Serveis en línia

PID_00270026

Rubén Mondéjar Andreu
Carles Pairot Gavaldà

Temps mínim de dedicació recomanat: 4 hores

Universitat Oberta de Catalunya

Rubén Mondéjar Andreu **Carles Pairot Gavaldà**

L'encàrrec i la creació d'aquest recurs d'aprenentatge UOC han estat coordinats pel professor: Javier Cánovas Izquierdo (2020)

Primera edició: febrer 2020

Av. Tibidabo, 39-43, 08035 Barcelona
Realització editorial: FUOC

Índex

Introducció

Després d'estudiar i treballar el món dels jocs multijugador, ens falta veure un pas més. Des de fa anys, hi ha plataformes que ens permeten connectar els nostres jocs i gaudir d'una sèrie de serveis per complementar els nostres projectes, siguin o no siguin jocs multijugador.

Avui, aquests serveis són molt coneguts i no és difícil trobar jocs que ens ofereixin, per exemple, superar un seguit de reptes o compartir les fites assollides amb amics.

D'altra banda, és important saber com hem aconseguit desplegar aquesta lògica remota. Aquí és on entren en joc tots els conceptes i les tecnologies del costat del servidor: els *backends,* els sistemes distribuïts, les bases de dades, la computació al núvol, etc. Veurem aquests conceptes i com s'hi integren amb els clients (els mateixos videojocs).

Per il·lustrar millor tot el que hem vist en aquest mòdul, treballarem amb un projecte basat en la integració amb una plataforma *cloud* real, tot estudiant els serveis de registre de jugadors i el servei d'estadístiques que ofereix. Això ens permetrà aprendre a integrar altres funcionalitats o serveis en línia per a qualsevol tipus de videojoc.

Objectius

En aquest mòdul didàctic es presenten a l'estudiant els coneixements necessaris per assolir els objectius següents:

1. Estudiar les plataformes de serveis en línia orientats als videojocs.

2. Estudiar el concepte de *cloud computing* i com aplicar-lo als videojocs.

3. Integrar les funcionalitats de registre d'usuaris i estadístiques en un joc multijugador que ofereix una plataforma real de tipus *cloud* amb el motor Unity.

1. Plataformes de serveis en línia

Podem afirmar sense por d'equivocar-nos que la majoria dels jugadors habituals, avui, tenen perfils en les plataformes més conegudes com Steam, Xbox Live o PlayStation Network. Aquestes plataformes ofereixen serveis amb moltes característiques, tant per als jugadors com per als mateixos jocs, i inclouen *matchmaking*, estadístiques, fites assolides, taules de classificació i partides al núvol, entre d'altres.

Com que l'ús d'aquests serveis en els videojocs actuals és tan freqüent, els jugadors esperen que cada joc, fins i tot els d'un sol jugador, s'integri d'alguna manera en un d'aquests serveis. En aquest apartat revisarem els serveis més habituals i interessants que podem trobar.

1.1. Característiques principals

Amb tantes opcions, convé considerar en quina plataforma integrarem els nostres jocs. En la majoria dels casos, l'elecció es fa en funció de la plataforma en què es llança el joc. Per exemple, tots els jocs de Xbox One han d'estar integrats en el servei de jocs de Xbox Live, encara que en aquest cas concret també se'ns permet integrar-los en altres plataformes amb Windows 10. En aquest sentit, disposem de diverses opcions per a ordinadors basats en els sistemes operatius més coneguts, com Windows, MacOS i Linux. Però, sens dubte, el servei més popular en aquestes plataformes és, avui, el servei de Valve Software, Steam. En canvi, en plataformes mòbils, tenim principalment Game Center per a iOS i Google Play Games per a Android i iOS.

Abans d'escriure qualsevol codi específic per a un servei de jugador, cal estudiar com integrar el codi en el nostre videojoc. L'opció més immediata pot ser agregar directament crides al codi de servei del jugador allà on calgui. En el nostre cas, cridaríem directament a les funcions de de Steamworks SDK en tots els arxius que necessitin utilitzar el servei de jugador. Tanmateix, hi ha dos motius pels quals no s'aconsella fer-ho:

- En primer lloc, això pot requerir que cada desenvolupador de l'equip tingui un cert grau de familiaritat amb Steamworks, ja que el codi que l'utilitza s'estendrà a través de la seva base de codi.

- En segon lloc, i el més important, això fa que sigui molt més difícil d'integrar un servei de jugador diferent en el joc. Això és particularment preocupant en el cas dels jocs multiplataforma, ja que, com s'ha esmentat, les diferents plataformes tenen diferents restriccions a l'hora de permetre utilitzar el servei de jugador.

Les principals característiques que proporcionen plataformes com Steam són les següents:

- **Autenticació**: es facilita un ventall de diferents API per administrar l'autenticació i la propietat de l'usuari.

- **Comunitat**: l'API de la comunitat és un conjunt de funcionalitats que permeten accedir a informació sobre altres jugadors, inclosa però limitada a: nom d'usuari, avatar i grups als quals pertany actualment.

- **Estadístiques i fites assolides**: proporciona un mètode fàcil i eficaç d'emmagatzemar estadístiques de joc persistents i informació relativa a l'assoliment d'objectius.

- **Marcadors**: proporciona un conjunt sòlid d'API orientades a permetre que els jugadors puguin veure el marcador o les taules de classificació de cada joc.

- **Partides en línia**: Steam Cloud representa la manera més senzilla de sincronitzar les dades dels jocs desats al núvol, i permet als seus jugadors mantenir el seu progrés en el joc sense complicacions en canviar entre dispositius, o fins i tot, després d'un desagradable bloqueig de l'ordinador.

- ***Matchmaking***: Steamworks ofereix un excel·lent conjunt d'eines per a *matchmaking* multijugador ideal tant per a jocs basats en servidors com per a jocs orientats al *lobby*.

- ***Networking***: es proporciona una capa d'abstracció de xarxa per reduir les dificultats logístiques que comporta enviar dades a través d'Internet.

- **Sistemes antitrampa**: Valve Anti-Cheat facilita una capa addicional de seguretat en les seves competitives experiències multijugador. És molt similar a un escàner de virus i té una llista de trucs coneguts que es poden detectar.

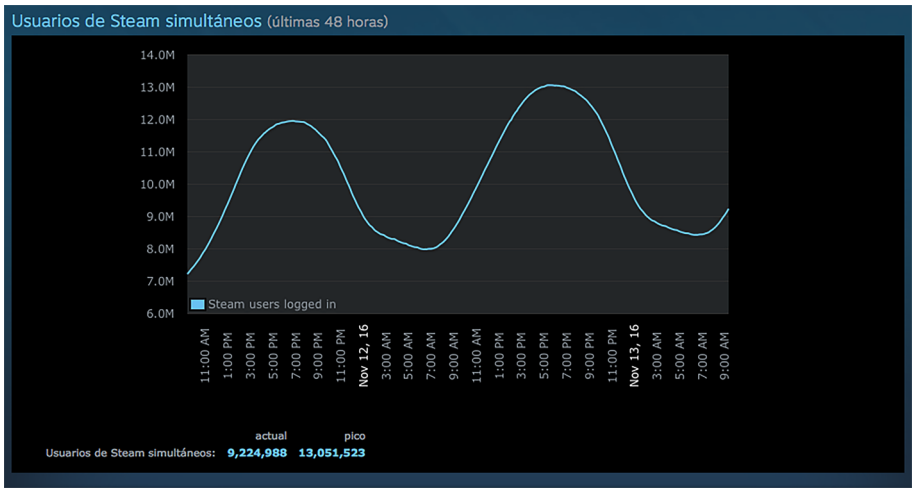
Steamworks.NET va ser dissenyat per seguir el més a prop possible de l'API nativa Valve C++ API i compta amb una cobertura del 100% de l'API Steamworks nativa en totes les interfícies.

1.2. Estadístiques

Una característica molt típica i bastant genèrica dels serveis en línia és la capacitat de recollir i mostrar diferents estadístiques dels jugadors. D'aquesta manera, és possible navegar pel nostre perfil o el d'un amic i veure com s'ha

completat cada joc; i plataformes com Steam o Google Play Games fins i tot permeten visualitzar estadístiques globals, com ara el nombre de jugadors de cada joc durant un període de temps específic.

Figura 1. Estadístiques de jugadors simultanis a Steam



Per accedir a aquesta informació, normalment hi ha alguna manera de consultar el servidor per veure les estadístiques del jugador, així com una manera d'actualitzar i escriure nous valors al servidor. Tot i que és factible utilitzar sempre el servidor com a font principal de les dades, de manera que escrivim i llegim la informació en remot, generalment és una bona idea emmagatzemar a la memòria cau els valors de la memòria local del joc.

Totes aquestes dades generades pels jugadors durant el joc i emmagatzemades als servidors serveixen per a l'anàlisi de jocs; i per a cada joc en concret es pot definir amb flexibilitat quines dades es poden recopilar, des de les més genèriques, com les hores de joc, el progrés actual o la puntuació total, fins a detalls més específics de cada joc, com el percentatge d'ús d'*ítems*, armes, tipus d'enemics eliminats, distància recorreguda, curses guanyades, etc. A partir d'aquestes dades es poden extreure mètriques com la freqüència de certs aspectes: com utilitzen els jugadors un element concret, quan arriben a un cert nivell o si realitzen alguna acció específica del joc.

Des del punt de vista del desenvolupador, aquestes dades poden ser de gran importància per millorar el joc. Per exemple, es pot ajustar el grau de dificultat de certs nivells on els jugadors es troben generalment en situacions extremes, ja sigui perquè el nivell és massa difícil o, tot el contrari, massa fàcil de completar.

A més de la informació individual, aquestes plataformes presenten diferents tipus d'estadístiques més globals, tant públiques com privades:

- **Estadístiques globals**: nombre de jugadors concurrents, totals, pics del dia en els cent jocs amb més jugadors.

- **Estadístiques de hardware i software**: el procés de recopilació de dades sobre quin tipus de hardware i software estan utilitzant els jugadors, ja sigui automàticament o a través d'enquestes, ajuda els desenvolupadors a saber qui són tècnicament els seus jugadors.

- **Geolocalització dels jugadors**: la procedència dels jugadors és primordial per poder realitzar una estratègia de màrqueting adient, així com planificar un llançament esglaonat, en futures actualitzacions o versions del joc.

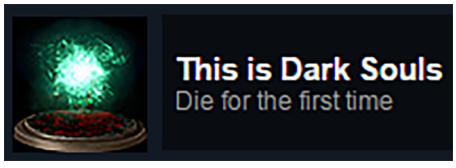
- **Compres *in-game***: saber quant i qui ha gastat diners en un joc és molt important si el sistema de monetització utilitzat es basa en aquest tipus d'estratègia comercial.

- **Informes personalitzats**: aquest servei també pot generar informes elaborats amb tota la informació recopilada de cada jugador, com les fites assolides més difícils que hagi desbloquejat, les hores totals jugades, els jocs més jugats, l'horari preferit per jugar, etc., i enviar-los automàticament perquè el jugador els vegi com un valor afegit a la plataforma.

1.3. Assoliments (*achievements*)

Una altra característica popular dels serveis dels jugadors són els assoliments. Encara que molts altres jocs individuals desenvolupen els seus propis reptes o secrets de manera interna, la primera implementació d'un sistema d'assoliments de fàcil accés i multijocs va ser l'Xbox Live, amb l'Xbox 360 Gamerscore, introduït per Microsoft el 2005. Aquest sistema va ser adaptat per altres plataformes i ha anat evolucionant tots aquests anys, tot i que encara destaca per la seva senzillesa i, avui, segueix essent molt comú i demandat pels jugadors habituals.

Figura 2. Assoliment



El seu funcionament està estretament lligat a cada joc i consisteix a desbloquejar certes gestes en el perfil del jugador després que aconsegueixi superar-les. Alguns exemples d'assoliments inclouen esdeveniments individuals, com derrotar un *boss* al final del joc o acabar-lo a un nivell concret de dificultat. Altres assoliments es donen com a estadística acumulativa en el temps, per exemple, guanyar deu combats consecutius.

Assoliment

També conegut com a trofeu, distintiu, premi, segell, medalla o desafiament, és un metaobjectiu definit fora dels paràmetres d'un joc. Aquesta funcionalitat mira d'incentivar el jugador per allargar la vida útil dels jocs.

Trofeus platí

Una variant dels assoliments complets van ser els trofeus platí introduïts en PSN, que mostren d'una manera més visible que el jugador ha completat tots els assoliments del joc.

Els assoliments són una tècnica per mirar de fer que el jugador entengui millor tot el que li ofereix el joc, d'animar els jugadors a explorar amb més èmfasi o provar estils de joc completament diferents. Els assoliments permeten als jugadors comparar el seu progrés entre si i promouen la competitivitat entre ells.

Per entendre com funcionen els assoliments, cal revisar els elements bàsics associats a cadascun d'ells:

- **Identificador**: és una cadena única generada per la plataforma per referir-se a l'assoliment en els seus clients de jocs.
- **Nom**: és un nom curt de l'assoliment (per exemple, «Sense rival»).
- **Punts**: la quantitat d'experiència que s'afegirà al perfil del jugador.
- **Descripció**: és una descripció concisa de l'assoliment per ajudar el jugador a entendre de què es tracta (per exemple, «acaba el joc en la dificultat més alta»).
- **Icona**: imatge associada a l'assoliment.

La llista d'assoliments d'un joc normalment mostra la icona, els punts i el nom de cada assoliment que hagi estat desbloquejat; en canvi, els assoliments bloquejats o ocults apareixen amb menys informació.

Cada joc té una puntuació màxima (per exemple, 1.000 punts, i cada assoliment hi aporta una part –entre 5 i 200 punts, per exemple).

D'altra banda, els assoliments poden ser designats com a estàndards o incrementals. En general, un assoliment incremental implica que un jugador progressi gradualment fins a aconseguir l'assoliment durant un període de temps més llarg. A mesura que el jugador avança cap a l'assoliment incremental, el joc ha d'informar del progrés parcial del jugador a la plataforma de serveis. En qualsevol cas, un cop informada la plataforma, aquesta s'encarrega de validar l'assoliment i, en cas afirmatiu, retornar la resposta al joc perquè aquest pugui notificar al jugador en quin moment assoleix la fita i progressa al següent nivell.

1.4. Classificacions (*leaderboards*)

Les **taules de classificació** o els **marcadors** són una manera de facilitar classificacions per a determinats aspectes d'un joc, com ara una puntuació o un temps per completar un nivell en concret.

En general, les taules o els marcadors es poden consultar tant en termes de rànquing com en rangs concrets, per exemple, en relació amb els amics del jugador en la mateixa plataforma.

Per exemple, els marcadors de Steam, es poden crear a través del lloc web de gestió per a desenvolupadors de Steamworks o es poden crear amb la programació d'una crida del seu SDK. Aquesta manera de treballar és molt freqüent a la resta de les plataformes.

El procés típic funciona de la manera següent. Al final d'un joc (o en un moment que s'hagi determinat), el joc envia la puntuació del jugador a una o més taules de classificació que s'hagin creat. Els serveis de joc comproven si aquesta puntuació és millor que l'entrada de classificació actual del jugador per a la puntuació diària, setmanal o de tots els temps. Si és així, els serveis de jocs actualitzen les taules de classificació corresponents amb la nova puntuació.

Per recuperar els resultats d'un jugador per a una taula de classificació, es pot sol·licitar un marc de temps (diari, setmanal o de tots els temps) i especificar si l'usuari desitja veure una taula de classificació social o pública. El servei de jocs realitza tot el filtrat necessari i, després, envia els resultats al client.

Un aspecte interessant de les taules de classificació és la possibilitat d'associar o pujar contingut generat pel joc associat amb una entrada de taula de classificació. Per exemple, una *speedrun* d'un nivell podria tenir una captura de pantalla associada o un vídeo que mostrés com s'ha aconseguit.

Speedrun

És una competició que té com a principal objectiu acabar un videojoc el més ràpid possible, generalment amb la màxima dificultat.

Alternativament, un joc de curses o aventura podria proporcionar corredors fantasma d'altres jugadors, que es poguessin descarregar i competir contra ells. Això permet maneres de fer que les taules de classificació siguin més interactives i que no es limitin simplement a llistar les millors puntuacions.

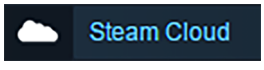
Figura 3. Marcadors i mode Time Trial amb fantasma de *Mirror's Edge*

Les taules de classificació poden ser una altra manera d'impulsar la competència entre els jugadors, tant per aficionats més *hardcore*, que lluitaran pel primer lloc en una classificació pública, com per als jugadors més *casual*, que estaran interessats a comparar el seu progrés entre els seus amics.

1.5. Persistència al núvol (*cloud saving*)

El servei ofereix una manera senzilla de guardar la progressió dels jugadors de manera remota. Concretament, aquest servei us permet sincronitzar les dades del joc d'un jugador en diversos dispositius. En integrar el nostre videojoc amb aquest servei podem recuperar les dades guardades per als jugadors, els quals poden seguir amb el joc a partir de l'últim punt desat.

Figura 4. Un exemple de persistència al núvol és el servei de Steam Cloud



Per exemple, si un joc s'executa en una consola, es pot utilitzar el servei de desats per permetre que un jugador inicií el seu joc en una altra consola o fins i tot un ordinador, sempre que ambdós permetin connectar-se al mateix servei (per exemple, Xbox Live) i després seguir jugant sense perdre cap pro-

grés. Aquest servei també es pot utilitzar per assegurar que el joc d'un jugador continua des d'on el va deixar, fins i tot si el seu dispositiu es perd, es trenca o el renova per un altre de més modern.

L'espai que ocupen sol ser reduït, per tant, aquest servei no té un cost addicional per la quantitat de dades emmagatzemades al núvol. Normalment, aquest espai ocupat s'associa amb la quota del jugador, que està convenientment dimensionada.

Els jocs poden seguir funcionant a la seva manera tradicional, tot llegint i escrivint les partides en local, especialment quan el dispositiu del jugador està fora de línia. La idea és que el joc, en detectar la connexió amb la plataforma, executi un procés de sincronització perquè de manera asíncrona les dades del joc siguin consistents en els servidors remots.

Òbviament, en utilitzar aquest tipus de servei, el joc pot trobar conflictes a l'hora de desar dades, ja que el jugador pot haver jugat en un dispositiu *offline* mentre ho feia alhora en un altre dispositiu en línia. En general, la millor manera d'evitar conflictes de dades és carregar sempre les dades més recents del servei quan l'aplicació s'inicia o es reprèn i es desen les dades al servei amb una freqüència raonable. No obstant això, no sempre és possible evitar conflictes de dades.

El joc ha de fer tot el possible per gestionar els conflictes de manera que les dades dels seus usuaris es conservin i que tinguin una bona experiència. Aquesta situació també ens la podem trobar amb altres serveis com marcadors o assoliments, de manera que es pot concentrar la sincronització de dades en un mateix procés, encara que, naturalment, això és molt més crític en el cas de les partides en tractar-se d'informació més sensible i específica.

1.6. *Matchmaking*

Aquest mecanisme està estretament lligat als jocs multijugador en què cal aparellar o agrupar els jugadors per fer partides equilibrades o no tant, públiques o privades, etc. El procés bàsic de preparació per jugar a un joc multijugador s'esdevé aproximadament com segueix:

- El joc busca un *lobby* basat en paràmetres personalitzables del joc. Aquests paràmetres poden incloure modes de joc o fins i tot nivell d'habilitat (si es realitza un *matchmaking* basat en habilitats).

- Si es troben un o més *lobbies* adients, el joc en selecciona un automàticament o el jugador el pot seleccionar d'una llista. Si no es troba cap *lobby*, el joc pot optar per crear-ne un per al jugador. En qualsevol cas, quan s'hagi trobat o creat el *lobby*, el jugador s'hi pot adherir.

- Al *lobby,* és possible configurar-hi encara més els paràmetres del joc següent, com ara personatges, mapa, etc. Durant aquest període, altres jugadors probablement s'uniran al mateix *lobby.* També és possible enviar-se missatges de xat entre ells, mentre els jugadors es troben al mateix *lobby.*

- Un cop el joc està preparat per començar, els jugadors s'uneixen al seu joc i surten del *lobby.* Normalment, això implica connectar-se a un servidor de joc (ja sigui un servidor dedicat o un servidor de joc). En el cas de no tenir cap servidor, els jugadors comencen a comunicar-se entre ells mitjançant connexions *peer-to-peer* abans de sortir del *lobby.*

Servidor de joc

Un servidor de joc s'encarrega de simular el món del videojoc tot permetent que els clients que hi ha connectats mantinguin consistent la seva pròpia versió del món. Així doncs, el servidor transmet als clients les dades sincronitzades amb aquesta finalitat. En canvi, els **servidors de joc dedicats** s'encarreguen de simular el món sense donar suport a dispositius d'entrada i sortida, excepte per a la seva administració. En aquest cas, doncs, els jugadors **no** poden jugar directament en un servidor de joc dedicat. Ho hauran de fer en un dels clients que hi estiguin connectats.

Tot seguit, estudiarem un sistema concret de *matchmaking* per entendre en què consisteix, quines particularitats o tècniques s'han definit i com s'aplica.

Equilibri en *matchmaking*

Predir la probabilitat de cada resultat del joc millora la competitivitat *matchmaking*, fet que permet reunir equips amb habilitats més equilibrades.

Basat en el sistema de puntuació Glicko, un mètode per avaluar l'habilitat d'un jugador en jocs d'habilitat, el conegut sistema TrueSkill compta amb un algoritme bayesià de classificació i sistema de *matchmaking* desenvolupat per Microsoft Research i establert en els serveis en directe de Xbox 360. Amb TrueSkill, un jugador acabat d'arribar a la plataforma es pot classificar correctament amb menys de vint partides. A TrueSkill, el rang d'un jugador es representa com una distribució normal $\aleph$ caracteritzada per un valor mitjà de μ (habilitat percebuda) i una variància de σ (la seguretat dipositada al valor μ).

TrueSkill

Utilitza un model matemàtic d'incertesa per abordar debilitats en sistemes de classificació existents.

Quan es produeix una coincidència, el sistema prova d'agrupar individus segons el seu nivell estimat d'habilitat. Si dos individus competeixen cara a cara i tenen el mateix nivell estimat d'habilitat amb una incertesa d'estimació baixa, cadascun té una probabilitat d'un 50% de guanyar una partida. D'aquesta manera, el sistema prova de fer que cada partida sigui tan competitiva com sigui possible.

Per evitar l'abús del sistema, la majoria dels jocs classificats tenen opcions relativament limitades per a *matchmaking*. Per disseny, els jugadors no poden jugar fàcilment amb els seus amics en els jocs classificats. Tanmateix, aques-

tes contramesures poden fracassar per tècniques com ara comptes alternatius i errades del sistema en què cada sistema té la seva pròpia qualificació TrueSkill individual. Per proporcionar jocs menys competitius, el sistema també permet jocs amb partides sense classificar, en què s'aparellen jugadors de qualsevol nivell d'habilitat estan aparellats, sense que aquests contribueixin a la qualificació de TrueSkill.

1.7. Altres serveis

A part de tots aquests serveis, en podem trobar alguns de similars o més aviat específics de xarxa, que solen diferir més segons la plataforma escollida. En aquest últim apartat veurem breument en què consisteixen els serveis que faciliten una cobertura per a la comunicació en xarxa entre els jugadors. En el cas de Steamworks o Google Play Games, es proporciona una sèrie de funcions per enviar paquets a altres jugadors tot creant una xarxa *peer-to-peer*.

Concretament, aquesta vegada prendrem d'exemple l'API multijugador en temps real de Google Play Games. Aquesta és suficientment flexible perquè puguem utilitzar-la per reescriure el nostre joc i fer que totes les instàncies enviïn les dades a través de la xarxa *peer-to-peer* subjacent.

Un tema important a tenir en compte és que per designar l'únic client que actua de *host* i establir la seva autoritat en les dades del joc, aquest serà l'encarregat de centralitzar la comunicació i transmetre dades als altres participants connectats a través del sistema de missatgeria de dades de Google Play Games.

A més, una vegada superada la fase de *matchmaking* per crear la partida, si la lògica del joc es basa en l'existència d'un «amfitrió» «propietari» del joc, serà el joc i no la plataforma el responsable d'implementar l'algorisme per determinar qui és el *host*.

Elecció de líder

Per triar el *host* es pot fer ús d'un algoritme d'elecció de líder.

Si reescrivim el nostre joc amb aquest propòsit, podem utilitzar els serveis de jocs de Google Play per transmetre dades als participants en una partida o permetre'ls intercanviar missatges entre ells. Com amb altres serveis, els missatges de dades es poden enviar utilitzant un protocol de missatgeria fiable o poc fiable facilitat pels serveis de la plataforma:

- **Missatgeria fiable**: lliurament de dades, la integritat i l'ordre estan garantits. Podem escollir notificar l'estat de lliurament mitjançant un retorn de crida. La missatgeria fiable és adequada per enviar dades no sensibles al moment. També es pot utilitzar missatgeria fiable per enviar grans conjunts de dades en què les dades es poden dividir en segments més petits, són enviades a través de la xarxa i després assemblades novament pel client receptor. Aquest tipus de missatgeria pot tenir una latència alta.

- **Missatges poc fiables**: el client de joc envia les dades només una vegada (*fire-and-forget*) sense cap garantia de lliurament de dades o dades que arribin en ordre. Tanmateix, la integritat està garantida, de manera que no cal afegir un codi de verificació o *checksum*. Aquest tipus de missatgeria té una latència baixa i és adequat per enviar dades sensibles al moment. La seva aplicació és responsable de garantir que el joc es comporti correctament si els missatges es descarten en la transmissió o si es reben fora d'ordre.

Òbviament, si un client que envia dades no està connectat als serveis de jocs de Google Play o el destinatari no està connectat, el missatge no es lliurarà. Per preservar les transmissions de missatges i evitar superar els límits de velocitat, les pràctiques recomanades per enviar dades són les següents:

- Enviar missatges només als participants que requereixin la informació, en comptes de transmetre-la a tots els participants. Si s'envia un missatge de difusió, cal excloure el participant del remitent de la llista de destinataris de difusió.

- Si s'envien dades amb el protocol de missatgeria fiable, cal mantenir la freqüència de les transmissions de missatges per sota d'un límit raonable (per exemple, cinquanta missatges per segon). Si es necessita enviar dades amb més freqüència, és recomanable utilitzar missatgeria no fiable.

2. *Cloud computing*

Des de fa uns quants anys i gràcies a la computació al núvol, s'ha facilitat molt que els desenvolupadors puguin allotjar, gestionar i mantenir serveis remots. Fins i tot els petits estudis es poden permetre els seus propis servidors dedicats. En aquesta secció veurem en què consisteix aquest paradigma, i com podem aprofitar-nos-en perquè el nostre joc, part d'ell o els serveis associats s'executin en servidors remots.

2.1. Descripció conceptual

Les plataformes al núvol han adquirit molta rellevància aquests anys gràcies als seus avantatges inherents, de manera que han transformat els serveis de *hosting* tradicional en una altra manera de vendre's al gran públic, en bona mesura gràcies a facilitar un servei de baix cost de gestió i administració de les infraestructures de hardware.

Un altre motiu important per adoptar els sistemes al núvol és la seva capacitat d'escalar ràpidament en ambients de ràpid creixement amb càrregues de treball massives. Assolir l'escalabilitat desitjada d'una manera flexible és una tasca complexa, que implica resoldre molts problemes concurrents, com mantenir l'equilibri de càrrega dels recursos, polítiques de replicació i coherència, i escalabilitat horitzontal de magatzems de dades persistents.

En aquests casos, els proveïdors de *cloud* han d'oferir *frameworks* amb API específiques per a serveis al núvol com per exemple Microsoft Azure, Amazon EC2 o Google App Engine. Les tecnologies al núvol van començar a emergir quan Amazon va irrompre al mercat el 2006 oferint els seus serveis web. De fet, Amazon s'ha convertit en un estàndard de núvol *de facto*, tot i que ara moltes altres companyies aporten solucions en un ecosistema empresarial molt actiu. Però hi ha diferents tipus de núvols i maneres de diferenciar-los segons la seva naturalesa.

D'acord amb la definició àmpliament utilitzada pel NIST ((National Institute of Standards and Technology):

> El *cloud computing* és un model per habilitar l'accés a la xarxa de manera omnipresent, convenient i sota demanda d'un conjunt compartit de recursos computacionals que es poden configurar (per exemple, xarxes, servidors, emmagatzematge, aplicacions i serveis), aprovisionar i alliberar ràpidament amb un mínim esforç de gestió o interacció amb el proveïdor del servei.

En funció de la capacitat facilitada al consumidor podem distingir entre tres models de servei:

- **Software com a Servei (SaaS)**: utilitzar les aplicacions del proveïdor que s'executen en una infraestructura al núvol. Les aplicacions són accessibles des de diversos dispositius client a través d'una interfície de client lleuger, com un navegador web (per exemple, correu electrònic basat en web) o una interfície de programa. El consumidor no gestiona ni controla la infraestructura subjacent del núvol, inclosa la xarxa, els servidors, els sistemes operatius, l'emmagatzematge o fins i tot les capacitats de les aplicacions individuals, amb la possible excepció dels paràmetres de configuració específics de l'aplicació de l'usuari.

- **Plataforma com a Servei (PaaS)**: desplegar en la infraestructura del núvol les aplicacions creades o adquirides pel consumidor tot utilitzant llenguatges de programació, biblioteques, serveis i eines suportades pel proveïdor. El consumidor no gestiona ni controla la infraestructura subjacent del núvol, els servidors, els sistemes operatius o l'emmagatzematge, però té control sobre les aplicacions desplegades i possiblement les configuracions per a l'entorn d'allotjament d'aplicacions.

- **Infraestructura com a Servei (IaaS)**: proveir processament, emmagatzematge, xarxes i altres recursos de computació fonamentals en què el consumidor és capaç de desplegar i executar software arbitrari, que pot incloure sistemes operatius i aplicacions. El consumidor no gestiona ni controla la infraestructura subjacent del núvol, sinó que té el control sobre els sistemes operatius, l'emmagatzematge i les aplicacions implementades, i possiblement un control limitat de components de xarxa selectes (per exemple, *firewalls* de *host*).

En funció de l'ús de la infraestructura del núvol, podem classificar els models d'implementació en els quatre següents:

- **Núvol privat**: ús exclusiu per una sola organització que comprèn múltiples consumidors (per exemple, unitats de negoci). Pot ser de propietat, administrada i operada per l'organització, un tercer o una combinació dels mateixos, i pot existir dins o fora de les instal·lacions.

- **Núvol públic**: ús obert pel públic en general. Pot ser de propietat, administrada i gestionada per una organització comercial, acadèmica o governamental, o alguna combinació d'aquestes. Se situen físicament en les instal·lacions del proveïdor del núvol.

- **Núvol híbrid**: composició de dues o més infraestructures de núvol diferents que segueixen essent úniques, però que estan unides per una tecnologia estandarditzada o propietària que permet la portabilitat de dades i aplicacions.

2.2. Característiques dels sistemes *cloud*

En els primers temps dels jocs en línia, l'allotjament dels mateixos servidors dedicats era una tasca molt complexa, començant per l'adquisició i el manteniment de grans quantitats de hardware, infraestructura de xarxes i personal de TI. L'estimació dels jugadors en el llançament era extremadament important, ja que una infraestructura sobredimensionada podia significar un cost addicional impossible de mantenir. Però encara pitjor era quedar-se curt i que els jugadors paguessin sense poder-se connectar a causa de restriccions de processament i amplada de banda.

El cas *Pokémon Go*

Encara avui, és difícil estimar el nombre d'usuaris potencials que pot atreure un joc. De fet, el 2016, Niantic va subestimà en una escala de magnitud (x10) els jugadors de *Pokémon Go*. El maig de 2018 la xifra d'usuaris actius d'aquest joc era de 147 milions.

Però com hem vist, el núvol ha resolt aquests problemes. Gràcies a l'abundant poder de processament disponible dels gegants d'Internet, com Amazon, Microsoft i Google, les companyies de jocs són capaces de gestionar aquests costos de manera molt més raonable. Altres serveis de tercers, com Heroku, faciliten la implementació amb serveis de gestió de servidors i bases de dades segons calgui. A més, hi ha serveis orientats directament als videojocs, com és el cas d'Unity Cloud Services o Photon Cloud, que ens permeten allotjar els jocs amb serveis més orientats i dedicats a les necessitats específiques dels videojocs.

Malgrat el reduït cost inicial del costat del servidor, encara trobem alguns inconvenients potencials que hem de considerar:

- **Complexitat**: utilitzar un grup de servidors dedicats és una tasca complexa, malgrat que el núvol proporciona la infraestructura i part del software d'administració. Però a més es requereix codi personalitzat d'administració de processos i màquines virtuals, així com adaptar-se a les APIs canviant.

- **Cost**: malgrat que el núvol disminueix el cost inicial i a llarg termini de manera significativa, no és totalment gratuït. L'augment de l'interès dels jugadors pot cobrir l'augment del cost, però depèn de cada cas.

- **La dependència de tercers**: hostatjar el joc als servidors d'Amazon, Microsoft o Unity, significa que el destí descansa a les seves mans. Tot i que les empreses ofereixen acords de nivell de servei que garanteixen un temps mínim d'inactivitat, encara hi ha un risc associat.

- **Canvis inesperats de hardware**: els proveïdors de *hosting* generalment garanteixen facilitar hardware que compleix amb certes especificacions mínimes. Això no els impedeix canviar de hardware sense previ avís, sempre que estigui per sobre de l'especificació mínima. Si de sobte introdueixen una configuració de hardware estranya que no s'ha provat, pot causar problemes.

Tot i que aquests inconvenients poden ser significatius, els beneficis tot sovint les superen:

- **Servidors confiables, escalables i de gran amplada de banda**: en models tradicionals no hi ha cap garantia que els jugadors amfitrions siguin els que tindran la millor amplada de banda quan els altres jugadors vulguin jugar. Amb l'allotjament al núvol i un programa d'administració de servidors, es pot activar qualsevol servidor a demanda per realitzar aquesta mena de tasques i garantir el bon funcionament de la partida.

- **Prevenció de trampes**: amb el control de tots els servidors és fàcil assegurar-se que estiguin executant versions no modificades i legítimes dels jocs. Això significa que tots els jugadors obtenen una experiència uniforme i fiable. A més, no només permet classificacions reals, sinó també que els jocs puguin utilitzar tots els serveis de la plataforma.

- **Protecció de còpia raonable**: la restricció de què els jocs siguin executats en servidors dedicats proporciona una forma de DRM (*Digital Rights Management*) *de facto,* no intrusiva. El fet de no alliberar executables del servidor dificulta el desplegament de servidors pirates o el desbloqueig de contingut de manera il·legal. També permet comprovar les credencials d'inici de sessió per a cada jugador, tot assegurant-se que realment té dret a jugar al joc.

Com a conclusió, aquests darrers anys els sistemes *cloud* han anat guanyant terreny en les solucions tradicionals d'infraestructura dedicada, principalment per la seva confiabilitat, la seva escalabilitat i el seu cost associat. Aquests aspectes són determinants a l'hora de facilitar la millor experiència multijugador als usuaris dels nostres videojocs en línia.

2.3. Integració amb sistemes *cloud*

El desenvolupament de servidors *backend* és un camp en evolució ràpida i constant, consistent en un conjunt d'eines que evolucionen ràpidament.

Afortunadament, hi ha molts llenguatges, plataformes i protocols dissenyats per facilitar la vida al desenvolupador de *backend*. Els *framewoks* moderns de desenvolupament web es poden veure com un conjunt de components que simplifiquen la programació HTTP, tot controlant automàticament els detalls de transport de baix nivell.

A més, els darrers anys, s'ha fet manifest que HTTP també és útil per crear una API estàndard via rest, tot utilitzant els verbs (GET, POST, etc.) i codis (200, 500, etc.) facilitats, a més d'altres conceptes com URI i encapçalaments.

Aquest estil arquitectònic anomenat RESTful ha demostrat ser una manera eficaç d'aprofitar HTTP, encara que certament no és l'únic enfocament vàlid. Actualment, hi ha una tendència clara a fer que els serveis utilitzin API REST i dades JSON, juntament amb *WebSockets* o *Server-Sent Events* (SSE) per complementar els serveis més interactius.

Aquestes són eines flexibles i àmpliament acceptades per al desenvolupament de servidors. És viable triar diferents eines per a la nostra plataforma *cloud* de serveis però clarament, per poder realitzar la integració dels nostres videojocs, es requereix una comprensió bàsica d'HTTP, JSON, REST, *WebSockets* i SSE, així com les principals tecnologies web involucrades.

2.3.1. L'estàndard JSON

A finals dels anys noranta i principis dels 2000, el llenguatge XML (*eXtensible Markup Language*) es va anunciar com el format universal d'intercanvi de dades que canviaria la manera d'estructurar documents. Al principi, moltes tecnologies van començar a utilitzar-lo, ja que durant aquest període va ser l'única opció de compartir dades lliurement.

Avui, JSON (*JavaScript Object Notation*) ha anat guanyant terreny com a llenguatge d'intercanvi de dades universal, principalment gràcies a la seva lleugeresa (el seu format és extremadament simple i ocupa menys caràcters que l'XML) i a la gran velocitat del seu processament (es pot interpretar directament com un objecte JavaScript).

Les dades en JSON es representen en forma de mapa, emmagatzemant parells d'informació clau/valor, mentre que en XML aquestes dades es representen com un arbre, cosa que implica que el seu processament sempre és més tediós i menys eficient en termes de cost d'espai (ús de memòria) i temps.

A efectes pràctics, JSON s'utilitza molt en els documents associats a les sol·licituds (*requests*) web (encapçalaments HTTP que especifiquen el *content-type* d'aplicació com a JSON) tant d'escriptura (creació POST, actualització PUT o actualització parcial PATCH), com de consulta (GET).

La utilitat principal des del punt de vista dels videojocs multijugador és facilitar les tasques de serialitzar objectes a documents textuals i viceversa, entre clients i servidors. Tot seguit veurem com es treballa amb aquest format en C#, a partir de la noció de JSON «estructurat», cosa que significa que s'ha d'indicar quines variables s'emmagatzemaran en les seves dades JSON tot creant una classe o estructura. Per exemple:

```
[Serializable]
public class MyClass {
    public int level;
    public float timeElapsed;
```

```
    public string playerName;
}
```

Amb la definició d'una classe C# simple que conté una sèrie de variables (per exemple, *level*, *timeElapsed*, i *playerName*) i que marquem com a *Serialitzable*, fet necessari per poder utilitzar el serialitzador JSON. Seguidament, crearem una instància d'aquesta classe i la serialitzarem en format JSON amb la funció *JsonUtility.ToJson()*:

```
MyClass myObject = new MyClass();
myObject.level = 1;
myObject.timeElapsed = 47.5f;
myObject.playerName = "Dr Charles Francis";
string json = JsonUtility.ToJson(myObject);
```

Això resultaria en la variable *json* que conté la cadena de caràcters:

```
{"level":1,"timeElapsed":47.5,"playerName":"Dr Charles Francis"}
```

Finalment, per convertir el JSON novament en un objecte, podem utilitzar la funció *JsonUtility.FromJson()*:

```
myObject = JsonUtility.FromJson<MyClass>(json);
```

D'aquesta manera, podem entendre la senzillesa de treballar amb aquesta eina, essent molt més complexa la tasca d'enviar i rebre dades via HTTP, REST o *WebSockets* que la de codificar-les en format JSON.

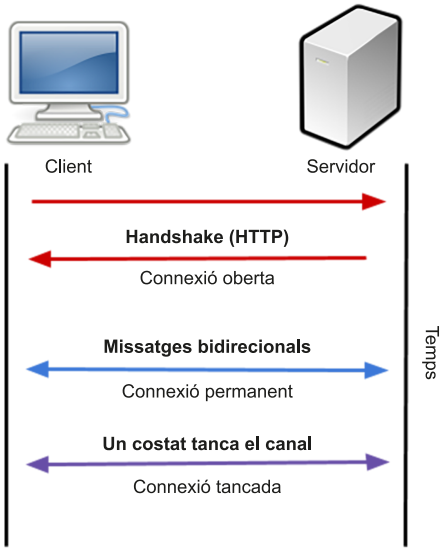
2.3.2. El protocol *WebSocket*

WebSockets és una extensió de la idea de *socket* tradicional, motivat per les carències inherents en el protocol HTTP, el qual va ser dissenyat per a la World Wide Web i, per tant, per ser utilitzat pels navegadors.

En tractar-se d'un protocol específic que funciona d'una manera particular, no és adequat per a cada necessitat, concretament ens referim a com HTTP gestiona les connexions, on es realitza una connexió per a cada sol·licitud, per exemple, per descarregar un document HTML o una imatge. S'obre un port/*socket* i es transfereixen les dades tot tancant la connexió just en acabar. Aquesta contínua obertura i tancament de connexions HTTP pot crear una sobrecàrrega i per a certes aplicacions –especialment les que requereixen respostes ràpides, interaccions en temps real o mostrar fluxos de dades– simplement no funciona.

L'altra limitació amb HTTP és que pertany al **paradigma *pull***, en el qual és l'interessat, en aquest cas el navegador, que sol·licita o extreu informació de l'origen, en aquest cas els servidors. El problema aquí és que el servidor només és capaç d'enviar dades al navegador en el moment de respondre una sol·licitud. Això significa que els navegadors han de consultar el servidor de manera contínua per obtenir nova informació tot repetint les sol·licituds cada pocs segons o minuts per esbrinar si hi ha noves dades.

Figura 5. Diagrama de connexió *Websocket*



A finals de la dècada del 2000, va sorgir un moviment per afegir *sockets* als navegadors, essent el 2011 l'any que el protocol *Websocket* es va estandarditzar. Això va permetre als desenvolupadors utilitzar aquest protocol, que gràcies a la seva flexibilitat permet crear noves formes de comunicació per transferir dades des de i cap a servidors des del navegador, així com comunicació *peer-to-peer* (P2P) o comunicació directa entre navegadors. La principal diferència amb el protocol HTTP és que un *WebSocket* roman obert al servidor tenint sempre un canal disponible per a la comunicació bidireccional. Això significa que les dades poden ser enviades al navegador en temps real a demanda, tot canviant la manera de treballar del paradigma *pull* al **paradigma *push***.

2.3.3. *Server-Sent Events* (SSE)

A part dels *WebSockets,* hi ha una altra metodologia per treballar amb el **paradigma *push***, de manera que el servidor enviï dades al navegador: això és mitjançant *Server-Sent Events* (SSE).

Estrictament parlant, el protocol HTTP no permet que el servidor enviï informació de manera activa. Tanmateix, hi ha una excepció: si el servidor declara al client que allò que enviarà a continuació és un flux (*stream*) de dades.

En altres paraules, que allò que enviarà de manera contínua és un *stream* en lloc d'un únic paquet de dades. D'aquesta manera, el client no tancarà la connexió i es quedarà a l'espera de rebre més dades de l'*stream* que el servidor li enviarà. Un exemple típic n'és la reproducció de vídeo. Essencialment, l'objectiu d'aquest tipus de comunicació és aconseguir la descàrrega d'un fitxer mitjançant *streaming*.

SSE utilitza aquest mecanisme per enviar informació de *tipus push* a través d'una *seqüència* al navegador. Òbviament es basa en el protocol HTTP i està estandarditzada com a part de HTML5 pel *Consorci* World Wide Web (W3C) i recolzat per tots els navegadors, excepte Internet Explorer / Edge.

SSE és força semblant als *WebSockets* en el sentit que ambdós estableixen un canal de comunicació entre el navegador i el servidor, i el servidor envia la informació al navegador. En general, els *WebSockets* són més potents i flexibles, ja que estableixen un canal de comunicació *full-duplex*, que permet la comunicació en ambdues direccions; mentre que SSE és un canal unidireccional que només permet al servidor enviar dades al navegador, perquè l'*streaming* és essencialment una descàrrega. Si el navegador necessita enviar informació al servidor, llavors realitza una altra petició HTTP.

Sigui com sigui, SSE té els seus avantatges respecte als *WebSockets*:

- SSE utilitza el protocol HTTP que està suportat per tots els servidors, mentre que *WebSockets* és un protocol independent.

- SSE és més lleuger i senzill d'utilitzar, mentre que el protocol *WebSocket* és relativament complex.

- SSE suporta les reconnexions per defecte, mentre que si fem servir *WebSockets* hem d'implementar aquesta funcionalitat nosaltres mateixos.

- En general, SSE només s'utilitza per transferir text; les dades binàries s'han de codificar com a text abans de poder-les transmetre, mentre que el protocol *WebSocket* suporta transferències binàries per defecte.

2.3.4. L'arquitectura REST

REST, o *REpresentational State Transfer*, és una proposta per crear API per a aplicacions de manera metodològica.

En les aplicacions web típiques i tradicionals, la creació d'*endpoints* REST mitjançant HTTP és la manera com es dissenya la gran majoria de les aplicacions. Qualsevol de les diverses tecnologies disponibles (per exemple, JavaScript) és molt similar en la manera com rep les sol·licituds d'informació i la manera com després les respon.

REST organitza aquestes peticions de manera predictible, tot utilitzant tipus d'operacions HTTP, o verbs, per construir respostes adequades. Les peticions s'originen en el client i els verbs HTTP inclouen GET, POST, PUT, DELETE, entre d'altres, que es corresponen amb operacions esperades, recuperació de dades, enviament de dades, actualització de dades i supressió de dades, respectivament.

REST és la manera més estesa d'estructurar l'API per a peticions web. Podem trobar fàcilment exemples en moltes aplicacions web que fan ús d'aquest sistema i exposen la seva API per poder realitzar peticions. Un exemple típic seria Twitter, que a més ens permet vincular-lo amb *OAuth* per poder utilitzar el seu sistema d'usuaris des de la nostra aplicació i accedir posteriorment a les seves dades. Això seria útil per poder tuitar els resultats de les nostres partides de manera automàtica des del joc.

OAuth

És un sistema basat en REST que permet l'autenticació utilitzant sistemes de tercers (per exemple, Google o Facebook) des d'una aplicació.

Però com que implica l'ús d'HTTP, també té associada la sobrecàrrega d'aquest protocol. Per a la majoria de les aplicacions, la informació només ha de ser transferida quan un usuari realitza una acció. Per exemple, en navegar per un lloc de notícies, un cop el navegador ha sol·licitat l'article, l'usuari està ocupat llegint-lo i no realitza altres accions. En aquests casos, la pràctica habitual sol ser la de tancar el *socket* per estalviar recursos en el servidor.

En qualsevol cas, si el tipus d'interacció és més exigent, com per exemple les aplicacions en temps real o les transferències de dades, la combinació HTTP i REST no és la més adient i segurament haurem d'optar per altres alternatives, com *WebSockets* o SSE.

3. Projecte: *Tanks!* En línia

En aquest projecte, integrarem el joc *Tanks!* amb un conjunt de serveis d'una plataforma *cloud* real que proporciona una sèrie de funcionalitats molt útils per als videojocs en línia. La motivació principal d'aquest exemple és veure com utilitzar les eines actuals per poder afegir funcionalitats remotes en els nostres desenvolupaments, com ara comunitats, suport a transmissions en directe, mons persistents, algoritmes de física o d'intel·ligència artificial distribuïts, etc. Encara que el propòsit del projecte no és tant afegir un conjunt de funcionalitats concretes, serveix de justificació per veure com integrar el nostre joc amb un proveïdor de serveis en línia.

3.1. PlayFab

Com hem vist, actualment hi ha diferents opcions de crear el nostre servidor o servidors, essent la computació al núvol una de les maneres que aporten més flexibilitat i escalabilitat. Tanmateix, l'ús del núvol sol estar associat amb un cost econòmic generalment mensual, semblant al que trobem en el *hosting* tradicional, però més orientat al consum de la plataforma.

Per a aquest projecte hem optat per una solució de tipus *PaaS* (utilitzada per desplegar el nostre codi), en la qual la lògica que requeixin els nostres videojocs s'executarà de manera remota als servidors del proveïdor, però disposant de certes capacitats d'administració.

Platform as a Service - PaaS

Plataforma al núvol en què podem desplegar el nostre codi sempre que estigui suportat pel proveïdor.

El 2014 es va anunciar la primera versió del servei de *backend* per a videojocs **PlayFab** per a Unity. El seu objectiu era proporcionar un conjunt de serveis necessaris per construir videojocs en línia, així com les eines necessàries per gestionar els videojocs un cop surten al mercat.

Es tracta d'una plataforma que ha anat evolucionant al llarg dels anys i que gaudeix d'un gran èxit per la quantitat i la qualitat dels serveis que ofereix, així com per la seva senzillesa.

Figura 6. Alguns dels estudis de videojocs més prestigiosos utilitzen PlayFab

El 2018, PlayFab va ser adquirit per Microsoft i es va rebatejar com a **Microsoft Azure PlayFab**. Actualment aquesta plataforma dóna suport a més de 2.500 jocs i compta amb més d'un bilió de comptes d'usuari creats. Alguns dels jocs que l'utilitzen són títols tan coneguts com *Rainbow Six Siege, Sea of Thieves*, *Minecraft*, *Angry Birds Seasons*, *Forza Horizon 4* o *Crackdown 3*.

Figura 7. Microsoft Azure PlayFab

Microsoft Azure PlayFab

Microsoft Azure PlayFab proporciona serveis orientats al desenvolupament de videojocs, com ara plataformes de distribució de jocs, taulers de comandament per entendre com funciona el joc, serveis per a jocs multijugador, monetització, anuncis i comunitats.

Monthly Active Users (MAU)

Mètrica que utilitza PlayFab, equivalent al nombre total de jugadors únics que generen activitat en el nostre joc durant un mes. Si la mateixa persona juga al nostre joc en dos dispositius diferents amb un compte enllaçat (per exemple, PC i mòbil), compta com una única MAU.

Els tipus de llicència de PlayFab inclouen diversos plans, amb un pla essencial gratuït, però amb limitacions, entre elles algunes característiques com la segmentació de jugadors, anàlisis avançades, tasques planificades, suport dedicat o acords de nivell de servei específics. Tanmateix, aquest tipus de llicència és ideal per a desenvolupadors *amateurs* i *indie* que estan començant i volen provar els serveis que ofereix PlayFab. La resta dels plans són de pagament i varien segons les necessitats i el nombre de MAU que utilitzen la infraestructura.

Les característiques més destacades de PlayFab es divideixen en tres àrees, que són **serveis orientats al joc**, **anàlisi en temps real** i **LiveOps**.

3.1.1. Serveis orientats al joc

Els serveis orientats al joc es posen a disposició del desenvolupador perquè pugui utilitzar-los d'una manera senzilla i àgil. D'aquesta manera, el desenvolupador es pot dedicar al desenvolupament del joc mateix i delega la implementació de funcionalitats que la plataforma ja facilita d'una manera eficient, eficaç i escalable.

Els serveis inclosos en aquesta categoria són:

- **Servidors multijugador**. Permeten construir, desplegar i escalar ràpidament el nostre joc en servidors multijugador confiables. Permet escalar dinàmicament des de 100 fins a 10.000.000 de jugadors o més, i gestionar automàticament les puntes de demanda sense repercutir en l'experiència de joc. Això s'aconsegueix mitjançant la utilització de la infraestructura facilitada per Microsoft Azure, que PlayFab utilitza de fons.

- ***Responsive matchmaking***. Permet ordenar, filtrar i ajuntar usuaris per obtenir la millor compatibilitat i diversió. Permet emparellar jugadors amb

habilitats similars mitjançant l'algoritme de *matchmaking* d'Xbox Live per augmentar l'experiència de joc de l'usuari.

- **Autenticació del jugador i comptes enllaçats**. PlayFab ofereix múltiples opcions d'autenticació, de manera que els usuaris poden utilitzar múltiples dispositius. Alguns dels mecanismes suportats són: *DeviceID* (en el cas d'iOS o Android), *username / password,* comptes de Google, GameCenter, Facebook, Steam, Twitch, etc.

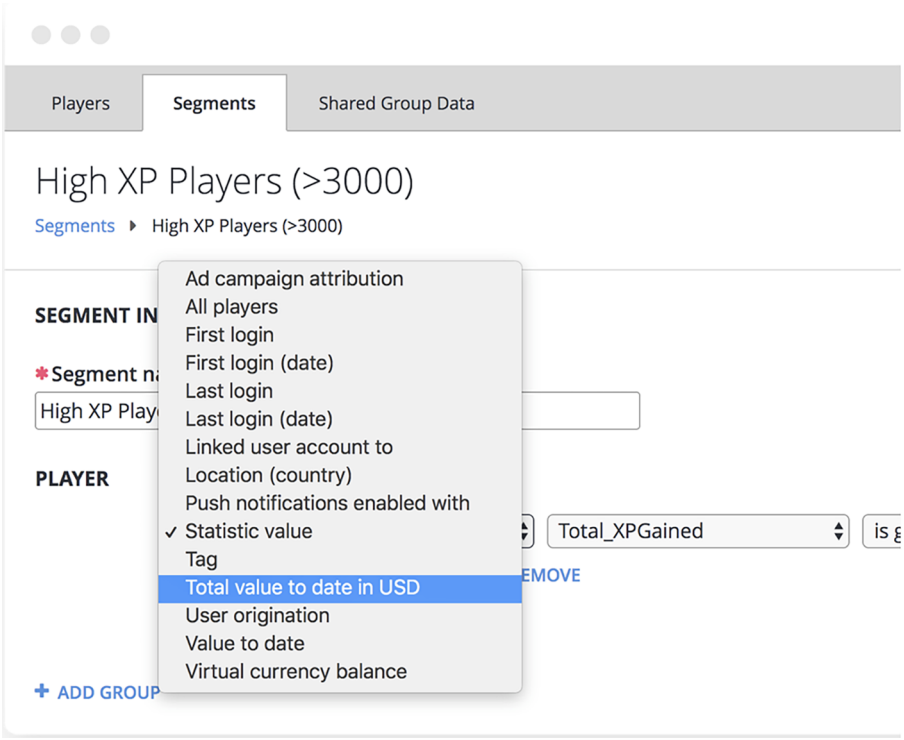
- **Emmagatzematge de dades del jugador**. Permet emmagatzemar informació associada amb el jugador i l'estat del joc al núvol tot evitant la pèrdua de dades. Les dades es poden compartir entre múltiples jocs i dispositius, altres jugadors, estadístiques, etc.

- **Segmentació de jugadors**. PlayFab permet executar accions en temps real cada vegada que un jugador entra o surt d'un segment.

Segment

Un segment permet definir grups de jugadors i realitzar accions exclusives en aquest grup. Per exemple, ens pot interessar agrupar tots els jugadors que sobrepassin un llindar de força definit en el joc per executar una acció concreta.

Figura 8. Un exemple de segmentació de jugadors



- **Classificacions**: es poden crear classificacions de qualsevol estadística del jugador, filtrar-les només per amics, mostrar la posició actual del jugador o qualsevol posició, etc.

- **Caràcters *in-game***: es poden emmagatzemar múltiples caràcters per jugador. Cada caràcter que es configura tindrà les seves pròpies dades, monedes virtuals i el seu inventari.

- **Enviament de notificacions *push***: PlayFab permet enviar notificacions *push* manualment o automàticament a jugadors individuals o a un segment concret.

- **Eines de comerç electrònic**: eines com la creació d'un catàleg dinàmic d'ítems, agrupacions, protecció antifrau, enviament de regals o compres entre jugadors, processament de pagaments mitjançant tercers (Apple Pay, Facebook o PayPal), generació de cupons per canviar-los per ítems, creació de múltiples monedes virtuals, etc.

3.1.2. Anàlisi en temps real

Aquest servei facilita una sèrie de taulers de comandament per poder controlar com evoluciona el nostre joc un cop l'hem iniciat. Aquesta informació és crucial per entendre el comportament dels jugadors en el nostre joc, fet que permet millorar-lo i adaptar-lo en versions posteriors. Els serveis inclosos són:

- **Monitorització de taulers de comandament en temps real**. Les dades s'actualitzen cada 2 minuts.

- **Cerca d'esdeveniments**. Filtratge a través de l'historial recent d'esdeveniments i cerca de jugadors específics, tipus d'esdeveniments o condicions d'errors.

- **Exportació d'esdeveniments**. Enviament d'esdeveniments en temps real a un *bucket* allotjat a Amazon S3, *webhook,* o qualsevol altre sistema d'anàlisi externa.

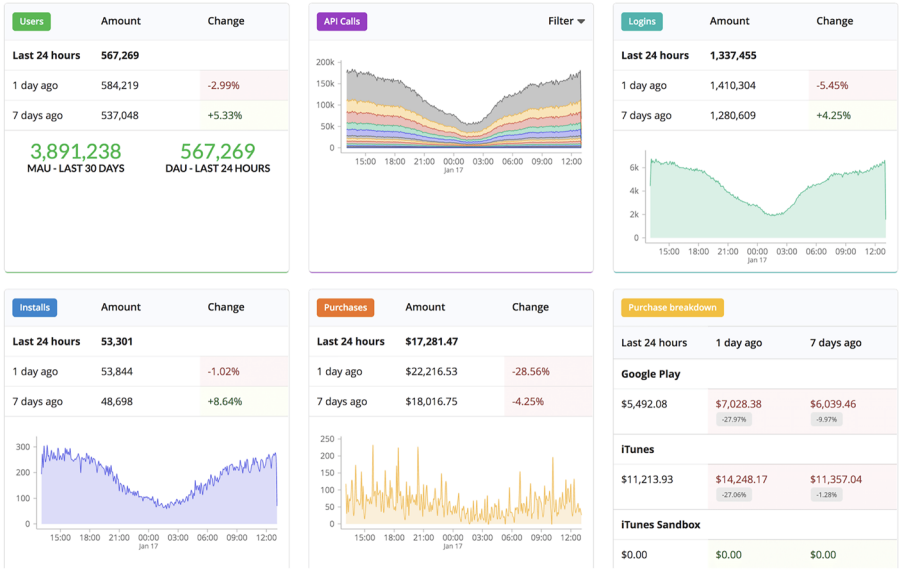
- **Informes de clients**. Dissenyats per facilitar un accés senzill als indicadors clau (*key performance indicators*, KPI) del nostre joc.

- **Depuració en temps real gràcies al flux d'esdeveniments**. Permet observar en temps real el flux d'esdeveniments des dels clients del joc, serveis de *backend* i extensions de tercers.

Integració amb Amazon S3

PlayFab permet multitud d'integracions amb proveïdors externs. Per exemple, pot exportar els esdeveniments del nostre joc a un *bucket* allotjat a Amazon S3. Aquesta integració es descriu aquí: https://api.playfab.com/docs/tutorials/landing-analytics/s3-archive

Figura 9. Exemple dels taulers de comandament en temps real de PlayFab



3.1.3. *LiveOps*

El concepte de *LiveOps* permet allotjar esdeveniments, executar experiments i recompensar els jugadors. Els serveis inclosos són els següents:

- **Gestió remota de la configuració del joc**. És un servei que permet emmagatzemar la configuració del joc al servidor per poder-ne modificar el comportament sense actualitzar els clients.

- **Emmagatzematge i distribució de fitxers del joc**. Permet actualitzacions remotes del nostre joc amb paquets de nous *assets*, DLC (*downloadable contents*) o altres fitxers del joc. L'accés dels clients del joc es fa globalment mitjançant CDN.

CDN (*Content Distribution Network*)

És una xarxa distribuïda geogràficament de servidors *Proxy* i els seus centres de dades. El seu objectiu és proporcionar una alta disponibilitat i un alt rendiment amb una distribució del servei que té en compte la localització dels usuaris finals.

- **Disparar accions basades segons esdeveniments en temps real**: es pot recompensar els jugadors i permetre'ls un conjunt més ric d'accions amb l'enviament de notificacions *push*, per exemple.

- **Tasques programades**: es tracta d'un servei per programar tasques una sola vegada o repetidament.

- **Publicació de titulars**: es poden publicar i editar missatges que veuran tots els jugadors.

- **Plantilles de correu electrònic**: es facilita la creació de plantilles de correu electrònic per a la nostra base de jugadors del joc. Això permet dirigir-nos als nostres jugadors per respondre les seves peticions, enviar-los enllaços que puguin intercanviar per un regal, etc.

3.1.4. Instal·lació del PlayFab SDK

Començarem a treballar amb PlayFab. Per fer-ho, crearem un projecte Unity buit i descarregarem l'Unity3D SDK de PlayFab. Baixarem l'*asset* **PlayFab Unity Editor Extensions *package***.

Descarregar Unity3D SDK

https://docs.microsoft.com/en-us/gaming/playfab/sdks/unity3d/quickstart

Figura 10. Descàrrega del PlayFab Unity Editor Extensions *package*

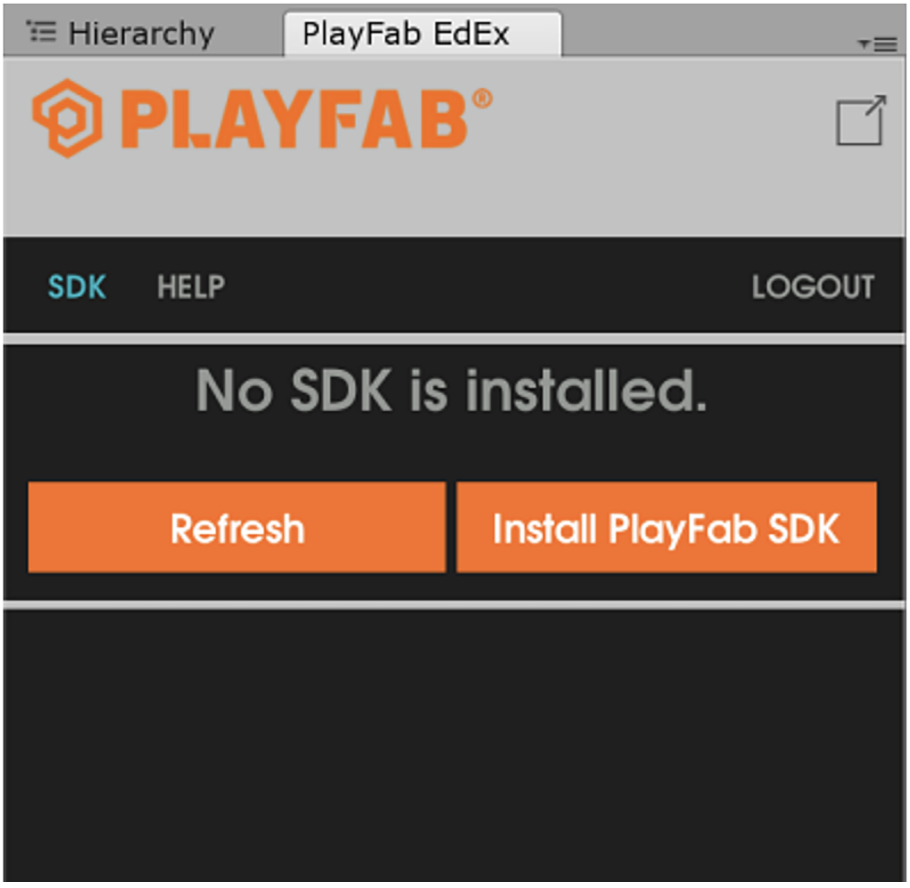
Download PlayFab SDK

The best way to acquire our Unity SDK is via our editor extensions - although you can directly download the Unity 3D SDK from our github page. PlayFab Unity3D SDK.

1. Download and Import the PlayFab Unity Editor Extensions package.
2. To import the the Unity Editor Extensions package, navigate to where the file was downloaded, and double-click on the .UnityPackage file. This will bring up the following window.

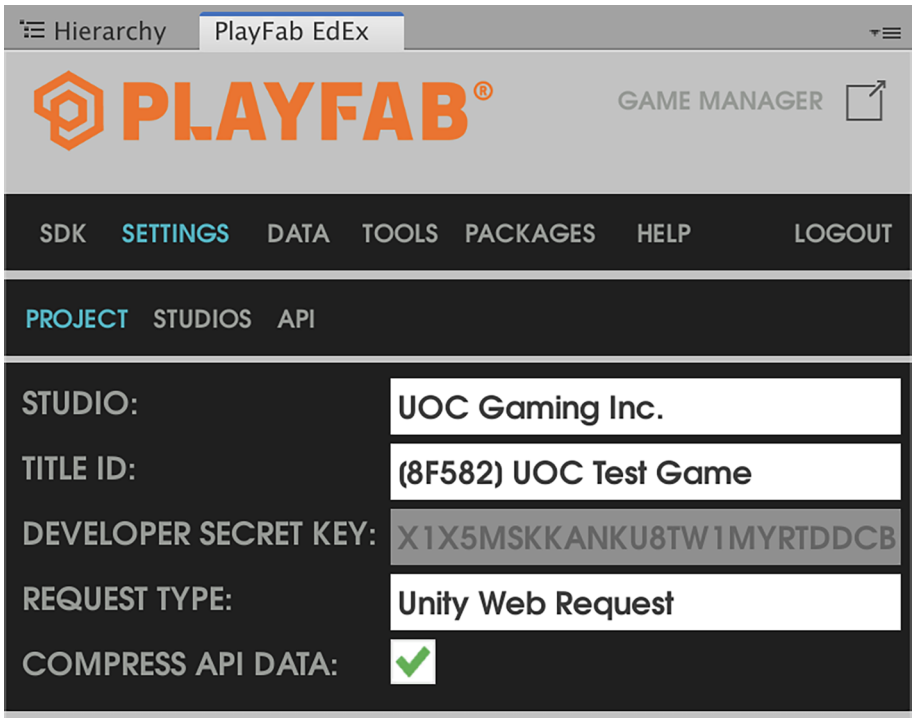
A continuació, importarem l'*asset* al nostre projecte Unity. Ha d'aparèixer una nova finestra *PlayFab EdEx*, des d'on crearem el nostre compte gratuït de PlayFab i tot seguit podrem instal·lar el *PlayFab SDK*:

Figura 11. Finestra Playfab EdEx



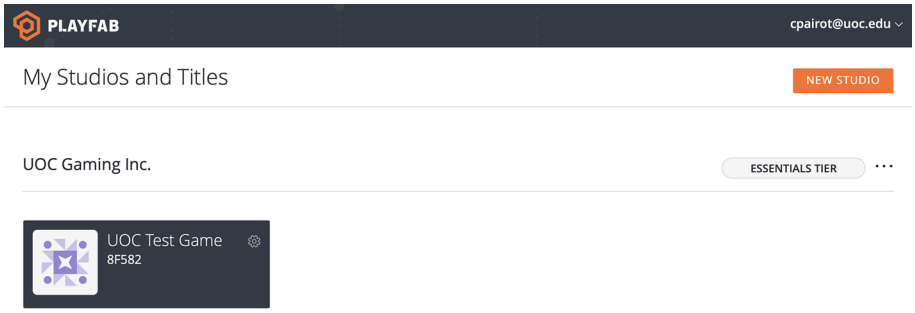
Si tot ha anat bé, anirem a *Settings* i establirem les dades del nostre projecte.

Figura 12. Settings



En aquest punt, pot succeir que en escollir l'Studio ens deixi sempre seleccionat el valor erroni *_**override**_*. Per solucionar-ho, anirem a https://developer.playfab.com/en-US/my-games i crearem un nou Studio. Un cop creat, en tornar a Unity ens permetrà escollir-lo i ja ens apareixerà *Developer Secret Key* amb els altres valors.

Figura 13. Llistat d'estudis i jocs al nostre compte de PlayFab



Després de tots aquests passos ja estem preparats per utilitzar PlayFab en el nostre projecte Unity.

3.1.5. Ús del servei de *login* de jugadors

PlayFab exposa tots els seus serveis mitjançant una API web REST. Per una qüestió de simplicitat i seguretat, totes les crides a les seves funcions són a través de peticions HTTP POST a través de SSL i totes les dades s'envien en format JSON, amb compressió opcional *gzip*.

SSL (*secure sockets layer*)

Protocol que ofereix comunicacions segures a través d'Internet mitjançant l'encriptació de les dades. Tots els llocs web que comencin amb el prefix **https://** l'utilitzen.

En aquest primer exemple utilitzarem el servei de *login* de jugadors de PlayFab. Aquest servei permet als usuaris del nostre joc registrar-se o entrar al seu compte de PlayFab mitjançant una adreça de correu electrònic i una contrasenya.

API de *login* de PlayFab

Podeu trobar exemples de l'API de *login* de PlayFab aquí: https://api.playfab.com/docs/tutorials/ landing-players/best-login

En primer lloc, crearem un nou *script* que ens permetrà gestionar tot aquest procés. Inclourem els *namespaces* de PlayFab a l'inici:

```
using PlayFab;
using PlayFab.ClientModels;
using UnityEngine;

public class PlayFabLogin : MonoBehaviour {
```

En el mètode *Start()* cal definir el camp **TitleId**. Aquest camp és el que podeu veure a la pantalla de *PlayFab EdEx*.

Figura 14. Camp Title Id

Així doncs, el fixem en el codi només per evitar que ens falli si aquest camp encara no s'hagués inicialitzat:

```
    public void Start() {
        if (string.IsNullOrEmpty (PlayFabSettings.staticSettings.TitleId)) {
```

```
            PlayFabSettings.staticSettings.TitleId = "8F582";
        }
```

Tot seguit, ja podem fer crides a mètodes de PlayFab. En aquest primer exemple cridarem al mètode *LoginWithCustomId()*, que es traduirà com una petició HTTP POST als servidors de PlayFab i que rep com a paràmetres d'entrada les dades que contindrà la petició en forma d'objecte *LoginWithCustomIDRequest*. És important destacar que, a més, li passem dues funcions que haurem d'implementar i que tenen la forma de *callback*, és a dir, que es cridarà a l'una o l'altra segons si el *login* s'ha realitzat correctament *OnLoginSuccess*) o ha fallat (*OnLoginFailure*).

```
    var request = new LoginWithCustomIDRequest { CustomId =
            "GettingStartedGuide", CreateAccount = true };
    PlayFabClientAPI.LoginWithCustomID (request, OnLoginSuccess, OnLoginFailure);
}
```

Així doncs, si el *login* ha tingut èxit, la funció s'anomenarà *OnLoginSuccess()*:

```
    private void OnLoginSuccess (LoginResult result) {
        Debug.Log ("Congratulations, you made your first successful API call!");
    }
```

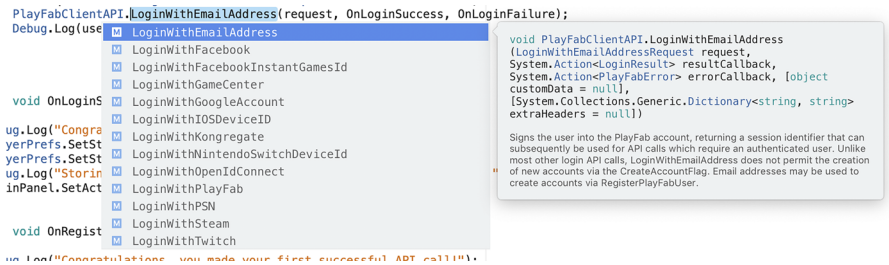
En canvi, si aquest falla, la funció s'anomenarà *OnLoginFailure()*:

```
private void OnLoginFailure (PlayFabError error) {
    Debug.LogWarning ("Something went wrong with your first API call.");
    Debug.LogError ("Here's some debug information:");
    Debug.LogError (error.GenerateErrorReport());
}
```

Repte 1

Proveu el funcionament d'aquest *script* en el vostre projecte d'Unity.

Figura 15. PlayFab permet fer *login* en el seu sistema mitjançant diferents proveïdors i serveis de tercers



En qualsevol cas, el sistema de *login* que hem utilitzat en aquest exemple és només amb propòsits demostratius. Ara veurem com fer un *login* més tradicional amb una adreça de correu electrònic i una contrasenya. Per fer-ho, canviarem la crida per *LoginWithEmailAddress()* en el mètode *Start()*:

```
var request = new LoginWithEmailAddressRequest { Email = userEmail,
                                                 Password = userPassword };


PlayFabClientAPI.LoginWithEmailAddress (request, OnLoginSuccess, OnLoginFailure);
```

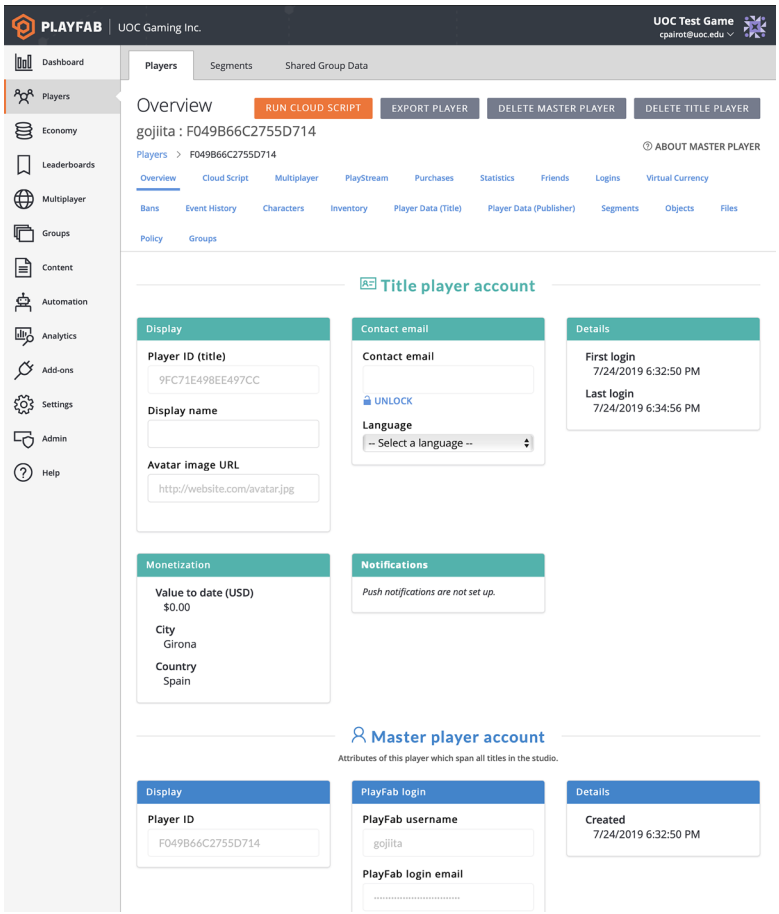
Si us hi fixeu, l'objecte que encapsula les dades que s'enviaran a la petició (*LoginWithEmailAddressRequest*) necessita dues variables de tipus *string,* que contindran l'adreça de correu de l'usuari i la seva contrasenya.

Repte 2

Si el *login* amb l'adreça de correu i la contrasenya falla, feu que l'usuari es registri automàticament en el sistema.

Quan els usuaris ja s'han donat d'alta en el vostre joc, podeu consultar a l'aplicació web de PlayFab les seves dades.

Figura 16. Overview de PlayFab



PlayFab proporciona una classe anomenada **PlayerPrefs** que permet emmagatzemar i consultar dades del jugador mitjançant els mètodes *SetString()* i *HasKey()*:

```
PlayerPrefs.SetString ("EMAIL", userEmail);

if (PlayerPrefs.HasKey ("EMAIL")) {
        Debug.Log ("Player's e-mail address is: " + userEmail);
}
```

En aquest exemple, la clau **EMAIL** estarà enllaçada amb aquest jugador per sempre amb el valor que establim, i ens permetrà així recuperar el seu valor en qualsevol moment des del núvol.

Si ens interessa, podem eliminar una clau concreta de les preferències del jugador o bé totes les que tingui associates amb els mètodes *DeleteKey()* i *DeleteAll()*:

```
PlayerPrefs.DeleteKey ("EMAIL");  // Deletes the EMAIL key
PlayerPrefs.DeleteAll();          // Deletes all keys
```

Repte 3

Aconseguiu que, quan l'usuari s'hagi autenticat a PlayFab per primera vegada, el vostre joc ja no us demani que us autentiqueu més en el sistema.

Repte 4

Creeu un *canvas* en l'escena que us demani el codi d'usuari, adreça de correu electrònic, contrasenya i que, en fer clic en un botó, passi totes aquestes dades al vostre *script PlayFabLogin*, i que aquest autentiqui o registri el vostre jugador.

3.1.6. Ús del servei d'estadístiques de jugadors

Veurem ara com utilitzar un altre dels serveis de PlayFab. En aquest cas, el servei d'estadístiques de jugadors. Per fer-ho, seguirem amb el codi de l'exemple anterior i començarem canviant el nom del nostre *script* de ***PlayFabLogin.cs*** a ***PlayFabController.cs***. Això ho fem perquè ara afegirem noves funcionalitats que ja no estan directament relacionades amb el *login* del jugador i és preferible que l'*script* tingui un nom més genèric.

API d'estadístiques de PlayFab

Podeu trobar exemples de l'API d'estadístiques de PlayFab aquí: https://api.playfab.com/docs/tutorials/ landing-players/player-statistics

Arribats aquí, primer crearem les noves variables a partir de les quals ens interessarà disposar d'estadístiques. Crearem, per exemple, variables per conèixer el nivell del jugador, el nivell del joc, la salut del jugador, el dany que ha infligit el jugador, així com la puntuació màxima que ha aconseguit el jugador:

```
public int playerLevel;
public int gameLevel;
public int playerHealth;
public int playerDamage;
public int playerHighScore;
```

A més, per gestionar adequadament aquestes estadístiques, ens interessa que només hi hagi una única instància de ***PlayFabController*** en execució al mateix temps. Per això, farem servir el patró de disseny *singleton*.

```
public static PlayFabController PFC;


private void OnEnable() {
    if (PlayFabController.PFC == null) {
        PlayFabController.PFC = this;
    }
    else {
```

```
            if (PlayFabController.PFC != this) {
                Destroy (this.gameObject);
            }
        }
        DontDestroyOnLoad (this.gameObject);
    }
```

Com veiem, en el mètode ***OnEnable()***, que es crida quan el Gameobject en qüestió s'activa, comprovarem si ja tenim una instància de *PlayFabController* activa o no. Si no la tenim, fixem el valor de la variable PFC a aquesta instància. En cas contrari, si la instància és diferent de la nostra, la destruïm perquè ja n'existeix una altra. Finalment, indiquem, mitjançant el mètode *DontDestroyOnLoad()*, que no es destrueixi el GameObject actual en carregar una escena nova. Amb aquest comportament aconseguim que només hi hagi una instància activa al mateix temps de l'*script PlayFabController*, seguint correctament el patró de *singleton*.

Patró de disseny *singleton*

En enginyeria del software, el *singleton* és un patró de disseny que restringeix la instanciació d'una classe a una sola instància. És útil quan es necessita un únic objecte que coordini les accions dins d'un sistema.

A continuació, tot seguint l'API de PlayFab, crearem un mètode *SetStats()* que s'encarregarà d'enviar les estadístiques del nostre jugador al servei de PlayFab corresponent:

```
public void SetStats() {
    PlayFabClientAPI.UpdatePlayerStatistics (new UpdatePlayerStatisticsRequest {
       Statistics = new List<StatisticUpdate> {
         new StatisticUpdate { StatisticName = "PlayerLevel", Value = playerLevel },
         new StatisticUpdate { StatisticName = "GameLevel", Value = gameLevel },
         new StatisticUpdate { StatisticName = "PlayerHealth", Value = playerHealth },
         new StatisticUpdate { StatisticName = "PlayerDamage", Value = playerDamage },
         new StatisticUpdate { StatisticName = "PlayerHighScore", Value = playerHighScore }
       }
    },
    result => { Debug.Log ("User statistics updated"); },
    error => { Debug.LogError (error.GenerateErrorReport()); });
}
```

Com veiem, ***PlayFabClientAPI.UpdatePlayerStatistics()*** és com anomenarem el mètode, tot passant-li una *request* HTTP (*UpdatePlayerStatisticsRequest*) i els *callbacks* d'èxit i error.

Al cos d'*UpdatePlayerStatisticsRequest* li passarem les parelles clau/valor equivalents a les estadístiques que volem emmagatzemar a PlayFab.

Per recuperar les dades d'estadístiques que hem emmagatzemat a PlayFab per a aquest jugador en concret, primer haurem d'implementar un mètode *GetStats()* de la següent manera:

```
      new GetPlayerStatisticsRequest(),
```

```
        OnGetStatistics,
        error => Debug.LogError(error.GenerateErrorReport())
    );
}
```

Aquest mètode simplement crida el mètode ***PlayFabClientAPI.GetPlayerStatistics()*** que, si té èxit, cridarà la nostra funció ***OnGetStatistics()***. Així doncs, haurem d'implementar aquesta funció, que s'encarregarà de carregar els valors de les estadístiques procedents de PlayFab a les nostres variables locals mitjançant una estructura de tipus *switch*:

```
public void OnGetStatistics (GetPlayerStatisticsResult result) {
    Debug.Log ("Received the following Statistics:");
    foreach (var eachStat in result.Statistics) {
        Debug.Log ("Statistic (" + eachStat.StatisticName + "): " + eachStat.Value);
        switch (eachStat.StatisticName) {
            case "PlayerLevel":
                playerLevel = eachStat.Value;
                break;
            case "GameLevel":
                gameLevel = eachStat.Value;
                break;
            case "PlayerHealth":
                playerHealth = eachStat.Value;
                break;
            case "PlayerDamage":
                playerDamage = eachStat.Value;
                break;
            case "PlayerHighScore":
                playerHighScore = eachStat.Value;
                break;
        }
    }
}
```

Ara que tenim tots els mètodes per actualitzar i obtenir les estadístiques, hem de decidir des d'on els cridem:

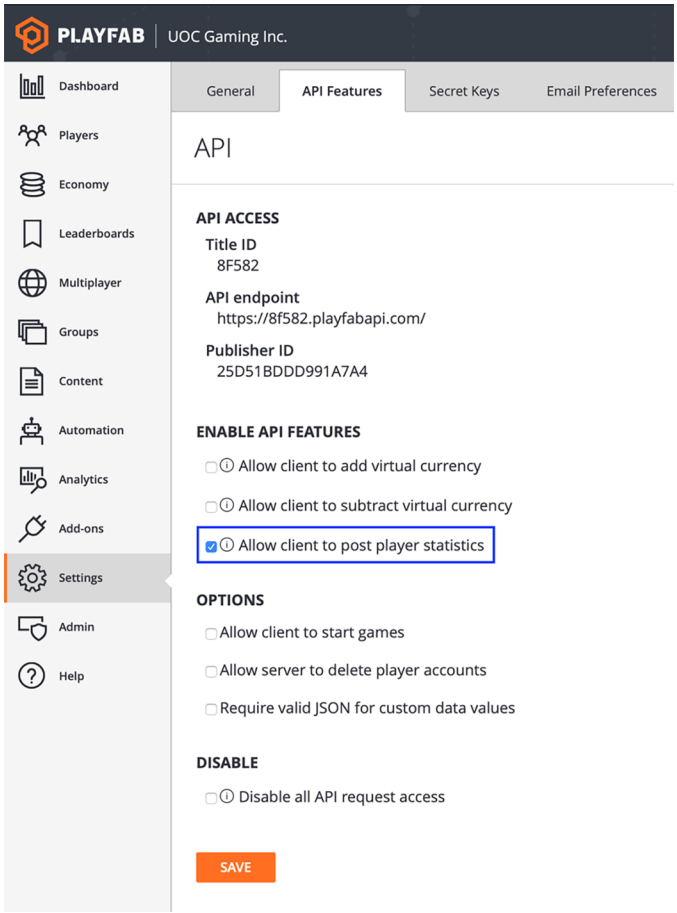
- El millor moment en què es pot cridar el mètode ***GetStats()*** és quan el jugador s'autentica correctament contra el servei de PlayFab, i això és dins del nostre mètode *OnLoginSuccess()*, així que l'actualitzem en consonància.

- Podrem cridar el mètode ***SetStats()*** sempre que necessitem enviar les estadístiques dels jugadors al núvol. Es pot fer mitjançant una tasca programada periòdica i es pot cridar des de qualsevol *script* del joc, tot accedint al

singleton amb ***PlayFabController.PFC.SetStats()***. En el nostre cas, podem cridar-les mitjançant un botó de la interfície.

Tanmateix, abans de poder utilitzar el servei hem de saber que PlayFab, per defecte, ens permet obtenir les estadístiques, però que per evitar possibles actualitzacions fraudulentes no permet actualitzar-les des dels clients. És per això que hem de permetre les actualitzacions de les estadístiques per part dels clients dirigint-nos al PlayFab *dashboard* del nostre joc, al menú **Settings** > **API Features** i marcar l'opció ***Allow client to post player statistics***.

Figura 17. Allow client to post player statistics



Amb això ja podrem provar el nostre projecte i, si ens dirigim a les estadístiques del jugador des del PlayFab *dashboard*, podrem veure com efectivament aquestes i qualsevol actualització s'han desat (a través del client d'Unity o des del *dashboard* directament es propaguen de manera adient).

Figura 18. Podem veure com s'actualitzen les estadístiques des d'Unity Editor

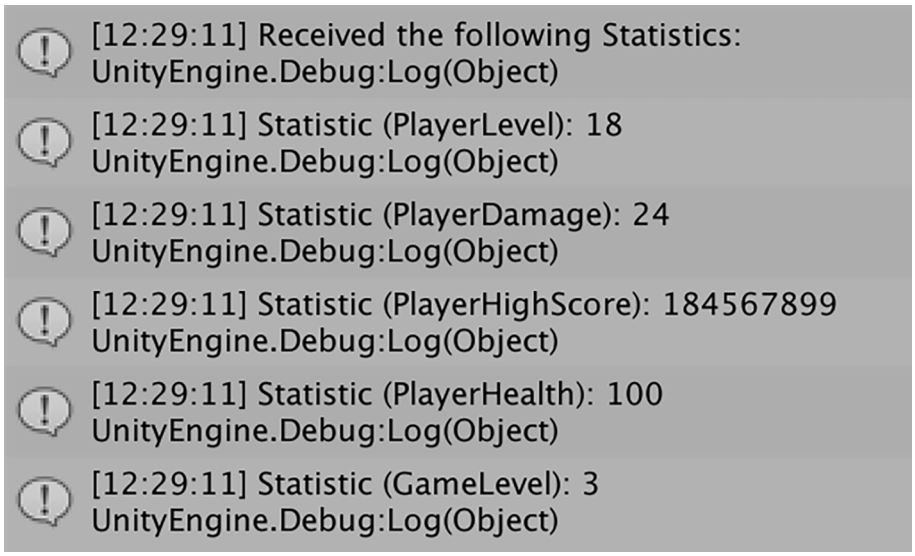
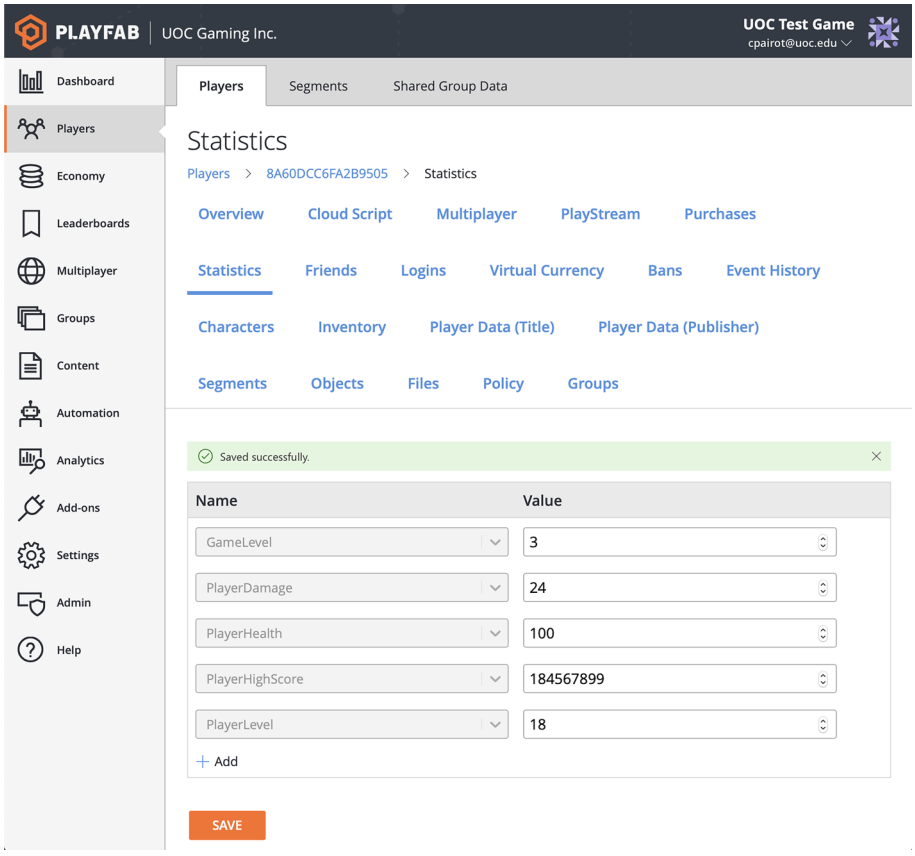


Figura 19. Estadístiques d'un jugador a PlayFab



Repte 5

Com ho faríeu per evitar actualitzar les estadístiques des del client i evitar així que clients fraudulents poguessin actualitzar il·lícitament les estadístiques d'un jugador?

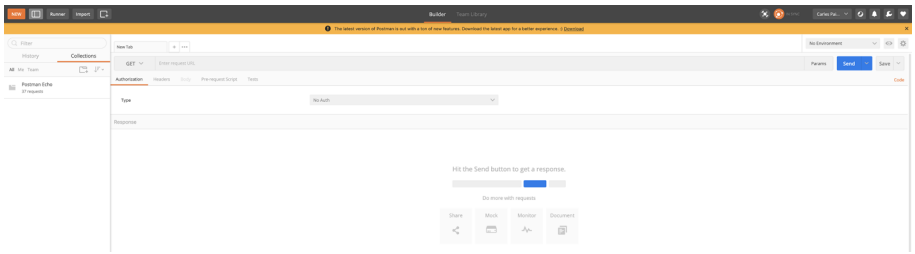
3.1.7. Crida a l'API de PlayFab mitjançant *Postman*

Saber com cridar manualment els mètodes de l'API web de PlayFab és de gran importància quan estem integrant el servei amb el nostre joc. Per exemple, a vegades ens pot interessar saber si un mètode funciona tal com s'esperava, o bé ens interessa cridar un mètode i analitzar les dades específiques de la resposta, o bé volem veure l'error que apareix en invocar un mètode. Hi ha moltes eines que faciliten la interacció directa amb API web, però una de les més populars és el *plugin* **Postman** per a Chrome.

Per a això, haurem de seguir aquests passos:

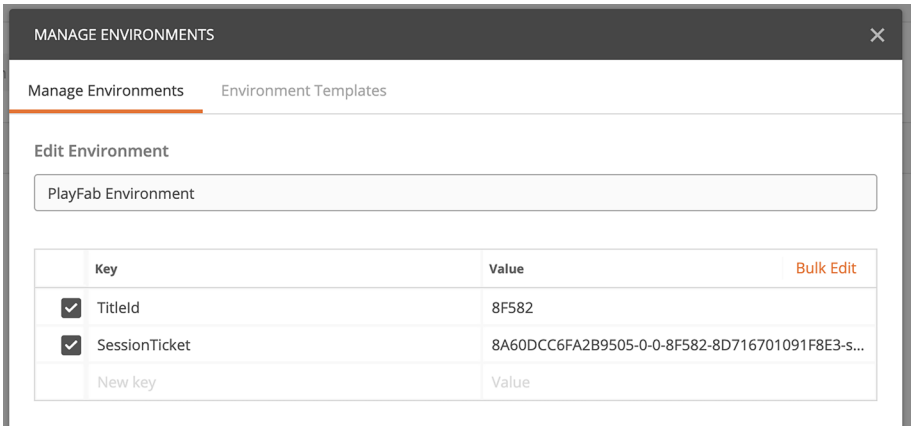
- Obtenir el navegador **Google Chrome**.
- Instal·lar el connector **Postman**. Un cop instal·lat, veurem una pantalla similar a aquesta:

Figura 20. *Plugin* Postman



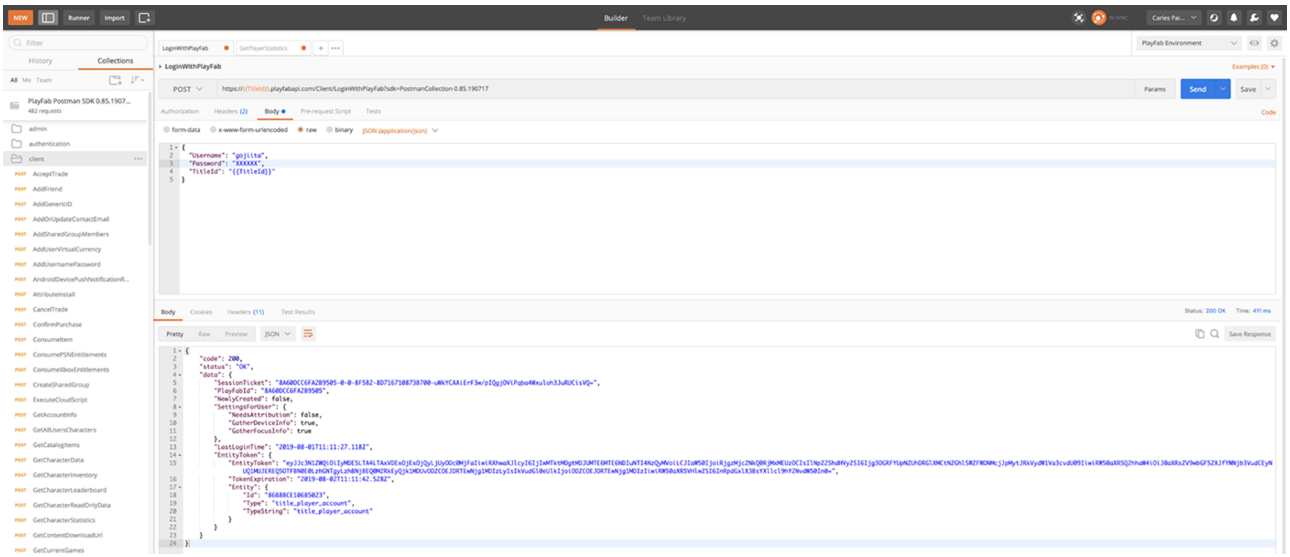
- Descarregar la JSON Collection de PlayFab des de https://api.playfab.com/downloads/postman i importar-la a Postman.
- Crear un nou entorn anomenat PlayFab Environment i afegir-hi les variables que s'utilitzaran en cada crida que fem a l'API. En aquest primer cas, necessitem com a mínim el ***TitleId***. Aquests paràmetres els podem consultar del projecte Unity que tenim o des del PlayFab *dashboard*.

Figura 21. Manage Environments



- Per comprovar que funciona correctament, podem obrir el mètode **Client > LoginWithPlayFab** i a la pestanya Body introduir-hi les dades d'un usuari (codi d'usuari i contrasenya) que estigui registrat al nostre joc. Si tot va bé, rebrem una resposta 200 OK d'aquest estil:

Figura 22. Respuesta 200 OK



- Si volem provar altres mètodes de l'API, segurament calguin altres paràmetres que haurem d'informar, com per exemple el *SessionTicket*, que té una durada màxima de 24 hores, de manera que després d'aquest període de temps, haurem de tornar a autenticar-nos i actualitzar el valor d'aquesta variable.

PlayFab i *Postman*

Teniu més informació sobre com cridar els mètodes de l'API de PlayFab des de Postman en aquest enllaç: https://api.playfab.com/docs/tutorials/ execute-playfab-api-via-postman.

Repte 6

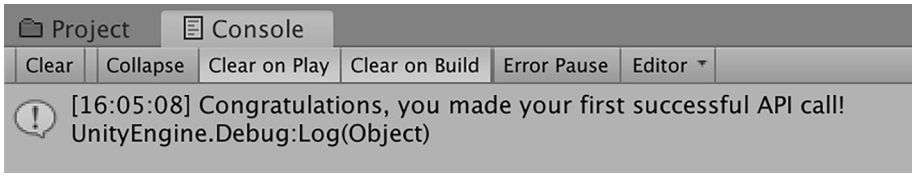
Crideu el mètode de l'API *GetPlayerStatistics()* amb Postman i obteniu les estadístiques del jugador que hem gravat anteriorment.

3.2. Solucions als reptes plantejats

Repte 1

En executar el projecte Unity, fem la primera crida correcta al servei d'autenticació de jugadors de PlayFab. Si observem la sortida a la consola, el resultat ha de ser el que s'esperava:

Figura 23. Resultat a la consola



Repte 2

Per completar aquest repte, hem de modificar el comportament del mètode ***OnLoginFailure()*** de manera que, quan s'executi, intenti crear el nou usuari.

Això s'aconsegueix cridant el mètode ***PlayFabClientAPI.RegisterPlayFabUser()*** passant-li com a paràmetres l'adreça de correu, la contrasenya i el codi de l'usuari.

```
private void OnLoginFailure (PlayFabError error) {
    Debug.Log ("User " + userEmail + " does not exist. Registering new player...");
    var registerRequest = new RegisterPlayFabUserRequest { Email = userEmail,
                                    Password = userPassword, Username = username };
    PlayFabClientAPI.RegisterPlayFabUser (registerRequest, OnRegisterSuccess,
                                    OnRegisterFailure);
}


}
```

També hem de registrar els *callbacks* de registre de nou usuari correcte (***OnRegisterSuccess***) i d'error (***OnRegisterFailure***).

```
private void OnRegisterSuccess (RegisterPlayFabUserResult result) {
    Debug.Log ("Congratulations, new user has been registered!");
}

private void OnRegisterFailure(PlayFabError error) {
    Debug.LogError (error.GenerateErrorReport());
}
```

Repte 3

Per implementar la funcionalitat de recordar a l'usuari que s'ha autenticat a PlayFab en el vostre joc, podem utilitzar la classe de ***PlayerPrefs*** d'Unity, que permet emmagatzemar parelles clau/valor que quedin automàticament associades al perfil del jugador actual.

Per això, primer emmagatzemarem les dades de l'usuari (adreça de correu electrònic i contrasenya) en les *PlayerPrefs* de l'usuari just en el moment que es produeixi una autenticació satisfactòria. És a dir, dins del mètode ***OnLoginSuccess()***, afegirem les línies següents:

```
PlayerPrefs.SetString ("EMAIL", userEmail);
PlayerPrefs.SetString ("PASSWORD", userPassword);
Debug.Log ("Storing " + userEmail + " credentials into Player Preferences.");
```

També podem afegir aquestes línies al mètode ***OnRegisterSuccess()***, que hem implementat en el repte 2 i que crea automàticament l'usuari si aquest no estava registrat prèviament. Doncs bé, en aquest escenari, també ens interessa emmagatzemar les credencials de l'usuari perquè no ens les tornin a demanar en iniciar novament el joc.

Finalment, serà al mètode ***Start()*** en el qual implementarem la lògica d' *accés* d'usuari si ja tenim les dades emmagatzemades en les *PlayerPrefs*:

```
if (PlayerPrefs.HasKey("EMAIL")) {
    userEmail = PlayerPrefs.GetString ("EMAIL");
    userPassword = PlayerPrefs.GetString ("PASSWORD");

    var request = new LoginWithEmailAddressRequest { Email = userEmail,
                                                     Password = userPassword };
    PlayFabClientAPI.LoginWithEmailAddress (request, OnLoginSuccess, OnLoginFailure);
    Debug.Log (userEmail + " user logged in automatically.");
}
```

Si analitzem aquest codi, primer comprovem si dins de les *PlayerPrefs* ja hem emmagatzemat un valor per a la clau de **EMAIL**. Si el tenim, llavors és que prèviament ja s'han enregistrat les credencials del jugador i, per tant, les carreguem a les variables *userEmail* i *userPassword* per procedir, tot seguit, a realitzar automàticament el *login* del jugador a PlayFab.

Com a nota addicional, si voleu restablir els valors de les vostres *PlayerPrefs*, podeu anomenar el seu mètode ***DeleteAll()***.

Repte 4

Aquest desafiament consisteix a vincular els diferents mètodes de l'*script* PlayFabLogin/PlayFabController amb elements d'interfície gràfica (UI) d'Unity.

Primer, haurem d'afegir mètodes a l'*script PlayFabLogin/PlayFabController* perquè, des de la interfície, es puguin actualitzar els valors de les variables *userName*, *userEmail* i *userPassword*:

```
public void GetUserEmail (string emailIn) {
    userEmail = emailIn;
}

public void GetUserPassword (string passwordIn) {
    userPassword = passwordIn;
}

public void GetUsername (string usernameIn) {
    username = usernameIn;
}
```

També afegirem un mètode al qual es cridarà quan fem clic sobre el botó de Login, i que invocarà l'API de PlayFab per autenticar el jugador:

```
public void OnClickLogin() {
    var request = new LoginWithEmailAddressRequest { Email = userEmail,
                                                     Password = userPassword };
    PlayFabClientAPI.LoginWithEmailAddress (request, OnLoginSuccess, OnLoginFailure);
}
```

Per acabar la part de *scripting*, declararem una variable pública per poder fer referència al panell de *login* que utilitzarà l'*script*:

```
public GameObject loginPanel;
```

Dins de l'Unity Editor, crearem un *canvas* molt senzill i afegirem un panell que anomenarem *LoginPanel*. Dins del *LoginPanel*, afegirem tres *InputFields* (Username, Email, Password) i un *Button* (Login).

Figura 24. Panell canvas

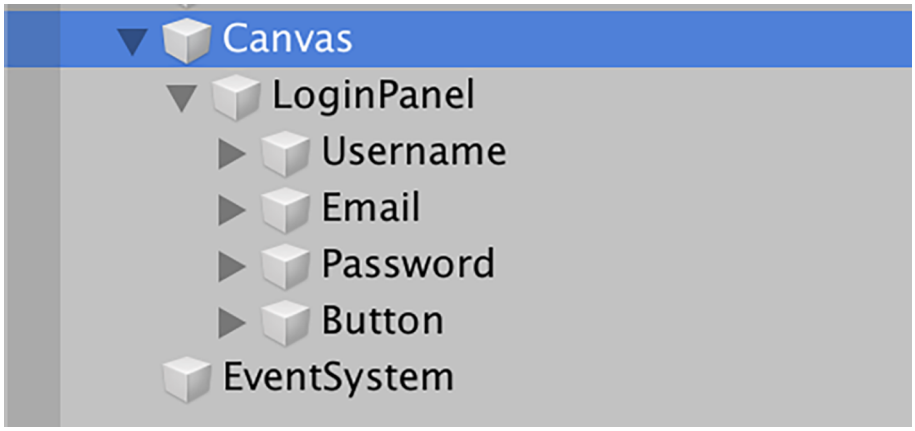
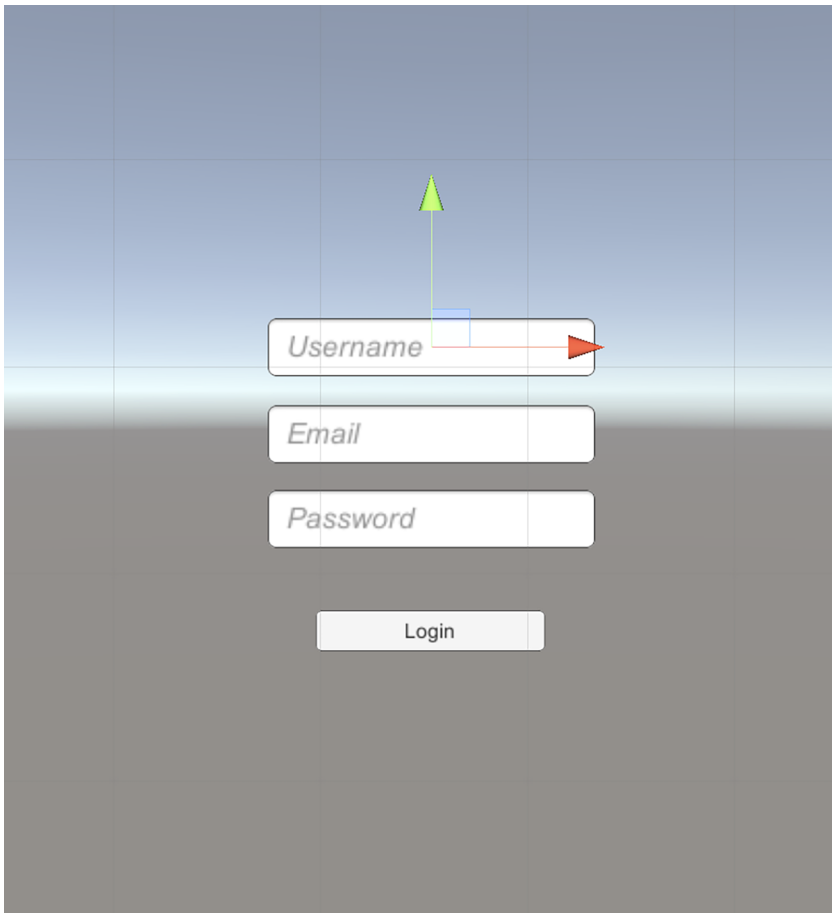


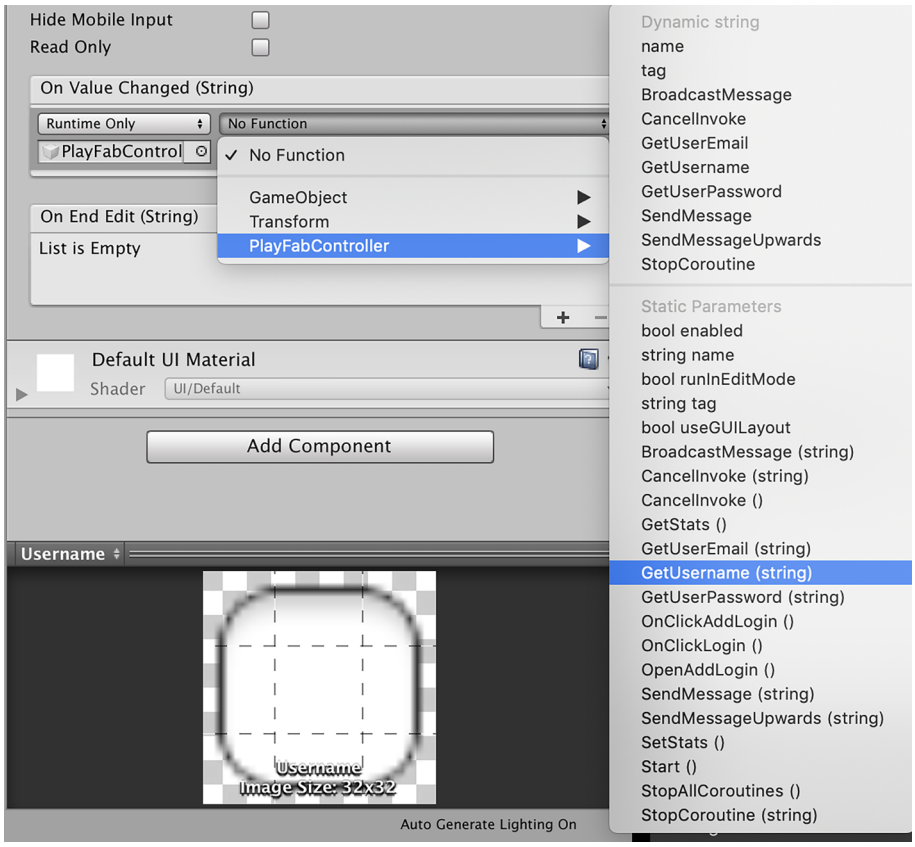
Figura 25. Panell LogIn



A cadascun dels *InputFields*, els vincularem en l'*On Value Changed*, el mètode corresponent del *PlayFabLogin*.

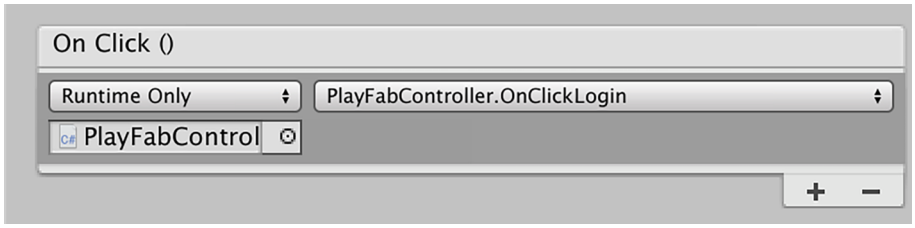
Per exemple, en l'*InputField Username*, cridarem el mètode *GetUsername()* de *PlayFabLogin/PlayFabController*:

Figura 26. InputField Username



El botó s'enllaçarà amb el mètode *OnClickLogin()* de *PlayFabLogin/PlayFabController*:

Figura 27. Configuració del botó



Finalment, arrossegarem el *LoginPanel* a la propietat Login Panel de l'*script*:

Figura 28. Play Fab Controller

Per donar-li un toc més professional, podem fer que un cop l'usuari s'hagi autenticat correctament, el *LoginPanel* desaparegui. Per aconseguir-ho, podem afegir aquesta línia als mètodes *OnLoginSuccess()* i *OnRegisterSuccess()*:

```
loginPanel.SetActive(false);
```

Repte 5

Com hem vist, l'actualització de les estadístiques dels mateixos clients és una pràctica eminentment insegura i des de PlayFab hem hagut de canviar el comportament predeterminat per permetre-ho.

Per aconseguir el mateix, però d'una manera molt més segura, necessitem que els valors de les estadístiques s'actualitzin des del mateix servidor. Però, com ho podem fer? La resposta de PlayFab a aquesta necessitat s'anomena ***cloud scripts***.

Els *cloud scripts* s'executen al núvol de PlayFab i tenen accés complet a l'API del servidor de PlayFab. El més importantment és que aquests *scripts* s'executen en el context d'un jugador autenticat de forma segura, de manera que es poden utilitzar per implementar lògica de joc que estigui a recer d'*exploits* que puguin venir de clients fraudulents.

Les funcions de tipus *cloud script* també poden realitzar crides a *endpoints* HTTP externs, com ara bases de dades o altres API privades, de manera que les fa molt flexibles de cara a la integració amb altres sistemes de *backend* ja existents.

Així doncs, per crear el nostre primer *cloud script*, anirem a **PlayFab Dashboard** > **Automation** > **Cloud Script** > **Revision** i afegirem el següent codi, per exemple, al final del fitxer que se'ns mostra:

```
handlers.UpdatePlayerStats = function (args, context) {
    var request = {
        PlayFabId: currentPlayerId, Statistics: [{
                StatisticName: "PlayerLevel",
                Value: args.Level
            },
                                                {
                StatisticName: "PlayerHighScore",
                Value: args.highScore
            },
                                                {
                StatisticName: "PlayerHealth",
                Value: args.Health
            }]
    };
    // The pre-defined "server" object has functions
    // corresponding to each PlayFab server API
    var playerStatResult = server.UpdatePlayerStatistics(request);
    return {messageValue: "Updated Cloud Stats"}
};
```

Simplement, el que hem fet ha estat crear una nova funció ***UpdatePlayerStats()*** a partir de la funció ja existent *makeAPICall()* per adaptar-la a les nostres necessitats. El llenguatge utilitzat en els *cloud scripts* és **JavaScript**, però si ens hi fixem, s'assembla molt al que hem estat fent fins ara en C# dins d'Unity.

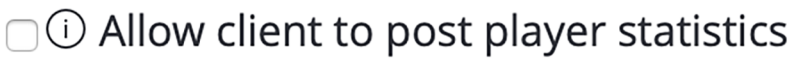
D'una banda, tenim una primera part que és la creació de la *request* en si, en la qual fixem els valors de les estadístiques que volem actualitzar i, d'altra banda, tenim la crida a l'API del servidor de PlayFab, en aquest cas, passant-li aquesta *request*.

És molt important que graveu aquests canvis amb el botó **Save Revision X**, on la X indica un número de revisió. Però no n'hi ha prou d'enregistrar-la, sinó que hem de desplegar (*deploy*) aquesta versió perquè s'executi en l'entorn de producció de PlayFab. Per fer-ho, premem el botó **Deploy Revision X**.

Figura 29. Botó Deploy Revision X

També podem aprofitar per desmarcar l'opció que permet als clients enviar estadístiques, fent així l'entorn molt més segur: **PlayFab Dashboard** > **Settings** > **API Features**.

Figura 30. API Features



Ara ja podem anar a Unity Editor per modificar l'*script* ***PlayFabController*** perquè cridi aquest *cloud script* (***UpdatePlayerStats()***) quan hagi d'actualitzar les estadístiques.

Creació de *cloud scripts* personalitzats

Per obtenir més informació sobre la creació de *cloud scripts* personalitzats, podeu seguir aquest enllaç:

https://api.playfab.com/docs/tutorials/ landing-automation/writing-custom-cloud - script

Per cridar el nostre *cloud script*, ho farem tot implementant el mètode ***StartCloudUpdatePlayerStats()***:

```
public void StartCloudUpdatePlayerStats() {
    PlayFabClientAPI.ExecuteCloudScript (new ExecuteCloudScriptRequest()
    {
        FunctionName = "UpdatePlayerStats",
        FunctionParameter = new { Level = playerLevel, highScore = playerHighScore,
                                  Health = playerHealth }, GeneratePlayStreamEvent = true,
    }, OnCloudUpdateStats, OnErrorShared);
}
```

Com veiem, cridarem la funció ***UpdatePlayerStats()*** passant-li els paràmetres de les estadístiques que cal actualitzar i els *callbacks* a cridar en cas d'èxit (***OnCloudUpdateStats***) i error (***OnErrorShared***).

El codi d'aquests dos *callbacks* és el següent:

```
private static void OnCloudUpdateStats (ExecuteCloudScriptResult result) {
    Debug.Log(PlayFab.PluginManager.GetPlugin<ISerializerPlugin>
         (PluginContract.PlayFab_Serializer).SerializeObject (result.FunctionResult));
     JsonObject jsonResult = (JsonObject) result.FunctionResult;
     object messageValue;
     jsonResult.TryGetValue ("messageValue", out messageValue);
     Debug.Log ((string) messageValue);
}

private static void OnErrorShared (PlayFabError error) {
    Debug.Log (error.GenerateErrorReport());
}
```

El mètode *OnCloudUpdateStats()* simplement deserialitza el valor de retorn (messageValue) de la crida a *UpdatePlayerStats()* i el mostra a la consola.

L'última tasca a fer és canviar l'acció que cridarà el botó d'actualitzar les estadístiques perquè, en comptes de cridar el mètode ***SetStats()*** –que ja no funcionarà perquè hem deshabilitat que els clients puguin actualitzar-les directament– es cridi el mètode ***StartCloudUpdatePlayerStats()***.

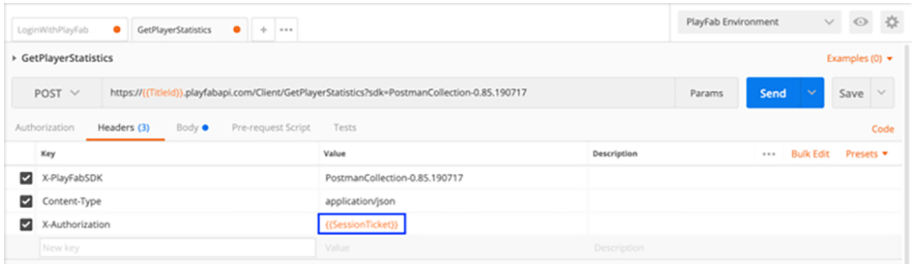
Figura 31. Canvi del mètode en acció *On Click()*

Repte 6

Per poder invocar el mètode ***GetPlayerStatistics()***, primer l'obrirem amb Postman des de **Client** > ***GetPlayerStatistics()***.

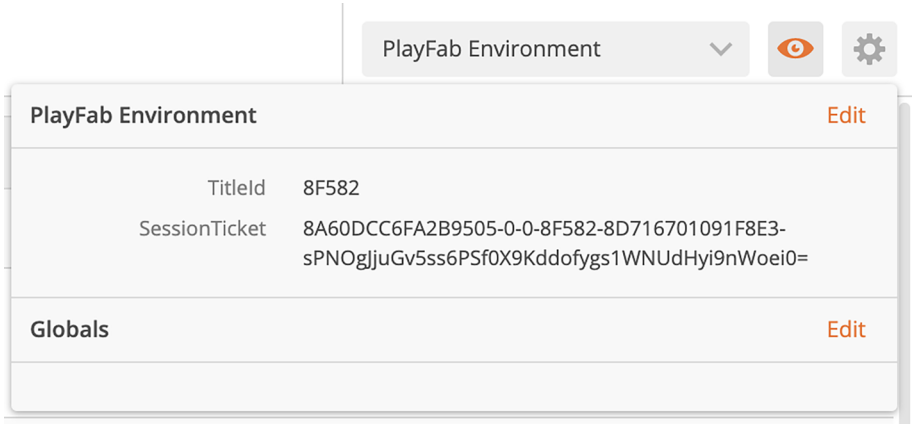
Si ens hi fixem, veurem que a la capçalera se'ns demana que informem el paràmetre *SessionTicket*, que haurem d'haver obtingut abans de l'autenticació de l'usuari cridant a ***LoginWithPlayFab()***:

Figura 32. Crida a LoginWithPlayFab



Establirem, per tant, el valor del *SessionTicket* a la variable d'entorn corresponent:

Figura 33. *SessionTicket* al PlayFab Environment



Si obrim el pestanya Body, veurem que en la *request* se li passa un paràmetre anomenat *StatisticNames*, que ens permet especificar quines estadístiques volem obtenir. Com que les volem aconseguir totes, simplement comentem aquest paràmetre:

Figura 34. Pestanya Body

Finalment, realitzem la invocació i si tot ha anat bé, ens ha de tornar les estadístiques del jugador que havíem establert prèviament:

Figura 35. Estadístiques del jugador

Resum

En aquest mòdul hem pogut repassar el que avui són els serveis indispensables per a qualsevol joc, encara que no es tracti ni d'un joc en línia ni multijugador. Aquests serveis són típics de plataformes molt populars, com l'exemple de Steam que hem estudiat. Però aquests serveis són, òbviament, del costat del servidor de cada plataforma.

Per entendre com funcionen aquests serveis, hem presentat el paradigma de la computació al núvol, que avui és la manera més recomanada de realitzar aquesta mena de tasques remotes. Com hem vist, hi ha diferents tipus de núvols, cadascun amb els seus avantatges i desavantatges. El cost és un dels principals inconvenients, compartit amb les solucions de servidor tradicionals, encara que potser una mica més just, ja que es factura per consum real.

Hi ha diferents alternatives a serveis *cloud* oficials per a Unity, com Microsoft Azure PlayFab. Són serveis de pagament que ofereixen una manera senzilla de desplegar el nostre videojoc. Alternativament, podem buscar altres *assets* que ens permetin integrar el nostre joc amb una plataforma concreta o específica del mateix *asset* i dotar-lo dels servidors dedicats.

Finalment, hem recuperat l'exemple treballat en Unity en els altres mòduls, *Tanks!*, per integrar-lo amb una plataforma *cloud* real com PlayFab. Aquesta plataforma de serveis en línia ofereix moltes funcionalitats que es beneficien de l'escalabilitat del núvol i d'una API de programació senzilla i robusta, basada en tecnologies obertes com HTTP i JSON.

Hem vist com s'utilitzen els serveis al núvol de PlayFab, concretament els serveis de registre i autenticació de jugadors, així com el servei d'estadístiques que per a més seguretat, s'ha de cridar directament des del servidor mitjançant *cloud scripts*.

Bibliografia

Alexandre, Thor (2005). *Massively Multiplayer Game Development 2 (Game Development)*. Rockland, MA: Charles River Media, Inc.

Armitage, Grenville; Claypool, Mark; Branch, Philip (2006). *Networking and Online Games: Understanding and Engineering Multiplayer Internet Games*. Hoboken, NJ: John Wiley & Sons, Ltd.

Coulouris, George; Dollimore, Jean; Kindberg, Tim (2005). «Distributed systems: concepts and design». En: *International computer science series* (vol. 4). Reading, MA: Addison Wesley.

Jimenez-Rodriguez, J.; Jimenez-Diaz, G.; Diaz-Agudo, B. (2011). «Match-making and case-based recommendations». En: *Workshop on Case-Based Reasoning for Computer Games. XIX Conferencia Internacional sobre Case Based Reasoning*.

Glickman, M. E. (1999). «Parameter estimation in large dynamic paired comparison experiments». *Applied Statistics* (vol. 48, pàgs. 377-394).

Herbrich, R.; Minka, T.; Graepel, T. (2007). «TrueSkill(TM): a bayesian skill rating system». *Advances in Neural Information Processing Systems* (vol. 20, pàgs. 569-576).

Tanenbaum, Andrew S.; Van Steen, Maarten (2006). *Distributed Systems: Principles and Paradigms* (2a ed.). Upper Saddle River, NJ: Prentice-Hall, Inc.

Yahyavi. Amir; Kemme, Bettina (2013). «Peer-to-peer architectures for massively multiplayer online games: A Survey». *ACM Comput. Surv*. (vol. 46, núm. 1, art. 9).