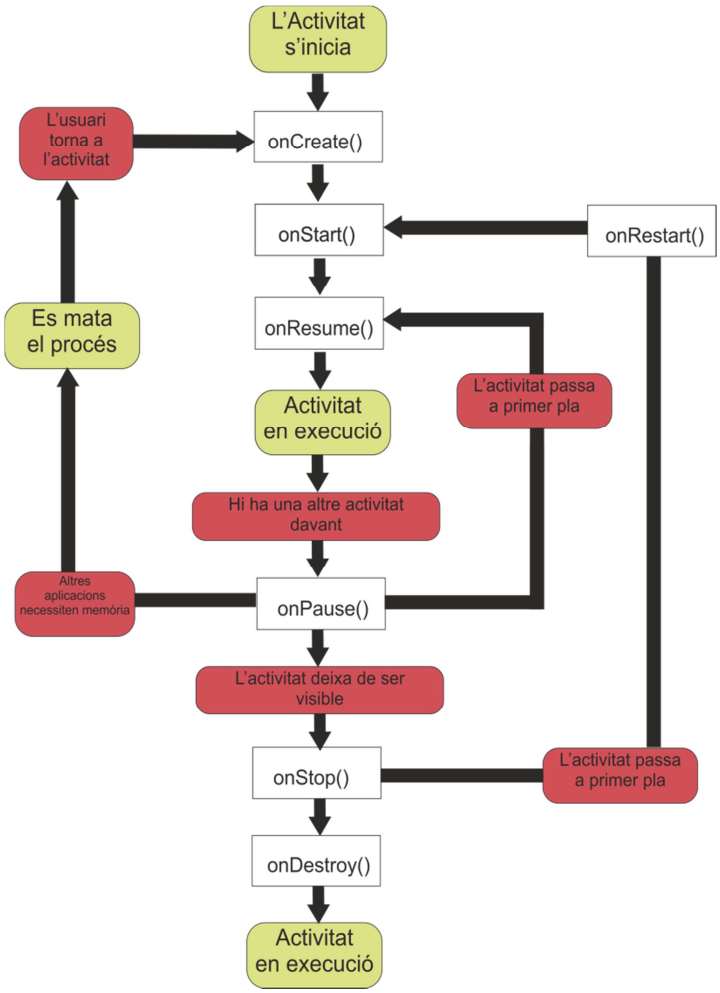
Les activitats són els elements més comuns dels blocs de construcció de les aplicacions, i representen una determinada activitat duta a terme per una aplicació i que porta associada típicament una finestra o interfase de usuari.

No contemplen únicament l'aspecte gràfic, sinó que aquest forma part del component Activity a través de vistes representades per classes com View (Vista) o els seus derivats.

Cada activitat s'implementa com una sola classe que s'estén a la classe base Activity. Aquesta classe mostrarà una interfície composta de vistes mitjançant arxius XML i respondrà a diversos events, ja que la majoria d'aplicacions permeten l'execució de varies accions a través de la existència de una o més pantalles.

### b) Cicle de vida de una activitat

Totes les activitats tenen un cicle de vida definit, que Android permet controlar, de manera que quan es dissenya una aplicació s'ha de tenir en compte aquest cicle de vida. Desde el moment que apareix una activitat en pantalla fins que s'oculta, aquesta passa per set passos.




**Imatge 3 - Els 7 passos del Cicle de Vida d'una Activitat.**

La classe base *Activity* defineix una sèrie d'esdeveniments que governen el cicle de vida de una activitat. Aquests són:

- **onCreate():** Quan l'activitat es crea per primer cop.
- **onStart():** Quan una activitat s'executa i es troba visible.
- **onResume():** Quan l'activitat comença a interactuar amb l'usuari.
- **onPause():** Quan l'activitat actual s'atura i l'activitat anterior es reinicia.
- **onStop():** Quan l'activitat ja no és visible per l'usuari.
- **onDestroy():** Abans que el sistema destrueixi la activitat.
- **onRestart():** Quan la activitat s'ha detingut i es reinicia de nou.

```
public class ExampleActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // The activity is being created.
    }
    @Override
    protected void onStart() {
        super.onStart();
        // The activity is about to become visible.
    }
    @Override
    protected void onResume() {
        super.onResume();
        // The activity has become visible (it is now "resumed").
    }
    @Override
    protected void onPause() {
        super.onPause();
        // Another activity is taking focus (this activity is about to be "paused").
    }
    @Override
    protected void onStop() {
        super.onStop();
        // The activity is no longer visible (it is now "stopped")
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        // The activity is about to be destroyed.
    }
}
```

### b) Intents

Android utilitza una classe anomenada Intent per moure's d'una pantalla a l'altre, que representa la voluntat de realitzar alguna acció, generalment associada a unes dades. Llançant un Intent, una aplicació pot delegar el treball a una altre, de manera que el sistema s'encarrega de buscar quina aplicació de entre les instal·lades es la que pot dur a terme l'acció sol·licitada

Els principals components d'un Intent són l'acció que es vol executar, i les dades sobre les quals ha d'operar.

Exemple d'execució d'un intent en que l'acció és iniciar una activitat i les dades seria la classe que fa referència a aquesta activitat:

```
case R.id.startgame:
    startActivity(new Intent(getApplication(),FightRoom.class));
    break;
case R.id.gameoptions:
    startActivity(new Intent(getApplication(),PersonalOptions.class));
    break;
case R.id.sharefacebook:
    break;
case R.id.friends:
    startActivity(new Intent(getApplication(),FriendsList.class));
    break;
case R.id.quitgame:
    //Close the aplication and return to the Home
    dialeg = QuitJiuJitsu_Battle();
    dialeg.show();
    break;
}
```

### *c) Broadcast intent provider & reciever*

Els *Broadcast* intents són objectes Intent que són emesos a altres activitats mitjançant els mètodes *sendBroadcast(),sendStickyBroadcast() o sendOrderedBroadcast()* d'una Activitat. S'utilitzen principalment com a sistema d'events i de missatgeria entre activitats.

Quan es crea un *Broadcast Intent*, té que incloure un *string* d'acció, a més de les dades opcionals i un *string* de categoria. Les dades s'afegeixen al broadcast intent utilitzant una parella *key-valor* mitjançant el mètode *putExtra()* per finalment, acabar sent enviades.

Exemple:

```
Intent intent = new Intent();
intent.setAction("com.example.Broadcast");
intent.putExtra("HighScore", 1000);
sendBroadcast(intent);
```

Aquestes dades poden ser recollides per les activitats que tinguin implementat un *Broadcast Intent Reciever*:

```
private BroadcastReceiver broadcastReceiver = new BroadcastReceiver()
    @Override
    public void onReceive(Context context, Intent intent) {
     updateUI(intent);
    }

};
```

### *d) Service*

Un servei en Android és un component d'aplicació que s'utilitza perquè una o més activitats puguin realitzar determinades accions o tasques mentre no estan interactuant amb l'usuari.

Un servei no és necessàriament un procés o un *thread*, a no ser que realitzin una tasca intensiva sobre la CPU del dispositiu, un servei s'executa sobre el *thread* principal com totes les activitats.

Per tal d'iniciar un servei, des de una activitat, és necessari utilitzar el mètode *this.startService()* o *this.bindService().*

- *startService(),* s'utilitza quan es desitja que dugui a terme una acció o tasca de *background.*
- *bindService()*, s'utilitza quan es desitja que una aplicació comparteixi les seves funcionalitats amb altres aplicacions.

Exemple d'inici d'un Servei en Android des de una Activitat:

```
service_start = new Intent();
service_start.setClass(LoginForm.this,WebSocketService.class);
service_start.setAction(WebSocketService.START_ACTION);
startService(service_start);
```

Primer es crea un nou intent que serà el que contindrà el servei, llavors s'assigna la classe que el farà servir i la classe que referencia el servei a utilitzar. Finalment s'inicia el servei. Llavors per tal de rebre la informació que proporciona el Servei, l'activitat haurà d'utilitzar un *Broadcast Reciever*.

### e) Android *Manifest*

L'arxiu AndroidManifest.xml es troba present en totes les aplicacions Android, i conté la informació necessària que s'ha de proporcionar al sistema operatiu per tal que es pugui executar, des de Activitats, Intents, Fragments, Services, permisos d'aplicació, etc...

Exemple de AndroidManifest.xml:

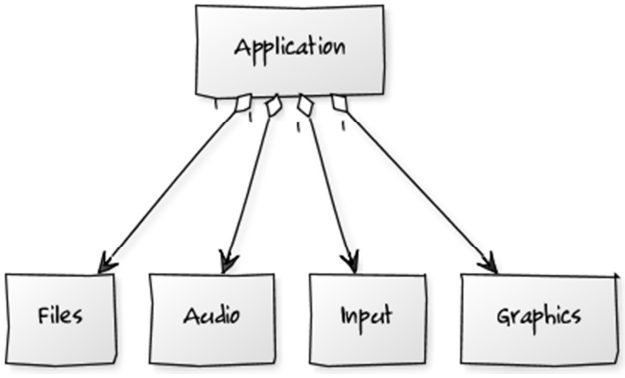
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.blackbeltgameinc.bjjheros"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="16" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.blackbeltgameinc.bjjheros.Splash"
            android:label="@string/app_name" >
            <intent-filter>
               <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name="com.blackbeltgameinc.bjjheros.MainMenu" >
        </activity>
        <activity
            android:name="com.blackbeltgameinc.bjjheros.SelectTeam" >
        </activity>
    </application>

</manifest>
```

### 5.2 EL *FRAMEWORK LIBGDX*

*LibGDX* és un *framework* multi-plataforma basat en *Java i OpenGL*, pensat exclusivament per la creació i desenvolupament de videojocs.

Ens proporciona diferents mòduls que ens facilita la creació dels nostres jocs, com gestió d'arxius, d'àudio, esdeveniments I/O i gestió de gràfics en forma de textures:

En aquest punt es presenten alguns conceptes previs per tal d'entendre el seu funcionament.




**Imatge 4 – Esquema de la relació de l'aplicació del Libgdx i els seus mòduls.**

### 5.2.1 Arquitectura d'una aplicació Libgdx

La classe principal de una aplicació Libgdx s'estén de *com.badlogic.gdx.ApplicationListener o de com.badlogic.gdx.Game*

Aquesta classe serà la encarregada de tractar determinats esdeveniments de la aplicació, com són *create, render, dispose,* etc… Aquests executats en l'escriptori no tenen el mateix comportament que quan s'executa el joc amb Android, però són molt similars.

La següent llista mostra els esdeveniments del cicle de vida d'aplicacions que poden ser disparats, i quan succeeix cada un:

- ***Create* –** S'executa un cop quan l'aplicació és creada.
- ***Resize* –** S'executa quan la pantalla del joc es redimensiona i el joc no està en estat de pausa. També s'executa un cop just després de l'*event create*.
- ***Pause* –** S'executa just abans que l'aplicació es destrueixi. En Android passa quan es prem el botó '*Home*' o arriba una trucada entrant. És un bon moment per guardar l'estat del joc en Android ja que no es garanteix que es resumeixi el mateix.
- **Render:** S'executa per el bucle de l'aplicació cada cop que es produeix el *rendering*. L'actualització del joc s'ha de dur a terme després de l'actual.
- ***Resume*:** només en android, quan l'aplicació rep el focus.
- ***Dispose*:** Quan l'aplicació es destrueix. Està precedida per l'estat de pausa.

Per tal de poder capturar i rebre aquests esdeveniments, es tindran que implementar aquests mètodes de la interfície *com.badlogic.gdx.ApplicationListener.* Quan libgdx crea el nostre joc , es crea un *thread* anomenat *Main loop* i el context de l'*OpenGL* s'uneix a ell. Tot el processament d'events s'executa dintre d'aquest fil separat, i no en el subprocés de la interfície d'usuari.

### 5.2.2 Les *Game Screens*

Les screens representen les pantalles de l'aplicació, on es volca tota la part gràfica de la mateixa, normalment se separen de la classe AplicationListener per tal de tenir separades les elements de control dels de pantalla.

Per tal d'aconseguir això en libgdx, es pot modificar la classe principal del joc per extendre *com.badlogic.gdx.Game,* la qual a la vegada implementa *ApplicationListener.*

Fent això, tindrem el suport de la interfície *com.badlogic.gdx.Screen,* que delega els mètodes de captura d'esdeveniments a una pantalla específica en cada temps.

D'aquesta maner la classe Game serà l'encarregada de crear les *screens* i activar-les (recordem, només una en cada moment). Cada instància de la pantalla serà l'encarregada de dibuixar-se ella mateixa. Això és molt útil perquè li dona a cada pantalla un abast limitat, el que facilita el manteniment del codi.

### 5.2.3 El paràmetre delta

El paràmetre delta s'utilitza per:

1) El jugador té un *smartphone* molt humil que només pot processar 10 *frames* per segon (FPS) del nostre joc.
2) El jugador té un dispositiu *quad-core* de última generació, capaç de processar fins a 100 *frames* per segon.

Per tal de poder definir la mateixa noció de temps en ambdós dispositius, s'utilitza el paràmetre delta en els seus càlculs matemàtics. Això assegura que el dispositiu de gama alta del nostre segon escenari no mostrarà el joc 10 vegades més ràpid que el primer dispositiu, però només la sortida de de més *frames* permetrà tenir una experiència de joc més suau.

### 5.2.4 *Scene2D*

Scene2d, és una característica de *libgdx* que prové de plenes abstraccions per renderitzar components en 2D.

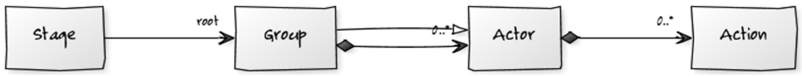
Scene2D és un mòdul de *libgdx* que facilita el manejat i el renderitzat de components 2D, que s'anomenen actors. Aquests actors viuen en una estructura dintre de un contenidor anomenat *Stage*. Ells segueixen la pista de diferents atributs de renderitzat, com una posició relativa al pare, color, visibilitat, dimensions, escala factors de rotació i molt més. Ells també són els responsables per la detecció de *hits*.

Exemples d'actors serien: botons, camps de text, imatges, enemics objectius, monedes, un avió que fem volar, trets, etc... Utilitzarem molt scene2d en el nostre joc. A més, és possible aplicar accions en els actors, com traslladar, rotar, escalar, o accions d'ocultar-se.

Els conceptes de scene2d serien:

- **Actor –** Un component 2D que sap com dibuixar-se.
- **Grup –** Un actor que conté altres actors.

- **Etapa –** Un motor que sol·licita els actors que es dibuixaran s'encarregarà de la interacció de l'usuari mitjançant la distribució dels events tàctils per els actors.
- **Acció –** Una funció que modifica les propietats dels actors en el temps.




**Imatge 5 - Diagrama UML de la relació.**

**Utilitzant Scene2d**

El primer és establir la Etapa (*Stage*) on els actors actuaran. Un bon lloc per implementar-la és dintre la pantalla (cada pantalla amb la seva pròpia etapa). Aquests serien el passos per manejar una etapa en el nostre joc:

1. Crear una instancia de l'Etapa en el constructor de la classe *AbstractScreen.*
2. Redimensionar el seu *viewpoint* quan la pantalla es redimensiona.
3. Cridar l'acte de l'Etapa i dibuixar mètodes dintre els mètodes de *screen* render.
4. *Desechar* la etapa en el mètode *Dispose* de la pantalla.

Un cop fet això, el que tenim que fer és afegir actors a la nostra etapa. Els nostres actors voldran sobreescriure mètodes com *act*() o *draw*(), que seran automàticament invocats per l'etapa en un determinat temps.

### 5.2.5 Les accions

Cada actor pot estar animat per accions. El que farem amb la nostra Imatge *Splash*, és afegir un conjunt de les següents accions:

1. Fade-in en 0.75 segons.
2. Esperar per 1.75 segons.
3. Desaparèixer en 0.75 segons.

Nosaltres també ens voldrem moure a la següent pantalla quan l'acció és completada, així que l'únic que tenim que fer, és afegir un *listener* a l'objecte accions.

Cal tenir en compte que no cal sobreescriure el mètode render() de Pantalla de *SplashScreen*, perquè la imatge *splash screen* és un actor que ha estat afegit a l'Etapa, i l'etapa és l'encarregada de renderitzar-lo.

Les accions que s'utilitzen en *libgdx* són:

- ***Concrete actions*** **–** Accions que modifiquen els actors directament (*FadeIn, FadeOut, MoveBy, RotateBy, ScaleTo*, etc).
- ***Support actions*** **–** Accions que agrupen o organitzen altres accions de forma específica (*Delay, Sequence, Parallel, Repeat, Forever*, etc).

Cada acció bé amb un mètode de fàbrica anomenat "$" que agafa els paràmetres requerits per treballar-hi correctament. Si vols crear una acció *FadeIn* que duri 2 segons:

```
FadeIn fadeInAction = FadeIn.$(2f);
```

La acció de *delay* té un mètode de fàbrica diferent. El codi següent crea l'acció *FadeOut* amb un *delay* de 5 segons:

```
FadeOut fadeOutAction = FadeOut.$(2f);
Delay delayAction = Delay.$(fadeOutAction, 5f);
```

Per tal de aconseguir l'efecte complet que volem, tenim que ajuntar ambdues accions:

```
Sequence sAction = Sequence.$(fadeInAction, delayAction);
```

I finalment, afegir l'acció a l'actor:

```
Actor.action(sAction);
```

Un fet important és que podem afegir interpoladors entre accions. Això vol dir que la nostra acció pot començar lenta i anar-se accelerant gradualment. Els Interpoladors defineixen el rang de canvi per una determinada animació.

### 5.2.6. Model del domini

El model del domini descriu les entitats, els seus atributs, rols i relacions per donat domini d'interès. La majoria de la nostra lògica de treball, romandrà en les entitats de domini, perquè en l'enginyeria de software és una molt bona idea de separar la lògica de treball. Fa el codi molt més directe a altres programadors, i inclús a nosaltres més tard. També ajuda quan canviem de tecnologies.

Amb l'ajuda de scene2D podem aconseguir això fàcilment, ja que cada una de les entitats dels dominis dibuixables pot mapejar un actor scene2d. Així és com separem el codi de treball del codi de presentació.

### 5.2.7 Sistema d'esdeveniments

Cada actor té una llista de *capture* i de *listeners* d'events. Ambdós *listeners* s'estenen la genèrica *com.badlogic.gdx.scenes.scene2d.EventListener.* Els *capture listeners* poden interceptar events abans que siguin agafats per l'*eventListene*r. Durant aquesta fase de captura, l'*event* és capturat des de l'arrel fins a l'actor objectiu. Si l'*event* no és cancel·lat per el *capture listener*, la fase normal s'activa i l'*event* és donat a l'actor objectiu fins a l'arrel de l'Etapa. Tenint en compte que la següent és possible:

- En comptes de consultar el mecanisme d'entrada per moure el nostre vaixell, podríem rebre els *events* d'entrada a mesura que succeeixen i modificar el nostre actor en conseqüència (*com.badlogic.gdx.scenes.scene2d.ActorListener*).
- Podem definir els nostres porpis *events*.
- Podem crear una jerarquia de detectors d'*events*, possiblement extenent un dels oyents enviats:
  - **ActorListener:** Escolta els *events* d'entrada, com *KeyDown /Up/Typed/, enter/exit, touchUp/Down* i així successivament.
  - **ChangeListener:** Escolta per canvis d'events, que és quan alguna cosa ha canviat en un actor.
  - **ActorGestureListener:** Fa fàcil de treballar amb *events* de gest, com *pinch zoom i fliing.*

Es poden reutilitzar event listeners afegint-los a diferents actors.

### 5.2.8. Animacions 2D

Com en altres *frameworks* de jocs, una animació 2d és una seqüència de imatges estàtiques (*frames*) mostrats amb una certa velocitat, creant una il·lusió de moviment.

**Utilitzant animacions:**

En *libgdx* utilitzem la classe com.badlogic.gdx.graphics.g2d.Animation per dur a terme animacions 2D. Aquesta classe guarda els *frames* de l'animació (En forma de instàncies de *TextureRegion*) i assigna el temps en el que cada *frame* tindrà que ser mostrat. Aquesta utilitat no dibuixa en absolut. Després de que s'hagi creat, utilitzarem el següent mètode:

```
public TextureRegion getKeyFrame(float stateTime, boolean looping)
```

Aquest mètode recupera el següent *frame* a ser dibuixat. Els paràmetres d'entrada són:

- *FloatStateTime*: Aquest és un valor acumulador que representa el temps passat des de que s'ha dut a terme la animació. Tenim que emmagatzemar aquest valor en algun lloc i afegir el temps delta a ell abans de cridar el mètode.
- *Boolean looping:* Si l'animació *bucla* o es para en l'últim *frame*.

Per tal de crear una instància d'animació, tenim que proporcionar les instàncies de *TextureRegion* i la duració del *frame* en segons. Com que utilitzem el *TexturePacker* per crear les nostres imatges *Atlas*, podem utilitzar la nostra instància *TextureAtlas* per trobar fàcilment tots els frames que composen l'animació només cridant textureAtlas.findRegions("*animation-name*"), el que retorna una llista de instàncies de *AtlasRegion*, que extent *TextureRegion.*

# 6. REQUISITS DEL SISTEMA

L'objectiu del meu projecte és el d'implementar una aplicació per la plataforma mòbil Android, que inclogui un joc multi-jugador (1 vs 1), el qual permeti gestionar i buscar partides, amics, comunicació mitjançant xarxes socials, etc... integrant-se en les diverses tecnologies OpenSource disponibles per la plataforma.

## 6.1 EL JOC

Aquest joc s'haurà d'integrar a l'aplicació mòbil mitjançant un dels diferents motors gràfics disponibles per Android, per tal de visualitzar el desenvolupament de la partida i permetre a l'usuari interactuar amb la mecànica de joc.

Per tant també serà necessari el dissenyar i crear una mecànica de joc que s'adapti a les meves necessitats i a les dels meus futurs usuaris. Serà important aprofitar les funcionalitats tàctils del dispositiu.

## 6.2 L'APLICACIÓ MÒBIL

L'aplicació mòbil, haurà de permetre el maneig i modificació de dades personals dels usuaris, gestió de partides, cerca d'amics, integració amb diferents xarxes socials i permetre la comunicació entre els usuaris de l'aplicació. Aquests requisits es detallen una mica més a continuació:

**a) <u>Maneig i modificació de dades personals</u>**

Per tal d'utilitzar l'aplicació els usuaris s'hauran de crear un compte dintre de la mateixa, registrant-se mitjançant un formulari. Gràcies a l'usuari i la contrasenya creats, i un cop dintre de l'aplicació podran modificar les seves dades personals i contrasenya. També s'haurà d'implementar un sistema per tal de que, en cas de que no recordin de les seves dades d'accés, les puguin recuperar.

**b) <u>Gestió de partides</u>**

Haurà de permetre als usuaris tant crear com cercar partides multi-jugador, i d'aquesta manera poder accedir a una partida ja creada per un altre usuari o bé realitzar d'amfitrió de la seva pròpia partida.

**c) <u>Integració amb xarxes socials</u>**

Una part molt important de les aplicacions mòbils actuals és la seva vessant comunicativa gràcies a l'ús de les xarxes socials, en que en tot moment pots informar del que estàs fent, estàs pensant, opines, etc... Per tant s'haurà d'integrar aquest aspecte en la nostra aplicació, de manera que se sàpiga que un determinat usuari està utilitzant la nostra aplicació i pugui mostrar la opinió sobre la mateixa.

**d) <u>Comunicació entre jugadors</u>**

S'ha de permetre la comunicació entre jugadors de l'aplicació mitjançant un sistema de missatgeria intern, per tal que puguin interactuar entre ells i reptar-se a partides i d' aquesta manera donar una sensació de comunitat a l'aplicació.

## 6.3 ARQUITECTURA DE COMUNICACIÓ

L'arquitectura de comunicació s'haurà de dissenyar i implementar per tal de permetre la comunicació entre els dos jugadors. Al tindre que gestionar diferents dades tant del joc com de l'aplicació, i emmagatzemar-les per posteriorment compartir-les, serà necessari la seva implementació seguint una arquitectura Client-Servidor.

Dintre el meu projecte, per tant, els dispositius mòbils jugaran el paper de client i es comunicaran entre ells mitjançant un servidor, que serà el responsable de proporciona'ls-hi els serveis necessaris per permetre'ls-hi comunicar-se entre ells, accedir a les seves dades personals, etc...

## 6.4 EL SERVIDOR

S'haurà d'implementar un servidor que permeti guardar les dades dels usuaris registrats, les seves estadístiques personals, les dels seus amics, guardar un registre de les partides realitzades amb tota la seva informació, etc... També s'haurà de definir quin sistema de comunicació utilitzarà per tal de interactuar amb els dispositius i de quina manera es compartiran les dades entre ells.

Per guardar tota aquesta informació serà necessari implementar-hi un sistema de base de dades, per tenir en tot moment aquestes dades disponibles per la seva consulta.

Un cop s'hagi dissenyat s'haurà d'implementar en una plataforma de *hosting* que suporti les tecnologies que haguem decidit utilitzar.

### 6.5 RESUM DELS REQUISITS

Així doncs podem diferenciar les nostres especificacions entre funcionals i no funcionals:

**Requisits funcionals**

- Implementació de l'entorn gràfic mitjançant un motor basat en OpenGL
- Disseny i implementació d'una mecànica de joc basada en torns aprofitant la funcionalitat tàctil del dispositiu
- Creació i gestió de comptes d'usuari
- Permetre la creació i cerca de partides multijugador
- Sistema de missatgeria intern
- Integració amb xarxes socials
- Implementació d'un servidor per la gestió de les dades, partides i comunicacions.

**Requisits NO funcionals**

- L'aplicació ha de respectar el sistema de gestió de recursos del sistema operatiu Android
- Les eines utilitzades han de ser OpenSource
- Implementació del servidor en una plataforma de *hosting* compatible.

# 7. ESTUDIS I DECISIONS

## 7.1 EL SISTEMA OPERATIU

El sistema operatiu escollit per desenvolupar la meva aplicació serà Android. El motiu principal d'aquesta decisió és que és el sistema operatiu per smartphones que ocupa actualment una major quota de mercat.

Ens permet l'ús de llibreries i aplicacions OpenSource desenvolupades en Java gràcies a la seva màquina virtual Dalvik, que executa codi java mitjançant el propi compilador d'Android. Disposa de suport per gràfics 2D i 3D mitjançant una variant pròpia de la API OpenGL, OpenGL ES, per tal d'escriure aplicacions gràfiques, el que ens resultarà molt útil de cara al desenvolupament del joc. Part molt important d'aquest elecció és la gran quantitat d'informació i comunitats que ens ha permès tenir al nostre abast molta documentació i exemples per tot tipus de d'aplicacions i funcionalitats.


**Imatge 6 – Logotip de Android.**

## 7.2 EL MOTOR GRÀFIC LIBGDX

LibGDX és un *framework* multi-plataforma basat en Java i OpenGL, pensat exclusivament per la creació i desenvolupament de videojocs.

Destaca per la seva velocitat en el dibuix de gràfics i la seva fàcil utilització, ja que el projectes són desenvolupats en Java i no és necessari de programar directament amb OpenGL. A més segueix una arquitectura d'aplicació semblant a Android el que permet integrar-la de forma intuïtiva a la plataforma, suportant des de la versió 2.0 fins la més actual.

Al ser multi-plataforma, el desenvolupament principal es pot dur a terme en la versió d'escriptori, el que facilita de sobre manera el desenvolupament, per després acabar integrant-la com una Aplicació d'Android en pocs passos.

Pel que fa a la documentació, la pàgina oficial ofereix la documentació necessària per la seva correcta utilització, a través d'exemples, tutorials, fòrums i la seva documentació oficial.

L'únic inconvenient inicial era que per defecte no inclou un suport multi-jugador, però es pot trobar una extensió que


**Imatge 7 – Llibreria gràfica libGDX.**

proporciona suport de xarxa i multi-jugador mitjançant *websockets* anomenada LibGDX-NET[2] desenvolupada per *pepedeab*. Això ens permet agregar suport per crear aplicacions client/servidor mitjançant comunicació *websockets.*

## 7.3 IMPLEMENTACIÓ DEL SERVIDOR AMB *NODEJS*

**Imatge 8 – Servidor d'aplicacions nodeJS.**

*Nodejs* és un *framework* de entrada/sortida (I/O) basat en esdeveniment mitjançant *javascript*. S'executa en el costat del servidor i està pensat per el desenvolupament d'aplicacions de xarxa escalables com pot ser un servidor web, una aplicació de xat o un videojoc multi-jugador online.

El servidor creat en Node.js, s'encarregaria de la lògica del joc i la comunicació amb els jugadors. També duria a terme la autentificació dels jugadors i de l'emmagatzematge de les dades del joc en una base de dades. Tot això és relativament fàcil utilitzant mitjançant mòduls de tercers per Node.js. Aquets mòduls poden ser instal·lats i gestionats en la nostra aplicació gràcies a l'eina NPM (Node Package Manager)**.**

La principal avantatge d'utilitzar *nodejs*, és la possibilitat de implementar una aplicació servidor de forma senzilla i àgil podent-la tenir funcional en pocs passos.

## 7.4 SISTEMA DE COMUNICACIÓ MITJANÇANT WEBSOCKETS

*Websockets* és una tecnologia de comunicació que proporciona un canal bidireccional sobre un *socket* TCP entre el servidor i els seus clients en temps real. Quan un client es connecta al servidor, es produeix un procés de negociació anomenat *handshake* entre els dos. Si la comunicació és acceptada per el servidor, li assigna una ID al client, amb la qual es mantindran comunicats. A partir d'aquest moment, cada cert temps el servidor va preguntant al client a través de *heartbeats* per veure si segueix connectat mitjançant aquesta *sessionID,* i el client li respondrà per continuar mantenint la comunicació entre els dos. En cas contrari el servidor donarà la connexió per finalitzada.

[2] https://github.com/pepedeab/libGDX-Net

Aquest tipus de comunicació és ideal per la comunicació amb aplicacions mòbils, ja que sempre que es produeixi un cert esdeveniment o acció sempre es tindrà comunicat ambdues parts.

En el nostre cas utilitzarem la llibreria socket.io dintre tant de *nodejs* com de la nostra aplicació Android que ens permetrà aquesta comunicació en temps real amb els jugadors de l'aplicació i el servidor.

### 7.4.1 Socket.io

*Socket.IO* és una llibreria Javascript per aplicacions web en temps real. Està composta de dues parts: una llibreria banda-client que funciona en la nostra aplicació, i una part servidor per node.js. Ambdós components tenen una API quasi idèntica. I al igual que *nodejs* funciona mitjançant esdeveniments.

#### a) Mòdul socket.io per *nodejs*

El mòdul Socket.IO ens permet usar el protocol *Websocket* dintre *nodejs*, però si fos necessari pot replegar-se en altres varis mètodes, tals com *Adobe Flash sockets, JSONP polling*, i *AJAX long polling*, mentre continua proporcionant la mateixa interfície. Tot i que es pot utilitzar simplement com un contenidor per a *WebSocket*, proporciona moltes més característiques, incloent la radiodifusió de múltiples tomes de corrent o l'emmagatzemament de dades associades amb cada client i asíncrones de I/O.

#### b) Llibreria Java Socket.IO-Client

En un principi es va decidir utilitzar la llibreria *IOSocket*, per la comunicació de l'aplicació amb el meu servidor *nodejs*, però per certs problemes que es detallaran més endavant, es va decidir finalment utilitzar la llibreria Socket.IO-Client.

Aquesta ens permet comunicara mitjançant el protocol *Websocket* i utilitzar el seu sistema d'esdeveniments, incloent també suport per *cookies*.

Vaig decidir d'utilitzar la comunicació mitjançant *websockets* en el projecte principalment perquè era de vital importància per tal de proporcionar una jugabilitat en temps real en el meu joc, a causa de la mecànica que es va decidir implementar, que la requeria en determinats moments.

En alguns casos es feia necessari, mantenir una comunicació contínua i asíncrona entre totes les parts de l'aplicació, amb el qual aquest tipus de tecnologia resultava ideal per

tal de aconseguir això.

### 7.5 BASE DE DADES MONGODB

En el disseny de la nostra aplicació, la base de dades ha d'estar ubicada a la part servidor i ha de ser aquest el que accedeixi i gestioni totes les dades que s'hi emmagatzemen.

He decidit utilitzar una base de dades *NoSQL* amb *mongoDB*. *MongoDB* s'utilitza per la creació i gestió de base de dades orientades a documents. És altament escalable, d'alt rendiment i sense esquema. Al estar orientada a document, li permet gestionar col·leccions de documents similars al format de dates JSON, fent més natural la gestió de les dades per part de les aplicacions.

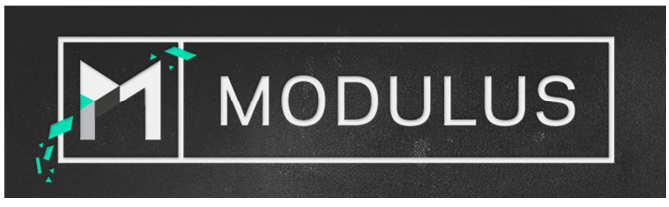

**Imatge 9 – Base de dades *mongoDB.***

A més, per tal de treballar-hi més còmodament amb *nodejs*, trobem un mòdul disponible anomenat *Mongoose*, que ens permet treballar amb aquestes col·lecció de documents modelant-los en objectes i permetent accedir-hi de forma asíncrona.

La decisió de utilitzar *MongoDB*, ve deguda principalment al tipus de dades que s'intercanvien tant el client com el servidor, en forma de objecte JSON de manera que és més fàcil treballar amb elles i manipular-les per part d'ambdues parts.

### 7.6 PLATAFORMA DE *HOSTING MODULUS.IO*

Durant la recerca d'una plataforma de *hosting* que suportés la comunicació mitjançant *websockets*, es van trobar diferents opcions, com *Heroku, nodejitsu, Openshift, modulus,* etc...




**Imatge 10 – Plataforma Hosting MODULUS.IO.**

El principal problema era que algunes d'elles no oferien un total suport per *Websockets*, eren de pagament, no oferien suport per les versions més modernes de *nodejs* o eren de difícil configuració.

Finalment es va escollir *Modulus*, per dues senzilles raons, la primera és la rapidesa i facilitat de *deployar* la nostra aplicació i tenir-la funcional de forma quasi immediata, i la segona que oferia una quota mensual gratuïta cada més valorada en 14.40$ per una sola aplicació.

# 8. ANÀLISI I DISSENY DEL SISTEMA

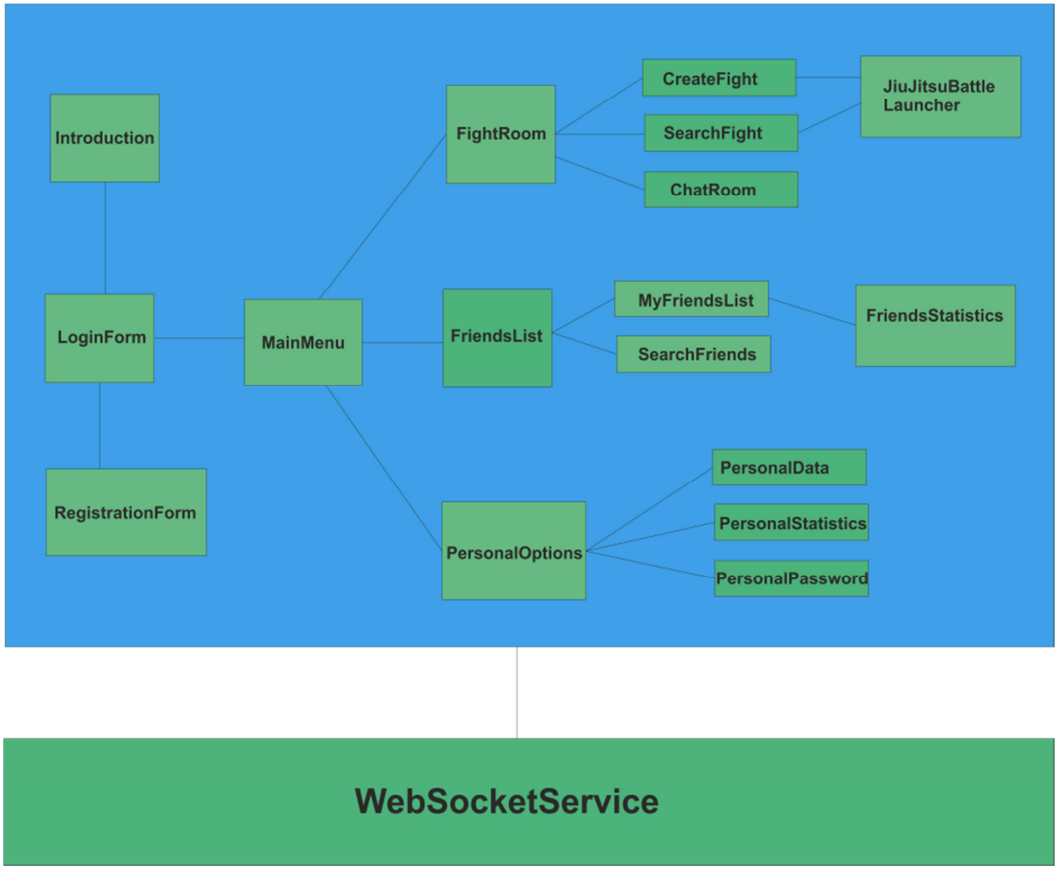
El meu projecte està dividit en dues parts ben diferenciades. Per una banda la part client, que és amb la que interactua directament l'usuari i que és la que s'encarrega de mostrar la interfície de la aplicació, i per l'altre la part servidor que s'encarrega de proporcionar els serveis corresponents als usuaris.

## 8.1 PART CLIENT, PART SERVIDOR I DISENY GLOBAL DE L'APLICACIÓ ANDROID

a) PART CLIENT

La part client de l'aplicació es composa de les classes o activitats Android i el *framework libgdx* que és l'encarregat d'implementar el joc dintre de l'aplicació.

**L'aplicació Android**El component Android de l'aplicació es composa de vàries activitats i d'un servei, les quals es relacionen unes amb les altres de la següent forma:




**Imatge 11 - Estructura d'activitats de l'aplicació Android.**

Cada una d'aquestes Activitats s'encarrega de cobrir una funcionalitat d'aplicació diferent:

- ***Introduction*:** És la primera pantalla que es veu quan s'inicia l'aplicació, en ella es mostren els crèdits inicials abans de carregar l'aplicació.
- ***LoginForm*:** Aquesta activitat és l'encarregada de logar els nostres usuaris en l'aplicació en cas de tenir una compte donat d'alta en la mateixa. En cas de no ser així, l'usuari podrà accedir a la activitat *RegistrationForm* amb la qual podrà crear 'sen una. Des de aquí s'ofereix també a l'usuari la opció de recuperar la seva conta mitjançant una adreça de correu electrònic, la qual generarà de nou el sistema i li enviarà a la direcció indicada.
- ***RegistrationForm*:** Formulari de registre on s'ompliran les dades personals de l'usuari per tal de ser donat d'alta en l'aplicació.
- ***MainMenu*:** L'Activitat que fa referència al menú principal de l'aplicació, des de ón l'usuari podrà accedir a totes les opcions disponibles. Des de aquesta activitat es permetrà connectar amb Facebook i Twitter, per tal de compartir informació.
- ***FightRoom*:** Des de la *FightRoom*, es podrà accedir a les opcions de creació i cerca de partides. També a la *ChatRoom* per tal de comunicar-se amb altres usuaris. De aquesta activitat s'estenen:
  **a)** ***CreateFight*:** Que permet crear una partida mitjançant un nom de partida i descripció.
  **b)** ***SearchFight*:** Que permet a l'usuari buscar partides a partir del seu nom.
  **c) ChatRoom:** On els usuaris de l'aplicació podran conversar per tal d'invitar-se a participar.

Des de la *CreateFight* i *Searchfight*, es podrà accedir a l'activitat que implementa el Joc, *JiujitsuBattle* Launcher.

- ***FriendsList*:** S'encarregarà de la gestió dels amics que tinguem en la nostra aplicació. D'ella s'estenen:

  **a ) *MyFriendsList*:** Mostra la llista dels usuaris que tenim com amics dintre de l'aplicació. Al seleccionar-los, s'obre l'activitat *FriendsStatistics* i ens permet consultar les seves estadístiques.
  **b) *SearchFriends*:** Ens permet cercar usuaris de l'aplicació i afegir-los a la nostra llista d'amics.

- ***PersonalOptions*:** S'encarregarà de oferir als usuaris consultar i modificar les seves dades personals. D'ella s'estenen:

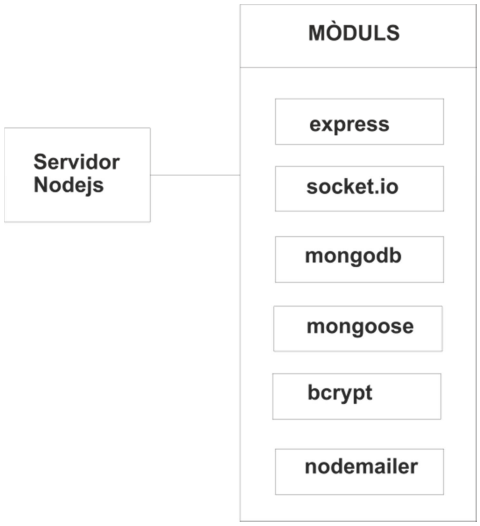
a) ***PersonalData*:** Des de on es pot consultar i modificar les nostres dades personals.
b) ***PersonalStatistics*:** Des de on es podran consultar les nostres estadístiques personals.
c) ***Personal Password*:** On podrem modificar el nostre *password* personal.

- ***JiujitsuBattleLauncher*:** Serà l'activitat encarregada d'implementar el *framework Libgdx* com una Aplicació d'Android.

Per altre banda l'aplicació constarà d'un servei que he anomenat *WebSocketService* amb el qual totes les activitats anteriorment descrites podran mantenir-se en comunicació amb el servidor mitjançant el protocol de comunicació *websocket*. Aquest servei s'executarà un cop s'iniciï l'aplicació i es tancarà quan es surti de la mateixa.

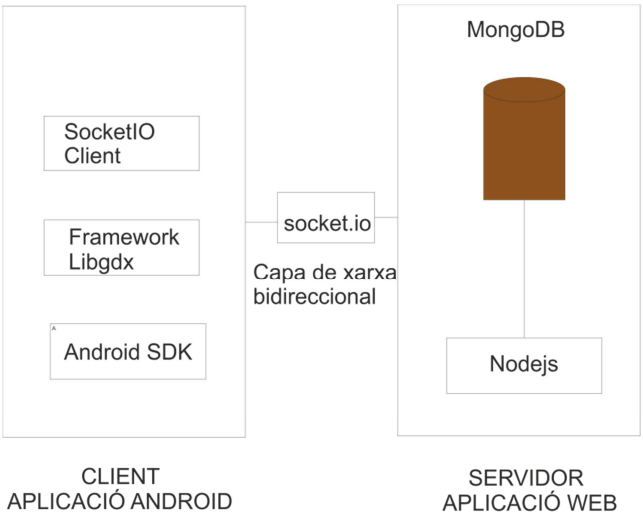
b) LA PART SERVIDOR

La part servidor del nostre projecte, es compondrà de un servidor nodejs, amb una sèrie de mòduls que ens garantitzaran poder oferir els serveis que els clients de l'aplicació necessiten:




- **Express:** És un framework web que ens permetrà implementar la nostra aplicació web de forma senzilla i àgil.
- **Socket.io:** Per la comunicació websocket entre el servidor i els seus clients.
- **MongoDB:** Mòdul per treballar amb la base de dades NoSQL mongodb.
- **Mongoose:** Ens permet treballar amb la base de dades de mongodb de forma intuïtiva mitjançant un intèrpret basat en objectes.
- **Bcrypt:** Aquest mòdul ens permet encriptar les contrasenyes dels nostres usuaris.
- **Nodemailer:** S'utilitzarà per recordar les dades del conte als usuaris que ho demanin mitjançant un mail.

c) DISSENY GLOBAL DEL SISTEMA:




El disseny global del sistema, doncs, es compondrà de una part client desenvolupada amb l'Android SDK , el framework Libgdx i amb la llibreria SocketIO-Client. Aquesta es comunicarà amb la part servidor mitjançant una comunicació socket.io bidireccional.

El servidor per la seva part, es composarà d'una base de dades MongoDB amb la qual emmagatzemarà la informació dels usuaris i de les seves partides. Serà l'encarregat de gestionar les sessions, registres i comunicació entre ells mitjançant el handshake realitzat durnat l'inici de cada connexió.

# 9. IMPLEMENTACIÓ I RESULTATS

A continuació es descriuen els passos seguits durant el projecte de desenvolupament, per tal d'implementar el meu sistema. Un cop dissenyat el nostre sistema i definits els components necessaris per tal de dur a terme totes les tasques definides en les especificacions, va arribar el moment de desenvolupar cada una de les funcionalitats. En els següents punts s'explica cada un dels passos seguits, dels problemes trobats i dels resultat obtinguts.

## 9.1. IMPLEMENTACIÓ DE LES ACTIVITATS DE L'APLICACIÓ

Cada una de les activitats realitzades per la meva aplicació, té una funcionalitat descrita en l'apartat anterior que s'havia d'implementar. Aquestes s'han composat d'una part de codi en forma de classe *java* i una vista XML mitjançant els anomenats *Layouts*.

Els *layouts* s'utilitzen per definir com es veurà la nostra activitat i quins components la composen, *scrolls*, imatges, camps de text, botons, etc...

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

Cada un d'aquests elements té unes propietats que determinen com es veurà a la pantalla, com la seva amplada, alçada, color, tema, si serà visible, entre d'altres coses.

Una de les propietats que més he utilitzat ha set la de onClick, que permet capturar events de pantalla mitjançant la funcionalitat tàctil del dispositiu.

```
public void sendButtonClick(View v) throws JSONException, IOException{
    //mytoast.show(R.string.login_error);
    JSONObject personalData = new JSONObject();
    //Collects the username data and puts it into JSONObject
    String usrmsg = userLogin.getText().toString();
    //Collects the password data and puts it into JSONObject
    String lgnmsg = passwordLogin.getText().toString();

    //Checks the fields
    if( checkForm( usrmsg, lgnmsg ) ) {
        personalData.put("username", usrmsg );
        personalData.put("password", lgnmsg );
        //We send this info to the server throught the websocket connection service
        Intent formdata = new Intent();
        formdata.setClass(LoginForm.this, WebSocketService.class);
        formdata.setAction(WebSocketService.MESSAGE_EMIT_LOGIN);
        formdata.putExtra("login", personalData.toString());
        startService(formdata);

    }
}
```

Aquesta propietat, és la utilitzada per tal de comunicar l'aplicació amb el servidor. En la imatge anterior es pot veure l'*event* de *onClick* de l'activitat de *LoginForm*, mitjançant la qual s'agafen el camps d'usuari i de *password* del formulari de *login*, s'empaqueten en un objecte JSON i s'envien al servidor per la seva validació mitjançant el servei *WebSocketService.*

### 9.2. EL SERVEI WEBSOCKETSERVICE

Per tal de poder capturar tots els *events* de comunicació entre les activitats i el servidor, va ser necessari implementar un Service d'Android anomenat *WebSocketService*. Aquest és una classe d'Android que s'estén la classe nativa Service per tal de capturar certs events tant interns com externs de l'aplicació.

El *WebSocketService* s'executa a l'inici de l'aplicació mitjançant la seva declaració en l'AndroidManifest.xml de la següent forma:

```
<service android:name="com.gantzer.jiujitsubattle.services.WebSocketService" android:enabled="true" >
```

Aquest es connecta en el servidor mitjançant socket.io: `socket = new SocketIO(SERVER_HOST);`

I es queda escoltant el possibles *events* que li puguin arribar des de el servidor.

```
public void on(final String event, IOAcknowledge ack, final Object... args) {
    mHandler.post(new Runnable() {
        @Override
        public void run() { //Crea un thread en el qual escolta els events
            Log.e(TAG,"Server triggered event '" + event +"'");

            Log.e(TAG,"args = " + args[0]);

            //String data = args[0].toString();
            String data = event;
            //We put the object into LinkStart converted into Json
            JSONObject json = null;
            JSONArray jsona = null;

            Intent Link_Start = new Intent(BROADCAST_ACTION);
            Link_Start.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK |Intent.FLAG_ACTIVITY_CLEAR_TOP );
            Link_Start.putExtra("event", data);


            try {
                json = getJSONData( args );
            } catch (JSONException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            Link_Start.putExtra("data", json.toString());

            sendBroadcast(Link_Start);
        }

});
```

Pel que fa a la comunicació amb les activitats es comunica amb elles mitjançant Intents. En aquest cas rep les dades del servidor i crea un intent *Link_Start*, li afegeix les dades rebudes amb la funció *putExtra*() i l'envia a l'aplicació mitjançant *sendBroadcast(Link_Start*).

A dintre de les activitats, s'ha agut d'incloure un *BroadcastReciever*, que s'encarrega d'escoltar els possibles *events* provinents del servei:

```
private BroadcastReceiver broadcastReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
     updateUI(intent);
    }

};
```

I s'obté el tipus d'*event* i les dades contingudes mitjançant la funció *updateUI*(Intent):

```
private void updateUI(Intent intent) {
    String event = intent.getStringExtra("event");
    String data = intent.getStringExtra("data");

    if (event.equals("loginOK")){  //If the login is successful
        //Then proceeds to save the session info and start the next activity
        //We need to save the user name into the preferences file, we save theusername

        try {   //If the login has been successfull, we save the personal data into a preference's file
            savePersonalData(data);
        } catch (JSONException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        Log.e("LoginForm","Dades rebudes: "+ data);
        startActivity(new Intent(getApplication(),MainMenu.class));
        LoginForm.this.finish();
    }
    if (event.equals("loginFail")){ //The login has failed
        mytoast.show(R.string.login_error);
    }
    if (event.equals("passChanged")){ //An email has been send with the new password
        mytoast.show(R.string.login_emailOK);
    }
    if (event.equals("dontExist")){ //The email introduced don't exist in the user's list
        mytoast.show(R.string.login_emailfail);
    }

}
```

On segons el tipus d'*event* rebut pel servidor es dur a terme una acció o una altre.

Des de les activitats també es poden enviar *events* i les dades corresponents al servidor, passant pel servei, mitjançant un intent. És necessari que indiquem el tipus d'*event* i incloguem les dades de la següent forma:

```
//We send this info to the server throught the websocket connection service
Intent formdata = new Intent();
formdata.setClass(LoginForm.this, WebSocketService.class);
formdata.setAction(WebSocketService.MESSAGE_EMIT_LOGIN);
formdata.putExtra("login", personalData.toString());
startService(formdata);
```

S'indica primer la classe des de la qual s'envia i el servei al qual enviem les dades, el tipus d'*event* amb *setAction*(), les dades amb *putExtra*(), i finalment enviem les dades amb *startService*(dades).

La implementació d'un Servei per gestionar la connexió, és molt útil en aquest cas, ja que permet no tindre que repetir codi entre cada una de les activitats i mantenir la connexió en el transcurs d'una a l'altre. A més té la avantatge de que en cas de voler afegir nous events, només tindríem que modificar la classe del servei i la activitat només s'hauria de preocupar de recollir i enviar les dades.

### 9.3. IMPLEMENTACIÓ DEL SERVIDOR AMB *NODEJS*

Per tal d'implementar la nostra aplicació servidor amb *nodejs*, hem creat un arxiu app.js on s'inclouran primer tots els mòduls que utilitzarem:

```
var express = require('express')
  , app = express()
  , http = require('http')
  , server = http.createServer(app)
  , io = require('socket.io').listen(server)
  , MemoryStore = express.session.MemoryStore
  , mongoose = require('mongoose')
  , nodemailer = require('nodemailer')
  , User = require('./models/users')
  , Match = require('./models/matches');
```

Llavors només farà falta indicar a quin port volem que escolti la nostra aplicació web:

```
server.listen(8080);
```

#### 9.3.1. El mòdul de socket.io

Alhora d'implementar el mòdul de socket.io primer establim l'event de connexió entre el servidor i els seus clients:

```
io.sockets.on('connection', function (socket) {
    //With socket.id we can obtain our session.id
    console.log('A socket with sessionID' + socket.id + ' connected!');
    io.sockets.send(socket.id);
```

I dintre s'inclourà les funcions perquè escolti els events que li arriben i actuï en conseqüència mitjançant la funció *socket.on('event', callbackfunction*).

Aquest funció on(), rep l'event corresponent i actua mitjançant una *callback function*. En aquest exemple es pot veure el seu funcionament:

```
socket.on('searchTheFight',function(data){
    var query = Match.findOne({});
    query.and([{name: data.name}, {status: "Created"}]);
    query.exec(function(err,match) {
        if(err){
            console.log(err);
        } else{
            if(match != null){
                io.sockets.emit('matchSearchOK', match);
            } else {
                io.sockets.emit('matchSearchFail', usernames);
            }
        }
    });
});
```

El servidor rep l'event *searchTheFight* i a continuació executa la funció on el paràmetre data són les dades enviades per el client. Finalment el servidor pot enviar una resposta al client mitjançant *io.sockets.emit('event', dades*).

### 9.3.2. La base de dades MongoDB

En la base de dades *mongodb*, definirem dos models de col·lecció, *users i matches*.

#### 9.3.2.1. El model *Users*

El model *users* s'encarregarà d'emmagatzemar els documents referents a les dades del usuaris, es compon del nom d'Usuari, del seu *password* encriptat, l'email, el seu nom, país, *bjjteam* i un *array* que contindrà la seva llista d'amics.

```
var mongoose = require('mongoose'),
    db = mongoose.createConnection('root:Gantzer87db@mongo.onmodulus.net', 'Mozedo6j'),
    Schema = mongoose.Schema,
    bcrypt = require('bcrypt'),
    SALT_WORK_FACTOR = 10;

//First we define the Scema
var usersSchema = new Schema({
    username: { type: String, index: { unique: true }},
    password: String,
    email: String,
    name: String,
    country: String,
    bjjteam: String,
    friends: []
});
```

#### 9.3.2.2. El model *matches*

El model *matches* s'encarregarà d'emmagatzemar els documents referents a les partides realitzades, es compon del nom de la partida, del tipus, del nom del primer

jugador, de la ID de connexió del primer jugador, del nom del segon jugador, de la ID de connexió del segon jugador, de l'estat de la partida, del guanyador, el mètode, el temps i la data.

```
var mongoose = require('mongoose'),
    db = mongoose.createConnection('root:Gantzer87db@mongo.onmodulus.net', 'Mozedo6j'),
    Schema = mongoose.Schema;

//First we define the Scema
var matchesSchema = new Schema({
    name: { type: String, index: { unique: true }},
    type: String,
    firstCompetitor: String,
    firstCompetitorID: String,
    secondCompetitor: String,
    secondCompetitorID: String,
    status: String,
    winner: String,
    method: String,
    time: String,
    date: String
});


module.exports = db.model('matches', matchesSchema);
```

Aquests models, poden ser utilitzats per l'aplicació mitjançant un *require* i apuntant a la ruta on es troben:

```
User = require('./models/users')
Match = require('./models/matches');
```

Llavors es pot treballar directament amb ells utilitzant les funcions pròpies de *mongoose ,find(), findOne(),save(), update()* de la següent forma:

```
User.findOne({username: data.username}, function(err,user){
```

### 9.4. SISTEMA DE GESTIÓ D'AMICS

Els amics de cada usuari es guarden dintre del camp *friends[]* del model *users* de la nostra base de dades, de manera que cada usuari conté en les seves dades personals un *Array de strings* amb el nom de tots els seus amics.

#### 9.4.1. Consultar estadístiques dels amics

Mitjançant l'activitat *FriendsStatistics*, es poden consultar les estadístiques de cada un dels nostres amics. Per tal de fer això s'envia un *event 'getMatches'* de consulta al servidor:

```
socket.on('getMatches', function(data){
    var query = Match.find({});
    query.or([{firstCompetitor: data.username}, {secondCompetitor: data.username}]);
    query.exec(function(err,matches) {
        if(err){
            socket.emit('getMatchesFail', matches);
            console.log(err);
        } else {
         socket.emit('getMatchesOK', matches);
        }
    });
});
```

Quan el servidor rep aquest event, busca dintre la col·lecció *matches* tots els documents en els quals ha participat l'usuari i els retorna a l'aplicació per la seva consulta.

### 9.4.2. Afegir amics

Per tal d'afegir un amic a la nostra llista d'amics, primer s'envia un event de consulta al servidor per veure si existeix. En cas de que el trobi, l'usuari rep la petició de si desitja afegir-lo a la seva llista.

Si la resposta és afirmativa, l'usuari envia *l'event addFriend* al servidor amb el nom de l'amic a afegir.

El servidor primer consulta si existeix l'amic dintre la nostra llista:

```
  var alreadyfriend = false;
      user.friends.forEach(function(friend){
          if(friend == data.friend){
              alreadyfriend = true;
          }
  });
```

En cas de que no existeixi, s'afegirà a la nostra llista d'amics i nosaltres també ens afegirem a la seva:

```
if(!alreadyfriend){
    User.update({username:data.username}, {$pushAll: {friends:[data.friend]}},{upsert:true},function (err){
        if(err){
            console.log(err);
        }else{
            console.log("Successfully added");
        }
    });
    //Then we need to save us as his friend
    User.update({username:data.friend}, {$pushAll: {friends:[data.username]}},{upsert:true},function (err){
        if(err){
            console.log(err);
        }else{
            console.log("Successfully added");
        }
    });

    //Then we search our user and get the updated data
    User.findOne({username: data.username}, function(err,userUpdated){
        if (err) {
            console.log(err);
        } else {
            console.log("Successfully added");
            io.sockets.emit('friendSearchUpdate', userUpdated);
        }
    });
```

Finalment actualitzem les nostres dades actuals enviant *l'event friendSearchUpdate.*

### 9.5. SISTEMA DE GESTIÓ DE PARTIDES

En el nostre joc les partides passen per diferents estats:

- **Created:** La partida acaba de ser creada a l'espera d'un rival per començar.
- **Ready:** Preparada per començar i a l'espera de la confirmació dels dos jugadors. En aquest estat la partida deixa de ser visible per els jugadors que busquen una partida.
- **Started:** La partida està en curs.
- **Finished:** La partida està acabada i per tant es coneixen els resultats de la mateixa. Només en aquest estat es podran consultar les seves estadístiques.

Per tal de que el servidor es pugui comunicar i relacionar ambdós usuaris de la partida, guardem la seva *sessionID* dintre dels camps *firstCompetitorID* i *secondCompetitorID.*

### 9.5. EL JOC AMB LIBGDX


**Imatge 12 – Logotip de Jiujitsu Battle.**

#### 9.5.1. Descripció del joc

El nostre joc s'anomena JiuJitsuBattle, i consisteix en un joc multi-jugador basat en l'art marcial del *Brazilian Jiu Jitsu*. En aquesta art marcial al igual que el judo, es realitzen competicions en les quals els competidors s'enfronten un amb l'altre per tal de veure qui és el millor.

Els combats tenen una duració d'entre 5 i 10 minuts i es poden acabar tant per una finalització (amb una clau) com per decisió recompte de punts. La mecànica de joc implementada per tant ha d'estar amb consonància amb aquestes regles. Mitjançant la capacitat tàctil del dispositiu, interactuarem amb els actors de la pantalla, movent-los i ordenant-los l'acció a realitzar.

### 9.5.2.Disseny i classes del joc

El nostre joc doncs es composarà dels següents elements:

**a) Classe principal:**

JiujitsuBattleGame estén la classe joc i és la que permet implementar les *screens* i els seu respectius actors dintre el meu projecte.

També la utilitzo per tal de passar la informació de la partida des de l'aplicació Android mitjançant el seu constructor.

```
public JiuJitsuBattleGame(String player, String username, String rival, String matchname)
{
    this.player = player;
    this.username = username;
    this.rival = rival;
    this.matchname = matchname;

}
```

Passo el nom d'usuari, el nom del seu rival, el nom de la partida i si aquest es tracta del primer jugador o del segon. Com que les altres classes són implementades per la classe Game podré accedir a aquests atributs des de qualsevol d'elles.

Aquest classe principal s'encarrega de donar pas a la primera pantalla de joc, *SplashScreen.*

**b) *SplashScreen*:**

Aquesta és la classe *Screen* Introductòria del nostre joc, mostrarà els noms dels competidors i esperarà la resposta d'inici per part dels dos mitjançant un tap a la pantalla.

En el moment en que els dos competidors confirmen la partida aquesta canvia el seu estat a *Started* i es carrega la següent pantalla *MatchScreen*.


**Imatge 13 – Inici de la partida, aplicació Juijitsu Battle.**

Quan els dos competidors accepten la partida, el servidor envia *l'event startMatch*:

```
if (event.equals("startMatch")){ //Si els dos usuaris han acceptat iniciar la partida
    stage.addAction( sequence( delay( 3f ), fadeOut( 0.75f ),
            new Action() {
            @Override
            public boolean act( float delta )
            {
                // Then we move to the Mat Screen
                try {
                    socket.disconnect();
                    //usic.stop();
                    game.setScreen( new MatScreen( game ) );
                } catch (JSONException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
                return true;
            }
        } ) );
}
```

Llavors *Splash Screen,* afegeix l'acció a l'*stage* perquè vagi desapareixen gradualment i inici la *Screen MatchScreen*.

**c) *MatchScreen*:**

Aquesta *screen* és la que s'encarrega d'implementar els actors i elements de la partida. Aquí és on es duran a terme totes les accions del nostre joc.

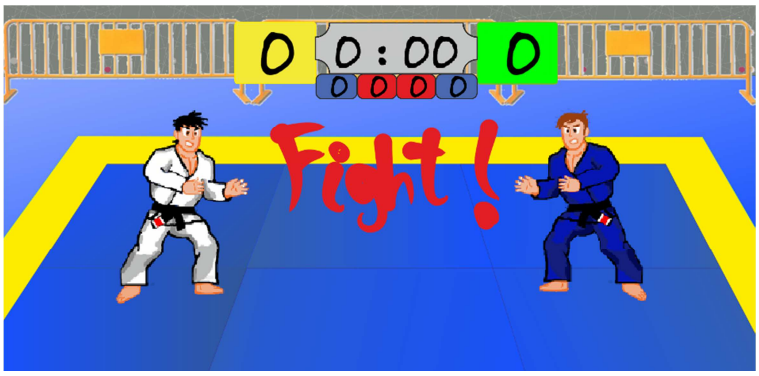
Els actors que la composen són:

- **ScoreBoard:** Fa referència al marcador del nostre joc, i es composa del temps de la partida i de les puntuacions de la mateixa.
- **CompetitorOne:** El primer jugador.
- **CompetitorTwo:** El segon jugador.
- **Event Message:** Els missatges que s'aniran mostrant per la pantalla.
- **GroundFight:** El primer jugador + el segon jugador quan el combat es dur a terme al terra.

Per tal de dibuixar els actors a la pantalla, és necessari crear-los i assigna'ls-hi uns certs atributs i afegir-lo a la pantalla mitjançant *stage.addActor(Actor*):

```
//We create and add the actor
competitorOne = CompetitorOne2D.create(new Competitor(), getAtlas(), 1);
//We add the player into the mat
//competitorOne.setInitialPosition(150, 160);
competitorOne.setX(150);
competitorOne.setPositionX(150);
competitorOne.setY(160);
competitorOne.setScaleX(1.35f);
competitorOne.setScaleY(1.35f);
//we add the ScoreBoard to the stage
stage.addActor(competitorOne);
```

Així doncs la disposició de la pantalla quedaria de la següent forma:

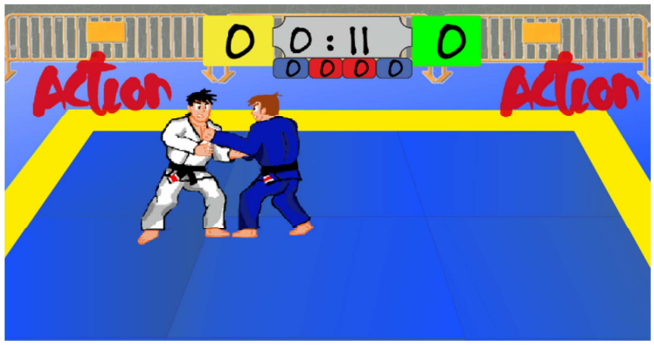



**Imatge 14 – Distribució de la pantalla, Juijitsu Battle.**

### 9.5.2. Estats del combat

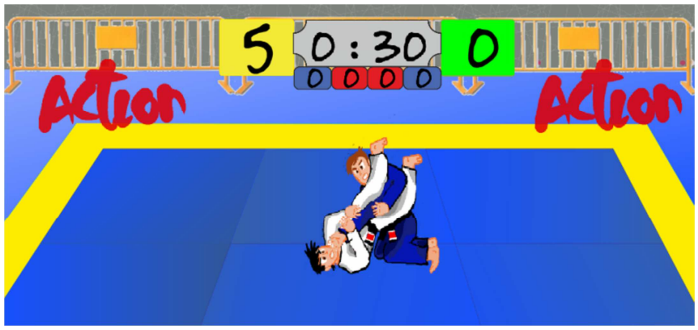
El combat ha estat implementat en tres etapes diferents:

- **Standup Fight:** Els competidors es mouen lliurement representats per els actors *CompetitorOne i CompetitorTwo*.
- **GroundFight:** Els competidors han entrat en contacte un amb l'altre i els actors *CompetitorOne i CompetitorTwo* desapareixen i apareix en el seu lloc l'actor *groundFight,* on s'animen conjuntament com una sola textura. A partir d'aquí el sistema de combat canvia a un sistema basat en torns on cada usuari decidirà el que ha de fer mitjançant un moviment de pantalla.




**Imatge 15 – Estat de combat, Juijitsu Battle.**

- **SubmissionFight:** un dels actors realitza una clau en un altre i llavors comença un mini-joc de presionar la pantalla tants cops com sigui possible. Basant-se en el nombre de cops presionats un guanyarà si el que ha fet la clau guanya el combat finalitza per *submission*, en cas contrari continua.




**Imatge 16 – Submision Fight, Juijutsu Battle.**

### 9.5.3. Canvis d'estat del combat

Els canvis d'estat del combat, s'introdueixen com accions de pantalla, en què sota unes certes condicions, l'estat canviarà.

#### a) De Standup a GroundFight:

```
stage.addAction(new Action() {

    @Override
    public boolean act(float arg0) {
        //If the distance between the two competitors is close then fadeout
        if((competitorTwo.getPositionX() - competitorOne.getPositionX() < 140) && fightStatus.equals(FIGHT_STATUS_STANDUP)){
            //Action
            competitorOne.addAction( sequence( fadeOut( 0f ),
                    new Action() {
                    @Override
                    public boolean act( float delta )
                    {

                        beginGroundFight();
                        return true;
                    }


            } ));

            competitorTwo.addAction( sequence( fadeOut( 0f ),
                    new Action() {
                    @Override
                    public boolean act( float delta )
                    {
                        return true;
                    }

            } ));
        }
```

```
stage.addAction(new Action() {

    @Override
    public boolean act(float arg0) {
        //If the distance between the two competitors is close then fadeout
        if((competitorTwo.getPositionX() - competitorOne.getPositionX() < 140) && fightStatus.equals(FIGHT_STATUS_STANDUP)){
            //Action
            competitorOne.addAction( sequence( fadeOut( 0f ),
                    new Action() {
                    @Override
                    public boolean act( float delta )
                    {

                        beginGroundFight();
                        return true;
                    }


            } ));

            competitorTwo.addAction( sequence( fadeOut( 0f ),
                    new Action() {
                    @Override
                    public boolean act( float delta )
                    {
                        return true;
                    }

            } ));
        }
```

Quan es detecta que els competidors estan a una certa distància un de l'altre, es fan desaparèixer els dos actors i es fa aparèixer l'actor *GroundFight* cridant el mètode *beginGroundFight()*. Depenent de quin sigui l'estat dels dos competidors s'aplicarà una textura o un altre.

#### b) De groundFight a a Submission

Es detecta si l'acció a realitzar té com a "reacció" *submission* en aquest cas s'atura momentàniament el moviment per detectar només els cops a la pantalla, finalment es fa un recompte dels "taps" per determinar el guanyador.

### 9.5.4. Detecció del moviment

Els moviments dels actors es determinen mitjançant la *classe GestureListener*, que detecta l'entrada que rep per pantalla, i actua en conseqüència. Cada moviment representa una acció concreta de l'actor.

El moviments que detecta els hem implementat de la següent forma:

- **Amunt:** l'actor es defensa. (*defense*)
- **Avall:** Adquireix una posició neutral (neutral)
- **Endavant:** Ataca (*attack*)
- **Enredera:** Contraataca (*sweep)*

En aquest cas podem veure com actua *GestureListener* segons la seva entrada en cas de la velocitat en l'eix de les X:

```
private void flingStandupFight(float velocityX, float velocityY, int button) {

    if(Math.abs(velocityX)>Math.abs(velocityY)){
        if(velocityX>0){
            Gdx.app.log("GestureDetectorTest", "Swipe RIGHTT!!! " + competitorOne.isMyAnimationFinished());
            if(player.equals("playerone")){
                competitorOne.setFighterAction("attack");
                competitorOne.addAction(Actions.moveTo(competitorOne.updatePositionX(50), 160, 1f));
            }
            else {
                competitorTwo.setFighterAction("sweep");
                competitorTwo.addAction(Actions.moveTo(competitorTwo.updatePositionX(50), 160, 1f));
            }
            move = "attack";

        }else if (velocityX<0){
            Gdx.app.log("GestureDetectorTest", "Swipe LEFTT!!!"  + competitorOne.isMyAnimationFinished());
            if(player.equals("playerone")){
                competitorOne.setFighterAction("sweep");
                competitorOne.addAction(Actions.moveTo(competitorOne.updatePositionX(-50), 160, 1f));
            } else{
                competitorTwo.setFighterAction("attack");
                competitorTwo.addAction(Actions.moveTo(competitorTwo.updatePositionX(-50), 160, 1f));
            }
            move = "sweep";
        } else {
            // Do nothing.

        }
```

Mitjançant l'event detectat se li assigna una acció o una altre a l'actor i aquest canvia el seu estat.

### 9.5.5. Animacions dels actors

Quan es crea un actor, es carreguen les textures corresponents de les animacions. Aquestes estant

**Imatge 17 – Exemple de combatent, Jiujitsu Battle.**

definides per la seqüència acció_X.png i s'executen en l'ordre seqüencial indicat per X.

Per exemple el moviment *sitdown* agafaria les textures sitdown_1.png, sitdown_2.png i sitdown_3.png i les executaria seqüencialment una rere l'altre mitjançant un determinat període de temps entre textura.

Aquestes es carreguen mitjançant el mètode `regions`

```
=textureAtlasPlayer.findRegions( "singleone").
```

Per tant, en el moment en que *GestureListener* detecta un moviment i canvia l'acció de l'actor aquest assigna una nova regió a l'actor perquè s'animi.

L'encarregat de fer això, és el mètode *act* de l'Actor:

```
@SuppressWarnings("unchecked")
@Override
public void act( float delta )
{
    super.act( delta );

    if(fighterAction.equals("attack")){
        changeRegion(fighterAction);
        animateCompetitor( delta );
    } else if (fighterAction.equals("defense")){
        changeRegion(fighterAction);
        animateCompetitor( delta );
    } else if (fighterAction.equals("neutral")){
        changeRegion(fighterAction);
        animateCompetitor( delta );
    } else if (fighterAction.equals("sweep")){
        changeRegion(fighterAction);
        animateCompetitor( delta );
    }

}
```

Aquí podem veure que segons l'acció detectada per el *Gesture listener* l'actor carrega unes textures o unes altres.

### 9.5.6. Implementació dels estats dels actors

El principal objectiu, alhora d'implementar els estats dels actors, va ser implementar-ho d'una forma estructurada de manera, que en cas de voler afegir nous estats i interaccions fos el més senzill possible.

Per tant es va definir la següent estructura:

En cas de les animacions s'emmagatzemaran dintre de un *HashMap* on cada interacció té assignades unes textures de l'*AtlasRegion* i s'identifiquen per el nom de l'animació.

```
interactions = new HashMap<String, Array<AtlasRegion>>();
```

Els estats s'emmagatzemen en un *HashMap* on s'indica el nom l'estat com a key i un *array* de *strings* on s'emmagatzemen cada una de les interaccions per les quatre direccions detectades:

```
fightStatus = new HashMap<String, String[]>();
```

Per exemple en el cas de l'estat *standup*:

```
states = new String[numberInteractions];
states[0] = "move_forward"; //attack
states[1] = "move_back";    //sweep
states[2] = "hands_up";     //defense
states[3] = "sit_down";     //neutral

fighterStatus.put("standup", states);
```

D'aquesta manera quan es rep un moviment d'*attack*, en l'estat de *standup*, s'anirà a buscar el contingut de *states*[0], i es buscarà l'animació *'move_forward'* dintre del *HashMap* d'animacions, el que retornarà les textures corresponents.

En el cas de l'actor *GroundFigh*t cada estat tindrà dos *arrays* d'interaccions, una per cada jugador, i segons quin hagi set el que hagi guanyat en l'enfrontament del torn, se'n carregarà una o una altre.

```
/******PlayerOne Interactions*****/
stateOne[0] = "singleone";  //attack
stateOne[1] = "guillotwo";  //sweep
stateOne[2] = "singledefensetwo";   //defense
stateOne[3] = "singledefensetwo";       //neutral
//states = stateOne;
fightStatus.put("grab11", states);

/******PlayerTwo Interactions*****/
stateTwo = new String[numberInteractions];
stateTwo[0] = "guilloone";  //attack
stateTwo[1] = "singletwo";  //sweep
stateTwo[2] = "singledefenseone";   //defense
stateTwo[3] = "singledefenseone";       //neutral
fightStatus.put("grab12", stateTwo);
```

A més a més en el cas de *groundFight,* cada acció implicarà una reacció, o sigui un canvi d'estat dels jugadors, aquests estats s'emmagatzemen dintre de un altre *HashMap*, on la interacció guanyadora de l'enfrontament entre els dos jugadors serà l'encarregada de obtenir l'estat següent. Aquest estat carregarà les seves interaccions corresponents i tornarà a esperar el resultat de l'enfrontament per tal de carregar una animació o un altre, i establir el nou estat.

### 9.6. XAT

En la sala de xat de l'aplicació, hem de poder parlar i poder veure les conversacions que tingui tothom en el moment en que estiguem connectats.

Per tal de fer-ho, un cop entrem i enviem un missatge, emetrem un event *'addUser'* al servidor amb el qual si és el primer cop que escrivim ens guardarà el nostre *sessionId* dintre de una *array* d'usuaris. Seguidament tots els missatges que li arribin, seran enviats a totes les *sessionsId* que té emmagatzemades fent un recorregut a l'*array* i d'aquesta manera els rebrà tothom que estigui a la sala.

```
socket.on('addUser', function(data){
    // we store the username in the socket session for this client
    socket.username = data.username;
    // add the client's username to the global list
    usernames[data.username] = data.username;
    // echo to client they've connected
    socket.emit('updateChat', 'SERVER', 'you have connected');
    // echo globally (all clients) that a person has connected
    socket.broadcast.emit('updateChat', data);
    // update the list of users in chat, client-side
    io.sockets.emit('updateUsers', usernames);
  });



  socket.on('sendChat', function (data) {
    // we tell the client to execute 'updatechat' with 2 parameters
    //io.sockets.emit('updateChat', socket.username, data);
    io.sockets.emit('updateChat', socket.username, data);
  });
```

### 9.7 INTEGRACIÓ DE LA API DE FACEBOOK I TWITTER

Per tal d'integrar facebook i twitter, primer caldrà que creem la nostra app dintre la seva comunitat de desenvolupament amb el que ens proporcionaran unes claus d'accés de l'aplicació. Posteriorment ens caldrà incloure les seves llibreries dintre del nostre projecte.

El següent pas serà el de autoritzar-nos en les respectives xarxes socials i un cop s'hagi acceptat la connexió mitjançant les claus proporcionades, podrem interactuar amb elles mitjançant les respectives API's.

# 10. CONCLUSIONS

Aquest projecte m'ha ajudat a adquirir els coneixements necessaris sobre el Sistema Operatiu Android i la seva Arquitectura, a més d'aprendre a integrar el seu SDK amb altres llibreries i tecnologies. Però sobretot m'ha ajudat a aprendre a com s'ha plantejar un projecte, com s'han d'organitzar les tasques, saber escollir la metodologia que millor s'adapti a les nostres necessitats, etc.

M'hagués agradat poder afegir més funcionalitats a la meva aplicació i potser haver-ne prescindit d'algunes altres, però en tot cas el estic satisfet pel treball fet.

Un dels motius d'això ha sigut la tasca de tindre que dissenyar tota la part gràfica del joc, les animacions, el sprites, etc.. Que m'ha robat molt de temps que podria haver dedicat en altres aspectes.

Però estic content d'haverme enfrentat a un projecte d'aquesta magnitud, tenint que aprendre i formar-me amb totes aquestes tecnologies que s'han comentat durant aquesta memòria, però ha valgut la pena tot el camí recorregut.

# 11. TREBALL FUTUR

En un futur es podria millorar varis aspectes de l'aplicació, i introduir-ne de nous, per tal de fer del joc una experiència més completa per l'usuari final.

Pel que respecte a l'aplicació Android en si, es poden afegir algunes funcionalitats de més i millorar les existents. Per una banda es pot millorar el sistema actual de gestió d'Amics incloent moltes més opcions, com la possibilitat d'introduir un sistema de missatgeria privada entre ells o la introducció de un sistema de peticions que permeti entre altres coses acceptar o rebutjar una petició d'amistat. També seria interessant poder integrar la API de facebook per tal de poder jugar amb els amics de la xarxa social, invitar-los a provar el joc, etc...

En el joc es podrien incloure molts més estats i animacions per tal de que resulti complet i no es faci repetitiu, millorar les animacions, incloure més elements en la mecànica, entre altres coses.

# 12. BIBLIOGRAFIA

LIBGDX:

http://glovecatstudio.com/?p=818

http://www.youtube.com/watch?v=eu0p9Jhn8_k

http://steigert.blogspot.com.es/search?updated-max=2012-02-16T16:52:00-02:00&max-results=3&start=9&by-date=false

http://code.google.com/p/libgdx/wiki/SpriteAnimation

http://libgdx.badlogicgames.com/

http://code.google.com/p/libgdx/wiki/SpriteAnimation

ANDROID:

Learning Android Autor:MArko Gargenta editorial: O'Reilly

Android desarrollo de aplicacions Autor: Wei-Meng Lee editorial: Anaya

Android. Guía para desarrolladores (Segunda Edición) ANAYA MULTIMEDIA/MANNING Frank Ableson, Robi Sen y Chris King 656 páginas ISBN: 978-84-415-2958-8 2327708 Mayo 2011

https://github.com/eryngii-mori/Android_nodejs_client

http://www.websmithing.com/2011/02/01/how-to-update-the-ui-in-an-android-activity-using-data-from-a-background-service/

http://developer.android.com/index.html

http://techblogon.com/alert-dialog-with-edittext-in-android-example-with-source-code/#

http://krishnalalstha.wordpress.com/2012/12/21/styling-sherlock-actionbar-change-actionbar-color/

FACEBOOK:

http://developers.facebook.com/docs/android/

http://www.kpbird.com/2013/03/android-login-using-facebook-sdk-30.html

http://blog.doityourselfandroid.com/2011/02/28/30-minute-guide-integrating-facebook-android-application/

http://www.integratingstuff.com/2010/10/14/integrating-facebook-into-an-android-application/

TWITTER:

https://dev.twitter.com/apps/new

http://www.maestrosdelweb.com/editorial/curso-android-trabajando-apis-facebook-twitter/

SOCKETS.IO:

http://www.danielbaulig.de/socket-ioexpress/

http://stackoverflow.com/questions/4641053/socket-io-and-session

NOSEJS:

http://devsmash.com/blog/password-authentication-with-mongoose-and-bcrypt

http://fernetjs.com/2013/02/mongoose-nodejs-modelos-parte-1/

http://tjstein.com/articles/fun-with-node.js/

http://mongoosejs.com/

http://mongoosejs.com/docs/2.7.x/docs/model-definition.html

# ANNEX

## 1. INSTALACIÓ DE L'APLICACIÓ

Per tal d'instal·lar l'aplicació en un dispositiu mòbil serà necessari:

1. Copiar l'arxiu compilat JiuJitsuBattle.apk dintre del dispositiu.
2. Instal·lar-lo mitjançant una aplicació de gestió d'arxius com *Astro*.
3. Jugar!

## 2. MANUAL D'USUARI

### a) Accés a l'aplicació:

Un cop dintre de l'aplicació JiuJitsuBattle, ens tindrem que logar amb la nostre conta d'usuari per tal de poder accedir a la aplicació.

En cas de no tenir-ne cap, serà necessari omplir el formulari de registre per tal de donar-nos d'alta.

Un cop haguem accedit tindrem vares opcions a la nostra disposició.

### b) Iniciar partida:

Desde aquí tindrem la opció de Crear una partida, buscar una partida existent, o bé chatejar amb altres usuaris:

- **Crear la partida:** Crearem una partida introduint un nom de partida i descripció, seguidament premerem el botó de crear i ens apareixerà un missatge indicant que s'està esperant al rival.
  Un cop en tinguem un, s'iniciarà el joc.
- **Buscar partida:** Introduirem el nom de la partida i se'ns mostrarà el resulat, en cas de trobar-ne alguna, ens hi podrem unir.
- **ChatRoom:** des de aquí podrem conversar amb els altres usuaris de l'aplicació mitjançant un xat.

**Joc:**

Un cop dintre del joc, s'inicia la partida mitjançant un missatge d'inici. Els combats duren 5 minuts com a màxim, i es poden acabar per finalització o per decisió. En cas d'arribar a decisió es farà un recompte dels punts aconseguits durant el transcurs de la mateixa.

Per moure el teu personatge lliurement, arrossega el dit cap endavant o cap enredera e la pantalla. Si et vols ajupir o aixecar-te arrossega cap amunt o cap avall respectivament.

Un cop estigis a prop podràs agafar el teu rival i començarà la lluita a terra. A partir d'aquest moment les decisions es duen a terme mitjançant torns en que qui prengui les millors decisions guanya!

**c) Llista d'amics:** La llista d'amics ens permet veure tots els amics que tenim a l'aplicació i consultar les seves estadístiques.

**d) Buscar amics:** Des de buscar amics, podem buscar altres usuaris de l'aplicació buscant pel seu nom d'usuari.

**e) Opcions:** el menú d'opcions ens permet tant veure com modificar les nostres dades personals, com consultar les estadístiques de cada partida que haguem realitzat.

**f) Sortir:** Des de aquí sortim de l'aplicació