Proyecto entrega 1 – DISEÑO

ANDRES FELIPE RUGE

JONATHAN ISAAC JURADO SARMIENTO

NICOLAS AGUILAR CHAPARRO

ANTON PATRIGNANI

OSCAR DANILO MARTINEZ BERNAL

PONTIFICIA UNIVERSIDAD JAVERIANA

FACULTAD DE INGENIERÍA

ESTRUCTURA DE DATOS

BOGOTA D.C.

2023

Acta de evaluación

Comentarios Textuales

1.1 Se notaron errores de conexión de flechas en los diagramas de flujo, principalmente en sus finalizaciones.
1.2 Se identifico, el funcionamiento incorrecto del comando f*ortificar* y se corrigió.
1.3 Se agrego la función de *Atacar*, la cual no se había implementado en la entrega anterior y sus subfunciones (funciones utilizadas por esta).
1.4 Se anexo la función de *Obtener Unidades*, la cual no se había implementado en la entrega anterior y sus subfunciones.
1.5 Se anexo a los TADs correspondientes las funciones (Atacar *y Obtener Unidades)* implementadas.

Correcciones realizadas con respecto a la primera

1.1
Se notaron errores de conexión de flechas en los diagramas de flujo, principalmente en sus finalizaciones.

- Se corrigieron los errores presentados en los diagramas de flujo siguiendo las indicaciones dadas por el profesor en la exposición de la entrega 1 del proyecto, su corrección se puede ver reflejada en la sección de este documento con respecto a los diagramas.

1.2 – 1.3 – 1.4
Principalmente las modificaciones que se realizaron fueron en los métodos que se emplean para realizar turno, debido a que solo se tenía una idea de cómo fortificar, pero se estaba realizando de manera incorrecta debido a que solo agregaba tropas a un territorio, pero esas tropas eran del jugador en general y no el desplazamiento de un territorio a otro. Mientras que, Por otro lado, para las acciones de asignar nuevas tropas y atacar se tuvo que implementar y crear desde cero ya que no se había creado nada con respecto a eso, por lo que a continuación explico cómo funciona los nuevos métodos bases creados para la correcta ejecución de turno:

1.fortificar

El método "fortificar" en la clase Risk es responsable de permitir que un jugador coloque fichas en sus territorios durante la fase de fortificación del juego. El jugador puede seleccionar un territorio de su propiedad y transferir un número específico de fichas disponibles a ese territorio.

El método comienza mostrando información relevante sobre el jugador en turno, como su nombre, color y la cantidad de fichas disponibles. Luego, muestra los territorios que el jugador posee para que pueda seleccionar uno en el que desee fortificar.

Después de ingresar el nombre del territorio, el método verifica si el territorio es válido y pertenece al jugador. Si el territorio no es válido, se muestra un mensaje de error y se solicita nuevamente al jugador que ingrese un nombre de territorio válido.

Una vez que se ha seleccionado un territorio válido, el jugador debe ingresar el número de fichas que desea mover desde sus fichas disponibles a ese territorio. El método verifica que el número de fichas ingresado sea válido y no exceda la cantidad de fichas disponibles. Si el número de fichas es inválido, se solicita nuevamente al jugador que ingrese un número válido.

Finalmente, el método llama a la función "moverFichasJugador" de la clase Risk para realizar la transferencia de fichas desde las fichas disponibles del jugador al territorio seleccionado. Si el jugador ha utilizado todas sus fichas disponibles, se llama a la función "turnoJugado" para pasar al siguiente jugador en turno.

En resumen, el método "fortificar" permite al jugador fortificar sus territorios durante la fase de fortificación del juego, seleccionando un territorio y transfiriendo fichas desde sus fichas disponibles a ese territorio. Este proceso se repite hasta que el jugador haya utilizado todas sus fichas disponibles.

2.atacar

El método "atacar" en la clase Risk se encarga de manejar la fase de ataques del juego. Durante esta fase, un jugador puede seleccionar un territorio de su propiedad para atacar a un territorio colindante controlado por otro jugador.

El método comienza mostrando información relevante sobre el jugador en turno, como su nombre y color, así como los territorios que posee. Luego, el jugador selecciona un territorio propio desde el cual desea lanzar un ataque.

El método verifica que el territorio seleccionado sea válido y pertenezca al jugador. Si el territorio no es válido, se muestra un mensaje de error y se solicita nuevamente al jugador que ingrese un nombre de territorio válido.

Después de seleccionar un territorio propio, el jugador puede elegir un territorio colindante para atacar. El método muestra los territorios colindantes disponibles y permite al jugador seleccionar uno de ellos. Se verifica que el territorio colindante sea válido y no pertenezca al mismo jugador. Si el territorio colindante no es válido, se muestra un mensaje de error y se solicita nuevamente al jugador que seleccione un territorio válido.

Una vez que se ha seleccionado un territorio colindante válido, se realiza el lanzamiento de dados y se determina el resultado del ataque. El método muestra el resultado del ataque y permite al jugador decidir si desea continuar combatiendo o no.

El jugador puede elegir seguir combatiendo con el mismo territorio o pasar a la siguiente fase del juego. El método muestra un mensaje al jugador y espera su respuesta. Si el jugador elige continuar combatiendo, se repite el proceso de lanzamiento de dados y resultado del ataque.

El método continúa ejecutándose hasta que el jugador decida pasar a la siguiente fase del juego. Una vez que el jugador decide pasar de fase, se sale del bucle y se finaliza la fase de ataques.

3.ResultadoAtaque

El método "resultadoAtaque" en la clase Risk simula un ataque entre dos territorios en el juego de Risk. Aquí está un resumen:

Se obtienen los jugadores atacante y defensor a partir del territorio defensor y se verifica que el atacante no esté atacando a sí mismo.

Se lanzan dados para el atacante y el defensor, generando resultados aleatorios entre 1 y 6, y se almacenan en vectores de enteros.

Se ordenan los resultados de los dados de mayor a menor para ambos jugadores para una comparación efectiva.

Se comparan los resultados de los dados del atacante y defensor para determinar cuántos dados se deben comparar. Se actualizan las unidades de ejército de cada jugador, restando las unidades pérdidas durante el ataque.

Si el territorio defensor queda vacío después del ataque, el atacante puede reclamarlo, actualizando el jugador propietario y eliminando el territorio de la lista del jugador defensor

Se muestra en la consola quién ganó el ataque según la cantidad de unidades perdidas por cada jugador. El resultado del ataque se muestra al final del proceso.

El método no devuelve un valor específico y se utiliza principalmente para simular y resolver un ataque entre territorios en el juego de Risk. En resumen, es una función que realiza la mecánica de los combates en el juego y muestra el resultado en la consola.

4.obtenerNuevasTropas

El método "CantidadNuevasTropas" en la clase Risk calcula la cantidad de nuevas tropas que un jugador puede recibir al comienzo de su turno. Aquí está un resumen:

Inicializa "nuevasUnidades" con un valor de 10, que son las tropas básicas al inicio del turno.

Calcula la cantidad de territorios ocupados por el jugador y divide este número por 3, sumándolo a "nuevasUnidades". Esto representa tropas adicionales por territorios ocupados.

Itera sobre los continentes para verificar si el jugador ha ocupado todos los territorios de cada continente. Si es así, agrega tropas adicionales basadas en el continente.

Verifica si el jugador tiene al menos 3 cartas en su mano. Calcula la cantidad de tropas adicionales que el jugador puede recibir por tener tríos de cartas.

Suma tropas adicionales si el jugador tiene un trío de cartas del mismo tipo (infantería, caballería o artillería). También suma si hay un trío de cartas de tipos diferentes o un par de cartas más un comodín.

Verifica si las cartas utilizadas en los tríos corresponden a territorios que el jugador posee y suma más tropas adicionales en ese caso.

Elimina las cartas utilizadas en los tríos de la mano del jugador.

El método devuelve el valor final de "nuevasUnidades", que representa la cantidad total de nuevas tropas que el jugador recibirá al comienzo de su turno. En resumen, se calculan las

tropas iniciales del jugador basándose en territorios, continentes ocupados y combinaciones de cartas, proporcionando una cantidad estratégica de tropas al inicio de su turno.

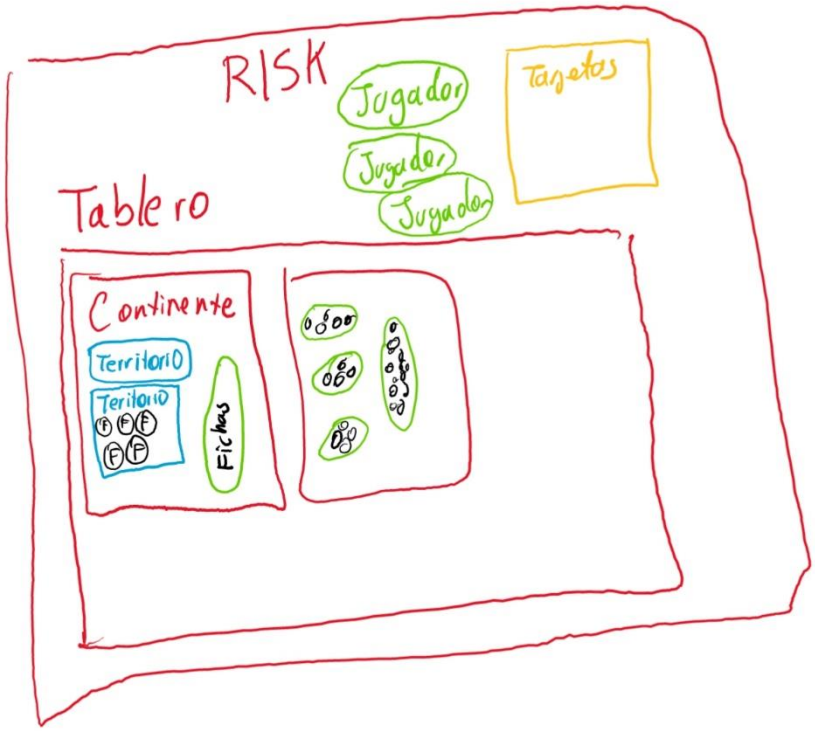
1.5
Se anexo a los TADs correspondientes las funciones (Atacar *y Obtener Unidades)* implementadas.

- Dado que las funciones no estaban implementadas, se realizaron como ya se explicó anteriormente y así mismo se anexaron sus funciones al diagrama de tads, aquí unos ejemplos:

territoriosColindantes(nombreTerritorio): - Obtiene los territorios colindantes de un territorio específico.

- resultadoAtaque(Territorioatacante, TerritorioDefensor): - Realiza el resultado de un ataque entre dos territorios.

Diseño de TADs



Estructuras del proyecto con operaciones principales:

1. Risk

- TAD Risk
    - Conjunto mínimo de datos
        - Jugadores, lista de jugadores, almacena todos los jugadores de partida
        - Cartas, lista de tarjetas, cartas con comodines, misiones y cartas con territorio y fichas
        - Continentes, lista de continentes, tiene los 6 continentes del juego.
        - Partida, verdadero cuando la partida se ha iniciado, falso cuando no se ha empezado una partida.
        - Ganador, verdadero cuando hay un ganador, falso cuando no hay ganador
        - TurnoActual, numero entero de 0 a 5 que indica cual es jugador en turno
        - Totalturnos, numero entero, indica el total de turnos que se han realizado
        - GrupoDeCartas, es la cantidad de cartas que se han intercambiado a lo largo del juego
    - Comportamiento (operaciones) del objeto
        - Risk(): - Constructor de la clase Risk.
        - iniciarPartida():- Inicia una nueva partida de Risk.
        - estadoPartida(): - Verifica el estado de la partida.
        - asignarGanador(): - Verifica si hay un ganador en la partida.
        - crearContinente(): - Crea un continente dentro del tablero.
        - CrearTarjetas(tipo, territorio, ficha, mision): - Crea una tarjeta de juego con los parámetros especificados.
        - resultadoAtaque(Territorioatacante, TerritorioDefensor): - Realiza el resultado de un ataque entre dos territorios.
        - CrearJugador(nombre, qJugadores): - Crea un jugador con el nombre especificado y crea su batallón inicial basado en la cantidad de jugadores.
        - colorJugador(): - Obtiene el color del jugador en turno.
        - moverFichasJugador(qFichas, continente, territorio): - Mueve una cantidad de fichas de un jugador a un territorio específico en un continente.
        - territoriosLibres(): - Verifica si hay territorios desocupados.
        - setGrupo_de_Cartas(valor): - Establece el valor del grupo de cartas en el juego.
        - qUnidades(qJugadores): - Calcula la cantidad de unidades basada en la cantidad de jugadores.
        - InicializarTerritoriosColindantes(risk): - Inicializa los territorios colindantes de cada territorio.
        - agregarTerritorioaJugador(nombreIngresado, nuevoTerritorio): - Agrega un territorio a un jugador dado su nombre.
        - AgregarTropas(jugador, total):- Agrega un total de tropas a un jugador.
        - CantidadNuevasTropas(jugador):- Calcula la cantidad de nuevas tropas que recibe un jugador en su turno.
        - infoContinente(): - Obtiene información de los continentes.
        - infoJug(): - Obtiene información de los jugadores.

- infoTerritorios(nameContinente): - Obtiene información de los territorios en un continente específico.
- estadoTerritorio(nameContinente, nameTerritorio): - Obtiene el estado de un territorio en un continente específico.
- turnoJugado(): - Aumenta en 1 el contador de turnos para avanzar al siguiente jugador en turno.
- getNameJugadorEnTurno(): - Obtiene el nombre del jugador en turno.
- getColorJugadorEnTurno(): - Obtiene el color del jugador en turno.
- indiceContinente(continente): - Obtiene el índice de un continente en la lista de continentes.
- indiceTerritorio(iContinente, territorio): - Obtiene el índice de un territorio en un continente específico.
- territoriosJugador(): - Obtiene los territorios de un jugador.
- getFichasJugadorEnTurno(): - Obtiene la cantidad de fichas que tiene el jugador en turno.
- buscarContinenteTerritorio(territorio): - Busca el continente al que pertenece un territorio.
- territorioJugador(continente, territorio): - Verifica si un territorio pertenece a un jugador en un continente específico.
- territoriosColindantes(nombreTerritorio): - Obtiene los territorios colindantes de un territorio específico.
- buscarTerritorio(nombreContinente, nombreTerritorio): - Busca un territorio por su nombre y el nombre del continente al que pertenece.
- getJugador(nombreJugador): - Obtiene un jugador por su nombre.
-
- getGrupo_de_Cartas(): - Obtiene el valor del grupo de cartas.
- estadoGanador(): - Verifica si hay un ganador en la partida.
- esTurnoJugador(nombreIngresado): - Verifica si es el turno del jugador con el nombre ingresado.
- jugadorExiste(nombreIngresado): - Verifica si existe un jugador con el nombre ingresado.
- turnosEnCero(): - Reinicia el contador de turnos a cero.
- territorioPerteneceAJugador(territorio): - Obtiene el jugador al que pertenece un territorio.
- LanzarDado(): - Lanza un dado y obtiene un número aleatorio de 1 a 6.

2. Jugador

- TAD Jugador
  - Conjunto mínimo de datos
    - Color, indica el color del jugador (verde, azul, rojo, amarillo, negro, gris)
    - Cartas, lista de cartas del jugador
    - Fichas, lista de fichas del jugador
    - NombreJugador, cadena de caracteres que indica el nombre del jugador en la partida.
    - Territorios,vector de punteros de territorios
  - Comportamiento (operaciones) del objeto
    - Jugador(std::string nombre, std::string nColor): - Constructor de la clase Jugador.

- obtenerColor(): - Obtiene el color del jugador.
- obtenerCartas(): - Obtiene las cartas del jugador.
- obtenerTotalFichas(): - Obtiene el total de fichas del jugador.
- obtenerFichas(): - Obtiene las fichas del jugador.
- getTerritorios(): - Obtiene los territorios del jugador.
- obtenerNombreJugador(): - Obtiene el nombre del jugador.
- agregarCarta(Carta carta): - Agrega una carta al jugador.
- agregarFicha(Ficha ficha): - Agrega una ficha al jugador.
- setTerritorio(Territorio* nuevoTerritorio): - Establece un territorio para el jugador.
- restarUnidades(int cantidadEliminar, std::string territorio): - Resta unidades a un territorio.
- eliminarTerritorio(Territorio* territorio): - Elimina un territorio del jugador.
- contarTerritorios(): - Cuenta los territorios que tiene el jugador.
-

3. Carta

- TAD Carta
  - Conjunto mínimo de datos
    - TipoCarta, cadena de caracteres, indica el tipo de carta que es (normal, comodín, misión)
    - Territorio, cadena de caracteres territorio que muestra (cualquiera de los 42 disponibles), nulo si es una carta misión o comodín
    - Ficha, cadena de caracteres, indica el tipo de ficha que muestra (infantería, caballería, artillería), nulo si es una carta misión o comodín
    - Misión, cadena de caracteres, indica la misión inscrita en la carta
  - Comportamiento (operaciones) del objeto
    - Carta(std::string tipoCarta, std::string territorio): Constructor de la clase Carta.
    - obtenerTipoCarta(): Obtiene el tipo de carta.
    - establecerTipoCarta(std::string nuevoTipo): Establece el tipo de carta.
    - obtenerTerritorio(): Obtiene el territorio asociado a la carta.
    - establecerTerritorio(std::string nuevoTerritorio): Establece el territorio asociado a la carta.
    - obtenerFicha(): Obtiene el tipo de ficha asociada a la carta.
    - establecerFicha(std::string nuevaFicha): Establece el tipo de ficha asociada a la carta.

    - obtenerMision(): Obtiene la misión asociada a la carta.
    - establecerMision(std::string nuevaMision): Establece la misión asociada a la carta.
    -

4. Continente

- TAD Continente
    - Conjunto mínimo de datos
        - Territorios, lista de territorios, almacena los territorios que se encuentran en determinado continente
        - Nombre del continente, cadena de cárteres que tiene el nombre del continente.
        - TerritoriosOcupados, int, retorna cantidad de territorios ocupados
    - Comportamiento (operaciones) del objeto
        - ocuparTerritorio(ficha, nTerritorio) recibe una Ficha de un jugador y la guarda en el territorio número nTerritorio.
        - AddTerritorio(nombre), crea un territorio dentro de un continente con relación al nombre que recibe.
        - MoverFicha(jugador, territorio1, territorio2), mueve una ficha del territorio1 al territorio vecino territorio2 de un respectivo jugador.
        - ObtenerNombre(), retorna el nombre del continente
        - InicializarTerritorio(nombre), adicionar al continente el territorio escogido por el usuario
        - Reclamado(indice), retorna el estado de reclamado de un territorio
        - getFichasEnTerritorio(indice), retorna cantidad de fichas en un territorio
        - buscarTerritorio(nombreTerritorio), busca territorio dentro del continente y lo retorna

5. Territorio

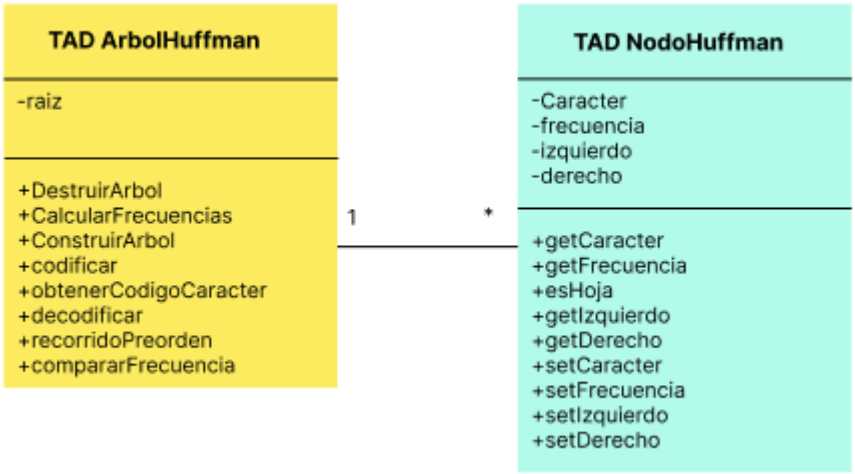
- TAD Territorio
    - Conjunto mínimo de datos
        - Fichas, lista de fichas que se encuentran dentro del territorio
        - TerritoriosColindandtes, vector de punteros, contiene los territorioscolindantes
        - Nombre, string, nombre del territorio
        - Reclamado, string, retorna quien es el propietario del territorio
        - Unidades, int, cantidad de tropas dentro del territorio
        - Id, int, identificador unico del territorio
        - NombreContienente, retorna el nombre del continente al cual pertenece
        - TerritoriosColindantes, vector de territorios, retorna territorios vecinos
    - Comportamiento (operaciones) del objeto
        - Territorio(std::string nuevoNombre): Constructor de la clase Territorio.
        -
        - addFicha(Ficha ficha): Agrega una ficha al territorio.

- deleteFicha(std::string jugador): Elimina una ficha del territorio asociada a un jugador.
- ChekFicha(std::string jugador): Verifica si el territorio tiene una ficha asociada a un jugador específico.
- ContarFichas(std::string Colorjugador): Cuenta la cantidad de fichas de un jugador en el territorio.
- GetQFichas(): Obtiene la cantidad total de fichas en el territorio.
- getNombre(): Obtiene el nombre del territorio.
- getReclamado(): Obtiene el jugador que reclama el territorio.
- esColindante(Territorio* otroTerritorio): Verifica si un territorio es colindante con otro territorio.
- obtenerFicha(std::string jugador): Obtiene una ficha asociada a un jugador específico en el territorio.
-
- getTerritoriosColindantes(): Obtiene una lista de territorios colindantes.
- setTerritoriosColindantes(Territorio* Territorio): Establece un territorio como colindante con el territorio actual.
- restarFichas(int cantidadEliminar): Resta fichas del territorio.
- setReclamar(std::string jugador): Establece el jugador que reclama el territorio.
-

6. Ficha

- TAD Ficha
    - Conjunto mínimo de datos
        - Color, cadena de caracteres que indica el color asignado al jugador que pertenece.
        - Figura, cadena de caracteres que hace referencia al tipo de ficha que es (infantería, caballería, artillería).
    - Comportamiento (operaciones) del objeto
        - GetColor(), retorna el color de la ficha (saber a cual jugador corresponde)
        - SetColor(color), recibe el color y este es asígnado a la ficha.

Diagrama de TADS:



Como podemos observar, la TAD principal, Risk, tiene relacion una a muchas con TADs Jugador, Carta, Continente, las cuales en si se relacionan con: Jugador – una a muchas con Carta y Ficha, y una a muchas con territorio; Continente – una a muchas con Territorio, el cual tiene una conexión con Ficha. Hay que tener en cuenta los limites que pone el enunciado al momento de definicion de dichas TADs.

De igual forma tenemos los TADs del arbol Huffman, el cual se constituye de 2 estructuras: el TAD ArbolHuffman y NodoHuffman. El nodo Huffman contiene los datos de cada nodo, tales como el carácter relacionado, el hijo izquierdo y derecho y la frecuencia del carácter. Los metodos principales se realizan en el TAD ArbolHuffman, el cual tiene como unico parametro la raiz de tipo NodoHuffman.

Procedimiento principal

El procedimiento principal consta de las siguientes suboperaciones (diagrama – mas abajo):

CrearContinente

Entradas: void

Salidas: void

Condiciones: crear nuevo Continente

IngresarComando

Entradas: void

Salidas: string cadena

Condiciones: no es null

Comandos del menu:

1) Inicializar con archivo

leerArchivo:

Entradas: string nombreArchivo

Salidas: void

Condiciones: nombreArchivo existe

2) Inicializar

InicializarJuego:

Entradas: Risk risk

Salidas: void

Condiciones: cantidad jugadores mayor o igual a 3 y hasta 6, continente exista, territorio exista en continente

Fortificar:

Entradas: Risk risk, bool init

Salidas: void

Condiciones: continente y territorio deben existir, fichas que el jugador desplaza deben ser menores a la cantidad total que tiene

3) Turno id_jugador

SeperarEspacio:

Entradas: string cadena, bool parametro

Salidas: string comando

Condiciones: subcadena.length > 0, subcadena dividida por space

4) Salir

5) Guardar nombre_archivo

CrearArchivo:

Entradas: string nombreArchivo

Salidas: void

Condiciones: creación de manera exitosa

6) Guardar_comprimido nombre_archivo

CrearArchivo:

Entradas: string nombreArchivo

Salidas: void

Condiciones: creación de manera exitosa

7) Help

8) Help comando

MostrarAyudaComando:

Entradas: string comando

Salidas: void

Condiciones: comando exista

Comandos y metodos

Risk

1) LanzarDado:

Descripcion - Lanzar un dado y retornar el número aleatorio de 1 a 6

Entradas - void

Salidas - int random

Condiciones - lanzar numero entero de 1 - 6

2) IniciarPartida:

Descripcion - pone la bandera de partida en verdadero

Entradas - void

Salidas - void

Condiciones - n/a

3) EstadoPartida:

Descripcion - retorna estado de la partida

Entradas - void

Salidas - bool

Condiciones - n/a

4) AsignarGanador:

Descripcion - asigna ganador y acaba partida

Entradas - void

Salidas - void

Condiciones - el jugador que gana tiene que existir

5) CrearContinente:

Descripcion - funcion para creación de continentes

Entradas - void

Salidas - void

Condiciones - utilizacion de continentes que existan el la vida real.

6) CrearTarjetas:

Descripcion - crear una carta de juego con los detalles proporcionados

Entradas - string tipo, string territorio, string ficha, string misión

Salidas - void

Condiciones - n/a

7) AgregarTropas:

Descripcion - agregar tropas a un jugador determinado al finalizar la ronda

Entradas - Jugador jugador

Salidas - n/a

Condiciones - el jugador debe existir

8) TurnoJugador:

Descripcion - aumenta la cantiad de turnos realizados, para poder avanzar al siguiente turno. Esta funcion realiza el avanzar en los jugadores mostrado el nombre del jugador.

Entradas - void

Salidas - void

Condiciones - n/a

9) InfoContinente:

Descripcion - retorna el nombre de un continente disponible

Entradas - ind indice

Salidas - string nombreContinente

Condiciones - n/a

10) InfoTerritorios:

Descripcion - guarda en una variable los territorios de un continente

Entradas - string nameContinente

Salidas - string retorno

Condiciones - ciclo de 6 iteracciones, el continente debe tener territorios

11) EstadoTerritorios:

Descripcion - retorna si el territorio el nombre del jugador quien ocupo el territorio o null si esta libre

Entradas - string nameContinente, string nameTerritorio

Salidas - bool estado

Condiciones - territorio y continente deben existir

12) CrearJugador:

Descripcion - recibo el nombre de un jugador y lo guardo en el vector de jugadores

Entradas - string nombre, int qJugadores

Salidas - void

Condiciones - n/a

13) ColorJugador:

Descripcion - retorna el color que se le debe asignar a un jugador

Entradas - void

Salidas - string color

Condiciones - n/a

14) QUnidades:

Descripcion - identifica las unidad ed de batallón inicial

Entradas - int qJugadores

Salidas - int unidad

Condiciones - n/a

15) InfoJug:

Descripcion - función de depuración

Entradas - void

Salidas - string

Condiciones - n/a

16) GetNameJugadorEnTurno:

Descripcion - obtener el nombre del jugador que tiene turno actual

Entradas - void

Salidas - string name

Condiciones - n/a

17) GetColorJugadorEnTurno:

Descripcion - obtener el color del jugador que tiene turno actual

Entradas - void

Salidas - string color

Condiciones - n/a

18) MoverFichasJugador:

Descripcion - mover ficha de un jugador a un territorio de un continente, cambiando el color de las tropas dependiendo del territorio

Entradas - int qFichas, string continente, string territorio

Salidas - bool

Condiciones - cantidad fichas mayor a 0, tiene que ser una cantidad de fichas que no exceda su cantidad de fichas actual del jugador actual.

19) IndiceContinente:

Descripcion - busca en el vector de continentes el indice de un continente

Entradas - string continente

Salidas - int indice

Condiciones - continente exista

20) IndiceTerritorio:

Descripcion - busca en el vector de territorio todos los indices

Entradas - int iContinente, string territorio

Salidas - int indice

Condiciones - indice de continente sea valido, territorio exista en el continente de entrada

21) TerritoriosJugador:

Descripcion - muestra los territorios del jugador actual en un continente

Entradas - string nameContinente

Salidas - string retorno

Condiciones - continente con el nombre nameContinente exista

22) CrearBatallon:

Descripcion – creacion del batallon de los jugadores

Entradas – int numero jugadores

Salidas - void

Condiciones - null

23) TerritoriosLibres:

Descripcion - indica si todos los territorios de todos los continentes han sido ocupados

Entradas - void

Salidas - bool

Condiciones – cantidad de ocupados mayor a 0

24) GetFichasJugadorEnTurno:

Descripcion - retorna la cantidad de fichas que tiene el jugador en turno

Entradas - void

Salidas - int

Condiciones - null

25) BuscarContinenteTerritorio:

Descripcion - busca y regresa el nombre del contiente del territorio que ingresó el usuario

Entradas – string territorio

Salidas – string continente

Condiciones – existan continentes y territorios

26) TerritorioJugador:

Descripcion - revisa que el territorio ingresado corresponda al jugador que se encuentra en turno

Entradas - string continente, string territorio

Salidas – bool status

Condiciones – existan continentes y territorios

27) EstadoGanador:

Descripcion – retorna el estado si se encontro el ganador

Entradas - void

Salidas – bool statusWinner

Condiciones - null

28) EsTurnoJugador:

Descripcion - retorna true, cuando el nombre ingresado es el mismo del jugador actual

Entradas – string nombreJug

Salidas – bool status

Condiciones – turnoActual > 0

30) TurnosEnCero:

Descripcion - pone el contador de turnos en cero para volver a empezar desde el jugador 0

Entradas - void

Salidas - void

Condiciones - null

2) Jugador

1) ObtenerColor:

Descripcion - obtencion del color de un jugador

Entradas - void

Salidas - string color

Condiciones - n/a

2) ObtenerNombreJugador:

Descripcion - obtencion del nombre de un jugador

Entradas - void

Salidas - string name

Condiciones - n/a

3) AgregarCarta:

Descripcion - agregar a las cartas que tiene el jugador una mas

Entradas - Carta carta

Salidas - void

Condiciones - n/a

4) AgregarFicha:

Descripcion - agregar una ficha mas al arreglo de fichas del jugador

Entradas - Ficha ficha

Salidas - void

Condiciones - n/a

5) ObtenerFichas:

Descripcion - retorna un vector de fichas

Entradas - void

Salidas - vector<Ficha> fichas

Condiciones - n/a

6) ObtenerTotalFichas:

Descripcion - obtener longitud del arreglo de fichas

Entradas - void

Salidas - int q

Condiciones - n/a

7) MoverFicha:

Descripcion - mover las fichas, retornando la primera

Entradas - void

Salidas - Ficha aux

Condiciones - existan fichas, fichas.length > 0.

3) Carta

1) getTerritorio:

Descripcion - obtener territorio de la carta

Entradas - void

Salidas - Territorio territorio

Condiciones - n/a

2) setTerritorio:

Descripcion - modificar territorios de la carta

Entradas - Territorio terr

Salidas - void

Condiciones - n/a

4) Continente

1) ObtenerNombre:

Descripcion - obtener nombre del continente

Entradas - void

Salidas - string name

Condiciones - n/a

2) setNombre:

Descripcion - modificar nombres de continentes

Entradas - string name

Salidas - void

Condiciones - n/a

3) OcuparTerritorio:

Descripcion - modificar estados relacionados con la ocupación de un territorio

Entradas - Ficha ficha, int nTerritorio, string color

Salidas - void

Condiciones - color sea igual a color usuario quien la atrapo, territorio con index nTerritorio debe existir

5) getNombreTerritorio:

Descripcion - obtener nombre del territorio

Entradas - int indice

Salidas - string name

Condiciones - territorio con indice exista

6) TerritorioValido:

Descripcion - buscar el nombre de un territorio en un continente y si existe, retorna true

Entradas - string territorio

Salidas - boolean status

Condiciones - n/a

7) AddTerritorio:

Descripcion - agrega territorio en un continente

Entradas - string territorio

Salidas - void

Condiciones - n/a

8) colorTerritorio:

Descripcion - obtener color de un territorio

Entradas - int terri

Salidas - string color

Condiciones - territorio con terri exista

9) GetFichasEnTerritorio:

Descripcion – obtener las fichas en un territorio determinado

Entradas – int indice

Salidas – int q

Condiciones – territorio[indice] exista

10) Reclamado:

Descripcion – obtener territorio reclamado

Entradas – int indice

Salidas - string

Condiciones - null

11) getTerritoriosOcupados:

Descripcion – retorna territorios con status ocupado

Entradas - void

Salidas – int index

Condiciones - null

5) Territorio:

1) AddFicha:

Descripcion - adicionar ficha

Entradas - Ficha ficha

Salidas - void

Condiciones - n/a

2) DeleteFicha:

Descripcion - eliminar ficha de la list de fichas de un jugador

Entradas - string jugador

Salidas - void

Condiciones - ficha exista en el territorio

3) GetFicha:

Descripcion - obtener la ficha final de un jugador

Entradas - string jugador

Salidas - Ficha ficha

Condiciones - jugador tiene que tener fichas en el territorio

4) CheckFicha:

Descripcion - verificar si existe la ficha del jugador

Entradas - string jugador

Salidas - bool status

Condiciones - ficha exista

5) SetReclamado:

Descripcion - cambiar estado del territorio como reclamado por un jugador en particular

Entradas - string jugador

Salidas - void

Condiciones - n/a

6) ContarFichas:

Descripcion - contar fichas de un jugador

Entradas - string jugador

Salidas - int q

Condiciones - n/a

7) GetQFichas:

Descripcion - obtener cantidad de fichas en un territorio

Entradas - void

Salidas - int cantidad

Condiciones - territorio exista

8) GetNombre:

Descripcion - obtener nombre del territorio

Entradas - void

Salidas - string nombre

Condiciones - n/a

9) AgregarVecino:

Descripcion - funcion para agregar un territorio vecino

Entradas - Territorio vecino

Salidas - void

Condiciones - territorio exista

6) Ficha

1) ObtenerColor:

Descripcion - obtener color de la ficha

Entradas - void

Salidas - string color

Condiciones - n/a

2) SetColor:

Descripcion - modificar color de la ficha

Entradas - string color

Salidas - void

Condiciones - n/a

3) ObtenerFigura:

Descripcion - obtener figura de la ficha

Entradas - void

Salidas - string figura

Condiciones - n/a

4) SetFigura:

Descripcion - modificar figura de la ficha

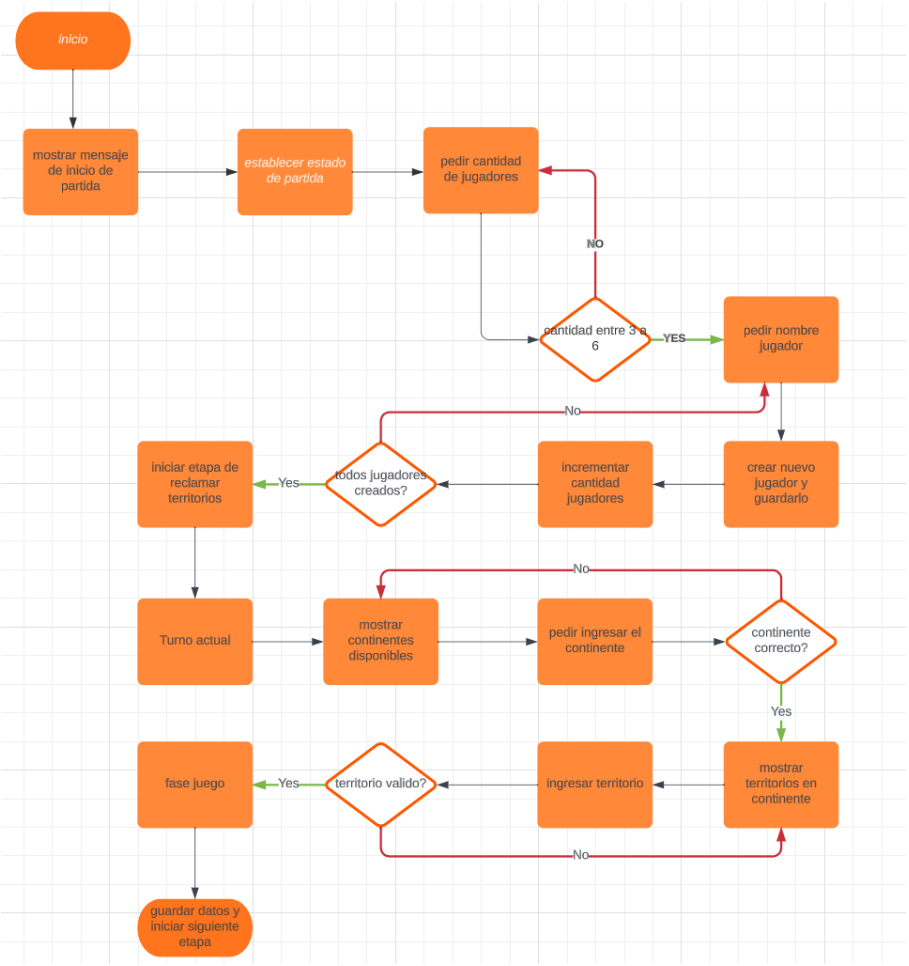
Entradas - string figura

Salidas - void

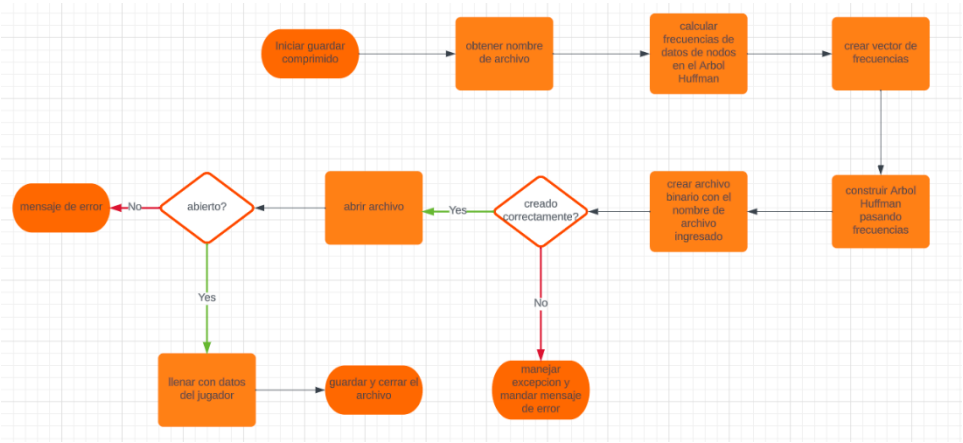
Condiciones - n/a

DIAGRAMAS DE FLUJOS

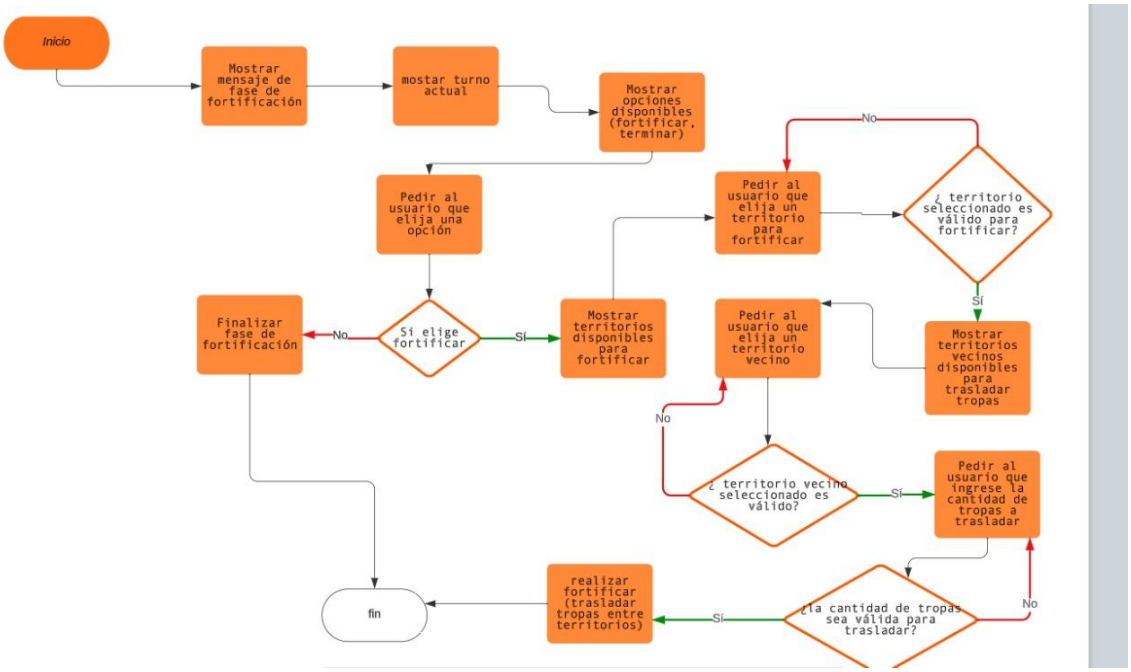
Inicializar



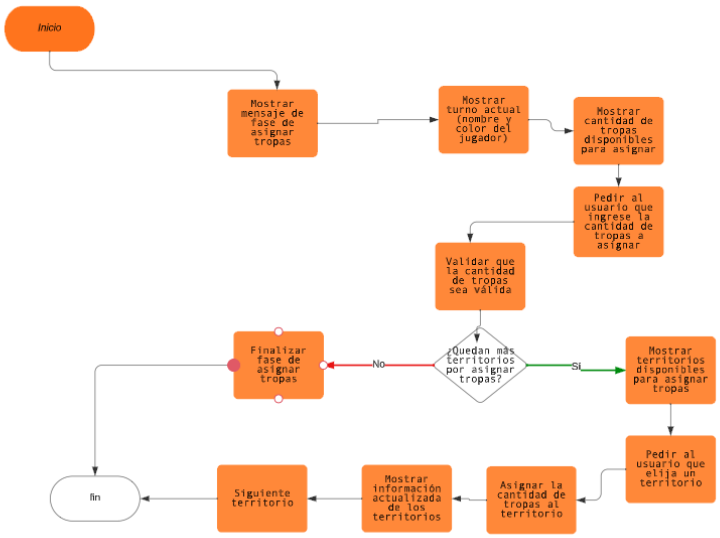
Guardar comprimido



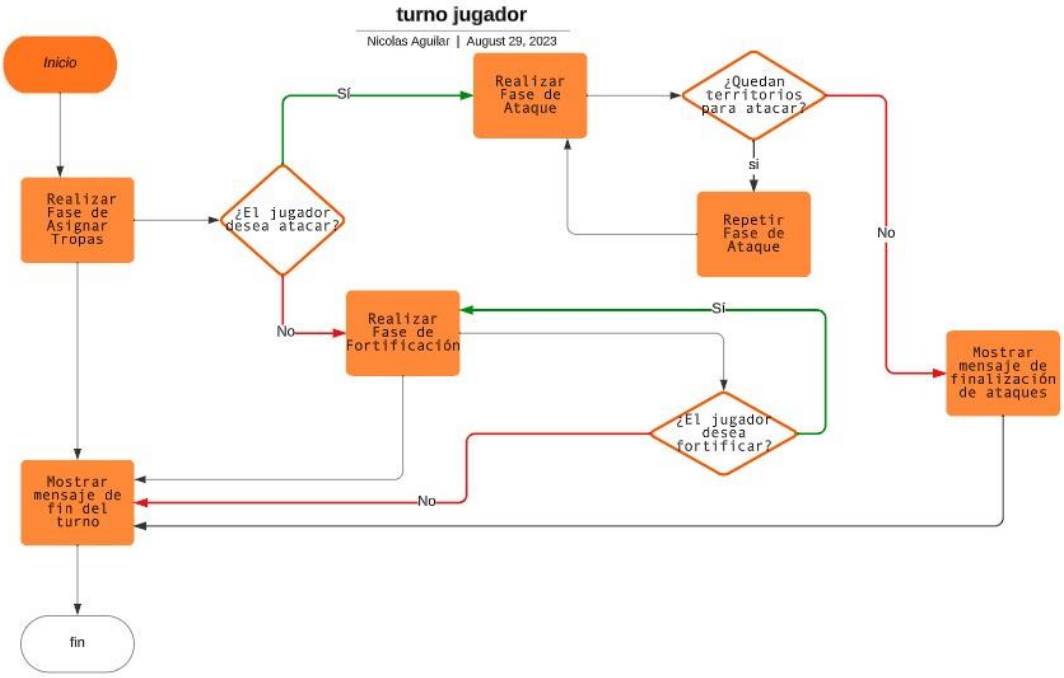
Fortificar



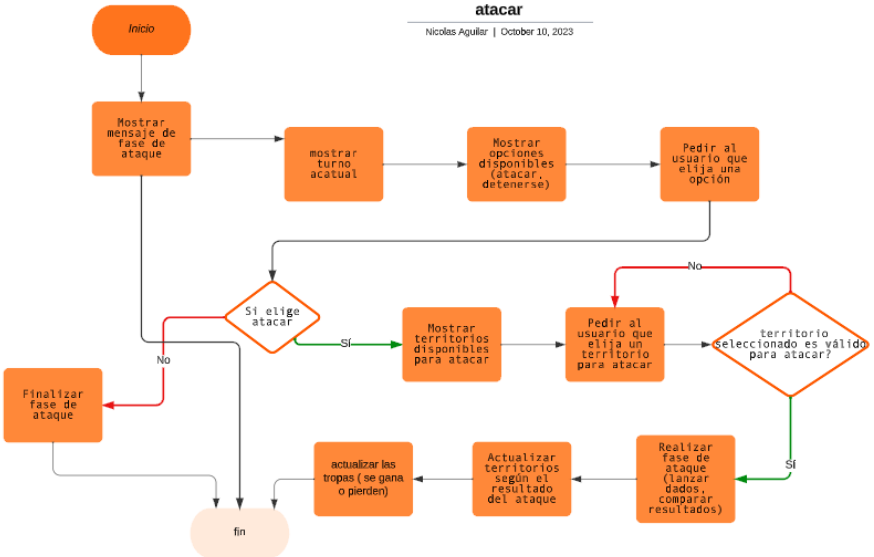
Asignar Tropas – Obtener Unidades



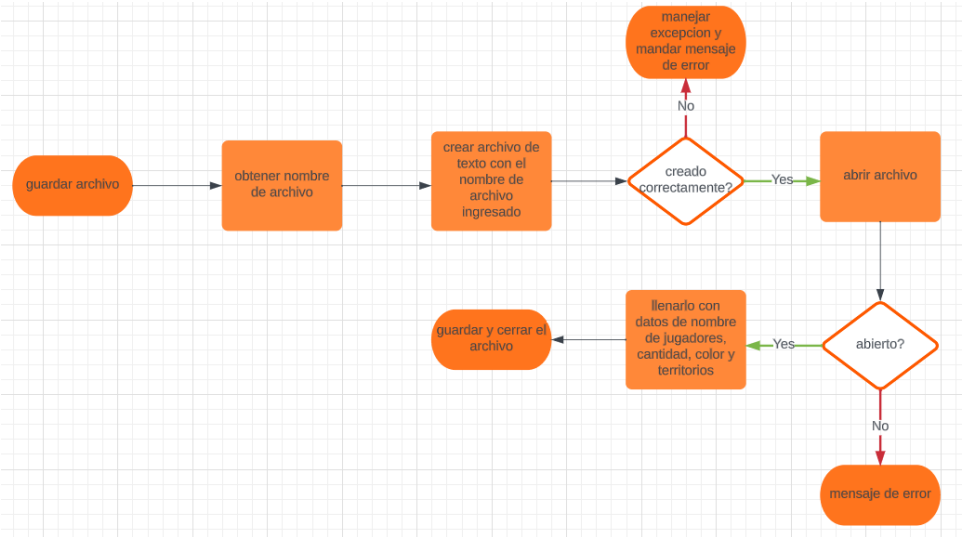
Turno jugador



Atacar



Guardar



PLAN DE PRUEBAS (INICIALIZAR)

Objetivo del plan de pruebas

Verificar el funcionamiento correcto de cada uno de los procesos esenciales en el programa, revisar posibles fallos o problemas algorítmicos y validar cada uno de estos procesos.

Identificación de requisitos

El programa debe cumplir con unos requisitos esenciales, dentro de los cuales están :

- Permitir jugar con una cantidad de jugadores entre 3 y 6.
- Permitir el acceso a cualquier comando que requiera el usuario según el menu de ayuda.
- Obtener la visualización de cada uno de los continentes y territorios del juego.
- Entre otros

Casos de prueba

- **Inicializar**

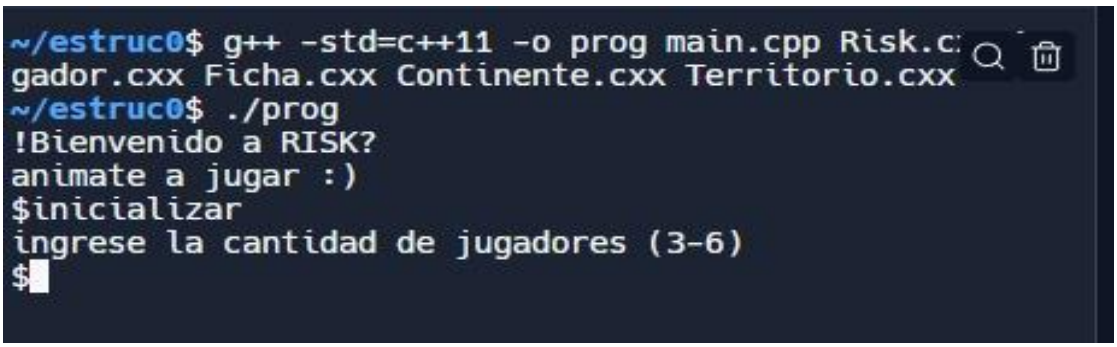
Resultado Esperado :

Al ejecutar el comando $inicializar en el programa, este deberá solicitarnos la cantidad de jugadores que habrá en la partida a crear, después de esto nos deberá permitir ingresar el nombre de cada jugador según la cantidad de jugadores y acto seguido se procederá con el reclamo de territorios de parte de cada jugador.

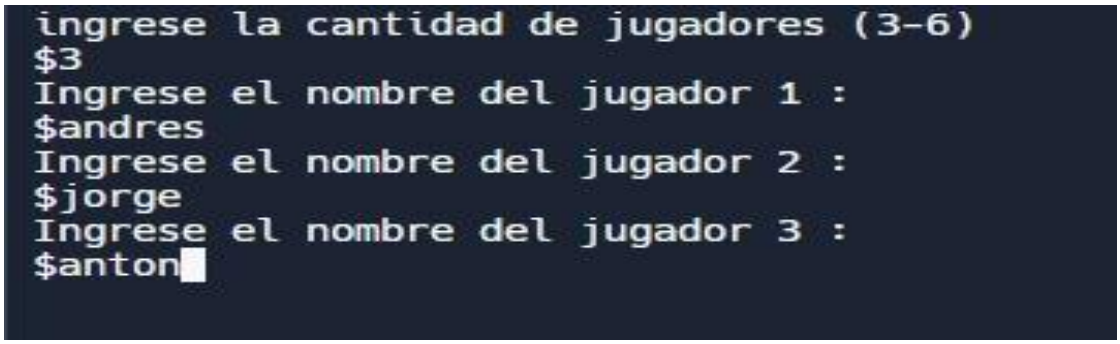
El proceso se hará de a turnos siguiendo el orden de ingreso de los nombres para cada turno, (por lo cual el primer jugador que ingreso su nombre será el primero en el orden del turno y así sucesivamente) en cada turno cada jugador podrá poner una ficha en el territorio de algún continente que este desee sin repetir territorios hasta que todo el mapa este completo.

Resultado Obtenido:

Como se puede evidenciar, el resultado del inicio fue igual al resultado esperado pidiendo observar cómo nos pide que ingresemos la cantidad de jugadores.



Acto seguido podemos ver cómo nos solicita los nombres de los jugadores según la cantidad de jugadores que se ingresó por lo cual este proceso también fue exitoso.



En la siguiente parte donde inicia el proceso de reclamar territorio, nuevamente el resultado obtenido es idéntico al resultado esperado, pudiendo observar cómo en un determinado turno el jugador de nombre “Jorge” puede elegir el continente disponible y acto seguido puede ver los territorios disponibles dentro de este continente.

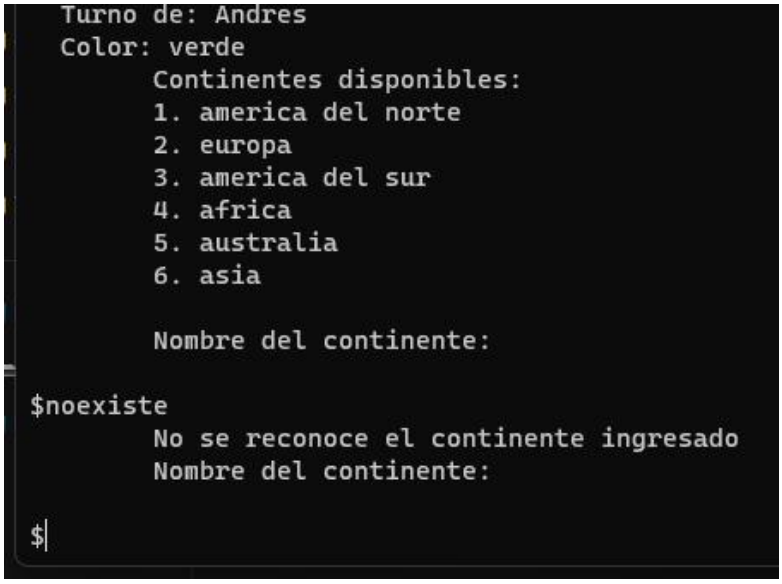
Posteriormente se evidencia que en las fichas del jugador Jorge se disminuyó una de sus fichas que ya fue colocada en ese territorio y después se pasa al turno del siguiente jugador que es “Anton” y así continuara el proceso hasta que se llene el mapa.

Casos de Error

La función de inicializar tiene implementada restricciones para casos no deseados como errores o malos ingresos de datos de parte del usuario.

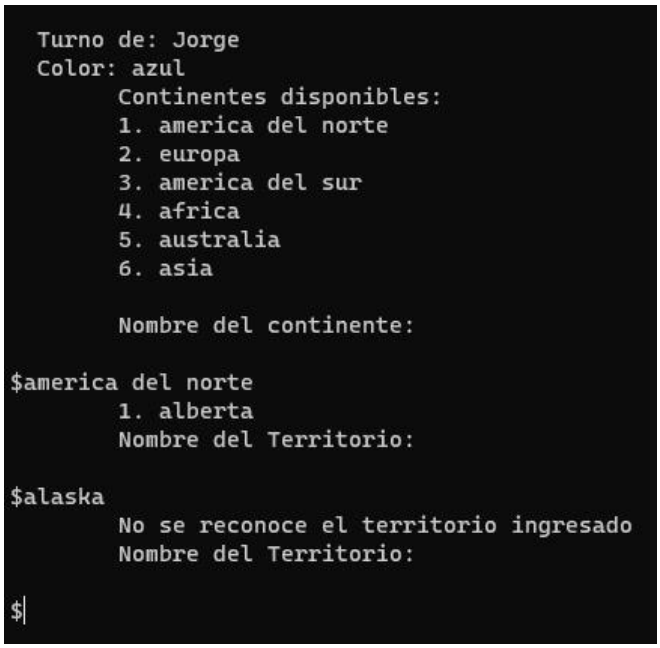
Algunas de estas restricciones son:

- Continente Invalido



Como se puede observar si el usuario digita un continente que no existe o algún otro tipo de carácter invalido, se le mostrara un mensaje indicando que el continente ingresado es invalido y le pedirá que ingrese nuevamente el continente deseado, por lo cual esta restricción se ha implementado de manera correcta y arroja el resultado esperado.

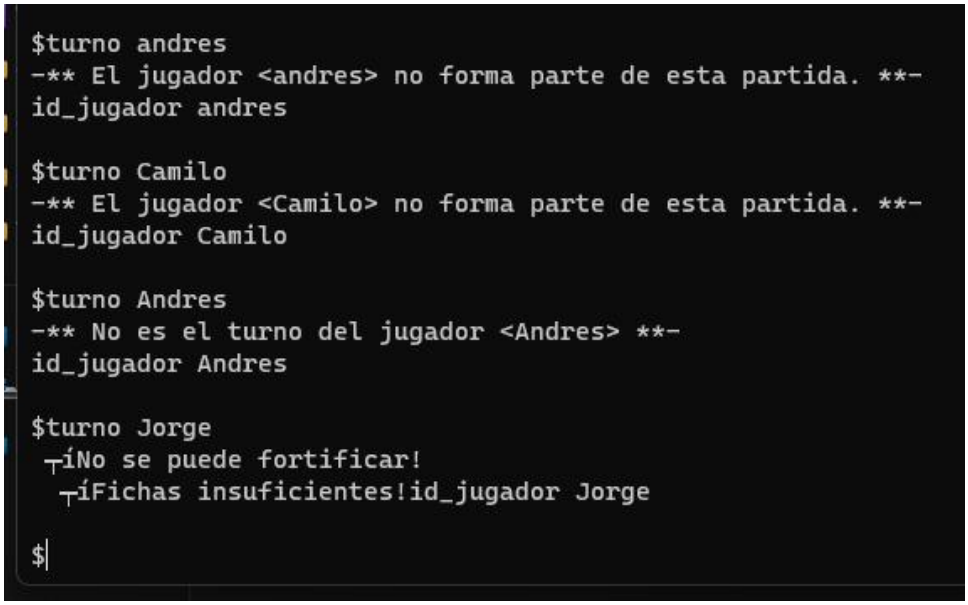
- Territorio Invalido



Al igual que pasa con el continente, si el usuario ingresa un territorio no existente, o que ya está ocupado por otro jugador o el mismo, o que es un carácter invalido, se le imprimirá un error diciendo que el territorio es invalido y se le solicitara que vuelva a digitar el territorio deseado.

Final Prueba de Inicializar

Turno invalido



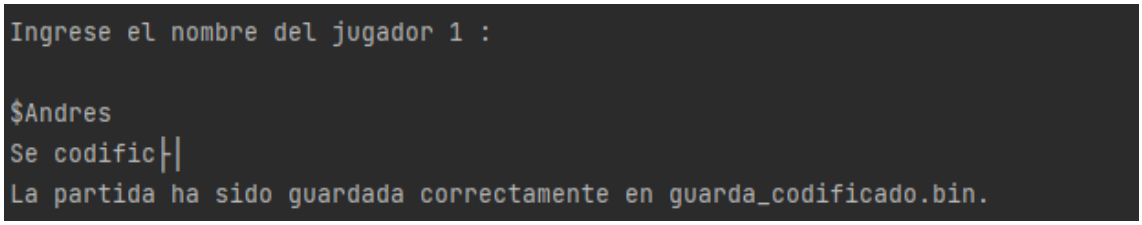
- **Guardar_Comprimido**

Resultado Esperado: La partida jugada se guarda correctamente en un archivo binario con la información de la cantidad de jugadores, y para cada jugador debe guardar: su nombre, su color en el juego, la cantidad de territorios que tiene(países) con el identificador de cada país y la cantidad de unidades del ejército de cada país, también debe guardar la cantidad de tarjetas que tiene ese jugador, y el identificador de cada una de esas tarjetas. Si el juego se guardo correctamente mostrara en pantalla" Comando Correcto"

Además si el juego no ha salido inicializado con anterioridad debe arrojar un error que indique que el juego no ha sido inicializado, o si por alguna razón el comando no funciona también debe arrojar un error.

Resultado Obtenido:

El resultado obtenido fue exitoso, es igual al esperado, a continuación, mostraremos como se refleja en pantalla.



```
       Nombre del continente:
Se codific├|
La partida ha sido guardada correctamente en guarda_codificado.bin.
Se codific├|
La partida ha sido guardada correctamente en guarda_codificado.bin.
Se codific├|
La partida ha sido guardada correctamente en guarda_codificado.bin.
Se codific├|
```

Como se puede observar al ejecutar el comando se guarda el archivo correctamente y muestra su mensaje en pantalla.

Ahora se puede observar que cada vez que se ejecuta algún comando para guardar algo, (por ejemplo el nombre de los jugadores , o los territorios que elige) se puede observar en el archivo que se crea, se guarda toda esta información en modo binario para el árbol de Huffman.

```
1001111100111
1101100110011100110
1
0001100011110
1100100

1001111100111
1101100110011100110
1
0001100011110
1100100
1101
```

Si queremos comprobar que se guardó el archivo con toda la información correctamente, podemos volver a cargar la partida y revisar que toda la información sea exactamente la misma. Como se puede ver a continuación.

////////////////////// imagen con cargado de la partida para revisar datos

Casos de error

- Juego no inicializado

Si un jugador intenta guardar un juego que ni siquiera ha sido inicializado se mostrara en pantalla un error que indique que no se ha inicializado el juego, tal como se muestra a continuación.

/// IMAGEN JUEGO NO INICIALIZADO