UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Desarrollo de un juego online de cartas para 4 personas en Android

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Aleixandre Jesus, Josep Vicent

Tutor: Sánchez Díaz, Juan

Curso 2018/2019

Agradecimientos

Antes que nada, me gustaría agradecer a mis padres que tanto me han apoyado para sacar adelante este proyecto incluso en los momentos más difíciles, y a mis amigos que me han animado a dar lo mejor de mí para lograrlo.

También me gustaría agradecer a mis compañeros de trabajo que mediante sus experiencias realizando sus propios trabajos he podido aprender una gran variedad de cosas sobre este tipo de proyectos y a mi tutor, que me ha ayudado mucho en las fases finales del proyecto.

Por último, me gustaría agradecer a toda esa gente anónima de internet que me ha ofrecido sus conocimientos para que pudiera llevar a cabo esta obra, a todos ellos, gracias.

Resumen

El trabajo a realizar consiste en la implementación de un juego real de cartas para la plataforma Android. La interfaz de juego se implementará con Unity y C#.

La comunicación entre cliente y servidor se realizará utilizando servicios de Unet. El jugador podrá crear un servidor de juego al cual se conectarán el resto de jugadores.

El servidor será el encargado de mantener el estado del juego y el que comunicará a los jugadores cuando es su turno, qué opciones tienen y la situación actual del juego para que el cliente pueda actualizar su información de partida y decidir la acción a realizar. Con esta información el cliente construirá el nuevo estado de la partida en la interfaz gráfica y mostrará los diálogos necesarios para que el usuario interactúe.

Todo el desarrollo se documentará apoyándose en algunos casos de modelos gráficos UML.

Índice

1. Introducción

1.1 Objetivos

El objetivo de este trabajo es el desarrollo de un juego de cartas de Android para cuatro jugadores en línea. La idea de este proyecto comienza por la elección de un juego de cartas ya sea uno existente o uno totalmente original, buscar el entorno de desarrollo que más se adapte a las necesidades del proyecto y traducir las mecánicas del juego escogido a las de un lenguaje de programación.

De esta manera, el juego de cartas se deberá poder jugar por hasta cuatro jugadores, deberá tener al menos un sistema de puntuación, unas normas básicas y además debe contar con un sistema de red que permita la conexión de los diferentes jugadores haciendo uso de las diferentes opciones que permita el entorno de desarrollo.

1.2 Metodología

Inicialmente, se van a realizar las tareas de pre-análisis, en las cuales se llevarán a cabo el análisis del estado del arte, la selección del juego y la selección del entorno, una vez se tenga claro lo que se va a realizar, se pasar a la fase de análisis donde se evaluarán las características que debe tener el proyecto y sus casos de uso.

Una vez realizado el análisis se procederá al diseño de la aplicación y su implementación donde se llevará a cabo el diseño de la interfaz y la arquitectura que seguirá el juego, una vez concluida la fase anterior, se procederá a la realización de las pruebas para que el juego funcione correctamente y para acabar los pensamientos finales del proyecto.

2.Pre-análisis

2.1 Estado del arte

La industria del videojuego actualmente está dominada por títulos AAA que rivalizan con los grandes éxitos de taquilla de Hollywood. En sus inicios, muchos juegos fueron desarrollados por grupos de uno o dos personas, esto fue mucho antes de que los juegos tuvieran gráficos fotorrealistas o que necesitaran equipos de cientos de personas para llevar a cabo su desarrollo.

Los humildes comienzos de la industria involucraron a una gran cantidad de personas que estaban descubriendo las posibilidades de este nuevo medio, avanzando tecnológicamente para optimizar la diversión. Los primeros desarrolladores de juegos crearon un marco para cada juego que se ha repetido durante décadas y, finalmente, se ha convertido en el ecosistema que tenemos hoy en día.

Echando la vista atrás se puede pensar que los juegos desarrollados por equipos reducidos con presupuesto muy reducido han pasado a ser cosa del pasado, pero este no es el caso, el espíritu de estos pioneros vive en forma de juegos independientes y desarrolladores independientes. Y ya no solo se centran en la diversión. Esta nueva clase de pequeños equipos de desarrollo se ha trasladado a todo el espectro de emociones humanas a la vez que mantiene la diversión intacta.

Los juegos *Indie* han logrado mantener vivo y bien el espíritu de innovación y colaboración. Impulsados por la pasión por crear y despreocupado con las debilidades que plagan el desarrollo de juegos AAA. Pequeños grupos de personas están generando experiencias de diferentes que atraen con éxito las emociones humanas. Ahora más que nunca, los juegos *indies* están liderando el impulso del medio para aprovechar la experiencia humana de la misma manera que lo hacen el arte, el cine y la música tradicionales.

Es difícil definir lo que significa un juego *indie* pues muchas veces se puede pensar que es un videojuego que se crea a menudo sin el apoyo financiero de un editor, pero ese ya no es el caso. *Xbox, Playstation, Nintendo, EA, Ubisoft y Activision* tienen sus propios programas implementados para garantizar que haya un ecosistema saludable de títulos *indie* disponibles en sus plataformas. Sin embargo, algunos de los títulos más importantes e innovadores se encuentran en la definición tradicional. Lo único que casi todos los *indies* tienen en común es que dependen de la distribución digital y no de los medios físicos.

Pero lo más importante que se debe saber sobre los juegos *indie* es que, al igual que las películas o música independientes, tradicionalmente son creadas por grupos pequeños. Muchas veces, incluso se repiten sobre la mecánica de juego establecida en formas nuevas y únicas. Aunque algunos realmente se convierten en algo que es especialmente vanguardista.

El lanzamiento de Nintendo Switch ha puesto los títulos *indie* en manos de muchas más personas. Su lanzamiento ha despertado un nuevo interés en los títulos independientes, ya que ahora puedes llevarlos a cualquier parte. Todos los desarrolladores están trasladando sus juegos a la consola porque el ecosistema es muy correcto. Esto ha llevado a un aumento general del interés en los juegos *indie* entre la población mayoritaria de jugadores. Si bien 2018 ha visto mucha innovación en títulos AAA, realmente, el año pasado destacó por sus títulos *indie*.

Los juegos *indie* se han consolidado como otra faceta de la industria del juego. Una faceta que celebra experiencias personales que se centran en una sola idea y agotan esa idea hasta que tienen un juego completo. La falta de grandes presupuestos de marketing y derechos de publicación significa que los juegos independientes no deben descansar completamente sobre sus laureles si quieren tener éxito. Estas condiciones son perfectas para crear grandes juegos, por lo que vemos experiencias muy centradas en la historia, el juego, el estilo de arte y, a veces, incluso la expresión personal.

Los juegos independientes se han convertido en experiencias más personales que las que un juego AAA puede ofrecer. Se sienten menos como si estuvieras jugando un éxito de taquilla y más como si estuvieras leyendo una novela de alguien con un estilo de escritura singular.

El juego *indie* perfecto es una extensión directa de las sensibilidades de su pequeño equipo de creadores o incluso de un solo creador. Entonces, en ese sentido, los *indies* se han convertido en el ejemplo más puro de los videojuegos como una forma de arte. Los desarrolladores independientes entienden la cita anterior más que nadie. Entienden que en alguna parte se encuentra un juego que solo podría venir de ellos. Es por eso que muchos juegos independientes tratan conceptos que son de naturaleza muy humana, aunque lo que realmente tratan podría ser completamente eliminado de la humanidad. El resultado final es algo completamente nuevo, que es lo más cercano a un autorretrato que un juego.

En lo que respecta a los juegos de cartas, existen una gran cantidad y variedad de ellos en Android, muchos de estos juegos son simples recreaciones de juegos ya existentes como por ejemplo *Zynga Poker - Texas Holdem,* el cual se muestra en la Figura 1, los juegos de *poker* y de apuestas son juegos muy populares puesto que existe una gran cantidad de público con ganas de usar su dinero para este tipo de ocio, no por nada los casinos siguen siendo a día de hoy uno de los negocios más rentables.

Figura 1: Imagen del juego *Zynga Poker - Texas Holdem*

Pasando a otros tipo de juegos de cartas que no involucran la puesta de dinero de la Figura 2, tenemos el UNO, un juego clásico que es un juego para diverse con los amigos o la familia, este tipo de juegos siempre ha sido muy recurrente, tiene mecánicas sencillas que no dependen tanto de la aleatoriedad como el *poker*.

Figura 2: imagen perteneciente al juego UNO para Android

Siguiendo el recorrido, no podría faltar uno de los tipos de juego de cartas que no solo tiene una gran cantidad de adeptos al género sino que produce una alta cantidad de ingresos, los juegos de cartas coleccionables, estos juegos no confían en las apuestas para ganar dinero sino que mediante las denominadas expansiones van aumentando la variedad de cartas, estos juegos muy a menudo tienen sus propias reglas de juego que nada tienen que ver con juegos tradicionales como el *poker*, mientras que el *poker* gana dinero durante las partidas, los juegos como Hearthstone (Figura 3) ganan dinero antes de ellas pues los jugadores deben comprar expansiones de cartas donde contienen cartas de diferentes rarezas, que es donde reside el gancho de estos juegos .

Figura 3: Imagen de juego de Hearthstone

Los juegos de cartas de Android cuentan con una variedad para todos los públicos, aunque hay juegos como el *poker* que deberían mantenerse fuera del alcance de los niños, la industria de juegos de smartphone y la de juegos de cartas en particular, es una de las que más ingresos genera dentro de la industria de los videojuegos.

2.2 Propuestas de desarrollo.

2.2.1 Original *vs* Tradicional

Como única pista inicial se tiene que debe ser un juego de cartas, aquí se plantea el primer problema, crear un juego totalmente original desde cero o la utilización de un juego de cartas ya existente e incluso se podría plantear una tercera opción la cual consistiría en utilizar un juego de cartas ya existente como pueda ser un *poker* y cambiar sus reglas de formas que se mantenga la esencia del juego pero sus mecánicas difieran en cierto grado a las reglas tradicionales del juego.

La creación de un juego original consistiría en visualizar un juego de cartas desde cero enfocándonos en que el juego debe de ser para 4 jugadores. Este método supondría diferentes ventajas e inconvenientes. La creación de un juego original le daría un carácter único al juego, permitiendo una mayor libertad a la hora de creación pero sin embargo un juego original supondría un mayor coste temporal y además una serie de ajustes a la hora de equilibrar sus reglas para todos los jugadores en su proceso creativo, cuanto más complejo sea el juego, más difícil resultará de equilibrar y sus pruebas deberán ser más exhaustivas.

La utilización de un juego tradicional como base de creación del juego tendría como principales ventajas una mayor simplicidad a la hora de desarrollarlo ya que sus reglas ya vienen fijadas y su fase de implementación y pruebas sería más sencilla por norma general que un juego original, además, un juego de cartas ya existente de cara al público sería mucho más fácil de promocionar que una idea nueva. Como desventaja se podría considerar que un juego tradicional requiere de una mayor estrategia de penetración de mercado y una menor libertad creativa en sus fases de implementación ya que debe ceñirse a las reglas.

La idea de basarse en un juego de cartas ya existente y modificar sus reglas para darle cierto grado de originalidad también es una gran opción que ayuda al empezar su desarrollo, pero después puede dar lugar a cambios insatisfactorios o que reducen su jugabilidad.

En conclusión, se ha llegado a la idea de que este proyecto se va a basar en un juego de cartas ya existente para así agilizar su fase de implementación y pruebas, por la cual, deberemos pasar primero un proceso de selección de los juegos que más se adapten a las necesidades del proyecto.

2.2.2 Principales propuestas.

Para la elección de el juego a desarrollar se ha llegado a la conclusión de que un juego tradicional se ajustaba más a los requisitos del proyecto y por lo tanto se han considerado 4 juegos diferentes:

- *Poker*, un juego clásico de los más populares en la actualidad aunque sus reglas son muy conocidas vamos a analizarlas enfocándonos a su desarrollo como aplicación, el juego podría ser de 2 a 6 jugadores incluyendo o no una inteligencia artificial que supondría la banca en cuyo caso podría ser de 1 a 6 jugadores, siguiendo las reglas del *Poker Texas Hold'em* se repartirán 2 a cada jugador al inicio de cada turno y se dispondría a la apuesta inicial, al acabar la ronda se dispondrán de 5 cartas en la mesa de las cuales 3 están boca arriba y 2 boca abajo, nuevamente se dispondría a otra ronda de apuestas la cual acabar se desvela una de las cartas boca abajo y se dispondrán otra ronda hasta que todas las cartas estén destapadas, al final ganaría la jugada aquel jugador que tuviera la combinación de cartas entre las de la mesa y su mano que siguiendo una regla de puntuación tuviera la combinación más alta, además de esto si el resto de jugadores se retiran el último recibirá el dinero de los apuestas sin necesidad de revelar sus cartas, si uno de los jugadores se queda sin dinero éste podrá seguir jugando pero no recibirá una cantidad mayor a el doble de su apuesta.

- Blackjack, otro juego bastante popular con mecánicas más simples que el poker, este juego puede ser de 1 a 8 jugadores, su objetivo es conseguir 21 puntos mediante lma de los valores de las cartas. Los valores de las cartas en el BlackJack son los siguientes: las cartas del 2 al 10 valen su valor, las figuras valen 10 y el AS vale 1 u 11 dependiendo de lo que le convenga al jugador. El juego seguiría un juego por turnos en los cuales cada jugador en su turno podrá pedir cartas hasta que decida plantarse, llegue a 21 o se pase. Los jugadores con blackjack ganan un 1,5x1 de su apuesta inicial y los que se aproximen más a 21 que el crupier ganarán 1x1 de la apuesta inicial mientras que si se pasan o tienen menos que el crupier, perderán su apuesta.

- Parejas, juego conocido que destaca por su simpleza, el juego consistiría en distribuir sobre un tablero un conjunto de cartas de número par de las cuales para cada una de las cartas haya otra igual o similar, el juego puede utilizar cualquier tipo de cartas ya se ala baraja internacional o otro tipo mientras cumplan las reglas anteriores. El objetivo es levantar dos cartas por cada turno de jugador y si ambas coinciden el jugador sumará punto, si las cartas coinciden se dejan destapadas mientras que las que no sean iguales se vuelven a su posición inicial boca abajo, gana el que más puntos consiga, si uno de los jugadores recibe más de la mitad de los puntos es declarado ganador automáticamente.

- Mentiroso, menos conocido que los anteriores, este juego consiste en repartir todas las cartas de la baraja entre los jugadores, cada ronda comienza con un jugador declarando un número y una cantidad de cartas y poniéndolas boca abajo en la mesa, entonces el siguiente jugador puede colocar más cartas y pasar el turno al siguiente jugador o destapar las cartas colocadas por el jugador anterior, si es mentira el jugador anterior recoge todas las cartas en la mesa y el jugador actual comienza una nueva ronda pero en cambio si es verdad el jugador actual recibe el monto de cartas en su lugar y el siguiente jugador comienza una nueva ronda. El

ganador es el primer jugador en quedarse sin cartas en la mano mientras que el perdedor es el último jugador con cartas en su mano.

2.2.3 Análisis de las propuestas.

Con los anteriores juegos propuestos todos ellos cumplen con lo establecido por el proyecto, todos ellos son jugables por cuatro personas. En cuanto a su implementación hay que tener varios puntos a considerar en el proceso de selección.

Complejidad: En cuanto a complejidad, podríamos decir que el juego de las parejas seria el mas sencillo pues sus reglas son sencillas solo requiere de memoria, que además permite pocas variaciones respecto a sus reglas originales, el siguiente más complejo podría tratarse del blackjack, el cual a pesar de ser un juego de cuatro jugadores, el resto de jugadores tienen menos impacto que el resto de juegos este juego además introduce además la mecánica de la apuesta lo cual añade otro nivel de profundidad en cuanto al control de recursos en el juego. Por su lado, el mentiroso, el cual requiere de estrategia y ofrece elecciones más complejas que los juegos anteriores dispone de un sistema de penalizaciones acumulativas haciendo las decisiones más arriesgadas conforme avanza una ronda además una mala jugada pone al jugador muy en desventaja forzando a los jugadores a pensar con cautela cada uno de sus movimientos. Por último, se ha considerado el juego más complicado en cuanto a mecánicas se refiere al *poker*, el cual es un juego que permite bastantes variaciones, pues tiene diferentes estilos, el juego no solo posee un sistema de apuestas sino que está centrado completamente en la competitividad respecto al resto de jugadores, lo cual supone que una mala apuesta corra en tu contra, pues para recuperarla debes arriesgar cantidades similares con probabilidad variable, requiriendo de estrategia, memoria y del control no solo del recursos sino del propio estilo de juego de cada jugador.

Otro punto a tener en cuenta es su popularidad, optando por un juego tradicional, un juego con mayor popularidad podría atraer a más jugadores que uno en el cual se mostrara menos interés, en este sentido basándonos en la popularidad de estos juegos en Google Play el más popular sería el *poker*, seguido de el blackjack después se encuentra el de las parejas y por último el menos conocido sería el mentiroso, destacar que el *poker* y el blackjack destacan muchísimo en popularidad, sobre los demás debido a que son juegos de casino que además pueden llevar a los jugadores a obtener beneficio económico más allá del beneficio lúdico del juego en cuestión.

Aunque es una característica muy avanzada de un juego Android, se va a valorar el coste económico de la elección, los juegos de cartas están muy relacionados con el mundo de las apuestas y los juegos como el *poker* y el blackjack para su comercialización deberían de disponer de una inversión inicial ya que los jugadores estarían forzados depositar dinero real por la moneda que se utilice dentro de este juego, tambien podrian disponer de un sistema de publicidad que permitiera ganar dinero para la aplicación, por otro lado el juego de las parejas y el mentiroso podrían disponer de un sistema de apuestas integrado pero sería mucho más opcional en este caso pues no son juegos cuya finalidad inicial sea la de apostar, lo cual no requeriría de una menor inversión inicial para su comercialización.

2.3 Juego Elegido

El juego elegido es el Blackjack, la popularidad del juego le otorga una gran ventaja respecto a los otros candidatos, aunque el juego de las parejas es bastante conocido, sus aplicaciones comerciales no son tan conocidas y en lo que respecta al *poker*, el blackjack tiene como ventaja unas reglas más marcadas que el *poker*, el cual tiene diferentes variantes y se debería escoger la más adecuada y como antes se ha dicho se ha preferido optar por un juego tradicional con reglas marcadas, dando ventaja a el blackjack sobre el *poker*, el coste aunque menos importante al ser una de las partes finales del proyecto, no es uno de los objetivos principales del proyecto pero considerando que el blackjack es uno de los juegos más populares, podría ser una de las opciones a considerar en un futuro.

2.4 Posibles entornos.

Para implementar el juego de Blackjack existen una gran cantidad de plataformas disponibles, pero en este apartado no vamos a analizarlas todas sino que escogeremos entre algunas de las más comunes, antes que nada los lenguajes a los cuales se le ha dado preferencia a la hora de implementar la aplicación han sido Java y C# debido a su conocimiento previo.

Existen muchas opciones a la hora de implementar juegos para Android desde entornos simples como el *Notepad* de Windows hasta motores de desarrollo de videojuegos como el Unreal Engine el cual necesita de una inversión monetaria para poder usar la plataforma. Para este proyecto para reducir el coste económico todo lo posible se van a considerar opciones gratuitas o *Freemium* para su desarrollo.

Investigando los entornos de desarrollo más utilizadas para aplicaciones Android y más concretamente juegos Android se ha llegado a las siguientes posibilidades

Figura 4: Logo de Unity

Unity es una de las competencias tecnológicas de mayor demanda y tiene una de las tasas de crecimiento proyectado más altas, de más del 35 % en los próximos dos años.

Unity dispone de una versión personal de coste gratuito, además de esta dispone de una versión plus para aficionados y una versión pro para profesionales en la creación de videojuegos, la que nos interesa para este proyecto es la versión personal ya que nos permite un beneficio de hasta 100 mil dólares anuales sin necesidad de pagar por su licencia, cabe destacar que las licencias no gratuitas ofrecen grandes ventajas en cuanto a estadísticas analíticas sobre la aplicación y su uso por los jugadores, muy útil enfocado a nivel profesional.

Unity utiliza principalmente dos lenguajes C# y UnityScript una variante de javascript. Unity se enfoca en un lenguaje orientado a objetos los cuales se les llama *GameObjects* que interactúan entre ellos en una misma escena. Estos objetos además disponen de diferentes módulos y scripts a las cuales pueden pertenecer o ser parte de ellas.

Unity ofrece una interfaz simple y eficiente poco cargada que permite trabajar sobre 2D y 3D de forma eficiente y sencilla, las cuales además se pueden combinar para obtener un gran número de resultados. Además también soporta multiplataforma, es decir, se puede realizar una aplicación y hacerla funcionar tanto en Windows como en Android teniendo en cuenta una serie de consideraciones sencillas.

Por otro lado, Unity ofrece un sistema para monetizar la aplicación ya sea por micro transacciones o por publicidad.

Figura 5: Logo de GameMaker Studio

Game Maker Studio es otro de las plataformas consideradas para el proyecto, desarrollada por YoYo Games, utiliza como lenguaje de programación Delphi y se la considera una plataforma más orientadas a novatos por su simpleza y facilidad de uso, se centra principalmente en el desarrollo 2D.

Game Maker Studio dispone de diferentes licencias, la más básica siendo la licencia Educativa de 30$ que ofrece desarrollo para Windows y Ubuntu, para poder publicar juegos en otras plataformas se requiere de licencias de un coste mayor. YoYo Games también ofrece licencias para desarrollo en entornos más avanzados como Consolas que además disponen de mayores herramientas que ayudan a mejorar y optimizar el rendimiento de los juegos.

La Comunidad de Game Maker Studios también supone uno de sus puntos más fuertes, pues tiene una gran actividad en sus foros donde profesionales responden a respuestas y ofrecen ayuda a los principiantes en el mundo de la creación de videojuegos, además de disponer de una extensa guía de usuario.

El lenguaje Delphi es una versión derivada de Pascal la cual se le llama Object Pascal, el cual, cuyo nombre indicada está más fuertemente orientada a objetos y a una programación más visual, la cual es soportada por las plataformas antes nombradas. Delphi también dispone de una sintaxis simple que facilita la interacción entre las funciones y la interfaz visual del juego.

Figura 6: Logo de Unreal Engine

Unreal Engine representa uno de los motores de creación de videojuegos más potentes del mercado, creado por la empresa Epic Games fue inicialmente lanzado en 1998 para el juego llamado *Unreal,* aunque inicialmente estaba pensado para la realización de juegos *shooter* de primera persona. Con el tiempo ha ido recibiendo actualizaciones y actualmente se encuentra en su versión 4.20.

Es un motor multiplataforma que soporta la gran mayoría de sistemas, lo cual supone un alto grado de portabilidad. Unreal Engine está programado en base al lenguaje C++ anteriormente usaba un lenguaje llamado Unreal Scripting pero en su última versión fue reemplazada por C++ y Visual Scripting que permite un desarrollo más visual e intuitivo al momento de implementar la interacción entre los distintos objetos de juego.

La interfaz de Unreal Engine presenta un diseño más profesional, no es un diseño realizado para principiantes, su interfaz es más especializada con un gran número de módulos que permite la realización de tareas especialmente complejas en lo que respecta al diseño gráfico. Unreal Engine está principalmente enfocado a juegos desarrollados en 3d. Una de las partes más destacables de la interfaz de usuario es las llamadas *Blueprints* que permiten un desarrollo más ágil y visual de scripts que los desarrollados en programación tradicional, este sistema consiste en un desarrollo en forma de flujo de las acciones que realizará el sistema, supone un sistema difícil de aprender por la gran libertad creativa de la cual dispone pero muy potente.

En cuanto al modelo de negocio de Epic Games respecto a su motor es sencillo, Unreal Engine es gratis tanto para desarrollo comercial, como desarrollo comercial para productos que obtengan menos de 3.000 $ por cuatrimestre, a partir de ese punto se deberá abonar unos honorarios por el uso del motor del 5%, no dispone de ninguna ventaja adicional por cuota mensual como otros motores, sin embargo dispone de una Tienda Online donde se pueden obtener modelos ni *packs* adicionales que pueden llegar a facilitar el desarrollo de ciertos juegos.

Figura 7: Logo de AppGameKit

AppGameKit desarrollado por la empresa *TheGameCreators*, es un entorno de desarrollo de videojuegos principalmente diseñado para Android e IOS, pero también soporta plataformas como Windows o Linux. Se caracteriza por ser una plataforma para desarrolladores novatos y aficionados, además se destaca su uso para la creación de juegos *Indie*.

AppGameKit dispone de un lenguaje propio que presenta características de C++ pero a un nivel mucho más simplificado para su fácil entendimiento en desarrolladores noveles. Presenta un bajo consumo de recursos, pues no destaca por ser una de las herramientas más potentes del mercado a la hora de crear videjuegos, recordemos que los videojuegos denominados *indie* no suelen destacar en el apartado gráfico, más bien, lo que caracteriza este tipo de juegos es la originalidad y su reducida inversión económica.

El producto en cuestión presenta una versión de cuota única que permitiría realizar el desarrollo de el juego sin ningún tipo de cargo adicional, independientemente de , además dispone de una versión de prueba y una versión gratuita para la educación, también dispone de diferentes versiones con mayor coste que presentan características más potentes y especializadas. su precio de venta de la aplicación base es de 80 €.

2.5 Entorno seleccionado

El entorno seleccionado ha sido Unity, Unity ofrece un buen tutorial introductorio y una gran comunidad, además los lanzamientos de juegos recientes demuestran que Unity es una gran herramienta de desarrollo de juegos independientes que ofrece una gran personalización para que los creadores adaptan el entorno a su gusto, otro aspecto a destacar es el costo el cual supone una licencia gratuita para uso de proyectos personales, la cual encaja en este tipo de proyecto.

3. Análisis

Una vez seleccionado el juego y la plataforma vamos a pasar a la fase de análisis, en este apartado vamos a desglosar las distintas partes antes del desarrollo y plantear los distintos problemas que se van a encontrar a la hora de realizar el juego de cartas.

En esta fase se plantea el diseño, la lógica y la estructura que seguirá el juego y las necesidades que estos plantean antes de llevarlas a cabo en la fase de desarrollo. Para finalizar, en la fase de análisis se especificarán los objetivos que el proyecto va a alcanzar y cuáles van a dejar para futuros desarrollo por diferentes razones.

3.1 Análisis de requisitos

Debido a que el proyecto está diseñado para Android, todo su diseño y desarrollo debería estar centrado en dicho entorno, por lo tanto, acotando ciertas partes del problema que nos pueden llevar a diversas soluciones. Para ello se plantean los tres puntos principales a analizar del juego:

Interfaz: Siendo para sistemas Android, la interfaz del juego debería adecuarse a las interfaces más comunes de este tipo de juegos. Los juegos de Android y más concretamente los juegos de móvil presentan interfaces más simples con una carga visual mucho más sencilla y más intuitiva que los juegos de otras plataformas, puesto que están pensadas para un entorno táctil y sesiones de juego mucho más cortas y esporádicas. Dicho esto el juego de cartas debería presentar una interfaz sencilla que muestre estrictamente la información más relevante para el usuario y lo guíe de una forma clara y precisa por el juego sin necesidad de un conocimiento elevado del medio.

mecánicas de juego: Para el juego a desarrollar se ha decidido basarse en el juego de cartas tradicional *blackjack*, por lo tanto el juego tendrá reglas similares o muy similares a como las del juego original. Al tratarse de un juego para Android las mecánicas deben quedar suficientemente simplificadas para enfocarse en un público más casual que en caso de no haber jugado nunca al juego original comprendan de forma sencilla en un par de partidas como se juega y cómo puede ganar en el juego.

multijugador: El juego debe estar pensado para cuatro jugadores o hasta cuatro jugadores y además debe permitir jugar online entre ellos, para ello se deberán investigar las distintas formas de las que el motor *Unity* hace uso para permitir conectividad entre los distintos jugadores y además que estructura de red es la más apropiada para llevar a cabo la conexión, otro de los puntos a destacar en este requisito es la compatibilidad en Android de los distintos tipos de conexión del motor porque cabe destacar que la aplicación deberá ser testeada en entornos Android pero se va a realizar su desarrollo en Windows.

3.2 Análisis Interfaz

Para la interfaz hay que realizar diferentes consideraciones a la hora de diseñar, las características que más deben destacar son la simplicidad y la sencillez de utilización tanto para nuevos usuarios como para usuarios ya experimentados.

Las cartas deben estar distribuidas de forma que se diferencie cada jugador de una forma clara y que no dé lugar a ninguna confusión, para ello, como el límite es de cuatro jugadores, se puede hacer uso de una distribución circular alrededor de la interfaz de juego.

El marcador también es uno de los puntos más importantes a destacar, no solo debe estar señalizado el marcador del jugador sino los marcadores del resto de jugadores por lo que un marcador cercano a la zona de cartas que tiene cada jugador para que, de esta forma, sea mucho más visual y coherente con el flujo del juego, los marcadores además deben de presentar la información clara pero sin que suponga una gran sobrecarga de datos para el usuario. Los marcadores deberán contener, la cantidad de la que dispone el jugador, la cantidad apostada y el la puntuación del jugador en dicha ronda, por lo que se debe ajustar esta información de una forma sencilla.

Por último, otra parte de la interfaz de juego son los botones, estos botones son los que controlarán y llevarán a cabo las decisiones de cada jugador, cada jugador dispondrá de unos botones para la realización de las jugadas pero por simplicidad todos los jugadores tendrán una botonera común la cual se habilitará únicamente para el turno del jugador, mientras un jugador esté tomando una decisión el resto de jugadores no podrán realizar ninguna jugada hasta que no sea su turno.

3.3 Mecánicas de juego

En el diseño de las mecánicas uno de los puntos más importantes a destacar es el flujo que va a seguir el juego, las reglas tienen que quedar bien establecidas y que no presenten contradicciones. Otro aspecto importante tener en cuenta en cuanto al flujo es que al tratarse de un juego Android si dentro de una misma jugado el control va pasando entre los jugadores cada poco tiempo puede resultar mucho más complicado, por lo tanto en cada turno, el jugador resolverá su jugada sin depender de los demás jugadores, es decir, el jugador irá pidiendo cartas hasta que decida plantarse, gane o pierda la ronda por superar la puntuación requerida, una vez este se resuelva pasará el control del juego al siguiente jugador.

El juego se divide en diferentes rondas, cada ronda tendrá la misma estructura pero al unirse a una partida todos los jugadores comenzarán con una cantidad fija de dinero, si el jugador se queda sin dinero, no podrá seguir jugando, el jugador podrá abandonar en cualquier momento la partida. Cada ronda está dividida en diferentes partes:

Apuesta, en esta fase todos los jugadores deberán decidir la cantidad que van a apostar y esperarán a que todos los jugadores acaban de apostar antes de pasar a la siguiente fase de juego.

Reparto de cartas, esta es la fase inicial donde se reparte una carta a cada uno de los jugadores, en esta fase además deberán quedar limpias las manos de todos jugadores respecto de la ronda anterior y que además reiniciará las puntuaciones de los jugadores para que empiece a contar la nueva ronda.

Demanda de cartas, después de realizar las apuestas cada jugador tendrá un turno para demandar cartas hasta que se plante, llegue a 21 o se pase de dicha puntuación según las reglas del juego, una vez cada jugador ha llegado a una conclusión comenzará el turno del crupier que intentará alcanzar 21 o acercarse lo suficiente como para que el mínimo de jugadores le gane la ronda, una vez el crupier ha finalizado se pasará a la última fase de la ronda.

Resolución, en esta fase se decide quienes han ganado, quienes han perdido y quienes han empatado contra el crupier, después de esto los jugadores ganadores se les asignará la cantidad correspondiente que han ganado y los que han empatado recibirán su apuesta inicial como reembolso, al acabar esta fase se comenzará otra vez con la fase de reparto de cartas hasta que todos los jugadores abandonen la sala.

Fase de Espera: esta fase empieza al acabar la ronda y es la fase intermedia donde los jugadores deciden si quieren seguir jugando o abandonan la partida.

Para mayor claridad se ha realizado un esquema (Figura 8), este muestra cómo interactúan las diferentes fases del juego y de la misma forma establece el flujo del juego.

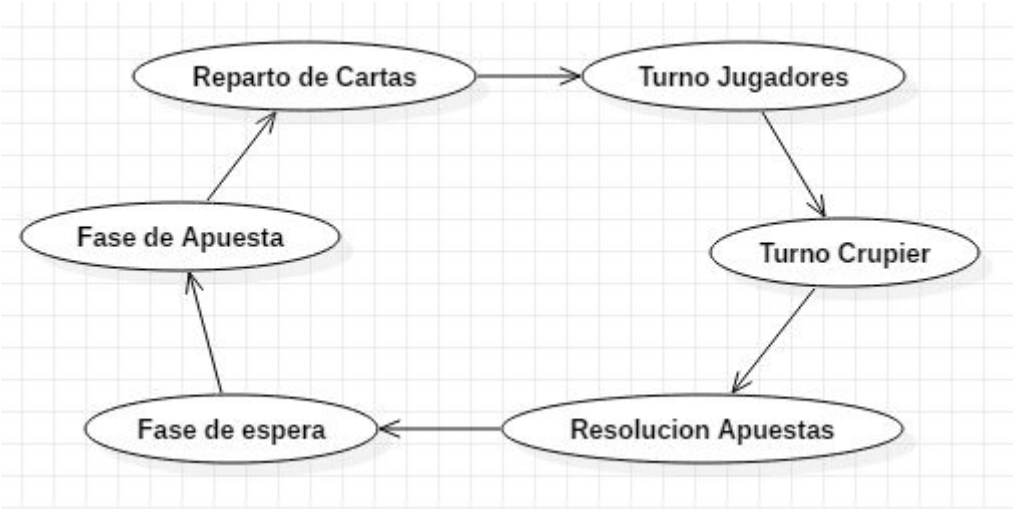

Figura 8: Fases del Blackjack

3.4 Multijugador

En el apartado multijugador de la aplicación su principal objetivo será la conexión entre los distintos jugadores para la realización del juego, este sistema multijugador deberá disponer de dos características principales, primero la capacidad de obtener la información de otro jugador para conectarse a su partida, la cual requerirá de una forma de identificación, y la segunda característica será el modo en que los jugadores de una misma partida comparten la información sobre las acciones que se llevan a cabo en las diferentes fases, permitiendo al juego organizar los turnos de los jugadores dentro de una misma jugada.

Para ello será necesario que algún componente de la aplicación sea capaz de actuar como servidor para que los clientes puedan interactuar en el juego, por ejemplo que un jugador cree la partida y el resto de jugadores se una a ella.

3.5 Casos de uso

En esta apartado se procederá a realizar el análisis de los casos de uso, primero se realizará un diagrama que muestre los usos que hará un jugador sobre la aplicación y posteriormente se analizarán uno a uno los casos resultantes.

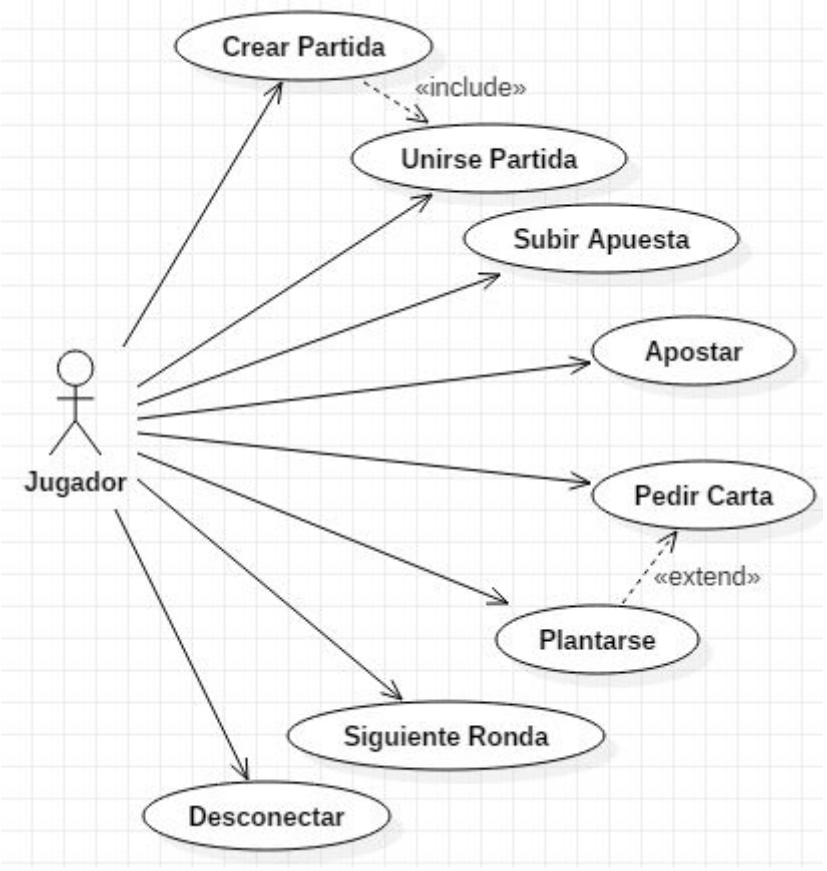

Figura 9: Diagrama de Casos de uso del juego

Caso de uso	Crear partida	
Actores	Jugador	
Resumen	El jugador inicia la aplicación e inicia la partida que comenzará con la primera ronda de juego	
Precondiciones	--	
Postcondiciones	El jugador se une a la partida y se ha creado una partida para que accedan nuevos jugadores.	
Incluye	Si el jugador crea una partida este se une a ella	
Extiende	--	
Hereda de	--	
Flujo de Eventos		
Entidad	**Paso**	**Acción**
Jugador	1	Pulsa el botón de crear partida en la pantalla de inicio
Sistema	2	Crea un servidor para que se unan más jugadores
Sistema	3	Crea un jugador para que el propietario pueda jugar
Sistema	4	Comienza el juego con la fase de apuestas
	5	
	6	

Caso de uso	Unirse a partida	
Actores	Jugador	
Resumen	El jugador inicia la aplicación y se une a una partida existente en cualquier momento de la ronda	
Precondiciones	Debe haber una partida creada en curso	
Postcondiciones	Se crea un nuevo jugador que debe esperar a que acabe la ronda si ya ha pasado la fase de apuesta	
Incluye	--	
Extiende	--	
Hereda de	--	
Flujo de Eventos		
Entidad	**Paso**	**Acción**
Jugador	1	Pulsa el botón de unirse en la pantalla de inicio
Sistema	2	Jugador espera a que acabe la ronda y llegue la fase de apuestas
Sistema	3	Comienza el juego con la fase de apuestas
	4	
	5	
	6	

Caso de uso	Subir apuesta	
Actores	Jugador	
Resumen	El jugador pulsa el botón de subir apuesta y su apuesta se incrementa en 50 monedas	
Precondiciones	El jugador debe haberse unido a una partida y el juego se encuentra en fase de apuestas	
Postcondiciones	Se ha incrementado la cantidad apostada	
Incluye	--	
Extiende	--	
Hereda de	--	
Flujo de Eventos		
Entidad	**Paso**	**Acción**
Jugador	1	Pulsa el botón subir apuesta
Sistema	2	El servidor acepta la cantidad apostada y aumenta la apuesta en 50
	3	
	4	
	5	
	6	

Caso de uso	Apostar	
Actores	Jugador	
Resumen	El jugador pulsa el botón de apostar y se confirma la apuesta del jugador	
Precondiciones	El jugador debe haberse unido a una partida, el juego se encuentra en fase de apuestas y este debe haber subido la apuesta al menos una vez.	
Postcondiciones	Se confirma la apuesta y una vez que todos los jugadores apuesten se pasa a la fase de juego	
Incluye	--	
Extiende	--	
Hereda de	--	
Flujo de Eventos		
Entidad	**Paso**	**Acción**
Jugador	1	Pulsa el botón de apostar
Sistema	2	El servidor espera a que todos los jugadores apuesten
Sistema	3	El sistema pasa a la fase de juego
Sistema	4	El sistema reparte dos cartas a cada jugador
	5	
	6	

Caso de uso	Pedir carta	
Actores	Jugador	
Resumen	El jugador pulsa el botón de pedir carta, recibe una carta aleatoria de la baraja y se recalcula su puntuación	
Precondiciones	El jugador ha realizado su apuesta, es su turno y no tiene una puntuación igual o superior a 21	
Postcondiciones	La puntuación se actualiza y en caso de tener una puntuación igual o superior a 21 no podrá volver a pedir carta	
Incluye	--	
Extiende	--	
Hereda de	--	
Flujo de Eventos		
Entidad	**Paso**	**Acción**
Jugador	1	Pulsa el botón de pedir carta
Sistema	2	El sistema reparte una carta aleatoria no usada al jugador
Sistema	3	El sistema recalcula la puntuación del jugador
Sistema	4	El sistema pasa el turno al siguiente jugador si la puntuación del jugador es 21 o superior
	5	
	6	

Caso de uso	Plantarse	
Actores	Jugador	
Resumen	El jugador pulsa el botón de plantarse y pasa el turno al siguiente jugador o al crupier en cuyo caso realizará su turno y luego se decidirán los ganadores y los perdedores	
Precondiciones	El jugador ha realizado su apuesta y es su turno.	
Postcondiciones	Al acabar el turno todos los jugadores, se pasa el turno al crupier y una vez termina el sistema reparte el dinero a los ganadores	
Incluye	--	
Extiende	Si al pedir carta el jugador tiene 21 o superior este no puede pedir más cartas y se planta.	
Hereda de	--	
Flujo de Eventos		
Entidad	**Paso**	**Acción**
Jugador	1	Pulsa el botón de plantarse
Sistema	2	El servidor pasa el turno a otro jugador o al crupier si este es el último jugador
Sistema	3	El crupier realiza su turno
Sistema	4	El sistema calcula los ganadores y perdedores
Sistema	5	Se reparte a lo ganadores el doble de la cantidad apostada
	6	

Caso de uso	Siguiente ronda	
Actores	Jugador	
Resumen	El jugador pulsa el botón de siguiente ronda y espera a que todos los demás jugadores decidan, una vez todos los jugadores están listos comienza otra vez la fase de apuestas	
Precondiciones	El jugador se ha plantado y se ha resuelto la ronda anterior.	
Postcondiciones	Una vez todos los jugadores están listos comienza la fase de apuesta de nuevo	
Incluye	--	
Extiende	--	
Hereda de	--	
Flujo de Eventos		
Entidad	**Paso**	**Acción**
Jugador	1	Pulsa el botón de siguiente ronda
Sistema	2	El servidor pasa el turno a otro jugador o al crupier si este es el último jugador
Sistema	3	El crupier realiza su turno
Sistema	4	El sistema calcula los ganadores y perdedores
Sistema	5	Se reparte a lo ganadores el doble de la cantidad apostada
	6	

Caso de uso	Desconectar	
Actores	Jugador	
Resumen	El jugador pulsa el botón de desconexión y abandona la partida	
Precondiciones	El jugador se ha unido a una partida	
Postcondiciones	Si el jugador ha realizado alguna apuesta no podrá recuperar las monedas	
Incluye	--	
Extiende	--	
Hereda de	--	
Flujo de Eventos		
Entidad	**Paso**	**Acción**
Jugador	1	Pulsa el botón de desconexión
Sistema	2	El sistema elimina al jugador de la partida
Sistema	3	La ronda sigue con los jugadores restantes, en caso de que se desconecte el jugador que ha creado la partida, esta se da por terminada
	4	
	5	
	6	

3.6 Diagrama de clases

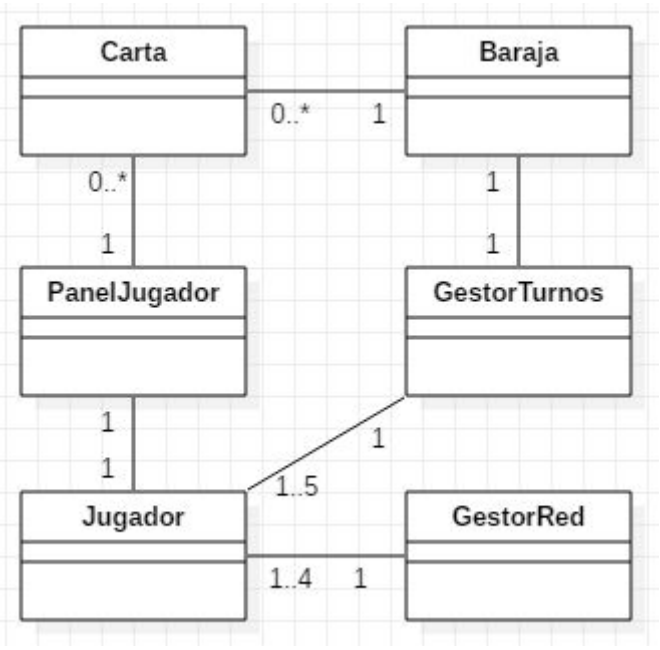

Figura 10: Diagrama de clases de la aplicación

Como se observa en el diagrama(Figura 10), la clase Carta es una de las clases más básicas del juego, se utiliza en la clase PanelJugador y Baraja. La clase PanelJugador representa los paneles de jugadores vistos en la interfaz, la clase PanelJugador se utiliza por la clase Jugador que es la clase que se asigna a cada uno de los jugadores y al crupier. La clase GestorConexiones usa la clase Jugador para añadirlos o quitarlos de la partida. Por último, la clase GestorTurnos utiliza las clases Baraja y Jugador para gestionar las rondas del juego.

4. Diseño

Siguiendo la progresión del apartado anterior, en el apartado de diseño se va a realizar detalladamente el diseño conceptual de la aplicación siguiendo los requisitos establecidos anteriormente, para ello, se irán traduciendo los requisitos anteriores.

Inicialmente se comenzará por el diseño de la interfaz para definir como el usuario interactúa con la interfaz, luego se procederá a la implementación de las clases y como llevan a cabo su función dentro de la aplicación.

4.1 Diseño de la interfaz

4.1.1 Pantalla de Inicio

Usando como base el esquema del apartado de análisis vamos a analizar los aspectos más importantes de la interfaz y su diseño.

Lo primero que hay que destacar es la pantalla de inicio, dicha pantalla supone la parte inicial de la aplicación, desde la cual se accede al menú de conexión de la aplicación.

Figura 11: Pantalla de inicio de la aplicación

La interfaz de conexiones, pese a que su implementación se explicará más adelante, permitiría crear una partida utilizando su primer botón que hará al usuario tanto servidor como cliente siendo el *host* de la partida y el jugador 1 al mismo tiempo, aunque como propietario de la partida no tendrá ninguna función extra que realizar, el segundo botón permite unirse a una partida ya creada como cliente.

Una vez cualquiera de las opciones se ha seleccionado, se cargará la pantalla, la cual constituye la pantalla principal del juego, donde se llevarán a cabo las partidas de *Blackjack*.

4.1.2 Pantalla de Juego

El soporte principal de la interfaz donde se colocan todos los objetos del tablero, este objeto consiste en un fondo verde que es uno de los colores distintivos de las mesas de juego de los casinos, este tablero también estará ajustado a la pantalla y con las propiedades que le permiten escalar a cualquier dimensión en una pantalla de dispositivo Android, esta función es esencial porque existen una gran cantidad de dispositivos Android que presentan pantallas muy diferentes a diferencia de los pc de sobremesa en las cuales el ratio de las pantalla suele ser muy similar entre los monitores.

El siguiente componente a destacar de la interfaz son los paneles del crupier y los jugadores, todos ellos muy similares entre sí que presentan la Figura 12.:

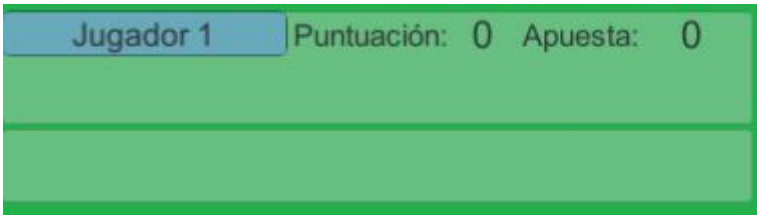

Figura 12: Diseño del panel del jugador

El marcador permite obtener información en tiempo real del estado en el que se encuentra, tanto la puntuación como la apuesta se actualiza después de cada acción realizada por el jugador permitiendo tomar las decisiones que sean más convenientes para el flujo del juego, cabe destacar que todos los jugadores dispondrán de la misma información que el resto de jugadores así pues el jugador 1 podrá saber las puntuaciones y las apuestas del resto de los jugadores y del crupier.

La parte del panel de cartas es donde se van almacenando las cartas que el jugador recibe durante una ronda, cuando llega el turno de un jugador el juego resalta el panel de dicho jugador en blanco para que se sepa cuál es el jugador activo en cada ronda. Para que los jugadores sientan más realismo las cartas no están completamente separadas unas de otras, sino que las cartas nuevas se van superponiendo y tapando las anteriores para dar un efecto de cartas en la mano emulando los juegos de mesa tradicionales, además sirve para que el jugador preste atención a la nueva carta que ha recibido de una forma limpia sin necesidad de indicadores añadidos. En la Figura 13 se puede apreciar como es el panel del jugador en mitad de una ronda:

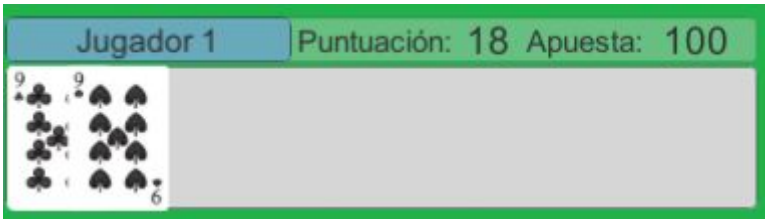

Figura 13: Panel de cada jugador activo

Para las cartas se ha utilizado una imagen libre de *copyright* del conjunto de cartas de la baraja original de *poker*, se puede observar su diseño en la Figura 14.

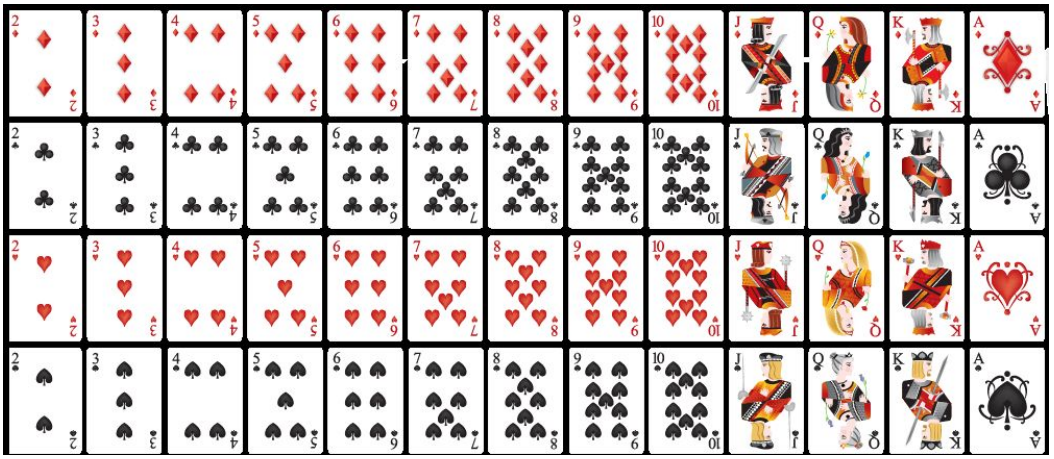

Figura 14: Imagen que representa las cartas de el juego.

Para que sirva de ayuda visual a los jugadores se ha diseñado un panel informativo, este componente de la interfaz está diseñada para guiar a los jugadores durante las distintas fases del juego, el texto irá cambiando indicando el estado del juego en todo momento además una vez terminada la ronda el texto mostrará cuánto dinero ha ganado o perdido el jugador al acabar la misma.

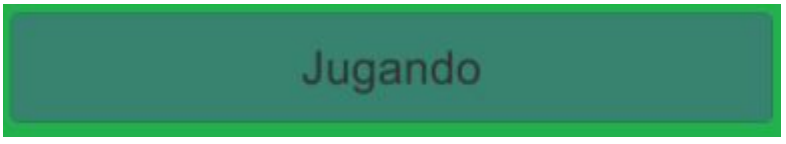

Figura 15: Texto informativo del estado de juego.

Mientras que la parte superior de la interfaz proporciona información sobre el estado de la ronda, la parte inferior proporciona al jugador las herramientas para que pueda tomar sus decisiones, este a su vez, se divide en dos partes, los botones y el marcador.

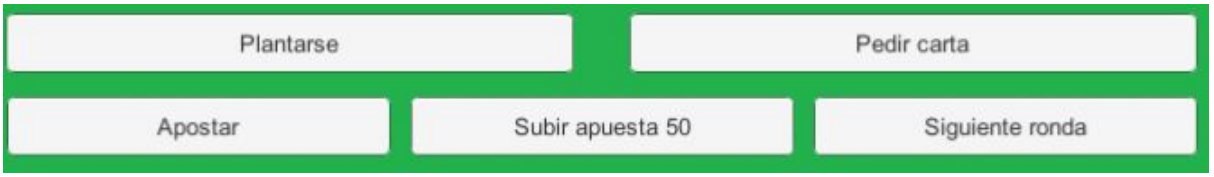

Figura 16: Botonera de la interfaz de juego

Los cinco botones, se habilitan en función de la fase en la que se encuentre la ronda, el botón de apostar se habilita únicamente en la fase de apuesta, los botones de Plantarse y Pedir carta se habilitan una vez sea el turno del jugador, estos botones seguirán habilitados mientras el jugador no se pase de 21 o decida plantarse, mientras se llevan a cabo los turnos del resto de jugadores todos los botones quedarán inhabilitados a la espera de que la ronda actual concluya, una vez terminada la ronda de juego, se habilita el botón de siguiente ronda, la cual, dará comienzo una vez una vez todos los jugadores hayan decidido si siguen jugando o dejan la partida.

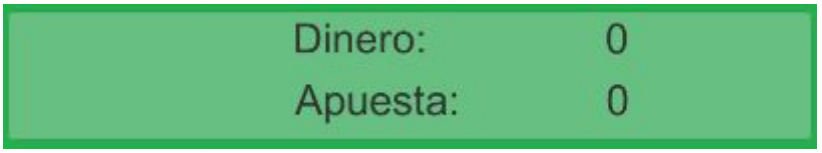

Figura 17: Marcador en la esquina inferior derecha de la interfaz

El marcador(Figura 17) muestra la cantidad de dinero que dispone el jugador en cada momento de la partida y la apuesta refleja simplemente cuánto dinero tiene en juego el jugador, cada jugador tendrá un marcador diferente, pues estos marcadores reflejan las estadísticas de los jugadores locales, se ha colocado a la derecha de los botones para que tenga fácil visibilidad a la hora de decidir la apuesta pero que no estorbe con el flujo del juego en la pantalla principal del juego.

Figura 18: Diseño final de la interfaz de juego

La interfaz final presenta el aspecto de la Figura 18, aunque no se puede apreciar los jugadores se encuentran distribuidos alrededor de la pantalla emulando las mesas de los casinos donde el crupier se coloca al frente y los jugadores se sitúan alrededor de la parte de la mesa con forma de media luna. El panel informativo se ha situado en el centro de la pantalla para que los jugadores no pierdan detalle de lo que está sucediendo en el juego aunque no sea su turno.

4.2 Diseño de clases

En este apartado, utilizando como referencia el diagrama de clases de la Figura 10, se dispondrá a analizar cada una de las clases y realizar su diseño para simplificar la fase de implementación.

4.2.1 Clase Carta

La clase Carta es la que identifica las cartas que hacen uso los jugadores. Esta clase es la clase más usada por las otras clases de la arquitectura pues, todo el juego gira alrededor de las cartas. La Clase carta se compone de las siguientes variables:

- **sprites:** esta variable recoge un array de *Sprites*[1], el array estará formado por las imágenes de cada una de las cartas, en este caso 52 más la carta vacía haciendo un total de 53 valores posibles, cuando una carta se desea que no tenga ningún valor y se oculte, esta tendrá asignado el *sprite* número 52 que es el *sprite* vacío.
- **índiceCarta:** esta variable identifica el valor de la carta, esta variable apunta a uno de los valores del array de *sprites* para que esta pueda tomar su imagen y pueda conocerse su valores utilizando uno de los métodos de la clase.

El método implementado en esta clase tienen como función traducir números enteros al *sprite* que corresponda para que sea mostrado en la interfaz de juego. La clase dispone del siguiente método:

- **asignarSprite:** este método tiene como función asignar el *sprite* de la carta correspondiente a partir de su posición en el array, por ejemplo, la posición 0 corresponde a la primera imagen de carta que es el dos de picas.

4.2.2 Clase Baraja

La clase Baraja es la que agrupa las cartas y permite gestionar este conjunto de cartas tanto para obtener una de sus cartas como para reiniciar el contenido para comenzar la siguiente ronda. La clase tiene las siguientes variables:

[1] Los *Sprites* es el nombre que reciben las imágenes 2D que componen los distintos valores que un objeto o personaje puede tomar, un ejemplo de esto es el conjunto de imágenes que toman los personajes de juegos 2D para su movimiento.

- **cartasActivas:** esta variable consiste en un array de enteros que inicialmente contiene los números del 0 al 51 para identificar cada carta de la baraja.
- **cartasUsadas:** esta array se inicializa a 99 en todas sus posiciones y tiene las mismas posiciones que en el array anterior, sirve para identificar las cartas que se han ido usando durante la ronda, para que no sean repetidas en la misma.

Los métodos implementados en esta clase tienen como función organizar la baraja ya sea restableciendo u obteniendo una carta aleatoria de ella, un aspecto importante del juego es que aunque la baraja se encuentre dividida en 4 palos, no es necesario distinguirlos entre ellos pues un 10 de picas y un 10 de corazones van a tener siempre la misma puntuación, lo único que se debe tener en cuenta es que no se utilicen más de 4 cartas del mismo número. La clase dispone de los siguientes métodos:

- **barajar:** este método es el que se lanza al incio de cada ronda para devolver todas las cartas a sus posiciones en el array, recorre ambos arrays anteriores situando en las posiciones correspondientes los números iniciales.
- **robarCarta:** este método devuelve una variable tipo Carta, al llamar este método obtiene un número aleatorio del 1 al 52 y recoge la posición en el array sustituyéndolo por un 99 en las cartasActivas y en las cartasUsadas los substituye por su número en dicha posición así identifica qué cartas están usadas y cuáles no, el método realizará distintas iteraciones si encuentra cartas repetidas hasta que encuentre una que no esté usada.

4.2.3 Clase PanelJugador

La clase panel jugador es la encargada de representar cada espacio del jugador en la pantalla de juego, tiene dos funciones principales, la primera es gestionar las manos de los jugadores y su otra función es mantener los marcadores actualizados en la pantalla. Las variables de esta clase son las siguientes:

- **puntuacionTexto**: representa el objeto acerca de la puntuación de cada jugador.
- **apuestaTexto**: representa el objeto acerca de la cantidad apostada de cada jugador.
- **mano**: esta variable consiste en un array de Cartas, como su nombre indica, representa la mano de cada jugador donde se irán insertando las cartas que el jugador irá pidiendo, por defecto este objeto tiene asignadas las cartas de la interfaz y su *sprite* al iniciar la ronda es el número 53, que corresponde con el *sprite* vacío.
- **numeroCartas**: este entero representa el número de cartas que tiene un jugador en la mano que coincide con la posición en el array en la cual se realizarán las modificaciones al obtener nuevas cartas.
- **puntuacion**: entero que representa la puntuación del jugador en la ronda actual.
- **apuesta**: entero que representa la cantidad apostada del jugador en la ronda actual.

- **imagenPanel**: esta variable identifica el fondo del panel.

Las funciones de esta clase están centradas en ofrecer las funciones a las clases superiores en la jerarquía para modificar las cartas y poder actualizar sus valores, por sí misma esta clase no puede modificar el flujo del juego. Dispone de los siguientes métodos:

- **obetnerPuntuacion:** este método calcula la puntuación a partir de las cartas en la mano, para ello realiza el módulo 13 de cada carta, si el resultado es de 0 a 9, esta suma 1 y devuelve el resultado, si es 10,11 o 12 esto significa que es una figura por lo que devuelve 10, además esta función también tendrá en cuenta que si se trata de un 1, la puntuación obtenida será la más adecuada para el jugador, siempre tendrá en cuenta que los 1 cuentan como 10 hasta que este supere 21 en cuyo caso contará como un 1.

- **actualizarPuntuacion**: esta función se encarga de asignar al texto de puntuación que corresponde al objeto de juego del marcador de la interfaz.

- **vaciarMano**: devuelve todas las cartas del array a su *sprite* vacío y las pone a 99 para que la mano cuente como vacía.

- **anadirCarta**: recibe un objeto tipo Carta como parámetro, la asigna a la posición indicada por numeroCartas y llama los métodos obtenerPuntuacion y actualizarPuntuacion para que calculen y asignen los datos correspondientes, además llama a la función asignarSprite para actualizar la imagen de la carta.

- **activarTurno**: este método recibe un color como parámetro y cambia el color que tiene la imagenPanel para identificar qué jugador está realizando su turno.

4.2.4 Clase Jugador

La clase jugador es la que representa de forma lógica a los jugadores de cada partida a diferencia de la clase PanelJugador que representaba a los jugadores de forma física en la interfaz. La función principal de esta clase, es proporcionar los comandos que los clientes enviarán al servidor para que realice ciertas funciones del juego, esta clase además se ha implementado en el objeto de juego Jugador el cual se crea cada vez que un nuevo jugador se conecta y se destruye cuando un jugador abandona la partida, toda la gestión de unirse a la partida la realizan las clases GestorTurnos y GestorRed. Todas las variables tienen el atributo *SyncVar*[2]*.* Estas son las siguientes variables de las cuales dispone la clase:

- **panelJugador**: Esta variable representa el panel del jugador en la interfaz donde el jugador podrá observar su mano, puntuación y apuesta, durante dicha ronda.

- **dinero**: Representa la cantidad disponible que tiene el jugador al inicio de cada ronda, puesto que el juego no tiene un sistema de monetización los jugadores

[2] *SyncVar* son variables de scripts que que se sincronizan desde el servidor a los clientes. Cuando se genera un objeto de juego, o un nuevo jugador se une a un juego en curso, se les envía el último estado de todas las *SyncVars* en los objetos en red que son visibles para ellos.

comenzarán cada partida con 100 monedas las cuales irán apostando durante las diferentes rondas que dure la partida.

- **apuesta**: representa la cantidad apostada por el jugador en dicha ronda, la cual condiciona la cantidad ganada o perdida al final de la ronda.
- **dineroTexto**: representa el objeto acerca del dinero actual del jugador en el marcador, cada jugador verá únicamente su marcador.
- **apuestaTexto**: representa el objeto acerca de la cantidad apostada del jugador en el marcador, cada jugador verá únicamente su marcador.
- **activo**: si el jugador ha entrado en la partida a mitad de ronda esta variable se encontrará a *false* hasta que comience una nueva fase de reparto de carta y su posterior apuesta, esta variable es necesaria para que no se produzcan errores como jugadores jugando a mitad de ronda sin apuesta o casos similares.
- **puntuacion**: representa la puntuación que tiene el jugador en su mano a cada momento, debe estar actualizada
- **crupier**: identifica mediante una variable tipo *bool* si este objeto jugador es el crupier, pues este no está controlado por ningún jugador sino por la inteligencia artificial asignada al servidor.
- **indiceJugador**: identifica a cada jugador en el reparto de turno así pues el jugador con índice 1 irá primero y el crupier tendrá asignado el índice 5 por defecto para realizar su turno en último lugar.

Las funciones principales de esta clase consisten en implementar las funcionalidades de las cuales harán uso los jugadores y además implementa la inteligencia artificial de la cual hará uso el servidor cuando sea el turno del crupier. Las funciones para los jugadores llevarán el atributo *command*[3] y estarán integradas en los botones de la interfaz. Las funciones son las siguientes:

- **subirApuesta**: este comando aumenta la cantidad apostada en 50, la cantidad apostada no podrá superar la cantidad disponible que tiene el jugador, esta función, además de modificar la variable sincronizada con el servidor, debe actualizar también los textos en el panel de jugador y en el marcador inferior de la interfaz.
- **apostar**: requiere de haber usado el comando anterior para poder utilizarse, su función es confirmar la apuesta anterior al servidor y , una vez todos los jugadores han confirmado su apuesta, el servidor cambia de estado de juego y le da el turno al primer jugador.

[3] el atributo *command* sirve para realizar acciones remotas en entornos de conectividad red, por ejemplo ejecutar funciones sobre objetos pertenecientes a un servidor.

- **pedirCarta**: obtiene una carta aleatoria de la baraja y utiliza el método añadirCarta para añadirla a la mano, mostrarla en el la interfaz y actualizar la puntuación del jugador que corresponda.
- **plantarse**: confirma la puntuación que ha conseguido el jugador en dicha ronda y devuelve el control del juego al servidor para que asigne el control a otro jugador
- **siguienteRonda**: envía una confirmación al servidor para que este pueda comenzar la siguiente ronda una vez todos los jugadores hayan aceptado o hayan abandonado la partida.
- **turnoCrupier**: esta función a diferencia de las anteriores lleva el atributo *server* en vez de *command* lo que significa que es una función que solo puede ser ejecutada por el servidor, implementa la inteligencia artificial del crupier que una vez se lanza irá sacando cartas hasta que tenga mayor puntuación que todos de los jugadores o en caso de que algún jugador tenga 21, el crupier intentará conseguir la misma puntuación para minimizar las pérdidas.

4.2.5 Clase GestorTurnos

La clase GestorTurnos es la encargada de gestionar todo el flujo del juego, esta clase es considerada la clase servidor, esta clase es la que se encuentra más arriba en la jerarquía y utiliza todas clases anteriores para la correcta sincronización del juego. Esta clase presenta las siguientes variables:

- **baraja**: variable tipo Baraja que representa la baraja que se va a ir utilizando durante la ronda de juego y se reiniciará al principio de cada ronda.
- **crupier**: variable de tipo Jugador que identifica al crupier
- **jugadores**: Lista de jugadores donde se almacenan todos los jugadores activos excepto el crupier, los jugadores se irán añadiendo y eliminando de forma enlazada según entren o se vayan los jugadores de la partida.
- **jugadorActual**: variable de tipo Jugador que identifica al jugador que está realizando su turno en la fase de juego.
- **indiceJugadorActual**: entero representativo del índice de jugador que está realizando su turno.
- **estadoJuego**: variable tipo cadena que sirve para identificar en qué fase se encuentra el juego.
- **botonSubirApuesta**: representa el botón SubirApuesta de la interfaz.
- **botonApostar**: representa el botón Apostar de la interfaz.
- **botonPedirCarta**: representa el botón PedirCarta de la interfaz.

- **botonPlantarse**: representa el botón Plantarse de la interfaz.
- **botonSiguienteRonda**: representa el botón SiguienteJugada de la interfaz.
- **textoInformativo**: representa el objeto texto que muestra el progreso de la ronda en el centro de la interfaz de juego.

La clase GestorTurnos tiene diferentes funciones tanto para clientes como para el servidor, las funciones de los clientes consisten principalmente en gestionar que botones están disponibles en cada fase del juego, las funciones del servidor se encargan de ir estableciendo los diferentes estados de juego y realizar las funciones que correspondan a dicho estado. Durante cada fase el servidor irá actualizando el texto informativo al centro de la pantalla. Dada la gran cantidad de funciones que intervienen se explicara de una forma más simplificada las funciones que intervienen en cada fase de juego.

- **Inicio de la partida**: Al comenzar la partida se inicializa un jugador para identificar al crupier con el atributo crupier a *true* y el atributo baraja, además , si la persona que ha creado la partida decide unirse, se crea un jugador con 1000 monedas y apuesta 0, ambos tendrán la variable activo a *true.*
- **Fase de apuesta:** En esta fase a los jugadores se les habilitará los botones de Subir Apuesta y Apostar que podrán utilizar para tomar sus decisiones, mientras, el servidor queda en un estado de “esperando a las apuestas” que periódicamente irá comprobando si todos los jugadores hayan apostado. Cuando todos los jugadores hayan apostado el servidor pasará a la fase de juego. A partir de esta fase todos los nuevos jugadores que se unan quedarán inactivos hasta que acabe la ronda, además, si cualquier jugador abandona la partida a mitad de ronda, este perderá su progreso y el servidor eliminará dicho jugador de la lista para seguir el juego sin problemas.
- **Turno de los jugadores:** tan pronto termine la fase anterior el servidor pasará el control al jugador con índice más bajo que esté activo, este jugador tendrá habilitados únicamente los botones de Pedir Carta y Plantarse, una vez este haya terminado su turno mediante el botón de Plantarse, el servidor actualiza las variables jugadorActual e indiceJugadorActual y dará el control al siguiente jugador, esto se repetirá hasta llegar el turno del crupier en cuyo caso el servidor pasará al siguiente estado, turno del crupier.
- **Turno del crupier:** en esta fase se lanzará la función de la clase Jugador turnoCrupier donde el crupier llevará a cabo su jugada y una vez termine, pasará a la siguiente fase calcular resultado.
- **Calcular resultado:** una vez todos los jugadores y el crupier tienen una puntuación, se les asignará a los jugadores que tengan mejor puntuación que el crupier el doble de la cantidad apostada, los jugadores que hayan superado 21 o tengan menos puntos que el crupier perderán el dinero apostado y los que hayan obtenido una puntuación similar a la del crupier recibirán un reembolso de la cantidad apostada, al

acabar esta fase el servidor continuará de forma automatizada a la fase de Ronda completa.

- **Ronda completa:** Durante esta fase se habilita el botón de Siguiente Ronda y el servidor queda a la espera de que todos los jugadores activos usen dicho botón o abandonen la partida, los jugadores cuyo dinero en el marcador sea 0 quedarán inactivos de forma indefinida hasta que decidan abandonar. Una vez todos los jugadores estén listos comienza otra vez la fase de apuestas.

4.2.6 Clase GestorRed

Esta clase sobrescribe los métodos de la clase *NetworkManager* de Unity que es la que se ocupa de entre otras cosas crear el servidor y unir clientes al mismo. Su función principal es la de complementar a las funciones que ya tenía integradas. Dispone de 2 atributos principales:

- **maxJugadores**: este atributo identifica el número máximo de clientes que puede haber a la vez conectados en un mismo servidor, este atributo está definido como *const* lo que significa que no puede ser modificado y su valor por defecto es 4.
- **jugadores**: array de tipo Jugador que representa los jugadores conectados.

Las funciones de esta clase implementa la conectividad inicial cliente-servidor. Dichas funciones son las siguientes:

- **OnServerAddPlayer**: función que asigna al jugador un objeto tipo Jugador.
- **OnServerRemovePlayer**: elimina al jugador del array y de la conexión.
- **OnServerDisconnect**: hace uso de la función anterior y elimina el objeto jugador vinculado al jugador saliente.

5. Implementación

En el siguiente apartado se detalla el funcionamiento del entorno empleado para y su implementación en base a la fase de diseño.

5.1 Unity

El componente más fundamental de Unity son las escenas, las escenas representan las pantallas del juego, aquí es donde el jugador realizará sus acciones y donde el desarrollador pondrá todos los componentes para formar toda la parte visual del juego. Antes de nada, los desarrolladores deben crear una escena vacía, esta escena vacía es para el desarrollador lo que un lienzo blanco es para un pintor. Normalmente, los juegos presentan más de una escena de juego por lo que el desarrollador deberá tratar de conectar las distintas escenas del juego para dotar de coherencia al mismo. El editor de Unity soporta edición multiescena lo que significa que se puede trabajar sobre dos escenas a la vez para facilitar su coherencia.

El siguiente componente fundamental son los *GameObjects*, dentro de una escena se pueden crear una los *GameObjects*. Cada objeto en tu juego es un *GameObject* . Esto significa que todo lo que se te ocurra en tu juego tiene que ser un *GameObject* . Sin embargo, un *GameObject* no puede hacer nada por sí solo, debe tener asignado propiedades antes de que pueda convertirse en un personaje, un entorno o un efecto especial.

Los *GameObject* son contenedores a los cuales se les puede agregar piezas para convertirlo en un personaje, una luz, un árbol, un sonido o cualquier otra cosa que se pretenda desarrollar. Cada pieza agregada se denomina componente y según el tipo de objeto que se quiera crear, se agregan diferentes combinaciones de componentes a un *GameObject* . Puedes pensar en un *GameObject* como una olla de cocina vacía, y los componentes como diferentes ingredientes que conforman tu receta de juego. Unity tiene muchos tipos de componentes integrados, y también es capaz de crear componentes personalizados utilizando la API de *Unity Scripting*.

Existen una gran variedad de componentes pero los que se van a utilizar principalmente en la implementación del proyecto van a ser los siguientes:

- *Image:* Asocia al *GameObject* una imagen en forma de *sprite* para que este lo muestre dentro de la escena.
- *Text:* Este componente muestra en la escena una caja con texto, es muy similar a los componentes *label* de otros entornos de programación.
- *Script:* Asocia una clase a un objeto, esta clase puede estar compuesta de varios objetos que están representados en la escena de Unity, de esta forma se consigue que los distintos objetos puedan interactuar entre ellos.

- *Button:* Añade un botón a la escena, el cual se puede configurar para que realice una acción al ser pulsado
- *Grid Layout Group:* Este componente permite agrupar los diferentes GameObjects asociados a este en forma de cuadrícula.
- *Network Identity:* Este componente se utiliza para dotar a los objetos de propiedades de red, si un objeto sin este componente utiliza una función de red, esta no funcionará.

Unity soporta diferentes lenguajes de programación para la realización de los *scripts*, el lenguaje que se va a utilizar para el desarrollo es C# aunque este presenta diferentes características predefinidas para su desarrollo en Unity.

Todos los scripts de Unity deben heredar obligatoriamente de *MonoBehaviour* o de una clase que ya este heredando del mismo. Esta clase implementa una gran cantidad de funciones para su uso sobre los *GameObjects* y son un aspecto clave del entorno para su funcionamiento. Otro aspecto importante a destacar es que Unity utiliza un flujo de ejecución para todos sus *GameObjects.* La clase *Monobehaviour* define funciones que se llevan a cabo en diferentes partes de dicho flujo, como la función *Start(),* que se ejecuta cuando se crea un objeto con un *script* asociado.

Para el proyecto se va a utilizar la clase *NetworkBehaviour,* esta clase implementa distintas funciones de red para añadir jugadores a la partida y definir los comportamientos de cliente y servidor para poder realizar conexiones en red.

5.2 Implementación de la interfaz

Usando el editor de Unity y el modelo conceptual visto anteriormente en la fase de diseño, se va a realizar su implementación mediante el editor de Unity usando los componentes mencionadas anteriormente. El proyecto está formado por dos pantallas principales las cuales representan dos escenas, la escena Inicio y la escena Blackjack.

5.2.1 Escena Inicio

Inicio supone la primera de las escenas de las cuales está formado el proyecto y debería tener el aspecto mostrado por la Figura 11 en la fase de diseño, su jerarquía en *Unity* sería la siguiente:

Figura 19: Componentes utilizados en la pantalla de inicio

Como se puede observar en la Figura 19, la pantalla de inicio se compone de la cámara principal, que enfoca donde se sitúa el visionado de la pantalla, el canvas, que tiene asignado un objeto llamada ImagenPortada con el componente *Image* permitiendo mostrar el fondo de la pantalla de inicio, y el network manager, que utiliza la interfaz de red proporcionada por Unity para dar soporte a sus conexiones.

5.2.2 Escena *Blackjack*

Figura 20: Componentes utilizados en la escena *Blackjack*

La escena de *Blackjack* dispone de la arquitectura mostrada en la Figura 21 y su objetivo es implementar el diseño de la Figura 18, el objeto tablero tiene asignada el componente *Image* para el fondo de la pantalla. La escena se compone de los objetos tipo PanelJugador, el texto informativo, los botones y el marcador. Los paneles representan las manos de cada jugador y tienen asignado el componente *Grid Layout Group* para organizar las imágenes de las cartas asociadas al panel. El texto informativo únicamente tiene el componente *Text* que irá siendo modificado conforme avance la partida. Los botones simplemente tendrán asignados el componente *Button* y el marcador estaría compuesto por un conjunto de objetos tipo *Text.*

El siguiente objeto a destacar de la interfaz son los paneles del crupier y los jugadores, todos ellos muy similares entre sí que presentan la siguiente jerarquía:

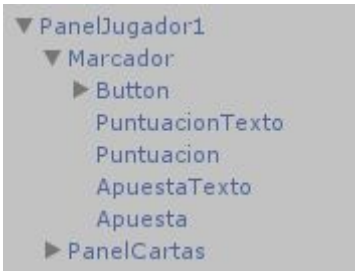

Figura 21: Esquema que muestra los objetos del panel del jugador

En la jerarquía se puede apreciar que los paneles de los jugadores están divididos en dos partes clave, la primera es el marcador, donde se muestra el número del jugador mediante un *Button deshabilitado* y la puntuación actual y la apuesta del mismo con componentes *Text.* La segunda parte es un panel donde se irán mostrando las cartas que el jugador vaya recibiendo, mediante un conjunto de imágenes obtenidas a partir de la función *slice* para obtener los diferentes sprites de las cartas a partir de la Figura .

5.3 Implementación de las *scripts*

Siguiendo las pautas del apartado de diseño de clases se programan las *scripts* correspondientes a las clases en C#, que a su vez deben cumplir el diagrama de clases de la Figura 10.

Las clases GestorRed, Jugador y GestorTurnos a diferencia de las otras *scripts,* heredan de *NetworkBehaviour*, pues son las encargadas de gestionar la conexión de los jugadores y su comunicación con el servidor.

La mayoría de las *scripts* están implementadas junto a los objetos de la interfaz, la única excepción es la clase Baraja la cual simplemente se utilizará para dar soporte a las funciones de la clase GestorTurnos.

La *script* Carta está asociada a las cartas del panel de los jugadores, todas las cartas estarán inicializadas con el sprite transparente y su índice a 99 para representar la mano de cartas vacía.

La clase PanelJugador está configurada siguiendo la Figura 22 en cada panel, se utiliza para cada unos de los paneles de los jugadore y el crupier, tiene asignados los textos de puntuación y apuesta del marcador del jugador que le corresponda

Figura 22: Configuración de la clase PanelJugador para cada jugador.

Mientras que el PanelJugador supone la representación visual del jugador, la *script* Jugador no tiene representación en la interfaz pero sirve para identificar a los jugadores que se unen a la partida. La clase Jugador está implementada en el editor de Unity mediante un objeto genérico que se crea al unirse un nuevo jugador y se destruye al desconectarse. Su configuración se muestra en la Figura 23.

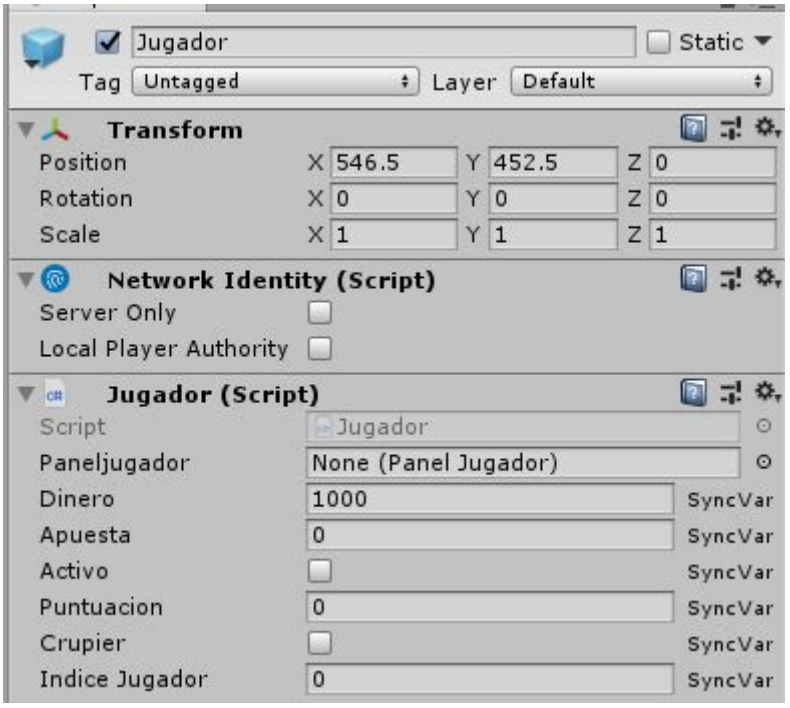

Figura 23: Objeto genérico Jugador.

La clase que gestiona las conexiones de los jugadores GestorRed, se encuentra asignada al objeto NetworkManager de la pantalla de inicio, este tiene asignado ambas escenas, ya que se encarga de cargar la escena correspondiente según las decisiones del jugador.

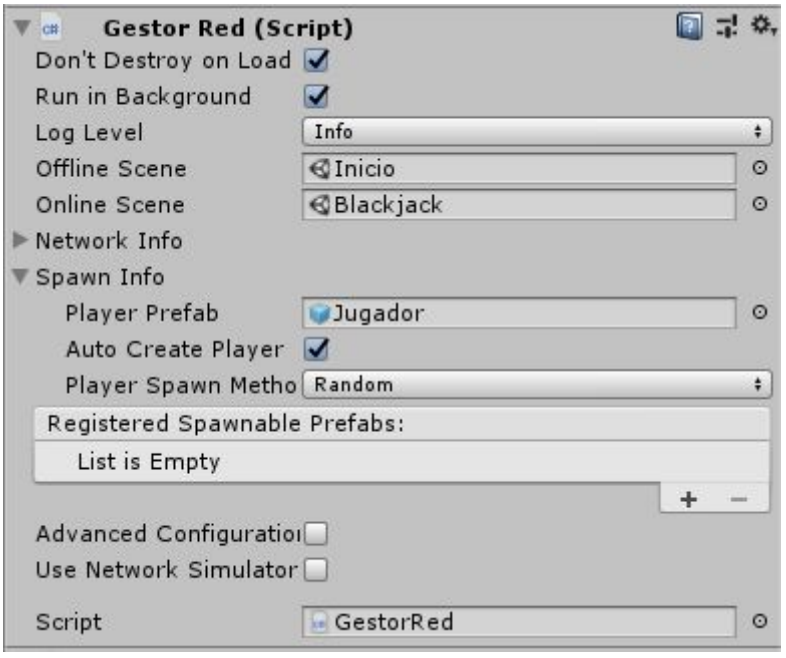

Figura 24: Configuración de la clase .

La *script* más importante y de mayor volumen es GestorTurnos que se encarga de llevar todo el flujo del juego, está asociada como componente a GestorJugadores y tiene asignados en sus atributos los textos del marcador, el panel informativo, los paneles de los

jugadores, la *script* baraja y los botones de la interfaz, como se muestra en la Figura 25. Aunque la interfaz de los jugadores esté asignada a un único *script* los jugadores verán interfaces diferentes puesto que GestorTurnos asigna a cada jugador su valor correspondiente en el marcador y habilita los botones en función del estado y del jugador actual.

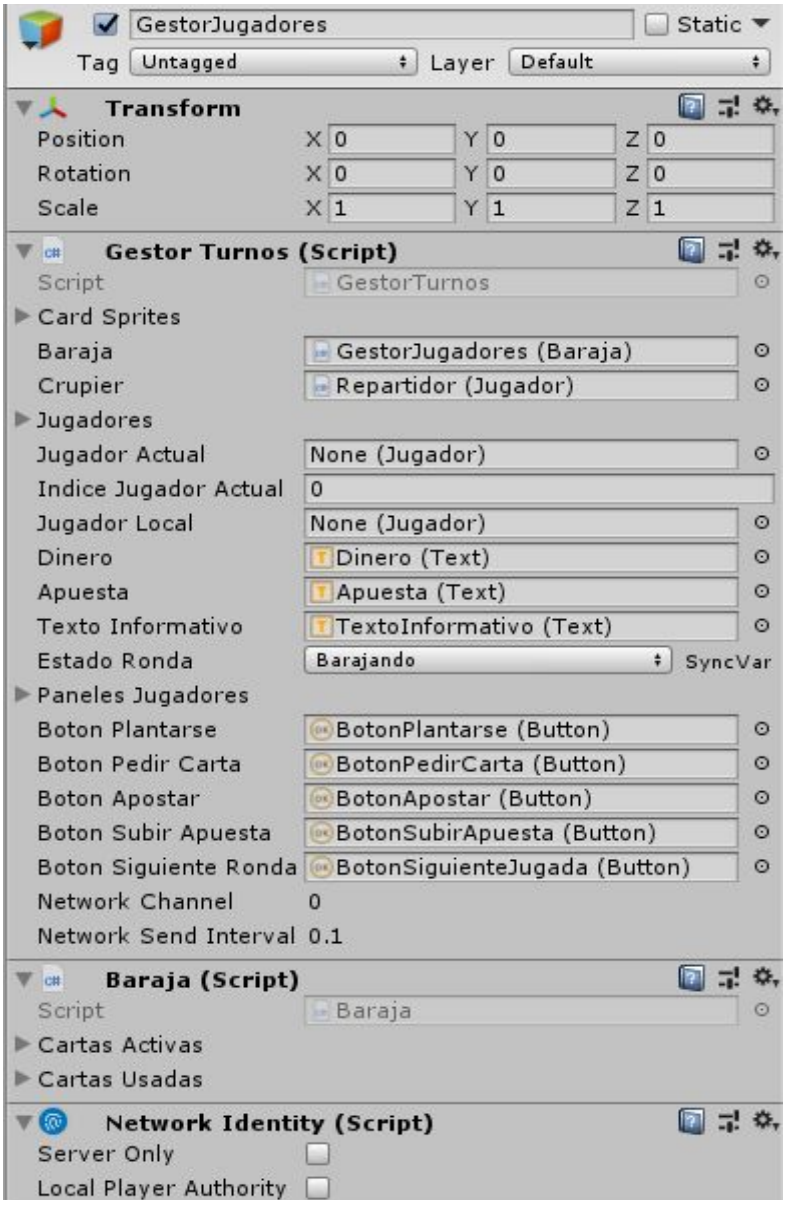

Figura 25: Objeto GestorJugadores configurado en Unity con la script GestorTurnos

6. Pruebas

Una vez se ha realizado el diseño y la implementación del proyecto es necesario realizar una fase de pruebas para que el producto cumpla con los requisitos que se han establecido en la fase de análisis.

Para la realización de las pruebas se han llevado a cabo diferentes *scripts* de prueba utilizando la tecnología NUnit que incorpora Unity. NUnit, es un framework integrado en Unity para dar soporte a pruebas, soporta todos los lenguajes disponibles en Unity. Para la realización de las pruebas Android, se ha utilizado un emulador instalado en una computadora de las siguientes características:

- Procesador: Intel i5 6500
- Memoria RAM: 16 GB
- Disco duro: SSD 250/ HDD 1TB
- Tarjeta gráfica: Gigabyte GeForce GTX 1060 6GB GDDR5

Las *scripts* de prueba se han realizado usando el editor de Visual Studio en el lenguaje de programación C#, que es el mismo lenguaje usado para su implementación. Las *scripts* utilizadas son las siguientes:

- **crearPartida**: esta prueba simplemente crea una nueva partida sin realizar ninguna función extra inicializa los jugadores y el gestor de turnos para comprobar que funciona correctamente.

- **ronda1jugador**: realiza una ronda de juego creando el servidor y un jugador, las órdenes se realizan a través de los comandos para imitar el comportamiento de los botones en la interfaz.

- **rondaXjugadores**: consiste en tres pruebas diferentes con X siendo el número de jugadores de la partida, realiza tres rondas consecutivas conectando un jugador nuevo al principio de cada ronda, de esta forma se evita que dichos jugadores tengan que esperar.

- **rondaConConexion**: esta prueba conecta distintos jugadores en diferentes momentos de la ronda ya comenzada para forzar la espera de los mismos y comprobar que un jugador nuevo no tendría ningún impacto en el juego hasta acabar la ronda.

- **rondaConDesconexion**: empieza la ronda con cuatro jugadores y los va desconectando en distintos momentos de la ronda para comprobar que no se produce ninguna excepción no tratada que pueda comprometer el funcionamiento de la aplicación.

- **finPartida**: esta prueba simula el final de una partida donde todos los jugadores han abandonado o todos los jugadores han perdido su dinero, forzando el servidor a cerrarse en caso de que el jugador propietario abandone el juego.

Utilizando la herramienta de gestión de pruebas de Unity se puede comprobar que el resultado ha sido positivo para todas las pruebas como se puede apreciar en la Figura 26.

Figura 26: Ventana del *Test Runner* de Unity donde se muestra el resultado de las pruebas.

Posteriormente a las pruebas realizadas usando *scripts* se han realizado pruebas manuales para comprobar el funcionamiento de la aplicación en un dispositivo Android real, en concreto, se ha utilizado el dispositivo: Xiaomi Pocophone F1 - Smartphone de 6.18" , 64GB y 4GB de RAM. Para ello se ha construido la aplicación para Android y se ha creado su .apk. La prueba ha tenido un resultado favorable y lo único que ha tenido que corregirse han sido algunos fallos visuales donde la interfaz no se ajustaba totalmente a la pantalla del dispositivo Android, una vez solucionado, se ha probado de nuevo con resultados exitosos.

7. Futuras mejoras

Para finalizar, debido a la falta de recursos, existen alguna ideas que se habían considerado inicialmente y no han llegado a ser implementadas, las siguientes ideas son las principales que no han podido llevarse a cabo:

Muchas de las aplicaciones de Android disponen de sistema de micro transacciones de las cuales se puede obtener beneficios económicos. Inicialmente, se había considerado establecer un sistema de micro transacciones para que mediante el juego los jugadores pudieran apostar dinero real y obtener beneficio, pero debido a que para una funcionalidad de este calibre requiere primero de una inversión económica considerable, se ha decidido descartar la idea. Para abrir una aplicación monetizada donde los jugadores puedan ganar dinero primero hay que tener una forma de pagarles, lo cual significa invertir cierta cantidad para que el sistema de monetización funcione y actualmente este proyecto no dispone de dichos recursos. Esta idea también supondría crear un sistema de cuentas con autentificación, seguridad en la aplicación y un sistema de monetización interna en la aplicación para que pudiera funcionar.

En un inicio, se había considerado la opción de publicar en Google Play la aplicación para que cualquier persona pudiera descargarla pero debido en parte a la complejidad añadida en la implementación del servicio de Google Play y a que una cuenta de desarrollador requiere de un pago inicial, esta opción se ha descartado por motivos muy similares que la propuesta anterior.

8. Conclusión

Este proyecto ha planteado una gran cantidad de retos, la selección del juego y el *framework* ha sido una de las etapas más importantes de proyecto, han condicionada el proyecto desde su principio pero a su vez ha proporcionado una pista para poder comenzar su desarrollo.

El análisis de requisitos ha sido de gran ayuda para acotar el problema y posteriormente dar una solución al mismo. La interfaz ha requerido de varios diseños para hallar la más adecuada, pues la usabilidad es un factor muy importante en los juegos de Android.

Unity ha sido otro de los grandes retos, un nuevo entorno requiere de aprendizaje y práctica, nunca había usado Unity y ha sido una experiencia interesante, la comunidad de Unity ha sido de gran ayuda para resolver una gran cantidad de problemas que se han ido presentando durante su implementación, en esta fase la parte más complicada ha sido la implementación de la arquitectura cliente-servidor para que pudiera funcionar como es debido. Una de las partes importantes a destacar es que Unity pese a ser un *framework* que no se ha estudiado durante el grado, está muy orientado a objetos como Java, que es uno de los lenguajes que se estudia en las asignaturas de Introducción a la programación y me ha ayudado sobremanera a entender el funcionamiento de Unity y de C#

En definitiva, el juego ha podido llevarse a cabo aprendiendo nuevas tecnologías pero haciendo uso de estructuras de datos conocidas, este proyecto ha sido una interesante forma de aplicar los conocimientos aprendidos y utilizar estos mismas para ampliar el campo de conocimiento.

9. Anexo

9.1 Manual de Usuario de la aplicación

La aplicación se inicia una vez instalado el .apk en Android y se mostrará la siguiente pantalla:

Figura 27: Pantalla de Inicio

Esta es la pantalla de inicio de la aplicación (Figura 27) que dispone de las siguiente funcionalidades:

- **Botón Crear Partida**: Crea una nueva partida utilizando tu dirección IP y puerto 7777 por defecto, una vez creada el jugador se unirá como Jugador1 a la partida y esta podrá dar comienzo con la fase de apuestas.
- **Botón Unirse a Partida**: El jugador se conecta a la dirección IP y puerto especificado en la caja de texto para unirse a una partida que no exceda el número de jugadores permitido, el jugador puede unirse en cualquier momento de la partida pero solo podrá jugar una vez llegue la fase de apuestas.
- **Caja de Texto**: Esta caja de texto sirve para introducir la dirección IP a la que se desea conectar el formato de la dirección IP es por ejemplo, 127.0.0.1:7777.

Una vez el jugador se una a partida por cualquiera de los métodos anteriores, se pasará a la pantalla de Juego.

Figura 28: Pantalla de Juego

Una vez el jugador accede a la pantalla de juego (Figura 28) el jugador se puede unir en cualquier fase del juego pero no podrá jugar hasta la fase de apuesta, en la fase de apuesta se habilitarán dos botones:

- **Botón Subir Apuesta**: El jugador subira en 50 la apuesta cad avez que pulse este botón, el jugador no podrá apostar más dinero del que dispone.
- **Botón Apostar**: El jugador confirma su apuesta, este botón no tendrá ningún efecto si no se ha subido la apuesta por lo menos una vez.

Al acabar la fase de apuesta comenzará la fase de juego donde se repartirán dos cartas a cada jugador y comenzarán uno por uno a pedir cartas para obtener la mayor puntuación posible, durante el turno del jugador se habilitan los siguientes botones:

- **Botón Pedir Carta**: pide al crupier una carta y actualiza la puntuación, si se obtiene una puntuación de 21 o superior a 21, el jugador se verá forzado a plantarse, obteniendo la mejor puntuación en caso de ser 21 y perdiendo la ronda en caso de ser superior a 21.
- **Botón Pantarse**: confirma la puntuación actual, para plantarse voluntariamente se debe tener una puntuación inferior a 21.

Sea cual sea el resultado de la fase anterior, una vez todos los jugadores realicen su turno, será el turno del crupier, una vez este acabe su turno, se comparan todas la puntuaciones con las del crupier y se deciden los ganadores, una vez acaba la ronda los jugadores tendrán disponible otro botón:

- **Botón Siguiente Ronda**: este botón confirma que el jugador quiere seguir jugando y una vez todos los jugadores hayan aceptado o abandonado la partida se procederá nuevamente con la fase de apuestas.

Durante toda la ronda también habrá un botón habilitado que es el botón de desconexión **stop,** una vez pulsado el jugador abandonará la partida y si estaba apostando perderá la apuesta.

10. Referencias

- Google Play, juego Zynga Poker-Texas Holdem:[Internet] último acceso 10 junio 2019 https://play.google.com/store/apps/details?id=com.zynga.livepoker
- Google Play, juego UNO!™ [Internet] último acceso 10 junio 2019 https://play.google.com/store/apps/details?id=com.matteljv.uno
- Google Play, juego Hearthstone [Internet] último acceso 11 junio 2019 https://play.google.com/store/apps/details?id=com.blizzard.wtcg.hearthstone
- Página oficial de Unity: [Internet] último acceso 12 junio 2019 https://unity.com/es
- Página oficial de GameMaker Studio: [Internet] último acceso 14 junio 2019 https://www.yoyogames.com/gamemaker
- Página oficial de Unreal Engine: [Internet] último acceso 14 junio 2019 https://www.unrealengine.com/en-US/
- Página oficial de AppGameKit: [Internet] último acceso 15 junio 2019 https://www.appgamekit.com/
- Manual de Usuario, Unity: [Internet] último acceso 28 junio 2019 https://docs.unity3d.com/es/current/Manual/UnityManual.html
- Documentación de NUnit: [Internet] último acceso 18 junio 2019 https://nunit.org/docs/2.6.3/docHome.html
- Imagenes Baraja de *poker* [Internet] último acceso 18 junio 2019 http://www.thehouseofcards.com/cards/card-images.html
- 2019 Video Game Industry Statistics, Trends & Data [Internet] WePC último acceso 19 junio 2019 https://www.wepc.com/news/video-game-statistics/
- 10 best card games for Android [Internet] Joe Hindy ,Android Authority último acceso 29 junio 2019 https://www.androidauthority.com/best-android-card-games-581095/
- How to Create a Multiplayer Game in Unity [Internet] Renan Oliveira, gamedevacademy.org último acceso 30 junio 2019 https://gamedevacademy.org/how-to-create-a-multiplayer-game-in-unity/
- Página oficial StarUML [Internet] último acceso 1 julio 2019 http://staruml.io/
- Jared Halpern, Developing 2D Games with Unity: Independent Game Programming with C#. Apress, 2018.