UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA EN INFORMÁTICA

INFORME DE C#

PARADIGMA DE PROGRAMACIÓN

Nombre: Cristian Espinoza
Profesor: Miguel Truffa Montenegro
Ayudante: René Zarate
Fecha de Entrega: 12 de diciembre

Santiago de Chile

2 - 2016

TABLA DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE DIAGRAMAS

CAPÍTULO 1. INTRODUCCIÓN

Se continua, con el tema expuesto en el informe anterior, es sabido que los lenguajes de programación han ido evolucionando. Estos cambios radican en la búsqueda de nuevas modalidades de resoluciones de problemas, es así como surge el paradigma orientados a objetos, que basa en la creación de objetos mediante el uso de clases.

En el presente informe se dará a conocer las ventajas y desventajas que se tendrá al momento de abarcar el proyecto, retroalimentando con el desarrollo expuesto en los paradigmas anteriores.

1.1 DESCRIPCIÓN DEL PROBLEMA

El problema consiste en diseñar un juego llamado "*Batalla Naval*", el cual tiene una temática de derribar *ship* enemigos. Se autogenera una matriz de N x M, donde dichas dimensiones son ingresadas por el usuario. El jugador enemigo debe posiciones sus *ships* en N/2 de la matriz y, el usuario en la otra mitad de la matriz.

El principal objetivo es derrotar al enemigo que dicho objetivo se logra al momento de derribar la cantidad total de *ships* que contiene el enemigo u usuario.

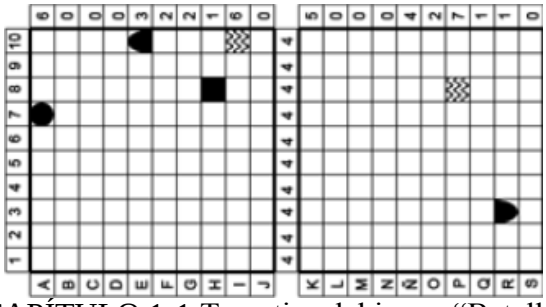

Figura CAPÍTULO 1-1:Tematica del juego "Batalla Naval".

1.2 OBJETIVOS DEL PROYECTO

El objetivo es realizar un programa que represente el juego *"Batalla Naval"* desarrollado en el lenguaje de programación Microsoft *Visual Studio*, ocupando el paradigma de programación orientados a objetos.

Los objetivos del proyecto son implementar ciertas funciones las cuales recaen en las siguientes:

- Realizar una matriz de N x M dimensiones.
- Probar si la matriz creada, es válida para poder comenzar una partida.
- Designar las posiciones de los *ships* del oponente en N/2 x M de la matriz y colocar en la otra mitad un *ship* del usuario en alguna posición ingresada.
- Indicar de forma interactiva la posición que tomara el barco en la matriz.
- Realizar ataques de forma interactiva dentro de la matriz de juego.
- Crear una conexión de la aplicación con Web-Service.

1.3 ASPECTOS DE IMPLEMENTACIÓN

El principal alcance que se presenta es el cumplimiento de las funcionalidades pedidas en el enunciado entregado.

Se llevó a cabo la aplicación en el intérprete "*Microsoft Visual Studio Community 2015 versión 14.0.25431.03 Update 3",* en el lenguaje de programación C#.

La estructura del proyecto se organiza en dos carpetas, una que contiene los modelos y otra con las vistas que contiene la aplicación.

Se importó distintos tipos de bibliotecas, con el fin de poder controlar e solucionar problemas que fueron descubiertos, algunos de fue la conexión a la base de datos implementada y al servidor.

1.4 FUNDAMENTOS TEÓRICOS

El paradigma orientado a objetos se caracteriza por definir los programas en términos de comunidades de objetos. Tener en cuenta que los objetos son entidades que tienen un determinado estado e identidad, objetos con características iguales dentro de la aplicación se agrupan en plantillas, que también son llamas clases.

Los objetos disponen de mecanismos de interacción entre ellos denominados *métodos*, que favorecen el cambio de estado y comunicación entre ellos.

La programación orientada a objetos difiere de la programación estructurada, en la que los datos y procedimientos están separados y sin relación, en cambio, la programación orientada a objetos primero de encarga de definir los objetos y luego mandar mensajes para que realicen sus métodos por sí mismo.

Agregar que algunas de las herramientas que utiliza este paradigma son herencia, cohesión, polimorfismo, acoplamiento y encapsulamiento. Utilizadas para la comunicación y relaciones que se pueden lograr plantear dentro de la aplicación.

Los conceptos de la programación orientada a objetos(POO) tiene origen en *Simula 67,* un lenguaje diseñado para crear simulaciones, creado por *Ole-Johan Dahi y Kristen Nygaard.* La POO se convirtió en un estilo de programación dominante a mediados de los años 1980.

La estructura básica del lenguaje son las clases, que se componen de atributos (variables) y métodos (funciones).

CAPÍTULO 2. DESARROLLO DE LA APLICACIÓN

2.1 PROYECTO

Dentro del siguiente capítulo se le otorgará una visión general de la aplicación, como fue programado y que problemas resuelven sus partes.

Además, se proveerá de un diagrama de clases, el cual se adjuntará dentro la sección de Anexo.

Dentro de la aplicación se ocupó la organización de código denominado como MV (modelo-vistas).

2.1.1 VISTAS

Para la creación de las vistas se ocupa ventanas de tipo "*Windows Form".* Principalmente, esta aplicación consta de vistas encargadas de interactuar con el usuario para poder llevar acabo el desarrollo de la aplicación.

En esta ocasión se cambió la temática de la aplicación, se ocupó dramática de futbol, de tal manera que el proyecto se hace llamar "soccerShips", además de modificar los nombres a:

- Barcos: Los barcos de la aplicación son equipos de futbol, dentro de los cuales se encuentran: "Universidad de chile", "Unión española", "Colo colo", "Universidad católica", etc.
- Disparos: Los disparos dentro de la aplicación se modificaron los goles, es decir, que cada vez que un usuario apunte a un equipo de futbol este se considera como un gol hecho.

Las vistas que se implementaron dentro de la aplicación fueron las siguientes:

- vistaPrincipal: Se encarga de dar la primera impresión al usuario, con distintas opciones que puede realizar dentro de ella, las cuales son: Partida rápida, Ingresar, Créditos, Salir.
- vistaMenu: Vista que aparece al momento de ingresar con un usuario, le permite hacer las siguientes acciones: Jugar y Perfil de usuario.
- vistaIngresar: Vista que permite ingresar a un usuario a la aplicación.
- vistaPrevBatalla: Vista que solicita la información con la cual se va a desarrollar la aplicación.
- vistaTerreno: Vista principal del juego, ya que es donde se realiza todo el desarrollo de la aplicación.

Cada una de las vistas anunciadas anteriormente se pueden observar el diseño otorgado en el anexo del informe.

Tener en cuenta que dentro de C# las vistas también son consideradas clases.

2.1.2 CLASES

Dentro de los aspectos de la programación orientada a objetos, como se mencionó anteriormente las clases es uno de los componentes que llevan el rigor de la aplicación, es donde se almacén los atributos y métodos que se irán a ocupar.

Para el desarrollo de la aplicación se ocupó la siguiente solución que se observa en las siguientes imágenes:

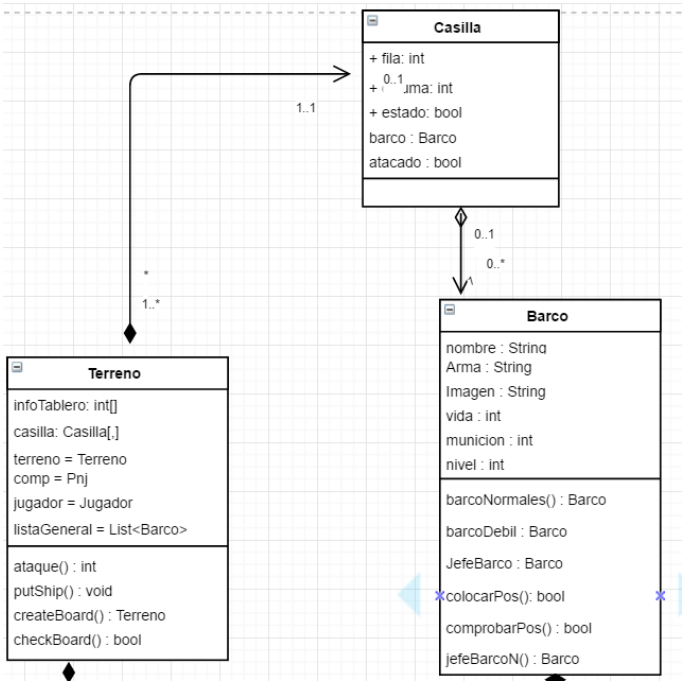

Diagrama de clase 2.1.2-1: Terreno.

Como se observa se tiene que Casilla va a contener de 0 a 1 *ship*, es decir, que la casilla puede estar ocupada por un barco como también existe la posibilidad que este en estado NULL(denominada una casilla vacía).

- Casilla: En esta clase estamos almacenando atributos los cuales contiene la información de cada casilla, se puede decir que el atributo fundamental es de tipo Barco, ya que no indicara si en la casilla actual se posiciona algún *ships.*
- Barco: En esta clase tenemos almacenados todos los tipos de barcos que se encuentran dentro de la aplicación. Además, se tiene almacenado los métodos que:

- Generan los distintos tipos de barco, que son los siguientes:
  - JefeBarco() – JefeBarcoN(): Métodos que se encargan de generar el *ship* jefe del usuario, los cuales tiene un nivel igual a 3.
  - barcoDebil(): Método que se encarga de generar la cantidad solicitada de barco débiles que serán ocupados dentro de la aplicación, los cuales tiene un nivel igual a 1.
  - barcoNormales(): Método que se encarga de generar la cantidad solicitada de barco normales que serán ocupados dentro de la aplicación, los cuales tiene un nivel igual a 2..
- Método que otorga al *ship* las posiciones que ocupara dentro de la matriz enemiga, además de verificar si las posiciones asignadas están disponibles para poder posicionarlo, los métodos son los siguientes:
  - comprobarPos(): Método que se encargan de generar y asignar las posiciones que ocuparan los *ships* del enemigo dentro de la matriz de juego.
  - colocarPos(): Método que se encarga de verificar si la posición asignada se encuentra disponible para posicionar un *ship.*

Por otra parte, tenemos la clase Terreno que estará compuesta por 1 a muchas (1..*) Casillas, esto se produce porque la matriz implementado es de tipo Casilla, dicha solución se implementó así, con el fin de poder tener un mejor manejo de información dentro de cada una de las casillas, lo cual permitirá implementar soluciones factibles y eficientes.

- Terreno: En esta clase tenemos almacenado el arreglo bidimensional de Casillas el cual será la matriz que se crea dentro de la aplicación, además debemos tener en cuenta que contiene los métodos principales de la aplicación que son los siguientes:
  - createBoard(): Método que se encarga de la creación de la matriz de juego según las dimensiones que sean ingresadas por él usuario.

    Recibe como parámetros lo siguiente:
    - Terreno: Esta será la variable donde almacenaremos la matriz que se creará.
    - VistaTerreno: Esto permite ir agregando cada objeto casilla que se vaya creando a la vistaTerreno específicamente al panel que se encuentra en ella.
  - PutShip(): Método que se encarga de posicionar los barcos del usuario en la casilla que sea seleccionada (En esta funcionalidad se ocupa programación orientada a eventos).

Recibe como parámetro lo siguiente:

- Fila y columna: Coordenadas de donde se desea posicionar el *ship*.
- Estado: Esta es una variable booleana que nos entrega la información si la casilla se encuentra ocupada o no.
- Terreno: Variable donde se encuentra la matriz.
- Barco: *Ship* que se desea colocar en la matriz.
- Este método tiene un procedimiento de la siguiente manera:
  - Verifica si la coordenada seleccionada por el usuario pertenece al terreno de él.
  - Verifica si el estado de la casilla está disponible para poder colocar un *ship* en dicha posición.
  - Luego verifica el nivel del *ship* y procede a colocarlo dentro de la matriz de juego.

- CheckBoard(): Método que se encarga de verificar si la matriz creada está apto para poder realizarle la batalla.

Recibe como parámetro un terreno, con el cual procede a verificar si cumple condiciones para poder ser creada la matriz.

- Ataque(): Método que se encarga de realizar el ataque a algún *ship* seleccionado por el usuario. Se debe mencionar que es el encargado de dar la dramática a la aplicación.

  Recibe como parámetro lo siguiente:

  - CasillaMia: Casilla seleccionada de donde se encuentra el *ship* que va a realizar el ataque, esto nos permite acceder a la información del barco (nivel, munición).
  - CasillaEnem: Casilla seleccionada de donde se quiere efectuar el ataque, nos permite verificar si existe algún *ship* en dicha ubicación, es ese caso podemos acceder a la información de dicho barco para verificar su situación final después del ataque.
  - Terreno: Variable donde se encuentra almacenada la matriz, además es donde se va efectuando cada ataque que se realiza.
  - Este método permite realizar el ataque tanto del usuario como de la computadora, tiene un procedimiento de la siguiente manera:
    - Primero verifica si la casilla seleccionada se encuentra en el terreno del adversario.
    - Luego verifica que la casilla seleccionada contenga un barco con el cual se va a realizar el ataque al oponente.

- ❖ En seguida de eso, verifica que la casilla seleccionada para efectuar el disparo, sea una casilla enemiga.
- ❖ Terminado el proceso anterior, procede a realizar el ataque de la computadora, que está programada para que ataque a dicho con una *ship* al azar que se encuentra dentro de su terreno a una posición random que se autogenera, además de realizar las comprobaciones respectivas.

Por otro parte en la siguiente imagen 2.1.2-2 se puede observar que desde la clase Usuario heredan las Clases Pnj y Clase Personaje, además se observa que Usuario puede contener de 1 a muchos (1..*) Barcos, además de estar contenido por un clase Terreno.

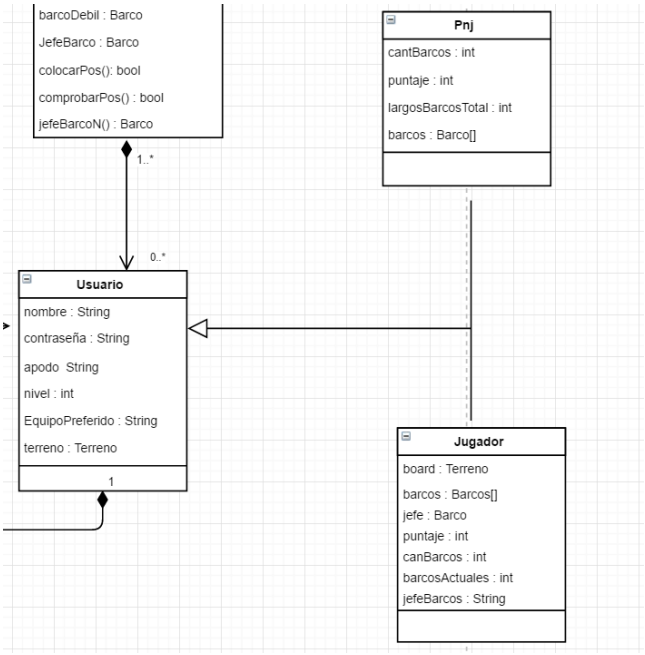

Diagrama de clase 2.1.2-2: Usuario.

- Usuario: En esta clase tenemos almacenado ciertos atributos que son importante para el trascurso de la aplicación, uno de ellos es terreno, ya que se podrá guardar la matriz creado en los atributos de cada usuario, además de poder acceder de manera mucho más factible a la información de cada uno de las casillas y *ships*.

Y como se mencionó anteriormente tenemos que desde usuario heredan dos clases las cuales son:

- Jugador: Esta clase como se mencionó hereda de clase Usuario, además de contener atributos propios de la clase que identifican a un objeto jugador dentro

de la aplicación, estos permiten poder guardar información que se va obteniendo durante la partida (puntajes, *ships* derrotados, etc.).

- Pnj: Esta clase nos permite almacenar los barcos que tendrá en la matriz el enemigo, al mismo tiempo esto nos permite saber la ubicación de cada uno de ellos y las vidas que le van quedando al ir transcurriendo la partida.

2.1.3 BASE DE DATOS Y WEB SERVICE

Dentro de la aplicación, la base de datos implementada actúa como un almacenamiento de datos del usuario, ya que dentro de ella se guarda el nombre, contraseña, apodo, equipoPreferido y puntaje del usuario. Para poder acceder a la información que contiene la base de datos se ha hecho un mediador, el mediador que se ocupa es un proyecto externo al juego denominado Web-service.

Este proyecto externo es el encargado de conectar la base de datos con el juego "SocerSips", esto es posible porque se importar el proyecto "Web-service" y se utiliza como si fuera una clase más de la aplicación, entonces al instanciarlo se genera un objeto de tipo Web-service, el cual contiene métodos que permite hacer consultas a la base de datos.

Lo mencionado anteriormente se puede ver relejado en la siguiente imagen:

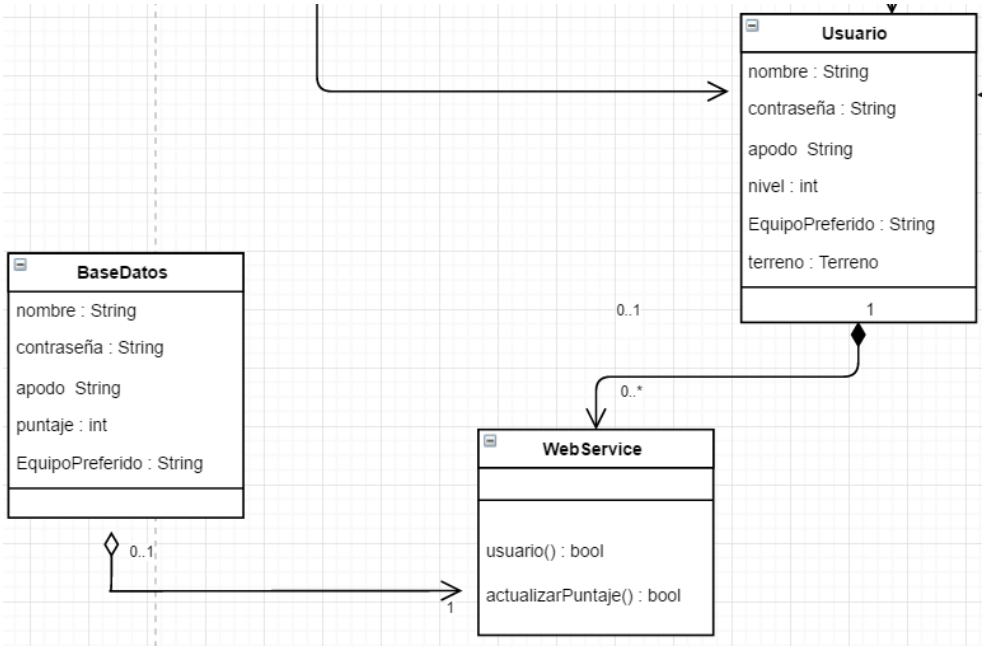

Diagrama de clase 2.1.3-3: Base de datos y web-service.

Por último, agregamos que los métodos que contiene web-service, están relacionados con el usuario, es decir, la implementación dada va adaptada y pensada en el usuario. Los métodos son los siguientes:

- ActualizarPuntaje(int puntaje, string usuario) : Como se puede apreciar recibe como parámetros al usuario que se le va a realizar la modificación del Score (puntaje), esto se hace de manera asincrónico como fue solicitado en el enunciado del proyecto.
- registrarUsuario(string nombre, string contrasena, string apodo, string equipoPreferido) : Este método nos permite registrar a un nuevo usuario en la aplicación, con el fin de poder guardar su puntaje al momento de que decida jugar una partida, ya que en caso de no estar registrado el puntaje no es guardado dentro del juego.
- usuario(string nombre, string contraseña) : Este método nos permite verificar si el usuario al ingresar a la aplicación esta registrado dentro de ella.
- equipoUs(string nombre), puntajeUs(string nombre) y apodoUs(string nombre);: Sirven para obtener el equipo ,puntaje y apodo que tiene el usuario en la base de datos.

2.1.4 ANALISIS DE RESULTADOS

Los resultados de esta aplicación se pueden observar de mejor forma que los laboratorios anteriores, ya que se implementó un interfaz gráfica.

A continuación, se muestra algunos de los resultados que otorga la aplicación:

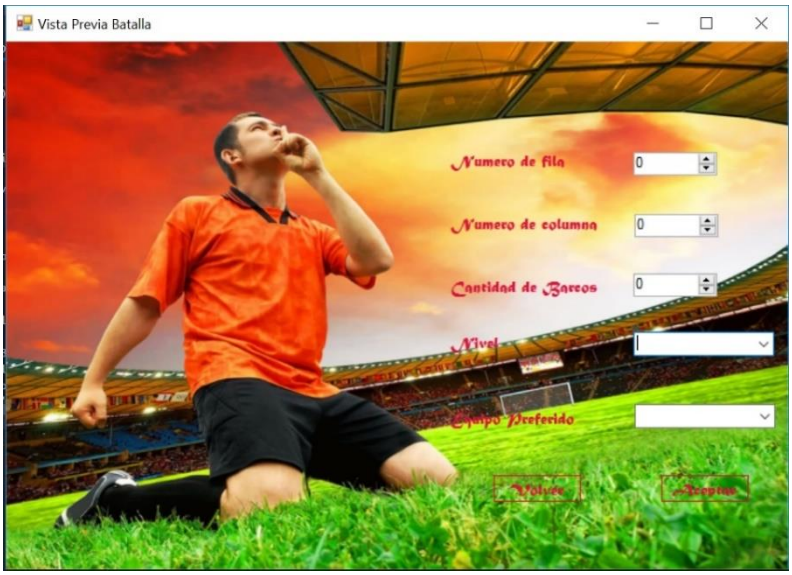

Figura 2.1.3-2: Figura de selección de datos

Como se puede apreciar, en la imagen 2.1.4-2 se da la opción al usuario de poder ingresar las dimensiones que desea crear la matriz, además de ingresar la cantidad de barcos y nivel con el cual desea jugar.

Por último, se puede apreciar que se da la opción de escoger su equipo preferido que pasa a ser su *ship jefe.*

Como resultados de lo anterior, al momento de ingresar la totalidad de sus datos y de forma correcta, se abre la siguiente vista donde se realiza la batalla, la cual se puede observar en la imagen 2.1.4-3.

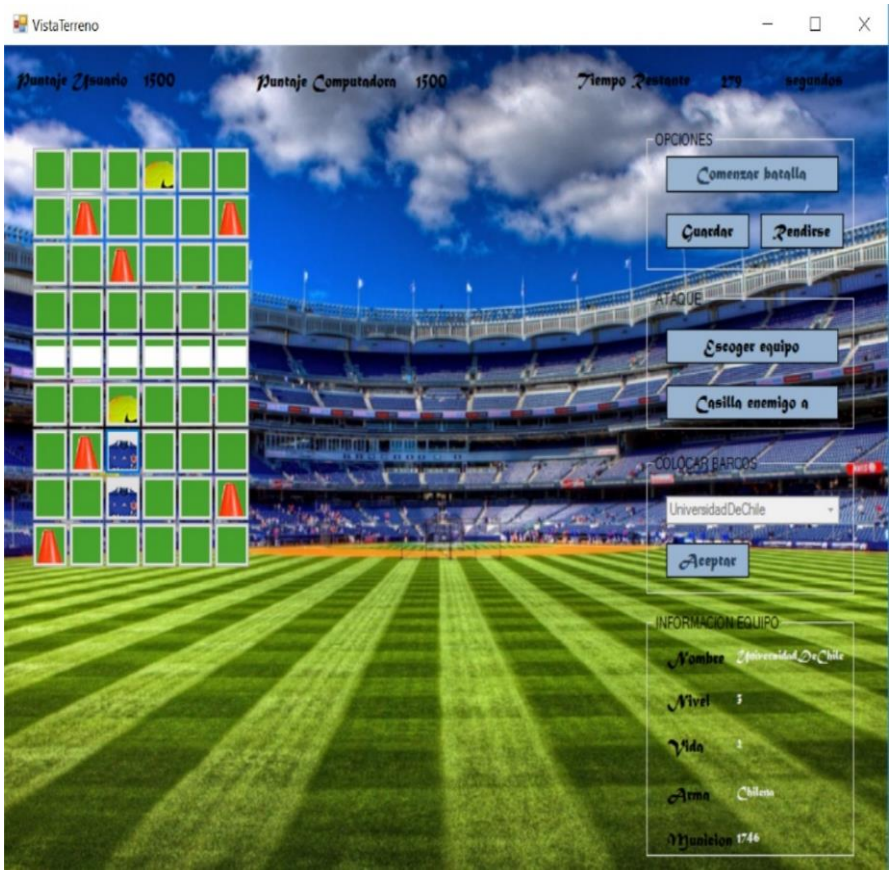

Figura 2.1.4-3: Vista de batalla.

Dicha vista es la que se encarga de mostrar la mayor cantidad de resultados al jugador.

1. Muestra cómo se va actualizando la matriz de juego, con un cono si el disparo fue fallido o con una pelota si el disparo fue exitoso.
2. Se puede observar la información de cualquier *ships* que se encuentre en la matriz (parte del usuario).
3. Se puede observar el puntaje que se obtiene al termino de todo ataque.
4. Permite hacer acciones al jugador, como se puede apreciar en cada uno de los botones que contiene la vista.

Se debe agregar que cada uno de las operaciones que va realizando se va monitoreando con un mensaje que es entregado en la vista, con el fin que pueda tener un grato trato con la aplicación.

Por último, se puede agregar que cada uno de los resultados de la aplicación son exitosos se logra realizar cada uno de lo solicitado en el enunciado entregado.

CAPÍTULO 3. CONCLUSIÓN

Para finalizar, la mayor dificultad dentro del desarrollo de la aplicación fue la planificación en base al nuevo paradigma, como se mencionó anteriormente en los laboratorios pasados el mayor obstáculo que se encontré siempre fue el mencionado. Una vez adaptándose a la manera de buscar las soluciones la dificultad disminuye.

Luego de haber terminado la aplicación se puede observar las facilidades que otorga el paradigma orientado a objetos, en base a eso uno se pude dar cuenta que es el mejor paradigma para recrear un juego en comparación a los paradigmas desarrollados anteriormente: Paradigmas imperativo, funcional y lógico.

Uno del gran plus que tiene el paradigma orientado a objetos es el diagrama de clases que aliviana bastante el trabajo de buscar soluciones, ya que permite relacionar las clases con lo necesario para poder lograr la solución de dicho problema, además de ayudar a la creación del código mediante que ya se tiene uno idea de las relaciones y soluciones que se van ir otorgando a cada uno de los sub-problemas que aparecen.

Como se mencionó en la sección de análisis de resultados, se ha podido entregar una aplicación la cual satisface la totalidad de requerimientos funcionales como no funcionales, en esa oportunidad no se logró desarrollar algún requerimiento extra debido al poco tiempo con el que se constaba.

Por último, se debe decir que el paradigma orientado a objetos es una herramienta poderosa para crear soluciones a los problemas, tanto como su planificación como por su codificación, además agregar que tiene un grato trato con el usuario.

CAPÍTULO 4. REFERENCIAS

Alvarez, M. A. (1999). *Qué es la programacion orientada a objetos.* -: -.

Arias, J. C. (-). *Creación de conexión con SQL Server y Visual C# mediante Web Service.* -: -.

CAPÍTULO 5. ANEXO

Se adjunta el diagrama de clases de la aplicación:

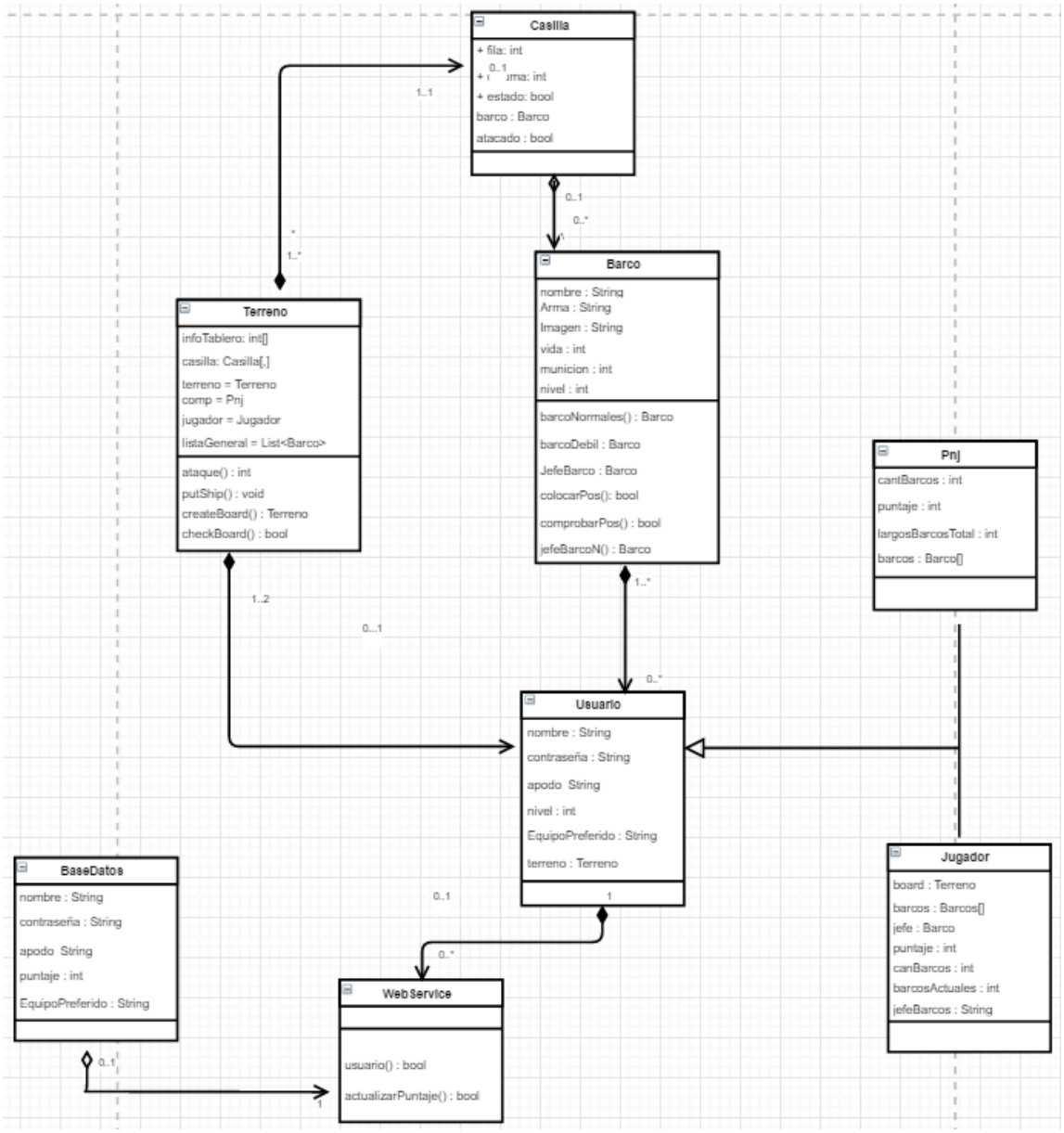

Diagrama de clase CAPÍTULO 5-4: Diagrama de clases de la aplicación.

Se adjuntan las vistas que contiene la aplicación:

Figura CAPÍTULO 5-4: Vista Principal.

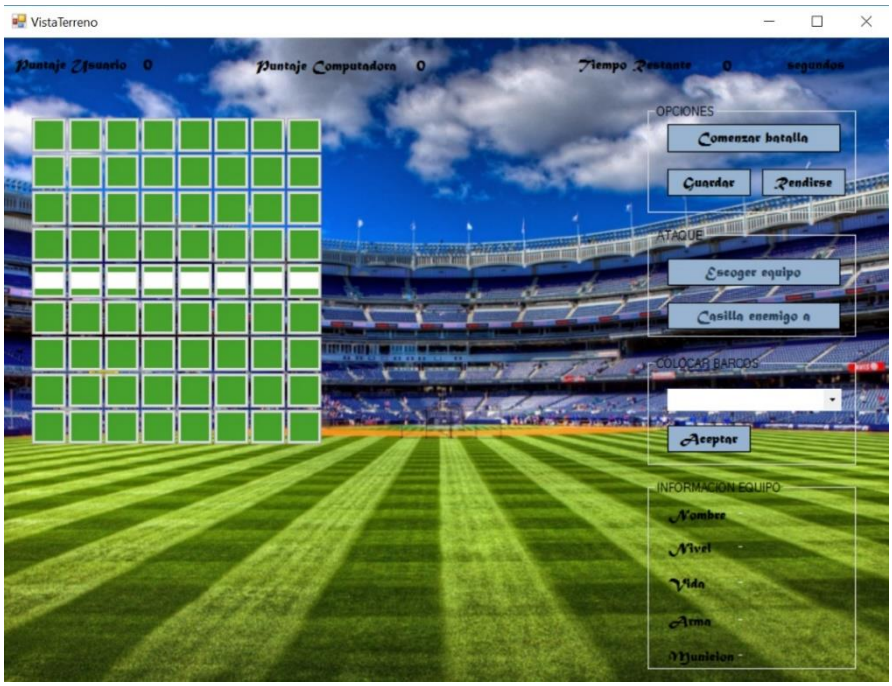

Figura CAPÍTULO 5-5: Vista Terreno.

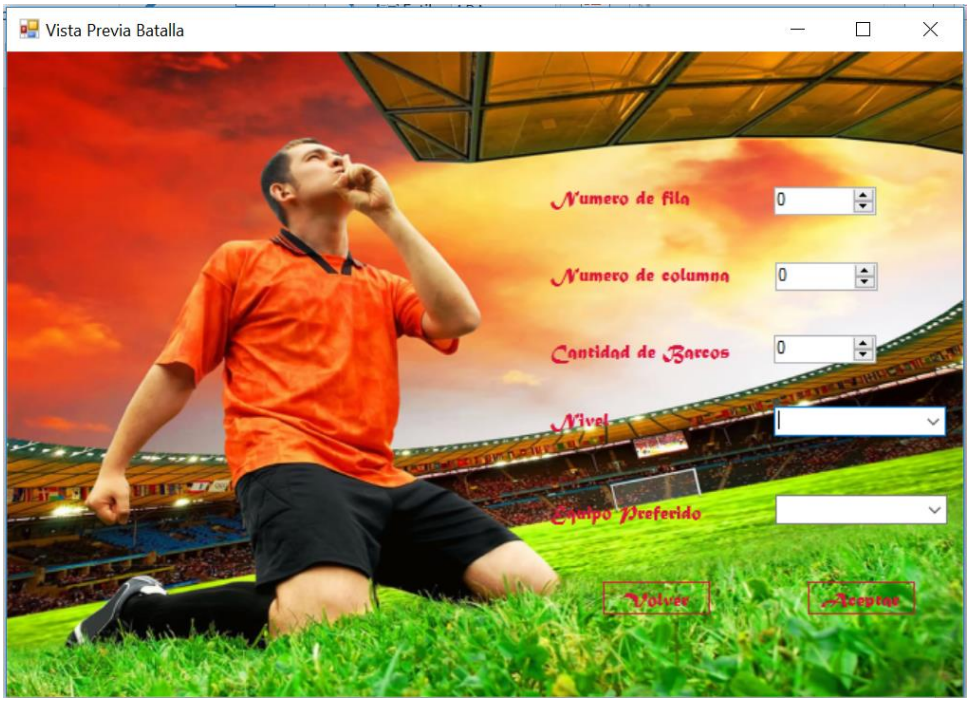

Figura CAPÍTULO 5-6: Vista Previa Batalla.

Figura CAPÍTULO 5-7: Vista Ingresar.

Figura CAPÍTULO 5-8: Vista Login.