Pontificia Universidad Javeriana
Departamento de Ingeniería de Sistemas
Estructuras de Datos
Proyecto del curso, 2023-30

1. Descripción del problema

Risk® es un juego de mesa basado en turnos, cuyo objetivo es la conquista de territorios a través de representaciones de ejércitos, eliminando jugadores oponentes en el proceso. Cada jugador inicia con una cierta cantidad de unidades de ejército ubicadas en algunos territorios del tablero. Para cumplir el objetivo, cada jugador debe atacar territorios vecinos, intentando ocuparlos en el ataque, lo cual se decide con el lanzamiento de dados. El juego termina cuando un jugador ha cumplido con su misión (que puede ser la ocupación de territorios específicos o incluso la ocupación de todos los territorios del tablero).

1.1. Tablero de juego

El juego utiliza un tablero con 42 territorios geográficos, agrupados en 6 continentes (América del Norte, América del Sur, Europa, África, Asia y Oceanía), los cuales tienen cada uno un color distintivo (amarillo, rojo, azul, café, verde y púrpura, respectivamente). La figura 1 muestra un ejemplo del tablero de **Risk**.

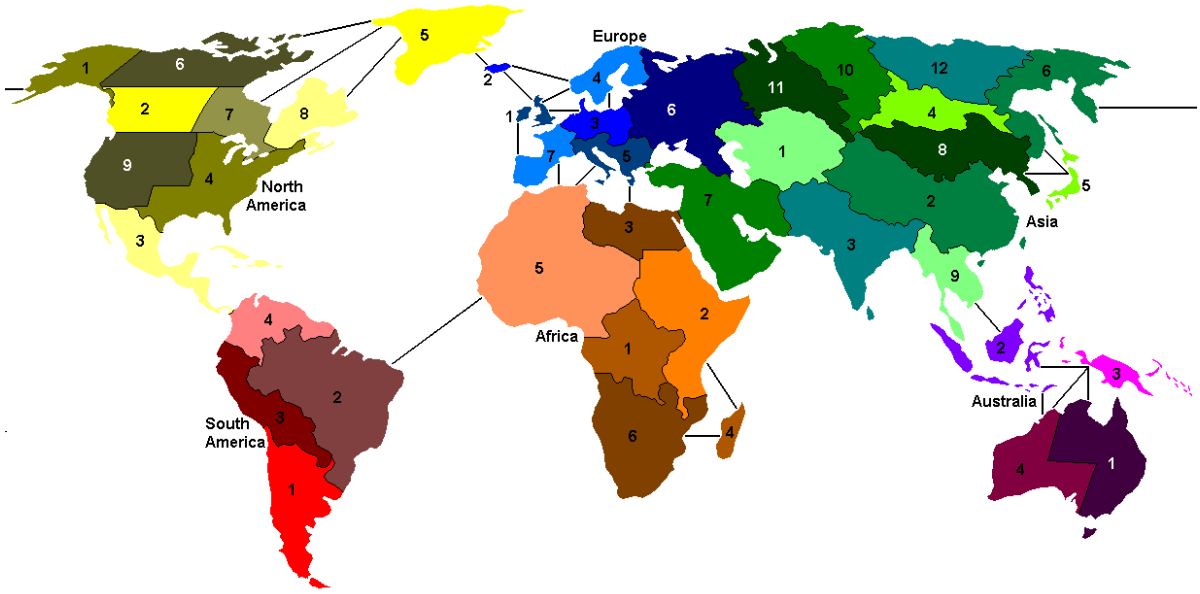


Figura 1: Tablero de juego de **Risk**. «Risk Game Map 2004 Edition» de Demonaire - `http://en.wikipedia.org/wiki/Image:Risk_game_map.png`. Disponible bajo la licencia CC BY-SA 3.0 vía Wikimedia Commons.

En la tabla 1 se describen los nombres de cada uno de los territorios, organizados de acuerdo al continente al que pertenecen.

1.2. Ejércitos

Para la ocupación de los territorios se utilizan 6 grupos de ejércitos, representados por pequeñas figuras que representan la infantería (una unidad de ejército), la caballería (una unidad de caballería equivale a 5 unidades de

América del Norte
1. Alaska
2. Alberta
3. América Central
4. Estados Unidos Orientales
5. Groenlandia
6. Territorio Noroccidental
7. Ontario
8. Quebec
9. Estados Unidos Occidentales

América del Sur
1. Argentina
2. Brasil
3. Perú
4. Venezuela

Europa
1. Gran Bretaña
2. Islandia
3. Europa del Norte
4. Escandinavia
5. Europa del Sur
6. Ucrania
7. Europa Occidental

África
1. Congo
2. África Oriental
3. Egipto
4. Madagascar
5. África del Norte
6. África del Sur

Asia
1. Afghanistán
2. China
3. India
4. Irkutsk
5. Japón
6. Kamchatka
7. Medio Oriente
8. Mongolia
9. Siam
10. Siberia
11. Ural
12. Yakutsk

Australia
1. Australia Oriental
2. Indonesia
3. Nueva Guinea
4. Australia Occidental

Cuadro 1: Nombres de los territorios del tablero de **Risk**.

ejército) y la artillería (una unidad de artillería equivale a 10 unidades de ejército); estos grupos también tienen colores diferentes que permiten distinguir a los jugadores participantes (verde, azul, rojo, amarillo, negro y gris). La figura 2 muestra un ejemplo de las figuras, donde el soldado representa la infantería, el caballo la caballería y el cañón la artillería.

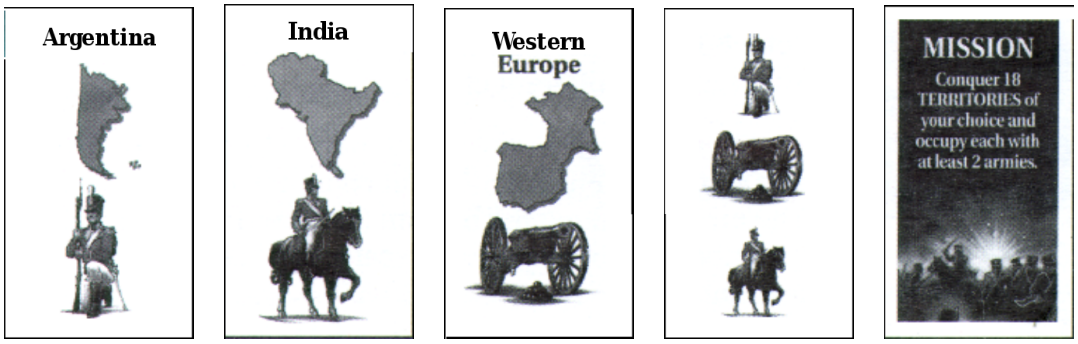
Figura 2: Figuras de ejércitos de **Risk**.

1.3. Tarjetas

El juego incluye también 56 cartas: 42 marcadas con un territorio y el dibujo de una infantería, una caballería o una artillería; 2 cartas "comodín" con los tres dibujos; y 12 cartas de Misión Secreta. Las cartas con los territorios se entregan en cada turno al jugador que ha ocupado un territorio. Las cartas con los ejércitos pueden usarse para solicitar unidades de ejército adicionales en el turno del jugador. Las cartas de Misión Secreta se utilizan sólo en la variante por misiones del juego. La figura 3 presenta algunos ejemplos de las cartas del juego.

1.4. Inicio del juego

El juego comienza con la ocupación de los territorios por los jugadores participantes. Para esto, cada jugador define el color con el que quiere participar, y luego toma una cantidad específica de unidades de infantería, de acuerdo a cuantas personas vayan a participar. Para 3 jugadores, cada uno toma 35 infanterías; para 4 jugadores, cada uno toma 30 infanterías; para 5 jugadores, cada uno toma 25 infanterías; y para 6 jugadores, cada uno toma 20 infanterías. Luego, por turnos cada jugador va ubicando una pieza de infantería en un territorio que él escoja, hasta que todos los territorios hayan sido ocupados por los jugadores, y todas las piezas de infantería asignadas hayan sido ubicadas en el tablero.



Infantería Caballería Artillería Comodín Misión

Figura 3: Ejemplos de tarjetas de **Risk**. `http://www.hasbro.com/common/instruct/risk.pdf`

1.5. Reglas de juego

Una vez ocupados los territorios, comienza el juego, en donde en cada turno el jugador puede escoger atacar territorios enemigos para reclamarlos. Cada turno implica los siguientes pasos:

1. Obtener y ubicar nuevas unidades de ejército. Los jugadores obtienen nuevas unidades de ejército en cada turno de acuerdo a:

   - Territorios: se cuentan los territorios que actualmente ocupa el jugador, se divide este número entre 3, y el resultado es la cantidad de unidades adicionales que puede reclamar.
   - Continentes: si el jugador ocupa todos los territorios de un continente, recibe la siguiente cantidad de unidades adicionales: por América del Sur o Australia, 2 unidades; por África, 3 unidades; por América del Norte o Europa, 5 unidades; y por Asia, 7 unidades.
   - Cartas: al completar 3 cartas en cualquiera de las siguientes combinaciones: con el mismo dibujo de ejército, con los tres dibujos de ejército, o cualquiera de los dos dibujos junto con una carta comodín; éstas se pueden intercambiar por una cantidad de unidades adicionales que dependen de cuántos grupos de cartas se han intercambiado por todos los jugadores: el primer grupo recibe 4 unidades, el segundo 6 unidades, el tercero 8 unidades, el cuarto 10 unidades, el quinto 12 unidades, el sexto 15 unidades, y de ahí en adelante el número de incrementa en 5 unidades cada vez. Adicionalmente, si al intercambiar las cartas algunas incluyen territorios ocupados por el jugador, se reciben 2 unidades extra por cada territorio, las cuales obligatoriamente deben ocuparse en el territorio específico.

2. Atacar un territorio vecino. El jugador debe escoger uno de los territorios que tiene ocupados para iniciar el ataque, y sólo podrá atacar territorios vecinos (pueden ser también aquellos conectados por líneas). Por ejemplo, si el jugador decide iniciar el ataque en Brasil podría atacar tanto a Argentina, Perú y Venezuela en América del Sur como a África del Sur. El resultado del ataque se define a través de los dados: el jugador atacante lanza 3 dados de color rojo, mientras que el jugador que defiende lanza 2 dados blancos. Los dados de uno y otro se emparejan y se comparan para determinar cuántas unidades de ejército pierde o gana cada uno: si el del atacante es mayor que el del defensor, el defensor pierde una unidad de ejército del territorio atacado; si el del defensor es mayor al del atacante, el atacante pierde una unidad de ejército del territorio desde el que se ataca; si hay empate, el defensor es quien gana, por lo que el atacante pierde una unidad de ejército de su territorio. El jugador atacante tiene la ventaja de que puede escoger dos de los tres dados con mayor valor para hacer la comparación. El proceso se repite mientras el atacante lo decida. Si en el ataque el territorio queda vacío (sin piezas del ejército del defensor), el atacante puede reclamarlo moviendo algunas de sus piezas de ejército allí.

3. Fortificar la posición. Antes de finalizar el turno, el jugador puede decidir mover algunas de sus unidades de ejército desde uno (y solo uno) de sus territorios a otro (sólo uno) de sus territorios vecinos.

Este proceso se repite hasta que un sólo jugador haya tomado control de los 42 territorios del tablero, momento en el cual se declara como ganador. Dependiendo de la variante del juego, se puede terminar también cuando un jugador haya cumplido su misión (dada por la tarjeta de Misión Secreta entregada al principio del juego), en cuyo

caso éste se declara como ganador. Para mayor información sobre las reglas de juego, se puede consultar el manual de **Risk** (`http://www.hasbro.com/common/instruct/risk.pdf`).

2. Descripción del proyecto

El objetivo del presente proyecto es construir un sistema de apoyo para el juego **Risk**. El sistema se implementará como una aplicación que recibe comandos textuales, agrupados en componentes con funcionalidades específicas. A continuación se describen los componentes individuales que conforman el proyecto.

2.1. Componente 1: configuración del juego

Objetivo: Los algoritmos implementados en este componente servirán para gestionar la inicialización del juego, así como el desarrollo por turnos del mismo. Este componente se implementará con las siguientes funciones:

- **comando:** `inicializar`
  posibles salidas en pantalla:
  (Juego en curso) `El juego ya ha sido inicializado.`
  (Inicialización satisfactoria) `El juego se ha inicializado correctamente.`
  descripción: Realiza las operaciones necesarias para inicializar el juego, de acuerdo a las instrucciones entregadas. De esta forma, el comando debería preguntar la cantidad de jugadores, para cada jugador su nombre o identificador, y luego, por turnos, preguntar a cada jugador en qué territorio desea ubicar sus unidades de ejército. En cada turno, el jugador sólo puede indicar un único territorio a ocupar. En este comando es clave utilizar una interfaz adecuada de asignación de territorios que sea fácil de seguir para los jugadores en pantalla.

- **comando:** `turno <id_jugador>`
  posibles salidas en pantalla:
  (Juego no inicializado) `Esta partida no ha sido inicializada correctamente.`
  (Juego terminado) `Esta partida ya tuvo un ganador.`
  (Jugador no válido) `El jugador <id_jugador>no forma parte de esta partida.`
  (Jugador fuera de turno) `No es el turno del jugador <id_jugador>.`
  (Turno terminado correctamente) `El turno del jugador <id_jugador>ha terminado.`
  descripción: Realiza las operaciones descritas dentro del turno de un jugador (obtener nuevas unidades, atacar y fortificar). De esta forma, el comando debería primero informar al jugador cuántas unidades adicionales puede reclamar, para luego preguntarle en cuáles de sus territorios las quiere asignar y en qué cantidad. A continuación, debería preguntar la configuración del ataque, desde cuál territorio y hacia cuál territorio, verificando las condiciones ya descritas. Luego debería informar los valores obtenidos con los dados, y la cantidad de unidades que se ganan o pierden. Este proceso se repite hasta que alguno de los dos territorios se quede sin unidades, o hasta que el atacante decida detenerse. Finalmente, el comando debería preguntar al jugador los territorios vecinos que desea seleccionar para la fortificación, así como la cantidad de unidades que se trasladarán de uno al otro. De nuevo, en este comando es clave utilizar una interfaz adecuada que sea fácil de seguir para los jugadores en pantalla.

- **comando:** `salir`
  posibles salidas en pantalla:
  (No tiene salida por pantalla)
  descripción: Termina la ejecución de la aplicación.

2.2. Componente 2: almacenamiento de partidas

Objetivo: Los algoritmos implementados en este componente servirán para guardar y recuperar juegos del disco, utilizando un proceso de compresión/decompresión basado en el algoritmo de codificación de Huffman.

2.2.1. Codificación de Huffman

Un principio muy importante alrededor de la compresión de datos es codificar cada símbolo de un mensaje usando la cantidad mínima posible de bits. Por ejemplo, si un mensaje estuviera escrito en lenguaje ASCII, el cual

tiene 256 símbolos diferentes, la cantidad mínima de bits por símbolo requerida sería de 8. Otro principio esencial es que, aquellos símbolos que aparecen más frecuentemente en un mensaje, sería útil codificarlos con menos bits que aquellos menos frecuentes, de tal forma que el mensaje comprimido ocupe el menor espacio posible.

La codificación de Huffman[1] tiene en cuenta los dos principios anteriores. Provee una forma de representar cada símbolo de un mensaje con la menor cantidad posible de bits, y al mismo tiempo permite codificar cada símbolo con una cantidad variable de bits, dependiendo de su frecuencia de ocurrencia en el mensaje. Para realizar el proceso de codificación y decodificación, se utiliza un árbol de Huffman. Éste es un árbol binario que representa una codificación de un conjunto de símbolos óptima: el símbolo que tenga una frecuencia más alta (el que más se repita) se representa con un número pequeño de bits. Un símbolo poco frecuente se representa con más bits. Un ejemplo de un árbol de Huffman se muestra en la Figura 4.

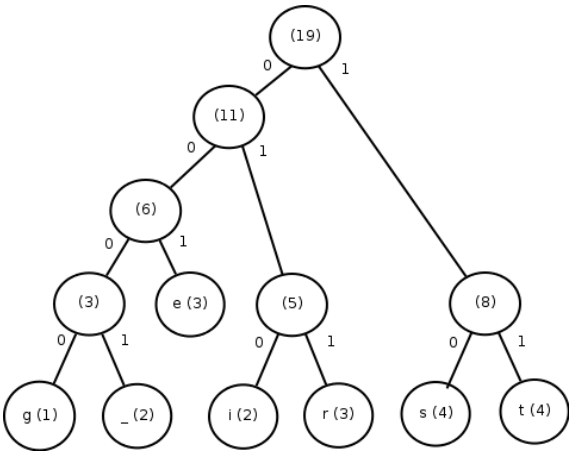


Figura 4: Árbol de Huffman.

En este árbol, cada nodo **hoja** almacena un símbolo del lenguaje utilizado en un mensaje y la cantidad de veces que se encuentra dicho símbolo en el mensaje (valor entre paréntesis), al cual llamaremos "frecuencia". Los **nodos intermedios y la raíz** no contienen símbolos, sino sólo un valor correspondiente a la suma de las frecuencias de sus nodos hijos. Note que los símbolos se encuentran exclusivamente en las hojas y cada símbolo aparece una sola vez en todo el árbol (no se repiten en el árbol).

Este árbol se utiliza para codificar y decodificar los símbolos de un mensaje. Cada arista entre un nodo padre y sus hijos se etiqueta con un "0" para el hijo izquierdo y con un "1" para el hijo derecho. Cada camino entre el nodo raíz y las hojas se puede representar como la concatenación de las etiquetas de las aristas que lo conforman. Esta representación corresponde a la codificación para cada uno de los símbolos. En este caso, por ejemplo, el símbolo que más se repite es la "t" (4 veces), y su codificación es "11", el cual corresponde a las etiquetas de las aristas del camino desde la raíz hasta el nodo con el símbolo "t"

Otros ejemplos de codificación son:

- La "s" se codifica como "10".
- El " _" se codifica como "0001".
- La "r" se codifica como "011".

Por ejemplo, la siguiente matriz de datos (imagen donde cada fila empieza y termina con el símbolo '+'):

```
+-----------------------------------+
+ ------------------------------   +
+ < Soy capaz de hacer el codigo! > +
+ ------------------------------   +
+         \   ^__^                  +
+          \  (oo)\_______          +
+             (__)\       )\/\      +
```

[1] http://en.wikipedia.org/wiki/Huffman_coding

```
+                    ||----w |          +
+                    ||      ||         +
+-----------------------------------+
```

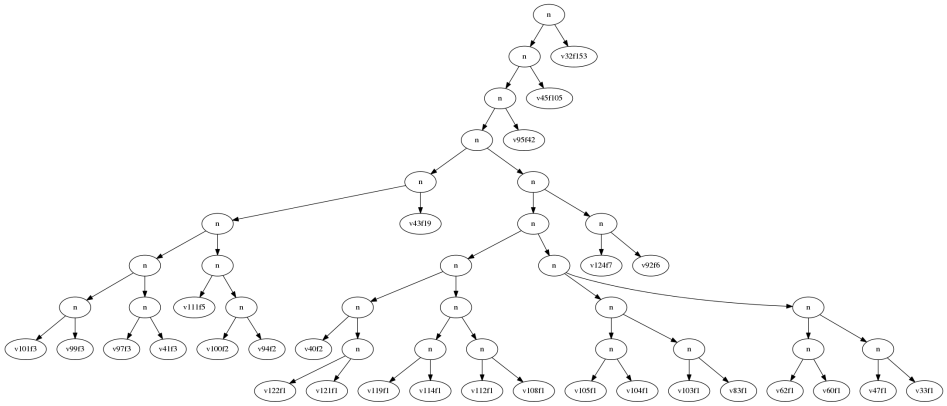
Tiene las siguientes características:

- Ancho de la imagen (W) = 36.
- Alto de la imagen (H) = 10.
- Valor (intensidad) máximo posible de la imagen (M) = 255.
- Los caracteres (que hacen las veces de intensidades) y sus frecuencias son:

ASCII	carácter	frecuencia
32		153
33	!	1
40	(	22
41	)	3
43	+	20
45	-	105
47	/	1
60	<	1
62	>	1
83	S	1
92	\	6
94	^	2
95	_	42
97	a	3

ASCII	carácter	frecuencia
99	c	3
100	d	2
101	e	3
103	g	1
104	h	1
105	i	1
108	l	1
111	o	5
112	p	1
114	r	1
119	w	1
121	y	1
122	z	1
124	|	7

- El árbol asociado es (las hojas tiene asociado un valor "vXfY", donde "X" es el código del símbolo y "Y" es su frecuencia):



2.2.2. Descripción del componente

El objetivo del componente 2 es implementar un árbol de Huffman para codificar y decodificar archivos de juego de Risk. En este contexto, el "mensaje" sería el archivo completo de juego y los "símbolos" del mensaje serían los diferentes caracteres presentes en el archivo (incluyendo cambios de línea y espacios). Dado que la secuencia para codificar un mensaje completo es una secuencia de "0"s y "1"s, este mensaje se puede separar en paquetes de 8 bits y guardar en un archivo binario. El archivo binario (de extension "bin", para el presente proyecto) tiene la siguiente estructura:

$n\ c_1\ f_1 \cdots c_n\ f_n\ w\ binary\_code$

donde:

- n es un número entero de 2 bytes que representa la cantidad de caracteres diferentes presentes en el archivo de juego que se va a almacenar.
- c_i y f_i son dos números entero de 1 y 8 bytes, respectivamente, que representan un caracter (código ASCII) y su frecuencia asociada (cuántas veces aparece en el archivo).
- w un número entero de 8 bytes que representa la longitud del archivo, es decir, la cantidad de caracteres que incluye originalmente.
- $binary\_code$ es la secuencia binaria que representa la i-ésima secuencia. Note que si la secuencia no es múltiplo de 8, se debe completar con los "0" necesarios.

Este componente se implementará entonces con las siguientes funciones:

- **comando:** `guardar <nombre_archivo>`
  salida en pantalla:
  (Juego no inicializado) `Esta partida no ha sido inicializada correctamente.`
  (Comando correcto) `La partida ha sido guardada correctamente.`
  (Error al guardar) `La partida no ha sido guardada correctamente.`
  descripción: El estado actual del juego es guardado en un archivo de texto. El archivo debe contener la cantidad de jugadores, y para cada jugador debe almacenar su nombre, su color de jugador, la cantidad de países que ocupa, el identificador de cada país junto con la cantidad de unidades de ejército en él, la cantidad de tarjetas que posee y el identificador de cada tarjeta. Note que este comando guarda un archivo de texto plano, sin codificación.
- **comando:** `guardar_comprimido <nombre_archivo>`
  salida en pantalla:
  (Juego no inicializado) `Esta partida no ha sido inicializada correctamente.`
  (Comando correcto) `La partida ha sido codificada y guardada correctamente.`
  (Error al codificar y/o guardar) `La partida no ha sido codificada ni guardada correctamente.`
  descripción: El estado actual del juego es guardado en un archivo binario (con extensión .bin) con la información (la misma que se almacenaría en un archivo de texto normal, ver comando `guardar`) comprimida, utilizando la codificación de Huffman en el formato descrito más arriba.
- **comando:** `inicializar <nombre_archivo>`
  posibles salidas en pantalla:
  (Juego en curso) `El juego ya ha sido inicializado.`
  (Archivo vacío o incompleto) "`nombre_archivo`" `no contiene información válida para inicializar el juego.`
  descripción: Inicializa el juego con los datos contenidos en el archivo identificado por `<nombre_archivo>`. El archivo debería contener la información descrita en el comando `guardar`. El comando debe poder inicializar el juego desde un archivo de juego normal (generado por el comando `guardar`) o un archivo binario con los datos comprimidos (generado por el comando `guardar_comprimido`).

2.3. Componente 3: estrategias de juego

Objetivo: Los algoritmos implementados en este componente servirán para ayudar a un jugador a tomar decisiones. Este componente se implementará con las siguientes funciones:

- **comando:** `costo_conquista <territorio>`
  salida en pantalla:
  (Juego no inicializado) `Esta partida no ha sido inicializada correctamente.`
  (Juego terminado) `Esta partida ya tuvo un ganador.`
  (Comando correcto) `Para conquistar el territorio <territorio>, debe atacar desde <territorio_1>, pasando por los territorios <territorio_2>, <territorio_3>, ..., <territorio_m>. Debe conquistar <n>unidades de ejército.`
  descripción: El programa debe calcular el costo y la secuencia de territorios a ser conquistados para lograr controlar el territorio dado por el usuario. El territorio desde donde debe atacar debe ser aquel que el jugador tenga controlado más cerca al dado por el jugador. Esta información se analiza desde el punto de vista del jugador que tiene el turno de juego.

- **comando:** `conquista_mas_barata`
  salida en pantalla:
  (Juego no inicializado) `Esta partida no ha sido inicializada correctamente.`
  (Juego terminado) `Esta partida ya tuvo un ganador.`
  (Jugador no válido) `La conquista más barata es avanzar sobre el territorio <territorio_1>desde el territorio <territorio_2>. Para conquistar el territorio <territorio_1>, debe atacar desde <territorio_2>, pasando por los territorios <territorio_3>, <territorio_4>, ..., <territorio_m>. Debe conquistar <n>unidades de ejército.`
  descripción: De todos los territorios posibles, calcular aquel que pueda implicar un menor número de unidades de ejército perdidas. Esta información se analiza desde el punto de vista del jugador que tiene el turno de juego.

2.4. Interacción con el sistema

La interfaz del sistema a construir debe ser una consola interactiva donde los comandos correspondientes a los componentes serán tecleados por el usuario, de la misma forma que se trabaja en la terminal o consola del sistema operativo. El indicador de línea de comando debe ser el caracter $. Se debe incluir el comando `ayuda` para indicar una lista de los comandos disponibles en el momento. Así mismo, para cada comando se debe incluir una ayuda de uso que indique la forma correcta de hacer el llamado, es decir, el comando `ayuda comando` debe existir.

Cada comando debe presentar en pantalla mensajes de éxito o error que permitan al usuario saber, por un lado, cuando terminó el comando su procesamiento, y por el otro lado, el resultado de ese procesamiento. Los mensajes de éxito o error deben seguir el formato indicado en este enunciado para cada comando. Los comandos de los diferentes componentes deben ensamblarse en un único sistema (es decir, funcionan todos dentro de un mismo programa, no como programas independientes por componentes).

3. Evaluación

Las entregas se harán en la correspondiente actividad de BrightSpace, hasta la media noche del día anterior al indicado para la sustentación de la entrega. Se debe entregar un archivo comprimido (único formato aceptado: .zip) que contenga dentro de un mismo directorio (sin estructura de carpetas interna) los documentos (único formato aceptado: .pdf) y el código fuente (.h, .hxx, .cxx, .cpp). Si la entrega contiene archivos en cualquier otro formato, será descartada y no será evaluada, es decir, la nota definitiva de la entrega será de 0 (cero) sobre 5 (cinco).

3.1. (10 %) Entrega 0: semana 3

La entrega inicial corresponderá únicamente a la interfaz de usuario necesaria para interactuar con el sistema. De esta forma, se verificará el indicador de línea del comando, y que el sistema realice la validación de los comandos permitidos y sus parámetros (Revisar en particular el numeral 2.4).

3.2. (30 %) Entrega 1: semana 6

Componente 1 completo y funcional. Esta entrega tendrá una sustentación durante la correspondiente sesión de clase, y se compone de:

- (30 %) Documento: se debe presentar documento en formato IEEE a doble columna y debe incluir:
  - Título
  - Autores
  - Resumen
  - Palabras clave
  - Introducción
  - Diseño: debe seguir las pautas de ingeniería que usted ya conoce: descripción de entradas, salidas y condiciones para el procedimiento principal y las operaciones auxiliares (comandos). Para la descripción de los TADs utilizados, debe seguirse la plantilla denida en clase. Además, se exigirán esquemáticos

(diagramas, gráficos, dibujos) que describan el funcionamiento general de las operaciones (comandos) principales.

  - Resultados obtenidos
  - Conclusiones
  - Referencias

- (5 %) Plan de pruebas. Adjuntar al documento de diseño un plan de pruebas, que siga las pautas vistas en clase, para el comando `inicializar`.
- (30 %) Código fuente compilable en el compilador gnu-g++ v4.0.0 (mínimo). Si el código fuente genera errores al momento de compilar con los parámetros vistos en clase, se obtiene un «0» como nota. Este porcentaje de la entrega será un promedio de la evaluación de cada comando.
- (35 %) Sustentación con participación de todos los miembros del grupo.

3.3. (30 %) Entrega 2: semana 12

Componentes 1 y 2 completos y funcionales. Esta entrega tendrá una sustentación durante la correspondiente sesión de clase, y se compone de:

- (25 %) Documento: mismas pautas de la entrega 1, y debe ser un documento acumulativo, incluyendo todo lo de la entrega anterior. Se debe entregar 2 documentos (ambos con la misma información) y uno de ellos debe tener resaltado la información nueva incluida en esta entrega.
- (10 %) Plan de pruebas. Adjuntar al documento de diseño un plan de pruebas, que siga las pautas vistas en clase, para el comando `guardar_comprimido`.
- (25 %) Código fuente compilable en el compilador gnu-g++ v4.0.0 (mínimo). Si el código fuente genera errores al momento de compilar con los parámetros vistos en clase, se obtiene un «0» como nota. Este porcentaje de la entrega será un promedio de la evaluación de cada comando.
- (40 %) Sustentación con participación de todos los miembros del grupo.

3.4. (30 %) Entrega 3: semana 18

Componentes 1, 2 y 3 completos y funcionales. Esta entrega tendrá una sustentación (del proyecto completo) entre las 8am y las 12m del último día de clase de la semana 18, y se compone de:

- (25 %) Documento: mismas pautas de la entrega 1, y debe ser un documento acumulativo, incluyendo todo lo de la entrega anterior. Se debe entregar 2 documentos (ambos con la misma información) y uno de ellos debe tener resaltado la información nueva incluida en esta entrega.
- (10 %) Plan de pruebas. Adjuntar al documento de diseño un plan de pruebas, que siga las pautas vistas en clase, para el comando `costo_conquista`.
- (25 %) Código fuente compilable en el compilador gnu-g++ v4.0.0 (mínimo). Este porcentaje de la entrega será un promedio de la evaluación de cada comando.
- (40 %) Sustentación con participación de todos los miembros del grupo.