Universidad ORT

Obligatorio 1

Programación de Redes

Juan Andrés Vico (nro. 200135)
Sebastián Caraballo (nro. 200562)

03-10-2018

Índice

1. Alcance de la Aplicación

1.2 Servidor

- Aceptar pedidos de conexión de un cliente.
    - El servidor debe ser capaz de aceptar pedidos de conexión de varios clientes a la vez.
- Dar de alta un jugador.
    - El sistema permitirá dar de alta jugadores. Los jugadores tienen un nickname (sobrenombre) y un avatar (foto). En caso de que un usuario quiera dar de alta un usuario existente no será posible hacer esto y se le informará al usuario.
- Mostrar los jugadores registrados.
    - El servidor deberá mostrar una lista de todos los jugadores registrados en el mismo.
- Mostrar los jugadores conectados actualmente al sistema.
    - El servidor deberá mostrar una lista (que se debe actualizar automáticamente) de los jugadores que están actualmente conectados al sistema.
- Iniciar partida
    - 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.
- Finalizar partida
    - Diferentes escenarios para finalizar una partida:
        - El tiempo terminó y hay sobrevivientes vivos, sobrevivientes ganan.
        - Quedan sólo sobrevivientes, sobrevivientes ganan.
        - Queda sólo un monstruo vivo, gana el jugador que tenía dicho monstruo.
        - El tiempo terminó y sólo hay monstruos vivos, nadie gana.
        - Si un jugador muere no puede seguir jugando y deberá esperar a una nueva partida para volver a jugar.
- Permitir a un jugador unirse a la partida activa.
    - 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.
- Mostrar resultado de partida.
    - Cuando una partida termina el servidor deberá enviar un mensaje con el resultado a todos los jugadores conectados actualmente al sistema.

1.2 Cliente

- Conectarse y desconectarse al servidor.
    - El cliente deberá ser capaz de conectarse y tambien desconectarse a un servidor.
- Dar de alta un jugador.
    - El sistema permitirá dar de alta jugadores, los jugadores tienen un nickname (sobre nombre) y un avatar (foto).
- Permitir al jugador conectarse al juego.
    - El jugador enviará su nickname y en caso de que alguien más (otro cliente) no lo esté usando se sumará al juego (no a la partida activa).
- Permitir a un jugador unirse a la partida activa.
    - Luego de que un jugador se haya conectado al juego podrá intentar conectarse a la partida activa, en caso de que en el servidor exista una partida activa se pasará a la selección de rol y luego a la partida, en caso de no haber una partida activa el sistema le informará de esto al jugador.
- Permitir al jugador seleccionar un rol antes de iniciar la partida.
    - El jugador puede elegir entre dos roles, monstruo o sobreviviente. El monstruo tiene 100 (cien) puntos de vida y un poder de ataque de 10 (diez). El sobreviviente tiene 20 (veinte) puntos de vida y un poder ataque de 5 (cinco). El monstruo sólo podrá atacar a sobrevivientes y monstruos, el sobreviviente puede sólo atacar a los demás monstruos.
- Permitir al jugador realizar acciones de atacar durante la partida activa.
    - El jugador podrá atacar durante la partida activa. Monstruo ataca a cualquiera competidor mientras que sobreviviente ataca solo a monstruos.
- Permitir al jugador realizar acciones de mover durante la partida activa.
    - El jugador podrá moverse hasta dos lugares.
- Mostrar el resultado de la partida activa cuando termina.
    - Una partida puede terminar en diferentes escenarios, el manejo de esta será exclusivo del servidor y aquí sólo mostraremos el resultado.

2. Arquitectura

La arquitectura utilizada es la arquitectura de cliente - servidor. La arquitectura cliente - servidor es un modelo de diseño de software en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Un cliente realiza peticiones a otro programa, el servidor, quien le da respuesta.

2.1 Decisiones tomadas

2.1.1 Desde el punto de vista del servidor

Se optó por separar la lógica de negocio de la clase que ejecuta el servidor. Esta decisión fue tomada con el fin de poder aumentar la extensibilidad del proyecto, separando claramente las responsabilidades entre la entidades. De esta manera, si se decide cambiar la implementación de la lógica, se hará sin afectar el funcionamiento de la clase Server.

Dentro del paquete de Lógica, se encuentran las clases que van a efectuar la lógica del servidor.

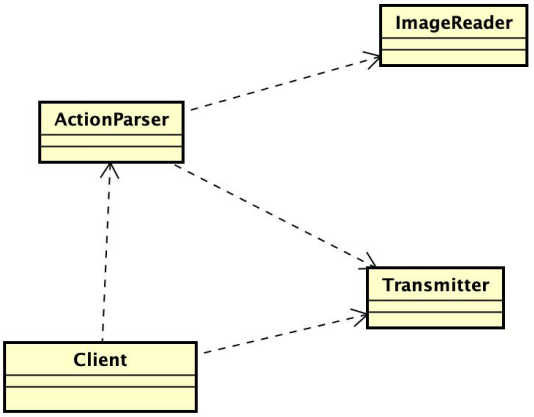
La clase Transmitter, se va a hacer cargo de transportar y recibir los paquetes entre el cliente y el servidor. En ella se define el protocolo con el cual se comunican ambas partes de la arquitectura. Luego Game tiene la implementación de las funciones referidas a la lógica del juego, mientras que ActionParser interpreta los comandos del cliente y llama a las funciones correspondientes.

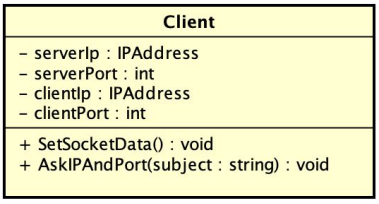
2.1.2 Desde el punto de vista del cliente

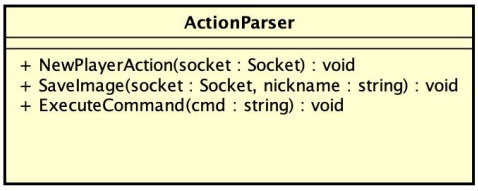
En este proyecto también nos encontramos con la clase Transmitter, donde en ella, como en el servidor, se va a implementar el protocolo del que vamos a hablar en más detalles en el siguiente punto.

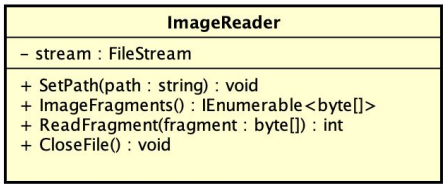
3. Diagramas

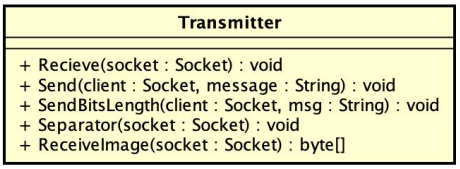
3.1 Client



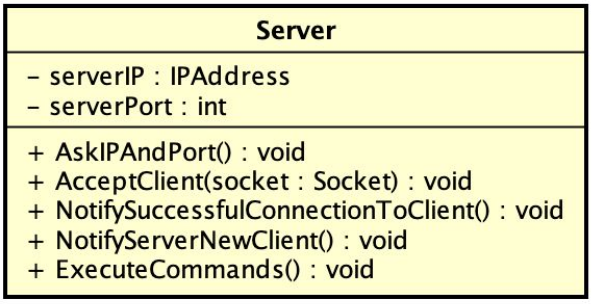




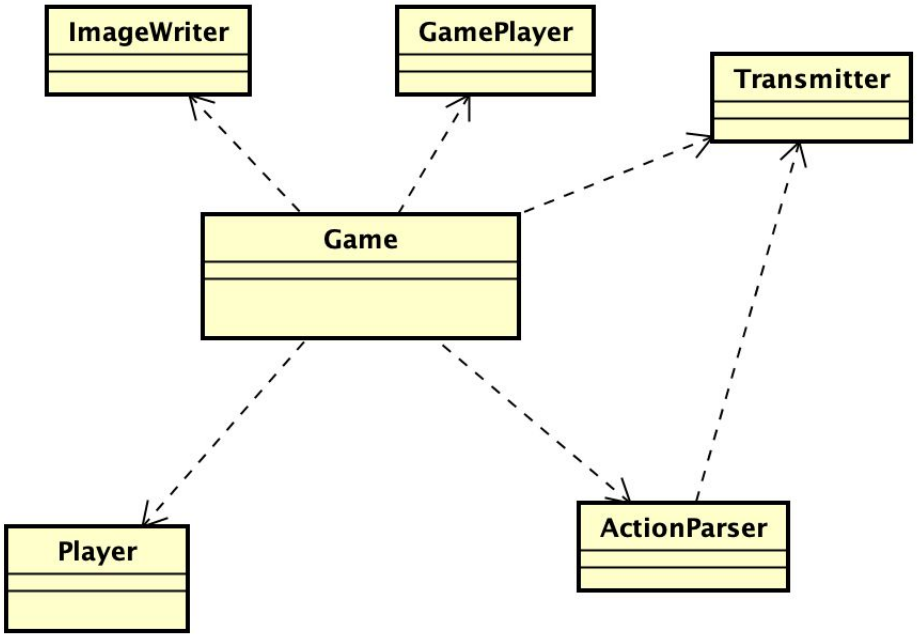


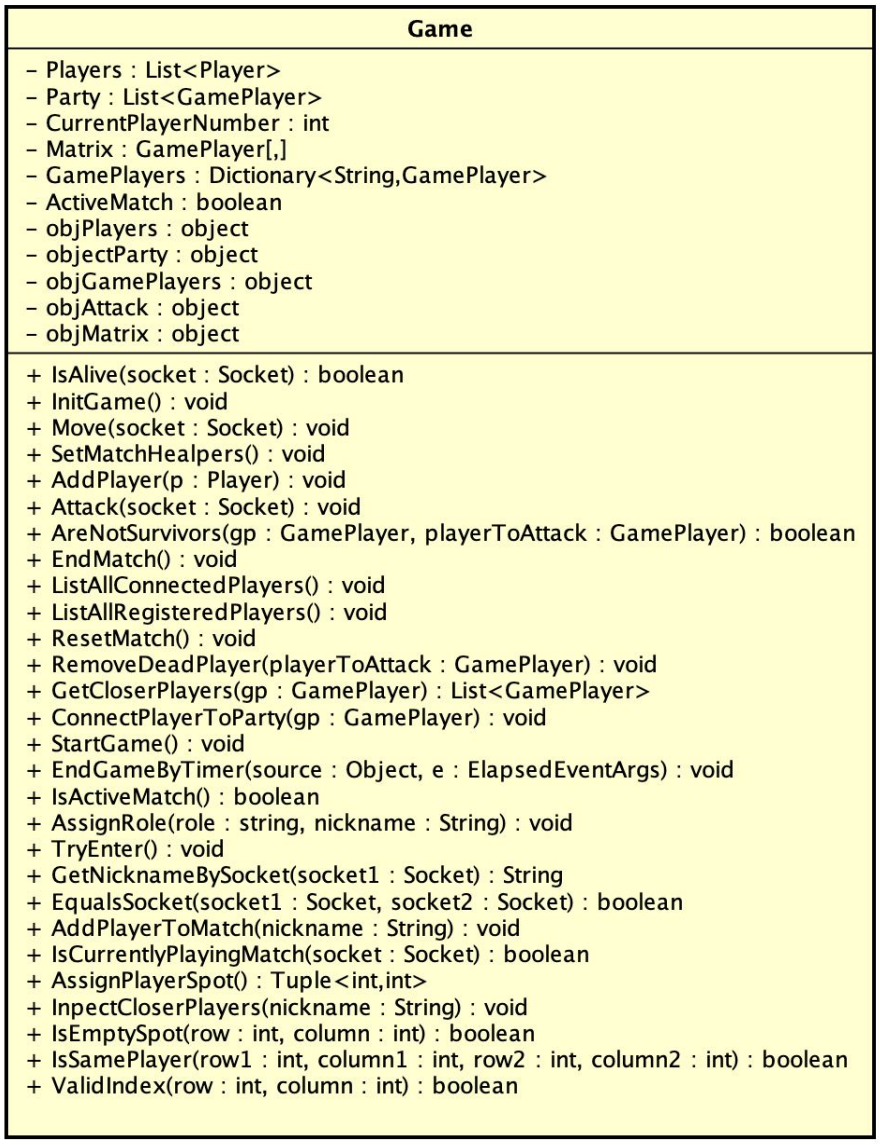


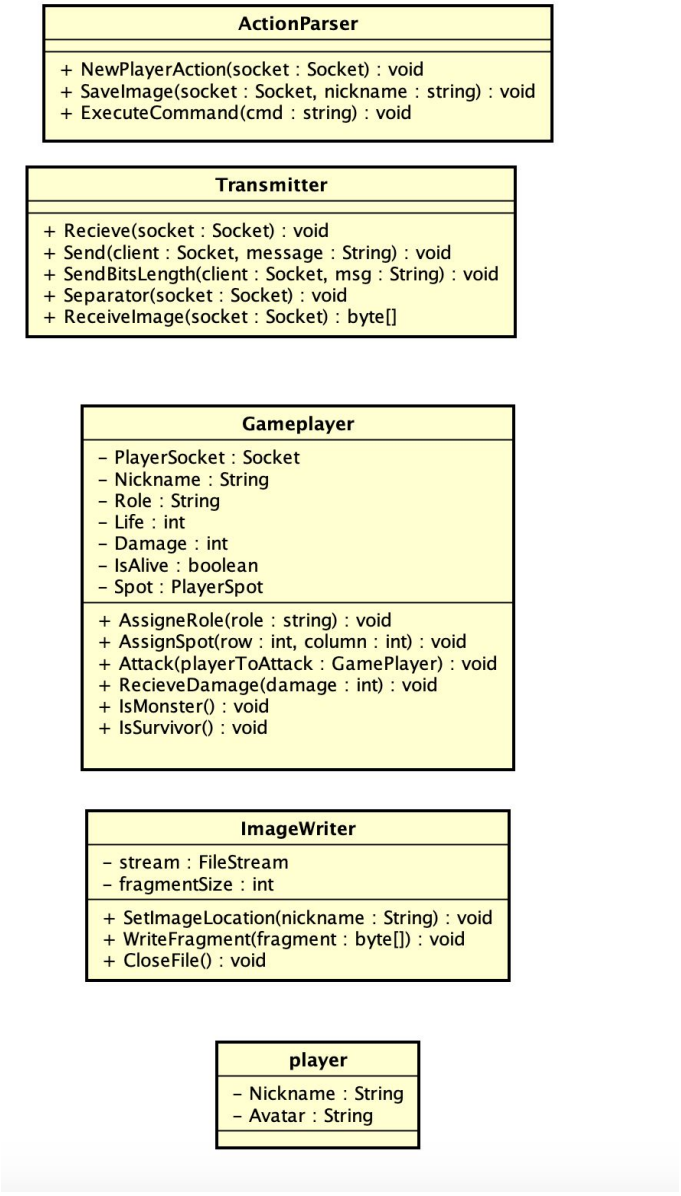
3.2 Server



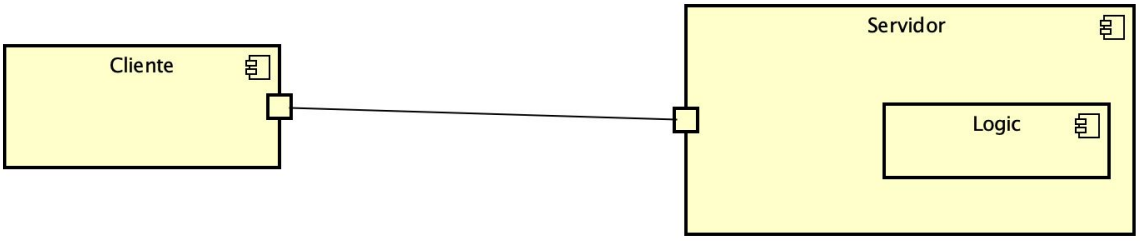
3.3 Server Logic




Game
- Players : List<Player>
- Party : List<GamePlayer>
- CurrentPlayerNumber : int
- Matrix : GamePlayer[,]
- GamePlayers : Dictionary<String,GamePlayer>
- ActiveMatch : boolean
- objPlayers : object
- objectParty : object
- objGamePlayers : object
- objAttack : object
- objMatrix : object
+ IsAlive(socket : Socket) : boolean
+ InitGame() : void
+ Move(socket : Socket) : void
+ SetMatchHealpers() : void
+ AddPlayer(p : Player) : void
+ Attack(socket : Socket) : void
+ AreNotSurvivors(gp : GamePlayer, playerToAttack : GamePlayer) : boolean
+ EndMatch() : void
+ ListAllConnectedPlayers() : void
+ ListAllRegisteredPlayers() : void
+ ResetMatch() : void
+ RemoveDeadPlayer(playerToAttack : GamePlayer) : void
+ GetCloserPlayers(gp : GamePlayer) : List<GamePlayer>
+ ConnectPlayerToParty(gp : GamePlayer) : void
+ StartGame() : void
+ EndGameByTimer(source : Object, e : ElapsedEventArgs) : void
+ IsActiveMatch() : boolean
+ AssignRole(role : string, nickname : String) : void
+ TryEnter() : void
+ GetNicknameBySocket(socket1 : Socket) : String
+ EqualsSocket(socket1 : Socket, socket2 : Socket) : boolean
+ AddPlayerToMatch(nickname : String) : void
+ IsCurrentlyPlayingMatch(socket : Socket) : boolean
+ AssignPlayerSpot() : Tuple<int,int>
+ InpectCloserPlayers(nickname : String) : void
+ IsEmptySpot(row : int, column : int) : boolean
+ IsSamePlayer(row1 : int, column1 : int, row2 : int, column2 : int) : boolean
+ ValidIndex(row : int, column : int) : boolean

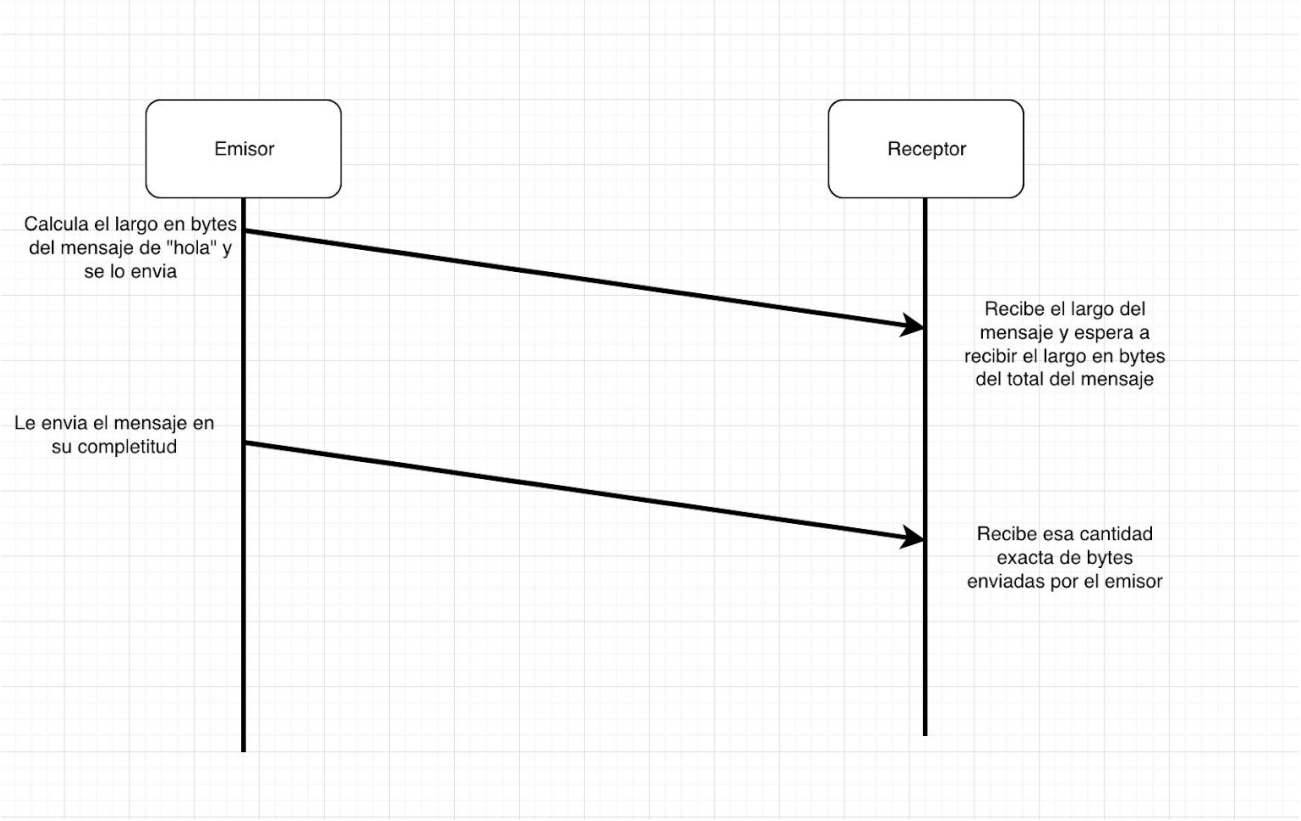


3.2 Diagrama de componentes



4. Protocolo

El protocolo consta de dos partes imprescindibles para que el funcionamiento del envío de paquetes entre el emisor y el receptor sea el esperado y evitar enviar bytes que luego no van a ser usados, o menos bytes de los que realmente se necesitaban para el mensaje.
Esto a su vez hace más eficiente la interacción entre el emisor y el receptor permitiendo el uso óptimo y eficaz de recursos.
La primer parte consta en tomar el largo del mensaje que quiero mandar y de esta manera así enviarle a la otra parte, el largo de bytes que voy a enviar en dicho mensaje.
Este largo de bytes que voy a mandar en el mensaje es un paquete que tiene largo 4, siempre el mismo largo de bytes para todos los mensajes. Entonces el receptor recibe ese paquete que ya sabe que es de largo 4 y obtiene el largo del mensaje que va a recibir.
El emisor le envía el mensaje completo, llegando en su completitud al receptor con un número exacto de bytes a recibir.



4.1 Transmisión de imagen

A continuación se explicara como fue el mecanismo de transmisión de imágenes entre cliente y servidor. Desde el punto de vista del cliente, se hace uso de la clase ImageReader, la misma se va a hacer cargo de traer la imagen del sistema de archivos del usuario cliente. Utilizamos una librería de clase llamada FileStream, la misma se encarga de obtener la imagen. Notar que la imagen no es obtenida en su totalidad, ni es guardada en memoria lo que lo hace más eficiente. El funcionamiento consta en partir la imagen en fragmentos e ir enviando fragmento por fragmento al servidor. De esto se va a encargar el método que se llama ImageFragments, localizado en la clase mencionada. Desde el punto de vista del servidor, este va a recibir fragmento por fragmento y de la misma manera los va a ir escribiendo en la ruta especificada por el FileStream.

5. Manejo de errores

El manejo de errores que utilizamos fueron las excepciones. Donde se muestran los mensajes contenidos en las mismas.

6. Mecanismos de concurrencia

Los mecanismos que utilizamos con el fin de lidiar con la concurrencia fueron los locks. Estos se encuentran en la clase Game que es donde se va a proceder con la ejecución del juego y por ende es donde se encuentran las zonas más críticas de concurrencias en el programa. Cabe destacar que dicha clase Game se encuentra en el servidor.
Los puntos críticos o mejor dicho, zonas de mutua exclusión fueron las siguientes:

- Cuando se agrega un jugador
- Cuando dicho jugador desea conectarse a la party
- Cuando se agrega un jugador de la party a un match
- Cuando se produce un ataque
- Cuando se desea hacer un movimiento

Los objetos en los cuales se utilizan los locks son:

En la lista de Players, cuando se desea agregar un jugador.

```
lock (objPlayers)
{
    Players.Add(p);
}
```

En la lista Party de GamePlayer, cuando se conectan jugadores a la party.

```
lock (objParty)
{
    Party.Add(gp);
}
```

En el diccionario GamePlayers a la hora de remover un jugador muerto.

```
lock (objGamePlayers)
{
    GamePlayers.Remove(playerToAttack.Nickname);
}
```

En el GamePlayer que participa en un ataque.

```
lock (objAttack)
{
    gp.Attack(playerToAttack);
}
```

En la matriz del juego, cuando un jugador intenta moverse a otra posición.

```
lock (objMatrix)
{
    Matrix[gp.Spot.Row, gp.Spot.Column] = null;
    Matrix[possibleSpot.Row, possibleSpot.Column] = gp;
    gp.Spot = possibleSpot;
}
```

Estas son zonas críticas ya que los procesos de un cómputo se hacen simultáneamente, y pueden interactuar entre ellos. Los cálculos (operaciones) pueden ser ejecutados en múltiples procesadores, o ejecutados en procesadores separados físicamente o virtualmente en distintos hilos de ejecución. Lo cual hace que se vea afectado el buen funcionamiento del sistema si no son manejados con precaución.

7. Funcionamiento de la aplicación

7.1 Instrucciones

Para el manejo de la aplicación tanto como desde el punto de vista del servidor como el del cliente, se deberán ingresar las respectivas direcciones IP y puerto para poder generar una conexión cliente-servidor.

Para el cliente, a la hora de ejecutar el comando *NEWPLAYER*, se deberá ingresar el nickname del jugador y a su vez una imagen. Esta imagen debe estar ubicada en la carpeta del ejecutable del proyecto. Y cuando se va a ingresar el nombre de la imagen por consola la misma deberá ingresarse como se encuentra el nombre del archivo junto con su extensión.

Ejemplo:
Si el usuario desea ingresar una imagen que se llama crash.png, la misma deberá estar ubicada en la carpeta del ejecutable y una vez que se le pida el ingreso de la imagen por consola el usuario deberá ingresar ‘crash.png’.