# Manual del Programador

Por Flavio Villanueva, DNI 41579082, PADRON 108014

## Carta.h

**Carta(TipoCartas tipo);**
PRE: Recibe el tipo de carta que se va a cargar
POST: Crea una nueva carta de ese tipo

**~Carta();**
POST: Destructor de Carta

**void agregarCarta();**
POST: Agrega 1 al contador de cartas de ese tipo

**void removerCarta();**
POST: Remueve 1 carta del contador

**TipoCartas getTipo();**
POST: Devuelve el tipo de carta que es

**int getCantidad();**
POST: Devuelve la cantidad de ese tipo de cartas almacenadas

## Coordenada/CoordenadaDouble.h

Ambas clases hacen lo mismo, una con enteros y otra con double

**Coordenada(int x, int y, int z); CoordenadaDouble(double x, double y, double z);**
PRE: Require 3 numeros
POST: Construye una coordenada con los numeros especificados

**getX(); getY(); getZ();**
POST: Devuelve el fragmento de la coordenada en especifico

**Coordenada* getCoordenada();**
POST: Devuelve un puntero a la coordenada completa

**setX(int x); setY(int y); setZ(int z);**
PRE: Requiere el número de esa coordenada
POST: Cambia ese fragmento de la coordenada por el especificado

**SetCoordenada(int x, int y, int z);** PRE: Requiere los 3 numeros
POST: Cambia la coordenada por los 3 numeros

**friend std::ostream& operator<<(std::ostream& os, const Coordenada& c);**

**friend std::istream& operator>>(std::istream& is, Coordenada& c);**

**friend bool operator==(const Coordenada& c1, const Coordenada& c2);**

**friend bool operator!=(const Coordenada& c1, const Coordenada& c2);**

**friend Coordenada operator+(const Coordenada& c1, in i);**

**friend Coordenada operator-(const Coordenada& c1, in i);**

PRE: Se requiere usar los operadores >> o << o == o != o + o -
POST: Sobreescribe dichas operaciones de la clase para hacerlas inmediatamente (en caso de suma o resta, suma i a todas las direcciones)

# Ficha.h

**Ficha(TipoFichas tipo, Coordenada posicion, int jugadorOwner, int turnosInactiva, bool protegido);**

PRE: Recibe el tipo de ficha, la posicion, el jugador que lo controla, la cantidad de turnos que está inactiva (0), y si está protegido (false) POST: Construye la ficha

**TipoFichas getTipo();**
POST: Obtiene el tipo de la ficha

**Coordenada getPosicion();**
POST: Obtiene la coordenada de la ficha

**void setPosicion(Coordenada posicion);**
PRE: Requiere una coordenada POST:
Graba la coordenada actual

**void setTipo(TipoFichas tipo);**
PRE: Requiere un TipoFichas
POST: Graba el TipoFichas actual

**int getJugadorOwner();**
POST: Devuelve el id del jugador con ownership

**void setJugadorOwner(int jugadorOwner);** PRE: Requiere un entero
POST: Cambia el ownership

**void setTurnosInactiva(int turnosInactiva);**
PRE: Requiere un entero
POST: Cambia el número de turnos inactiva

**bool estaInactiva();**
POST: Devuelve true si tiene más de 0 turnos de inactividad restante

**void restarTurnoInactiva();**
POST: Si aun tiene turnos de inactividad, resta 1 al final del turno

**TipoTerreno getTipoTerreno();**
POST: Retorna el TipoTerreno

**void setTipoTerreno(TipoTerreno tipo);**
PRE: Requiere un TipoTerreno
POST: Cambia el tipo del terreno de la ficha

**bool estaProtegido();**
POST: Devuelve si la ficha está protegida

**void setProtegido(bool estado);**
PRE: Requiere un bool
POST: Cambia el estado por el indicado

# Tablero.h

**Tablero(int n, int m, int l);** PRE:
Recibe 3 enteros positivos
POST: Construye un Tablero con
las dimensiones especificadas

**~Tablero();** Destructor del tablero

**void setTData(int n, int m, int l, T data);**
**void setTDataC(Coordenada* pos, T data);**
PRE: Recibe 3 enteros positivos
como Coordenada y el dato
POST: Setea el dato en la celda especificada

**T getTData(int n, int m, int l);**
**T getTDataC(Coordenada* pos);**
PRE: Recibe 3 enteros positivos
como Coordenada
POST: Devuelve el dato de la
celda especificada

**int getTamanioX();**
PRE: Devuelve el tamaño del tablero en X

**int getTamanioY();**
PRE: Devuelve el tamaño del tablero en Y

**int getTamanioZ();**
PRE: Devuelve el tamaño del tablero en Z

**bool inRange(int n, int m, int l);**
PRE: Recibe 3 enteros positivos
como Coordenada
POST: Devuelve true si la coordenada
está dentro del tablero

**void setCoordenada(int n, int m, int l);**
PRE: Recibe 3 enteros positivos
como Coordenada
POST: Setea la coordenada

# Renderizador.h

**RGBAPixel**
**codigoColorSegunCelda(TipoTerreno**
**capaCelda);**
PRE: Requiere un TipoTerreno
POST: Devuelve un RGBAPixel con el
color correspondiente

**bool capaExiste(TipoTerreno capaCelda);**
PRE: Requiere un TipoTerreno
POST: Retorna true si corresponde a una capa

**double gradosARadianes(double grados);**
PRE: Recibe un double correspondiente
a grados de ángulo
POST: Lo devuelve convertido a radianes

**void getAngulos(double angulos[6], int lado);**
PRE: Requiere un array de 6 doubles, y un int
POST: configura los ángulos acorde al lado

**void**
**aplicarProyeccionIsometrica(CoordenadaDou**
**ble* pixel, int lado);**
PRE: Requiere un CoordenadaDouble* y un
int POST: Altera el coordenada para adaptarse
a la proyección 3d de la matriz

**bool coloresSonIguales(RGBApixel color1,**
**RGBApixel color2);**
PRE: Requiere 2 RGBApixel
POST: Devuelve true si son el mismo color

**bool colorEnRango(RGBApixel**
**color);** PRE: Requiere un RGBApixel
POST: devuelve true si todos sus valores
están entre 0 y 255

**bool pixelEnRango(int px, int py,**
**Coordenada imgSize);**
PRE: Requiere 2 ints y un Coordenada
POST: Devuelve true si está entre 0 y
los valores de imgSize

**int pixelSizeGet(RGBApixel color);**
PRE: Requiere un RGBApixel
POST: Devuelve el tamaño de diametro del pixel a pintar

**bool pixelSizeEnRango(Coordenada pixelPos, Coordenada imgSize);**
PRE: Requiere 2 coordenadas
POST: Devuelve true si el ancho del pixel a pintar no excede el marco de la imagen

**void pintarEntidad(BMP* image, Coordenada pixelPos, RGBApixel color, Coordenada imgSize);**
PRE: Requiere un BMP*, 2 Coordenada, y RGBApixel
POST: Guarda la informacion de la foto dentro del BMP*

**Coordenada getPixelOffset(int lado, int size);** PRE: requiere 2 ints
POST: Devuelve el offset que requiere para estar centrada la imagen

**RGBApixel getColor(Ficha* celda, bool esFicha);** PRE: Requiere una ficha* y un bool
POST: Devuelve el color que le corresponde a esa ficha si es del jugador actual

**void imprimirBMP(Coordenada imgSize, BMP* image, Tablero<Ficha*>* tablero, int jugador);**
PRE: Requiere Coordenada, BMP*, Tablero<Ficha*>*, y un int
POST: Hace el handling para llenar los datos de la foto

# Nodo.h

**Nodo(T data);**
PRE: Recibe un dato genérico
POST: Construye un nodo con el dato recibido

**T getNData();**
POST: Devuelve el dato guardado en el nodo

**void setNData(T data);**
PRE: Recibe un dato genérico
POST: Setea el dato del nodo

**Nodo<T>* next();**
POST: Devuelve el nodo siguiente

**Nodo<T>* prev();**
POST: Devuelve el nodo anterior

**void setSig(Nodo<T>* sig);**
POST: Linkea el nodo siguiente al puntero

**void setAnt(Nodo<T>* ant);**
POST: Linkea el nodo anterior al puntero

# Mazo.h

**Mazo() / ~Mazo()**
POST: Constructor y Destructor respectivos

**void agregarCarta(TipoCartas tipo);** PRE: Requiere un TipoCartas
POST: Agrega una nueva carta a su lista, o aumenta su cantidad

**void removerCarta(TipoCartas tipo);** PRE: Requiere un TipoCartas
POST: elimina la cantidad de dicho TipoCartas de la lista, y si no queda más la remueve

**Carta* obtenerCarta(TipoCartas tipo);** PRE: Requiere un TipoCartas
POST: Devuelve la carta de su lista con dicho tipo

**bool estaVacio();**
POST: Devuelve si el mazo está vacío

**int obtenerCantidadCartas(TipoCartas tipo);** PRE: Requiere un TipoCartas
POST: Devuelve la cantidad de ese tipo que tiene

**Lista<Carta*>* obtenerMazo();**
POST: Retorna el mazo

**void usarCarta(TipoCartas tipo);**
PRE: Requiere el TipoCartas
POST: Ejecuta su respectivo llamado al uso de carta

**std::string tipoDeCartaGlobal(TipoCartas tipo);**
PRE: Requiere un TipoCartas
POST: Devuelve el TipoCartas en formato de string

# Lista.h

**Lista();**
**~Lista();**

**void irANodo(int x);**
PRE: Recibe un entero positivo
POST: Itera el cursor de la lista hasta el entero recibido

**void iterar(Iteracion iteracion);**
PRE: Recibe el tipo de iteración
POST: Itera la lista una vez, o hasta el final

**void resetIter();**
POST: Reinicia el iterador en la primer posición

**void assign(T data);**
PRE: Recibe un dato genérico
POST: Y lo asigna al nodo del iterador

**Nodo<T>* getNodo();**
POST: Devuelve el nodo del iterador

**T getLData(int x);**
PRE: Recibe un entero positivo
POST: Obtiene el dato del nodo del iterador

**void add(T data);**
PRE: Recibe un dato genérico
POST: Crea un nuevo nodo al final de la lista

**void remove(int x);**
PRE: Recibe un int de índice
POST: Elimina un nodo de la lista

**int getIter();**
POST: Obtiene la posicion del iterador

**int getSize();**
POST: Obtiene el tamaño de la lista

**void goTo(int x);**
POST: va a la iteración asignada

# Jugador.h

**Jugador(std::string nombre);**
PRE: Requiere un string
POST: Crea el jugador con ese nombre

**~Jugador();**
POST: Destruye el jugador

**std::string getNombre();**
POST: Retorna un string del nombre del jugador

**int getTesorosRestantes();**
POST: Retorna la cantidad de tesoros que le queda al jugador

**Lista<Ficha*>* getListaFichas();**
POST: Retorna la lista de fichas que posee el jugador

**Mazo* getMazo();**
POST: Retorna el mazo del jugador

**int getLenFichas();**
POST: retorna la cantidad de fichas que posee el jugador

**void agregarCarta(TipoCartas tipo);** PRE: Requiere un TipoCartas
POST: Agrega una carta a su mazo

**void removerCarta(TipoCartas tipo);**
PRE: Requiere un TipoCartas
POST: Remueve una carta de su mazo

# Juego.h

**Juego(int jugadores);**
PRE: Requiere un int
POST: Crea un nuevo juego para X jugadores

**void iniciarJuego(int jugadores);**
PRE: Requiere un int
POST: Ordena el mapa para la cantidad de jugadores requerido

**void jugar();**
POST: Inicia y termina el juego

**int mostrarEstadoPartida();**
POST: Muestra si la partida sigue en pie, se finalizó prematuramente o quién ganó.