Trabajo Final - Estructuras de Datos y Algoritmos 2018Q1

Enunciado y requerimientos

Se deberá realizar un juego de Snow Bros. entre dos jugadores. Cada jugador corre el programa en una máquina distinta.

El juego se conecta mediante la librería Boost.Asio en modo TCP/IP utilizada durante el cuatrimestre. El protocolo de comunicación está definido por la cátedra y sus detalles se encuentran más adelante.

La interfaz del usuario deberá ser gráfica y se le deberá permitir al usuario controlar el juego con el mouse y el teclado empleando la librería Allegro con la que se trabajó durante el cuatrimestre también.

Será imperativa la correcta modularización del programa y el uso de C++ ya que simplifica la programación.

Se emplearán las reglas de Snow Bros. que se encuentran en el sitio https://en.wikipedia.org/wiki/Snow_Bros#Gameplay con algunas excepciones que serán aclaradas más adelante. Se incluirán también links con explicación de las reglas y detalles del juego para su correcta comprensión.

No deberá crearse un "jugador virtual" de Snow Bros., sino que el problema se reduce a lograr una correcta comunicación entre dos máquinas, no permitir que ningún jugador viole las reglas que se explicarán en breve, detectar los distintos resultados del juego (ganar o perder), diseñar un entorno gráfico interactivo e intuitivo que se pueda manejar mediante el mouse y detectar silencios suficientemente pronunciados como para entender que se ha perdido la comunicación entre las máquinas.

Detalles importantes (puntos principales de evaluación):

- Correcta modularización.
- Uso de objetos cuando corresponda (su uso desmedido puede tornarse catastrófico).
- Separación clara entre el front-end y el back-end.
- Correcta diferenciación entre el "motor" del juego y el manejo de cada periférico.
- Interfaz gráfica empleando Allegro (front-end).
- Uso del mouse y el teclado empleando Allegro (back-end).
- Uso del canal de comunicaciones empleando Boost:Asio (back-end).
- Uso de timers empleando Allegro o Boost:timer (back-end).
- Respeto absoluto del protocolo de comunicación (la interoperabilidad entre versiones de 2 grupos será considerada crucial).
- Aplicación de todos los conceptos vistos en clase según corresponda: ***la aplicación de cada concepto supone un ámbito particular en el que aplicarlo; se evaluará el ámbito en que se aplicó el concepto tanto como la correcta aplicación del mismo.***

Dinámica del juego

- ***Introducción***

La dinámica se encuentra en https://en.wikipedia.org/wiki/Snow_Bros#Gameplay, y tomaremos dichas reglas como las oficiales de Snow Bros para nuestra implementación. Sin embargo, con el afán de mejorar la claridad de las reglas y facilitar el juego vamos a repasarlas y cuando corresponda las modificaremos.

El párrafo inicial explica que el juego Snow Bros se puede jugar con hasta dos jugadores. En nuestro caso, siempre se jugará de a dos, no estando disponible la posibilidad single player salvo que durante el juego alguno de los personajes sea eliminado en cuyo caso continuará el otro hasta que termine el juego o pierda.

Continuando con las reglas, las mismas mencionan que cada 10 niveles hay un jefe. Nosotros ignoraremos el jefe y el juego se completará (se gana) cuando se pasaron 10 niveles.

Respecto de las pócimas, no utilizaremos ninguna de las que están en el juego. Tampoco jugaremos con la calabaza malvada que aparece cuando se demora en pasar un nivel.

- ***Nick & Tom***

Utilizaremos escenarios de 12 filas por 16 columnas, de forma que tanto Nick & Tom como los diferentes enemigos ocupan siempre uno de los 192 casilleros del escenario. Al moverse pasarán de un casillero a otro.

Tomaremos los siguientes parámetros: un salto dura 1.2s y avanza 2 filas hacia arriba. Se indica según se explica más adelante.

Un desplazamiento dura 300ms y avanza una columna en la dirección en la que mira el jugador (salvo que el jugador se encuentre con los límites del escenario y no pueda seguir avanzando). Se indica como se explica más adelante.

Se puede saltar y desplazarse al mismo tiempo. Para ello se deberá indicar que se desea saltar y al mismo tiempo desplazarse. En dicho caso se avanzan dos filas hacia arriba y una columna en el sentido en que mira el jugador (excepto que se encuentre en los límites del escenario).

Cuando un jugador o enemigo cae lo hace a razón de 300ms por fila.

Cada vez que Nick & Tom tiran nieve la misma tiene un alcance de hasta tres columnas. Por lo tanto, si hay un enemigo en la columna siguiente a donde se encuentran el mismo será impactado por la misma congelándolo (y por lo tanto evitando que se siga moviendo). De lo contrario avanzará un casillero adicional en la misma dirección. Si allí hay un enemigo sufrirá el mismo efecto. Caso contrario, el proyectil de nieve avanzará un casillero más. Si hubiera allí un enemigo el mismo será impactado por el proyectil e inmovilizado. Caso contrario el proyectil se diluye en la tercera columna y no avanza más perdiendo su efectividad.

Tanto Nick como Tom son inmunes los proyectiles de su compañero.

Se necesitan 3 impactos para generar una bola de nieve de un enemigo. La bola de nieve bloquea el paso, de forma que si hay una bola de nieve y se avanza sobre ella se la mueve. Para hacerla rodar, se debe posicionar en el casillero contiguo a la bola de nieve y mirando a ella se debe disparar. Si la bola de nieve rueda sobre el jugador ésta lo arrastra pero

no lo mata. La bola se moverá a 100ms por columna, y si rebota 7 veces contra la misma pared, desaparece.

El proyectil avanza a razón de 100ms por columna, y se podrán disparar cada 150ms como mínimo. Si un enemigo toca a Tom o Nick aquel que fue tocado pierde una vida. Los enemigos serán generados por alguna de las máquinas. Cada jugador comienza con 3 vidas. Si un jugador muere, el otro seguirá jugando. En caso de subir de nivel con un jugador muerto, revivirá con una vida. En caso de perder una vida, el jugador que la haya perdido reaparecerá en su posición inicial del mapa tras 5 segundos, y será inmune durante 3 segundos.

En el caso en el que un jugador intente moverse a una casilla ocupada por un enemigo completamente congelado, la bola y el jugador se moverán a razón de 600 ms por columna. Si el enemigo no está completamente congelado, no se lo podrá mover y no afectará al movimiento del jugador En cuanto a empujar al enemigo hacia arriba o dejarlo caer, subirá/caerá a la misma velocidad (300ms por fila).

- ***Enemigos***

Los enemigos se congelan durante 30 segundos. Pasado ese tiempo comienzan a descongelarse. Tiene 4 estadios de descongelamiento de 10 segundo cada uno. Luego se descongelan completamente y vuelven a su movimiento natural.

En el juego habrá tres diferentes tipos de enemigos, cuyas características se explican a continuación:

Enemigos	**Características**
Purple Guys 	Estos enemigos no tienen proyectiles y simplemente se desplazan con el objetivo de hacer contacto directo con el jugador. En cada movimiento que hace este enemigo se debe decidir si este irá hacia el jugador mas cercano(con probabilidad 0.6), se moverá hacia una dirección aleatoria(probabilidad 0.3) o se quedará quieto por 0.3s (probabilidad 0.1). En caso de moverse hacia el jugador mas cercano este será elegido a través del algoritmo de Dijkstra (en el caso de haber dos caminos de igual longitud se elige el camino más “clockwise”, comenzando desde la izquierda) y se seguirá ese camino con un paso. Su movimiento será a una razón de 300ms/columna.
Green Fatties 	Estos enemigos son similares a los Purple Guys, pero con un ataque de fuego. En cada movimiento, el enemigo decide si dispara con probabilidad 0.3 o si camina en el sentido en el que estaba mirando con probabilidad 0.6. En caso contrario a ambos, decide si saltar, moverse en la dirección contraria o quedarse quieto por 0.5 segundos de manera equiprobable. Una vez que dispara, se queda quieto por 0.9 segundos. La velocidad del proyectil será de 300ms/columna y tiene un alcance de 3 casilleros. Este enemigo se mueve a 300ms/columna.

Crazies 	Estos enemigos también son muy similares a los Purple Guys pero más agresivos. En cada movimiento se mueven en la dirección en la que estaban mirando con una probabilidad de 0,75. En el caso que no continúen en la dirección anterior, es equiprobable que salten, que se muevan en la dirección contraria o que se queden quietos por 0,5 segs. Son más veloces, se mueven a una razón de 200ms/columna.

**Aclaración:* si alguno de los enemigos intentará moverse a algún lugar fuera del mapa, es decir hacia una pared, este deberá realizar el movimiento contrario.

- ***Puntaje***

Por cada disparo que impacte en un enemigo el jugador gana 10 puntos. Una vez congelado un enemigo al golpearlo se consiguen 500 puntos y 20 puntos por cada vez que la bola de nieve rebote contra una pared. Si rebota contra una pared con un jugador en ella, el mismo gana 200 puntos por cada vez que lo haga.

Si se elimina a un enemigo con una bola de nieve hecha a partir de otro se consiguen mil puntos. En caso de eliminar a más de uno cada uno aporta 1000 puntos más que el anterior (2 enemigos representan 1000+2000=3000 puntos, 3 enemigos 1000+2000+3000=6000 puntos, etc.).

Convenciones y definiciones

Para estandarizar el proceso de transmisión de mensajes vamos a definir algunas convenciones que van a ayudar a que dos programas realizados por grupos distintos se puedan comunicar entre sí sin lugar a ambigüedades.

La primera regla que vamos a definir consiste en el método para nombrar las jugadas. Una máquina deberá informarle a la otra la jugada a realizar, y para evitar confusiones se adoptarán los siguientes criterios:

1. En **ambas** máquinas el layout del escenario será el mostrado en la figura 1.
2. Se diferenciarán los casilleros siguiendo el sistema de coordenadas de abajo donde se anota primero la fila con letras de la "A" a la "L" (A, B, C, D, …, L) y luego las columnas con números del 1 al 16 (1, 2, 3, 4, …, 16). Cada enemigo y jugador ocupan un casillero.

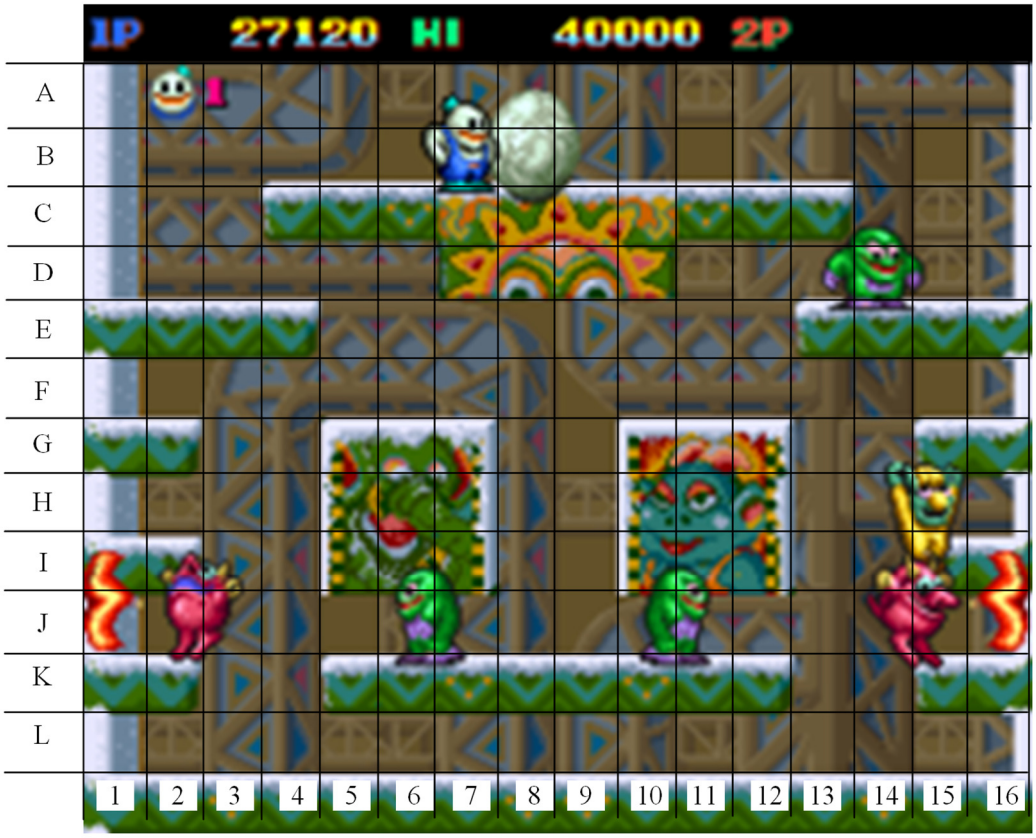

Figura 1

3. Para especificar un movimiento, se indicará el personaje a mover y la coordenada a donde se dirige. Para el caso de los disparos se indicará también el personaje y la coordenada a donde se dirige el disparo. El cliente le enviará un request al servidor como se detalla en la sección ***Protocolo de comunicación.*** Para especificar el accionar de los enemigos, se envía un paquete por cada enemigo especificando a donde se dirige y si ataca o no. En este caso, el servidor genera el movimiento y se los comunica al cliente.

4. Cada vez que Nick o Tom le tiran nieve a un enemigo, simplemente indicaremos de vuelta la letra que representa al personaje que realiza la acción y la coordenada (fila y columna) a dónde cae la nieve. En el caso de estar frente a una bola de nieve, ambas máquina saben la posición de cada enemigo y cada jugador por lo que pueden resolver la situación.
5. Los escenarios se especifican en un archivo de texto estilo CSV (comma separated values). El formato de este archivo y cómo nombrar a los personajes se encuentran en el ***Anexo 1***.

La segunda regla que vamos a definir refiere al método en que el usuario indicará la acción que desea realizar. Como se explicó en la introducción del trabajo práctico, se emplearán el mouse y el teclado como interfaz entre el usuario y el tablero. Esta regla contempla los siguientes criterios:

1. El usuario utilizará la flecha izquierda para moverse hacia la izquierda, la flecha derecha para moverse hacia la derecha, la flecha hacia arriba para saltar y la barra espaciadora/ 0 del keypad para tirar nieve
2. Se pueden incluir botones para operar con el mouse como opciones para salir del juego también utilizando el mouse. Más allá de ello el mouse en este TP no tiene mayores usos.

Vamos a definir también, una tercera regla referente a la operación del juego:

1. El sistema le deberá informar al usuario lo que sucede en **todo momento del juego**.
2. Deberá proveerle una forma de abandonar el juego en cualquier momento.
3. Además, una vez comenzado el juego, se deberá llevar un registro de la cantidad de vidas de cada jugador, su puntaje y el nivel por el que van para que el jugador sepa cuánto falta para terminar el juego. También se deberá llevar un registro del tiempo que hace que se inició el juego.
4. Se le deberán mostrar todos estos datos al usuario en todo momento.

Por último, vamos a adoptar una regla referente a lo que se ha de transmitir entre máquinas. A continuación los criterios que hacen a esta regla:

1. Las máquinas sólo transmitirán jugadas válidas, según lo descripto a continuación.
2. La máquina que recibe la jugada, antes de representarla en el escenario la deberá analizar. Si la jugada fuera inválida será tomada como un error de comunicación y se dará por terminado el juego (ver ***Protocolo de comunicación*** para más detalles). Caso contrario será representada en el escenario.
3. Cada jugada será enviada inmediatamente después de haber sido realizada
4. La transmisión entre máquinas estará regida por el protocolo descripto a continuación bajo el título ***Protocolo de comunicación***. Si se violara alguna de las reglas de dicho protocolo se dará por terminado el juego informándole al usuario que hubo un error de sincronismo entre las máquinas.

Protocolo de comunicación

A continuación, se detalla el protocolo de comunicación, tanto en handshake como en el tamaño, forma y contenido de los mensajes.

- ***Esperando conexión***

  Se deberá iniciar la comunicación a una dirección IP indicada por el usuario y al puerto 13225 que utilizaremos como puerto de Snow Bros.

  Primero se intentará como cliente. Si no se logra establecer la comunicación (del otro lado no hay nadie) se deberá generar un número aleatorio N entre 2000 y 5000 y la máquina esperará N milisegundos a que se establezca la comunicación como cliente.

  Si ello no sucediera se cancelará el intento como cliente para iniciar en modo servidor. El servidor esperará entonces a que se establezca una comunicación, que el usuario cancele el programa o que surja algún error suficientemente catastrófico como para que el programa entienda que el sistema no se puede recuperar y deba abortar su ejecución.

  Vale recordar los ítems 1 y 2 de la tercera regla definida en ***Convenciones y definiciones***, donde se establece que se deberá informar de todo lo que sucede al usuario y se le permitirá salir del programa en cualquier circunstancia.

  Una vez establecida la comunicación se pasará a ***Máquinas conectadas***.

- ***Máquinas conectadas***

  La máquina que al momento de establecerse la conexión había sido iniciada en modo servidor pasa entonces a gobernar la secuencia del juego y la llamaremos servidor. La otra se llamará cliente.

  La primera acción que debe realizar el servidor es pedir el nombre del cliente enviando un paquete NAME (paquete que no posee campo de datos, solamente consiste en un encabezado de un byte).

  El cliente al recibir el paquete NAME deberá contestar con un paquete NAME_IS conformado de la siguiente manera: un byte de encabezado con el valor definido por NAME_IS seguido de un campo de datos de longitud variable. Éste consiste en: un byte que indica la longitud del nombre, en formato unsigned char (de 0 a 255 caracteres de largo) seguido de una cadena variable de bytes que contienen el nombre del jugador en formato ASCII.

  No deberá enviarse ningún carácter no imprimible en el nombre del jugador. Si bien la longitud máxima del nombre está limitada a 255 caracteres no es necesario imprimir todos en la pantalla (está permitido truncar el nombre).

  El paquete de NAME_IS toma entonces la siguiente forma (1 byte por celda):

NAME_IS	**COUNT**	***Nombre***				
0x11	0x05	A(0x41)	R(0x52)	I(0x49)	E(0x45)	L(0x4C)

  Una vez que el servidor recibe el nombre del oponente (paquete NAME_IS) contesta con un ACK (paquete que tampoco contiene campo de datos, se constituye simplemente de un encabezado de 1 byte).

Es ahora el cliente quien debe averiguar el nombre del servidor. Por lo tanto, al recibir el ACK le enviará al servidor un paquete NAME. El servidor contestará con el paquete NAME_IS de la misma forma que antes lo había hecho el cliente y éste último contestará con un ACK al recibir el paquete NAME_IS del servidor.

Cuando el servidor reciba el ACK a su paquete NAME_IS, enviará un paquete MAP_IS conformado de la siguiente manera: un byte de encabezado con el valor definido por MAP_IS seguido por un campo que detalla el mapa: Cada byte enviado será la letra que estaba guardada en el csv, siguiendo el mismo orden que ese archivo (Comienza con A1, A2,... sigue con A16,B1,B2… y termina con L15,L16). Luego del detalle del mapa, habrá un byte de checksum (que puede tomar valores de 0 a 255) que se debe calcular según lo definido en el Anexo 1.

Entonces el paquete MAP_IS toma la siguiente forma (1 byte por celda):

MAP_IS	**A1**	**A2**	**A3**	**…**	**L16**	***Checksum***
0x12	N/N ∈ M	N/N ∈ M	N/N ∈ M		N/N ∈M	N/N ∈ [0x00; 0xFF]

El cliente hará el checksum del mapa al que hace referencia el servidor y de ser válido responderá ACK. Caso contrario responderá con un error siguiendo la política de errores que se especifica más adelante.

Luego, el servidor le indicará al jugador los movimientos iniciales de los enemigos con el paquete ENEMY_ACTION, el cual está explicado en la sección ***Comienzo del juego***. El cliente al recibir aquel paquete, responderá con un ACK.

Por último, el servidor responderá luego de 5 segundos con un paquete GAME_START, el cual no tiene cuerpo. Al llegar este mensaje al cliente, este empezará el juego y responderá con un ACK, y el servidor empezará al llegarle el ACK del cliente.

- ***Comienzo del juego***

Una vez comenzado el juego, ambos jugadores podrán moverse y disparar en cualquier instante. Para comunicar la acción, se diferenciará si el que acciona es el servidor o el cliente. Cualquier acción que ocurra será enviada por el servidor al cliente, mientras que el cliente solo enviará paquetes de request para las acciones que realice su usuario, esperando luego la acción por parte del servidor.

Para indicar un movimiento el servidor utilizará el paquete MOVE, indicando que jugador realiza el movimiento y solamente la posición (fila-columna) de destino. Entonces, el paquete de MOVE toma la siguiente forma:

MOVE	***Player***	**FIL_DE**	**COL_DE**
0x31	T(0x54) o N(0x4E)	N/N ∈ [0x41;0x4C]	N/N ∈ [0x41;0x4C]

Si se quisiera indicar un ataque, el servidor enviará un paquete ATTACK, indicando que jugador la realiza y la última posición que alcanza el disparo (en caso de choca con una pared se pondrá dicha columna). El paquete ATTACK toma la siguiente forma:

ATTACK	***Player***	**FIL_DE**	**COL_DE**
0x32	T(0x54) o N(0x4E)	N/N ∈ [0x41;0x4C]	N/N ∈ [0x41;0x4C]

Para comunicar una acción del cliente, el mismo enviará el paquete ACTION_REQUEST, indicando luego si se trata de un MOVE o un ATTACK y la posición final, que representa a dónde se mueve o la última posición que alcanza el disparo respectivamente.

Entonces, el paquete de ACTION_REQUEST toma la siguiente forma:

ACTION_REQUEST	***Action***	**FIL_DE**	***COL_DE***
0x33	M(0x4D) o A(0x41)	N/N ∈ [0x41;0x4C]	N/N ∈ [0x41;0x4C]

Por último, el servidor se encargará de determinar las acciones de cada monstruo y las comunicará al cliente con el paquete ENEMY_ACTION. Al comenzar a ejecutar la acción de un monstruo el servidor determinará la acción siguiente y la comunicará.

Al principio del juego, con el paquete GAME_START se comenzará a ejecutar las acciones correspondientes y luego a calcular las acciones siguientes.

Cabe aclarar que la determinación de la acción siguiente siempre será definir en la dirección que se mueve el monstruo, quedarse quieto o lanzar un proyectil, tal como fue aclarado en la sección ***Dinámica del juego.***

El paquete ENEMY_ACTION consta de un campo que indica el ID del monstruo, definido en la sección ***Anexo 1***, indicando luego si se trata de un MOVE o un ATTACK y la posición final. El paquete de ACTION_REQUEST toma entonces la siguiente forma:

ENEMY_ACTION	***MonsterID***	***Player***	***FIL_DE***	***COL_DE***
0x34	N/N ∈ [0x00; 0xFF]	T(0x54) o N(0x4E)	N/N ∈ [0x41;0x4C]	N/N ∈ [0x41;0x4C]

En todos los casos, ante un paquete del servidor el cliente le responderá con un ACK.

Al destruir a todos los enemigos de un mapa, los jugadores se podrán mover por el mapa libremente durante 5 segundos. Luego, comenzará la misma rutina de MAP_IS hasta GAME_START del inicio, cargando el mapa del siguiente nivel hasta el nivel 10.

- ***Final del juego***

  El juego termina ante alguna de las siguientes dos situaciones:

  1. Luego de una acción, de un jugador o enemigo, el servidor detecta que el último jugador perdió su ultima vida. En este caso los jugadores perdieron el juego.
  2. Luego de un ataque de un jugador el servidor detecta que se eliminó al último enemigo del nivel (estando en el nivel final). En este caso los jugadores lograron completar el juego.

  Para estos casos el servidor enviará al cliente un paquete GAME_OVER o WE_WON respectivamente los cuales están constituidos únicamente por un encabezado de un byte.

  Al recibir cualquiera de estos dos paquetes, el cliente validará estas situaciones, enviando un paquete ERROR en caso de que no ocurran y actuando según

se establecerá más adelante. Si los paquetes son válidos el cliente avisará de la situación al usuario y lo interrogará para saber si desea volver a jugar. En el caso afirmativo le responderá al servidor con un paquete PLAY_AGAIN y en el caso negativo con un paquete GAME_OVER. Estos paquetes tampoco poseen campos de datos.

Si el servidor recibiera ERROR actuará como se indica más adelante. En el caso de recibir GAME_OVER le explicará a su usuario que del otro lado no desean seguir jugando y contestará con un paquete ACK. En ese momento ambas máquinas se desconectarán cerrando la comunicación entre ellas.

Por último, si se recibiera un paquete PLAY_AGAIN se interrogará al jugador para saber si desea seguir jugando enviándose GAME_OVER en caso negativo (el cliente actuará del mismo modo que se mencionó antes para el servidor al recibirlo) y MAP_IS en caso positivo, continuando a partir de este punto como ya se indicó en la sección ***Máquinas conectadas***. Cabe aclarar que los paquetes NAME y NAME_IS son obviados porque estos solo corresponden a la primera inicialización.

- ***En cualquier momento de la comunicación***

  Cuando la comunicación se encuentre en la etapa descrita en ***Comienzo del juego*** si una máquina no recibe un paquete de respuesta válido después de 2 minutos y medio de enviado el propio, el juego se considera colgado y se finaliza advirtiéndole al usuario que hubo un error de comunicación (se toma un margen de medio minuto para ecualizar cualquier diferencia de relojes que pudiera existir entre máquinas así como el tiempo de tráfico).

  Por otro lado, no se impondrá límite de tiempo fuera de la etapa ***Comiendo del juego*** cuando se espere input del usuario (por ejemplo que envíe su nombre o decida si desea volver a jugar). Para los paquetes de respuesta al input de usuario (ACK, NAME, etc.) se estipula un tiempo de 2 minutos y medio también.

Pueden existir diversos errores de comunicación: por ejemplo, no se ha respondido dentro del tiempo estipulado, se recibe un paquete erróneo, una jugada inválida, un checksum de mapa erróneo, un paquete inexistente, etc.

La máquina que detecta un error de comunicación enviará un paquete de ERROR y luego cerrará el canal de comunicación abierto. No esperará un ACK del otro lado ya que no puede establecer con certeza que la comunicación no se haya corrompido.

En todos los casos si se recibe un paquete de ERROR se debe cerrar el canal de comunicación indicando al usuario que hubo un error en la comunicación.

Se le debe dar al usuario la posibilidad de cerrar el programa en cualquier momento. En dicho caso deberemos informárselo apropiadamente al usuario remoto enviándole el paquete QUIT.

Al recibir un paquete de QUIT, la máquina deberá responder con un paquete ACK, informarle al usuario lo sucedido y cerrar el canal de comunicación. Vale aclarar que la respuesta al paquete QUIT con un paquete ACK también debe circunscribirse en el marco de tiempo de 2 minutos y medio, siendo así, si después de 2 minutos y medio de enviado el paquete QUIT, no se recibiera el paquete ACK del

contrincante, se deberá concluir que hubo un error y se procederá a informarlo de la manera descripta arriba.

Nótese que los paquetes QUIT y ERROR no poseen campo de datos. Están constituidos únicamente por un encabezado de un byte.

- ***Definición de los encabezados de cada paquete:***

Comando	*Valor (hexadecimal)*
ACK	0x01
NAME	0x10
NAME_IS	0x11
MAP_IS	0x12
GAME_START	0x20
MOVE	0x31
ATTACK	0x32
ACTION_REQUEST	0x33
ENEMY_ACTION	0x34
WE_WON	0x40
PLAY_AGAIN	0x50
GAME_OVER	0x51
ERROR	0xFE
QUIT	0xFF

Anexo 1 - Mapas

El diseño particular de los mapas, su estructura, cantidad y tipo de enemigos queda a cargo de los alumnos. Sin embargo, se deberá seguir la notación especificada en este anexo para comunicar con el servidor el mapa al cliente. Como simplificación, los mapas solo podrán tener pisos horizontales.

Especificación de la notación de los mapas

1. El mapa se debe armar en un archivo de texto tipo CSV y su extensión debe ser ".csv".
2. Se constituye de 12 filas por 16 columnas
3. El separador puede ser tanto un carácter ',' como ';'.
4. El terminador de línea puede ser tanto un CR (13d) como un LF (10d) o combinaciones de ellos.
5. La tabla que hace al mapa empieza en la fila A columna 1 termina en la fila L columna 1
6. Cada posición inicial de los enemigos y de los jugadores, y cada piso o pared será especificado con una letra.
7. El mapa es case insensitive (no distingue mayúsculas de minúsculas).
8. Todos los jugadores y enemigos empezarán mirando hacia el centro del mapa (desde las columnas 1 a 8 mirarán hacia la derecha, y de las columnas 9 a 16 a la izquierda).
9. Se le asignará en el programa un ID a cada monstruo de manera de poder identificarlos posteriormente. Ordenando los valores del csv en el sentido de filas ascendentes (A - L) y columnas ascendentes (1- 16), el ID será 0 para el primer monstruo que aparezca en este orden, e irá incrementando en uno para los siguientes.
10. Las letras que representan a cada personaje vendrán dadas por la siguiente tabla:

Personaje	***Letra que lo representa***
Tom	'T' (0x54)
Nick	'N'(0x4E)
Purple Guy	'P'(0x50)
Green Fattie	'G'(0x47)
Crazy	'C'(0x43)
Floor/Wall	'F'(0x46)
Nada	'E'(0x45)

Se llamará al conjunto M como la unión de todos los valores posibles de cada cuadrado del mapa: □ = [□□□□] ∪ [□□□□] ∪ [□□□□] ∪ [□□□□] ∪ [□□□□] ∪ [□□□□] ∪ [□□□□]

Verificación de los mapas (Checksum)

Para asegurarnos de que ninguno de los jugadores alteró el mapa y que por lo tanto ambas máquinas cuentan con la misma información, el servidor deberá enviar un checksum del archivo del mapa que luego será validado por el cliente en su propio archivo de mapa.
El checksum se calcula teniendo en cuenta todos los bytes que constituyen al archivo del mapa aplicando el siguiente algoritmo:

index(n) = Tabla[index(n-1) XOR Archivo(n)]
donde:
n toma valores de 1 hasta la longitud del archivo.
Archivo(n) refiere al byte n-ésimo del archivo.
index toma valores entre 0 y 255.
index(0) = 0.
Tabla constituye un arreglo de 256 bytes definidos que se especificarán en breve.

Resulta entonces checksum = index(longitud del archivo).

```
Tabla = {
     98,  6, 85,150, 36, 23,112,164,135,207,169,  5, 26, 64,165,219, //  1
     61, 20, 68, 89,130, 63, 52,102, 24,229,132,245, 80,216,195,115, //  2
     90,168,156,203,177,120,  2,190,188,  7,100,185,174,243,162, 10, //  3
    237, 18,253,225,  8,208,172,244,255,126,101, 79,145,235,228,121, //  4
    123,251, 67,250,161,  0,107, 97,241,111,181, 82,249, 33, 69, 55, //  5
     59,153, 29,  9,213,167, 84, 93, 30, 46, 94, 75,151,114, 73,222, //  6
    197, 96,210, 45, 16,227,248,202, 51,152,252,125, 81,206,215,186, //  7
     39,158,178,187,131,136,  1, 49, 50, 17,141, 91, 47,129, 60, 99, //  8
    154, 35, 86,171,105, 34, 38,200,147, 58, 77,118,173,246, 76,254, //  9
    133,232,196,144,198,124, 53,  4,108, 74,223,234,134,230,157,139, // 10
    189,205,199,128,176, 19,211,236,127,192,231, 70,233, 88,146, 44, // 11
    183,201, 22, 83, 13,214,116,109,159, 32, 95,226,140,220, 57, 12, // 12
    221, 31,209,182,143, 92,149,184,148, 62,113, 65, 37, 27,106,166, // 13
          3, 14,204, 72, 21, 41, 56, 66, 28,193, 40,217, 25, 54,179,117, // 14
    238, 87,240,155,180,170,242,212,191,163, 78,218,137,194,175,110, // 15
     43,119,224, 71,122,142, 42,160,104, 48,247,103, 15, 11,138,239  // 16
};
```

Bibliografía

1. http://en.wikipedia.org/wiki/Pearson_hashing (Pearson Hashing. Checksum algo.)
2. https://en.wikipedia.org/wiki/Snow_Bros#Gameplay

Aquellos que quiera mejorar su programación en C++ pueden consultar:

1. Effective C++ Third Edition 55 Specific Ways to Improve Your Programs & Designs. Scott Meyers. Addison Wesley Professional. 12/05/2005.
2. ISBN: 0-321-33487-6.
3. Design Patterns: Elements of Reusable Object-Oriented Software. Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides. Addison-Wesley Professional. 10/11/1994. ISBN: 0-201-63361-2.
4. The C++ Standard Library: A Tutorial and Reference. Nicolai M. Josiuttis. Addison Wesley. 06/08/1999. ISBN: 0-201-37926-0.
5. The Art of C++, Herbert Schildt. McGraw-Hill. 12/04/2004. ISBN: 0-072-25512-9.
6. C++: The Complete Reference, 4th Edition, Herbert Schildt. McGraw-Hil, 19/11/2002. ISBN: 0-072-22680-3.

Links adicionales

1. https://www.retrogames.cz/play_658-NES.php Emulador del juego online
2. https://gamefaqs.gamespot.com/nes/587632-snow-brothers/faqs/19225 FAQs