Primeramente tener en cuenta que tipos de estructuras vamos a utilizar para así mismo hacer las relaciones, en esta vez lo ideal es tener en cuenta todas las clases, al igual que los atributos..

Lo ideal también es consultar con Yesenia la info con que vamos a hacer el diagrama

Para abordar este proyecto, necesitaremos diseñar varias clases que representen diferentes partes del juego y su funcionamiento. Basándome en los requisitos que has proporcionado, propongo la siguiente estructura de clases en los paquetes `model` y `ui`, además de las excepciones necesarias para manejar posibles errores:

### Clases del Paquete `model`:

1. \*\*Board\*\*:

- \*\*Atributos\*\*:

- `int rows`: número de filas del tablero.

- `int columns`: número de columnas del tablero.

- `int seedsCount`: cantidad total de semillas en el tablero.

- `Cell[][] cells`: matriz de celdas que componen el tablero.

- \*\*Métodos\*\*:

- `void initializeBoard()`: inicializa el tablero con las semillas y enlaces aleatorios.

- `void linkCellsRandomly(int numberOfLinks)`: crea enlaces aleatorios entre las celdas.

- `void placeCharactersRandomly()`: coloca a Rick y Morty aleatoriamente en el tablero.

- `boolean allSeedsCollected()`: verifica si todas las semillas han sido recolectadas.

- `void displayBoard()`: muestra una representación ASCII del tablero.

- `void displayLinks()`: muestra una representación ASCII de los enlaces entre las celdas.

2. \*\*Cell\*\*:

- \*\*Atributos\*\*:

- `int id`: identificador único de la celda.

- `boolean hasSeed`: indica si la celda contiene una semilla.

- `boolean hasPortal`: indica si la celda contiene un portal.

- `Cell linkedCell`: referencia a la celda enlazada mediante un portal.

- \*\*Métodos\*\*:

- `void setSeed(boolean hasSeed)`: asigna la presencia de una semilla en la celda.

- `void setPortal(Cell linkedCell)`: asigna un portal hacia otra celda.

3. \*\*Player\*\*:

- \*\*Atributos\*\*:

- `String name`: nombre del jugador.

- `int seedsCollected`: contador de semillas recolectadas por el jugador.

- `Cell currentPosition`: referencia a la celda actual del jugador en el tablero.

- \*\*Métodos\*\*:

- `void rollDice()`: simula el lanzamiento del dado y mueve al jugador en el tablero.

- `void move(int steps)`: mueve al jugador la cantidad de pasos indicada por el dado.

- `void collectSeed(Cell cell)`: recolecta una semilla si la celda actual la contiene.

- `void teleport(Cell destination)`: realiza el teletransporte si la celda actual tiene un portal.

### Clases del Paquete `ui`:

1. \*\*GameConsole\*\*:

- \*\*Métodos\*\*:

- `void startGame()`: método principal que inicia el juego y maneja el flujo por turnos.

- `void displayMenu(Player currentPlayer)`: muestra el menú de acciones para el jugador actual.

- `void displayScore()`: muestra el marcador de semillas recolectadas por cada jugador.

- `void displayResult(Player winner)`: muestra el resultado final del juego.

### Clases del Paquete `exceptions`:

1. \*\*InvalidInputException\*\*:

- Excepción personalizada para manejar entradas inválidas del usuario durante la interacción con el juego.

### Consideraciones Adicionales:

- Implementar el manejo de excepciones para validar las entradas del usuario.

- Utilizar recursividad para recorrer el tablero y realizar acciones como moverse y recolectar semillas.

- Mantener un registro histórico de puntajes utilizando estructuras como arreglos, colecciones y árboles binarios de búsqueda.

- Guardar los puntajes en archivos para preservar el histórico de juegos.

Este diseño básico te proporciona una estructura inicial para comenzar a implementar tu proyecto. A medida que avances en la implementación, podrás ajustar y expandir estas clases según sea necesario para cumplir con todos los requisitos del juego. ¡Espero que este diseño te sea útil para planificar tu proyecto! Si tienes más detalles o necesitas ayuda con alguna parte específica, no dudes en preguntar.