Trabajo Práctico 2: Invasión Galáctica V1.0

Materia: CB100 - Algoritmos y Estructuras de Datos

Integrantes: Alaniz Patricio Javier, Ramos Ignacio Joaquín, García Martín Ariel, Avila Natalia Silvana, Ajnota Alarcon Mayra Belen

Grupo: 6 en órbita

Cuatrimestre: 1C-2025

Cuestionario

1) ¿Qué es un svn?
El software Subversion, también llamado SVN, es un sistema de control de versiones de código abierto. Subversion (SVN) permite a los equipos consultar versiones anteriores de archivos y realizar un seguimiento de sus cambios a lo largo del tiempo.

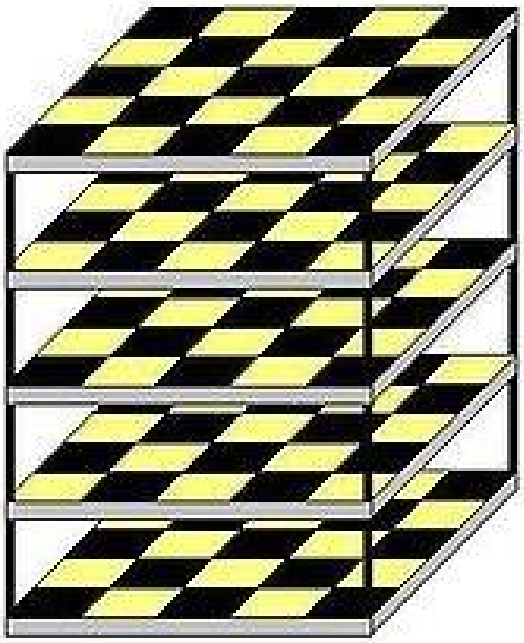
2) ¿Que es una “Ruta absoluta” o una “Ruta relativa”?
Las rutas absolutas son las que comienzan desde el directorio raíz (/), proporcionando una ubicación completa en el sistema.
Las rutas relativas comienzan desde el directorio de trabajo actual en lugar del directorio raíz. Utilizan símbolos especiales para representar posiciones relativas en el sistema de archivos. El símbolo “.” representa el directorio de trabajo actual, mientras que “..” representa el directorio padre o el directorio anterior.

3) ¿Qué es git?
Git es un software que nos permite controlar las versiones de los archivos de nuestro proyecto. Siendo una versión el estado del archivo después de realizar un conjunto de cambios. Incluso nos permite trabajar con control de versiones de manera local en nuestra computadora, sin necesidad de estar conectado a un servidor remoto.

Esto significa que podemos mantener un registro detallado y organizado de todo lo que ocurre en nuestros archivos, asegurando que cada cambio esté documentado y pueda ser recuperado en cualquier momento.

Informe

El tablero tiene esta forma



El juego tiene como máximo 6 jugadores y como mínimo 2. El tablero es de NxNxN donde N puede ir desde 4 hasta 10 inclusive.

Las coordenadas que se piden son de X, Y y Z:
X: cuántas posiciones se mueve a la derecha

Y: cuántas posiciones se mueve hacia arriba
Z: en qué “nivel” del tablero está

Una vez comenzado el juego, se puede cargar una partida o empezar una desde cero.
Los jugadores tienen que decidir dónde situar sus bases una vez se registren todos. Al comienzo del juego cada jugador tiene una base y 0 naves.
El turno de jugar de cada jugador es equivalente al orden en el que se registraron. Siempre se tienen 8 opciones pero no todas se pueden ejecutar, varía según el estado del juego.

Las opciones que tiene cada jugador son:

1-Agregar nave: agrega una nave en las coordenadas pasadas siempre y cuando no se tenga una nave propia, no haya radiación activa, no se encuentre una nave/base enemiga y no haya un satélite propio o enemigo.

2- Mover nave: lo mismo que Agregar nave pero si se encuentra una base enemiga y esta está debilitada se la conquista.

3- Agregar satélites: Agrega un satélite en las coordenadas pasadas siempre y cuando no se tenga un

satélite propio. Si se encuentra un satélite enemigo se lo destruye pero no se agrega ningún satélite propio.

4- Atacar sector: Se puede atacar un sector siempre y cuando el jugador tenga naves con que hacerlo. El sector a atacar no puede ser un sector con radiación, nave/base propias ni satélites propios.

5- Robar carta: La carta robada es aleatoria y solo se puede robar una por turno.

6- Usar carta: Se devuelve la primera carta robada siempre y cuando haya cartas en el mazo.

7- Alianza: Crea una alianza entre dos jugadores. Los aliados comparten la posición de sus naves y satélites. Se puede generar una alianza siempre y cuando existan más de 2 jugadores.

8- Dejar de jugar: Finaliza el juego con la opción de guardar la partida en un archivo .txt.

Las cartas que pueden contener el mazo de los jugadores son:
1. Campo de fuerza: Aumenta la defensa de una base durante un número de turnos.

2. Rastreador cuántico: Revela si hay naves enemigas en un radio de L (valor random) sectores desde la posición elegida.
3. Doble salto hiperespacial: Permite mover una nave dos veces en el mismo turno.
4. Base adicional: permite agregar una base adicional.
5. Sumar vida a base: se le agrega vida extra a la base seleccionada a parte del escudo que ya contiene .
6. Daño extra de nave: se le agrega un daño adicional que hace al atacar a parte del que ya tenía (Daño inicial de la nave: 1).
7. Nave hace daño en área: la nave seleccionada hace daño en área que rodea las coordenadas pasadas.
8. Nave obtiene escudo: la nave seleccionada a parte de la vida que tiene por defecto (vida inicial: 1) se le agrega un escudo (valor del escudo adicional: 3).

El juego se guarda en un archivo txt. Para poder continuar una partida guardada se necesita la ruta relativa del archivo txt.

Se usa para representar a las bases, naves, satélites y radiación:

Base enemiga

Base del jugador

Nave del jugador

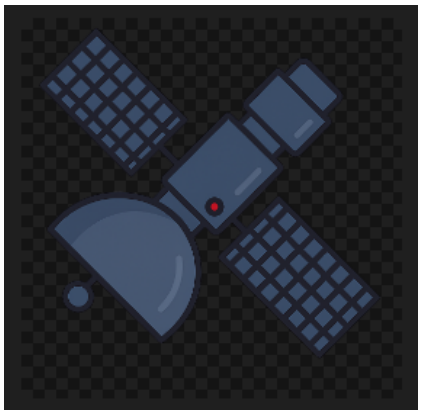
Nave enemiga

Nave aliada

Radiación

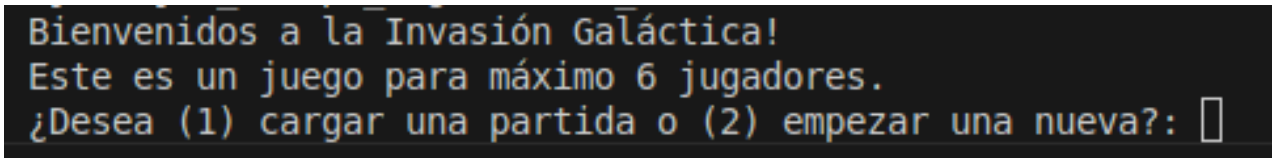
Satelite del jugador

Satélite enemigo



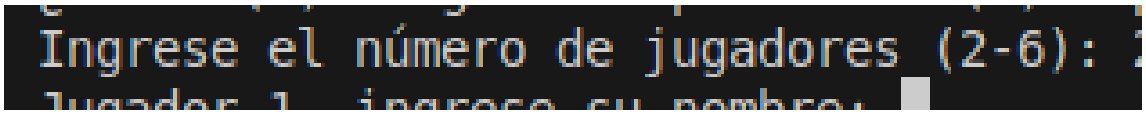
Satélite aliado

Manual del usuario



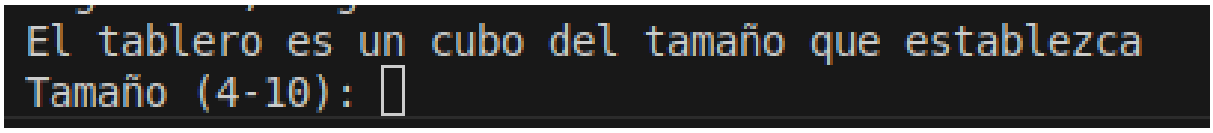
No se puede continuar si la respuesta NO es 1 o 2. Si se elige la opción 1 se debe tener la ruta relativa del archivo .txt en el que se debió guardar la partida.

Si se elige la opción 2:



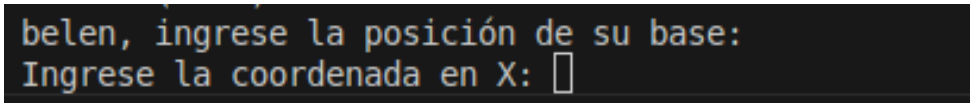
La cantidad de jugadores no puede ser menor a 2 ni mayor a 6 de lo contrario no se puede continuar. Una vez que se valide que el número de jugadores es válido se pide los nombres de cada uno.

Una vez se guarden los nombres se pide el tamaño del cubo.

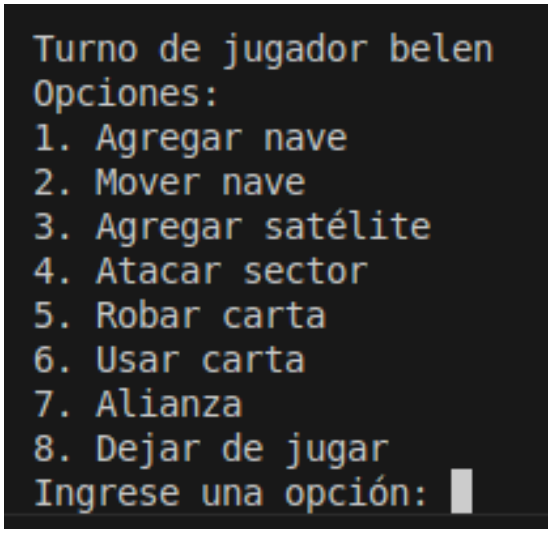


No puede ser menor a 4 ni mayor a 10. El cubo resultante es de NxNxN por ejemplo si se elige 4, el cubo es de 4x4x4.

Luego de ser creado el cubo que se va a usar durante el juego, que en principio no contiene nada, se piden las coordenadas de la base que va a tener cada jugador.



Una vez se agreguen las bases de cada uno de los jugadores, comienza el juego



Reglas:

1-No se pueden realizar acciones en sectores con radiación activa.
2-No se pueden agregar ni mover naves a sectores ocupados por:

- Otra nave propia.
- Una nave o base enemiga (salvo que la base esté debilitada y sea un movimiento).

- Un satélite propio o enemigo.

3-No se puede atacar:
- Si el jugador no tiene naves.
- Sectores con: naves propias, bases propias, satélites propios y radiación activa

4-No se puede agregar un satélite:
- Si ya hay un satélite propio en ese sector.
- Si hay un satélite enemigo, este se destruye pero no se agrega uno nuevo.

5-No se puede robar más de una carta por turno.

6-No se puede usar una carta si:
- No se robó ninguna carta.
- El mazo está vacío.

7-No se puede formar una alianza si hay solo dos jugadores activos.

8-No se puede repetir turno ni realizar más de una acción por turno.

9-No se pueden colocar bases en posiciones inválidas al inicio del juego.

10-No se puede guardar la partida sin confirmar la opción de finalizar el juego.

11-No se puede cargar una partida sin especificar una ruta válida al archivo .txt.

12-No se pueden realizar acciones fuera del tablero (coordenadas fuera de rango).

13-No se puede conquistar una base enemiga si no está debilitada.

14-No se puede mover una nave si no hay ninguna en juego.

Manual del programador

Sistema de Piezas

El sistema de piezas está diseñado bajo el patrón de herencia, donde Pieza es la clase padre y las otras piezas las hijas

Clase padre: Pieza

La clase Pieza es la clase padre que define la estructura común para todas las piezas del tablero. Contiene los

atributos y métodos fundamentales que comparten todas las piezas del juego.

Clases hijas:

Clase Base

Representa las bases espaciales de los jugadores, que son estructuras defensivas con alta resistencia.

Validaciones Específicas

- Valida que todos los parámetros sean correctos mediante creacionValida()
- Lanza RuntimeException si la creación es inválida

Clase Nave

Representa las unidades móviles de combate que pueden atacar, moverse y ser mejoradas con cartas.

Validaciones Específicas

- El daño inicial debe ser mayor a cero
- Valida la creación mediante la clase padre

Clase Satélite

Representa Satélites espía que pueden detectar movimientos enemigos en un rango específico.

Validaciones Específicas

- El radio de detección debe ser mayor a cero
- Escudo inicial es siempre 0

Clase Radiación

Representa áreas con radiación activa que bloquean el movimiento durante un número determinado de turnos**.**

Características Especiales

- No tiene dueño (duenio = null)
- No se reduce la duración en el primer turno
- Sobrescribe creacionValida() con validaciones especificas

Clase Vacio

Representa sectores vacíos del tablero donde no hay ninguna pieza activa.

Características Especiales

- No tiene dueño (duenio = null)
- Vida y escudo siempre son 0

Otras clases:

Alianza

Crea una alianza válida entre dos jugadores diferentes y con duración positiva. Lanza errores si:

- Alguno de los jugadores es null
- Son el mismo jugador.
- La duración de la alianza es 0 o menor

Jugador

Guarda todo lo que tiene y hace un jugador en el juego. O sea, el jugador no es solo un nombre, sino que tiene un montón de cosas que maneja:

- Bases, naves y satélites:
  La clase usa listas para guardarlas y tiene métodos para agregar nuevas, sacar alguna y también para ver cuáles tiene.

- Cartas:
  El jugador tiene una “cola” de cartas, o sea, las cartas se usan en orden (primero en entrar, primero en salir). Podés agregar cartas y sacar la siguiente carta para usarla.

- Alianzas:
  El jugador puede formar alianzas con otros jugadores. La clase guarda esas alianzas y tiene métodos para agregar o sacar alianzas, saber

quiénes son sus aliados, y si es aliado de alguien en particular.

- Nombre:
  No puede ser vacío o nulo.

- ¿Puede atacar?
  Si tiene al menos una nave, el jugador puede atacar. Esa es la lógica que tiene el método puedeAtacar().

Carta

Cuando creás una carta, tenés que pasarle tres datos (nombre, descripción, tipo). Y acá es donde entra la clase ValidacionesUtils para asegurarse de que no se está ingresando información invalida. No puede ser nulo.

- Tiene que tener al menos 3 caracteres (para nombre y descripción).
- Te saca los espacios de más adelante/atrás (trim()).

Si algo está mal, tira una RuntimeException, o sea, un error que rompe en tiempo de ejecución.

Hay métodos para ver cada atributo:

- getNombre()
- getDescripcion()
- getTipo()

Esto es útil para mostrar info de la carta en el juego o cuando el jugador la saca de la cola de cartas y querés mostrar por consola.

Están los setNombre(), setDescripcion() y setTipo() pero son privados. Una vez creada la carta, no se cambia. Esto está bueno porque asegura que la carta no se modifique después mágicamente en medio del juego.

CartaUtils

crearListaDeCartasBase(int cantidad) te da una lista de cartas listas para usar, con varias copias de cada tipo. Si le pasás un número menor a 1, te tira una excepción porque no tendría sentido crear cero o menos cartas.

Devuelve una ListaSimplementeEnlazada<Carta>, o sea, una lista con las cartas generadas. Usa un bucle for para repetir la creación de 8 cartas distintas, y eso lo repite tantas veces como indiques.

Con esta clase no tenés que escribir a mano cada carta. Solo llamás a este método y tenés un mazo de cartas base completo para repartir o usar en el juego.

Mazo

Usa una pila enlazada (PilaEnlazada<T>) para guardar las cartas porque así el último que entra es el primero que

sale: como si las cartas mezcladas se apilaran boca abajo, y vos sacás la de arriba.

CargadoDePartida

Le pasás un archivo .txt con información guardada y esta clase se encarga de reconstruir todo: el tablero, los jugadores, sus piezas, alianzas, etc.

GuardadoDePartidaEnArchivo

Esta clase es responsable de guardar el estado actual del juego (tablero, jugadores, alianzas y piezas especiales) en un archivo de texto, permitiendo posteriormente su restauración.

Menú

La clase Menu es el controlador principal del Juego Invasión Galáctica se encarga de inicializar, mostrar y gestionar todas las opciones del juego en cada turno de los jugadores.

Funciones Principales:

Inicio del Juego

- Muestra un mensaje inicial y permite cargar al usuario una partida o iniciar una nueva.
- Crea el tablero y los jugadores con sus nombres, validando que no se repitan.

- Posiciona las bases iniciales para cada jugador.

Turnos de juego

- En cada turno, muestra un menú con acciones posibles (como mover naves o usar cartas).
- Cambia de jugador automáticamente después de una acción válida.
- Termina el juego cuando queda un solo jugador.

Acciones disponibles

- AGREGAR_NAVE, AGREGAR_SATELITE: permite al jugador colocar nuevas piezas en el tablero.
- MOVER_NAVE: desplaza una nave dentro del tablero.
- ATACAR_SECTOR: permite atacar una pieza enemiga.
- ROBAR_CARTA y USAR_CARTA: gestiona el uso de cartas del mazo.
- ALIANZA: permite formar alianzas temporales entre jugadores.
- DEJAR_DE_JUGAR: cierra el juego (con posibilidad de guardar).

Lógica del juego

- Verifica que todas las coordenadas y distancias sean válidas.

- Controla que no se ataque a aliados y ni a uno mismo.
- Detecta piezas enemigas usando satélites y cartas.
- Maneja capturas de bases enemigas y eliminación de jugadores sin base.
- Reduce la duración de escudos, alianzas y radiación al final de cada turno.

Guardado y carga

- Guarda la partida en un archivo .txt.
- Carga una partida existente, reconstruyendo tablero, jugadores y alianzas.

Visualización

- Dibuja el tablero en un archivo .bmp en cada turno con los elementos visibles.

Estructuras internas utilizadas

- Usa Jugador [ ] y ListaSimplementeEnlazada<Jugador> para manejar jugadores.
- Utiliza ListaSimplementeEnlazada<Alianza> para alianzas.
- Utiliza ListaSimplementeEnlazada<Carta> para cartas.

- Utiliza Tablero, Bitmap, Carta, Pieza, Satelite, Base, Mazo,Nave y Coordenada.
- Usa un scanner para entrada por consola.

Resumen

La clase menu coordina todo el juego, desde la configuración inicial hasta la ejecución del juego y su final, gestionando las reglas, turnos, acciones y efectos de las cartas. Es el nexo entre el jugador y la lógica del juego.

Bitmap

Bitmap permite crear, modificar y guardar una imagen BMP pixel por pixel. La usamos para visualizar el tablero tridimensional que se va a usar en el juego por lo que cada jugador va a poder ver sus naves propias/enemigas, bases propias/enemigas, satélites propios/enemigos y radiación durante el transcurso del juego

Sector

Representa una celda del tablero tridimensional en el juego. No permite crear sectores nulos. Permite obtener y asignar valores a ese sector además de obtener las coordenadas en las que se encuentra.

Tablero

La clase Tablero representa la estructura principal de un tablero de juego tridimensional. Su función es organizar y manejar los sectores del tablero, donde se colocan las distintas piezas del juego. Permite inicializar el tablero, asignar piezas en posiciones específicas y acceder tanto a sectores como a las piezas colocadas. También mantiene un registro de las piezas activas (no vacías).

Coordenada

Representa una posición tridimensional (x, y, z) en el tablero del juego. Está clase permite identificar, comparar y usar posiciones en el espacio del juego.
La clase se utiliza:

- Para ubicar y comparar la posición de naves, bases y satélites.
- Para buscar piezas en una ubicación específica del tablero.
- Para validar si dos piezas del juego están en la misma posición.

ValidacionesUtils

Clase de utilidades que contiene validaciones comunes para evitar errores en el tiempo de ejecución.
Sirve para verificar errores antes de usarlos y lanzar excepciones si no cumplen ciertas condiciones.

Sistema de Estructuras

Contiene implementaciones básicas de estructuras de datos lineales usando enlaces(nodos).

Sistema de Enums

Estas enums definen tipos fijos de valores utilizados en el juego. Sirven para mejorar la legibilidad y control de errores en el juego.

Tests

Contiene clases de pruebas automatizadas que usan JUnit para validar el comportamiento de distintas clases del tp, asegurando que funcionen correctamente ante distintos escenarios.