11-4-2020

**PROYECTOS INFORMÁTICOS II**

**MEMORIA DEL PROYECTO** EJECUCIÓN Y CONTROL  
 **MONOPOLY**

DANIEL – AYOUB – RAÚL

**ÍNDICE**

[1. Control de documentación 2](#_Toc38725022)

[2. Historial de modificaciones 2](#_Toc38725023)

[3. Fase de ejecución del proyecto 3](#_Toc38725024)

[3.1 Definición del modelo de desarrollo 3](#_Toc38725025)

[3.2 Definición del entorno de trabajo 4](#_Toc38725026)

[3.3 Análisis de requisitos 5](#_Toc38725027)

[3.4 Control de bugs 0](#_Toc38725028)

[3.5 Diagrama de clases 0](#_Toc38725029)

[3.6 Diagrama de componentes 1](#_Toc38725030)

[3.7 Implementación del UML 2](#_Toc38725031)

[3.7.1 Clase Jugador 2](#_Toc38725032)

[3.7.2 Clase Humano 3](#_Toc38725033)

[3.7.3 Clase Dado 4](#_Toc38725034)

[3.7.4 Clase DatosPartida 4](#_Toc38725035)

[3.8 Control de versiones 4](#_Toc38725036)

[4. Fase de control del proyecto 5](#_Toc38725037)

[4.1 Responsabilidades relacionadas con la calidad 6](#_Toc38725038)

[4.2 Procesos de calidad en el proyecto 7](#_Toc38725039)

# Control de documentación

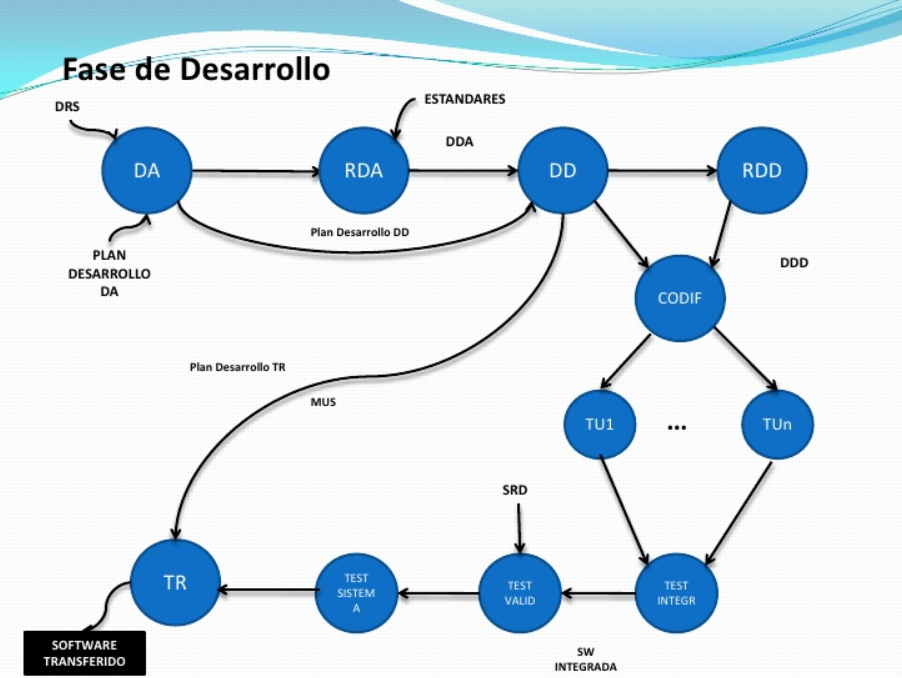
|  |  |  |
| --- | --- | --- |
| **Elaborado por:** | **Revisado por:** | **Aprobado por:** |
| Alumnos de la asignatura (desarrollo) | Alumnos de la asignatura (revisado) | Equipo de la Universidad |
| **Nombre:** | **Nombre:** | **Nombre:** |
| Daniel Ortega  Ayoub Dachour  Raúl García | Daniel Ortega  Ayoub Dachour  Raúl García | [Jordi José](https://opencampus.uols.org/user/view.php?id=1347&course=872) |
| **Fecha:** | **Fecha:** | **Fecha:** |
| 08/05/2020 |  |  |
| **Firma:** | **Firma:** | **Firma:** |
|  |  |  |

# Historial de modificaciones

|  |  |  |  |
| --- | --- | --- | --- |
| **VERSIÓN** | **FECHA DE VIGENCIA** | **MODIFICADO POR:** | **MODIFICACIÓN REALIZADA** |
| 1.0 | 27/03/2020 | Daniel Ortega | Creación de documento |
| 1.1 | 27/03/2020 | Daniel Ortega | -Recuperar casos de uso y requisitos  -Planificación detallada del proyecto |
| 1.2 | 08/04/2020 | Raúl García | - Integrar partes que faltan de la tarea anterior  - Incluir diagrama de clases |
| 1.3 | 09/04/2020 | Raúl García | - Repaso apartado 3.  - Añadir apartado 6 “Fase de control del proyecto” |
| 1.4 | 10/04/2020 | Raúl García | - Ajuste formato  - Añadir contenido apartado 5. Descripción de clases  - Añadir contenido apartado 6 |
| 1.5 | 11/04/2020 | Raúl García | - Añadir contenido apartado 6 |
| 1.6 | 11/04/2020 | Daniel Ortega | - Ajuste formato  - Adaptar fase control y calidad |
| 1.7 | 25/04/2020 | Daniel Ortega | -Se divide documento en DOS. Este queda únicamente para ejecución y control -Añadida columna test en requisitos y casos de uso para controlar si se ha testeado -Añadido apartado bugs y descripción clases humano y todas las de utilidades |

# Fase de ejecución del proyecto

La fase de ejecución consiste en la coordinación de personas y otros recursos para llevar a cabo el plan. Es el lugar donde se realiza el trabajo real. El noventa por ciento o más de los esfuerzos del proyecto se gastan durante esta fase, y se completa cuando se cumple la meta del proyecto. La fase de ejecución consta de las siguientes acciones: desarrollo del código, creación de casos de prueba y el establecimiento de la documentación del usuario.

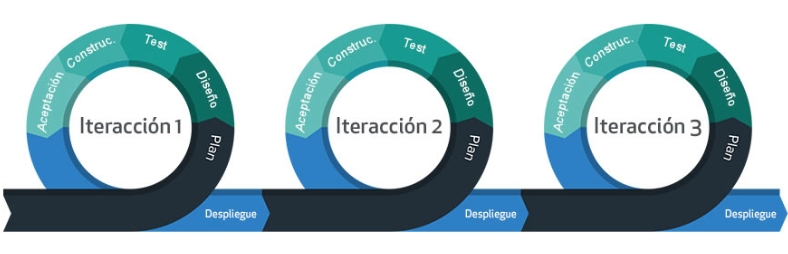


# Definición del modelo de desarrollo

El modelo de desarrollo será de forma ágil, es decir, por Sprint. Algunas de las ventajas de esta metodología son las siguientes:

* Se trabaja con mayor velocidad y eficiencia, ya que se trabaja en entregas parciales del producto.
* Ahorro de tiempo y costes, ya que al ser un desarrollo más eficiente y rápido, el tiempo invertido es menor
* Mejora la motivación e implicación del equipo de desarrollo
* Mejoran la satisfacción del cliente ya que interactúan constantemente con los nuevos desarrollos
* No se desarrollan características innecesarias, solo lo que el cliente necesita y establecido por prioridad
* Alertar de forma rápida tanto de errores o problemas ya que se hace test y validación constante

Esta figura muestra como cada iteración será una entrega parcial y de dicha entrega se realizarán todas las fases correspondientes. En el caso tradicional, el test se realizaría al terminar el proyecto, cosa que no creemos conveniente.



# Definición del entorno de trabajo

Por las garantías que ofrece, hemos seleccionado el IDE **Netbeans**. Como ya se ha comentado en la reunión Kickoff, algunas de las garantías de este IDE son:

**Ventajas de Netbeans**

* Lenguaje Multi-Plataforma: Código escrito en java es leído por un intérprete, por lo que funcionará en cualquier plataforma
* Manejo automático de memoria
* Gratuito
* Reutilización de módulos
* Incluye templates y Wizards
* Instalación y actualización simple

Además, también se utilizará **GitHub** para compartir el código fuente con todo el equipo y con el objetivo de trabajar sobre versiones de software.

# Análisis de requisitos

A modo de recordatorio, se dejarán todos los requisitos analizados con los stakeholder a tener en cuenta por el grupo de desarrollo en esta fase de ejecución, ya que se deberán aplicar.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Nombre** | **Tipo** | **Prioridad** | **Capa** | **Aplicado** | **Test** |
| Programación MVC | No funcional | Alta | Todas |  |  |
| Programación siguiendo buenas prácticas (comentarios, estructura…) | No funcional | Alta | Todas |  |  |
| Soporte de 2 a 8 jugadores | Funcional | Alta | Controlador |  |  |
| Incluir Robots con una IA simplista | Funcional | Baja | Controlador | No se incluye | |
| Deberá soportar importación de partidas mediante ficheros | Funcional | Alta | Modelo |  |  |
| Deberá soportar guardado de partidas mediante ficheros | Funcional | Alta | Modelo |  |  |
| Deberá soportar carga de partidas mediante ficheros | Funcional | Alta | Modelo |  |  |
| El jugador puede crear una partida nueva | Funcional | Alta | Controlador |  |  |
| Se deberá aportar el código fuente del programa | No funcional | Alta | Todas |  |  |
| Consola de texto para visualizar la información de las acciones | Funcional | Alta | Vista |  |  |
| El jugador debe poder mover la ficha | Funcional | Alta | Controlador |  |  |
| El jugador debe poder tirar el dado | Funcional | Alta | Controlador |  |  |
| El jugador debe poder comprar una propiedad | Funcional | Alta | Controlador |  |  |
| El jugador debe poder comprar una propiedad | Funcional | Alta | Controlador |  |  |
| El orden de salida será aleatorio | Funcional | Alta | Controlador |  |  |
| El jugador deberá ver la interacción del juego (Tablero y consola de texto) | Funcional | Alta | Vista |  |  |
| Incluir cartas de suerte y comunidad | Funcional | Alta | Modelo |  |  |
| Jugador podrá visualizar cartas suerte y comunidad | Funcional | Alta | Vista |  |  |
| Sólo se podrá adquirir una propiedad por turno ya sea mediante la compra o la negociación con otro jugador. Las casillas no compradas no saldrán a subasta. | Funcional | Media | Controlador |  |  |
| Se usan dos dados para el juego | Funcional | Alta | Controlador |  |  |
| Un jugador pierde cuando entra en bancarrota y tiene un saldo negativo. | Funcional | Alta | Controlador |  |  |
| Los jugadores recibirán 200€ cada vez que pasen por la casilla de salida mientras queden propiedades sin dueño. | Funcional | Alta | Controlador |  |  |
| Si se sacan dobles se vuelve a tirar, si se sacan dobles 3 veces seguidas el jugador irá a la cárcel. | Funcional | Alta | Controlador |  |  |
| Cuando un jugador pierde, sus propiedades pasan al jugador con quien ha contraído la deuda o a la banca en caso que la deuda sea con ella (en este caso las casillas pasarán a estar en venta y las edificaciones se destruirán). | Funcional | Media | Controlador |  |  |
| Las multas las gestiona la banca creando un fondo de dinero que será obtenido por el primer jugador que caiga en parking gratuito. | Funcional | Media | Controlador |  |  |
| Cuando un jugador va a la cárcel se mueve a la casilla (11). Deberá estar tres turnos y puede tirar dados y salir con dobles o pagar 50€ de multa. Si está tres turnos en la cárcel deberá pagar 50€ de multa y tirar dados. (No se volverá a tirar aunque haya sacado dobles). | Funcional | Media | Controlador |  |  |
| El número total de casas es de 32 y el de hoteles 12. | Funcional | Media | Controlador |  |  |
| Cada jugador empieza con 1500€ | Funcional | Media | Modelo |  |  |

De igual forma, se irá detallando si se ha realizado cada caso de uso en la columna “Aplicado o comentarios”

La siguiente tabla pretende plasmar los casos de uso a desarrollar entrando más en detalle en las características de cada uno de ellos.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | Nombre | CU Rel. | Descripción | Trigger | Requisitos Previos | Consecuencias | Excepciones | Aplicado o comentarios | Test |
| 1 | Crear partida |  | Creará la partida con la configuración inicial seleccionada | El usuario selecciona la opción crear partida | La aplicación se ha de abrir y ejecutar correctamente | Se genera una nueva partida |  |  |  |
| 2 | Empezar partida | 1, 14 | Empezará la partida tras crearla o cargarla, en este último caso será desde la última acción realizada.  Si la partida es nueva, el usuario empieza con 1500 euros. | El usuario selecciona la opción empezar partida | La partida ha debido ser creada o cargada | Carga la nueva partida o una ya guardada previamente |  |  |  |
| 3 | Lanzar Dados | 2 | El usuario podrá lanzar los dados, que serán dos y obtendrá un resultado y en función de este podrá avanzar o se desbloquearán varias acciones. | El usuario hace clic en la opción lanzar dados | La partida tiene que estar empezada y todo lo que ello requiere | El usuario avanzará el número de casillas que resulte de tirar los dados. En caso de estar en la cárcel, dobles equivalen a salir de ella. |  |  |  |
| 4 | Mover ficha | 3 | En función del número obtenido en lanzar dados, se deberá mover X casillas la ficha | Al lanzar los dados, se llamará a esta función | El usuario ha debido de tirar los dados | Se moverán X casillas | Está en cárcel no moverá ficha a no ser que haya sacado dobles |  |  |
| 5 | Comprar propiedad | 3,4 | El usuario puede comprar una casa de un color si cae en la casilla correspondiente. El sistema indicará el precio de la propiedad y el usuario elegirá si comprar o no.  El usuario también podrá comprar una propiedad a otro usuario. Por lo que podría hacer una oferta. (relacionado con Caso de uso 8) | El usuario elegirá si comprar propiedad o no. | Se deberá verificar que el usuario tenga saldo para pagar. La partida ha tenido que ser empezada y los dados lanzados. |  | El usuario ya tiene la casa. Por lo que hay que pasar al caso de uso de pagar.  El usuario no tiene saldo |  |  |
| 6 | Elegir carta | 3,4 | El jugador podrá elegir dos cartas, de la suerte y de la comunidad. Sacan Estas cartas cuando se posan sobre las casillas correspondientes y se debe seguir las instrucciones que haya en ellas. | La ficha del jugador cae en una casilla que requiere coger carta | Que haya empezado la partida y el jugador pueda tirar los dados. Significa que no esté en bancarrota ni en la carcel | Se debe tomar una carta y se deberá restar una a la baraja |  |  |  |
| 7 | Pagar Banca | 3,4 | En el caso de caer en una casilla que requiera pagar multa, el jugador tendrá que pagar a la banca un importe X | Caer en casilla que requiera pagar | Que haya empezado la partida y el jugador pueda tirar los dados. Significa que no esté en bancarrota ni en la carcel | Se le restara un importe X al usuario |  |  |  |
| 8 | Oferta a jugador |  | El jugador puede hacer una oferta para comprar una propiedad a otro jugador. También existe la opción de intercambiar casas, por lo que esta oferta puede ser un cambio por otra propiedad o comprar por un importe a introducir | Usar la función de hacer oferta: El jugador deberá introducir la cantidad o la propiedad a intercambiar | Si la oferta es de dinero, verificará el importe del jugador. Si la oferta es de una casa, se deberá tener como mínimo una. Además de que el jugador no puede estar en bancarrota ni en la carcel | Si el jugador B acepta, quitarle la propiedad y/o el importe al jugador A |  |  |  |
| 9 | Vender propiedad |  | El jugador puede ofrecer la venta de su propiedad a otro jugador.  Se deberá introducir el importe por el que se desea vender. | Usar la función de vender propiedad: El jugador deberá seleccionar la propiedad y por cuanto dinero la desea vender | Verificar que la propiedad realmente exista y también que el jugador no esté en la cárcel o bancarrota | Quitarle la propiedad una vez la venta se haya realizado al jugador A y dárselo al jugador B |  |  |  |
| 10 | Ir a carcel | 3 | El jugador irá a la cárcel si cae en dicha casilla. Permanecerá en ella hasta que tirar dados sea dobles o se abone la cantidad de 50 euros | Caer en la casilla cárcel |  |  |  |  |  |
| 11 | Salir carcel | 10,7 | Tirar dados sea dobles, entonces el usuario podrá salir de la cartel en el siguiente turno. También podrá salir si abona 50 euros de multa (pagar a la banca, caso 7) | Tirada de dados dobles o pagar a la banca | El usuario debe estar ya en la cárcel |  |  |  |  |
| 12 | Casilla sailda | 3,4 | Al pasar por la casilla de salida, se le ingresará un importe X al usuario | Cuando se tiren dados y la ficha pase por dicha casilla |  |  |  |  |  |
| 13 | Pagar a jugador | 3,4 | Si el usuario cae en una propiedad de otro jugador. Este le deberá de pagar lo que corresponda | Cuando la ficha caiga en la casilla con una propiedad de otro jugador | La casilla debe tener la propiedad de otro jugador. | El jugador podrá hacer una oferta cambiándole una casa o tendrá que pagar.  La consecuencia mayor es que el jugador entrará en bancarrota y perderá la partida. |  |  |  |
| 14 | Guardar partida | 1,2 | La partida tendrá que estar empezada y el usuario podrá en cualquier momento elegir cuando guardar la partida. Todos los valores se guardarán en ficheros de texto externos que podrán ser cargados posteriormente. | Usuario selecciona la opción guardar partida | La partida debe estar empezada |  |  |  |  |
| 15 | Cargar partida | 13 | El usuario podrá hacer uso de la función cargar partida que obtendrá los valores guardados previamente | El usuario selecciona la opción cargar partida | Debe existir al menos una partida guardada | La partida actual será reemplazada por la nueva |  |  |  |
| 16 | Salir del juego | N/A | El usuario saldrá de la aplicación | El usuario selecciona la opción Salir | Debe estar el juego iniciado | Saldrá del juego completamente |  |  |  |
| 16 | Jugador pierde la partida | 7, 13 | Cuando entra en bancarrota o saldo negativo | Al entrar en bancarota | Tener un saldo superior a 0 y que pase a ser inferior a 0 |  |  |  |  |

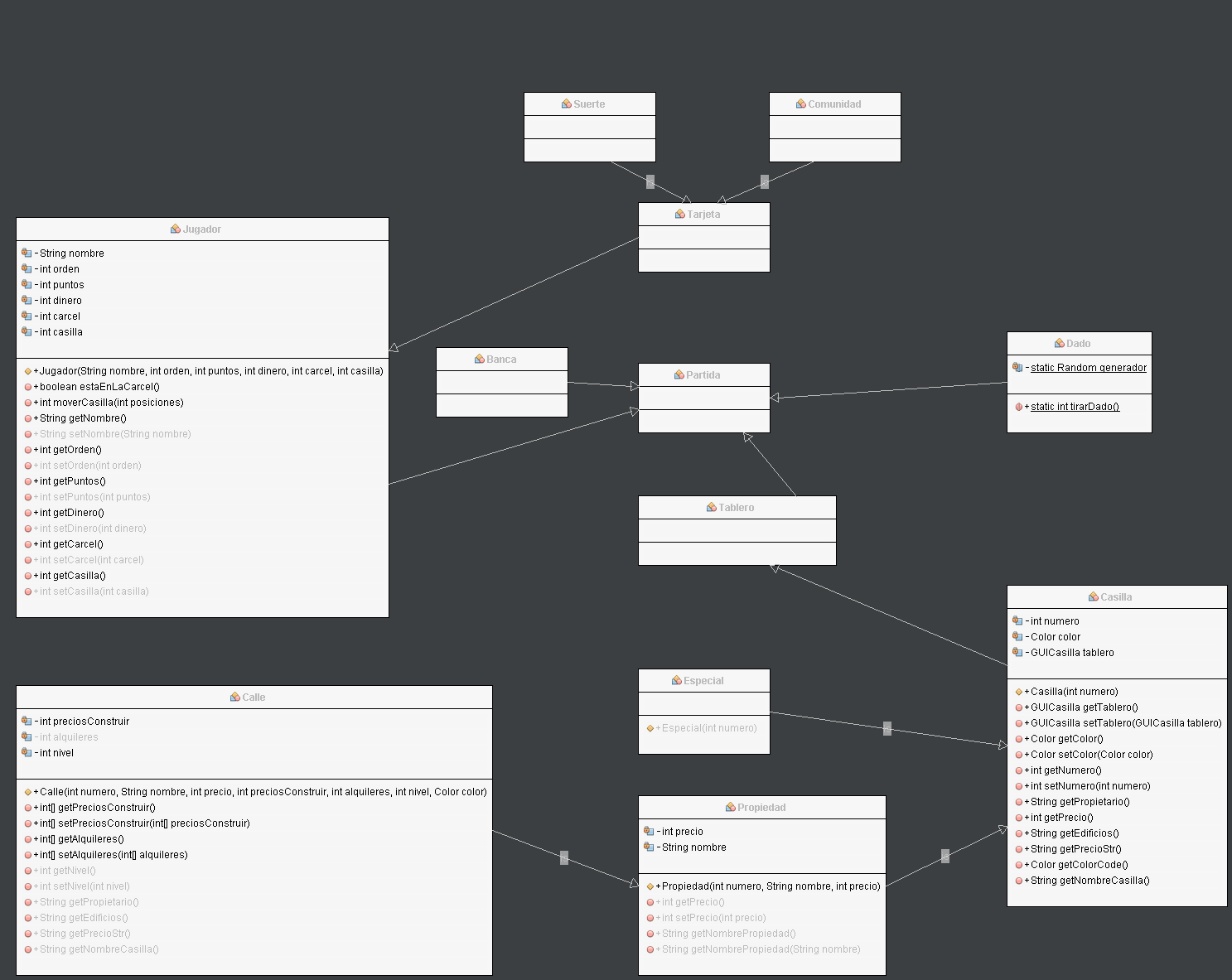
# Control de bugs

A continuación se define una tabla para tener el control de todos los bugs encontrados y disponer de una trazabilidad de si se ha arreglado o no.

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Fecha** | **Descripción** | **¿Arreglado?** |
| 1 | 21/04/20 | Salta excepción nullpointer al salir del juego. Casualmente después de guardar una partida |  |
| 2 | 21/04/20 | Al recuperar partida no respeta el turno del juego guardado |  |
| 3 | 22/04/20 | Desaparece a veces alguna de las fichas cuando se cae en la casilla 3 |  |
| 4 | 22/04/20 | A veces, al caer en la caja comunidad no saca carta |  |
| 5 | 23/04/20 | Al pasar por la casilla de salida, si nos sobra tirada la ficha se queda en la propia casilla salida. En la siguiente tirada de ese jugador se suma la tirada + la sobrante anterior |  |
| 6 | 24/04/20 | Cárcel se sale sin sacar dobles y a la primera vez de tirar el dado. |  |
| 7 | 25/04/20 | Opción salir. Revisar |  |

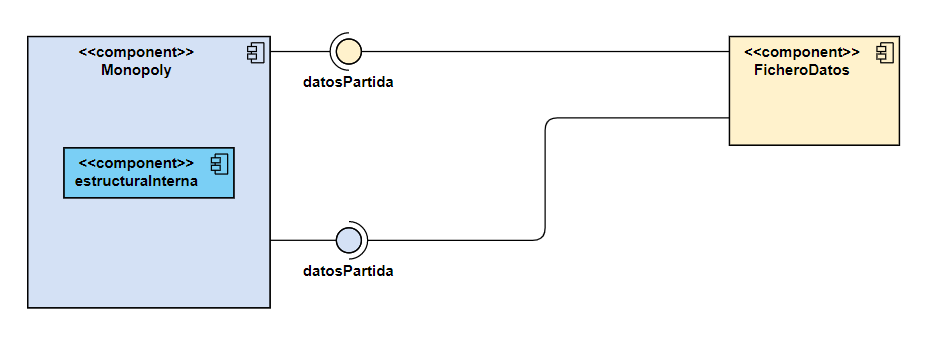
# Diagrama de clases

Se incluye el diagrama de clase que se irá evolucionando a lo largo del proyecto. Algunas clases se eliminarán y otras aparecerán según necesidad. Por ahora el diagrama es muy básico y simple. Se ha desarrollado con el pluging de Netbeans EasyUML por la comodidad que conlleva a la hora de llevar el diagrama a código Java.



# Diagrama de componentes

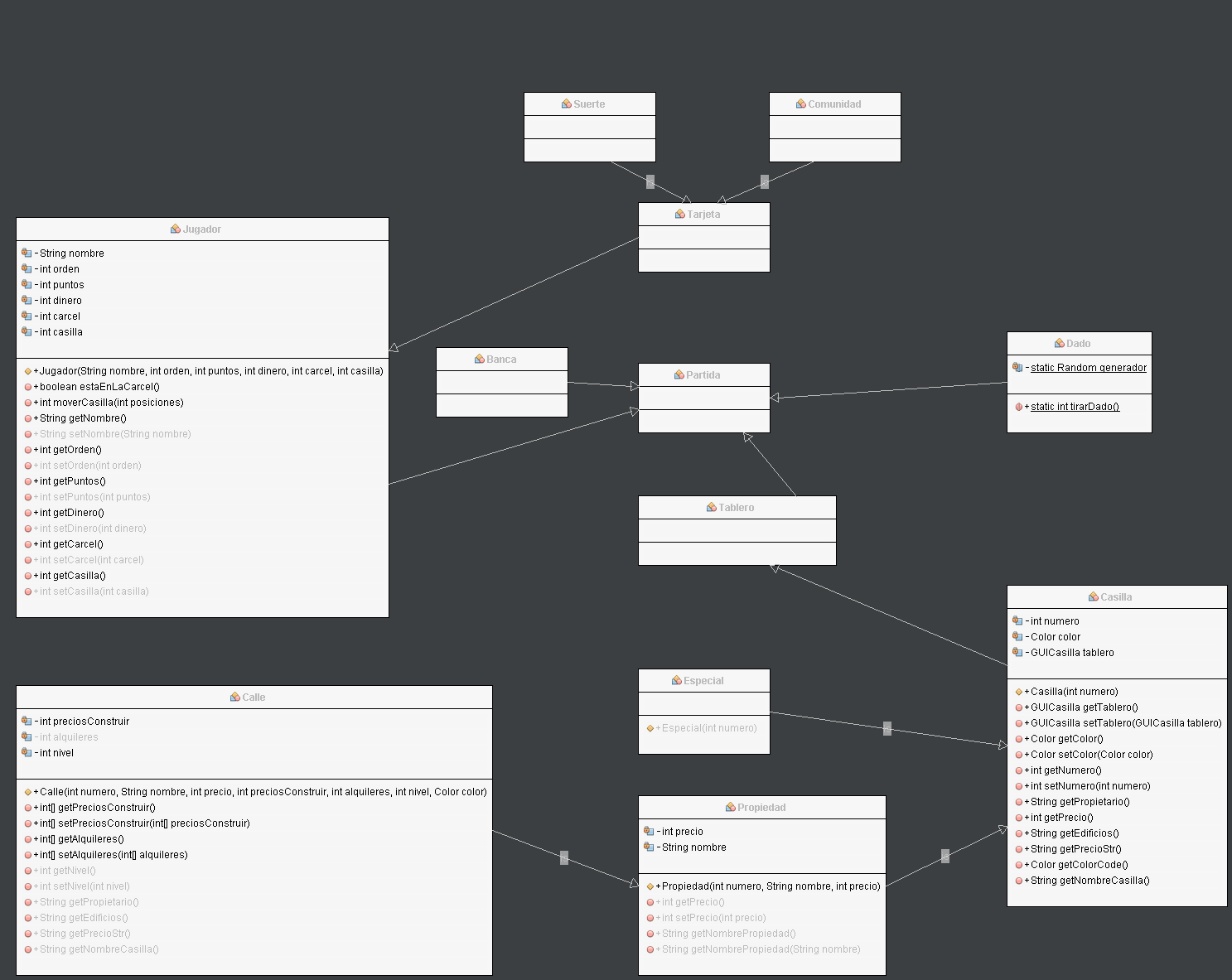
A continuación, adjuntamos el diagrama de componentes. Que aunque es muy básico nos ayudará por si en el futuro queremos ampliar funcionalidades de nuestra aplicación.



# Implementación del UML

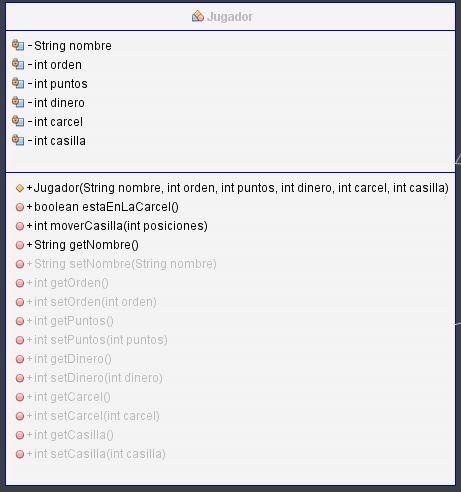
A continuación se ha a explicar de una forma general el diagrama de clases y cuáles de ellas forman la aplicación, para qué existen, sus atributos más importantes y métodos que la integran.

**Diagrama UML**:



# Clase Jugador

La clase jugador, es la clase que representa un jugador con sus atributos y sus métodos. Es una clase genérica por lo que hará de superclase de dos subclases más especializadas.



**Atributos**:

* **nombre** (String): es el nombre que tendrá el jugador, tanto si es humano como computador.
* **orden** (int): es el orden de juego que tiene el jugador asignado. Este orden se asigna de una forma aleatoria.
* **puntos** (int): son los puntos que tiene el jugador en el juego actual.
* **dinero** (int): el dinero que tiene actualmente el jugador.
* **carcel** (int): son los turnos que lleva el jugador en la cárcel si está en ella.
* **casilla** (int): es la casilla en la que está actualmente el jugador.

**Métodos**:

* **Jugador**: método constructor con parámetros.
* **estaEnCarcel()**: indica si el jugador está o no en la cárcel. Devuelve un boolean con el resultado de la consulta.
* **moverCasilla(int posiciones)**: posiciona al jugador en el número de casilla pasado por parámetro. Devuelve un entero con la posición final.
* **Métdodos get/set()**: métodos get/set para establecer o obtener el valor de los atributos de la clase.

# Clase Humano

La clase humano extiende de jugador, ya que un humano sería un jugador y otro podría ser, por ejemplo, la IA, que, en este caso está fuera del alcance del proyecto.

\*AÑADIR tabla de diagrama de clases

**Métodos**:

* **Humano**: método constructor con parámetros.

# Clase Dado

\*AÑADIR tabla de diagrama de clases

La clase dado está enfocada al uso de dicha funcionalidad, tirar el dato.

**Atributos**:

* **Generador** (int): **Número aleatorio.** Utiliza la función random(). Necesario importar java.util.random

**Métodos**:

* **tirarDado**: Devuelve un número aleatorio que será usado para establecer el valor de la tirada del dado

# Clase DatosPartida

\*AÑADIR tabla de diagrama de clases

La clase dado está enfocada a establecer y obtener los datos actuales de la partida

**Atributos**:

* **Generador** (int): **Número aleatorio.** Utiliza la función random(). Necesario importar java.util.random

**Métodos**:

* **tirarDado**: Devuelve un número aleatorio que será usado para establecer el valor de la tirada del dado

# Control de versiones

Como hemos comentado en el apartado 4.11 (Definición del entorno de trabajo), el control de versiones y trabajo conjunto se realizará con el entorno VCS Github. Los motivos que hayamos elegido este SW es que tenemos relativamente bastante experiencia y es fácil de usar por todos los miembros del equipo de desarrollo.

La estructura de funcionamiento a alto nivel de esta aplicación se explica en la siguiente imagen:



Además del proyecto en Netbeans, hemos incluido el borrador de la memoria del proyecto (este mismo documento) y el cronograma con los hitos del proyecto. De esta forma, podemos tener toda la documentación sincronizada para todos los miembros del equipo.

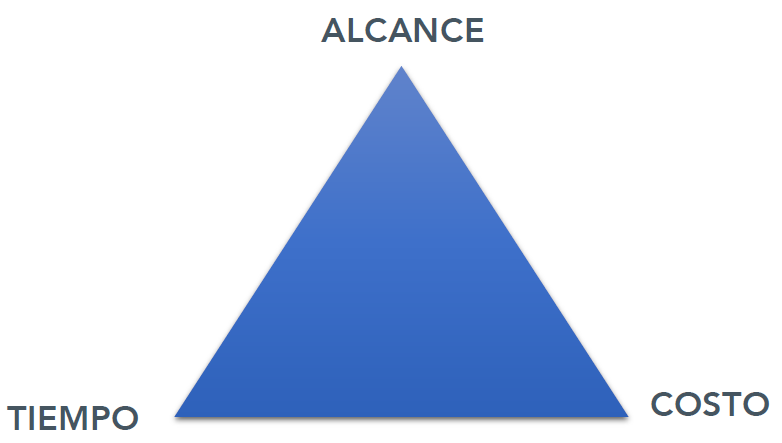
Cada desarrollador ha instalado el programa Github en su equipo local, de esta forma se crea el entorno local de trabajo. Seguidamente un desarrollador ha creado en la plataforma web la estructura del proyecto en Netbeans y documentación en el servidor de Github, esta estructura tendrá una dirección URL de acceso y todos los desarrolladores han clonado en su dispositivo local con el objetivo de vincular la carpeta local con el servidor común, para nuestro trabajo la carpeta en el servidor se accede desde la siguiente URL:

<https://github.com/raulgarcia03004/Monolopy.git>

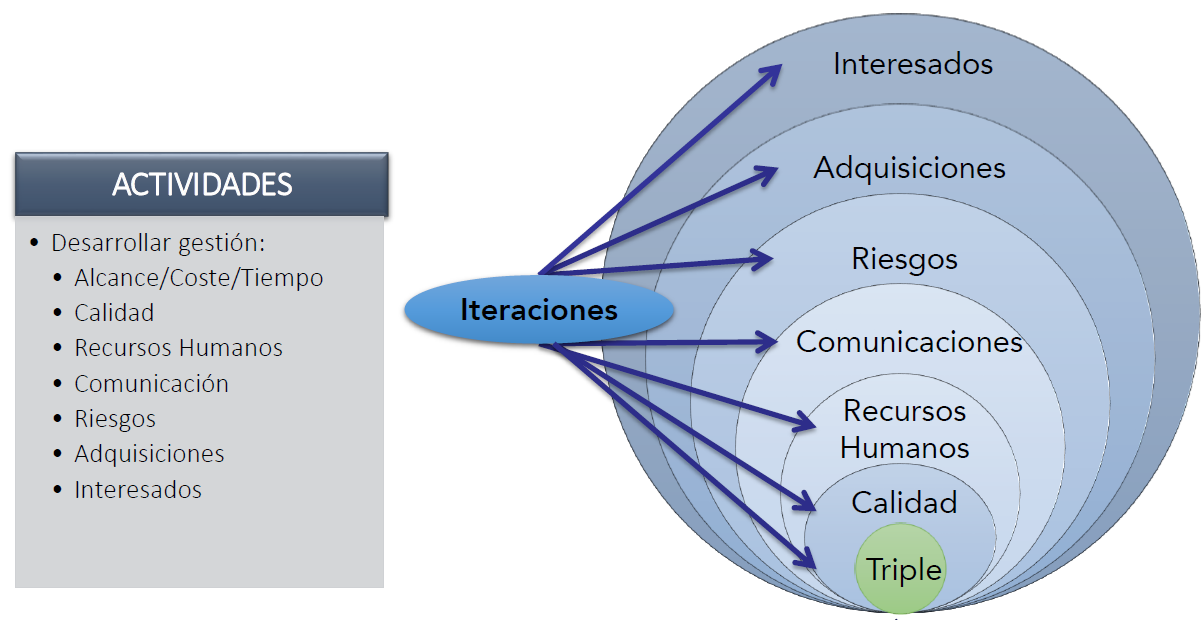
# Fase de control del proyecto

En esta fase se incluye los procesos y actividades que permite asegurar que el proyecto satisfará las necesidades por las cuales fue iniciado.

Siempre nos apoyaremos en la triple restricción: Alcance, Costo y Tiempo para equilibrar la balanza entra lo que espera nuestro cliente “Inv2AS” versus EL tiempo y costo estimado en la fase de planificación. Esta triple restricción, en su estado óptimo, tendría el aspecto de un triángulo equilátero como el de la figura siguiente:



Nosotros nos basaremos en todo momento en esta triple restricción para alcanzar los objetivos del proyecto. No obstante existe un nuevo paradigma en el cual nos apoyaremos en alguna ocasión. Este modelo se denomina “Triple restricción ampliada”



# Responsabilidades relacionadas con la calidad

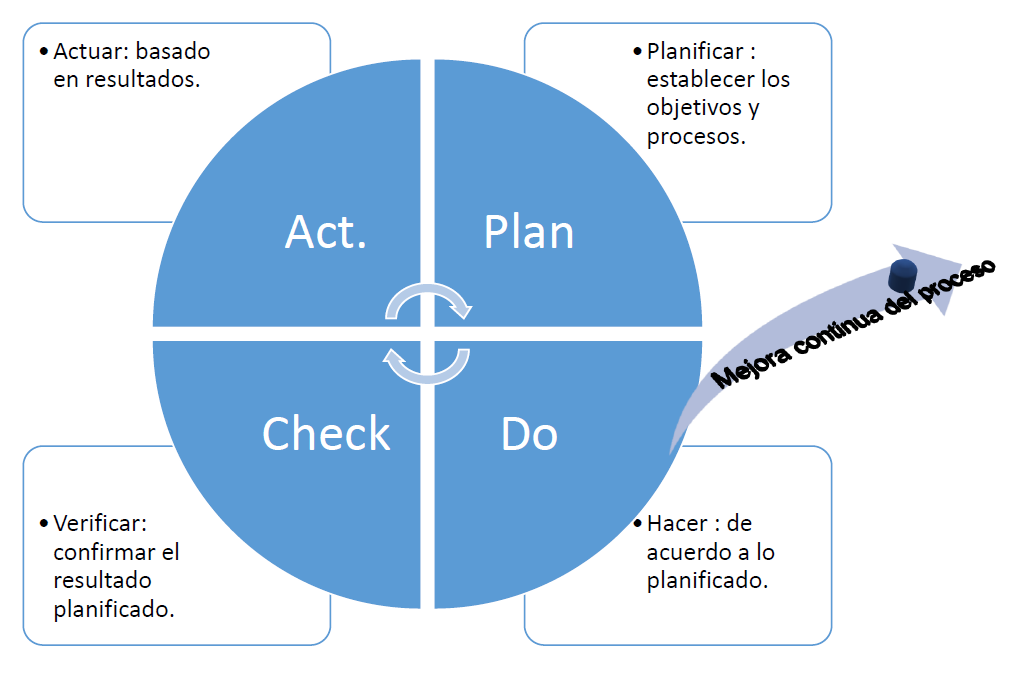
Si bien la calidad de cualquier producto de software depende de todas las personas involucradas durante el proceso de creación del mismo, la mayor responsabilidad sobre la calidad final del producto debería ser garantizada por un profesional del software testing. Dicho profesional buscará apoyar con su metodología, técnicas y herramientas, la calidad del producto mediante actividades de pruebas bien definidas, para proporcionar la confianza adecuada de que un producto o servicio cumplirá las expectativas y necesidades de los usuarios en términos de efectividad, eficiencia, usabilidad y satisfacción. De todas formas en “LMC”

* **Senior Manager**: responsable de la calidad en la empresa.
* **Project Manager**: último responsable de la calidad del proyecto.
* **Equipo del Proyecto**: responsables de su propio trabajo.
* **Responsable de calidad**: responsable de certificar que se están cumpliendo los requisitos del proyecto. No puede depender del PM

# Procesos de calidad en el proyecto

|  |  |
| --- | --- |
|  |  |
| **Planificación de calidad**:   * Encontrar estándares de calidad para el producto y la gestión del proyecto. * Crear estándares específicos para el proyecto * Determinar el trabajo a desarrollar * Determinar cómo se medirán los productos y actividades * Balancear las necesidades de calidad con el alcance, coste , tiempo y satisfacción |
| **Realizar Aseguramiento de Calidad**:   * Auditar los resultados obtenidos * Determinar si el proyecto cumple con las políticas y procesos de la organización * Identificar las oportunidades de mejora * Recomendar cambios y acciones correctoras |
| **Realizar Control de Calidad**:   * Medir los resultados según los estándares |

Para el control de calidad, LMC se basa en la mejora continua o ciclo PDCA. El Ciclo PDCA constituye una estrategia de mejora continua de la calidad en cuatro pasos, también se lo denomina espiral de mejora continua y es muy utilizado por los diversos sistemas utilizados en las organizaciones para gestionar aspectos tales como calidad (ISO 9000), medio ambiente (ISO 14000), salud y seguridad ocupacional (OHSAS 18000), o inocuidad alimentaria (ISO 22000).



Para nuestro proyecto, nos ayudará a revisar qué se está haciendo, si se está haciendo bien y si podemos mejorarlo, tanto para este proyecto (Monopoly Inv2AS) como para futuros proyectos que puedan surgir.

Para llevar a cabo este proceso, LMC implanta unas medidas y actividades concretas para el desarrollo del ciclo PDCA. Estas medidas son:

* Asignar un **líder** del proceso: en nuestro caso es la misma persona que desarrolla el rol de PM.
* Revisar los **niveles** de calidad del proyecto: cada vez que se va a cumplir un hito, en LMC revisamos (auditamos) si se ha cumplido al 100% las metas de dicho hito.
* Revisar los posibles valores de **riesgo** del proyecto: en estas auditorías intentamos identificar riesgos que se puedan dar.
* Establecer medidas de **corrección**: estableceremos medidas de corrección para los riesgos identificados.
* Analizar la **eliminación** de aquellos indicadores que ya no se consideren necesarios.