P.F. ENTERTAINMENT

Carlos Castaño Moreno	carlos051247@uma.es
Nuria Rodríguez Tortosa	nurirt36@uma.es
Daniel García Rodríguez	dani.gr@uma.es
Javier Leiva Dueñas	javi__@uma.es
Javier Lanceta Salas	jlancetasalas@uma.es
Guillermo Tell González	guillermotellg@uma.es
Julián Castro Coloma	julianc2b@uma.es
Pablo Fernández Serrano	pablofs02@uma.es

Enlace a repositorio: GitHub
Enlace a página web: Trivial

Contenido

1. INTRODUCCIÓN

Con nuestra aplicación llamada **Sabelotodo**, lo que buscamos es entretener y permitir que los usuarios puedan jugar a un juego tradicional, de manera online y gratuita mientras aprenden.

Nuestro primer proyecto será una adaptación de Trivial con nuevos objetivos e ideas, como partidas rápidas. Iremos subiendo actualizaciones en nuevos parches.

2. INSTRUCCIONES DEL JUEGO

Se trata de un juego individual, que se juega sobre un tablero de casillas de distintas modalidades temáticas: deportes, animales, conocimientos generales, historia y ciencia, en las que habrá que responder diferentes preguntas de opción múltiple, y en el que habrá también casillas con efectos especiales. El jugador dispone de un número inicial de vidas; de un contador de puntuación, que irá incrementándose si las preguntas se han respondido correctamente; una ficha, para señalizar la posición actual del jugador; y un dado, que servirá para decidir el número de casillas para avanzar.

El tablero, en cualquier caso, se genera de forma aleatoria al inicio de la partida, a excepción de las casillas especiales, que siempre tendrán una misma posición asignada. El juego presenta dos modos de partida, una partida desafío, donde se tiene un tablero de treinta casillas, que es dividido en tres dificultades: fácil, medio y difícil, y por otro lado, un modo de partida rápida, que permite al jugador seleccionar la dificultad deseada, y donde se dispone de un total de veinte casillas. Esta es la única diferencia entre los modos de juego, ya que el resto de la mecánica del juego, que explicamos a continuación, es exactamente igual.

La partida empieza en la primera casilla (por la izquierda), generándose en ese momento la pregunta de la modalidad que presenta la casilla. El jugador dispone de un tiempo mostrado por pantalla para responder dicha pregunta, sumándole puntuación si acierta la respuesta correcta sin haber agotado el temporizador o, por el contrario, su número de vidas se verá reducido en una unidad. Tras esto el jugador tirará el dado, y avanzará tantas casillas como indique el mismo. En el momento en el que el jugador presente cero vidas, se acabará la partida. Si por otro lado se llega a una casilla especial, ocurrirá el efecto mostrado, como puede ser la pérdida de vidas, retroceso de casillas...

Si no se han agotado las vidas, la partida finalizará cuando se alcance la última casilla, mostrándole al jugador la puntuación total obtenida en la partida.

3. ROLES

Para la asignación de roles, se ha procurado que todos los puestos del proyecto estén cubiertos por más de una persona, para minimizar el impacto que puede tener el abandono de un miembro, de tal manera que siempre haya alguien responsable de cada parte.

Tras una reunión de todo el equipo, y mediante previo consenso, se han asignado todos los roles, teniendo en cuenta las fortalezas y debilidades de cada miembro del equipo.

Planificación	Diseño	Implementación	Pruebas	Scrum Master	Analista
Tell, G.	Castaño, C.	Leiva, J.	Tell, G.	Castaño, C.	Tell, G.
Castaño, C.	Fernández, P.	Lanceta, J.	Leiva, J.	Castro, J.	Rodríguez, N.
Rodríguez, N.	Rodríguez, N.	Fernández, P.	Lanceta, J.	García, D.	
	Castro, J.	García, D.	Castro, J.		

Product Owner: D. Javier Troya Castilla.

4. GESTIÓN DEL RIESGO

En este apartado, realizamos un análisis de todos los posibles peligros a los que nos enfrentamos emprendiendo este proyecto. Comentaremos con detalle cada uno de ellos, así como la probabilidad de que ocurran y la estrategia empleada para resolverlos, y que no supongan un gran problema en cuanto a la planificación de desarrollo.

1. Fuga de personal (Tipo: Personal, Efectos: Tolerables): debido a baja por enfermedad, o la decisión de algún miembro del equipo que no sigue interesado en continuar. Es algo poco probable de que ocurra de forma voluntaria, ya que el equipo presenta gran interés en el desarrollo y finalización de esta aplicación.

   Estimamos que, si falta una persona, quedan suficientes desarrolladores para seguir según lo previsto, por lo que su impacto en el desarrollo sería bajo, y una manera de solucionarlo es buscar a personal que esté interesado y cualificado por este tipo de proyectos.

2. Complejidad del producto a desarrollar (Tipo: Tecnológico, Efectos: Serios): causado en ocasiones por la falta de "realismo" por parte de todo el equipo, al imaginar un producto que no es capaz de cumplir con la cronología exigida por parte del cliente, o incapacidad de desarrollo. Desde el comienzo de este proyecto estamos procurando en todo momento, que el producto se ciña a nuestras posibilidades tanto a nivel de desarrollo, como del tiempo disponible. Por lo tanto, hay que buscar un equilibrio entre cubrir las necesidades inicialmente planteadas, como la capacidad de desarrollo. En el caso de que se diera tendría un impacto elevado sobre el producto final.

3. Fallo de la base de datos (Tipo: Tecnológico, Efectos: Serios): dado que dependemos de un repertorio de preguntas, que sin duda alguna constituye el núcleo del proyecto, y sin la que se podría llevar a cabo el correcto funcionamiento de la aplicación. Es algo poco probable de que ocurra, debido a que tenemos pensado realizar copias de seguridad de estos datos, para así tenerlos siempre disponibles, o al menos una parte importante que garantice el correcto funcionamiento mientras se repara la conexión con la misma, así que su efecto sería bajo.

4. Falta de motivación (Tipo: Personal, Efectos: Catastróficos): que puede ser derivada de la propia complejidad que conlleva el desarrollo, o bien porque alguien crea que no se cumplen los objetivos que tenía pensados. Esta situación es de las menos probables que ocurran, pero una forma de evitar que ocurra es intentar que el producto final sea consensuado por todos los integrantes del proyecto. Al igual que la fuga de personal supondría un impacto bajo en el trabajo.

5. Cambio en el planteamiento inicial del proyecto (Tipo: Organización, Efectos: Tolerable/Catastrófico): que puede ser motivado por no haber entendido exactamente los requerimientos del cliente, por falta de personal, falta de recursos temporales, o de cualquier otra incidencia. Estimamos que puede presentar un gran impacto en el normal desarrollo del proyecto, en la medida de cómo se modifica todo lo inicialmente planteado. En esta situación lo mejor es la comunicación con el cliente, para poder encontrar una solución, así como la reunión de todo el equipo, para tomar una decisión de manera conjunta.

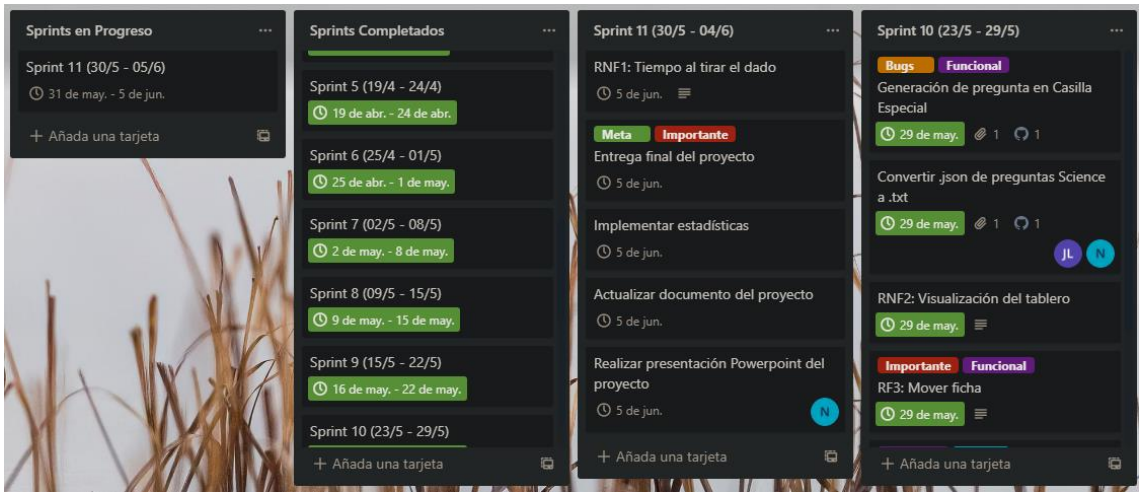
6. Pérdida de parte del proyecto (Tipo: Tecnológico, Efectos: Serios): debido a no almacenar debidamente alguna sección de código, o alguna parte de la memoria. Creemos que no es probable que ocurra, porque procuramos tener todas las copias de seguridad necesarias, y notificar los pequeños cambios que se vayan produciendo, en el caso de que se produzca una pérdida de código importante tendría un elevado impacto en el proyecto, ya que aumentaría el tiempo necesario para su finalización.

   Puede deberse a múltiples situaciones, y diversas partes del proyecto pueden verse involucradas, por lo que en cada caso habría que ver como asegurar que tenga el menor impacto posible, y conlleva una gran prevención, por parte de todo el equipo, para asegurarnos de que todo en lo que se vaya avanzando, lo tengamos correctamente guardado.

5. PLANIFICACIÓN

Para la organización de las subtareas del proyecto decidimos emplear la metodología Scrum, ya que nos permite tener más flexibilidad a los posibles cambios y adaptarnos a los contratiempos en el desarrollo del proyecto, además de permitir a todos los miembros del equipo a tomar parte de todas las partes del proyecto.

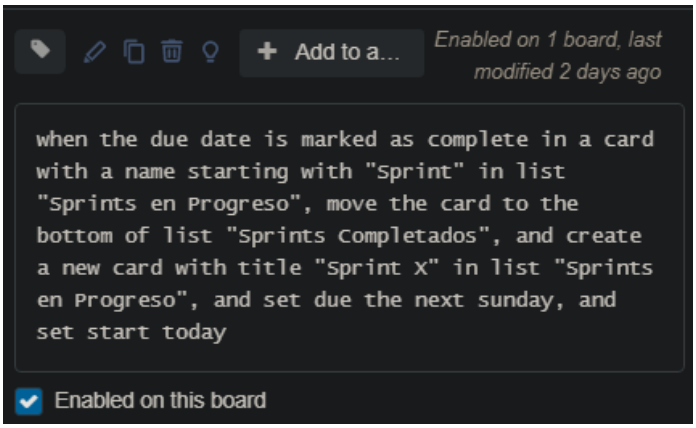
Por otro lado, gracias a su metodología incremental e iterativa, vamos comprobando la corrección de las partes anteriores al *sprint* actual, a través de reuniones semanales con todo el equipo.



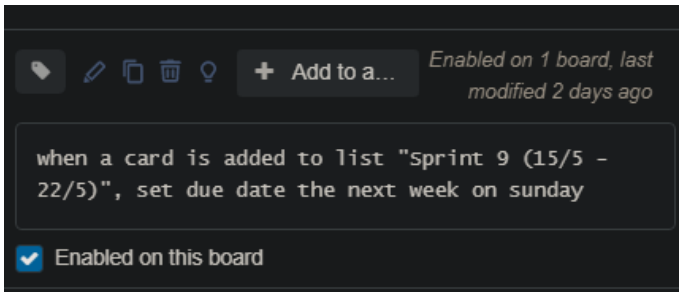
Captura de las listas de tareas en Trello

4.1 POWER-UPS

1. Automatización QoL con **Butler**:
    a. Botón “Comenzar tarea”, que añade al usuario a la tarea y la marca como “En progreso”.
    b. Botón “Terminar tarea”, que marca como cumplidas todas sus subtareas, y elimina la etiqueta “En progreso”.
    c. Botón global “Mover completadas”, que mueve todas las tarjetas completadas a una lista auxiliar “Completed tasks”. **(archivado)**
    d. Botón global “Completar sprint”, que mueve de vuelta las tarjetas en “Completed tasks” a su sprint correspondiente. **(archivado)**
    e. Regla semiautomática para crear nuevos sprints semanales:



    f. Regla automática para asignar fecha de fin a las tareas de un sprint (actualizada cada semana):



6. REQUISITOS

En este apartado, presentamos los distintos requisitos surgidos de una entrevista con el cliente, así como una posterior reunión de todo el equipo. Para ello, tras la obtención de estos, los hemos especificado, analizado, y gestionado, para finalmente verificarlos y validarlos (Ingeniería de requisitos). Mostramos los requisitos explicados y el diagrama UML generado con el programa Magic Draw.

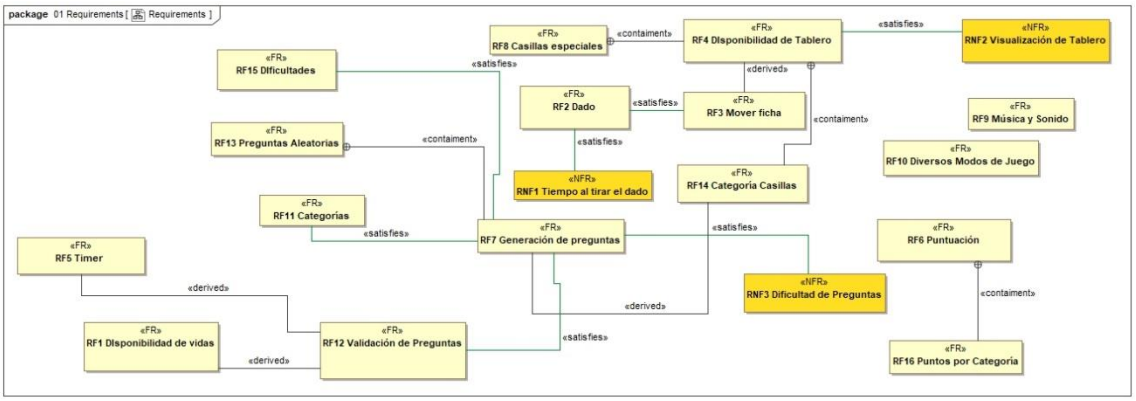


Diagrama de Requisitos

TARJETAS DE REQUISITOS

RF1: Disponibilidad de vidas

Como cliente,

Quiero tener disponibilidad de vidas.

Para tener un sistema que nos permita calcular el número de errores que comete el usuario y poder terminar la partida antes de llegar a la victoria si se excede el máximo de ellas.

Prueba de aceptación

1. Al principio el usuario empezará con un número determinado de vidas.
2. En el caso de que el usuario conteste una pregunta de manera errónea, o se quede sin tiempo se le restará una vida.
3. El usuario tendrá la oportunidad de ganar una vida si cae en una de las casillas especiales del tablero.
4. Si el usuario se queda sin vidas, el juego terminará y habrá perdido.

RF2: Dados

Como usuario/jugador,

Quiero poder tirar dados.

Para poder avanzar sobre el tablero.

RF3: Mover ficha

Como sistema,

Quiero que se pueda mover la ficha del jugador cuando tire el dado.

Para que se garantice el correcto funcionamiento del juego, y además que la partida no entre en estado de bloqueo.

RF4: Disponibilidad de tablero

Como cliente,

Quiero tener siempre disponible al menos un tablero previamente generado.

Para que se garantice el que pueda siempre jugar a una partida, aun habiendo fallo en la generación de este.

Prueba de aceptación

1. Al iniciar una nueva partida, se procurará generar un tablero aleatorio

preferiblemente, con una distribución uniforme de casillas de categorías de preguntas.
2. El tablero generado tendrá una forma lineal, con casillas de categorías de preguntas, y otras casillas especiales que pueden suponer por ejemplo la pérdida de vidas.
3. En caso de error, existirá un tablero de repuesto, para que siempre se garantice su disponibilidad.

RF5: Timer

Como cliente,

Quiero que el juego cuente con un temporizador.

Para limitar cuánto tiempo el jugador tiene para responder la pregunta.

RF6: Puntuación

Como cliente,

Quiero que el juego tenga un sistema de puntuación cuando respondes preguntas.

Para que los jugadores puedan comparar puntuaciones entre ellos.

RF7: Generación de preguntas

Como sistema,

Quiero tener disponibles las preguntas en una base de datos.

Para poder generar preguntas de diferentes categorías y dificultades.

Prueba de aceptación

- Se recibe una petición de pregunta al caer en una casilla.
- Se accede al fichero que almacene las preguntas de la categoría escogida.
- Se filtran las preguntas según la dificultad escogida.
- Se elige una pregunta aleatoria de entre las filtradas.
- Nos aseguramos de que no pueda volver a ser escogida la misma pregunta.

RF8: Casillas especiales

Como cliente,

Quiero que el tablero disponga de casillas especiales que modifiquen el juego.

Para mejorar la jugabilidad.

RF9: Música y Sonido

Como usuario,

Quiero que se reproduzca música de fondo y ciertos sonidos como consecuencia a algunas acciones.

Para mejorar la experiencia de manera indirecta.

RF10: Diversos Modos de Juego

Como usuario,

Quiero se pueda seleccionar al inicio de la partida qué modo se quiere jugar.

Para añadir más diversidad con la misma base.

RF11: Categorías

Como usuario,

Quiero tener preguntas de distintas categorías.

Para que el juego tenga más variedad.

RF12 Validación de preguntas

Como sistema,

Quiero comprobar que la respuesta sea la correcta.

Para contar correctamente el progreso de cada jugador.

Prueba de aceptación

1. Al responder una pregunta, comprobar que su respuesta es la correcta.

2. Comprobar que el usuario ha respondido la pregunta dentro del tiempo establecido.
3. Sumar puntos de esa categoría a la estadística del usuario si ha acertado la pregunta.
4. Indicar que el usuario ha perdido una vida si falla la pregunta.
5. Indicar que el usuario ha perdido una vida si no ha contestado la pregunta antes de que el timer llegue a 0.

RF13: Preguntas Aleatorias

Como cliente,

Quiero que las preguntas se generen de forma aleatoria.

Para que no puedan salir preguntas repetidas.

RF14: Categoría Casillas

Como cliente,

Quiero que las casillas del tablero se dividan en las distintas categorías.

Para tener las preguntas divididas según su categoría.

RF15: Dificultades

Como cliente,

Quiero que haya distintos niveles de dificultad.

Para mejorar la jugabilidad según la habilidad del jugador.

RF16: Puntos por categoría

Como usuario,

Quiero obtener distintos puntos según la categoría de la pregunta.

Para medir mi conocimiento en las distintas categorías.

RNF1: Tiempo al tirar el dado
Como usuario, **Quiero** que no tarde mucho la animación y el funcionamiento del dado. **Para** agilizar el funcionamiento del juego.
RNF2: Visualización de tablero
Como usuario, **Quiero** que el tablero y la interfaz se vea correctamente y sea agradable a la vista.
RNF3: Dificultad de preguntas
Como usuario, **Quiero** que la dificultad de las preguntas mostradas sea acorde a la dificultad elegida. **Para** que la dificultad del juego sea coherente.

7. CASOS DE USO

CU1. Crear partida

Contexto de uso: Cuando el jugador lo requiera, podrá empezar una nueva partida en el modo de juego seleccionado.

Precondiciones y activación: El jugador está conectado al juego en el menú principal.

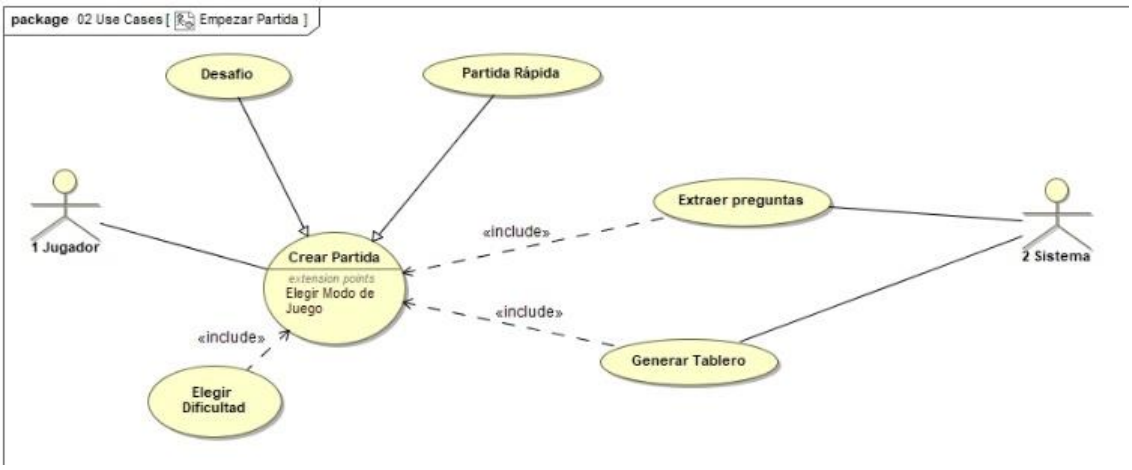
Garantías de éxito / Postcondición: Se crea una nueva partida.

Escenario principal:

1. El jugador selecciona una nueva partida.
2. El jugador selecciona el modo de juego y la dificultad.
3. El sistema genera un nuevo tablero.
4. El sistema genera la ficha que representa al jugador.
5. El sistema extrae un conjunto de preguntas.
6. Se muestra al jugador la interfaz del juego con el tablero generado.

Escenarios alternativos:

- El sistema no consigue generar el tablero y se mostrará un error por pantalla.



Crear partida

CU2. Elegir Dificultad

Contexto de uso: Al crear la partida, el jugador podrá seleccionar la dificultad de la partida.

Precondiciones y activación: El jugador ha decidido empezar una partida.

Garantías de éxito / Postcondición: El jugador ha podido seleccionar la dificultad sin problema.

Escenario principal:

1. El jugador selecciona el modo de juego “Partida rápida”.
2. El jugador selecciona dificultad.
3. Se genera el tablero con preguntas de dicha dificultad seleccionada.

Escenarios alternativos:

- No se puede seleccionar la dificultad.

CU3. Desafío

Contexto de uso: el jugador elige este modo y el sistema genera 3 tableros.
Precondiciones y activación: hay que darle a crear partida y posteriormente, elegir este modo.
Garantías de éxito / Postcondición: se generan los 3 niveles correctamente.
Escenario principal:

1. Crear partida.
2. Seleccionar modo “Desafío”.
3. Generación de tableros.
4. Comienzo de la partida.

Escenarios alternativos:

- Elección de la partida rápida.
- No se generan correctamente los tableros.

CU4. Partida Rápida

Contexto de uso: el jugador elige este modo y se elige un tablero.
Precondiciones y activación: elección de este modo y de la dificultad.
Garantías de éxito / Postcondición: correcta generación del tablero.
Escenario principal:

1. Crear partida.
2. Seleccionar modo “Partida rápida”.
3. Seleccionar dificultad.
4. Generación del tablero.
5. Comienzo de partida.

Escenarios alternativos:

- Elección del modo “desafío”.
- No se puede generar el tablero.
- No se puede seleccionar dificultad.

CU5. Extraer preguntas

Contexto de uso: El jugador ha empezado una partida, y el sistema extrae del fichero de bases de datos de preguntas, diversas preguntas de distintas categorías para la dificultad de la partida.
Precondiciones y activación: El jugador ha empezado una partida.
Garantías de éxito / Postcondición: Se han extraído varias pregunta con la dificultad seleccionada y de varias categorías.
Escenario principal:

1. El jugador comienza una partida.
2. El sistema extrae varias preguntas de la base de datos de todas las categorías, de la dificultad adecuada.
3. El sistema almacena esas preguntas en un archivo temporal.

Escenarios alternativos:

1. No se extrae ninguna pregunta.
2. Las preguntas extraídas contienen duplicados.
3. La pregunta extraída no es de la dificultad de la partida**.**

CU6. Generar tablero

Contexto de uso: El jugador ha seleccionado las opciones de partida y el sistema ha generado el tablero.
Precondiciones y activación: Se ha seleccionado la dificultad y el tipo de partida
Garantías de éxito / Postcondición: El tablero se ha generado sin ningún tipo de error y sin ningún bucle infinito, pudiendo recorrerse desde el inicio hasta el final.

Escenario principal:

1. El jugador selecciona la dificultad.
2. El jugador selecciona el tipo de partida.
3. El jugador le da a confirmar.
4. El sistema llama a la función para crear tablero.
5. El tablero se crea correctamente y se inicia la partida.

Escenarios alternativos:

- El tablero generado no se genera con el camino recurrible.
- No se invoca a la función generar tablero.
- El jugador se sale de la partida.

CU7. Jugar

Contexto de uso: Cuando la partida es creada, el jugador podrá jugar en ella.
Precondiciones y activación: El jugador ha seleccionado un modo de juego y el sistema ha creado la partida.
Garantías de éxito / Postcondición: El jugador podrá jugar la partida seleccionada.
Escenario principal:

1. El jugador tirará un dado.
2. El sistema moverá al jugador por el tablero en proporción al número que ha sacado el jugador en el dado.
3. El sistema seleccionará una pregunta y se la mostrará al jugador.
4. El sistema activa el temporizador.

5. El jugador debe responder a esa pregunta antes de que el tiempo expire.
6. Si el jugador acierta, el sistema deberá actualizar la puntuación.
7. Si el jugador falla, el sistema le quitará una vida.
8. Si el jugador llega a la meta o el jugador se ha quedado sin vidas, el sistema termina la partida.

Escenario alternativo:

- No se puede tirar el dado**.**
- El jugador no visualiza la pregunta correctamente.
- El sistema no reduce el número de vidas cuando el jugador falle una pregunta o no responda antes de que el tiempo se agote.
- El temporizador no funciona correctamente.
- El sistema no termina la partida.

CU8. Tirar Dado

Contexto de uso: El jugador podrá tirar los dados al inicio de cada turno.
Precondiciones y activación: Empieza el inicio del turno del jugador.
Garantías de éxito / Postcondición: Se ha hecho clic correctamente en el dado que ha dado el número de casillas que va a avanzar.
Escenario principal:

1. El jugador ya ha creado partida.
2. Se seleccionó dificultad.
3. Inicia el turno del jugador.
4. El jugador tira el dado para poder avanzar.
5. El jugador avanza tantas casillas como el dado indique.
6. Responde la pregunta (puede fallar sin perder todas las vidas o acertar).
7. Vuelve a iniciar turno para tirar de nuevo los dados.

Escenarios alternativos:

- Se acabaron las vidas y no hay más inicios de turno.
- Se acabó el tablero y no hay más inicios de turno.
- Se acaba la partida, pero el jugador vuelve a crear partida.

CU9. Seleccionar pregunta

Contexto de uso: Cuando el jugador cae en una casilla, el sistema elige una pregunta.
Precondiciones y activación: El jugador está en partida y ha tirado el dado.
Garantías de éxito / Postcondición: Se le muestra una pregunta al usuario.
Escenario principal:

1. El jugador cae en una casilla.
2. Dependiendo del tipo de casilla se selecciona una categoría de pregunta.

3. Se le muestra una pregunta de selección múltiple o de verdadero o falso.

Escenarios alternativos:

- No se genera la pregunta correctamente.
- No se dispone de más preguntas**.**

CU10. Acertar Pregunta

Contexto de uso: El jugador responde a una pregunta.
Precondiciones y activación: Se le presenta una pregunta y se selecciona una respuesta.
Garantías de éxito / Postcondición: Se responde correctamente.
Escenario principal:

1. Se le muestra la pregunta escogida.
2. El jugador responde.
3. El sistema valida la respuesta.
4. Ha seleccionado la respuesta correcta.

Escenarios alternativos:

- Tras seleccionar la respuesta, el sistema se queda colgado sin validar la respuesta.

CU11. Fallar Pregunta

Contexto de uso: El jugador responde a una pregunta.
Precondiciones y activación: Se le presenta una pregunta y se selecciona una respuesta.
Garantías de éxito / Postcondición: Se responde de manera incorrecta.
Escenario principal:

1. Se le muestra la pregunta escogida.
2. El jugador responde.
3. El sistema valida la respuesta.
4. Ha seleccionado la respuesta incorrecta.

Escenarios alternativos:

- Tras seleccionar la respuesta, el sistema se queda colgado sin validar la respuesta.

CU12. Perder Vida

Contexto de uso: El jugador ha respondido a la pregunta de manera errónea o se ha quedado sin tiempo.
Precondiciones y activación: El sistema valida la respuesta como incorrecta o el timer es 0.
Garantías de éxito / Postcondición: Se le reduce una vida al jugador.
Escenario principal:

1. Si el sistema detecta una de las dos condiciones de pérdida de vida.

2. Se resta una vida al total del jugador.
3. Se actualiza el contador de vidas.

Escenarios alternativos:

- No se resta la vida correctamente.
- Al acertar una pregunta, se le resta una vida.

CU13. Temporizador

Contexto de uso: Cuando el jugador recibe una pregunta a responder, el sistema comienza el temporizador.
Precondiciones y activación: El jugador está en partida y ha tirado el dado.
Garantías de éxito / Postcondición: Se muestra el temporizador funcional al usuario.

Escenario principal:

1. El jugador cae en una casilla.
2. El jugador recibe una pregunta y se inicia el temporizador (mostrándose).
3. El usuario responde a la pregunta.
4. El temporizador se detiene.

Escenarios alternativos:

- El jugador no responde a la pregunta, y el temporizador se agota.
- El temporizador no se inicia correctamente.
- El temporizador no se detiene.

CU14. Avanzar casillas

Contexto de uso: El jugador tira el dado y avanza tantas casillas como indique el dado.
Precondiciones y activación: El usuario tira el dado.
Garantías de éxito / Postcondición: En la imagen, la ficha avanza a la siguiente casilla.
Escenario principal:

1. El usuario lanza el dado y este muestra un número del 1 al 6.
2. El sistema mueve la ficha tantas casillas como indica el dado.

Escenarios alternativos:

- El número de casillas que quedan es menor que el número que ha salido en el dado por lo que solo se avanzan las casillas que quedan y termina el tablero (CU16).

CU15. Actualizar puntuación

Contexto de uso: El sistema actualiza la puntuación tras cada turno.
Precondiciones y activación: El jugador ha respondido la pregunta y avanzado de casilla terminando así ese turno.

Garantías de éxito / Postcondición: Los datos de puntuación han cambiado respecto al anterior turno.
Escenario principal:

1. El jugador responde una pregunta.
2. El sistema actualiza la puntuación en función del tiempo restante en el timer.

Escenarios alternativos:

- El jugador abandona la partida antes de responder. La puntuación se queda igual que en el anterior turno.

CU16. Terminar Tablero

Contexto de uso: El jugador ha recorrido todo el tablero.
Precondiciones y activación: El jugador ha sacado en el dado una cantidad mayor que las casillas que le quedan por recorrer.
Garantías de éxito / Postcondición: Aparece una pantalla indicando que el jugador ha terminado el tablero.
Escenario principal:

1. El jugador tira el dado.
2. Saca un número mayor que el número de casillas que le quedan por recorrer por lo que termina el tablero.
3. El sistema muestra una pantalla indicando que el jugador ha acabado el tablero.

Escenarios alternativos:

- El jugador abandona la partida antes de llegar al final.

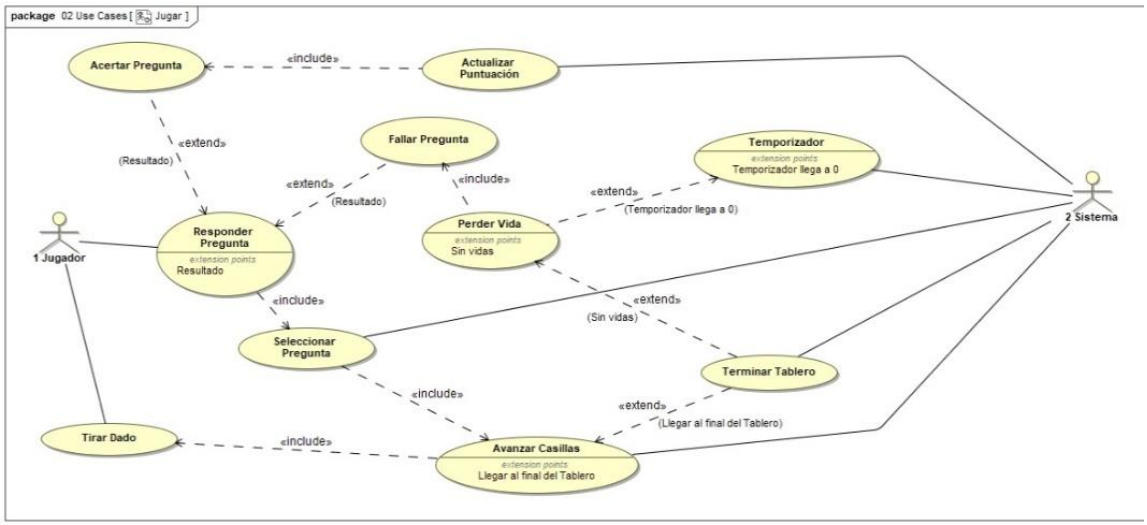
CU17. Responder Pregunta

Contexto de uso: La posición del jugador está en una determinada casilla, y se selecciona de forma aleatoria una pregunta, asociada a una dificultad y una modalidad.
Precondiciones y activación: El jugador debe tener al menos una vida.
Garantías de éxito / Postcondición: Se selecciona de forma correcta la pregunta del fichero de datos asociado a la modalidad (repertorio de preguntas).
Escenario principal:

1. El jugador dispone de un determinado tiempo para responder a la pregunta.
2. Gana puntuación en función del tiempo empleado.
3. En caso de respuesta incorrecta o agotamiento de tiempo pierde una vida.

Escenarios alternativos:

- No se carga correctamente la pregunta.
- El temporizador no funciona correctamente.



Jugar

CU18. Terminar partida

Contexto de uso: El jugador ha acabado la partida.
Precondiciones y activación: Para llegar a este estado el jugador debe haber completado el tablero o haber perdido todas las vidas.
Garantías de éxito / Postcondición: Se acaba la partida y no deja jugar más en esta.
Escenario principal:

1. Se pierden todas las vidas o se llega al final del tablero.
2. El sistema termina la partida y no permite jugar más.

Escenario alternativo: El sistema no consigue terminar la partida.

CU19. Mostrar Puntuación

Contexto de uso: El sistema muestra la puntuación conseguida durante la partida.
Precondiciones y activación: La partida tiene que haber terminado.
Garantías de éxito / Postcondición: Se muestra la puntuación correctamente en pantalla.
Escenario principal: La puntuación se muestra correctamente al jugador.
Escenarios alternativos: El sistema no muestra la puntación correctamente.

CU20. Jugar otra vez

Contexto de uso: El jugador selecciona jugar otra vez y el sistema crea una nueva partida.
Precondiciones y activación: La partida anterior tiene que haber terminado.
Garantías de éxito / Postcondición: Se genera una nueva partida.
Escenario principal: El sistema permite crear una nueva partida.
Escenarios alternativos: El sistema no permite jugar otra vez.

CU21. Compartir Puntación

Contexto de uso: El jugador podrá compartir la puntuación conseguida durante la partida.

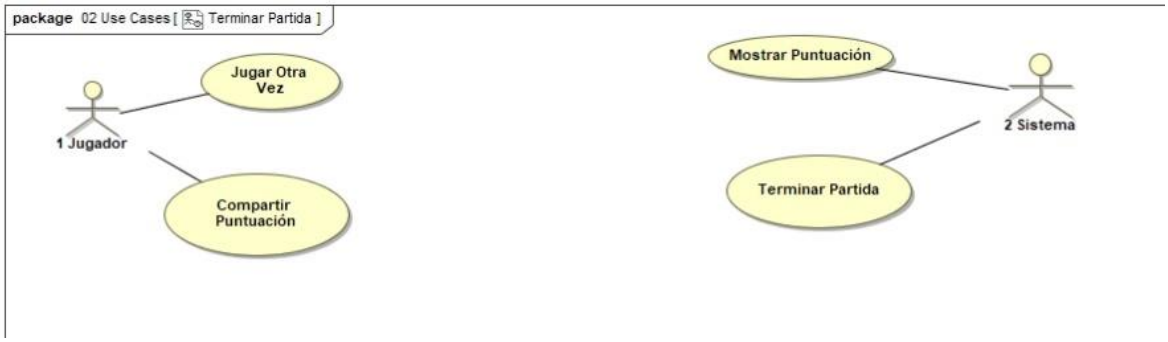
Precondiciones y activación: La partida tiene que haber terminado y se ha mostrado la puntuación.

Garantías de éxito / Postcondición: Se comparte la puntación en la aplicación elegida.

Escenario principal:

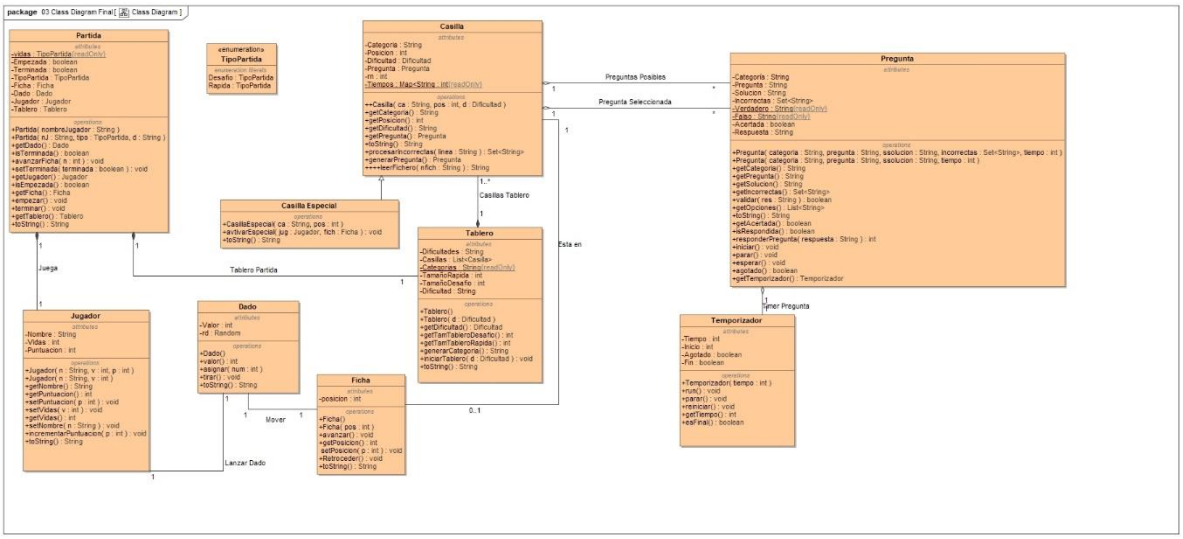
1. El jugador selecciona la opción de compartir.
2. Se muestran las diferentes opciones para compartirla.
3. El jugador elige cómo compartir la puntación.

Escenarios alternativos: El sistema no permite compartir la puntuación



Terminar partida

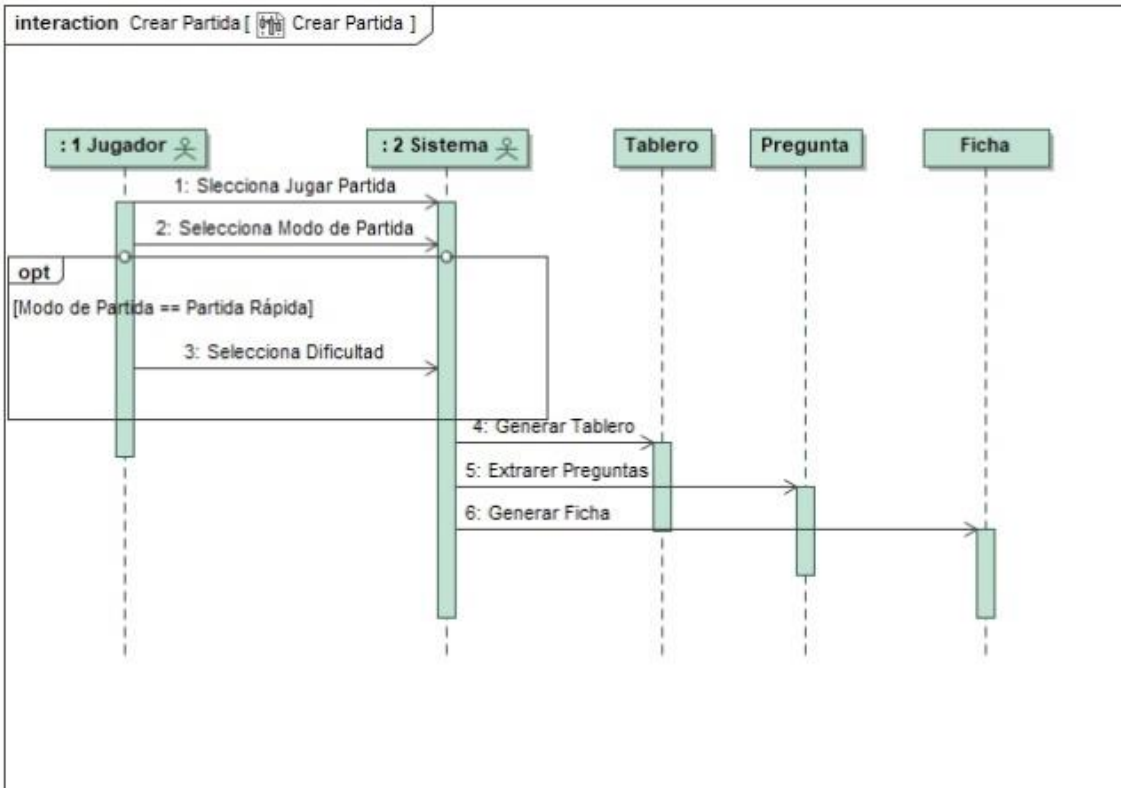
8. MODELOS DE DOMINIO



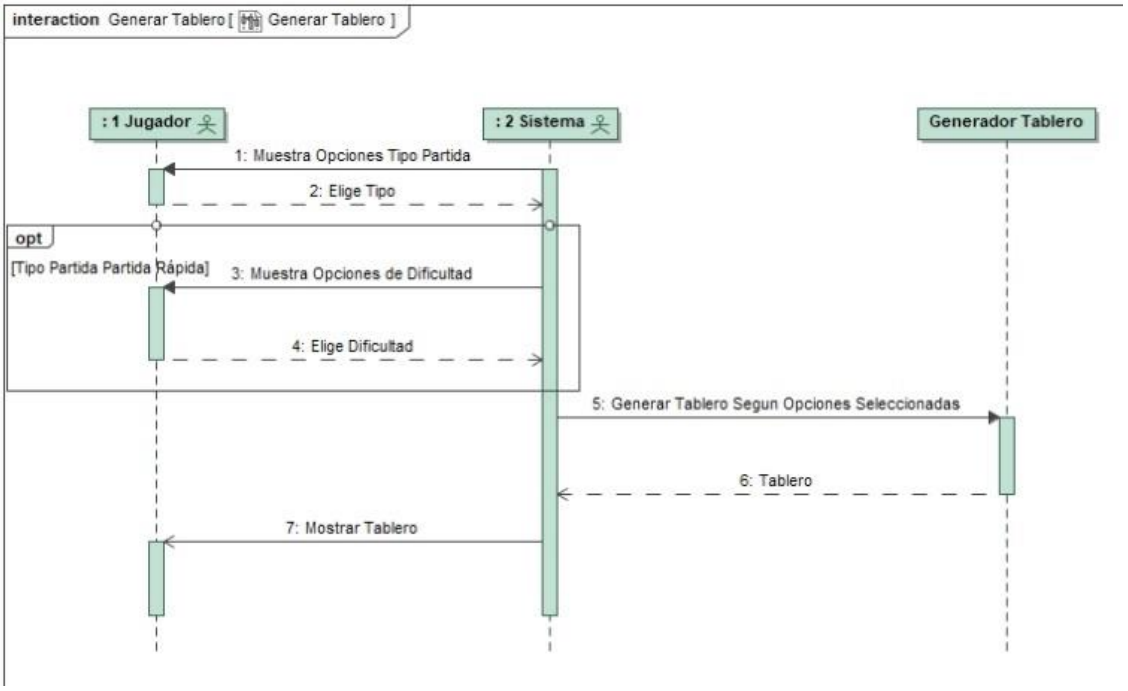
9. DIAGRAMAS DE SECUENCIA

Crear partida:

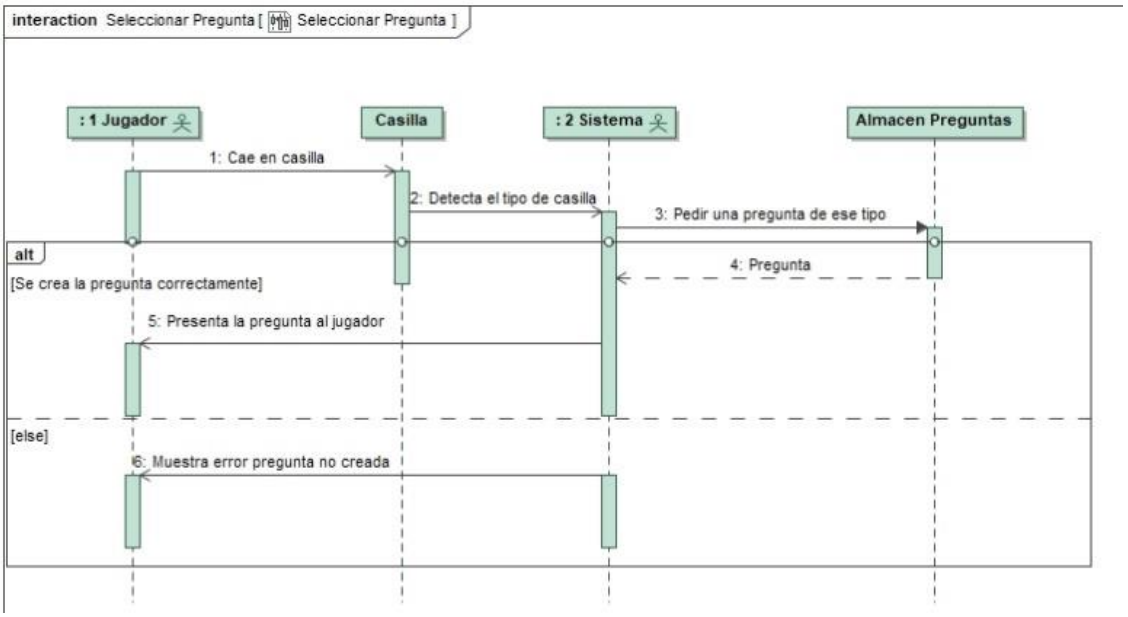
El jugador selecciona “Jugar partida”, posteriormente, el jugador selecciona modo de partida y si dicho modo es “partida rápida”, el jugador deberá seleccionar la dificultad con la que desea jugar. Después, el sistema genera el tablero, extrae un conjunto de preguntas y generará la ficha.



Generar tablero:
El sistema muestra los tipos de partida y el jugador elige un tipo. Si ha elegido partida rápida el sistema muestra las opciones de dificultad, y el jugador elige el que prefiera. A continuación, el sistema llama a la función para generar el tablero con las opciones seleccionadas, y muestra el tablero generado.

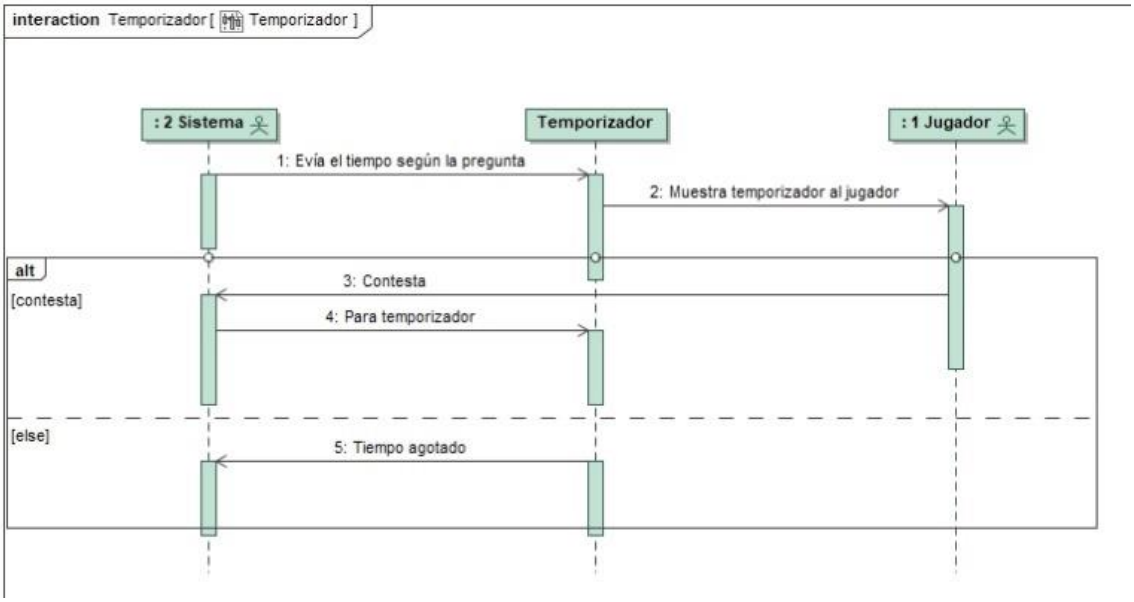


Seleccionar pregunta:
El sistema reconoce la casilla en la que ha caído el jugador después de moverse. Luego, intenta extraer una pregunta de este tipo, si lo consigue se la muestra al jugador, si por el contrario no lo consigue, mostrará un mensaje de error.



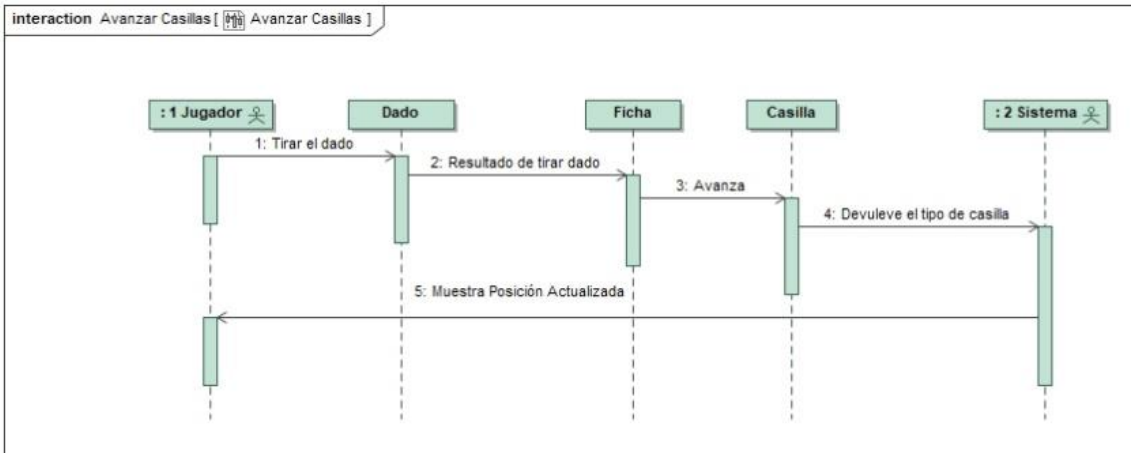
Temporizador:

El sistema, según la dificultad de la pregunta, pone un temporizador con el tiempo apropiada y se le muestra al jugador. Si el jugador contesta, el sistema parará el temporizador, si no, se acaba el temporizador.



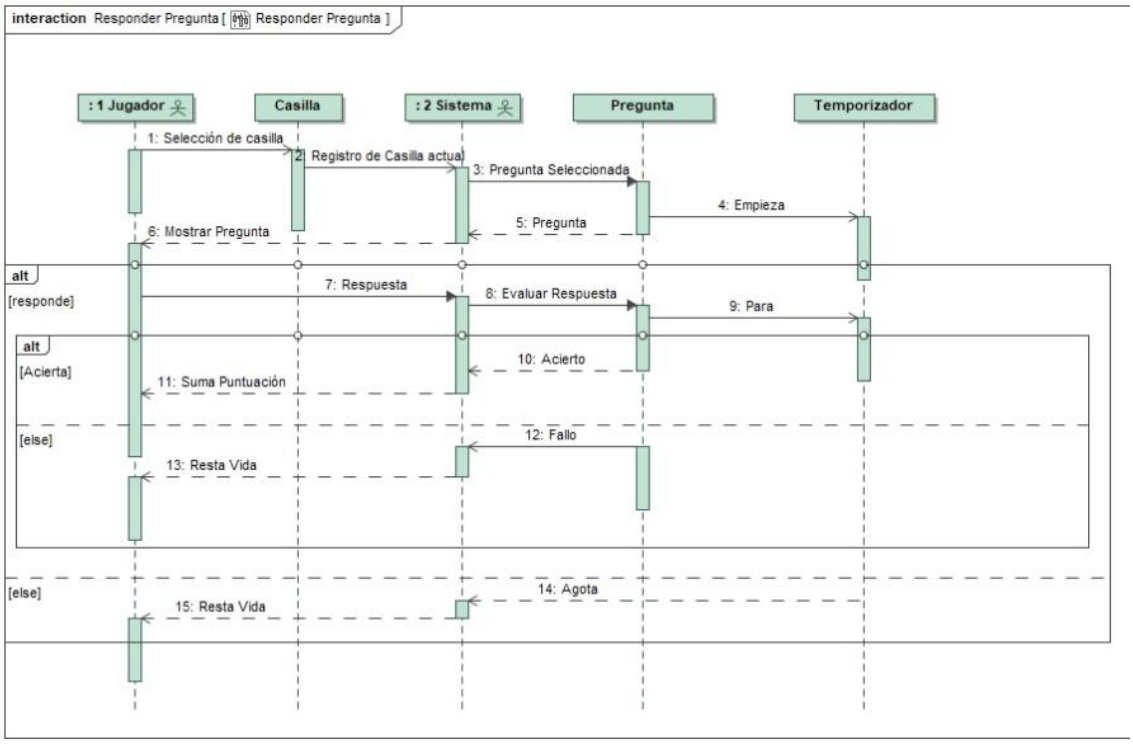
Avanzar casilla:

El jugador tira el dado, el resultado se mostrará por pantalla y eso hará que la ficha avance casillas. Posteriormente, la casilla le indica al sistema de qué tipo es, pregunta o suerte y el sistema mostrará la posición actualizada.



Responder preguntas:
En este escenario, primero el jugador estará situado en una casilla, bien en la casilla inicial de la partida, o bien en una que haya sido seleccionada posteriormente a tirar el dado, que será registrada en el sistema. Seguidamente el sistema mostrará la pregunta al jugador, empezando el tiempo disponible para responder. Cuando todo lo anterior haya sucedido, tenemos dos posibilidades:

- El jugador responde la pregunta a tiempo, por lo que el sistema parará el temporizador que estaba activo, y posteriormente evaluará la pregunta, sumándole puntuación en caso de acierto (en función del tiempo empleado), y restándole una vida en caso de fallo.
- Agotamiento del temporizador, debido a que el jugador no ha respondido en el tiempo disponible, y le supondrá la pérdida de una vida.



10. PRUEBAS JUNIT

- Pruebas Dado

```
package PrimeraFila.PFENTERTAINMENT;

import static org.junit.jupiter.api.Assertions.assertTrue;

import java.util.HashSet;
import java.util.Set;

import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import Modelo.Dado;

public class DadoTest {
	Dado d;

	@BeforeEach
	public void init() {
		d = new Dado();
	}

	@AfterEach
	public void terminate() {
		d = null;
	}

	@Test
	public void siTiroElDadoDevuelveUnValor() {
		d.tirar();
		assertTrue(d.valor() > 0 && d.valor() < 7);
	}

	@Test
	public void elDadoDevuelveTodosLosPosiblesValores() {
		Set<Integer> lista = new HashSet<>();
		for (int i = 1; i < 7; ++i) {
			lista.add(i);
		}
		while (!lista.isEmpty()) {
			d.tirar();
			if(lista.contains(d.valor())) {
				lista.remove(d.valor());
			}
		}
		assertTrue(lista.isEmpty());
	}
}
```

- Pruebas Temporizador

```
package PrimeraFila.PFENTERTAINMENT;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertThrows;
import static org.junit.jupiter.api.Assertions.assertTrue;

import java.util.concurrent.TimeUnit;

import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import Modelo.Temporizador;

public class TemporizadorTest {
	Temporizador t;

	@BeforeEach
	public void init() {
		t = new Temporizador(3);
	}

	@AfterEach
	public void terminate() {
		t = null;
	}

	@Test
	public void alPrincipioElTimerNoEstaAgotado() {
		assertFalse(t.getResultado());
	}

	@Test
	public void siSeRespondeAntesDeTiempoNoEstaAgotado() {
		t.start();
		try {
			TimeUnit.SECONDS.sleep(1);
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
		assertFalse(t.getResultado());
	}

	@Test
	public void siNoSeRespondeTiempoEstaAgotado() {
		t.start();
		try {
			TimeUnit.SECONDS.sleep(4);
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
		assertTrue(t.getResultado());
	}
```

```
	@Test
	public void
alReiniciarElTemporizadorElTiempoSeQuedaIgualQueAlPrincipio() {
		int inicio = t.getTiempo();
		t.start();

		try {
			TimeUnit.SECONDS.sleep(4);
		} catch (InterruptedException e) {
			e.printStackTrace();
		}

		t.reiniciar();
		assertEquals(inicio, t.getTiempo()-1);
	}

	@Test
	public void siSeCreaElTimerConUnValorIncorrectoLanzaExcepcion() throws
IllegalArgumentException {
		Exception exception =
assertThrows(IllegalArgumentException.class, () -> new Temporizador(-5));
	    assertEquals("Valor incorrecto.", exception.getMessage());
	}

}
```

- Pruebas Pregunta

```
package PrimeraFila.PFENTERTAINMENT;

import static org.junit.jupiter.api.Assertions.assertTrue;
import static org.junit.jupiter.api.Assertions.assertAll;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.when;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.Set;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import Modelo.Casilla;
import Modelo.Pregunta;

public class PreguntaTest {

	Pregunta pregunta;

	@BeforeEach
	public void init() {
		Set<String> incorrectas = new HashSet<String>();
```

```
            incorrectas.add("A");
            incorrectas.add("B");
            incorrectas.add("C");
            pregunta = new Pregunta("Categoria", "Pregunta", "Solucion",
incorrectas);

      }

      @Test
      public void laPreguntaSeHaCreadoCorrectaMente() {
            Casilla cMock = mock(Casilla.class);
            when(cMock.getPregunta()).thenReturn(pregunta);

            Pregunta p = cMock.getPregunta();

            assertAll(
                        () -> assertEquals("Pregunta", p.getPregunta()),
                        () -> assertEquals("Solucion", p.getSolucion())
                        );
      }

      @Test
      public void lasIncorrectasTienenQueSerTresONull() {
            Casilla cMock = mock(Casilla.class);
            when(cMock.getPregunta()).thenReturn(pregunta);

            Pregunta p = cMock.getPregunta();

            assertTrue(p.getIncorrectas() == null ||
p.getIncorrectas().size() == 3);
      }

      @Test
      public void elMetodoValidarFuciona() {
            Casilla cMock = mock(Casilla.class);
            when(cMock.getPregunta()).thenReturn(pregunta);

            Pregunta p = cMock.getPregunta();

            assertAll(
                        () -> assertTrue(p.validar("solucion")),
                        () -> assertFalse(p.validar("B"))
                        );

      }

      private List<String> copiaLista(List<String> l){
            List<String> res = new ArrayList<>();
            for(String str : l) res.add(str);
            return res;
      }

      @Test
      public void lasOpcionesAparecenSoloUnaVez() {
            Casilla cMock = mock(Casilla.class);
            when(cMock.getPregunta()).thenReturn(pregunta);
```

```
            Pregunta p = cMock.getPregunta();
            List<String> opciones = p.getOpciones();
            List<String> opcionesCopia = copiaLista(opciones); //para no
modificar la original
            boolean res = false;

            Iterator<String> it = opciones.iterator();
            while(it.hasNext() && !res) {
                  String op = it.next();
                  opcionesCopia.remove(op);
                  res = !(opcionesCopia.contains(op));
            }

            assertTrue(res);
      }

      @Test
      public void laSolucionEstaEnLasOpciones() {
            Casilla cMock = mock(Casilla.class);
            when(cMock.getPregunta()).thenReturn(pregunta);

            Pregunta p = cMock.getPregunta();

            boolean apareceSolucion = false;
            String solucion = p.getSolucion();
            List<String> opciones = p.getOpciones();

            Iterator<String> it = opciones.iterator();
            while (it.hasNext() && !apareceSolucion) {
                  apareceSolucion = it.next().equalsIgnoreCase(solucion);
            }

            assertTrue(apareceSolucion);
      }
}
```

11. HERRAMIENTAS

Para la gestión y desarrollo del proyecto, así como para la constante comunicación de todo el equipo, destacamos el uso de las siguientes aplicaciones:

- **Microsoft Word**: Redacción del presente documento (memoria del proyecto).
- **Discord**: Comunicación y organización de las sesiones síncronas del proyecto.
- **GitHub**: Almacenamiento y gestión del repositorio del proyecto online.
- **Git**: Manejo del repositorio del proyecto de forma local.
- **WhatsApp**: Comunicación entre los miembros.
- **Trello**: Organización de las partes del proyecto.
- **Magic Draw:** realización de diagramas.