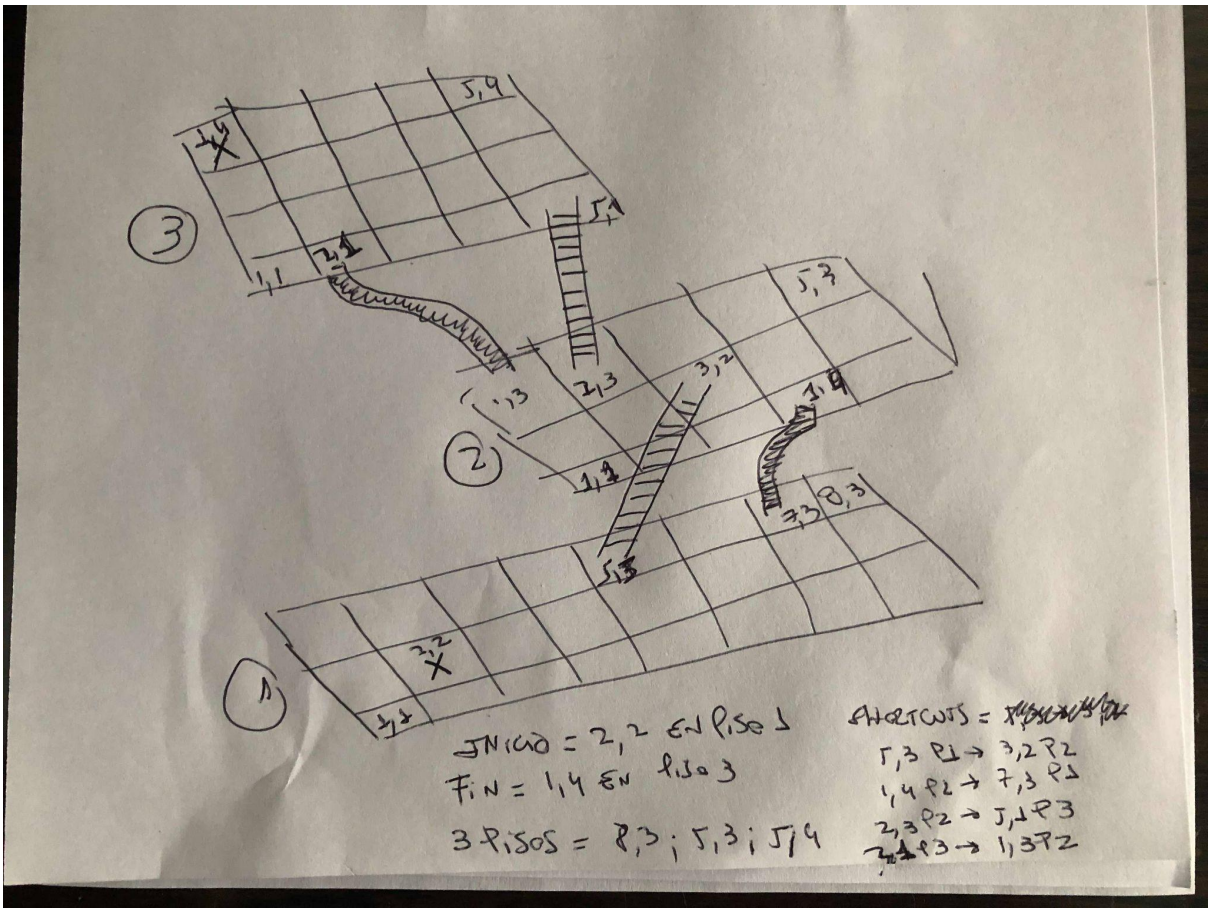
Ladders & Slides 3D

¡La empresa tiene una super idea de un juego que se cree romperá el mercado! Hay que implementar el modelo que soportará el juego más allá del front que se use. Debe ser una implementación robusta y estar desarrollada con TDD para asegurar la calidad.

El juego consta de un conjunto de pisos rectangulares, uno arriba del otro, de una extensión particular (pueden ser distintas) y compuesto por casillas dadas por sus coordenadas **x@y** que empiezan en la posición **1@1**. El objetivo es que los jugadores, que salen de una misma celda inicial de alguno de estos pisos, lleguen a la posición final que puede estar en el mismo u otro piso.



Para moverse, el jugador tiene que tirar un dado de **12** caras dos veces, una vez para la coordenada **x** y otra vez para la coordenada **y**. El dado de 12 caras va de los números 1 a 12, pero deben ser interpretados como de -5 a 6 al momento de moverse. Por ejemplo, si el Jugador1 está en el piso 1, posición 1@1 y tira un 7 y luego un 10, eso significa que tiene que sumar 1@4 a su posición actual, quedando en el piso 1, posición 2@5. Es requisito que al jugar se “tire un dado” porque se piensa ofrecer más adelante opciones de dados “con peso”, etc.

El jugador no puede “caerse” del piso. O sea que siempre como máximo (o mínimo) queda en el borde del piso. No rebota contra los bordes.

Para poder moverse de un piso a otro, hay “atajos” (shortcuts) que pueden ser escaleras (stairs) o toboganes (slides). Los atajos van desde una posición de un piso a la posición de otro piso.

En el caso de las escaleras, el piso al que llega debe estar arriba del piso del que sale.

En el caso de los toboganes, el piso al que llega debe estar debajo del piso del que sale.

Tanto las escaleras como los toboganes pueden saltear pisos. O sea, una escalera puede ir del piso 1 al piso 3, de manera análoga un tobogán puede ir del 5 al 2.

Cuando un jugador cae en la posición de donde sale un atajo, se mueve automáticamente a la posición de llegada del atajo.

Para no tener problemas al momento de mover un jugador al caer en los atajos, hay que validar que la construcción del juego sea correcta respecto de la configuración de los atajos. Por ejemplo, no puede haber más de un atajo saliendo de una misma posición de un piso, hay que evitar ciclos de atajos, no puede salir un atajo desde la posición inicial ni final, etc.

Se deben hacer todas las validaciones necesarias para asegurar que el juego esté bien construido respecto de los atajos y otras cosas que considere necesarias, pero para minimizar la cantidad de validaciones, para esta versión del modelo se puede asumir que:

1) siempre hay más de un jugador (como mínimo deben jugar dos jugadores)

2) siempre hay más de un piso (como mínimo deben haber dos pisos)

3) la extensión de los pisos es correcta (entero mayor en 1 en x e y)

4) la posición inicial y final son correctas (están en pisos y posiciones correctas)

5) siempre se puede llegar y salir de cada piso

6) las posiciones en los pisos se crean válidos (posiciones y pisos estrictamente positivos, etc)

Para que un jugador gane y termine el juego hay que caer justo en la posición final.

Ayudas:

- **CircularReadStream** es un stream que no se acaba nunca. Ejemplo de uso:

  ```
  stream := CircularReadStream on: #(10 20) moving: NullUnit new.
  stream current. ➜ 10
  stream next ➜ 20
  stream current ➜ 20.
  stream next ➜ 10
  ```

  Tener en cuenta que un ReadStream común no responde el mensaje current y la primera vez que se le envía el mensaje #next, devuelve el primer elemento, a diferencia del CircularReadStream que devuelve el segundo.

- El mensaje **#combinations: k atATimeDo: aBlock**, implementado en **SequenciableCollection**, aplica aBlock en las combinaciones de dimensión k de los elementos del receptor. Por ejemplo:

  ```
  combinations := OrderedCollection new.
  #(1 2 3 4) combinations: 2 atATimeDo: [ :combination |
          combinations add: combination copy ].
  combinations ➜ an OrderedCollection(#(1 2) #(1 3) #(1 4) #(2 3) #(2 4) #(3 4))
  ```

Entrega:

1. Entregar por mail el fileout de la categoría de clase **ISW1-2022-1C-2doParcial** que debe incluir toda la solución (modelo y tests). El archivo de fileout se debe llamar: **ISW1-2022-1C-2doParcial.st**
2. Entregar también por mail el archivo que se llama **CuisUniversity-nnnn.user.changes**
3. Probar que el archivo generado en 1) se cargue correctamente en una imagen “limpia” (o sea, sin la solución que crearon) y que todo funcione correctamente. Esto es fundamental para que no haya problemas de que falten clases/métodos/objetos en la entrega.
4. Realizar la entrega enviando mail a la lista de Docentes: ingsoft1-doc@dc.uba.ar con el **Subject: LU nnn-aa - Solución 2do Parcial 1c2022**
5. Subir a sus repos grupales los archivos **CuisUniversity-nnnn.image** y **CuisUniversity-nnnn.changes**. Debe **zipearlos** previamente para reducir su tamaño o podría dejar sin espacio disponible a sus compañeros. **Pueden eliminar las imágenes del 1er parcial para liberar espacio.**
6. Deberán subirlos al main branch de sus respectivos repos (tenga en cuenta hacer pull antes de ser necesario), y al subdirectorio **/Parcial2/LUnnn-aa/**

IMPORTANTE:

No retirarse sin tener el ok de los docentes de haber recibido el mail con la resolución.