Simulador de Risk - Manual de usuario

Jair Santiago Vargas Saenz, Juan Pablo Hernández Ceballos, Pablo Enrique Quintero Callamand

Bogotá D.C,

Colombia

23 de Agosto de 2023

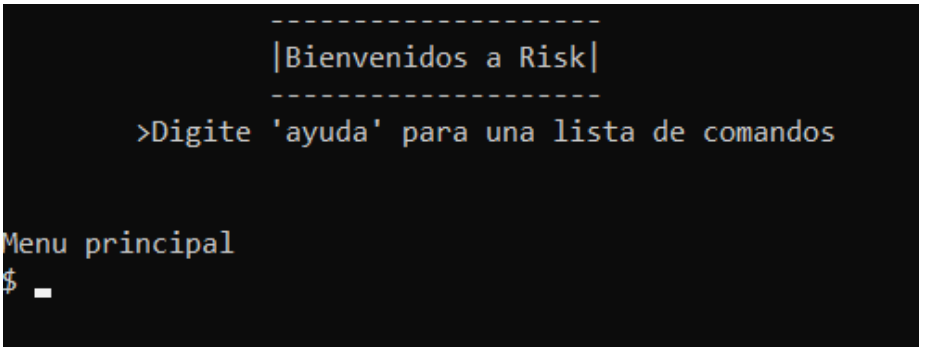
Facultad de Ingeniería, Pontificia Universidad Javeriana
Estructuras de datos - Ing. John Corredor Franco
Proyecto de clase

Introducción del programa

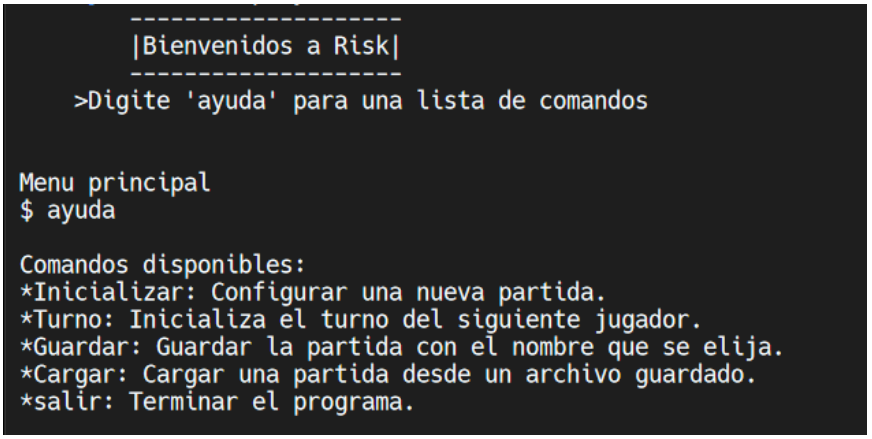
Este programa de consola está diseñado para simular el juego de mesa Risk, donde el usuario podrá empezar una partida nueva (o cargar una ya empezada). Además de simular todos los aspectos necesarios para jugar una partida completa de Risk (orden de turnos, lanzamiento de dados, recibir cartas, asignar unidades, etc.), el programa también cuenta con la opción de guardar una partida en curso a un archivo de texto o a un archivo binario comprimido para que el usuario pueda cargar más adelante su partida y continuar donde la dejó.

Interfaz de usuario

La primera vez que se abre el programa se le muestra al usuario el menú principal, junto con el recordatorio de que puede usar el comando *ayuda* para obtener una lista de comandos válidos.

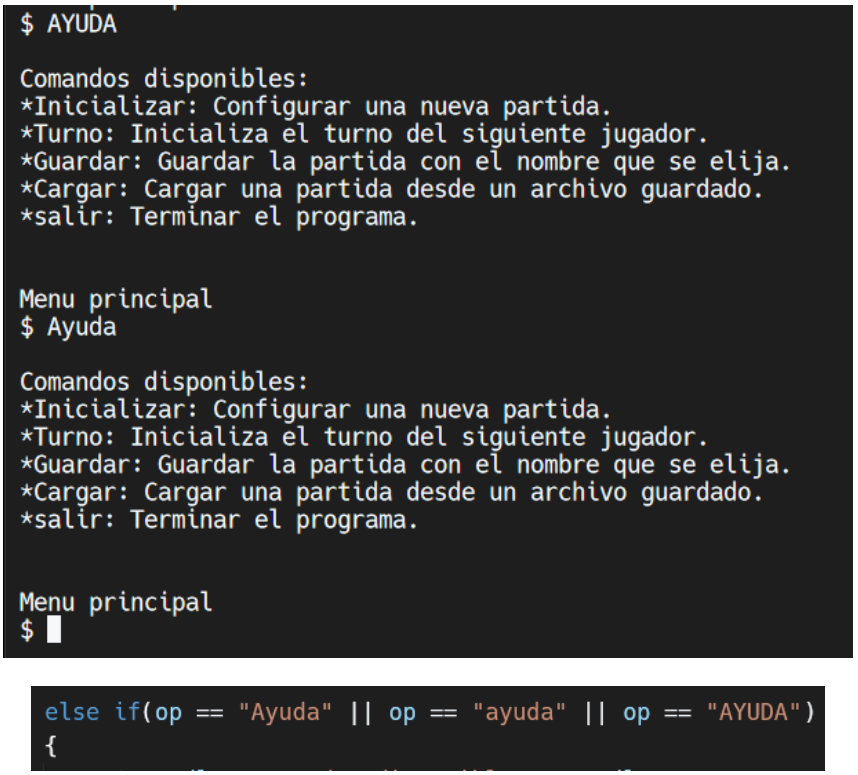


Menú principal del programa



Comando de ayuda

El programa está diseñado para poder reconocer los comandos a pesar de que estén en mayúscula



Ejemplo de identificación de comandos

Comandos de juego

Como ya se mencionó, el programa cuenta con todos los comandos necesarios para crear, guardar y emular una partida de Risk.

Inicialización de partidas

El comando *Inicializar* lleva al usuario al menú de configuración de una nueva partida, donde se le pedirá que establezca el número de jugadores (de 3 a 6), para posteriormente preguntarle a cada jugador que introduzca su identificador, escoja un color de unidad y se le asigna su cantidad de unidades de infantería, dependiendo de la cantidad de jugadores. Una vez todos los jugadores estén registrados se le preguntará a cada uno en qué territorio desean poner sus unidades.

Siguiente Turno

El comando *Turno* pregunta por el identificador del jugador cuyo turno sigue, en el caso de dar el id de un jugador cuyo turno no es el siguiente el programa dará un mensaje de error. Cuando el jugador correcto empiece su turno, podrá hacer cualquiera de las acciones permitidas durante su turno:

1. *Recibir unidades:*

Primero se le informará al jugador cuántas unidades puede reclamar.

2. *Fortificar un territorio suyo:*

El jugador debe escoger qué territorio fortificar, cuantas unidades va a trasladar y desde qué territorio.

3. *Atacar un territorio adyacente:*

El jugador escogerá un territorio al que atacar, se le informará a qué jugador le pertenece y ambos pelearán, la batalla continuará hasta que el atacante gane, pierda o el defensor se rinda. Al final, se le dirá a ambos jugadores cuántas unidades tienen y, si el atacante ganó, se le preguntará cuántas unidades va a mover a su territorio.

Guardar una partida

Con el comando *Guardar* el usuario podrá generar un archivo, con el nombre que él defina, con toda la información de la partida actual para que pueda terminar el programa sin perder el progreso. Se puede escoger entre generar un archivo de texto o un archivo binario comprimido.

Cargar una partida

Con el comando *Cargar* el usuario podrá cargar el archivo generado por el comando *Guardar* y continuar con su partida, simplemente debe ingresar el nombre con el que guardó su archivo.

Comandos de estrategia

Se cuenta con comandos adicionales de ayuda que cada jugador podrá activar durante su turno para recibir estrategias y ayuda del programa

Guia de conquista
Conquista más barata

Diseño de TADs

A partir del análisis de los requisitos y funcionalidades necesarias para la implementación del juego, se han desarrollado los siguientes Tipos Abstractos de Datos (TADs). Estos TADs se diseñaron para representar de manera eficiente y estructurada las distintas entidades y componentes del juego, permitiendo una gestión ordenada de los jugadores, territorios, cartas y demás elementos que componen la mecánica del juego

TAD de comando

Atributos:

comando: Un string que representa el nombre del comando.

argumento: Un string que representa el argumento del comando, si lo hay.

Operaciones:

crear_comando(comando, argumento): Crea un nuevo comando con los atributos especificados.

obtener_comando(comando): Devuelve el comando.

obtener_argumento(comando): Devuelve el argumento del comando, si lo hay.

TAD de tablero

Atributos:

num_jugadores: Un número entero que representa el número de jugadores en el juego.

jugadores: Un vector de jugadores que representa los jugadores en el juego.

turnos: Un vector de strings que representa los turnos de los jugadores en el juego.

Operaciones:

inicializar_juego(num_jugadores, jugadores): Inicializa el juego con el número especificado de jugadores y los jugadores especificados.

asignar_turno(jugador): Asigna el turno al jugador especificado.

obtener_turno(jugador): Devuelve el turno del jugador especificado.

obtener_costo_conquista(casilla): Devuelve el costo de conquistar la casilla especificada.

obtener_conquista_mas_barata(): Devuelve la casilla con el costo más bajo para conquistar.

TAD de jugador

Atributos:

nombre: Un string que representa el nombre del jugador.

fichas: Un número entero que representa el número de fichas del jugador.

Operaciones:

crear_jugador(nombre, fichas): Crea un nuevo jugador con los atributos especificados.

obtener_nombre(jugador): Devuelve el nombre del jugador.

obtener_fichas(jugador): Devuelve el número de fichas del jugador.

TAD de casilla

Atributos:

jugador: Un string que representa el jugador que controla la casilla.

costo_conquista: Un número entero que representa el costo de conquistar la casilla.

Operaciones:

crear_casilla(jugador, costo_conquista): Crea una nueva casilla con los atributos especificados.

obtener_jugador(casilla): Devuelve el jugador que controla la casilla.

obtener_costo_conquista(casilla): Devuelve el costo de conquistar la casilla.

TAD Territorio

Conjunto mínimo de datos:

- nombre_territorio, string, determina el nombre o título del territorio
- Color_territorio, string, determina el color asignado para el territorio
- propietario_territorio, string, determina el jugador dueño del territorio
- territorios_alrededor, vector de territorios, determina los territorios circundantes
- unidadesDeEjercito_territorio, int, determina las unidades de ejercito en el territorio

Comportamiento:

- Territorio(string n, string c), constructor de objetos de tipo territorio
- getNombre(), retorna un string con el nombre del territorio
- getPropietario(), retorna un string con el id del jugador propietario del territorio
- establecerPropietario(string prop), establece que jugador es dueño de ese territorio
- verificarExistenciaPropietario(), retorna un booleano si el territorio fue asignado a un jugador
- agregarTerritorioAlrededor (Territorio* ter), establece los territorios que se encuentran al lado de otros
- getTerritorioAlrededor(), retorna un vector de tipo territorio con todos los territorios colindantes
- getUnidadesDeEjercito_territorio(), retorna un int con las unidades de ejército en el territorio
- setUnidadesDeEjercito_territorio(int cantidad_unidades), establece las unidades de ejercito en un territorio

- eliminarUnidadesTerritorio(), elimina una unidad de territorio

TAD Continente

Conjunto mínimo de datos:

- nombre_continente, string, determina el nombre del continente

- color_contienete, string, determina el color asociado al continente

- territorios_contiente, vector de territorios, determina los territorios asociados a un continente

Comportamiento:

- Continente(string n, string c), constructor de objetos de tipo continente

- getNombre(), retorna un string con el nombre del continente

- getColor(), retorna un string con el color asociado al continente

- getTerritoriosContinente(), retorna un vector de territorios asociados a un continente

- agregarTerritorios(Territorio t), agrega un territorio al vector de territorios asociado al continente

TAD Tarjeta

Conjunto mínimo de datos:

- tipo, string, determina el tipo de la tarjeta (artilleria, infanteria y caballeria)
- territorio, territorio, determina el territorio asociado a una tarjeta
- unidaesDeEjercito_tarjeta, determina las unidades de ejercito de la tarjeta asociadas a su tipo

Comportamiento:

- Tarjeta(string t, Territorio* te, int e), constructor de objetos de tipo tarjeta
- Tarjeta(string t, int e), constructor de objetos de tipo tarjeta los cuales no tienen ningún territorio asignado
- getTipo(), retorna un string con el tipo de una tarjeta
- getTerritorio(), retorna la referencia a un objeto de tipo territorio asociado a una carta
- getEjercito(), retorna las unidades de ejército asociadas con el tipo de la tarjeta

TAD Jugador

Conjunto mínimo de datos:

- nombre_jugador, string, determina el nombre del jugador

- id_jugador, string, determina el id asociado a cada jugador
- color_jugador, string, determina el color asociado a cada jugador
- unidadesDeEjercito_jugador, int, determina las unidades de ejército que puede usar cada jugador
- tarjetas_jugador, vector de tarjetas, determina las tarjetas que tiene un jugador
- territorios_jugador, vector de territorios, determina los territorios que el jugador ha conquistado

Comportamiento:

- **Jugador(string n, string c, string id), constructor de objetos de tipo jugador**
- **getNombre(); retorna un string con el nombre del jugador**
- **getColor(), retorna un string con el color asociado al jugador**
- **getId(), retorna un string con el id asociado a cada jugador**
- **getUnidadesDeEjercito(), retorna un int con las unidades de ejército del jugador**
- **setUnidadesDeEjercito(int u), establece las unidades de ejército del jugador**
- **agregarTarjeta (Tarjeta* t), añade una tarjeta al vector de tarjetas del jugador**
- **mostrarTarjetas (), muestra por pantalla un vector con las tarjeta del jugador**
- **agregarTerritorio(Territorio* t), añade territorios al vector de territorios asociado al jugador**
- **getTerritorios_jugador();, retorna un vector de territorios asociados al jugador**
- **eliminarUnidadesEjercito(), elima una unidad de ejercito del jugador**

TAD Tablero

Conjunto mínimo de datos:

- continentes_tablero, lista de continentes, determina todos los continentes del juego

- jugadores_tablero, vector de jugadores, determina todos los jugadores que participan en el juego

- tarjetas_tablero, vector de tarjetas, determina todas las tarjetas que se van a repartir en el juego

Comportamiento:

- inicializarJuego(int cantJug, vector<Jugador> jugadores), inicializa todos los objetos y valores necesarios para el juego

- inicializarContientes(), inicializa todos los continentes con sus respectivos territorios para el juego

- inicializarJugadores(vector<Jugador>jugadores), inicializa a todos los jugadores que van a participar en el juego

- inicializarTarjetas(), inicializa todas las tarjetas que se requieren para el juego

- agregarContinentes(), añade continentes al juego para su posterior inicialización

- tirarDados(), funcion auxiliar que sirve para atacar y defenderse

- calcularCantUnidadesNuevas(string idJug), determina las cantidades nuevas de unidades de ejercito para cada jugador a partir de combinaciones de tarjetas

- faseDraft(). determina la primera fase de cada jugador en un turno donde despliega sus unidades de ejército

- faseAtacar(, determina la segunda parte de cada jugador en un turno donde el jugar ataca para conseguir nuevos territorios

- faseFortalecer(), determina la tercera parte de cada jugador en donde moviliza unidades de ejército para defender territorios.

- faseDraft(). determina la primera fase de cada jugador en un turno donde despliega sus unidades de ejército

- faseAtacar(, determina la segunda parte de cada jugador en un turno donde el jugar ataca para conseguir nuevos territorios

- faseFortalecer(), determina la tercera parte de cada jugador en donde moviliza unidades de ejército para defender territorios.

TAD Comando

Conjunto mínimo de datos:

- comando, string nombre y string descripción, atributo generado a partir de una estructura conformada por el nombre y el argumento de un comando ingresado por consola
- extern const std::vector<Comando> comandos, Comando, vector que almacena objetos de tipo comando

Comportamiento:

- ayuda(string comando_especifico =""), imprime por pantalla todos los comando de ayuda

- validarComandoInicializar(string argumento), valida los argumentos del comando inicializar

- validarComandoGuardar(string argumento), valida los argumentos del comando guardar

- validarComandoTurno(string argumento), , valida los argumentos del comando turno

- validarComandoGuardarComprimido(string argumento),, valida los argumentos del comando guardar comprimido

- validarComandoCostoConquista(string argumento), valida los argumentos del comando costo_conquista

- vector <Jugador> inicializarSinArgumento(int& numjug), retorna los jugadores que van a participar en la partida

- inicializarConArgumento(const string &argumento), valida el argumento del comando de inicializar con argumento