Universidad ORT Uruguay
Facultad de Ingeniería

# Programación de Redes

# Obligatorio 1

María José Ventura - 192495
María Victoria Martínez - 200587
Grupo M6B, 2018

# ÍNDICE

# 1. Abstract

Este documento contiene una descripción detallada acerca de la solución que implementamos para resolver la letra del Obligatorio I de Programación de redes. La cual incluye descripción, justificación, diagramas, modelos y evidencias del trabajo realizado.

# 2. Especificación de Requerimientos

## 2.1 Alcance

El proyecto consta de dos aplicaciones, un servidor que permitirá que se conecten clientes, evaluará las acciones que estos tomen evaluando su estado y un cliente que permitirá a los jugadores conectarse, elegir un rol y jugar en la partida activa del sistema.

## 2.2 Descripción General

### 2.2.1 Funcionalidades del producto

La plataforma brindará las siguientes funcionalidades:

a. Conexión de un cliente al servidor.
b. Iniciar partida.
c. Terminar partida.
d. Crear perfil del jugador.
e. Permitir al jugador elegir un rol (monstruo o sobreviviente) en partida activa.
f. Permitir al jugador jugar una partida luego de haber elegido un rol.
g. Permitir al jugador que se encuentra en una partida activa realizar acciones.

### 2.2.2 Requerimientos futuros

- ACA VAN LAS FUNCIONALIDADES QUE NO LLEGAMOS A HACWER

## 2.3 Requerimientos

### 2.3.1 Requerimientos Funcionales

#### 2.3.1.1 Aplicación Cliente

##### 2.3.1.1.1 RF01- Conexión y desconexión al servidor.

- Descripción: El cliente deberá ser capaz de conectarse y desconectarse a un servidor.

##### 2.3.1.1.2 RF02- Registro nuevo cliente.

- Descripción: El sistema permitirá dar de alta clientes, éstos tendrán un username, contraseña y un avatar(imagen).
- Especificación: Caso de uso 1.4.1

##### 2.3.1.1.3 RF03- Permitir a un cliente conectarse al juego.

- Descripción: El cliente enviará su username y contraseña y se sumará al juego (no a la partida activa).
- Especificación: Caso de uso 1.4.1

##### 2.3.1.1.4 RF04- Permitir a un cliente seleccionar un rol antes de unirse a la partida activa.

- Descripción: El cliente puede elegir entre dos roles, sobreviviente o monstruo.
  El monstruo tiene 100 puntos de vida (HP), y un poder de ataque(AP) de 10.
  El sobreviviente tiene 20 puntos de vida y poder de ataque de 5.
- Especificación: Caso de uso 1.4.2

##### 2.3.1.1.5 RF05- Permitir al jugador unirse a la partida activa.

- Descripción: Luego de que un cliente se haya conectado al juego, podrá intentar conectarse a la partida activa siendo un jugador con el rol seleccionado.

##### 2.3.1.1.6 RF06- Permitir al jugador realizar acciones durante la partida activa.

- Descripción: El sistema deberá permitir a los jugadores realizar acciones en la partida. Se pueden mover en todas las direcciones con un radio de uno y atacar. El monstruo podrá atacar a sobrevivientes y monstruos mientras que el sobreviviente solo a monstruos.

##### 2.3.1.1.7 RF07- Mostrar el resultado de la partida activa cuando termina.

- Descripción: La aplicación deberá mostrar el resultado de la partida activa al finalizar la misma.

#### 2.3.1.2 Aplicación Servidor

##### 2.3.1.2.1 RF01- Aceptar pedidos de conexión de un cliente.

- Descripción: El servidor debe ser capaz de aceptar pedidos de conexión de varios clientes a la vez.

##### 2.3.1.2.2 RF02- Registro nuevo cliente.

- Descripción: El sistema permitirá dar de alta clientes, éstos tendrán un username, contraseña. En caso de que un usuario quiera dar de alta un usuario existente no será posible realizar esta acción y se le informará al usuario.

##### 2.3.1.2.3 RF03- Ingreso de imagen.

- Descripción: El sistema permitirá al cliente subir una imagen que será su avatar. En caso de que el cliente ya tenga una, esta se sobre escribirá.

##### 2.3.1.2.4 RF04- Mostrar los jugadores registrados

- Descripción: El servidor deberá mostrar una lista de todos los clientes registrados en el mismo.

##### 2.3.1.2.5 RF05- Mostrar los jugadores conectados actualmente al sistema.

- Descripción: El servidor deberá mostrar una lista de los jugadores que están actualmente conectados al sistema.

##### 2.3.1.2.6 RF06- Iniciar partida.

- Descripción: El servidor deberá poder iniciar una y sólo una partida activa, la misma durará tres minutos. No deben permitirse acciones de interacción con el sistema servidor mientras una partida está en juego.

##### 2.3.1.2.7 RF07- Permitir a un jugador unirse a la partida activa.

- Descripción: Cuando el servidor se encuentra en modo partida activa, es posible que se conecten jugadores no conectados al sistema y se unan a la partida activa.

##### 2.3.1.2.8 RF08- Finalizar partida.

- Descripción: El servidor deberá finalizar la partida activa según distintos escenarios.
    a. El tiempo terminó y hay sobrevivientes vivos, ganan los sobrevivientes.
    b. El tiempo terminó y sólo hay monstruos vivos, no gana nadie.
    c. Quedan sólo sobrevivientes, sobrevivientes ganan.
    d. Queda sólo un monstruo vivo, gana el jugador que tenía dicho monstruo.

Si un jugador muere, no puede seguir jugando y deberá esperar a una nueva partida.

#### 2.3.1.2.9 RF09- Mostrar resultado de la partida

- Descripción: Cuando una partida termina, el servidor deberá enviar un mensaje con el resultado a todos los jugadores conectados actualmente al sistema. Se volverá a permitir ver las opciones mencionadas en RF03 y RF04, así como también poder iniciar otra partida.

## 2.4 Casos De Uso

### 2.4.1 Conexión al juego y Registro de nuevo cliente.

- **Descripción:** Permite al cliente ingresar a la aplicación.
- **Actores:** Clientes de la aplicación.
- **Pre condición:** -
- **Flujo normal:**
  1. El sistema muestra la pantalla de login y solicitará usuario y contraseña.
  2. El sistema mostrará mensaje de aprobación.
- **Flujos alternativos:**

  2.1. El usuario que se ingresa ya está en uso, pedirá los datos nuevamente.

  2.2. El usuario no existe, se registrará un nuevo cliente con los datos ingresados.

  2.3. El servidor no está conectado se enviará : “There was a problem connecting to the server, the app will exit”. Al usuario presionar una tecla se cierra la consola.

### 2.4.2 Seleccionar rol

- **Descripción:** Permite al cliente seleccionar un rol para unirse a la partida.
- **Actores:** Usuarios de la aplicación.
- **Pre condición:** -
- **Flujo normal:**
  1. El cliente selecciona la opción ‘Join Game’ del menú principal.
  2. Se le mostrará el menú Roles.
  3. El cliente seleccionará una opción del menú.
  4. El jugador es agregado a la partida.
  5. El sistema lo llevará a la pantalla de la partida.
- **Flujos alternativos:**

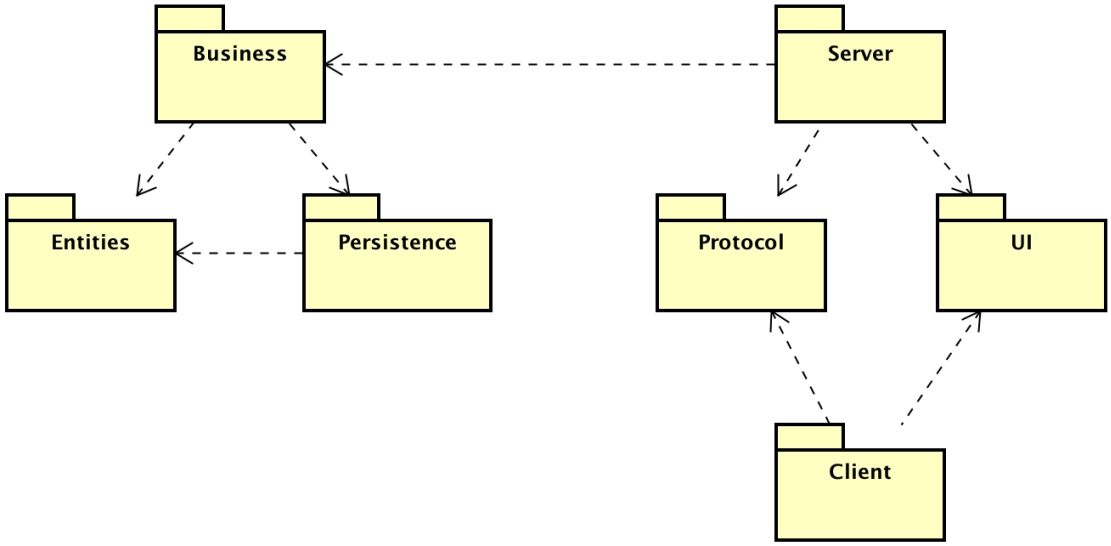
3.1 El jugador selecciona ‘Survivor’ y la partida ya tiene 3 survivors, se le pedirá al cliente que intente nuevamente y seleccione Monster. Vuelve al paso 2.

3.2 La partida está llena o el tiempo límite para unirse ya terminó. Se le notificará al jugador, informando el tiempo que resta para terminar la partida activa.

3.3 El jugador selecciona la opción ‘Exit’. El sistema volverá al menú principal.

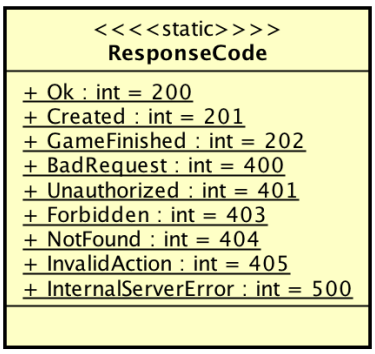
# 3. Descripción de la arquitectura

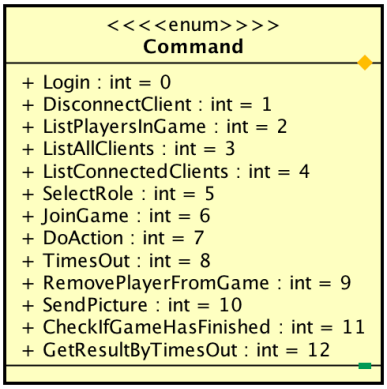
## 3.1 Diagrama de paquetes




## 3.2 Client -> Protocol <- Server

La comunicación entre los clientes y el servidor se realiza a través del protocolo que creamos, en el cual están definidos dos tipos de mensajes, *Request* y *Response*, que se manejan imitando el protocolo HTTP, esto se puede ver en la clase estática *ResponseCode*. En la misma, definimos los códigos que el servidor le enviará a un cliente dentro de una Response. Están presentes los comunes de HTTP, como *GameFinished* e *InvalidAction* que son propios y nos facilitan al momento de trabajar con la response.
Por otro lado, en las Requests, creamos un enumerado *Command,* que nos permite definir las diferentes operaciones que los clientes pueden pedirle al servidor que ejecute. Más adelante veremos que estas son traducidas del Client al Server mediante *Router.*
Al diseñarlo de esta manera, consideramos lograr un modelo escalable, que nos permite ser flexibles al momento de tener que considerar nuevas funcionalidades o modificaciones en los requerimientos, habiéndonos basado en un protocolo existente y verificado.



## 3.2 Business <- Persistence

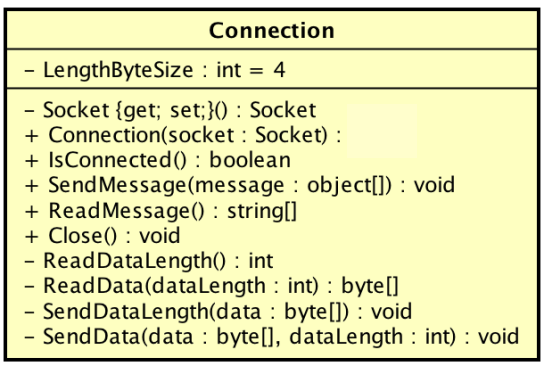
Business es el proyecto que contiene y maneja las restricciones del negocio que posee el servidor para gestionar a sus clientes. El mismo cuenta con tres clases:

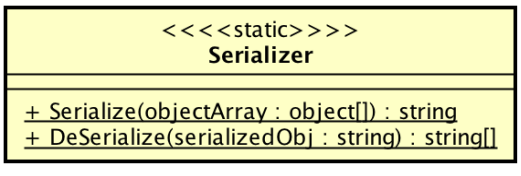
- GameLogic: Se encarga del login/logout de los clientes, obtener las listas que se necesitan mostrar. A su vez, se encarga de la lógica de la partida del juego, y al ser quien se comunica con el server, luego se comunica con *ActionLogic* y *PlayerLogic* para que puedan realizar su trabajo. GameLogic inicia la partida, chequea que los clientes que quieren jugar están conectados, verifica que el tiempo de la partida no se haya terminado y por último finaliza la partida y obtiene su resultado.
- ActionLogic: Es la encargada de la lógica de los movimientos. Verifica si la acción ingresada por el cliente es válida, si se puede realizar, si hay jugadores alrededor, si se mató a alguien, y si los turnos son correctos entre otras cosas.
- PlayerLogic: Aquí se maneja el crear el jugador a partir del cliente y su rol seleccionado, se lo agrega a la partida si es posible (no está llena y el límite de tiempo para unirse no ha pasado), se les asigna una posición en el tablero, y más.

# 4. Justificación de Diseño

## 4.1 Connection

Cuando el cliente se conecta al servidor, se utiliza *ClientProtocol* donde se obtiene una instancia de *Connection* (nueva conexión) en la cual hay un socket que está conectado a otro socket del lado del servidor. Esta conexión se guarda como una property privada *SocketConnection* en *ClientController*, de manera de poder usarla cada vez que el cliente quiera realizar una acción.
La clase *Connection* tiene definidos aquellos métodos necesarios para enviar y recibir información entre dos endpoints usando arrays de objects. Decidimos utilizar este tipo ya que nos permite enviar información en distintos formatos, tanto strings como int, o List<string> sin ningún problema, no importando tamaño ni forma del contenido.
Además de esto, se utiliza la serialización y deserialización utilizando la clase Serializer que convierte de object[] a string y vice versa. Al no utilizar arrays dentro de arrays para pasar información, decidimos dejar esta última clase bastante simple. Pasamos de object[] a string poniendo un '|' al final de cada objeto, y en el DeSerialize realizamos un Split('|') que nos devuelve un array de string con el que es muy fácil trabajar y es suficiente para cubrir el alcance del obligatorio.



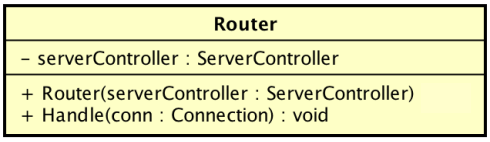



## 4.2 Routing

En el lado del servidor mantenemos un proceso corriendo dentro de un loop en el cual se aceptan conexiones a clientes y los se crean nuevos hilos. El mismo ejecuta el método Handle del *Router.*
La función del Handle, es llamar al método correspondiente del *ServerController* dependiendo de el comando que se ha recibido en la *Request*. Este comando es identificado al construir una *Request* con el mensaje enviado por la *Connection*.
Estas request son constituidas dentro del método BuildRequest en *ClientController* de la siguiente manera:

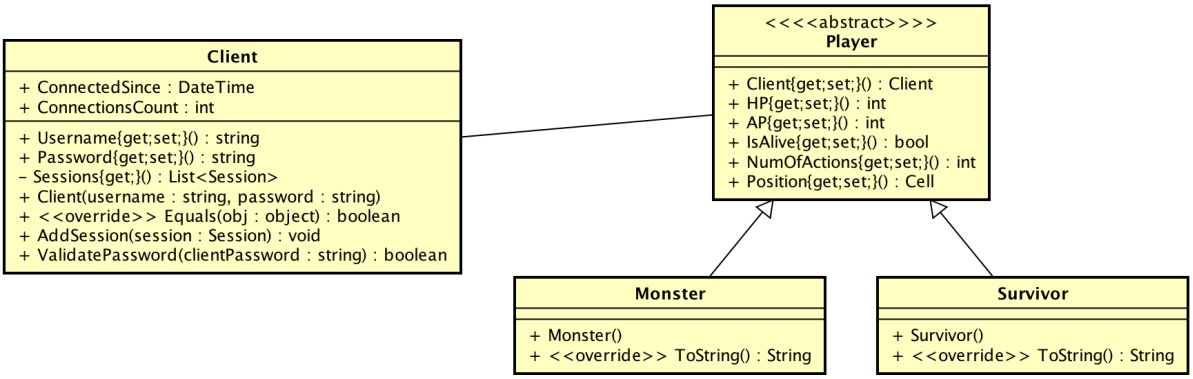
Command - UserToken - Información

El primer objeto es de la clase *Command* explicado en el punto 4.2, que es recibido por el Handler para con un switch indicarle al servidor qué método ejecutar.
El segundo campo, envía un string auto generado por el servidor al conectarse un cliente, con el cual se identifica la sesión. Si la *Request* es Login, este será vacío.
El tercer campo tendrá información variable, dependiendo de qué requiere cada acción del servidor.




## 4.3 Client & Player

Decidimos separar el concepto de cliente y jugador para facilitar la gestión de los mismos. Los clientes son aquellos que están conectados a la aplicación pero que no han seleccionado la opción 'Join Game' en la cual se elige un rol y se une a la partida activa. De esta manera podemos identificar fácilmente a todos los clientes conectados al sistema.
El jugador se crea al seleccionar un rol, esto es ya que decidimos que *Player* sea una clase abstracta de la cual heredan Monster y Survivor que tendrán distintos HP (Health Points) y AP(Attack Points). Cada Player guardara la información del cliente que lo creó, incluyendo username, password y avatar, así como también una posición en forma de *Cell,* la cual explicaremos a continuación, la cantidad de actions que ha realizado y un bool IsAlive.
Al terminar la partida, o abandonarla, este jugador es eliminado, no solo de la partida, sino del sistema, lo que nos permite volver a una nueva partida con otro rol y empezar de 0.



## 4.4 Board & Cell

En cuanto al tablero, decidimos crear una clase Board que contiene el ancho y largo del mismo (siendo estos 8 por reglas del juego), y un array de dos dimensiones de la clase *Cell.* Por lo tanto, el board es básicamente un conjunto de celdas. Éstas contienen sus coordenadas x e y, así como también un jugador, que será null si nadie está sobre ella, de esta manera evitamos que haya dos jugadores en el mismo lugar.
Consideramos que era la manera más práctica para guardar los estados del campo de juego y también mostrarlo como hacemos con DrawBoard en *BoardUI.*

## 4.5 Subir imagen

Con respecto a la subida de archivo de imagen, lo hacemos pasando los bytes de a partes siguiendo la recomendación de los docentes. Dicho esto, leemos de un FileStream con el path de la foto del cliente hasta 9999 bytes (sucesivamente hasta completar el tamaño original), los cuales se guardan en un array, y para mandarlos por el socket los pasamos a StringBase64 (tambien en 'partes'). Esto fue debido a nuestra diferenciación entre Request y Response. El objeto Request es un array de tipo string, por ende nos resultó práctico enviar tramos de strings (que por detras son bytes) y del lado del server simplemente tener un string global cuyo valor vamos concatenando con el siguiente que nos llega del cliente. Esto ocurre mientras el Command recibido sea el de 'partes', y es en la última trama que el comando cambia a 'parte final' y del lado del servidor armamos un Image de System.Drawing y lo guardamos. Dichas imágenes quedan guardadas en la carpeta donde está corriendo el servidor (el exe), en el caso de la carpeta MonsterGame, en Server/bin/Debug, teniendo el username del cliente como nombre.

## 4.6 Manejo de Errores

Para el manejo de errores en la plataforma utilizamos try y catch de distintas Exceptions. Las mismas son algunas como *SocketException,* brindadas por Visual Studio, y otras que creamos con el fin de controlar de manera específica los problemas que pueden ocurrir. En la consola se mostrarán los mensajes de dichas excepciones para que los clientes puedan comprender qué esta sucediendo. Por ejemplo, en el caso de haber algún problema con el servidor, la aplicación se cerrará, por lo que el cliente recibirá el mensaje:

"*There was a problem connecting to the server, the app will exit.*"

Las excepciones propias heredan de una de dos clases padres *ActionException* y *BusinessException,* organizadas según el tipo. ActionException serán aquellas como *MovementOutOfBoundsException* o *CellAlreadyContainsAPlayerException* que controlan aquellos errores que pueden aparecer al jugador realizar una acción (moverse, atacar).
BusinessException son todas aquellas relacionadas con la lógica del juego y la conexión al sistema, como *FullGameException, LoggedPlayerIsDeadException y ClientAlreadyConnectedException.*

## 4.7 Mecanismos de Concurrencia

Con respecto al control de concurrencia, utilizamos el mecanismo de Locks, creando los necesarios para manejarla en los lugares necesitados. Estos están en *GameLogic* que es la clase de lógica de negocio que se comunica con el Server, en cada método que es llamado por el *ServerController* se bloqueará el lock correspondiente evitando así los problemas. Por ejemplo, el login utiliza el lock LoginLock al igual que GetLoggedClient(), GetLoggedPlayers() entre otros, esto es ya que por ejemplo, si quiero ver los jugadores y uno se está conectando en ese instante, no haya complicaciones.

# 5. Reglas y Consideraciones de Monster Game

## 5.1 Instalación

Para el procedimiento de correr el programa hay dos opciones:

a. Abrir Visual Studio 2015, abrir la solución MonsterGame, seleccionar 'Establecer proyectos de inicio múltiples' y seleccionar Client y Server como 'Iniciar'. Para configurar la ip se procede abriendo el archivo app.config del proyecto Client y server respectivamente y eligiendo para el server un puerto que se encuentre libre y la ip de la máquina donde est;a corriendo.
b. Dirigirse a las carpetas release de Client y Server, modificar el archivo Client.exe y Server.exe con las ips de cada máquina donde corran los mismos. En el caso del client se debe configurar además la ip del servidor.

## 5.2 Cierre de consola Cliente y Server

- Los botones de cierre de las consolas estarán deshabilitados para evitar problemas.
- El server podrá cerrarse eligiendo la opción 3 del menú.
- El cliente podrá cerrar su consola seleccionando la opción 4 del menú principal.
- En caso de cerrar el server y el cliente estar conectado, forzaremos el cierre de los sockets, esto lanza una excepción que muestra el mensaje:

  "Se ha anulado una conexión establecida por el software en su equipo host"

  Luego, captura la excepción correspondiente mostrando:

  *"There was a problem with the server, the game will exit."*

  pidiendo el ingreso de una tecla, y por último cerrando la consola como debe.

## 5.3 Cliente

- El Username y Password deberán tener al menos 5 caracteres.
- El avatar se podrá subir con la opción 2 del menú principal.
- Si se sube una imagen y el jugador ya tenía un avatar, este se sobre escribirá.

## 5.4 Jugador

- Al unirse a la partida se le asignará una posición random.
- Su cantidad de movimientos inicial será igual a la máxima cantidad de los demás jugadores.
- Deberá elegir un rol para poder unirse y jugar en la partida.
- Si hay tres jugadores, al ingresar el cuarto se verificará que haya un monstruo en la partida, sino lo hay se le pedirá al jugador seleccionar dicho rol.
- Si la partida está llena, o el tiempo límite para ingresar caducó, se le informará esto al cliente, indicando cuánto tiempo falta para que termine la partida activa.
- El jugador aparecerá en el tablero con su inicial en la celda donde se encuentra.

## 5.5 Partida

- La partida comienza, y el tiempo empieza a correr una vez que el primer jugador se une.
- Se podrán unir nuevos jugadores hasta los 2 minutos de comenzada la partida.
- La cantidad de jugadores máxima es 4.
- Si un jugador se muere, le aparecerá un mensaje de espera y que presione una tecla para proseguir. Esto aparecerá tantas veces el jugador presione una tecla hasta que la partida haya terminado. En ese caso se le informará el/los ganadores.

### 5.5.1 Finalización

- Si hay sólo sobrevivientes al momento que el tiempo de unirse a la partida ha terminado, éstos ganan automáticamente.
- Si la partida ha finalizado por tiempo, se le avisará a los clientes que pertenecen o han pertenecido a ella, junto con el resultado de la misma.

#### 5.5.1.1 Modos y ganadores

1- Quedan sólo sobrevivientes, y el tiempo límite para unirse a la partida ha terminado, todos ellos ganan.
2- Queda sólo un monstruo vivo, y el tiempo límite para unirse a la partida ha terminado, ganará el jugador que tenía dicho monstruo.
3- El tiempo terminó y hay sobrevivientes vivos, sobrevivientes ganan.
4- El tiempo terminó y solo hay monstruos vivos, nadie gana.

## 5.6 Acciones

- El jugador podrá realizar dos acciones por turno. (Move + Move, Move + Attack, Attack + Move, Attack + Attack)
- El jugador se podrá mover en todas las direcciones con radio uno.
- No hay un tiempo restringido para que un jugador realice sus acciones.
- Un mensaje como : *"You are next to: Hector15(Monster)"* aparecerá cuando el jugador realice una acción y se encuentre cercano (radio de uno) a otro jugador o jugadores.
- Si la acción es inválida ese turno no se le contará.
- No se discriminarán los espacios o mayúsculas/minúsculas de los comandos ingresados.

### 5.6.1 Movimiento

MOVLetraNúmero ej.: MOVA5.

- Si hay otro jugador en la posición a la que desea ir se le avisará.
- Si un jugador llega a los límites del terreno este no podrá moverse para afuera de los mismos.
- Moverse al mismo lugar donde el jugador se encuentra se considera un movimiento inválido.

### 5.6.2 Ataque

ATTUsername ej.: ATTHector15.

- El jugador podrá atacar a otro jugador si se encuentra próximo a él.
- Si el jugador es un monstruo podrá atacar a cualquiera, sobreviviente sólo podrá atacar a monstruo.
- Un mensaje como: *"You have killed Hector15"* aparecerá cuando el jugador mate a otro jugador.

### 5.6.3 Salida del juego

Exit

- El jugador saldrá de la partida y se eliminará de la misma y el sistema.
- El jugador puede salir y entrar a la partida las veces que desee (mientras no haya terminado o esté llena) empezando así de cero cada vez.
- Si un jugador muere no podrá salir de la partida.