

2. Manual de Usuario

Batalla Campal se juega de 2 a 10 jugadores, cada uno introduce de 2 a 10 soldados en un tablero tridimensional. Se establece un tamaño de mapa, que va de 2x2 a 1000x1000 casilleros y una altura que puede ir de 2 a 20 niveles de casilleros. Una vez echo esto se elige una clase de mapa predeterminado, entre los 4 posibles, introduciendo un numero del 1 al 4. A modo de ejemplo, el juego inicia solicitando los datos previamente descriptos:

```

1 Bienvenidos, ingrese la cantidad de jugadores: 2
2 Ingrese la cantidad de soldados por jugador con la que van a jugar: 2
3 Ingrese el largo del mapa: 10
4 Ingrese el ancho del mapa: 10
5 Ingrese el alto del mapa: 3
6 En que mapa desea jugar? (1 - 4): 3

```

Luego de determinar todas las características del mapa, se crearan archivos de extension "BMP", uno por cada nivel de altura, para cada jugador, los cuales se actualizan antes de cada accion (como puede ser mover un soldado o lanzar un misil). Cada jugador ve solo sus propias unidades en el mapa.

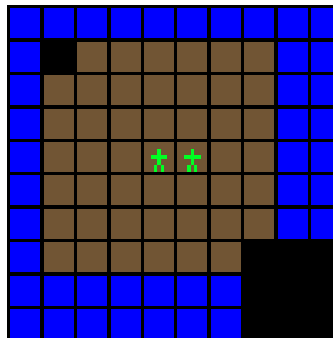


Figura 1: Dos soldados en plena batalla

Existe un mazo general, el cual contiene una cantidad de cartas igual a 4 veces la cantidad de jugadores en el juego. Las cartas del mazo estan mezcladas de forma aleatoria entre los 6 tipos de cartas que hay.

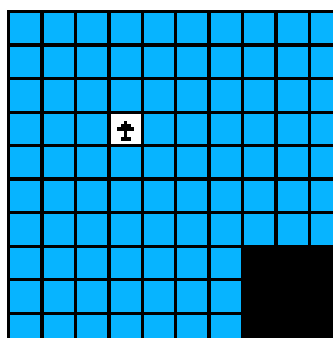


Figura 2: Avión de batalla en las alturas

Las posibles **Cartas** a levantar del mazo son:

- **Misil:** carta que agrega un misil adicional en el próximo turno al jugador que la obtenga. El area de daño de un misil es de 3x3x3 casilleros. Matando a las unidades que alcance e inhabilitando los casilleros.

- **Nuevo avión:** permite colocar un avión únicamente en un casillero que sea de aire, una vez colocado no se podrá mover. Tener un avión permite disparar 2 veces por turno.
- **Nuevo Barco:** permite colocar un barco, solamente en un casillero cuyo terreno sea agua, una vez colocado no se podrá mover. Tener un barco permite disparar un misil una vez por turno.
- **Somnífero:** el siguiente jugador pierde su turno
- **Francotirador:** permite realizar un disparo extra de forma inmediata
- **Harakiri:** aniquila automáticamente a una de las unidades del jugador que ha sacado la carta

Los casilleros del tablero pueden encontrarse en 3 estados:

- **Vacio:** significa que no hay unidades en él y está habilitado para ser ocupado con una unidad.
- **Ocupado:** indica que hay una unidad en él.
- **Inhabilitado:** no se pueden colocar unidades en él. Un casillero se inhabilita luego de un disparo, o una pelea a muerte entre 2 soldados o un ataque de misil. En el mapa se ve de color negro.

Gana el último jugador que tenga soldados en el mapa. Puede ocurrir un empate en caso de que los jugadores se queden sin soldados al mismo tiempo.

3. Manual de Programador

Para el desarrollo de este juego se ha utilizado memoria dinamica. La clase que maneja y se encarga de la lógica del juego es la clase Partida. Esta clase posee a su vez punteros a objetos como han de ser el tablero, los jugadores y el mazo de cartas. El tablero se ha implementado a través de una lista de listas de lista a objetos de tipo Casillero. Esto es, dando una forma tridimensional al tablero, donde cada nodo de las listas son punteros a Casilleros.

Para el manejo de los jugadores en la partida se ha decidido implementar un arreglo de punteros a Jugador, así como también para la implementación del mazo de cartas del juego.

Cada jugador posee cierta cantidad de unidades las cuales pueden ser variables a lo largo del juego, por lo cual, se ha decidido implementar una lista de unidades para tal fin. Para diferenciar cada tipo de unidad, se ha declarado un tipo enumerativo

Para el manejo del posicionamiento de las unidades en el tablero, cada nodo en las listas del tablero posee un puntero a Casillero, el cual posee un puntero tanto a Unidad como a Jugador y una Coordenada, de esta forma se determina a quien pertenece la unidad que se sitúa en el casillero. Asimismo cada objeto de tipo Unidad posee un puntero a una Coordenada, de forma tal que se puede acceder a un Casillero a partir de una Unidad de un jugador, así como también, se puede acceder a una unidad a partir de un Casillero en particular.

Para la creación del tablero se debe especificar las medidas correspondientes de largo, ancho y alto, así como también uno de los cuatro mapas que se han desarrollado

```
1 Tablero::Tablero(unsigned int largo, unsigned int ancho, unsigned int alto, int nroMapa)
```

Para la entrada de datos por consola se han implementado los siguientes métodos de la clase Partida, los cuales se encargan de la validación de los datos de entrada a lo largo de todo el programa:

```
1 void Partida::pedirDatos(unsigned int& mapaLargo, unsigned int& mapaAncho, unsigned
   ↪ int& mapaAlto)
2 unsigned int Partida::ingresarNumeroYValidar(int minimo, int maximo)
```

Dentro de la clase Partida existen dos métodos importantes que instrumentan la lógica del juego, delegando las tareas a otros métodos, primeramente a saber:

```
1 void Partida::inicializarPartida()
```

Se encarga del pedido de datos de entrada para inicializar el tablero con las medidas indicadas, así como también de establecer tipo de terreno de cada Casillero, de la creación del arreglo de punteros a Jugador, de la creación del mazo de cartas de forma aleatoria, y de solicitar y gestionar el posicionamiento los soldados iniciales de cada jugador. El segundo método importante es:

```
1 void Partida::siguienteTurno()
```

El cual se encarga del manejo de turnos, en los cuales cada jugador puede realizar los disparos que tiene disponibles, lanzar misiles, mover unidades y sacar una carta del mazo de cartas, en cada etapa se exporta el tablero para que el usuario pueda ver en las correspondientes imagenes bitmap de su tablero, escondiendo las unidades del enemigo. Asimismo se chequea luego de cada acción si es que las condiciones de finalización del juego se han terminado.

```
1 void Partida::siguienteTurno(){
2     unsigned int nroJugadorEnTurno = this->getTurno() % this->getCantidadJugadores();
3     cout << endl << "[***]\tTurno del Jugador #" << nroJugadorEnTurno;
4     this->realizarDisparosJugador(nroJugadorEnTurno,
   ↪     this->jugadores[nroJugadorEnTurno]->getCantidadDisparosDisponibles());
```

```

5     if(this->haTerminado() == true){
6         return;
7     }
8     this->lanzarMisilesJugador(nroJugadorEnTurno,
9     ↪ this->jugadores[nroJugadorEnTurno]->getCantidadMisilesDisponibles());
10    if(this->haTerminado() == true){
11        return;
12    }
13    this->moverUnidad(nroJugadorEnTurno);
14    if(this->haTerminado() == true){
15        return;
16    }
17    this->sacarCartaDelMazo(nroJugadorEnTurno);
18    if(this->haTerminado() == true){
19        return;
20    }
21    this->exportarTablero(nroJugadorEnTurno);
22    this->turno++;
23 }

```

Esta condición de finalización está dada por la cantidad de soldados que los jugadores tienen. Si existe como máximo un jugador con soldados, entonces el juego ha terminado. Esto es, porque existe sólo un jugador con soldados, o los soldados de todos los jugadores han muerto.

```

1 bool Partida::haTerminado(){
2     if(this->getCantidadJugadoresConSoldados() <= 1){
3         cout << endl << "El juego ha terminado!";
4         return true;
5     }
6     return false;
7 }

```

En caso de que todos los soldados de un jugador hayan muerto, el resto de las unidades (aviones y barcos) huyen con gran temor del campo de batalla por medio del siguiente método

```

1 void Partida::jugadorEmprendeRetirada(unsigned int nroJugador){
2     if(this->jugadores[nroJugador] == NULL){
3         return;
4     }
5     Unidad* unidad = this->jugadores[nroJugador]->buscarUnidad(1);
6     Coordenada* coordenada;
7     Casillero* casillero;
8     while(unidad != NULL){
9         coordenada = unidad->getPosicion();
10        if(!coordenada){
11            throw "ERROR EN COORDENADA AL EMPRENDER RETIRADA";
12        }
13        casillero = this->tablero->getCasillero(coordenada->getLargo(),
14        ↪ coordenada->getAncho(), coordenada->getAlto());
15        if(!casillero){
16            throw "ERROR EN CASILLERO AL EMPRENDER RETIRADA";
17        }
18        this->jugadores[nroJugador]->removeUnidad(unidad);
19        casillero->inhabilitar();
20    }
21 }

```

```

19     }
20     cout << endl << "Al morir todos sus soldados, las unidades del jugador #" <<
    ↪     nroJugador << " han emprendido retirada";
21 }

```

Para la generación de archivos de tipo BMP, se ha implementado el siguiente método:

```

1 void Partida::exportarTablero(unsigned int nroJugador);

```

El cual a partir del número de jugador indicado, recorre completamente todos los casilleros del tablero, indicando aquellos que están inhabilitados, y en caso de que estén ocupados por unidades enemigas, se muestran al jugador como si estuviesen vacías, esto es, mostrando el tipo de terreno del casillero.

En más detalle, primeramente crea todos los modelos BMP a ser utilizados (tablero, unidades y terreno). Lee todo el tablero, casillero por casillero en ancho, largo y alto mediante el método **Tablero::getCasillero()** que devuelve un puntero a casillero.

Con ese puntero a Casillero se obtienen todos los datos del casillero mediante sus diferentes métodos, estado mediante **getEstado()**, terreno mediante **getTipoDeTerreno()** y la coordenada de este mediante **getCoordenada()**.

Una vez que lee el estado del casillero, si este está inhabilitado, copia el modelo de terreno inhabilitado a ese casillero mediante **copiarBMPAlMapa()**. Si el casillero está ocupado, se revisa que tipo de unidad es la que ocupa ese casillero y mediante **copiarBMPAlMapa()** se agrega su modelo correspondiente al BMP del tablero.

En caso de que el casillero se encuentre vacío, se determina qué tipo de terreno posee el casillero y luego mediante **copiarBMPAlMapa()** se agrega el modelo correspondiente al mapa.

El BMP del tablero se exporta finalmente mediante el método **WriteToFile()** perteneciente a la librería **EasyBMP**. El nombre de los diferentes archivos BMP que se exportan (1 por cada nivel de altura del mapa) va a depender del jugador que sea, se realiza una conversión del número del jugador y del nivel del mapa, a tipo de dato char, para poder utilizarlo como parámetro al método **WriteToFile()**, los archivos van tener nombre del tipo "jugador<NumeroJugador>altura<NumeroNivelAltura>.bmp".

Para la implementación de los distintos tipos de casilleros se ha creado una nueva librería o header llamada **visuales.hpp**. La cual posee las siguientes funciones:

```

1 void dibujarCasilleros(BMP &Tablero);

```

Procedimiento que se encarga de dibujar las líneas que separan cada uno de los casilleros. Lo hace cada 10 píxeles, por lo tanto cada casillero mide visualmente 9x9 píxeles. Este procedimiento se vale de dos sub-funciones llamadas **dibujarCasillerosHorizontal()** y **dibujarCasillerosVertical()**, que cada uno como su nombre indica se encarga uno de dibujar las líneas horizontales y las verticales respectivamente.

El siguiente procedimiento se encarga de establecer los valores básicos del tablero BMP.

```

1 void crearBMPTablero(BMP &TableroVisual,unsigned int ancho,unsigned int largo);

```

Establece su tamaño y dibuja sus casilleros mediante **dibujarCasilleros()**. Para pasar de los valores de ancho y largo que recibe por parámetro a su adaptación en píxeles. Se pasan esos parámetros a la función **adaptarMedidas()**, que multiplica ambos parámetros por 10. Una vez adaptados sus valores, se los asigna al BMP del tablero mediante **establecerMedidas()**, que a su vez se vale al método **setSize()** de **EasyBMP** para hacerlo.

```
1 void pintarArea(BMP &Tablero,int xInicial,int xFinal,int yInicial,int yFinal,int
  ↪ r,int g,int b);
```

Como su nombre lo indica, esta función se encarga de pintar un área especificada de un objeto BMP del color designado mediante sus parámetros. Pinta desde xInicial a xFinal ,y desde yInicial hasta yFinal, del color que se forme de la combinacion de los parametros r,g y b, que se le asignan a cada pixel dentro del area especificada al objeto BMP.

```
1 void crearBMPUnidades(BMP &Soldado,BMP &Barco,BMP &Avion);
```

Esta función se encarga de asignar a todos los BMP de las unidades sus respectivos modelos de 9x9 pixeles, lo hace mediante las funciones **crearBMPSoldado()**, **crearBMPAvion()** y **crearBMPBarco()**.

```
1 void crearBMPTiposTerreno(BMP &Tierra,BMP &Agua,BMP &Aire,BMP &TInhabilitado);
```

Procedimiento que se encarga de asignarle a todos los BMP que son para los casilleros vacios, sus respectivos modelos de 9x9 pixeles cada uno. El terreno de un casillero puede ser del tipo:

- Agua (Color azul)
- Tierra(Color marron)
- Aire(Color celeste)
- Terreno inhabilitado (Color negro)

Esto es realizado mediante las funciones **crearBMPAgua()**, **crearBMPAire()**, **crearBMPTierra()** y **crearBMPTInhabilitado()**.

```
1 void copiarBMPAAlMapa(BMP& Mapa,BMP Modificacion,int xInicial,int yInicial);
```

Esta función se ha implementado para copiar un objeto BMP llamado modificacion (una unidad o un tipo de terreno) a su casillero equivalente en pixeles, los valores xInicial y yInicial son la coordenada x e y del casillero a donde se debe copiar el BMP. Para adaptar esos valores a pixeles se usa la funcion **adaptarValoresBMP()**.