JOC D'OTELLO MULTIJUGADOR PER TORNS EN SISTEMES IOS

Memòria

TREBALL FINAL DE CARRERA
Albert Montserrat Gambús
Enginyeria Técnica en Informàtica de Gestió

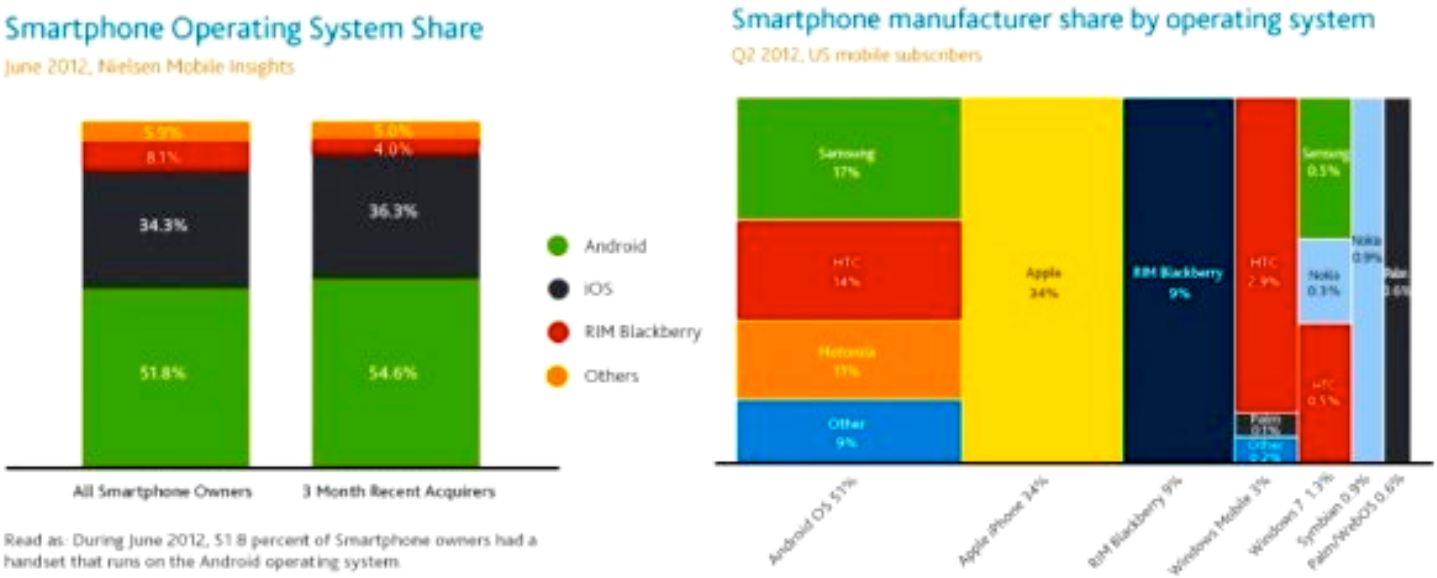
CONSULTOR
Jordi Ceballos Villach

PUBLICACIÓ
7 de Gener de 2013

Index:

1. Introducció

El mercat de la telefonia mòbil s'ha vist revolucionada des de no fa pocs anys per l'aparició de sistemes operatius tant revolucionaris com són el sistema iOS i el sistema Android. Cada cop els SmartPhones tenen una quota de mercat més elevada fins al punt de representar dos terceres part dels mòbils que es compren en aquests moments. Aquest fet és degut en gran part a la gran quantitat d'aplicacions que ofereixen aquests sistemes operatius, la qual cosa el fa molt atractius de cara el públic. [1]



1

Tota aquesta situació ha fet que la demanda de programadors i d'empreses que puguin desenvolupar aplicacions per aquests dispositius hagi augmentat enormement i per tant cada cop apareixen més empreses especialitzades en la programació d'aplicacions per mòbils. És en aquest context en el qual m'he decantat per realitzar el treball final de carrera, consolidant els coneixements adquirits en tota la carrera i aplicant-los als sistemes iOS i a la seva gama de dispositius (iPhone, iPod i iPad).

2. Descripció del projecte

Com a treball final de carrera el meu objectiu és realitzar una aplicació mòbil que permeti a l'usuari realitzar partides d'otello a través de la xarxa. El programa es realitzarà en llenguatge Objective-C, és a dir, per a dispositius en sistema iOS com ara iPhone, iPod i iPad.

El joc funcionarà per torns i serà possible buscar jugadors per correu electrònic d'usuaris ja registrats.

El sistema de joc per torns s'ha posat molt de moda, sobretot si es potencia afavorint el seu ús amb les xarxes socials, amb la seva màxima expressió en el joc Apalabrados, el qual permet jugar al joc de taula Scrable amb amics de forma remota i per torns. El que es planteja en aquest projecte és realitzar un joc semblant, però desviant el tema principal al joc de l'otello.

Inicialment l'usuari haurà d'ingressar un correu electrònic i un nom d'usuari. Un cop fet això, l'usuari ja no haurà de tornar a registrar-se a no ser que tanqui la sessió expressament.

1 http://todoiphone.net/noticias/ios-y-android-dominan-el-mercado-de-los-smartphones/

A continuació tindrem una pantalla on podrem crear una nova partida, i a més tindrem tres llistes: la de partides que esperen la nostra jugada, la de partides que esperen la jugada de l'adversari i la de partides acabades.

En cas de crear un nou joc, accedirem a una pantalla de recerca on podrem trobar usuaris registrats. Seleccionarem un dels resultats i automàticament s'iniciarà la partida.
En cada moviment, el jugador adversari rebrà una notificació push que l'avisarà que ja pot fer la seva jugada.

Per aconseguir aquest objectiu i el bon funcionament de l'aplicació, serà necessari implementar una base de dades al servidor amb diferents taules. Les principals seran Usuari, Partides i Moviments. A part necessitarem altres taules com per exemple una taula amb registres de les claus de cada dispositiu per poder rebre missatges push, entre d'altres.

La complexitat d'aquesta aplicació recau en diversos àmbits. Primerament, s'haurà de programar un motor de joc, on donat un taulell d'otello, l'usuari pugui arrossegar una nova peça al taulell i comprovar si és una jugada vàlida i en cas que sigui així, veure com canvia el color de les peces.

Per altra banda, tindrem la complexitat de crear un sistema de comunicació remota, a través de consultes a fitxers php, que faran de comunicadors entre la base de dades i el dispositiu. S'haurà d'implementar un sistema d'emmagatzematge dels usuaris, de les partides i dels moviments, entre d'altres variables necessàries pel bon funcionament de l'aplicació.

3. Objectius

Els meus objectius que pretenc assolir amb la realització d'aquest projecte són principalment consolidar els meus coneixements adquirits, ja sigui al llarg de la carrera com també al llarg de la meva experiència professional. El que busco es poder perfeccionar principalment el nivell de programació en objective-c, però també de PHP i MySQL, així com proposar-me el repte de confeccionar un bon motor de joc per tal de que el joc d'otello es desenvolupi sense problemes.

Per altra banda, un bon objectiu pot ser el fet de dissenyar un bon sistema de base de dades per tal de que les consultes i l'emmagatzematge de les dades es faci de la forma més eficient possible.

4. Funcionalitats

Dins d'aquest projecte ens podem trobar amb diferents funcionalitats que poden dur a terme els usuaris:

- Registre a l'aplicació:

  Aquest pas és un requisit indispensable per tal de poder avançar amb l'aplicació. Sense un registre l'usuari no podrà continuar utilitzant el programa. Aquesta funcionalitat, consisteix en que l'usuari introduirà un correu electrònic i un nom d'usuari, els dos hauran de ser únics dins la base de dades. Ho enviarà al servidor i aquest serà el responsable d'analitzar les dades, i en cas de ser correctes, retornar la informació del nou usuari. Aquesta informació serà sempre present a nivell intern a l'aplicació per saber si l'usuari està registrat i quin és l'usuari que està en funcionament en cada moment.

- Tancament de sessió:

  En qualsevol moment, l'usuari podrà tancar la sessió. En aquest moment, ho comunicarà al servidor i el programa l'enviarà a l'inici, en la pantalla de registre, amb l'opció d'entrar amb un usuari i contrasenya existents.

- Creació d'una partida:

  En el menú inicial, l'usuari tindrà la opció d'iniciar una partida nova. Aquesta acció es podrà dur a terme buscant un correu del sistema, buscant un nom d'usuari o iniciant una partida aleatòria. En les dues primeres opcions, la partida començarà immediatament i es notificarà a l'adversari per missatgeria push quan es realitzi la primera jugada. En l'últim cas, el servidor consultarà en una llista d'usuaris en espera. En cas de trobar un usuari idoni pel sol·licitant, aquest iniciarà una nova partida igual que en el cas anterior. En cas de no trobar-ne, el sol·licitant serà el que ingressarà a la cua d'espera i se li notificarà per missatge normal.

- Confecció d'una jugada:

  Realitzar una jugada consistirà en arrossegar una nova peça al taulell en una posició correcte. Amb això l'usuari podrà enviar la jugada o desfer i tornar a l'estat inicial de la partida. Si confirma, el moviment s'enviarà al servidor. Aquest el registrarà i enviarà un missatge Push a l'adversari per notificar-li que ja està a punt per rebre la seva jugada.

5. Planificació

Les fases de les que constarà aquest projecte es poden resumir de forma clau amb les 3 entregues de les PAC.

Podríem anomenar i definir les fase de la següent manera:

- Fase de planificació (PAC1):

  En la fase de planificació, el nostre objectiu serà primerament determinar quin projecte voldrem realitzar i el seu funcionament. Tenir clar el producte final que volem aconseguir és clau per tal de poder realitzar una planificació satisfactòria.

  Posteriorment, haurem d'analitzar les fases de treball que tindrà la nostra aplicació per tal de poder-lis assignar una magnitud de temps i realitzar una relació de dependència entre elles.

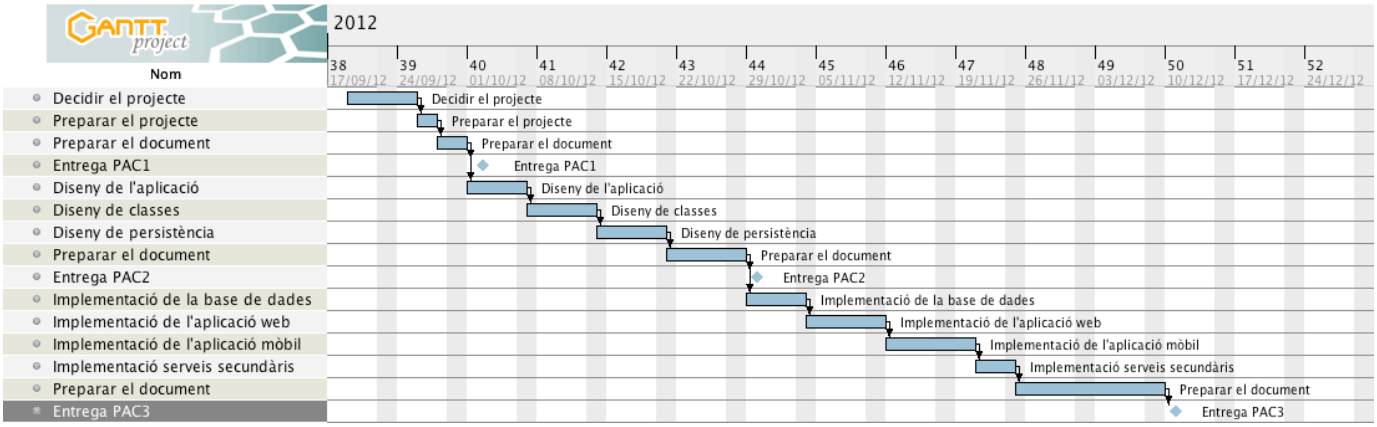
- Fase de disseny (PAC2):

  Aquesta fase es tractarà d'analitzar més a fons les funcionalitats que hem plantejat en la fase de planificació. Això ens servirà per adonar-nos de detalls que haurem de tenir en compte en la fase d'implementació.

- Fase d'implementació (PAC3):

  Finalment tindrem la fase d'implementació, on haurem d'aplicar tot el que hem estat planificant i especificant en les fases anteriors. Requerirà d'anar seguint els passos detallats en la fase de planificació, de manera ordenada, per tal de poder realitzar el treball de forma correcte.

Diagrama de Gantt



6. Eines

L'eina principal per desenvolupar una aplicació per iOS és el XCode. L'XCode és l'eina bàsica per programar en qualsevol llenguatge en Objective-C, ja sigui aplicacions per mac o aplicacions per iOS.

Aquesta eina es va actualitzant a mida que van augmentant les versions dels sistemes operatius i actualment ja donen suport als nous sistemes (iOS 6.0 i MAC OSX Mountain Lion).

Per altra banda, utilitzaré el programa NetBeans 7 per desenvolupar tota la part de servidor. NetBeans és un software especialment utilitzat per la programació d'aplicacions el java, però també recull molts altres llenguatges, entre els que es troba el PHP. Permet la sincronització remota dels fitxers que s'estan programant per FTP, la qual cosa fa que sigui molt pràctic, ja que tot el que es programa, es puja automàticament al servidor.

Per últim, la implementació del servidor i de la base de dades es farà via web, utilitzant qualsevol navegador, tipus Firefox, Safari, Chrome, etc.

7. Requeriments previs

En aquest document recordarem conceptes tractats en punts anteriors i els analitzarem més a fons. En global, es tractarà de concretar el disseny de l'aplicació en tots els seus àmbits, des d'analitzar els possibles actors que utilitzaran l'aplicació fins a determinar el detall de les tecnologies a utilitzar.

Per fer aquest anàlisi partim de la base de que el nostre objectiu serà realitzar un joc per torns d'Otello que utilitzarà els sistemes iOS per tal de presentar-se a l'usuari.

Per fer això haurem de crear un entorn de base de dades que contingui la informació de totes les partides i de tots els usuaris en el servidor i un seguit de fitxers php que interactuïn entre aquest i els dispositius mòbils.

A més, necessitarem d'una espècie de moderador o administrador que pugui accedir a la base de dades via web per tal de poder resoldre possibles problemes que es puguin produir amb l'ús de l'aplicació.

7.1. Actors

Per tant, en aquesta aplicació ens trobarem amb els diferents actors que intervindran en el decurs de l'aplicació:

- Usuari anònim

  Aquest usuari només tindrà accés a una part reduïda de l'aplicació, ja que qualsevol usuari que vulgui jugar haurà d'estar registrat. Bàsicament, només tindrà accés a les primeres pantalles de registre i login.

- Usuari registrat

  Un cop es passen les primeres pantalles de registre, l'usuari ja podrà accedir a la resta de funcionalitats de l'aplicació.

- Usuari administrador

  Aquest usuari tindrà accés web als servies que oferta l'aplicació mòbil pel que fa a base de dades i podrà afegir, modificar i eliminar dades, en el cas que sigui necessari.

7.2. Funcionalitats

Tenint en compte els actors que hem descrit anteriorment, podem llistar un seguit de funcionalitats que tindran lloc al llarg de l'aplicació:

- Registre a l'aplicació

  L'usuari accedirà a l'aplicació i aquesta li preguntarà un correu electrònic per tal de que el sistema pugui crear un nou usuari. El correu es comprovarà amb la base de dades per evitar que ja estigui registrat i en cas contrari afegirà un registre nou a la base de dades. El sistema generarà una contrasenya per l'usuari i li enviarà al correu especificat al fer el registre.

- Iniciar sessió amb un usuari ja registrat

  Donat un usuari ja registrat i que, per qualsevol motiu, s'ha tancat la seva sessió, aquest podrà accedir a l'aplicació a través d'un usuari i contrasenya.

- Modificar el perfil de l'usuari

  Tindrem una pantalla de configuració on cada usuari podrà modificar el seu perfil, així com la seva contrasenya.

- Tancar la sessió

  L'usuari podrà tancar sessió en qualsevol moment. Això farà que qualsevol altre moment que vulgui accedir haurà de passar pel procés d'iniciar sessió o de registre d'un nou usuari.

- Creació d'una partida amb un correu conegut

  Per crear una partida l'usuari podrà accedir a una pantalla de recerca d'usuaris, on introduint el correu podrà verificar que l'usuari està regitrat i per tant invitar-lo a jugar. El sistema crearà una partida nova i prepararà totes les variables per tal de que l'usuari que ha creat la partida pugui fer la jugada.

- Creació d'una partida amb un usuari anònim

  Es podrà crear una partida amb un usuari aleatori. Aquesta funcionalitat s'implementarà amb l'existència d'una taula d'usuaris en recerca de noves partides. Si hi ha algun usuari a la cua, aquest s'assignarà al usuari que ha fet la consulta. Sinó, aquest passarà a formar part de la cua d'espera fins que un altre usuari faci una sol·licitud de partida aleatòria.
- Confecció d'una jugada

  Un cop la partida ja ha estat creada, per torns, cada usuari podrà fer la seva jugada. La partida anirà canviant d'estat en relació a quin usuari li toca moure. La jugada es comunicarà de forma remota a l'oponent, per avisar-lo que ja pot realitzar la seva jugada.

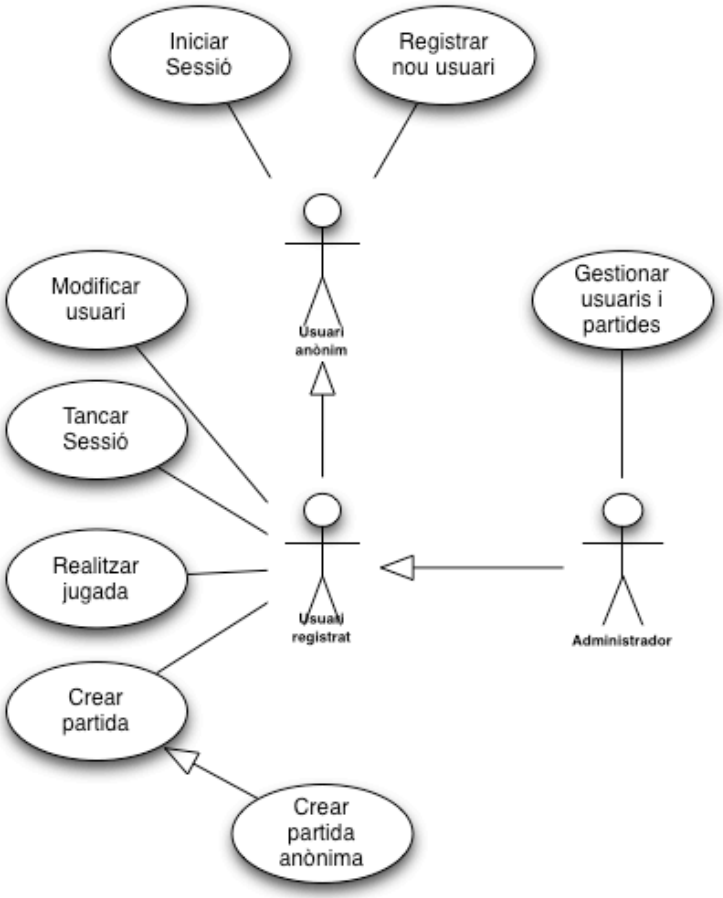
- Administració de usuaris i partides

  Aquesta tasca serà representada per l'administrador del sistema, que tindrà accés a les dades via web i podrà modificar-les en el cas que fos necessari.

8. Anàlisis del sistema

8.1. Diagrama general de casos d'us

En el següent gràfic s'exposen els diferents actors que intervenen en l'aplicació i les seves funcionalitats.



Aquesta il·lustració demostra gràficament tot el que hem comentat en l'apartat 1. Tenim els actors que hem descrit, l'usuari anònim, l'usuari registrat i l'administrador. Per altra banda, tenim les funcionalitats que ja hem descrit assignades a cadascun dels actors.

8.2. Casos d'us

8.2.1. Registre a l'aplicació

- **Actors**
  Usuari anònim

- **Precondició**
  L'usuari ha hagut de descarregar-se l'aplicació de l'App Store i accedir a ella.

- **Postcondició**
  L'usuari s'ha creat correctament o s'alerta a l'usuari d'algun error

- **Flux normal**
  1. L'usuari introdueix un correu electrònic
  2. L'aplicació l'envia al servidor
  3. Aquest la processa, registra l'usuari i retorna un xml amb la informació de l'usuari creat
  4. L'aplicació mòbil rep la informació de forma transparent i l'emmagatzema al dispositiu mòbil

- **Fluxos alternatius**
  3. El servidor rep el correu, el processa, comprova que l'usuari ja existeix i retorna un xml amb un missatge d'error.
  4. L'aplicació mòbil rep la informació de forma transparent i mostra un missatge a l'usuari.

- **Excepcions**
  - Si l'usuari escriu un correu incorrecte l'aplicació l'avisarà abans de passar al pas 2
  - Si hi ha algun error de connexió o algun problema amb el processament del correu, s'avisarà a l'usuari amb un missatge d'error.

8.2.2. Iniciar sessió amb un usuari ja registrat

- **Actors**
  Usuari anònim

- **Precondició**
  L'usuari ha hagut de descarregar-se l'aplicació de l'App Store i haurà hagut de crear prèviament algun compte .

- **Postcondició**
  L'usuari ha iniciat la sessió correctament o s'alerta a l'usuari d'algun error

- **Flux normal**
  1. L'usuari introdueix un correu electrònic i una contrasenya
  2. L'aplicació l'envia al servidor
  3. Aquest la processa, comprova que la contrasenya sigui correcta i retorna un xml amb la informació de l'usuari
  4. L'aplicació mòbil rep la informació de forma transparent i l'emmagatzema al dispositiu mòbil

- **Fluxos alternatius**
  3. El servidor rep el correu i la contrasenya, el processa, comprova que la contrasenya no és la correcta i retorna un xml amb un missatge d'error.
  4. L'aplicació mòbil rep la informació de forma transparent i mostra un missatge a l'usuari.

- **Excepcions**
  - Si l'usuari escriu un correu incorrecte l'aplicació l'avisarà abans de passar al pas 2
  - Si hi ha algun error de connexió o algun problema amb el processament del correu i la contrasenya, s'avisarà a l'usuari amb un missatge d'error.

8.2.3. Modificació del perfil de l'usuari

- **Actors**
  Usuari registrat

- **Precondició**
  L'usuari ha hagut d'iniciar sessió correctament i estar dins de l'aplicació.

- **Postcondició**
  L'usuari ha modificat correctament les seves dades i/o la contrasenya o s'alerta a l'usuari d'algun error

- **Flux normal**
  1. L'aplicació mostra a l'usuari la seva informació personal en camps editables
  2. L'usuari modifica els camps que vol canviar i prem el botó d'enviar
  3. L'aplicació l'envia al servidor
  4. Aquest la processa, registra els canvis a la base de dades i torna un missatge a l'aplicació de OK o FAIL.
  5. L'aplicació mòbil rep la resposta, la processa i mostra un missatge a l'usuari confirmant la modificació.

- **Fluxos alternatius**

- **Excepcions**
  - Si hi ha algun error de connexió o algun problema amb el processament de les dades, s'avisarà a l'usuari amb un missatge d'error.

8.2.4. Tancar la sessió

- **Actors**
  Usuari registrat

- **Precondició**
  L'usuari ha hagut d'iniciar sessió correctament i estar dins de l'aplicació.

- **Postcondició**
  L'usuari haurà tancat la sessió.

- **Flux normal**
  1. L'usuari prem el botó de finalitzar sessió
  2. L'aplicació envia la petició al servidor
  3. Aquest processa i registra que l'usuari ha tancat la sessió eliminant l'identificador del mòbil de la base de dades i retorna un missatge de finalització OK o FAIL.
  4. L'aplicació mòbil rep la informació de forma transparent i l'emmagatzema al dispositiu mòbil i tanca la sessió de l'usuari tornant-lo a la pantalla d'inici.

- **Fluxos alternatius**

- **Excepcions**
  - Si hi ha algun error de connexió o algun problema amb el processament de les dades, s'avisarà a l'usuari amb un missatge d'error.

8.2.5. Creació d'una partida amb un correu conegut

- **Actors**
  Usuari registrat

- **Precondició**
  L'usuari ha hagut d'iniciar sessió correctament i estar dins de l'aplicació i haver seleccionat l'opció de crear una nova partida.

- **Postcondició**
  La nova partida ha estat creada i s'espera a l'usuari que l'ha creat a fer la primera jugada.

- **Flux normal**
  1. L'usuari prem el botó de buscar usuari per correu
  2. L'aplicació obra una pantalla amb un buscador i una taula.
  3. L'usuari introdueix el correu i prem buscar.
  4. L'aplicació envia el correu al servidor el qual retornarà l'usuari en cas de ser trobat.
  5. L'usuari seleccionarà l'adversari
  6. L'aplicació enviarà la petició al servidor el qual crearà una nova partida i enviarà resposta amb les dades necessàries en xml.
  7. L'aplicació rebrà les dades i les processarà per obrir la pantalla de joc i passarà a esperar la primera jugada

- **Fluxos alternatius**

  4. L'aplicació envia el correu al servidor i no troba cap usuari. Aquest tronarà una llista d'usuaris buida.
  5. L'usuari podrà tornar a escriure el correu i reiniciar el pas 3 o sortir.

- **Excepcions**
  - Si hi ha algun error de connexió o algun problema amb el processament de les dades, s'avisarà a l'usuari amb un missatge d'error.

8.2.6. Creació d'una partida amb un usuari anònim

- **Actors**
  Usuari registrat

- **Precondició**
  L'usuari ha hagut d'iniciar sessió correctament i estar dins de l'aplicació i haver seleccionat l'opció de crear una nova partida.

- **Postcondició**
  La nova partida ha estat creada i s'espera a l'usuari que l'ha creat a fer la primera jugada o bé l'usuari ha estat posat a la llista d'espera.

- **Flux normal**
  1. L'usuari prem el botó de començar una partida amb un usuari aleatori
  2. L'aplicació envia la sol·licitud al servidor.
  3. El servidor comprova si hi ha algun usuari a la cua d'espera.
  4. En cas d'haver-hi algun usuari, aquest crea la partida amb els dos usuaris i retorna la informació en xml al usuari final.
  5. L'aplicació rebrà les dades i les processarà per obrir la pantalla de joc i passarà a esperar la primera jugada

- **Fluxos alternatius**

  4. En cas de no haver-hi cap usuari a la cua d'espera, l'usuari es posarà a la cua i el servidor enviarà aquesta informació a l'aplicació.
  5. L'aplicació avisarà a l'usuari de que ha estat posat a la llista d'espera i el farà sortir de la pantalla.

- **Excepcions**
  - Si hi ha algun error de connexió o algun problema amb el processament de les dades, s'avisarà a l'usuari amb un missatge d'error.

8.2.7. Confecció d'una jugada

- **Actors**
  Usuari registrat

- **Precondició**
  L'usuari ha de tenir una sessió oberta, una partida iniciada, i haver seleccionat la partida per la qual vol fer la jugada dins de la llista de partides que esperen la seva jugada.

- **Postcondició**
  La jugada ha estat processada i enviada al servidor i notificada a l'oponent.

- **Flux normal**
  1. L'aplicació mostra a l'usuari la partida amb totes les dades i les peces en el lloc on es van quedar l'últim cop. També es mostra l'última jugada de l'oponent.
  2. L'aplicació li mostra les opcions que té per tal de col·locar una nova peça.
  3. L'usuari arrossega la peça a una de les opcions que el programa li ha ofert.
  4. L'usuari prem el botó d'enviar la jugada al servidor.
  5. L'aplicació prepara les dades perquè puguin ser llegides pel servidor i li envia.
  6. El servidor les processa, les emmagatzema a la base de dades i li envia un missatge a l'oponent en forma de notificació push per avisar-lo de que ja està llest per fer la seva jugada. El servidor també retorna un missatge de OK o FAIL a l'usuari que ha enviat la jugada.
  7. L'aplicació rep la resposta, la processa i informa a l'usuari de forma corresponent.

- **Fluxos alternatius**

  5. L'usuari prem el botó de desfer l'acció realitzada i torna al punt 2.

- **Excepcions**
  - Si hi ha algun error de connexió o algun problema amb el processament de les dades, s'avisarà a l'usuari amb un missatge d'error.

8.2.8. Administració d'usuaris i partides

- **Actors**
  Administrador

- **Precondició**
  L'administrador ha de tenir un nom d'usuari i una contrasenya.

- **Postcondició**
  S'ha modificat correctament la informació de la base de dades.

- **Flux normal**
  1. L'administrador obra una interfície web a través d'un navegador i accedeix al portal d'administració.
  2. El servei web li demana un usuari i una contrasenya.
  3. L'administrador ingressa la informació i l'envia al servidor
  4. Aquest la processa i li dona accés o li informa d'un usuari i contrasenya incorrectes.
  5. Un cop a dins, la web li ofereix un seguit de taules a les quals modificar.
  6. L'administrador selecciona una taula.
  7. La web li dona la informació de la taula amb totes les seves entrades i estructura.
  8. L'administrador selecciona una entrada a modificar o una part de l'estructura.
  9. La web li ofereix una interfície de modificació.
  10. L'administrador modifica les dades i les envia.
  11. El servidor les processa i el retorna a la interfície de detall
  12. L'administrador pot realitzar tantes vegades com vulgui el pas 8 o sortir.

- **Fluxos alternatius**

- **Excepcions**
  - Si hi ha algun error de connexió o algun problema amb el processament de les dades, s'avisarà a l'usuari amb un missatge d'error.

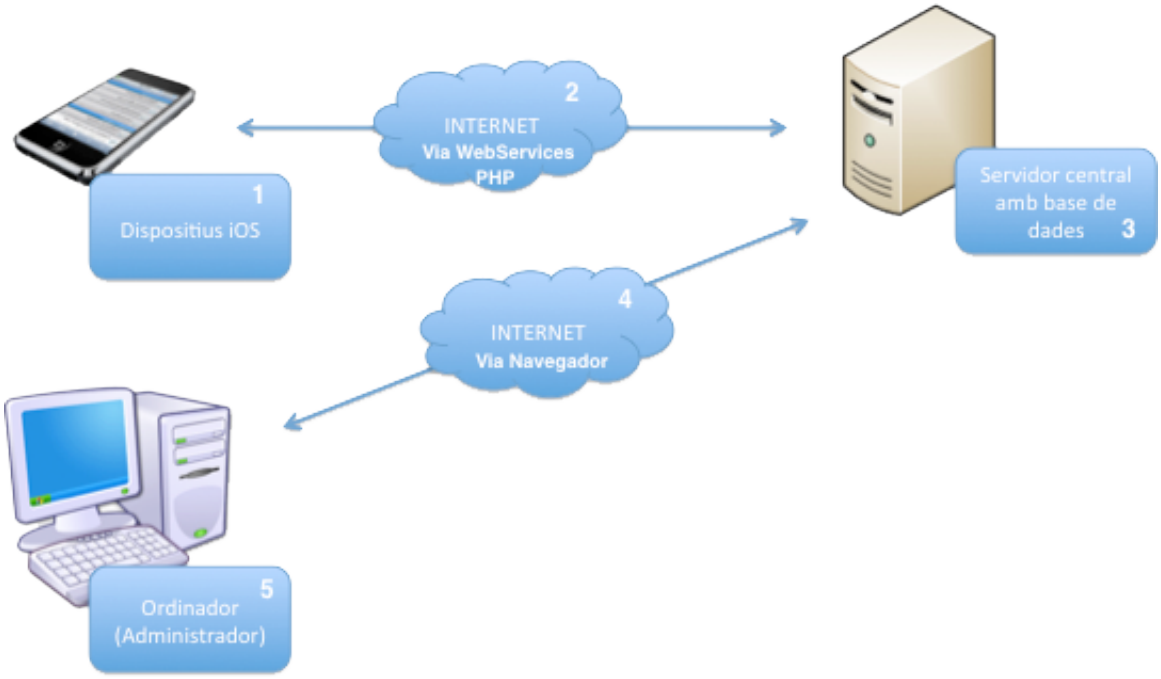
9. Disseny

9.1. Arquitectura global

El disseny de l'arquitectura de l'aplicació, donat que només serà per accés des de dispositius de tipus iOS, i donada la seva similitud entre dispositius, serà un disseny força senzill.

Per una banda tindrem el disseny general de l'aplicació que tal i com podem veure en el gràfic de continuació es tractarà de tres nodes clau:

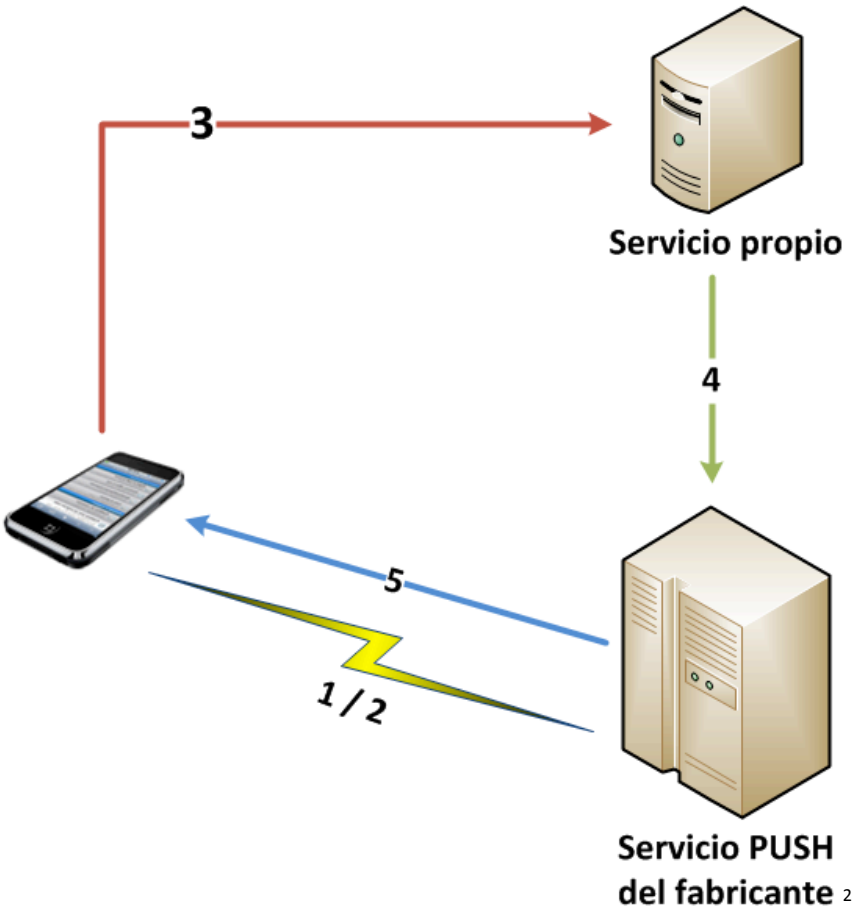
- El servidor central (figura 3), que serà el servidor d'aplicacions i el que contindrà el sistema de base de dades.
- Els dispositius mòbils (figura 1) que es connectaran al servidor central mitjançant internet a través de consultes a web services allotjats al servidor (figura 2). Aquests web services seran scripts en PHP que a partir d'uns paràmetres passats mitjançant consultes tipus GET, retornaran un conjunt de dades en format XML. A la vegada, els dispositius mòbils també utilitzaran el mateix tipus de crides als web services per tal de afegir i modificar dades del servidor.
- L'ordinador de l'administrador (figura 5) que utilitzarà un navegador amb internet (figura 5) per connectar-se al servidor i poder modificar les dades pertinents.



Per altra banda, també és interessant analitzar l'arquitectura global del sistema de les notificacions push.

Tal i com podem veure en el gràfic de continuació tenim tres components importants pel seu bon funcionament:

- El dispositiu mòbil, que rebrà les notificacions i també pot enviar peticions al servidor per enviar-ne de noves.
- El servidor, que rebrà les peticions per enviar noves notificacions o en crearà per si sol.
- Un servei que proporciona Apple de notificacions, que en realitat és un servidor que rep les sol·licituds de servidors de tercers amb unes claus i uns certificats que l'informen de a quina aplicació van destinats i a quin usuari ha d'arribar.



[2]

Tal i com podem observar en el gràfic, el sistema de missatgeria push que proporciona Apple funciona de la següent forma. Primerament tenim el procés de registre de la clau de l'usuari al servidor:

1. Primerament, un pas que no s'observa en el gràfic, el desenvolupador ha de generar uns perfils que contenen el requeriment de sol·licitar l'enviament de missatgeria push a l'usuari. Quan l'usuari obri per primer cop l'aplicació se li sol·licitarà si vol rebre push o no d'aquesta aplicació. En cas afirmatiu, s'inicia tot el procés de registre. En cas contrari, el procés acaba aquí.
2. L'aplicació rep la resposta afirmativa de la sol·licitud i rep un seguit de caràcters anomenats "tokens", que identificaran el dispositiu i l'aplicació.
3. Aquest token es registrà a la base de dades del servidor propi a través d'un dels web services.

[2] http://www.jasoft.org/Blog/post/Arquitectura-Como-funcionan-las-notificaciones-en-los-telefonos-moviles-iPhone-Windows-Phone.aspx

Un cop la clau de l'usuari ja s'ha enregistrat al servidor, ja està disponible per tal de poder enviar-li missatgeria push en qualsevol moment que ho necessitem. Paral·lelament, el desenvolupador haurà hagut de crear i modificar uns certificats del portal de Apple, que es pujaran al servidor per ser tractats junt amb les claus del usuaris. Per tant, en el moment de necessitar enviar un missatge push a un usuari passarem a realitzar el següents passos:

1. Obtindrem la clau de l'usuari del servidor
2. Crearem una connexió socket al servidor de missatgeria push de Apple. En aquest punt ens trobem en que depenent de si estem amb una aplicació en fase de desenvolupament o bé és una aplicació ja descarregada del AppStore, haurem de fer una connexió a un servidor o a un altre:
    a. Per aplicacions en desenvolupament, haurem de fer una connexió a ssl://gateway.sandbox.push.apple.com, pel port 2195.
    b. Per aplicacions en producció, la connexió es farà a ssl://gateway.push.apple.com al port 2195.
3. En aquest socket, a banda de l'informació de la clau i de passar el certificat de l'aplicació, haurem de passar el que s'anomena payload. El payload són uns paràmetres tipus clau/valor en format json. Aquesta paràmetres informaran al servidor de push del missatge a enviar, del so que volem que soni quan l'usuari rebi la notificació i del número que volem que es quedi marcat en la icona de l'aplicació.
4. Amb aquest socket, si tot ha anat correctament, l'usuari rebrà el missatge push en el seu dispositiu.

9.2. Decisions tecnològiques

9.2.1. Dispositiu mòbil

En el cas de la programació en el dispositiu mòbil, la decisió de quin llenguatge utilitzar és evident, ja que per programar per dispositius d'Apple mòbils és imprescindible programar en llenguatge iOS.

9.2.2. Servidor

En el servidor s'ha decidit utilitzar una plataforma Linux per diferents motius:

- Econòmicament, són plataformes lliures
- És una plataforma fàcilment configurable
- Permet l'execució de llenguatge PHP
- Entre d'altres

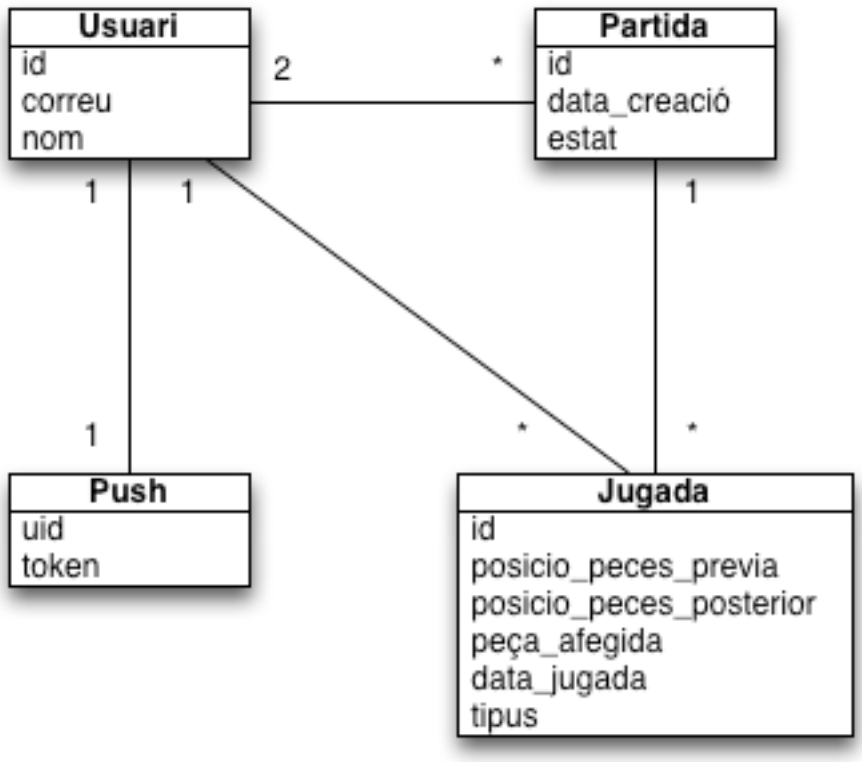
Per tant, pel llenguatge de programació al servidor, utilitzarem PHP, ja que és una eina molt potent per dissenyar i programar aplicacions web.

Per la comunicació entre el servidor i els dispositius mòbils s'utilitzaran estructures XML que generaran les aplicacions PHP i que llegiran els dispositius.

El servidor tindrà una base de dades MySQL.

9.3. Diagrama estàtic de disseny

L'aplicació presenta el següent esquema del diagrama de classes:



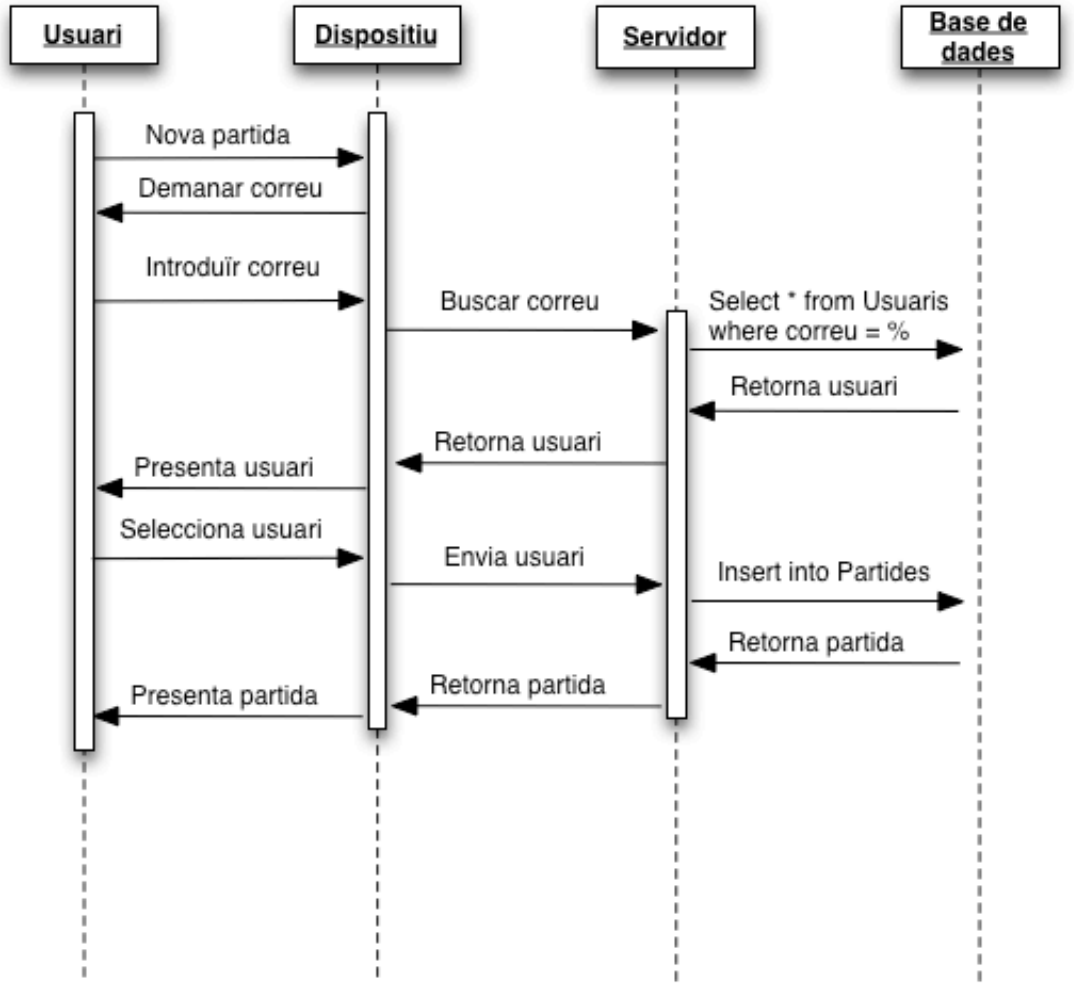
Com podem veure, tenim tres classes clau: Usuari, Partida i Jugada.
A part, tenim la classe Push, ja que és independent de l'usuari. El push serà registrat per cada dispositiu, amb el seu identificador únic i el token corresponent. Aquest identificador s'assignarà a un usuari en el moment d'iniciar sessió, i s'esborrarà de l'usuari en el moment de tancar la sessió.

Per altra banda, tenim que cada usuari pot tenir diverses partides obertes, sense límit, però cada partida només tindrà la referència de dos usuaris, els dos participants en la partida.

Cada partida tindrà diverses jugades, però a la vegada, cada jugada només podrà tenir associada una partida i un usuari que l'ha executat. També tenim que un usuari pot tenir tantes jugades com faci falta, en totes les diferents partides.

9.4. Diagrama de seqüència

Com tots els casos d'us són molt similars, s'ha analitzat el diagrama de seqüència del cas d'ús més complicat, que és el de crear un nova partida. En la resta de casos, el diagrama de seqüència seria pràcticament igual, però amb la diferència de que la consulta a la base de dades seria només una.



Com podem veure, tal i com hem explicat en la descripció del cas d'us, l'usuari demanaria iniciar una nova partida i el dispositiu li oferiria una interfície perquè poses un correu.
Un cop introduït, aquest s'enviaria al servidor per buscar si coincideix amb algun usuari registrat. El servidor farà un consulta a la base de dades i en cas de trobar algun usuari, el retornarà al dispositiu, el qual el presentarà a l'usuari.

Un cop fet això, l'usuari seleccionarà el resultat de la seva recerca i el dispositiu enviarà l'usuari al servidor per tal de que es creï la nova partida. Finalment, retornarà la informació de la nova partida a l'usuari perquè pugui començar a jugar.

9.5. Disseny de la persistència

Seguint amb el model de classes que hem descrit anteriorment en l'apartat 3.3., dissenyarem el model de la persistència de la següent manera:

USUARI (id, nom, contrasenya, correu, uid)
uid és clau forana a la taula PUSH

Aquesta taula representa el registre dels usuaris on tindrem els camps que ens interessa registrar.
El camp id serà l'identificador de l'usuari per tota l'aplicació i el qual es farà servir per vincular els usuaris amb la resta de taules. A part registrarem el nom, la contrasenya de l'usuari, el correu i l'identificar del telèfon mòbil amb el qual l'usuari està jugant.
Aquest últim camp és molt important, ja que s'utilitzarà per saber si un usuari té sessió activa amb algun dispositiu. En cada operació amb el servidor es farà una comprovació de sessió en el qual veurem si el usuari realment té assignat aquest identificador de dispositiu.

PUSH (uid, token)

En aquesta taula farem un registre dels identificadors dels dispositius i el seu token. Això servirà per poder enviar missatges push.

PARTIDA (id, data_creacio, estat, blanques, negres)
blanques i *negres* són claus foranes a la taula USUARI

Aquesta taula serà l'encarregada de registrar les partides. Tindrem l'identificador que serà únic per cada partida, la data de creació, l'estat en el que està la partida, que podrà ser 0, en cas de ser una partida sense jugades, 1 en cas de ser una partida que s'està jugant, i 2 en cas de ser una partida acabada.
Per últim tindrem l'identificador de l'usuari que juga com a blanques i el del que juga com a negres.

JUGADA (id, posició_peces_previa, posició_peces_posterior, peça_afegida, data_jugada, usuari, partida)
usuari és clau forana a la taula USUARI
partida és clau forana a la taula PARTIDA

Aquesta última taula registrarà els moviments de les partides. Serà clau per determinar l'estat de les peces en la partida i també per determinar a qui li toca fer el següent moviment. Gracies a un algoritme que s'implementarà en l'aplicació, es podrà saber qui té el torn de la jugada.
Registrarem, per tant, la posició de les peces abans de la jugada, i la de després, amb una codificació especial, que serà del format <columna><fila><color>:<columna><fila><color>:<columna><fila><color>:... i així per les 64 celles del tauler. Registrarem també la peça afegida amb el mateix format anterior, la data de la jugada, l'usuari que l'ha realitzat i l'identificador de la partida.

CUA (id, idUsuari, data_entrada)
idUsuari és clau forana a la taula USUARI

Aquesta taula registra usuaris que ha demanat iniciar jugades amb un jugador aleatori. Quan no hi ha ningú en aquesta taula amb qui jugar, l'usuari s'afegirà a la taula.

10. Prototip

10.1. Pantalla d'inici

Com podem veure, en aquesta pantalla tenim dos botons, el d'iniciar sessió o el de crear un nou usuari. Aquesta pantalla només accedirà l'usuari anònim i per tant, serà només quan un usuari que no tingui una sessió activa encengui l'aplicació.

10.2. Pantalla iniciar sessió

Aquesta pantalla s'accedeix des de la pantalla d'inici, prement el botó d'iniciar sessió.
En aquest punt, l'usuari podrà introduir el seu correu i la seva contrasenya per tal d'accedir a l'aplicació.

10.3. Pantalla de registre

En aquesta pantalla també accedirem des de la pantalla d'inici, i serà per tal de que l'usuari pugui introduir el seu correu (dos vegades per motiu de seguretat) i el seu nom.
Un cop fet això i enviat al servidor de forma correcta, l'usuari s'haurà creat i podrà accedir a la resta de l'aplicació.

10.4. Pantalla Home

Aquesta serà la pantalla principal dels usuaris amb la sessió iniciada. Es tractarà d'una pantalla que donarà accés a totes les accions que pot executar un usuari registrat. Podrà crear una nova partida, podrà accedir a una partida creada i podrà accedir a la pantalla d'administració del seu compte.

Tindrà el resum de partides obertes i acabades, separades per grups.

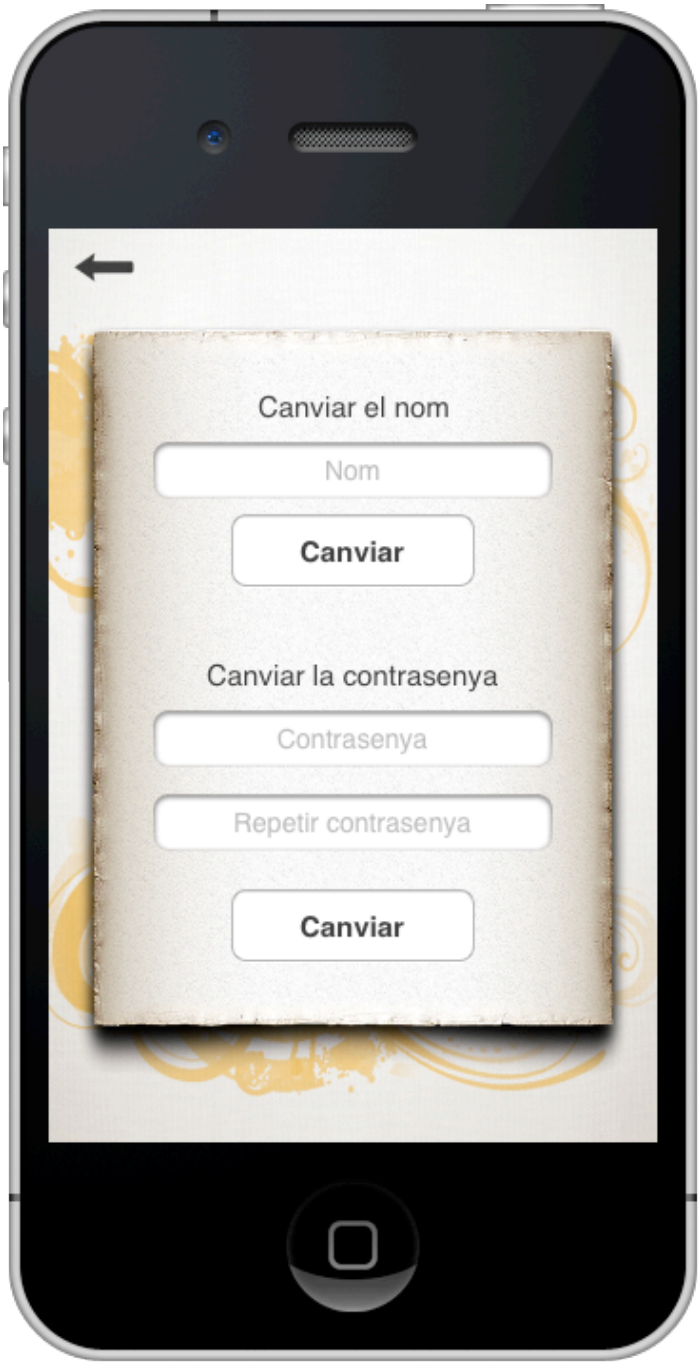
10.5. Pantalla joc

Aquesta és la pantalla més important de l'aplicació. Aquí s'executa el joc. Apareix un taulell amb les seves fitxes col·locades en la posició en les quals l'usuari les va deixar l'últim cop. En aquest moment, l'usuari podrà arrossegar la peça que hi ha en el menú inferior fins a una de les caselles que s'il·luminaran. Un cop fet el moviment, podrà desfer-lo o enviar-lo.
En el menú superior tindrem el recompte actual de punts.

10.6. Pantalla nova partida

Aquesta pantalla serveix per iniciar una nova partida.
Aquí l'usuari introduirà un correu i ho enviarà al servidor. En cas de que el correu correspongui a algun usuari registrat, apareixerà a la taula de sota i l'usuari podrà seleccionar-lo per iniciar una nova partida amb ell.

10.7. Pantalla administració



En aquesta pantalla l'usuari podrà canviar el seu nom amb el qual el veuen els altres usuaris i podrà canviar la seva contrasenya.

11. Implementació:

11.1. Referència d'arxius:

En aquest apartat repassarem per sobre el conjunt de fitxers font que hem desenvolupat, per tal de fer més fàcil el seguiment posterior de l'explicació de la implementació. Començarem parlant dels fitxers dins del projecte de objective-c, els quals s'executaran el dispositiu mòbil i posteriorment comentarem els fitxers en PHP que seran els encarregats d'executar els web services en el servidor.

11.1.1. Objective-C

Com a tot projecte de objective-c, el primer fitxer font que tenim és el **AppDelegate**. Aquesta és la classe principal de tota aplicació de iOS. Es la classe que està sempre activa i representa el delegat general de l'aplicació. Rep crides a mètodes tals com quan l'aplicació s'està encenent, o quan l'usuari surt de l'aplicació, etc. Aquesta classe ens serveix per executar funcions que volem que estiguin actives en tot moment de l'aplicació, amb independència de la pantalla a la que estiguem.

Posteriorment, podem diferenciar els fitxers font que representaran pantalles concretes de l'aplicació i els fitxers que realitzen tasques independents a la interfície de l'usuari, com entitats de classes o classes gestores.

- Pantalles

CheckUserViewController, que serà la pantalla que comprovarà l'existència d'una sessió oberta al servidor per simplificar l'ús de cara a l'usuari.

InicioHomeViewController, pantalla pont, que permetrà a l'usuari, seleccionar si vol crear un nou usuari o entrar amb un usuari ja registrat.

Des d'aquesta última accedirem a **InicioLoginViewController**, pantalla per iniciar sessió amb un usuari registrat o bé a **InicioRegistroViewController** on podrem registrar un nou usuari.

HomeViewController, pantalla d'inici per un usuari amb sessió activa. Apareixerà una llista de partides i les opcions d'accedir a la pantalla per crear una nova partida i la d'accedir a una pantalla per editar els detalls del compte.

A partir d'aquí es podrà accedir a **PartidaViewController**, pantalla de detall de la partida i que servirà per realitzar jugades i interactuar amb el joc. També podem accedir a **ProfileViewController** on podrem editar els detalls del compte amb sessió activa i a **NovaPartidaViewController** per crear una nova partida, tant amb un usuari conegut com amb un usuari aleatori.

- Entitats

Les entitats que utilitzarem per gestionar tot el codi de forma més optima i eficient seran:

Usuari, classe que registrarà tots els detalls de l'usuari, tals com l'identificador, el nom i el correu.
Partida, classe que servirà per crear instàncies de les partides actives i poder gestionar els seus detalls.
Jugada, al igual que les anteriors, aquesta classe registrarà els detalls d'una jugada.

- Controls

Els controls són classes creades expressament per facilitar la programació i fer més eficaç el seu rendiment.

AlertViewProcessando

Classe que crearà una instància global d'una alerta. Molt útil per situacions en les que una pantalla mostra una alerta i és una altra la encarregada de tancar-la. També fa més eficient la programació, ja que només ens hem d'encarregar de mostrar l'alerta i tancar-la, enlloc de crear una nova instància d'una alerta, guardar-la en alguna variable, i posteriorment recuperar la variable per tancar-la.

TBXML

Llibreria encarregada de llegir i descodificar les respostes XML del servidor.

Reachability

Llibreria que determina l'estat de les connexions del dispositiu. Serveix per determinar si l'usuari està connectat a internet, i poder avisar-lo en cas que no ho estigui.

EGORefreshTableHeaderView

Classe que genera una sub-vista dins d'una taula (com és el cas de la pantalla HomeViewController), la qual escolta esdeveniments com per exemple arrossegar la taula avall. Servei per donar una forma senzilla i pràctica de refrescar les dades d'una taula.

GestorConsultes

Aquesta classe unifica en un sol mètode funcions com les de descodificar les respostes XML del servidor, fent ús de la classe TBXML. Són funcions que es repeteixen en diferents punts del programa. D'aquesta forma s'evita repetir codi i s'unifica tot en un mateix punt.

11.1.2. PHP

A continuació comentarem els fitxers fonts en PHP que estaran allotjats al servidor, els quals accedirem des dels dispositius mòbils a través de connexions http.

GestorConsultes.php, fitxer que serveix per unificar codi que s'utilitza en diferents php tals com rebre el detall d'una partida a través del seu identificador.

GestorMensajes.php, que serà la classe encarregada d'enviar els missatges push. D'aquesta forma, en un sol fitxer tenim la funcionalitat d'enviar push, sense haver de preocupar-nos enlloc més dels certificats necessaris.

I a partir d'aquí tindrem diferents php, per situacions molt puntuals els quals el seu nom ja descriu suficientment la seva funcionalitat:

RegisterPush.php
addUser.php
canviarContrasenya.php
canviarNom.php
enviarJugada.php
getPartides.php
getUserWithUID.php
iniciarPartida.php
loginUser.php
logout.php
mailContrasena.php

11.2. Anàlisis del codi i del funcionament de l'aplicació:

11.2.1. Usuaris:

- Comprovació d'usuaris

En l'inici de l'execució del programa, l'aplicació comprovarà si està assignada la variable de usuari amb sessió activa.

```
NSUserDefaults *prefs = [NSUserDefaults standardUserDefaults];
NSData *userData = [prefs objectForKey:@"UserLoged"];

if(userData!=nil){
    HomeViewController *viewcontroller = [[HomeViewController alloc]
        initWithNibName:@"HomeViewController" bundle:nil];
    self.navigationController = [[UINavigationController alloc] initWithRootViewController:
        viewcontroller];
    [self.navigationController setNavigationBarHidden:YES];
    self.window.rootViewController = self.navigationController;
    [self.window makeKeyAndVisible];
}else{
    CheckUserViewController *viewcontroller = [[CheckUserViewController alloc]
        initWithNibName:@"CheckUserViewController" bundle:nil];
    self.navigationController = [[UINavigationController alloc] initWithRootViewController:
        viewcontroller];
    [self.navigationController setNavigationBarHidden:YES];
    self.window.rootViewController = self.navigationController;
    [self.window makeKeyAndVisible];
}
```

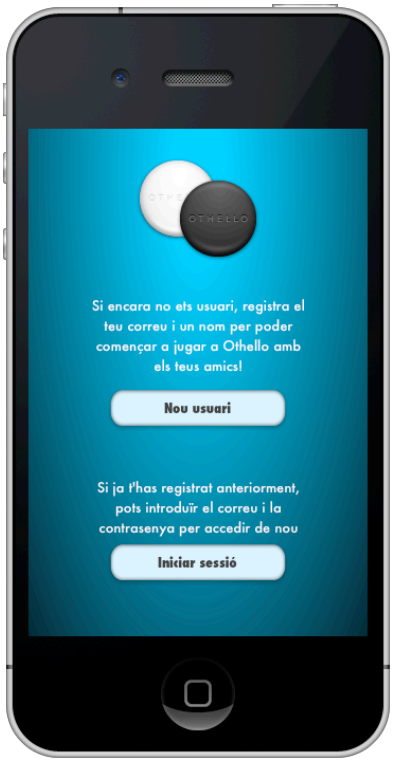
AppDelegate.m (Objective-C)

Tal i com mostra el codi, en cas de que hi hagi algun usuari amb sessió oberta, anirem a la pantalla Home, que analitzarem més endavant. En cas contrari, passarem a la pantalla CheckUserViewController.

Aquesta pantalla té una utilitat molt important de cara a l'usuari. Fa una consulta php getUserWithUID.php del servidor que determina si hi ha algun usuari a la base de dades que tingui l'identificador del dispositiu com a paràmetre uid.

En cas de ser així, aquest retornaria totes les seves dades al programa per poder iniciar sessió automàticament sense haver de molestar a l'usuari. Bàsicament, serveix perquè si algun usuari que ja s'ha registrat i iniciat sessió elimina l'aplicació i la vol tornar a instal·lar, no hagi de tornar a introduir un usuari i una contrasenya per entrar.

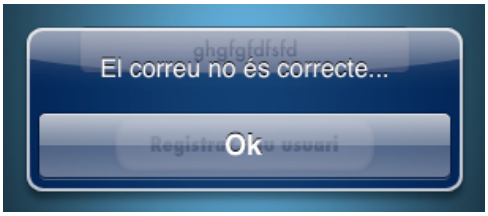
En el cas que aquest php no retorni cap usuari, l'aplicació passarà a la pantalla InicioHomeViewController per tal de poder decidir si vol iniciar sessió amb un usuari registrat o registrar-ne un de nou.



- Registre d'un nou usuari

Per registrar un nou usuari, tenim he d'introduir un correu i un nom pel qual volem ser reconeguts en el joc.

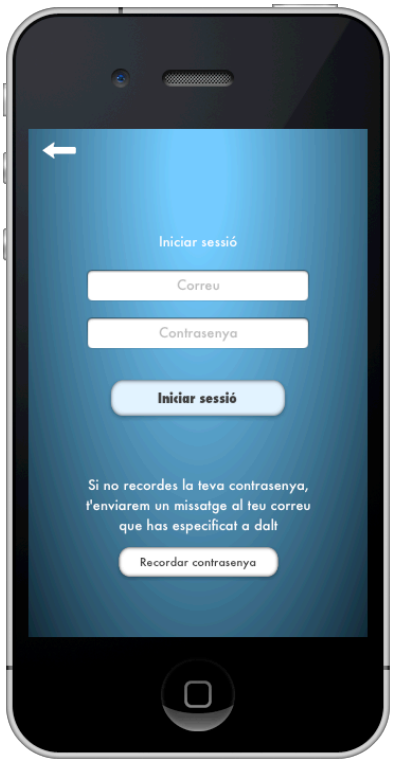
L'aplicació farà els controls pertinents per saber si es tracta d'un correu correcte o si es tracta d'un nom o correu vuits. En aquests casos, mostrarem un missatge d'error a l'usuari.



En cas de ser correcte, enviarem una petició al servidor al php addUser.php que afegirà l'usuari concret i retornarà tota la informació perquè pugui iniciar sessió.

Aquest php també generarà una contrasenya per l'usuari que emmagatzemarà a la base de dades, i el mateix php enviarà un missatge al correu especificat amb la contrasenya.

- Iniciar sessió



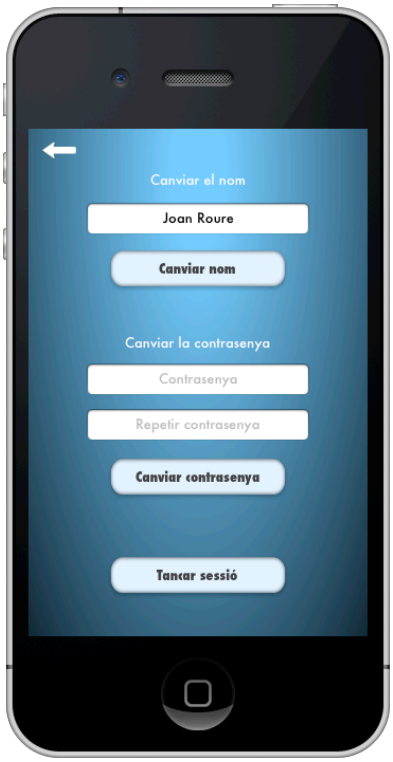
La pantalla d'iniciar sessió és molt similar a la de registre, amb la principal diferència de que enlloc d'un nom, hem de posar una contrasenya. En aquest cas, el php utilitzat serà el loginUser.php. El resultat serà el mateix que en el cas de la pantalla de registre, però sense registrar un nou usuari.

```php
$sql = "select * from usuaris where correu = '".$mail."'";
$query = mysql_query($sql);

$found = false;
$passwordCorrecto = NULL;

$idUser = NULL;
$nombreUser = NULL;
$mailUser = NULL;

if(($fila = mysql_fetch_array($query))) {
    $found = true;
    $passwordCorrecto = $fila['contrasenya'];

    $idUser = $fila['id'];
    $nombreUser = $fila['nom'];
    $mailUser = $fila['correu'];
}


echo '<?xml version="1.0" encoding="UTF-8" standalone="yes"?>';
echo '<userinfo>';
if(!$found){
    echo "<error>Usuari o contrasenya incorrecte</error>";
}else{
    if($password == $passwordCorrecto){
        $sql = "update usuaris set uid = '".$uid."' where id='".$idUser."'";
        if(mysql_query($sql)){
            echo "<id>".$idUser."</id>";
            echo "<nom>".$nombreUser."</nom>";
            echo "<correu>".$mailUser."</correu>";
        }else{
            echo "<error>Error al iniciar sessió</error>";
        }
    }else{
        echo "<error>Usuari o contrasenya incorrecte</error>";
    }
}
echo '</userinfo>';
```

loginUser.php (PHP)

Aquesta pantalla té una particularitat, i és la possibilitat d'enviar un correu a l'usuari amb la contrasenya per tal de poder recuperar-la en cas d'haver-la oblidat. Aquesta opció, comprovarà el correu introduït a la casella de dalt i farà una consulta al servidor en el php mailContrasena.php, on agafarà les dades de la base de dades i enviarà el correu pertinent.

- Administració del compte



Un cop iniciada la sessió, dins la pantalla del llistat de partides, tenim l'opció d'administrar el compte.

Aquí podrem canviar de nom, canviar la contrasenya o tancar la sessió.

En el cas que vulguem canviar de nom, haurem de canviar el nom que ja tenim i acceptar. El programa comunicarà amb el php canviarNom.php que processarà el nom i el canviarà a la base de dades. Tornant al final un OK o FAIL en cas d'èxit o error.

Abans de continuar explicant, hem de destacar un tros de codi que veurem a tots els php referents a accions amb usuaris amb sessió activa. Sempre que l'aplicació contacti amb un php amb un identificador d'usuari per realitzar una acció, també passarem al php el identificador del dispositiu (uid). Amb això, farem una comprovació a la base de dades, per veure que realment en el servidor estigui activa la sessió amb aquest dispositiu.

Pot donar-se el cas d'un usuari que inici sessió amb un dispositiu i un correu donat i per qualsevol motiu el mateix correu inicia sessió en un altre dispositiu. Per tant, el dispositiu amb sessió activa al servidor serà l'últim, i per aquest motiu, quan el primer intenti fer qualsevol acció amb l'identificador d'usuari, aquest rebrà un missatge de sessió tancada i se'l enviarà a la pantalla InicioHomeViewController, perquè pugui tornar a iniciar sessió o registrar un nou usuari.

Per aquest motiu aquest tros de codi el veurem repetit en diferents php.

Seguint amb l'administració d'usuari, podrem canviar la contrasenya igual que canviem el nom, però comunicant amb el php canviarContrasenya.php. El programa comprovarà que les dues contrasenyes siguin iguals per tal de continuar.

L'opció de tancar sessió també serà amb comunicació amb el servidor, ja que hem d'eliminar el camp uid de l'usuari. Un cop fet això, també eliminarem la variable de sessió oberta de l'aplicació.

```
-(void)finDoingLogout:(NSString *)result{
    [[AlertViewProcesando sharedInstance] cerrarAlertaProcessando];

    if(result==nil){
        UIAlertView *myAlert = [[UIAlertView alloc] initWithTitle: nil
                                                          message:@"Error al tancar la sessió..." delegate:nil cancelButtonTitle:@"Ok"
                                                              otherButtonTitles: nil];

        [myAlert show];
        return;
    }else if(![result hasPrefix:@"OK"]){
        UIAlertView *myAlert = [[UIAlertView alloc] initWithTitle: nil
                                                          message:@"Error al tancar la sessió..." delegate:nil cancelButtonTitle:@"Ok"
                                                              otherButtonTitles: nil];

        [myAlert show];
        return;
    }


    NSUserDefaults *prefs = [NSUserDefaults standardUserDefaults];
    [prefs setObject:nil forKey:@"UserLoged"];

    NSMutableArray *mutable = [[NSMutableArray alloc] initWithArray:self.navigationController.viewControllers];

    InicioHomeViewController *view = [[InicioHomeViewController alloc] initWithNibName:@"InicioHomeViewController" bundle:[NSBundle
        mainBundle]];
    [mutable insertObject:view atIndex:0];

    [self.navigationController setViewControllers:mutable];

    [self.navigationController popToRootViewControllerAnimated:YES];

}
```

ProfileViewController.m (Objective-C)

11.2.2. Partides

- Pantalla inicial (Llistat de partides i eines)

Aquesta pantalla serà la pantalla inicial per a un usuari amb sessió oberta. Tindrem primerament un botó per crear una nova partida, i un botó per accedir a la pantalla d'administració d'usuari que hem comentat prèviament.

La pantalla tindrà la funció de carregar totes les partides que l'usuari té obertes i mostrar-li a l'usuari agrupant-les per partides en les quals hem de fer la tirada, partides en les quals l'adversari ha de fer la tirada i partides que ja no es poden fer més moviments i per tant estan acabades.

Per carregar les partides, el programa consultarà al php getPartides.php. Aquest php, com sempre, comprovarà si el identificador del dispositiu i el identificador d'usuari coincideixen per poder comprovar si la sessió està activa també al servidor. En cas afirmatiu, retornarà totes les partides en format xml.

```
NSMutableArray *mutablePerJugar = [[NSMutableArray alloc] init];
NSMutableArray *mutableEsperant = [[NSMutableArray alloc] init];
NSMutableArray *mutableAcabades = [[NSMutableArray alloc] init];

NSData *userData = [prefs objectForKey:@"UserLoged"];
Usuari *usuari = [NSKeyedUnarchiver unarchiveObjectWithData:userData];

for(Partida *partida in mutable){
    if(partida.estat==0){
        if(partida.negres.idUsuari == usuari.idUsuari){
            [mutablePerJugar addObject:partida];
        }else{
            [mutableEsperant addObject:partida];
        }
    }else if(partida.estat==1){
        NSInteger torn = [PartidaViewController getTornForPartida:partida];
        //Jugada *ultimaJugada = [partida.jugades objectAtIndex:[partida.jugades count]-1];
        if(torn == usuari.idUsuari){
            [mutablePerJugar addObject:partida];
        }else{
            [mutableEsperant addObject:partida];
        }
    }else{
        [mutableAcabades addObject:partida];
    }
}

arrayPartidesAcabades = [NSArray arrayWithArray:[mutableAcabades sortedArrayUsingSelector:@selector(compare:)]];
arrayPartidesEsperant = [NSArray arrayWithArray:[mutableEsperant sortedArrayUsingSelector:@selector(compare:)]];
arrayPartidesPerJugar = [NSArray arrayWithArray:[mutablePerJugar sortedArrayUsingSelector:@selector(compare:)]];

[table reloadData];
```

HomeViewController.m (Objective-C)

Tal i com podem veure en el codi, un cop processades les dades del servidor, separem les partides en cada una de les tres categories. L'estat 0 de la partida (que és l'estat inicial de la partida en la qual ningú ha fet encara cap jugada), tindrà el torn l'usuari que sigui les negres. En cas de se estat 1 (que significa que la partida s'està jugant), el torn es treu a partir de l'última jugada. Aquesta funció fa una crida a un mètode estàtic de la classe PartidaViewController que surt a continuació:

```
+(NSInteger)getTornForPartida:(Partida*)_partida{
    if(_partida.estat == 0) return _partida.negres.idUsuari;


    Jugada *ultimaJugada = [_partida.jugades objectAtIndex:[_partida.jugades count]-1];

    NSArray *blanquesMovimentsPossibles = [PartidaViewController getMovimentsPossiblesForPartida:ultimaJugada andColor:@"b"];
    NSArray *negresMovimentsPossibles = [PartidaViewController getMovimentsPossiblesForPartida:ultimaJugada andColor:@"n"];

    if(ultimaJugada.usuari == _partida.blanques.idUsuari){
        if([negresMovimentsPossibles count]!=0) return _partida.negres.idUsuari;
        else if([blanquesMovimentsPossibles count]!=0) return _partida.blanques.idUsuari;
        else return -1;
    }else{
        if([blanquesMovimentsPossibles count]!=0) return _partida.blanques.idUsuari;
        else if([negresMovimentsPossibles count]!=0) return _partida.negres.idUsuari;
        else return -1;
    }

}
```

PartidaViewController.m (Objective-C)

Com podem veure, agafem l'última jugada. També utilitzem una funció estàtica de la mateixa classe que ja analitzarem en la pantalla de la partida, que ens retorna una llista de moviments possibles per a un color en concret.

En una situació normal, és l'adversari del que ha fet l'última jugada el que té el torn, però es pot donar el cas en el que l'últim que ha fet la jugada no sigui el que li toqui jugar, ja que pot ser que l'adversari no pugui fer cap moviment, amb la qual cosa el mateix usuari ha de fer una segona tirada. Per tant, primerament es mira si l'adversari del que ha fet l'última jugada pot tirar, per la qual cosa tindrà el torn, i en cas contrari serà el mateix usuari el que tirarà.

Aquesta pantalla conté una funcionalitat interessant que fa que per refrescar la taula s'hagi d'arrossegar cap avall. Fent això apareix una cel·la oculta. Un cop ha aparegut aquesta cel·la, si deixem anar recarregarem la taula.

Una altra funcionalitat interessant és que la pantalla, cada cop que apareix, ja sigui per primer cop o cada vegada que es tira enrere, consulta una variable global que determina si ha de refrescar la taula o no. Aquesta variable global, es posa a 1 cada cop que s'ha fet una jugada, o s'ha creat una partida nova.

D'aquesta manera, no molesta l'usuari quan simplement fa una consulta a una partida qualsevol i tira enrere, sinó que només forçarà el refrescat de la taula en els moments que realment ho necessiti.

També, tal i com es pot veure en el codi, cada cop que passem per la pantalla, assignem aquesta pantalla el delegat del Push i del ApplicationDidEnterForeground. El tema del Push el comentarem en detall més endavant. Pel que fa a l'altre delegate, tota aplicació iOS té una classe Delegate. En el nostre cas, el AppDelegate.m. En aquesta classe, es registren esdeveniments, com per exemple que l'aplicació entri en estat background perquè l'usuari ha sortit de l'aplicació amb el boto de home del mòbil, o que pel contrari torni a l'estat Foreground, quan torni a entrar a l'aplicació. És aquest últim punt el que interessa detectar per refrescar la taula. D'aquesta manera evitem que un usuari que mantingui l'aplicació en background molt de temps pugui tornar a entrar sense fer cap comprovació ni refrescar informació antiquada.

- Nova partida

Aquesta pantalla ofereix la possibilitat a l'usuari d'iniciar una nova partida de dos maneres diferents.

La primera opció requereix de dos coses: primera, conèixer el correu del teu adversari per poder introduir-lo correctament, i la segona, i més important, que el correu de l'adversari estigui registrat en el sistema. En cas contrari, retornarà error.

La segona opció no requereix de correu electrònic, ja que s'inicia una partida amb qualsevol usuari que estigui esperant per començar una nova partida.

Tot i que semblin dos procediments diferents, el php és el mateix, però amb paràmetres diferents. El php que s'utilitzarà per fer això és el iniciarPartida.php. Com sempre, passarem l'identificador d'usuari i el uid per poder fer la comprovació de sessió activa. En cas de fer una partida amb correu conegut, també passarem aquest correu com a paràmetre.

En el cas de partida amb correu conegut, el php simplement iniciarà una partida on l'usuari que ha creat la partida serà les negres i començarà tirant. En cas de fer una partida amb usuari aleatori, el que farà el php serà comprovar la taula Cua, buscant usuaris per ordre d'inserció a la taula, s'agafarà el primer i es farà servir per iniciar la partida igual que en el cas anterior. En el cas que no hi hagi cap usuari a la cua, l'usuari que ha intentat crear la partida serà el que entrarà a la cua i es retornarà perquè ho pugui comunicar a l'usuari.

```php
if(isset($_REQUEST['mailAdversari'])){
    $mailAdeversari = $_REQUEST['mailAdversari'];

    $sql = "select id from usuaris where correu = '$mailAdeversari'";
    $result = mysql_query($sql);
    if(($fila = mysql_fetch_array($result))){
        $adversariId = $fila['id'];
    }

    if($adversariId == NULL){
        echo "FAIL-NOT FOUND";
        exit();
    }
}else{
    $sql = "select * from cua where idUsuari <> $idUsuari order by data_entrada";
    $result = mysql_query($sql);
    if(($fila = mysql_fetch_array($result))){

        //Agafa el primer usuari de la cua com a adversari
        $adversariId = $fila['idUsuari'];
        $data_entrada = $fila['data_entrada'];

        $sql = "delete from cua where idUsuari = $adversariId and data_entrada = '$data_entrada'";
        mysql_query($sql);

    }else{

        //No hi ha usuaris esperant iniciar partides i per tant es posa a la cua
        $sql = "insert into cua(idUsuari,data_entrada) values($idUsuari,'$dateString')";
        if(mysql_query($sql)){
            echo "FAIL-CUA";
        }else{
            echo "FAIL-ERROR";
        }
        exit();

    }
}
$sql = "insert into partides(data_creacio,estat,blanques,negres) values('$dateString',0,$adversariId,$idUsuari)";
if(mysql_query($sql)){
    $idPartida = mysql_insert_id();
    $sql = "insert into jugades(posicio_peces_previa,posicio_peces_posterior,peca_afegida,data_jugada,usuari,partida)"
        " values('".TAULELLBUIT."','".TAULELLINICI."','-','$dateString',-1,$idPartida)";
    if(mysql_query($sql)){
        echo '<?xml version="1.0" encoding="UTF-8" standalone="yes"?>';
        include_once 'GestorConsultes.php';
        echo GestorConsultes::getPartida($idPartida);
    }else{
        echo "FAIL-ERROR";
    }
}else{
    echo "FAIL-ERROR";
}


?>
```

iniciarPartida.php (PHP)

- Partida

Aquesta pantalla és la més important i també la més complexa. És la que s'encarregarà d'imprimir l'estat actual de les partides i en el cas que sigui el torn de l'usuari, deixar-li fer una jugada (o més d'una, depenent del cas). Aquesta pantalla està dotada de múltiples funcions per gestionar la partida tals com funcions que determinen quines fitxes s'han de girar afegint una fitxa en una posició concreta, o saber quants moviments possibles tenim depenent del color.

Un aspecte clau per entendre el funcionament d'aquesta pantalla i de les seves funcions és la codificació de l'estat de les partides. Per aquest joc hem realitzat una codificació especial que determina en un sol String la posició de totes les peces. El codi tindrà el següent format:

<columna><fila><color>:<columna><fila><color>:<columna><fila><color>:...

Fins a les 64 cel·les que té el tauler.

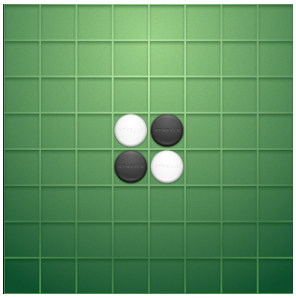
D'aquesta manera, per exemple, un taulell buit és seria com el següent:

```
#define kTaulellBuit "11-:12-:13-:14-:15-:16-:17-:18-:21-:22-:23-:24-:25-:26-:27-:28-:31-:32-:33-:34-:
    35-:36-:37-:38-:41-:42-:43-:44-:45-:46-:47-:48-:51-:52-:53-:54-:55-:56-:57-:58-:61-:62-:63-:64-:
    65-:66-:67-:68-:71-:72-:73-:74-:75-:76-:77-:78-:81-:82-:83-:84-:85-:86-:87-:88-"
```

El taulell inicial correspondria a la següent codificació:

```
#define kTaulellInicial "11-:12-:13-:14-:15-:16-:17-:18-:21-:22-:23-:24-:25-:26-:27-:28-:31-:32-:33-:
    34-:35-:36-:37-:38-:41-:42-:43-:44b:45n:46-:47-:48-:51-:52-:53-:54n:55b:56-:57-:58-:61-:62-:63-:
    64-:65-:66-:67-:68-:71-:72-:73-:74-:75-:76-:77-:78-:81-:82-:83-:84-:85-:86-:87-:88-"
```

Que significaria el següent:



Tal i com podem veure, a la columna 4, fila 4, tenim una fitxa de color blanc (b). A la columna 4, fila 5, tenim una fitxa de color negre (n). La qual cosa correspon a la codificació de dalt 44b:45n.

Un cop explicada la codificació, explicarem el funcionament de la pantalla. Per fer-ho dividirem l'explicació de la pantalla en tres grans grups:

Tutorial

La primera vegada que un usuari passi per aquesta pantalla, el mostrarà un seguit de missatges que ajudaran a l'usuari a comprendre el funcionament del joc. L'usuari sempre podrà o bé seguir amb el tutorial o be saltar-se'l.

```
NSString *hasMakeTutorial = [prefs objectForKey:@"tutoPartida"];
if(hasMakeTutorial == nil){
    alertTutorial = [[UIAlertView alloc] initWithTitle:@"Tutorial" message:@"Aquest és el primer
        cop que accedeixes a una partida. Llegeix les següents indicacions per comprendre les
        instruccions del joc." delegate:self cancelButtonTitle:@"Saltar"
        otherButtonTitles:@"Següent", nil];
    [alertTutorial show];
    pasTuto = 0;
}
```

```objc
pasTuto++;
if(buttonIndex==alertTutorial.cancelButtonIndex) pasTuto = 5;

NSString *missatgeTuto = @"";
NSString *missatgeBoto = @"Següent";
switch (pasTuto) {
    case 1:
    {
        missatgeTuto = @"En el joc d'Othello l'objectiu principal és aconseguir que al tauler
            hi hagi el màxim número possible de peces del teu color. Per aconseguir-ho, has de
            col·locar una peça del teu color en alguna casella en la qual quedin una o més
            peces del color de l'adversari entre la peça afegida i alguna altra peça del teu
            color (verticalment, horitzontalment o en diagonal).";
        missatgeBoto = @"Següent";
    }
        break;

    case 2:
    {
        missatgeTuto = @"Per aconseguir aquest objectiu has d'arrossegar la fitxa que està
            fora del taulell fins a una casella on s'aconsegueixin girar una o més peces de
            l'adversari.";
        missatgeBoto = @"Següent";
    }
        break;

    case 3:
    {
        missatgeTuto = @"Un cop hagis fet la teva jugada podràs, o bé enviar la jugada amb el
            botó de baix a la dreta, o bé desfer la jugada i tornar a l'estat inicial amb el
            botó de baix a l'esquerra. A la part superior podràs veure en tot moment el
            recompte de punts.";
        missatgeBoto = @"Següent";
    }
        break;

    case 4:
    {
        missatgeTuto = @"Ara ja saps tot el necessari per poder jugar! Molta sort i sobretot,
            pensa bé la jugada abans d'enviar-la!";
        missatgeBoto = @"Jugar!";
    }
        break;

    default:
        break;
}
```

PartidaViewController.m (Objective-C)

Funcionament del joc:

A l'entrar a la pantalla, invocarem la funció imprimirPartida:WithAnimacion:toView que serà l'encarregada d'imprimir l'estat inicial de la partida. Aquesta funció, es farà servir tant a l'inici com quan l'usuari seleccioni el boto de desfer.

```
-(void)imprimirPartida:(Partida *)_partida WithAnimacion:(BOOL)_animacion toView:(UIView *)
    _viewTaulell{

    Jugada *ultimaJugada = [_partida.jugades objectAtIndex:[_partida.jugades count]-1];
    NSArray *celles = [ultimaJugada.posicio_peces_previa componentsSeparatedByString:@":"];

    [self imprimirCelles:celles toView:_viewTaulell];


    //return;
    if(_animacion){
        NSInteger columna = [[ultimaJugada.peca_afegida substringWithRange:NSMakeRange(0, 1)]
            integerValue];
        NSInteger fila = [[ultimaJugada.peca_afegida substringWithRange:NSMakeRange(1, 1)]
            integerValue];
        NSString *color = [ultimaJugada.peca_afegida substringWithRange:NSMakeRange(2, 1)];

        NSString *posCella = [NSString stringWithFormat:@"%d%d",columna,fila];

        NSString *posicioFinal = [self addFitxaDeColor:color atView:_viewTaulell inCell:posCella
            forCelles:celles];

        NSLog(@"Posicio final: %@",posicioFinal);

        [self imprimirPuntuacioAmbPosicioFinal:posicioFinal];
    }else{


        NSLog(@"Posicio final: %@",ultimaJugada.posicio_peces_posterior);
        NSArray *celles = [ultimaJugada.posicio_peces_posterior componentsSeparatedByString:@":"];
        [self imprimirCelles:celles toView:_viewTaulell];


        [self imprimirPuntuacioAmbPosicioFinal:ultimaJugada.posicio_peces_posterior];
    }

}
```

PartidaViewController.m (Objective-C)

Com podem veure, aquesta funció agafa l'última jugada de la partida (sempre agafarem l'última jugada de la partida, ja que és la única que ens interessarà) i divideix la posició prèvia en cel·les amb la codificació explicada prèviament. L'entrada sempre tindrà animació, a no ser que l'estat de la partida sigui 0 (la qual cosa significa que ningú ha fet encara una jugada, amb la qual cosa, no és possible fer una animació). En aquest últim cas s'imprimirà l'estat final de forma directa.

```
-(void)imprimirCelles:(NSArray *)_celles toView:(UIView *)_viewTaulell{
    for(UIView *subview in _viewTaulell.subviews){
        [subview removeFromSuperview];
    }

    for(NSString *cella in _celles){
        NSInteger columna = [[cella substringWithRange:NSMakeRange(0, 1)] integerValue];
        NSInteger fila = [[cella substringWithRange:NSMakeRange(1, 1)] integerValue];
        NSString *color = [cella substringWithRange:NSMakeRange(2, 1)];

        if([color compare:@"-"]==NSOrderedSame) continue;

        UIImage *image = nil;
        if([color compare:@"b"]==NSOrderedSame) image = [UIImage imageNamed:@"blanca.png"];
        else image = [UIImage imageNamed:@"negra.png"];

        UIImageView *imageView = [[UIImageView alloc] initWithFrame:CGRectMake((float)(columna-1) *
            (_viewTaulell.frame.size.width/8.0), (float)(fila-1) * (_viewTaulell.frame.size.height/8.0
            ), (_viewTaulell.frame.size.width/8.0), (_viewTaulell.frame.size.height/8.0))];
        [imageView setImage:image];

        NSString *posCella = [NSString stringWithFormat:@"%d%d",columna,fila];

        [imageView setTag:[posCella integerValue]];

        [_viewTaulell addSubview:imageView];
    }
}
```

PartidaViewController.m (Objective-C)

Com podem veure, la funció rep una llista de cel·les i una vista a on les ha d'imprimir. Primerament, borra tot el que hi hagi a la vista. Un cop fet això, analitza cada una de les cel·les enviades. En cas que el color sigui un guió, no fa res. En cas contrari, crea una imatge de la fitxa del color corresponent i la situa a la vista segons les coordenades de la columna i la fila.

Un fet important a comentar és les sentències:

NSString *posCella = [NSString stringWithFormat:@"%d%d",columna,fila];
[imageView setTag:[posCella integerValue]];

Aquestes sentències el que fan és donar una referència aquesta imatge amb el número format per la seva columna i fila. Aquesta referència serà clau a l'hora de fer les animacions al afegir una nova fitxa, tal i com veurem a continuació.

En cas que optem per l'opció d'animar, imprimirem la situació de la partida inicial i farem una crida a la funció addFitxaDeColor:atView:inCell:forCelles. Aquesta funció serà reutilitzada per imprimir les animacions al tirar una fitxa nova.

Analitzem a fons aquesta funció:

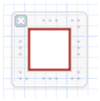
Aquesta funció rep un color, una vista on afegir la peça, una referència de la posició on afegir-la i el llistat de cel·les actual de la partida.

Primerament, el que fem és afegir la fitxa al taulell:

```
NSInteger columna = [[_cella substringWithRange:NSMakeRange(0, 1)] integerValue];
NSInteger fila = [[_cella substringWithRange:NSMakeRange(1, 1)] integerValue];

UIImage *image = nil;
if([_color compare:@"b"]==NSOrderedSame) image = [UIImage imageNamed:@"blanca.png"];
else image = [UIImage imageNamed:@"negra.png"];

CGRect newFrame = CGRectMake((float)(columna-1) * (_viewTaulell.frame.size.width/8.0), (float)
    (fila-1) * (_viewTaulell.frame.size.height/8.0), (_viewTaulell.frame.size.width/8.0),
    (_viewTaulell.frame.size.height/8.0));


NSArray *nib = [[NSBundle mainBundle] loadNibNamed:@"ComponentsPartida" owner:self options:nil];
UIView *view = [nib objectAtIndex:0];
[view setFrame:newFrame];
[_viewTaulell addSubview:view];


UIImageView *imageView = [[UIImageView alloc] initWithFrame:newFrame];
[imageView setImage:image];
[imageView setAlpha:0.0];
NSString *posCella = [NSString stringWithFormat:@"%d%d",columna,fila];
[imageView setTag:[posCella integerValue]];

[_viewTaulell addSubview:imageView];

[UIView animateWithDuration:0.3 delay:0.2 options:UIViewAnimationOptionCurveEaseIn animations:^{

    CATransform3D transform2 = CATransform3DMakeScale(1.1, 1.1, 1.1);
    imageView.layer.transform = transform2;
    [imageView setAlpha:1.0];
} completion:^(BOOL finished) {
    [UIView animateWithDuration:0.3 delay:0.0 options:UIViewAnimationOptionCurveEaseOut
        animations:^{
        CATransform3D transform2 = CATransform3DMakeScale(1.0, 1.0, 1.0);
        imageView.layer.transform = transform2;
    } completion:^(BOOL finished){
    }];
}];
```

PartidaViewController.m (Objective-C)

Tal i com veiem, fem el mateix que en la funció d'impressió de fitxes, però amb dues diferències. Un és el fet d'afegir un element que carreguem del fitxer ComponentsPartida que és un requadre vermell que marcarà la peça que s'ha afegit per tal de que l'usuari ho vegi més clarament.



L'altre diferència que és realitzem una petita animació d'escalat que durarà un total de 0.6 segons a la vegada que fem que passi de transparent a opac, la qual cosa farà remarcar el moviment.

```
NSMutableArray *celles = [[NSMutableArray alloc] init];
for(int xDirection = -1; xDirection<=1; xDirection++){
    for(int yDirection = -1; yDirection <= 1; yDirection++){
        if(xDirection == 0 && yDirection == 0) continue;
        [celles addObjectsFromArray:[PartidaViewController getFitxesPerGirarForCelles:_celles
            addingFitxa:_color inCell:_cella xDirection:xDirection yDirection:yDirection]];
    }
}
```

PartidaViewController.m (Objective-C)

A continuació busquem en les 8 direccions possibles les fitxes que es giraran amb la peça afegida. Per fer això, utilitzarem la funció estàtica getFitxesPerGirarForCelles:addingFitxa:inCell:xDirection:yDirection. Aquesta funció l'explicarem més endavant i també serà reutilitzada per determinar els moviments possibles d'un jugador en concret.

Un cop tenim les cel·les que s'han de girar, realitzem el canvi i fem l'animació.

Un dels temes importants a comentar són el fet de realitzar l'animació, que dura 0.3 segons, on primerament realitzem una primera animació que durà la fitxa del color original fina la posició vertical, i en aquest punt on la fitxa passa a ser tant fina que es torna invisible es farà el canvi de imatge i es tornarà a rotar per quedar en la posició inicial.

Un altre tema important és que aquesta funció registra els canvis que es fan en una variable string anomenada finalStringCelles. Aquesta variable serà la que es retornarà i això servirà per formar la jugada que realitzi l'usuari, la qual s'enviarà al servidor perquè la processi.

Analitzem ara la funció getFitxesPerGirarForCelles:addingFitxa:inCell:xDirection:yDirection:

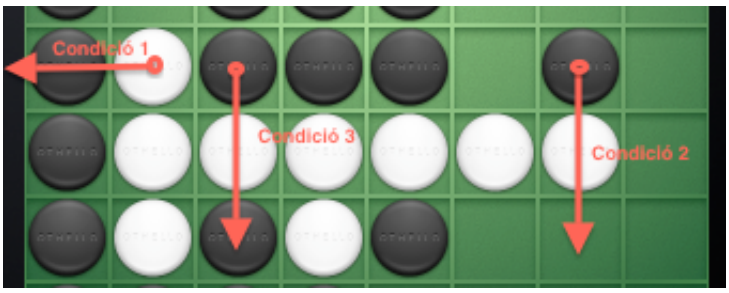
```
+(NSArray *)getFitxesPerGirarForCelles:(NSArray*)_celles addingFitxa:(NSString *)_color inCell:
    (NSString *)_cella xDirection:(NSInteger)xDirection yDirection:(NSInteger)yDirection{
    NSInteger columna = [[_cella substringWithRange:NSMakeRange(0, 1)] integerValue];
    NSInteger fila = [[_cella substringWithRange:NSMakeRange(1, 1)] integerValue];

    NSMutableArray *mutable = [[NSMutableArray alloc] init];
    while(true){
        columna = columna + xDirection;
        fila = fila + yDirection;
        if(columna < 1 || columna > 8 || fila < 1 || fila > 8) {
            [mutable removeAllObjects];
            break;
        }

        NSString *posCella = [NSString stringWithFormat:@"%d%d",columna,fila];

        NSString *color = @"-";
        for(NSString *cella in _celles){
            if([cella hasPrefix:posCella]) color = [cella substringWithRange:NSMakeRange(2, 1)];
        }

        if([color compare:@"-"]==NSOrderedSame){
            [mutable removeAllObjects];
            break;
        }else if([color compare:_color]==NSOrderedSame) break;

        [mutable addObject:posCella];
    }

    return mutable;
}
```

PartidaViewController.m (Objective-C)

Tot i que pugui sembla senzilla, aquesta funció és la clau de tot el joc. El correcte funcionament d'aquesta funció garanteix el correcte funcionament de l'aplicació.
Rep una llista de cel·les que representa la situació del taulell, el color a afegir, la posició de la cel·la que afegim i les direccions que prenem per fer la comprovació.

Es crearà una llista buida que anirem omplint a mida que trobem peces que es puguin girar.
La funció entra en un bucle infinit, que pararà en algun dels següents casos:

- Si la cel·la a analitzar està fora del taulell
- Si no hi ha fitxa en la posició que s'analitza
- I si la fitxa que s'analitza és del mateix color del que s'afegeix.



Si no passa cap d'aquestes condicions, la cel·la s'afegirà a la llista. En el cas d'arribar a la condició 1 o 2, les fitxes que s'hagin afegit a la llista es boraran, ja que significarà que tot i tenir peces del color contrari en la trajectòria no tenim cap peça del nostre color a l'altre punt de la fila.

Finalment tornarem la llista de fitxes a girar.

Un cop hem analitzat el funcionament principal de la pantalla, hem d'analitzar el que succeeix quan l'usuari comença a arrossegar la fitxa que té disponible i la deixa anar en algun punt de la pantalla.

La vista que representa la imatge de la fitxa registra l'esdeveniment d'arrossegar-la. La funció encarregada d'enregistrar aquest esdeveniment és la funció moureFitxa.

Aquesta funció rep com a paràmetre l'esdeveniment que l'ha generat.

Aquest esdeveniment pot estar en tres situacions clau: UIGestureRecognizerStateBegan, UIGestureRecognizerStateChanged i UIGestureRecognizerStateEnded, que com és de suposar és quan comença, quan canvia i quan acaba l'esdeveniment.

Analitzem part per part. En la primera fase, comprovem l'última jugada, que ja pot ser l'última de la partida o bé l'última jugada realitzada pel jugador. Amb aquesta jugada busquem els moviments possibles i inserim unes vistes transparents en les cel·les corresponents als moviments possibles trobats. Aquestes vistes ens serviran en l'estat final de l'esdeveniment.

La segona fase, únicament movem la fitxa, 35 píxels per sobre del punt de moviment, fent així que la fitxa no quedi tapada pel dit mentre es mou. La fitxa quedarà lleugerament per sobre del dit en tot moment.

La última fase és la més important. En aquesta última fase és quan fem servir les vistes transparents inserides anteriorment. Iterem entre elles per veure si el punt on s'ha quedat la fitxa al deixar-la anar coincideix amb alguna d'elles. En cas negatiu, la fitxa torna a la posició inicial. Analitzem el que passa si coincideix amb alguna d'elles:

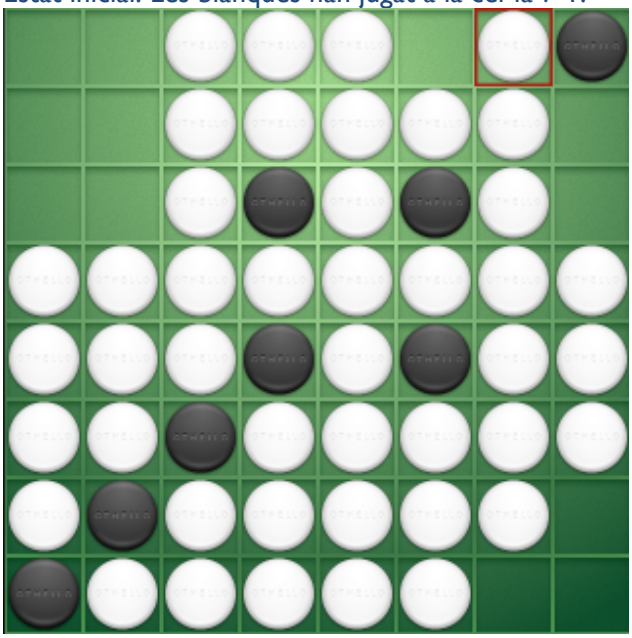
Primerament, esborrem la fitxa i afegim una nova fitxa en la posició marcada amb la funció que anteriorment hem analitzat addFitxaDeColor:atView:inCell:forCelles. És a dir, que la fitxa inserida no té res a veure amb la fitxa que hem mogut, però a nivell usuari sembla així.

Tal i com hem explicat anteriorment, aquesta funció retorna un estat final de la jugada. Amb aquest estat final, i coneixent l'estat inicial que hem fet servir per afegir la fitxa, creem una nova jugada amb totes les seves variables (estat previ, estat final i peça afegida).

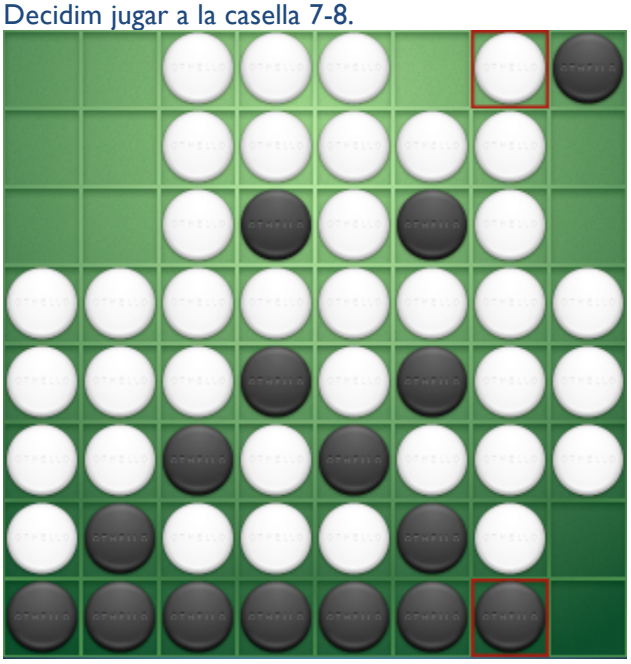
Finalment, fem una comprovació molt important. Mirem el nombre de moviments possibles propis i de l'adversari amb la nova situació. En cas que l'adversari no pugui fer més jugades, poden passar dos coses. Que l'usuari tampoc pugui fer més jugades, el que significaria que la partida s'acabaria amb aquesta jugada o que l'usuari hagi de fer una nova jugada per poder continuar. En la primera situació, es pregunta a l'usuari si realment el que vol és acabar amb la partida. En cas que decideixi que si, s'enviarà la jugada automàticament. En cas contrari, la partida es refrescaria a l'estat inicial i l'usuari hauria de tornar a fer el seu moviment. En el segon cas, l'usuari haurà de fer més moviments fins que l'adversari tingui l'opció de jugar o cap dels dos pugui jugar. Totes aquestes jugades s'afegiran a una llista que s'enviaran al servidor posteriorment.

Analitzem una situació on hem de fer múltiples jugades:

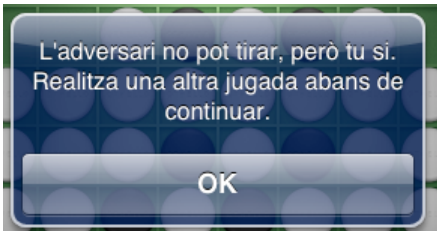
Estat inicial. Les blanques han jugat a la cel·la 7-1.



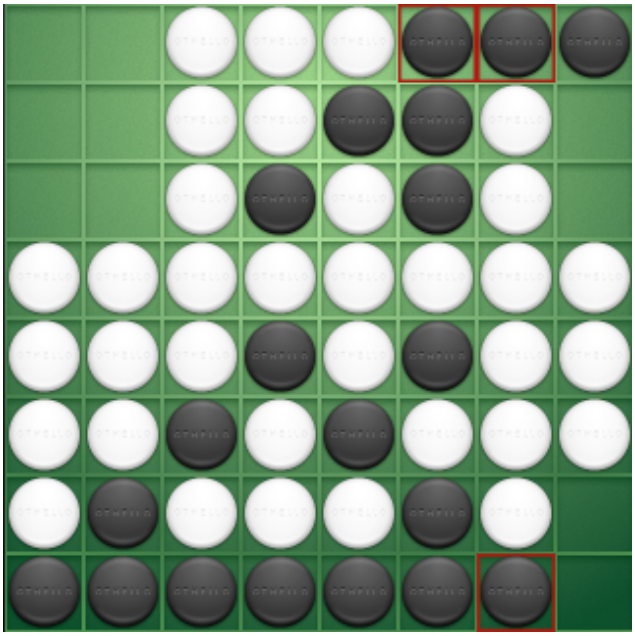
Decidim jugar a la casella 7-8.



Com podem observar les blanques no poden moure, ja que cap casella faria girar cap fitxa negra, així que el programa adverteix a l'usuari que ha de continuar movent.

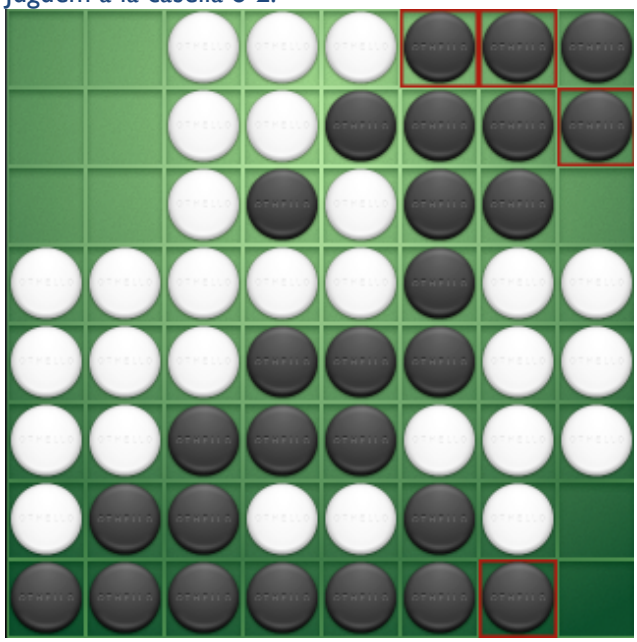


Decidim tirar a la casella 6-1:



La situació és igual. L'adversari no pot jugar i se li advertirà que ha de fer un nou moviment.

Juguem a la casella 8-2:



Tal i com veiem, ara sí que l'adversari pot fer una jugada. Pot tirar a la casella 8-3 i girar les dues fitxes 6-3 i 6-4.

Ara l'usuari ha de decidir si enviar les tres jugades o refrescar la posició inicial i tornar a fer les tres jugades o unes de diferent.

Ara ja hem analitzat el funcionament general del joc. Falta analitzar la interacció d'aquesta pantalla amb el servidor.

Interacció amb el servidor:

Un cop el jugador vol enviar les jugades, primerament se li pregunta si realment vol enviar la jugada. En cas afirmatiu, iniciem la consulta al servidor.

Iterem cada una de les jugades que hem realitzat i fem la consulta al php enviarJugada.php. Per cada iteració, fem una comprovació de si és la última jugada que s'envia. En cas afirmatiu, avisarem a l'adversari amb un missatge push.

En cada iteració, si el resultat no és OK, sortirem de la funció i enviarem un missatge d'error.

Com sempre, primerament farem la comprovació de sessió activa al servidor. Posteriorment, inserim la jugada i actualitzem el valor d'estat de la partida. En cas d'haver d'avisar enviarem el missatge push analitzarem a continuació.

11.2.3. Push

Pel correcte funcionament de la missatgeria push hem de seguir uns certs passos.

- Base de dades

Necessitarem una base de dades al servidor que registri els identificadors dels dispositius juntament amb l'anomenat "Token", que serà el que s'enviarà als servidors de Apple per tal de que ells enviïn el missatge corresponent.

El registre d'aquesta taula el farem amb el següent codi:

```
[[UIApplication sharedApplication] registerForRemoteNotificationTypes:
    (UIRemoteNotificationTypeBadge | UIRemoteNotificationTypeSound | UIRemoteNotificationTypeAlert
    )];
[self reregisterPush];
```

AppDelegate.m (Objective-C)

Aquest tros de codi comunica a l'aplicació que requereix un registre de notificacions push. Això fa que a l'iniciar l'aplicació l'usuari rep un missatge conforme si accepta rebre notificacions push. En cas afirmatiu, accedim a la funció application:didRegisterForRemoteNotificationsWithDeviceToken:. Aquesta funció reb un token i el registra al servidor fent la crida al php RegisterPush.

- Certificats

Per poder enviar missatges push al servidor d'Apple, és necessari que el socket que enviem vagi acompanyat d'un certificat. Aquest certificat es descarrega del panell de iTunesConnect i es puja al servidor.

- Enviament dels missatges

Per enviar els missatges es farà servir el php GestorMensajes.php. Aquesta php crea una classe amb funcions estàtiques.

```php
public static function sendMensajeToPartida($idPartida,$idUser,$estat) {

    $link = mysql_connect($_SESSION['bd_host'], $_SESSION['bd_user'], $_SESSION['bd_pass']);
    if(!$db = mysql_select_db("tusamigosinvisibles_com_otello", $link)){

        echo "ERROR Base de datos";
        exit ();

    }

    mysql_query("SET NAMES 'utf8'");


    $sql = "select * from usuaris where id = (select blanques from partides where id = $idPartida) "
    ."or id = (select negres from partides where id = $idPartida)";
    $nombreUser = "";
    $uid = "";
    $query = mysql_query($sql);
    while(($fila = mysql_fetch_array($query))) {
        if($fila['id']==$idUser){
            $nombreUser = $fila['nom'];
        }else{
            $uid = $fila['uid'];
        }

    }

    $mensaje = "";
    if($estat ==  "2"){
        $mensaje = "$nombreUser ha realitzat la jugada final en una de les partides que esteu jugant. "
        ."Entra i comprova el resultat final.";
    }else{
        $mensaje = "$nombreUser ha realitzat la seva jugada en una de les partides que esteu jugant. "
        ."Entra i comprova la seva jugada.";
    }



    ini_set( "display_errors", 0);

    $sql = "select token from push where uid = '$uid'";
    $query = mysql_query($sql);
    if(($fila = mysql_fetch_array($query))) {
        $token = $fila['token'];

        $token = str_replace("-"," ", $token);
        self::sendPushIphoneDev($token, $mensaje, "1", "coin.caf");
        //self::sendPushIphoneProd($token, $mensaje, "1", "default");
    }

    return true;
}
public static function sendPushIphoneProd($deviceToken, $texto, $numBadge, $sound){

    define('APNSHOST', 'gateway.push.apple.com');
    define('APNSPORT', 2195);
    define('APNSCERT', 'otelloProd.pem');

    $payload['aps'] = array('alert' => $texto , 'badge' => $numBadge, 'sound' => $sound);
    $payload = json_encode($payload);
    $streamContext = stream_context_create();
    stream_context_set_option($streamContext, 'ssl', 'local_cert', APNSCERT);
    $apns = stream_socket_client('ssl://' . APNSHOST . ':'
            . APNSPORT, $error, $errorString, 2, STREAM_CLIENT_CONNECT, $streamContext);
    $apnsMessage = chr(0) . chr(0) . chr(32) . pack('H*', str_replace(' ', '', $deviceToken))
            . chr(0) . chr(strlen($payload)) . $payload;
    fwrite($apns, $apnsMessage);
    socket_close($apns);
    fclose($apns);

}
```

GestorMensajes.php (PHP)

Aquest php, fa una recerca dins la taula d'usuaris, de l'identificador de l'adversari i del nom del que ha fet la jugada.

Posteriorment, crea un missatge depenent de l'estat de la partida i finalment busca el token dins la taula de push. En cas de trobar el token, es crida a la funció sendPushIphoneProd. Com podem veure, aquesta última funció fa crea un socket, que s'enviarà al host gateway.push.apple.com, pel port 2195 i amb el certificat que hem comentat anteriorment. A banda, se li passarà el tipus d'alerta que es vol, amb el text i el so que volem que reprodueixi.

12. Línies obertes

Personalment, crec que aquest projecte és força complet. Però com tot, sempre pot millorar-se o afegir funcionalitats.

Una possible funcionalitat que se li podria afegir en cas de tenir més temps per desenvolupar seria la opció de realitzar una recerca d'usuaris per poder jugar. Amb això, no faria falta haver d'introduir el correu exacte de l'usuari, sinó simplement realitzar una recerca i marcar un usuari.

Una altra millora possible seria la de poder quedar-se a la pantalla de la partida en el moment que ja has realitzat la jugada, i poder esperar en la mateixa pantalla fins que arribés la jugada de l'adversari. Això, tot i que no sembla massa important, podria afegir un punt de fluïdesa a l'aplicació. Requeriria molts més controls en la pantalla de la jugada, a més de controls per rebre missatgeria push dins de la pantalla de la partida. S'hauria de controlar molt bé que no es pogués donar la situació de realitzar més d'una jugada.

A part, per fer més interessant el joc, podríem realitzar algun tipus de puntuació en relació a la diferència entre els punts aconseguits i els punts de l'adversari en cada partida. Això donaria un balanç de cada partida, que es podria sumar al total de partides realitzades. Amb aquest total, podríem realitzar una espècie de ranking.

13. Conclusions

Al inici d'aquest projecte l'objectiu era realitzar un joc en el qual els usuaris poguessin realitzar partides a distància per torns. Intentant imitar la dinàmica i la metodologia de joc que tant èxit ha tingut el famós joc d'Apalabrados, he desenvolupat un joc que utilitza el joc d'Othello com a eix central. Al plantejar aquest projecte, el meu principal objectiu era el de crear un joc a distància, donant especial importància a la intercomunicació del dispositiu amb el servidor i de la manera de poder realitzar jugades a distància sense perdre la integritat de les dades. Amb tot això, no havia pensat en el fet de que a part de tota aquesta infraestructura hauria de, a més a més, desenvolupar la lògica del joc en sí. Tot i que l'Othello no és un joc especialment complicat, té forces variables a tenir en comte.

Crec que en general, la solidesa i eficàcia del joc és molt bona. La forma de tractar les partides fa que mai es puguin fer més jugades de les que pot fer l'usuari, ja que en tot moment es controla el torn en relació a les variables de la partida (l'usuari que ha efectuat la última jugada, i la possibilitat de l'adversari de jugar o no amb el tauler actual).

Pel que fa a la comunicació amb el servidor i la sincronització de les dades, s'ha tingut en compte en tot moment el fet de controlar molt bé que realment sigui l'usuari amb sessió activa el que està realitzant les jugades. Només podrà realitzar qualsevol acció un usuari que tingui un dispositiu associat en el servidor i aquest identificador coincideixi amb el que està realitzant l'acció.

Les notificacions Push són un altre dels grans elements dels que disposa aquesta aplicació. No només s'ha configurat un so especial pel joc, el qual el diferencia de la resta de joc sense ni tan sols mirar el dispositiu quan arriba la notificació, sinó que li dona molt més fluïdesa. No fa falta estar pendent del dispositiu per saber quan l'adversari ha jugat i tu pots jugar, ja que de seguida que això passi l'usuari rebrà un missatge push avisant-lo.

En general, tot i les possibles millores que s'han comentat en l'apartat de *Línies obertes*, crec que el resultat final és un joc dinàmic, robust i força addictiu.