GUADARRAMA CHAVEZ SAMMY | juárezz garcía ángel de jesús

tecnologías para la construcción de software

JUAN CARLOS PÉREZ ARRIAGA

documentación del proyecto final

desarrollo del juego uno

12 de enero de 2021

**Tabla de contenido**

[1. Introducción 5](#_Toc61381261)

[2. Modelo de dominio 6](#_Toc61381262)

[3. Modelo de datos 7](#_Toc61381263)

[4. Diagrama de casos de uso 9](#_Toc61381264)

[5. Descripciones de casos de uso 10](#_Toc61381265)

[6. Reglas del juego 17](#_Toc61381266)

[7. Prototipos 19](#_Toc61381267)

[8. Selección del Framework de desarrollo 26](#_Toc61381268)

[9. Estándar de codificación 27](#_Toc61381269)

[9.1 Convención de nombres 27](#_Toc61381270)

[9.2 Formatos o estilos 27](#_Toc61381271)

[**9.2.1** **Identación** 27](#_Toc61381272)

[**9.2.2** **Comentarios** 28](#_Toc61381273)

[9.3 Patrones de escritura de código 28](#_Toc61381274)

[**9.3.1** **Declaraciones** 28](#_Toc61381275)

[**9.3.2** **Espacio en blanco** 38](#_Toc61381276)

[**9.3.3** **Referencia a los métodos y variables de clase** 41](#_Toc61381277)

[**9.3.4** **Asignaciones de variables** 42](#_Toc61381278)

[**9.3.5** **Uso de paréntesis** 43](#_Toc61381279)

[9.4 Manejo de excepciones 43](#_Toc61381280)

[9.5 Documentación técnica 46](#_Toc61381281)

[**9.5.1** **<summary>** 46](#_Toc61381282)

[**9.5.2** **<returns>** 46](#_Toc61381283)

[**9.5.3** **<exception>** 47](#_Toc61381284)

[**9.5.4** **<see>** 47](#_Toc61381285)

[**9.5.5** **<param>** 47](#_Toc61381286)

[10. Análisis estático del código 48](#_Toc61381287)

[11. Conclusiones 50](#_Toc61381288)

**Tabla de Figuras**

[Figura 1. Modelo de dominio. 6](file:///D:\Juego-UNO\documentacion\DocumentacionFinal.docx#_Toc61381289)

[Figura 2. Diagrama Entidad – Relación. 7](file:///D:\Juego-UNO\documentacion\DocumentacionFinal.docx#_Toc61381290)

[Figura 3. Modelo Relacional. 8](file:///D:\Juego-UNO\documentacion\DocumentacionFinal.docx#_Toc61381291)

[Figura 4. Diagrama de casos de uso. 9](file:///D:\Juego-UNO\documentacion\DocumentacionFinal.docx#_Toc61381292)

[Figura 5. Ventana "Registrar cuenta". 19](file:///D:\Juego-UNO\documentacion\DocumentacionFinal.docx#_Toc61381293)

[Figura 6. Ventana "Registro en espera". 20](file:///D:\Juego-UNO\documentacion\DocumentacionFinal.docx#_Toc61381294)

[Figura 7. Ventana "Inicio". 21](file:///D:\Juego-UNO\documentacion\DocumentacionFinal.docx#_Toc61381295)

[Figura 8. Ventana "Verificar código". 21](file:///D:\Juego-UNO\documentacion\DocumentacionFinal.docx#_Toc61381296)

[Figura 9. Ventana "Recuperar cuenta". 22](file:///D:\Juego-UNO\documentacion\DocumentacionFinal.docx#_Toc61381297)

[Figura 10. Ventana "Creación sala". 23](file:///D:\Juego-UNO\documentacion\DocumentacionFinal.docx#_Toc61381298)

[Figura 11. Ventana "Partida". 23](file:///D:\Juego-UNO\documentacion\DocumentacionFinal.docx#_Toc61381299)

[Figura 12. Ventana "Unirse a sala". 24](file:///D:\Juego-UNO\documentacion\DocumentacionFinal.docx#_Toc61381300)

[Figura 13. Ventana "Iniciar sesión". 25](file:///D:\Juego-UNO\documentacion\DocumentacionFinal.docx#_Toc61381301)

[Figura 14. Ventana "Perfil". 25](file:///D:\Juego-UNO\documentacion\DocumentacionFinal.docx#_Toc61381302)

[Figura 15. Análisis estático en el servidor. 48](file:///D:\Juego-UNO\documentacion\DocumentacionFinal.docx#_Toc61381303)

[Figura 16. Análisis estático para el cliente. 49](file:///D:\Juego-UNO\documentacion\DocumentacionFinal.docx#_Toc61381304)

# **Introducción**

El presente documento tiene como objetivo presentar los artefactos relacionados con el diseño y desarrollo de la aplicación de escritorio del juego UNO. Todos estos se fueron ajustando según a las necesidades y el alcance del proyecto.

La presente sección es la primera de este documento, donde se detalla de manera general el objetivo que tiene y las partes que lo conforman. La siguiente sección presenta el modelo de dominio que se diseñó con las entidades necesarias, después se encuentra el modelo de datos, donde se muestra el diagrama Entidad – Relación y el Modelo Relacional para que los datos persistan.

En la siguiente sección se presentan los requerimientos funcionales con el diagrama de casos de uso y en seguida, las descripciones de esos casos de uso. En la sección posterior se detallan las reglas del juego que se definieron para el desarrollo del proyecto, siguiendo con la sección de los prototipos de las ventanas. Después, en la sección de selección del Framework se justica por qué se utilizaron dichos Frameworks o tecnologías.

Finalmente, se presentan las conclusiones donde se detallan las lecciones aprendidas con el desarrollo del juego y las partes que quedaron por hacer.

# **Modelo de dominio**

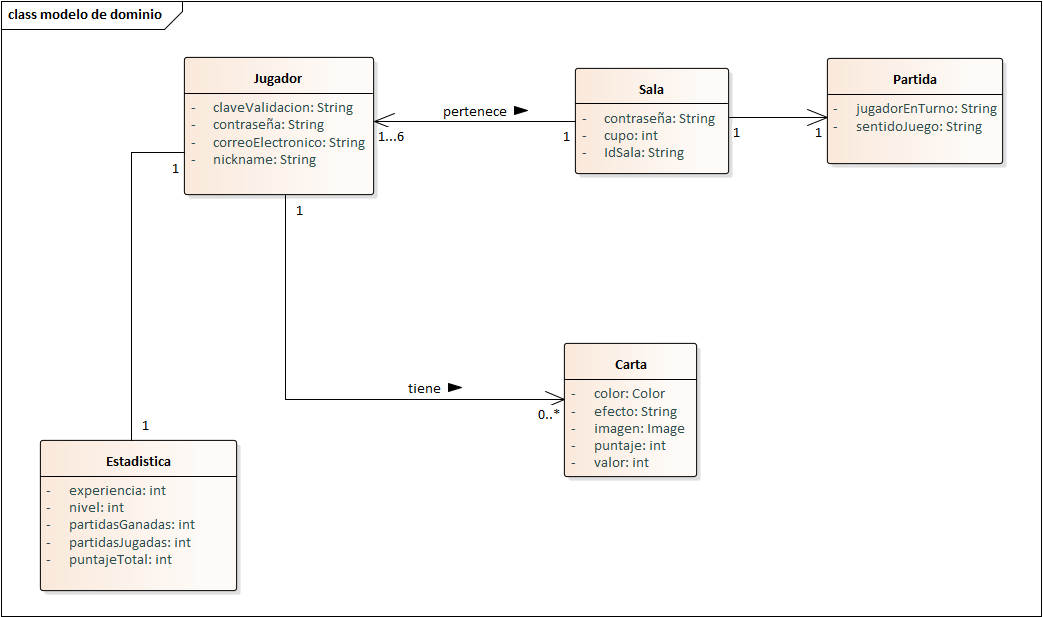
En la Figura 1 se muestra el modelo de dominio que se realizó para el juego UNO. Este define una base de las entidades que se tomaron en cuenta en el desarrollo de él.

Figura . Modelo de dominio.

# **Modelo de datos**

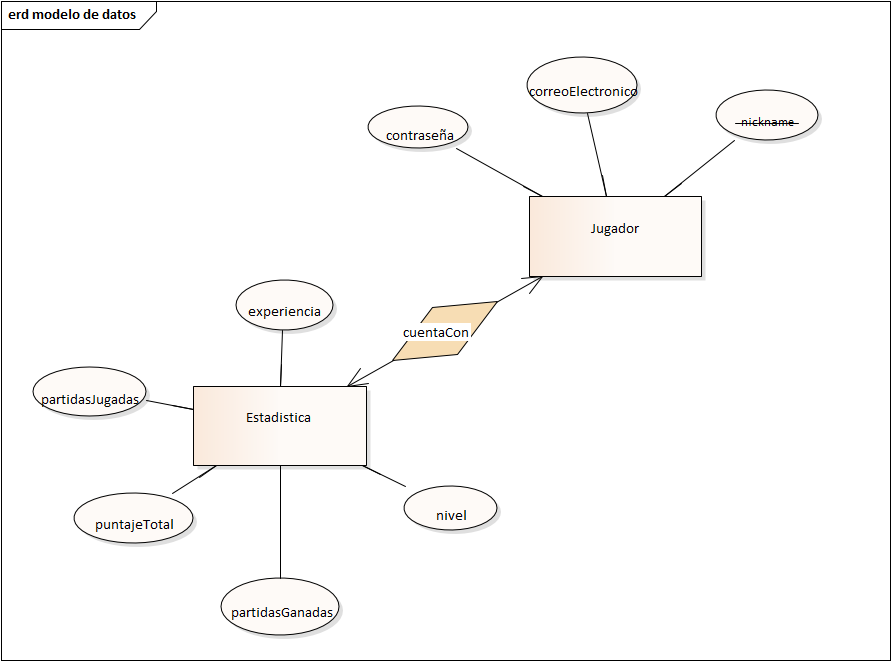
En la siguiente Figura 2 se muestra el diagrama Entidad – Relación para definir los datos que son necesarios que mantener.

Figura . Diagrama Entidad – Relación.

Esto, se tradujo a un Modelo Relacional que se muestra en la Figura 3.

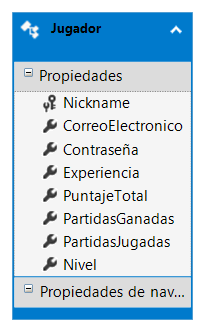
Este Modelo Relacional se encuentra ya en un sistema de gestión de bases de datos relacional, que es SQL Server.

Figura . Modelo Relacional.

# **Diagrama de casos de uso**

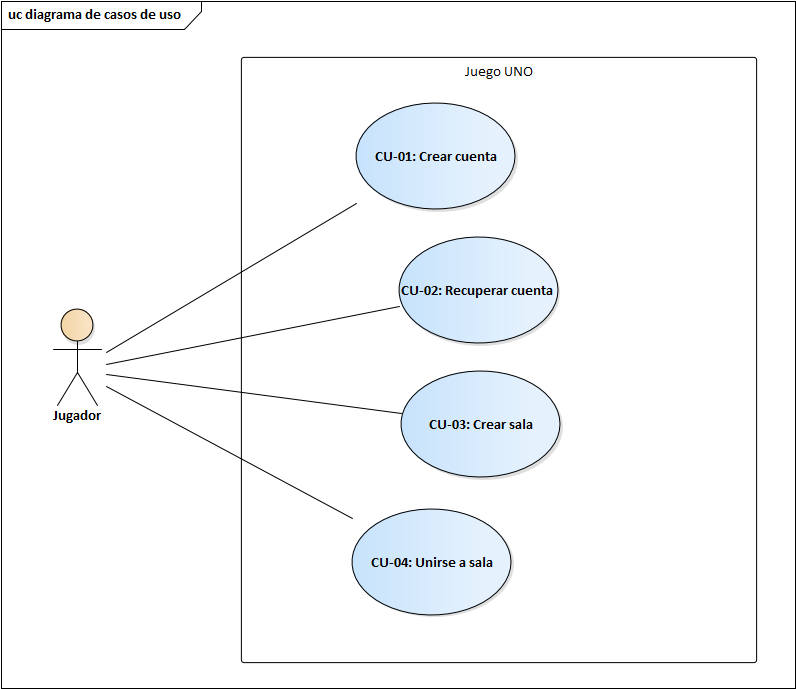
En la siguiente Figura 4 se muestra el diagrama de casos de uso para el Juego UNO.

Figura . Diagrama de casos de uso.

# **Descripciones de casos de uso**

A continuación, se presenta la descripción del caso de uso CU-01: Crear cuenta.

|  |  |
| --- | --- |
| **ID:** | CU-01 |
| **Nombre:** | Crear cuenta |
| **Autor(es):** | Sammy Guadarrama Chávez |
| **Fecha de creación:** | 12 de octubre de 2020 |
| **Fecha de actualización:** |  |
| **Descripción:** | El jugador crea una cuenta en el juego para poder crear sala o unirse a una y tener un seguimiento sobre el nivel en el juego y los puntos que lleva. |
| **Actor(es):** | **Jugador** |
| **Frecuencia de uso:** | No identificado |
| **Disparador:** | El **Jugador** selecciona el botón “Crear cuenta” en la ventana “Iniciar sesión” |
| **Precondiciones:** | No identificado |
| **Flujo Normal:** | 1. El Juego UNO muestra la ventana “Registrar cuenta” con un formulario para registrar una cuenta de un JUGADOR con los campos: Nombres, Apellidos, Correo electrónico, Nickname, Contraseña. Y los botones “Cancelar” y “Registrar”. 2. El **Jugador** ingresa la información solicitada en los campos y selecciona el botón “Registrar”. (Ver FA 2.1). 3. El Juego UNO verifica que los campos tengan información y que esté correcta, después guarda en la base de datos un JUGADOR con la información ingresada con un status “NoHabilitado”, crea una ClaveValidación y envía un correo con ella al correo electrónico especificado. Después, muestra la ventana “Registro en espera” con el mensaje “¡Ya casi terminamos! Te enviamos un correo a la dirección especificada con una Clave de validación para que confirmes tu registro. Por favor ingrésala en el siguiente campo.” Con el campo Clave validación y los botones “Enviar de nuevo” y “Aceptar”. (Ver FA 3.1, FA 3.2, EX1 y EX2) 4. El **Jugador** ingresa la Clave de validación y selecciona el botón “Aceptar”. (Ver FA 4.1). 5. El Juego UNO verifica que la Clave de validación sea correcta y muestra la ventana “Registro exitoso” con el mensaje: “Tu cuenta ha sido registrada exitosamente” y el botón “Aceptar”. (Ver FA 5.1). 6. El **Jugador** selecciona el botón “Aceptar”. 7. El Juego UNO muestra la ventana “Inicio”. 8. Termina caso de uso. |
| **Flujos Alternos:** | FA 2.1 – El Jugador cancela el registro de cuenta.   1. El **Jugador** selecciona el botón “Cancelar”. 2. El Juego UNO cierra la ventana “Registro cuenta” y muestra la ventana “Iniciar sesión”. 3. Termina caso de uso.   FA 3.1 – Todos o algunos de los campos están vacíos.   1. El Juego UNO muestra la ventana “Campos vacíos” con el mensaje: “Uno o varios campos están vacíos, por favor ingresa toda la información.” Y el botón “Aceptar”. 2. El **Jugador** selecciona el botón “Aceptar”. 3. El Juego UNO cierra la ventana “Campos vacíos” y regresa al paso 1 del flujo normal.   FA 3.2 – La información en los campos es incorrecta.   1. El Juego UNO muestra la ventana “Campos incorrectos” con el mensaje: “La información en uno o varios campos es incorrecta, por favor verifique la información.” Y el botón “Aceptar”. 2. El **Jugador** selecciona el botón “Aceptar”. 3. El Juego UNO cierra la ventana “Campos incorrectos” y regresa al paso 1 del flujo normal.   FA 4.1 – Reenvío de Clave de validación.   1. El **Jugador** selecciona el botón “Enviar de nuevo”. 2. El Juego UNO envía de nuevo la Clave de validación a la dirección de correo ingresada y muestra la ventana “Registro en espera”.   FA 5.1 – La Clave de validación es incorrecta.   1. El Juego UNO muestra la ventana “Clave incorrecta” con el mensaje: “La clave de validación ingresa es incorrecta, por favor verifique la información.” Y el botón “Aceptar”. 2. El **Jugador** selecciona el botón “Aceptar”. 3. El Juego UNO muestra la ventana “Registro en espera |
| **Excepciones:** | EX1 - El Juego UNO no pudo conectarse a la base de datos.   1. El Juego UNO muestra la ventana “Error de registro” con el mensaje: “No se pudo realizar el registro en estos momentos, por favor intente de nuevo.” Y el botón “Aceptar”. 2. El **Jugador** selecciona el botón “Aceptar”. 3. El Juego UNO cierra la ventana “Error registro” y vuelve al paso 1 del flujo normal.   EX2 – Error al enviar el correo electrónico para la validación de cuenta.   1. El Juego UNO muestra la ventana “Error correo” con el mensaje: “Ocurrió un error al enviar el correo de validación, por favor seleccione el botón Enviar de nuevo.” Y el botón “Aceptar”. 2. El **Jugador** selecciona el botón “Aceptar”. 3. El Juego UNO muestra la ventana “Registro en espera”. |
| **Postcondiciones:** | POST-1: El Juego UNO debe crear una cuenta para el jugador.  POST-2: El Juego UNO debe enviar un correo de validación con la Clave de validación. |
| **Incluye:** | No aplica |
| **Extiende:** | No aplica |
| **Prioridad:** | Alta |

Ahora, en la siguiente tabla se muestra la descripción para el caso de uso CU-02: Recuperar cuenta.

|  |  |
| --- | --- |
| **ID:** | CU-02 |
| **Nombre:** | Recuperar cuenta |
| **Autor(es):** | Sammy Guadarrama Chávez |
| **Fecha de creación:** | 12 de octubre de 2020 |
| **Fecha de actualización:** |  |
| **Descripción:** | El jugador recupera su cuenta al olvidar su contraseña para poder tener de nuevo acceso al juego. |
| **Actor(es):** | **Jugador** |
| **Frecuencia de uso:** | No identificado |
| **Disparador:** | El **Jugador** selecciona el botón “Recuperar cuenta” en la ventana “Inicio sesión”. |
| **Precondiciones:** | PRE-1: El Jugador debe ingresar su nombre de usuario. |
| **Flujo Normal:** | 1. El Juego UNO envía un Código de recuperación a la dirección de correo especificada y muestra la ventana “Verificar código” con el mensaje: “Enviamos un código de recuperación a su dirección de correo, por favor ingréselo en el siguiente campo:”, el campo Código de recuperación y los botones “Cancelar”, “Enviar de nuevo” y “Continuar”. 2. El **Jugador** ingresa el Código de recuperación y selecciona el botón “Continuar”. (Ver FA 2.1 y FA 2.2). 3. El Juego UNO verifica que el Código de recuperación sea correcto y muestra la ventana “Recuperación cuenta” con un formulario de los campos: Nueva contraseña, Confirmar nueva contraseña. Y los botones “Cancelar” y “Recuperar cuenta”. (Ver FA 3.1). 4. El **Jugador** ingresa la información solicitada y selecciona el botón “Recuperar cuenta”. (Ver FA 4.1). 5. El Juego UNO verifica que la Confirmación de contraseña coincida con la Nueva contraseña coincidan y actualiza la contraseña del JUJGADOR. Después, muestra la ventana “Cuenta recuperada” con el mensaje: “La información de tu cuenta ha sido actualizada, inicie sesión de nuevo.” Y el botón “Aceptar”. (Ver FA 5.1 y EX1). 6. El **Jugador** selecciona el botón “Aceptar”. 7. El Juego UNO muestra la ventana “Inicio sesión”. 8. Termina caso de uso. |

|  |  |
| --- | --- |
| **Flujos Alternos:** | FA 2.1 – El Jugador cancela el ingreso del código de recuperación.   1. El **Jugador** selecciona el botón “Cancelar”. 2. El Juego UNO cierra la ventana “Verificar código” y muestra la ventana “Iniciar sesión”. 3. Termina caso de uso.   FA 2.2 – Reenvío del código de recuperación.   1. El Jugador selecciona el botón “Enviar de nuevo”. 2. El Juego UNO regresa al paso 1 del flujo normal.   FA 3.1 – El código de recuperación es incorrecto.   1. El Juego UNO muestra la ventana “Código incorrecto” con el mensaje: “El código de recuperación es incorrecto, por favor verifique la información.” Y el botón “Aceptar”. 2. El **Jugador** selecciona el botón “Aceptar”. 3. El Juego UNO cierra la ventana “Código incorrecto” y muestra la ventana “Verificar código”.   FA 4.1 Cancelación de recuperación de cuenta.   1. El **Jugador** selecciona el botón “Cancelar”. 2. El Juego UNO cierra la ventana “Recuperación cuenta” y muestra la ventana “Iniciar sesión”. 3. Fin de caso de uso.   FA 5.1 Las contraseñas no coinciden.   1. El Juego UNO muestra la ventana “Contraseñas incorrectas” con el mensaje: “La confirmación de la contraseña no coinciden, por favor verifique la información.” Y el botón “Aceptar”. 2. El **Jugador** selecciona el botón “Aceptar”. 3. El Juego UNO cierra la ventana “Contraseñas incorrectas” y muestra la ventana “Recuperación cuenta”. |
| **Excepciones:** | EX1 - El Juego UNO no pudo conectarse a la base de datos.   1. El Juego UNO muestra la ventana “Error de recuperación” con el mensaje: “No se pudo actualizar la contraseña, intente de nuevo.” Y el botón “Aceptar”. 2. El **Jugador** selecciona el botón “Aceptar”. 3. El Juego UNO cierra la ventana “Error de recuperación” y muestra la ventana “Recuperación cuenta”. |
| **Postcondiciones:** | POST-1: El Juego UNO debe actualizar la contraseña del jugador. |
| **Incluye:** | No aplica |
| **Extiende:** | No aplica |
| **Prioridad:** | Alta |

En la siguiente tabla se muestra la descripción del caso de uso CU-03: Crear sala.

|  |  |
| --- | --- |
| **ID:** | CU-03 |
| **Nombre:** | Crear sala |
| **Autor(es):** | Sammy Guadarrama Chávez |
| **Fecha de creación:** | 12 de octubre de 2020 |
| **Fecha de actualización:** |  |
| **Descripción:** | El jugador crea una sala para poder invitar a demás jugadores a unirse a la sala mediante el Número de sala |
| **Actor(es):** | **Jugador** |
| **Frecuencia de uso:** | Aproximadamente 3 veces por uso del juego. |
| **Disparador:** | El **Jugador** selecciona el botón “Crear sala” en la ventana “Inicio”. |
| **Precondiciones:** | No identificadas |
| **Flujo Normal:** | 1. El Juego UNO muestra la ventana “Creación sala” con un formulario para crear una SALA con los campos: Contraseña y Cupo de la sala. Y los botones “Cancelar” y “Crear”. 2. El **Jugador** ingresa la información solicitada en los campos y selecciona el botón “Crear”. (Ver FA 2.1). 3. El Juego UNO verifica que los campos tengan información, de ahí crea la SALA con la información. Después, muestra la ventana “Sala actual” con el ID de la sala, el cupo y los jugadores dentro de la sala. (Ver EX1). 4. Termina caso de uso. |
| **Flujos Alternos:** | FA 2.1 – El Jugador cancela la creación de una sala.   1. El **Jugador** selecciona el botón “Cancelar”. 2. El Juego UNO cierra la ventana “Creación sala” y muestra la ventana “Inicio”. 3. Termina caso de uso. |
| **Excepciones:** | EX1 - El Juego UNO no pudo crear la sala.   1. El Juego UNO muestra la ventana “Error de creación” con el mensaje: “No se pudo crear la sala, por favor intente de nuevo.” Y el botón “Aceptar”. 2. El Jugador selecciona el botón “Aceptar”. 3. El Juego UNO cierra la ventana “Error de creación” y vuelve al paso 1 del flujo normal. |

|  |  |
| --- | --- |
| **Postcondiciones:** | POST-1: El Juego UNO crea una sala con la información ingresada.  POST-2: El Juego UNO muestra los jugadores en la sala. |
| **Incluye:** | No aplica |
| **Extiende:** | No aplica |
| **Prioridad:** | Alta |

Por última, en la siguiente tabla se muestra la descripción al caso de uso CU-04: Unirse a sala.

|  |  |
| --- | --- |
| **ID:** | CU-04 |
| **Nombre:** | Unirse a sala |
| **Autor(es):** | Sammy Guadarrama Chávez |
| **Fecha de creación:** | 12 de octubre de 2020 |
| **Fecha de actualización:** |  |
| **Descripción:** | El jugador se une a una sala mediante el Número de sala, para poder jugar con los otros jugadores que se encuentren en la sala. |
| **Actor(es):** | **Jugador** |
| **Frecuencia de uso:** | Aproximadamente 4 veces por uso del juego. |
| **Disparador:** | El **Jugador** selecciona el botón “Unirse a sala” en la ventana “Inicio”. |
| **Precondiciones:** | No identificadas |
| **Flujo Normal:** | 1. El Juego UNO muestra la ventana “Unirse a sala” con un formulario con el campo: Número de sala y contraseña. Y los botones “Cancelar” y “Unirse”. 2. El **Jugador** ingresa la información solicitada en los campos y selecciona el botón “Unirse”. (Ver FA 2.1). 3. El Juego UNO verifica que los campos tengan información, y verifica que el Número de sala y la contraseña exista, de ahí, muestra la ventana “Sala actual” con la información de la sala y los jugadores actuales. (Ver FA 3.1, FA 3.2 y EX1). 4. Termina caso de uso. |
| **Flujos Alternos:** | FA 2.1 – El Jugador cancela la unión a una sala.   1. El **Jugador** selecciona el botón “Cancelar”. 2. El Juego UNO cierra la ventana “Unirse a sala” y muestra la ventana “Inicio”. 3. Termina caso de uso.   FA 3.1 – Los campos están vacíos.   1. El Juego UNO muestra la ventana “Campos vacíos” con el mensaje: “Uno o los dos campos están vacíos, por favor ingresa la información.” Y el botón “Aceptar”. 2. El **Jugador** selecciona el botón “Aceptar”. 3. El Juego UNO cierra la ventana “Campos vacíos” y regresa al paso 1 del flujo normal.   FA 3.2 – La sala no existe.   1. El Juego UNO muestra la ventana “Sala inexistente” con el mensaje: “La sala a la que intenta ingresar no existe, por favor verifica la información e intente de nuevo.” Y el botón “Aceptar”. 2. El **Jugador** selecciona el botón “Aceptar”. 3. El Juego UNO cierra la ventana “Sala inexistente” y regresa al paso 1 del flujo normal. |
| **Excepciones:** | EX1 - El Juego UNO no pudo unirse a la sala.   1. El Juego UNO muestra la ventana “Error de unión” con el mensaje: “No se pudo unir a la sala, por favor intente de nuevo.” Y el botón “Aceptar”. 2. El **Jugador** selecciona el botón “Aceptar”. 3. El Juego UNO cierra la ventana “Error de unión” y vuelve al paso 1 del flujo normal. |
| **Postcondiciones** | POST-1: El Juego UNO muestra la ventana “Sala actual” con la información de la sala y los jugadores actuales. |
| **Incluye** | No incluye |
| **Extiende** | No extiende |
| **Prioridad** | Alta |

# **Reglas del juego**

El objetivo del juego es ser el primero en deshacerte de todas las cartas de tu mano. Para acumular puntos solo necesitas deshacerte de las cartas de tu mano antes que tus oponentes (Mattel, Inc., 2008).

El juego de consta de 108 cartas, las cuales están divididas en:

* 19 cartas azules - 0 a 9
* 19 cartas verdes - 0 a 9
* 19 cartas rojas - 0 a 9
* 19 cartas amarillas - 0 a 9

Cartas de acción

* 8 cartas Roba 2: 2 azules, 2 verdes, 2 rojas y 2 amarillas.
* 8 cartas Reversa - 2 azules, 2 verdes, 2 rojas y 2 amarillas
* 8 cartas Salta - 2 azules, 2 verdes, 2 rojas y 2 amarillas
* 4 cartas de Comodín de color
* 4 cartas de Comodín Roba 4

En la siguiente tabla se muestra la puntuación de cada carta que se definió que aplicaría al desarrollo:

|  |  |
| --- | --- |
| Tipo de carta | Puntos |
| Cartas con número (0-9) | Valor del número |
| Roba dos | 10 |
| Reversa | 10 |
| Salta | 10 |
| Comodín | 50 |
| Comodín Roba 4 | 50 |

A continuación, se presentan las reglas básicas del juego desarrollado. Para esto, se tomaron las cuentas propias del juego y se modificaron pequeños detalles.

1. A cada jugador se le reparte 7 cartas.
2. Hay una pila de cartas para Descartar y otra para Robar.
3. El sentido del juego se inicia hacia la derecha.
4. Cuando sea turno de algún jugador, debe hacer coincidir una de las cartas de su mano con la carta de hasta arriba de la pila para Descartar, ya sea, por número, color o las cartas de acción).
5. Cada carta de acción tiene las siguientes reglas:
   1. *Carta Roba 2*: cuando se tira esta carta, el siguiente jugador roba dos cartas y NO pierde su turno. Esta carta sólo se puede descartar sobre una carta del mismo color. Si ésta es la primera carta del juego, se sigue la misma regla de juego.
   2. *Carta Reversa*: cambia el sentido del juego. Si le tocara jugar al jugador de la izquierda de quien tira la carta, este jugador pierde su turno; en su lugar jugará la persona de la derecha, y el sentido se cambiará hacia la derecha. Esta carta sólo se puede descartar sobre una carta del mismo color.
   3. *Carta Salta*: al jugador que le toque jugar pierde su turno y tirará el siguiente de acuerdo con el sentido del juego. Esta carta sólo se puede descartar sobre una carta del mismo color.
   4. *Comodín de color*: esta carta se puede tirar con cualquier carta; incluso, puede ser utilizada en caso de tener alguna carta con el número o color correspondiente. El siguiente turno puede tirar una carta de cualquier color.
   5. *Carta Comodín Roba 4*: esta carta hace que el siguiente jugador robe 4 cartas de la pila para Robar. El siguiente turno puede tirar una carta de cualquier color.
6. Si el jugador en turno no tiene ninguna carta que coincida, tiene que robar una carta de la pila para Robar hasta que obtenga una carta del mismo número o color.

# **Prototipos**

En esta sección se presentarán los prototipos creados, estos tienen un enlace directo con las descripciones de los casos de uso anteriormente especificados.

En la siguiente Figura 5 se muestra el prototipo para la ventana “Registrar cuenta”.

Figura . Ventana "Registrar cuenta".

Ahora, se muestra en la Figura 6 el prototipo para la ventana “Registro en espera”.

Figura . Ventana "Registro en espera".

En la siguiente Figura 7 se muestra el prototipo para la vetana “Inicio”.

Figura . Ventana "Inicio".

En la siguiente Figura 8 se muestra el prototipo para la ventana “Verificar código”.

Figura . Ventana "Verificar código".

Ahora, el prototipo para la ventana “Recuperación cuenta” se muestra en la siguiente Figura 9.

Figura . Ventana "Recuperar cuenta".

En la siguiente Figura 10 se muestra el prototipo para la ventana “Creación sala”.

Figura . Ventana "Creación sala".

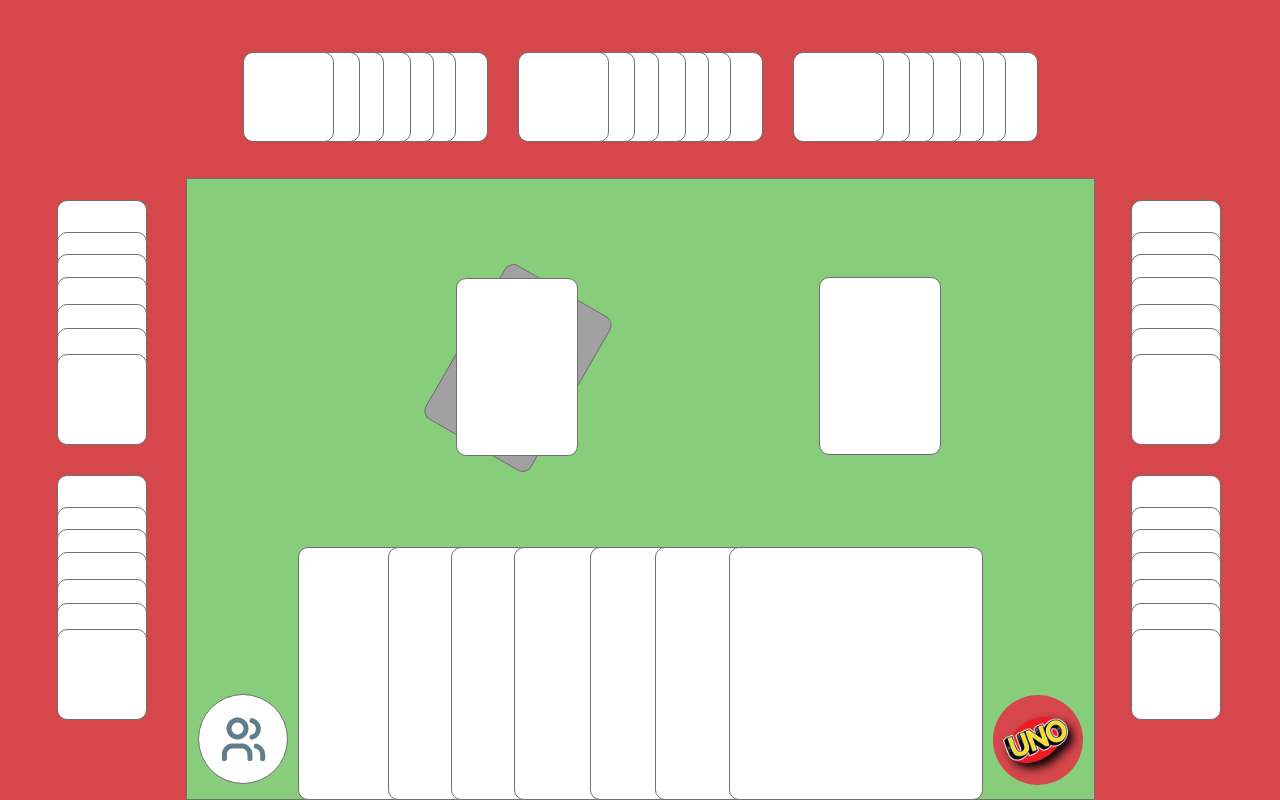
El prototipo para la ventana “Sala actual” se muestra en la siguiente Figura 11.

Figura . Ventana "Partida".

Siguiendo con el prototipo para la ventana “Unirse a sala”, se presenta la siguiente Figura 12.

Figura . Ventana "Unirse a sala".

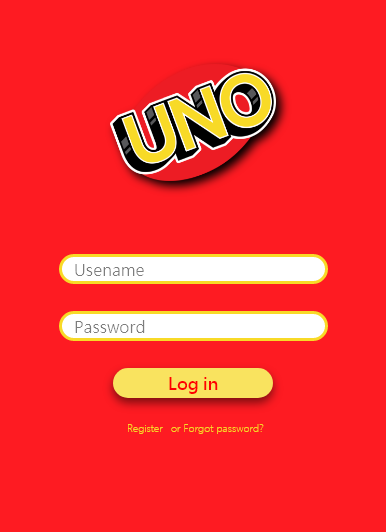
El prototipo para la ventana “Iniciar sesión” se muestra en la siguiente Figura 13.

Figura . Ventana "Iniciar sesión".

Por útlimo, en la siguiente Figura se muestra el prototipo para la ventana “Perfil”.

Figura . Ventana "Perfil".

# **Selección del Framework de desarrollo**

Para el desarrollo del Juego UNO, se utilizaron diferentes Frameworks y tecnologías. En la siguiente tabla se muestran y la justificación de su selección.

|  |  |
| --- | --- |
| Framework / Tecnología | Justificación |
| Windows Communication Fundation | Esta tecnología es ideal para aplicaciones de arquitectura orientada a servicios, donde estas aplicaciones están distribuidas y su comunicación es mediante mensajes.  Dado que esta tecnología se compone por Clientes, que son los que inician la comunicación y Servicios, los que esperan los mensajes de los clientes y responden a ellos, el Juego UNO cumple con estas características. Donde los Clientes (los jugadores que están en la interfaz gráfica) solicitan o envían mensajes sobre las cartas que juegan, si solicitan datos, etc., y los Servicios, de la partida, sala, administración de los jugadores, envían las respuestas a ellos. |
| Windows Presentation Fundation | Crea interfaces de usuario enriquecidas para aplicaciones de escritorio.  Necesario para que del lado de los clientes se pueda crear una interfaz gráfica amigable. WPF nos brinda una forma de crear las interfaces gráficas de manera declarativa, además de poder ir diseñando las ventanas de manera dinámica. |
| Entity Framework | Utilizado para el manejo de los datos del jugador ya que, al ser un mapeador relacional de objetos, permite trabajar con esos datos utilizando objetos ya del dominio. Esto elimina la necesidad de crear código y consultas para acceder a los datos.  Brinda un manejo de los datos en un nivel más alto. |

# **Estándar de codificación**

## **Convención de nombres**

Los nombres para las clases, paquetes, métodos, variables y constantes deben tener un nombre descriptivo con respecto al propósito de estas. Además, dicho nombre no debe de superar más de cuatro palabras. Evitar uso de adverbios y artículos.

Por ejemplo:

Si se desea calcular el área de un **círculo**, los nombres **aceptables** son:

* Clase o Interfaces: *Circle*
* Método: *CalculateCircleArea*
* Variable: *radius*
* Constante: *PI*

Los nombres **NO** aceptados son:

* Clase o Interfaces: *GeometricFigure*
* Método: *cArea*
* Variable: *r*
* Constante: CONST1

## **Formatos o estilos**

### **Identación**

#### **Espacios en blanco**

Utilizar la tecla “Tab” para identar el código en lugar de hacerlo utilizando espacios.

#### **Ajuste de línea**

Cuando una expresión sobrepase el margen del editor de código, no se debe dar ningún salto de línea para aumentar la legibilidad.

### **Comentarios**

#### **Comentarios de línea**

Son utilizados para describir una línea de código en la que las variables o métodos no pueden ser suficientemente descriptivas.

1. Los comentarios de línea deben de ir antes de la expresión que se desea aclarar.
2. Estos deben tener el mismo límite de identación que las líneas de código escritas.
3. La longitud del comentario no debe exceder la longitud del código que se comenta.
4. El uso de estos comentarios NO se debe utilizar al menos que sea totalmente necesario. Simplemente para aclarar el porqué de la línea de código.
5. El comentario debe iniciar con “//”. Por ejemplo:

*//Comentario genérico*

## **Patrones de escritura de código**

### **Declaraciones**

#### **Ubicación**

Ponga declaraciones solo al comienzo de cada bloque. (Un bloque es cualquier código rodeado de llaves "{" y "}"). No espere para declarar variables hasta su primer uso.

Ejemplo:

*void MyFunction()*

*{*

*//inicio del bloque del método*

*int variable1;*

*if(condition)*

*{*

*//inicio del bloque de if*

*int variable2;*

*}*

*}*

La única excepción es para el bucle *for*, que puede ser declarado como:

*for(int i = 0; i < 3; i++)*

#### **Inicialización**

Intente inicializar las variables locales donde se declaran. La única razón para no inicializar una variable donde se declara es si el valor inicial depende de algún cálculo que ocurra primero.

#### **Declaraciones de variables locales**

Se debe de hacer una declaración por línea sin importar que sean del mismo tipo.

Deben seguir el estilo camelCase: Este estilo define que la primera letra de cada palabra a excepción de la primera debe ser mayúscula.

Ejemplo correcto para declarar variables:

*int minusBase;*

*int majorBase;*

Ejemplo **NO** correcto para declarar variables:

*int minusBase, majorBase;*

#### **Declaración de clases e interfaces**

Cuando se codifican clases o interfaces en C#, se debe seguir las siguientes reglas:

1. No incluir espacios entre el nombre de un método y los paréntesis donde se encuentran los parámetros del método.

*//Correcto*

*public void ShowElements(List<Object> myArray)*

*{*

*//code*

*}*

*//Incorrecto*

*public void ShowElements (List<Object> myArray)*

*{*

*//code*

*}*

1. El nombramiento debe seguir el estilo PascalCase: Esta nomenclatura o estilo define que la primera letra de cada palabra debe ser mayúscula. Ejemplo: *IrregularPolygon*.
2. En las clases e interfaces, la llave de apertura debe aparecer una línea abajo de la declaración.

*//Correcto*

*public class Person*

*{*

*//code*

*}*

*//Incorrecto*

*public class Person{*

*//code*

*}*

1. En las clases, la llave de cierre debe comenzar en una línea nueva, alineada verticalmente con la declaración, excepto cuando es una declaración nula, el "}" debe aparecer inmediatamente después del "{".
2. Los métodos son separados por una línea en blanco.

*/\* DECLARACIÓN CORRECTA DE UNA CLASE \*/*

*public class MyClass : Object{*

*int variable1;*

*int variable2;*

*public MyClass(int i, int j){*

*variable1 = i;*

*variable2 = j;*

*}*

*public int DoSomething(){*

*//code*

*}*

*...*

*}*

*/\* DECLARACIÓN INCORRECTA DE UNA CLASE \*/*

*public class myClass : Object{*

*int variable1;*

*int variable2;*

*public myClass(int i, int j){*

*variable1 = i;*

*variable2 = j;*

*}*

*int doSomething(){}*

*...*

*}*

#### **Declaración de return**

Una declaración de retorno con un valor no debe usar paréntesis a menos que hagan que el valor de retorno sea más obvio de alguna manera. Ejemplo:

*return;*

*return vector1.size();*

*return (size ? size : defaultSize);*

#### **Declaraciones de if, if-else, if-else-if-else**

La declaración de un if debe tener la siguiente forma:

*if (condition)*

*{*

*statements;*

*}*

*//Incorrecto*

*if(condition){*

*Statements;*

*}*

La declaración de un if-else debe tener la siguiente forma:

*if (condition)*

*{*

*statements;*

*}*

*else*

*{*

*statements;*

*}*

*//Incorrecto*

*if(condition){*

*statements;*

*}else {*

*statements;*

*}*

La declaración de un if-else-if-else de tener la siguiente forma:

*if (condition)*

*{*

*statements;*

*}*

*else if (condition)*

*{*

*statements;*

*}*

*else if (condition)*

*{*

*statements;*

*}*

*//Incorrecto*

*if(condition){*

*statements;*

*}else if (condition){*

*statements;*

*}*

*else if (condition){*

*statements;*

*}*

Las declaraciones de un if siempre deben de llevar llaves sin importar que solo cuente con una línea de código.

#### **Declaraciones de un ciclo for**

Un ciclo *for* debe tener la siguiente forma:

*for (initialization; condition; update)*

*{*

*statements;*

*}*

*//Incorrecto*

*for(initialization; condition; update)*

*{*

*statements;*

*}*

Cuando use el operador de coma en la cláusula de inicialización o actualización de una instrucción *for*, evite la complejidad de usar más de cuatro variables. Si es necesario, use sentencias separadas antes del ciclo *for* (para la cláusula de inicialización) o al final del ciclo (para la cláusula de actualización).

*//Incorrecto*

*for (int i = 2 – index1 + index2 + index3 - 1; i <= index3; i++)*

*{*

*//code*

*}*

*//Correcto*

*int startPoint = 2 – index1 + index2 + index3 – 1;*

*for (int i = startPoint; i <= index3; i++)*

*{*

*//code*

*}*

#### **Declaración de un ciclo while**

Un ciclo *while* debe tener la siguiente forma:

*while (condition)*

*{*

*statements;*

*}*

*//Incorrecto*

*while(condition){*

*statements;*

*}*

#### **Declaración de un ciclo do-while**

Un ciclo *do-while* debe tener la siguiente forma:

*do*

*{*

*statements;*

*} while (condition);*

*//Incorrecto*

*do{*

*statements;*

*}while(condition);*

#### **Declaración de un switch**

Un *switch* debe tener la siguiente forma:

*switch (condition)*

*{*

*case ABC:*

*statements;*

*//keep going*

*case DEF:*

*statements;*

*break;*

*case XYZ:*

*statements;*

*break;*

*default:*

*statements;*

*break;*

*}*

*//Incorrecto*

*switch(condition){*

*case ABC:*

*statements;*

*break;*

*case DEF:*

*//code*

*}*

Cada vez que un caso no incluye una declaración de *break*, agregar un comentario donde normalmente estaría la declaración de ruptura. Esto se muestra en el ejemplo de código anterior con el comentario “*//keep going”* .

Cada *switch* debe incluir un *default*. El *break* en el *default* es redundante, pero evita un error de caída si luego se agrega otro caso.

### **Espacio en blanco**

#### **Líneas en blanco**

Las líneas en blanco mejoran la legibilidad al activar secciones de código que están relacionadas lógicamente.

Siempre deben usarse dos líneas en blanco en las siguientes circunstancias:

* Entre secciones de un archivo fuente.
* Entre las definiciones de clase e interfaces.

*//Correcto*

*public class Circle*

*{*

*//Code*

*}*

*public class Triangle*

*{*

*//code*

*}*

*//Incorrecto*

*public class Circle*

*{*

*//Code*

*}*

*public class Triangle*

*{*

*//code*

*}*

Siempre se debe usar una línea en blanco en las siguientes circunstancias:

* Entre métodos.
* Antes de un comentario de una línea.
* Entre secciones lógicas dentro de un método para mejorar la legibilidad.

*//Correcto*

*public class Circle{*

*private float radius;*

*public Circle(float radius){*

*//code*

*}*

*public float CalculateCircleArea(){*

*//this is a line comment*

*if (radius < 1)*

*{*

*//code*

*}*

*}*

*}*

*//Incorrecto*

*public class Circle{*

*private float radius;*

*public Circle(float radius){*

*//code*

*}*

*public float CalculateCircleArea(){*

*//this is a line comment*

*if (radius < 1)*

*{*

*//code*

*}*

*}*

*}*

#### **Espacios en blanco**

Tenga en cuenta que **NO** se debe utilizar un espacio en blanco entre el nombre de un método y su paréntesis de apertura.

Los espacios en blanco deben utilizarse en las siguientes situaciones:

* Debe aparecer un espacio en blanco después de las comas en las listas de argumentos.
* Todos los operadores binarios excepto “.” deben estar separados de sus operandos por espacios. Los espacios en blanco nunca deben separar los operadores unarios, como el unario menos, incremento ("++") y decremento ("-") de sus operandos. Ejemplo:

*//Correcto*

*public int DoMathStuff(int number1, int number2){*

*int response;*

*response = (number1 + number2) / 2;*

*}*

*myObject.getName();*

*while (isOpen)*

*{*

*n ++;*

*}*

*//Incorrecto*

*public int DoMathStuff(int number1,int number2){*

*int response;*

*response = (number1+number2) / 2;*

*}*

*myObject. getName();*

*while (isOpen)*

*{*

*n + +;*

*}*

* Las expresiones en una declaración *for* deben estar separadas por espacios en blanco. Ejemplo:

*for (int i = 0; i < 5; i ++)*

*{*

*//code*

*}*

*//Incorrecto*

*for(int i = 0;i < 5;i ++){*

*//code*

*}*

* Los Casts **NO** deben de ir seguidos de un espacio en blanco. Ejemplos:

*public void MyMethod((byte)aNum, (Object)x){*

*//code*

*}*

*//Incorrecto*

*public void MyMethod((byte) aNum, (Object) x){*

*//code*

*}*

### **Referencia a los métodos y variables de clase**

Evitar el uso de un objeto para acceder a una clase estática, variable o método. En su lugar, usa el nombre de la clase. Ejemplo:

*classMethod(); //Correcto*

*Circle.calculateArea(); //correcto*

*Circle circle1 = new Circle();*

*circle1.calculateArea (); //Incorrecto*

### **Asignaciones de variables**

Evita asignar a múltiples variables el mismo valor en una sola línea. Ejemplo:

*int variable1 = variable2 = 5;*

*//Correcto*

*int variable1 = 5;*

*int variable2 = variable1;*

No usar el operador de asignación (=) en un lugar donde puede ser fácilmente confundido con un operador de comparación. Ejemplo:

*if (variable1 = variable2)*

*{*

*…*

*}*

Debe ser escrito así:

*if(variable1 == variable2){*

*…*

*}*

No usar asignaciones anidadas. Ejemplo:

*total = (result1 = variable1 + variable2) + result2*

*//Correcto*

*result1 = variable1 + variable2;*

*total = result1 + resul2;*

### **Uso de paréntesis**

Usar paréntesis en expresiones que contengan operadores mezclados para evitar los problemas de precedencia.

Ejemplo incorrecto:

*if (a == b && c == d)*

*{*

*…*

*}*

Ejemplo correcto:

*if ((a == b) && (c == d))*

*{*

*…*

*}*

## **Manejo de excepciones**

Se usa un bloque *try* para particionar el código que podría verse afectado por una excepción. Los bloques *catch* asociados se usan para manejar cualquier excepción resultante. Un bloque *finally* contiene código que se ejecuta independientemente de si se lanza o no una excepción en el bloque *try*, como la liberación de recursos que se asignan en el bloque try. Un bloque try requiere uno o más bloques catch asociados, o finalmente un bloque, o ambos.

Un bloque *catch* puede especificar el tipo de excepción para capturar. El tipo de excepción debe derivarse de *Exception*. Es importante que se especifique cuál es la excepción que se va a manejar y no dejarla en general (*Exception*).

Ejemplo de un bloque de *try-catch-finally*.

*try*

*{*

*…*

*}*

*catch (Exception ex)*

*{*

*…*

*}*

*finally*

*{*

*…*

*}*

Por ejemplo, si se desea abrir una conexión a la base de datos con la clase MySqlClient, debemos especificar el tipo de excepción que puede ocurrir al hacer esa operación. Por ejemplo:

*public void DataBaseConnection(string infoConnection)*

*{*

*MySqlConnection connection = new MySqlConnection(infoConnection);*

*try*

*{*

*connection.Open();*

*}*

*catch(MySqlException ex)*

*{*

*…*

*}*

*finally*

*{*

*connection.Close();*

*}*

*}*

*//Incorrecto*

*public void DataBaseConnection(string infoConnection)*

*{*

*MySqlConnection connection = new MySqlConnection(infoConnection);*

*try*

*{*

*connection.Open();*

*}*

*catch(Exception ex)*

*{*

*…*

*}*

*finally*

*{*

*connection.Close();*

*}*

*}*

## **Documentación técnica**

Los comentarios de documentación XML son un tipo especial de comentarios que se agregan encima de la definición de un tipo o un miembro definido por el usuario. Son especiales porque los puede procesar el compilador para generar un archivo de documentación XML en tiempo de compilación (Microsoft, 2020).

En este proyecto vamos a hacer uso principalmente de las siguientes etiquetas, sin embargo, los que al menos deberían utilizarse son el número 1 y 5 de la lista posterior.

1. <summary>
2. <returns>
3. <exception>
4. <see>
5. <param>

### **<summary>**

La etiqueta <summary> agrega información breve sobre un tipo o miembro. A continuación, se muestra un ejemplo de cómo hacerlo:

|  |
| --- |
| /// <summary>  /// Descripción de la funcionalidad de la clase  /// </summary> |

### **<returns>**

La etiqueta <returns> describe el valor -\*devuelto de una declaración de método. Esta etiqueta debe ser usada antes de la firma de un método, en esta se especificará el tipo de dato devuelto por el método y en caso de ser necesario una observación por parte del programador. A continuación, se muestra un ejemplo de cómo hacerlo:

|  |
| --- |
| /// <returns>  /// Se espera un dato de tipo TipoDato observación opcional  /// </returns> |

### **<exception>**

Esta etiqueta proporciona una manera de documentar las excepciones que un método puede iniciar. Usado para indicar el surgimiento de una excepción que puede generarse durante el uso de la clase. Cuando se desea documentar una excepción se coloca la etiqueta seguida por el nombre de la excepción. A continuación, se muestra un ejemplo de cómo hacerlo:

|  |
| --- |
| /// <exception>  /// Este método puede retornar una excepción tipo MySqlException  /// </exception> |

### **<see>**

Esta etiqueta permite especificar un vínculo dentro del texto.

Cuando es necesario hacer referencia a otro elemento dentro de la misma clase. Se coloca la etiqueta seguida por el vínculo a la sección referenciada. A continuación, se muestra un ejemplo de cómo hacerlo:

|  |
| --- |
| /// Consulta <see cref=”referencia"/> para más información. |

### **<param>**

Esta etiqueta se usa para describir un parámetro dentro de un método, antes de colocar el nombre del parámetro se pone la etiqueta. A continuación, se muestra un ejemplo de cómo hacerlo:

|  |
| --- |
| /// <param>  ///este metodo usa el parametro project tipo Project  ///</param> |

# **Análisis estático del código**

Para realizar el análisis estático del código del juego, se utilizó la plataforma SonarQube, que es software libre y nos ayudó a obtener métricas para poder mejorar la calidad del código. También, nos apoyamos del SonarScanner para realizar lo anterior.

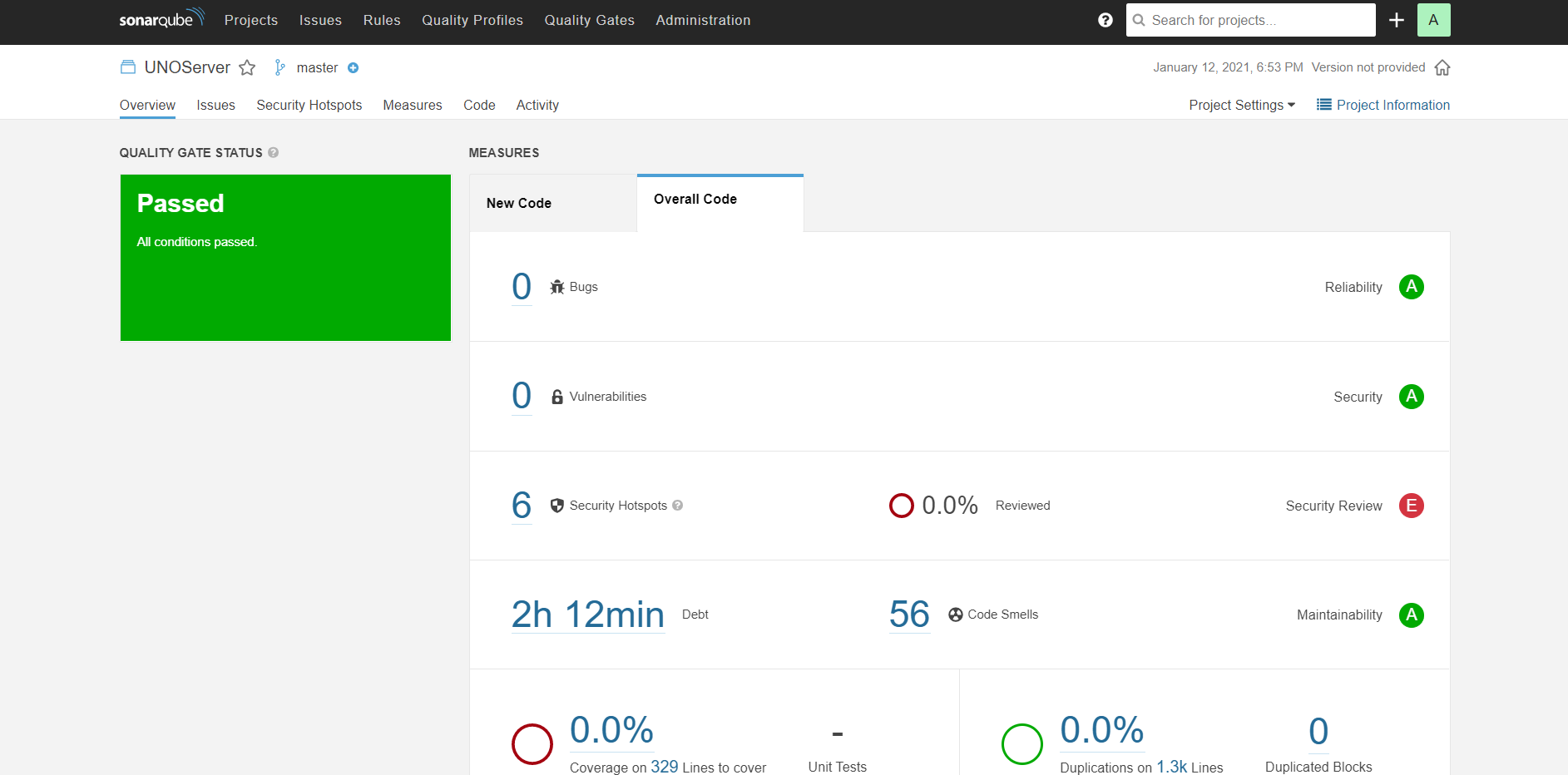
El análisis lo dividimos, uno en el servidor y otro en el cliente. A continuación, en la Figura 15, se muestra el resultado del análisis realizado en el servidor.

Figura . Análisis estático en el servidor.

Por un lado, se resolvieron algunos Code Smells que se presentaron que, generalmente, eran importaciones de algunas bibliotecas. Solo una ocasión se presentó uno relacionado a la complejidad ciclomática pero esto y la mayoría de los code smells se solucionaron. Respecto a los problemas de seguridad, se decidió que, para nuestro caso, no era ningún tipo de riesgo con la información. Esto porque era sobre generar números aleatorios que se usan en la creación de una sala que, desde nuestro punto de vista, no presenta un problema para la seguridad de la información de los jugadores.

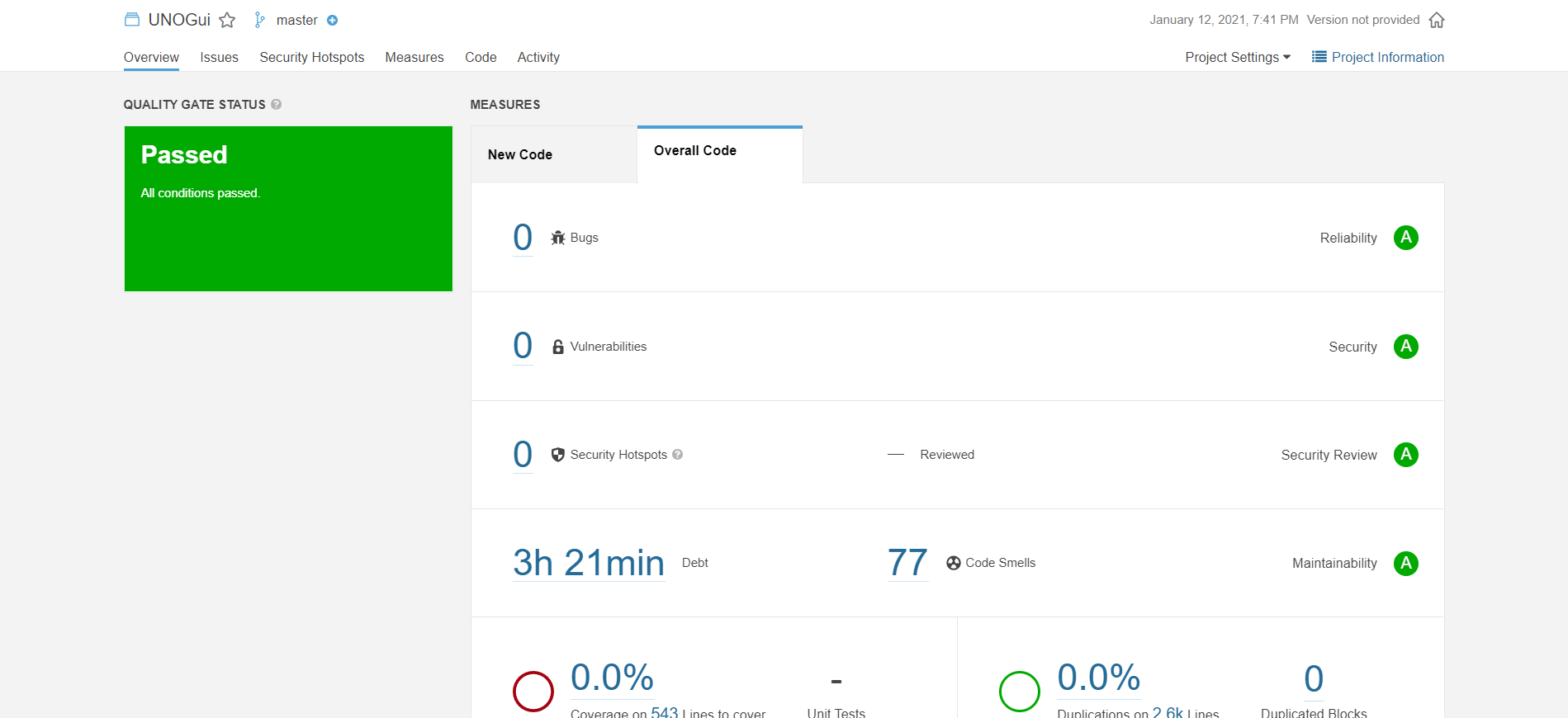
Ahora, en la siguiente Figura 16 se muestra el resultado del análisis realizado al cliente.

Figura . Análisis estático para el cliente.

Como se mostró, los problemas que se presentaron fueron sobre los code smells. Estos también tenían que ver con las importaciones de las bibliotecas y solo en algunos casos se relacionaban con nombres de variables. La mayoría de estos problemas se resolvieron.

# **Conclusiones**

El diseño y desarrollo de este proyecto ayudó a que adquiriera mayor conocimiento y experiencia en las tecnologías que existen para desarrollar un software. En primer lugar, tuve contacto con un Framework que brinda mayor facilidad para el manejo de la información en una base de datos, aprendí nuevos conceptos sobre depuración que me ayudaron a encontrar errores en mi código. De igual manera, me di cuenta sobre la curva de aprendizaje que se puede tener al intentar implementar una tecnología en tu proyecto, esto fue por el uso Windows Communication Fundation para poder realizar el juego. También obtuve mayor conocimiento sobre el desarrollo de las interfaces gráficas con WPF y los conceptos de internacionalización que fueron aplicados en él. Por último, pero no menos importante, el realizar un análisis estático al código que escribí ayudó a darme sobre los errores más comunes que puedo cometer y hasta cómo resolverlos. Sin duda fue un reto desarrollar este juego, pero la recompensa de esto es mayor para un futuro.

-Sammy

Durante el desarrollo del sistema aprendí nuevas herramientas que me podrían ayudar en un futuro, como lo es el uso de WCF (Windows Comunication Foundation) y el como se comporta una aplicación distribuida, esto complementado por las lecciones aprendidas en la clase de diseño me ayudo a ver un panorama mas amplio sobre como trabaja el software en un proyecto real. Además de lo anterior, el proyecto me ayudo a afinar mis conocimientos anteriores como es el manejo de la información y además en esta ocasión este tipo de manejo fue ayudado por un framework así que igualmente tuve que aprender el como se comporta uno y como se deben de seleccionar para futuras ocasiones. Con lo que conforma el desarrollo del proyecto aprendí mas sobre el proceso desde los requerimientos hasta la entrega, ya que durante el desarrollo tomamos en cuenta la adición de herramientas para el análisis de código estático que nos da un reporte con las posibles mejoras y la adición de elementos para internacionalizar el sistema y con esto ampliar más el rango de audiencia de nuestro juego. Por último, pienso que todo lo aprendido durante el proyecto es útil para otros proyectos y que ahora se que hay que tener en cuenta más áreas que solo la de escribir código.

-Angel