## Descripción de la práctica

Esta práctica desarrollada por Hao Xu y Diego Baldoví, trata de crear un agente y sus métodos para poder competir en una partida del juego de mesa de Catan. Este agente se ha desarrollado con un algoritmo genético para poder ser entrenado y obtener los mejores resultados posibles. A este Agente lo hemos llamado HaoDiego.

## Algoritmo genético

### Cromosoma

Pensamos varios tipos de estrategias que el agente podría hacer en una partida como preferencias en algunas decisiones como por ejemplo qué material preferiría obtener o qué tipo de edificación construir. Todas estas posibles preferencias o estrategias que habíamos pensado las representamos mediante probabilidades según qué estrategia es más importante, de esta forma no sigue la misma estrategia durante toda la partida y va adaptándose. Nuestra representación del cromosoma es la siguiente:

Texto

El contenido generado por IA puede ser incorrecto.Dentro del genoma hemos metido varias listas, cada una de estas representa una situación en la que tomar una decisión durante el juego y dentro de cada una de éstas hay varios campos con números entre 0 y 1, representando la probabilidad de que se escojan como estrategia a seguir y la importancia o eficiencia (a priori) de esa estrategia.

Los valores que se muestran en la imagen y de los que se hablarán son la probabilidad relativa, para facilitar el cálculo de la función de selección, al inicio de la partida se sustituyen los valores por su probabilidad acumulada dentro de la propia lista.

**Beginning\_priority**: Esta lista representa las estrategias a seguir al iniciar el juego cuando el agente debe decidir dónde colocar el primer pueblo y la primera carretera:

**MAX\_RESOURCES\_TYPE:** Esta estrategia prioriza colocar el pueblo inicial en un nodo cuyos terrenos adyacentes tengan la mayor diversidad de tipos de recursos y comprobando también que esos recursos entren dentro de la prioridad de materiales.

**MAX\_DICE\_PROB**: Por la teoría del límite de la estadística sabemos que a la larga hay mayor probabilidad de que ciertos números salgan más en las tiradas de dados como las que se hacen el principio de la partida, siguiendo esta lógica, esta estrategia prioriza colocar el pueblo inicial en un nodo cuyos terrenos adyacentes tengan los números más probables para que salgan como resultado de los dados.

**Build\_priority**: Esta lista representa las decisiones a tomar en la fase de construcción del turno de nuestro agente:

**CITY\_FIRST**: Esta estrategia prioriza la construcción de ciudades

**TOWN\_FIRST**: Esta estrategia prioriza la construcción de pueblos

**ROAD\_FIRST**: Esta estrategia prioriza la construcción de caminos

**PORT\_HUNTER**: Esta estrategia prioriza la construcción de pueblos cerca de puertos

**CARD\_SPAM**: Esta estrategia prioriza la obtención de cartas de desarrollo

**Material\_priority**: Esta lista representa las prioridades a buscar unos tipos de recursos u otros. Estas prioridades se usan sobre todo en la fase de comienzo del juego para ver donde colocar el pueblo inicial.

**Thief\_priority**: Esta lista contiene las estrategias a seguir cuando se tiene que mover al ladrón y hay que decidir dónde colocarlo. Ambas estrategias buscan que haya al menos un enemigo y que no haya ninguna construcción de nuestro agente en ese terreno.

**MAX\_PLAYERS**: Esta estrategia busca el terreno con más pueblos o ciudades enemigas adyacentes.

**MAX\_DICE\_PROB:** Siguiendo la lógica de la misma estrategia de la lista de beginning\_priority, esta estrategia prioriza colocar al ladrón en un terreno con el numero más probable de los dados.

**Función de selección**

Para poder usar estas probabilidades en el código de la partida, hemos implementado una función llamada **choose\_priority()** que genera un numero aleatorio entre 0 y 1 y comprueba en que rango de probabilidad cae dentro de la probabilidad acumulada de la lista, es decir, estas probabilidades se convierten en:

[0.5, 0.2, 0.2, 0.1]

[0.5, 0.7, 0.9, 1]

De esta forma si visualizamos la probabilidad entera como un gráfico se vería algo así:

De manera que al generar un numero aleatorio entre 0 y 1 es como si estuviéramos eligiendo una posición que se corresponde con un color y por tanto con una estrategia a seguir. Parecido a tirar una ruleta.

Es importante añadir que la suma de todos los valores de una determinada lista siempre deben sumar 1, es gracias a esto que podemos seguir la lógica descrita en este apartado.

### Fitness

Nuestra función de fitness es el numero medio de puntos que se obtiene en 100 partidas contra agentes random usando el archivo benchmark\_vs\_random.py

### Entrenamiento

Para entrenar el algoritmo genético y decidir cual es el mejor cromosoma hemos seguido los siguientes pasos:

-Hemos creado una serie de cromosomas/individuos con valores en las listas que a primera vista parecen aleatorios. Dejando un individuo como base y comparación.

-Se ejecutaron las 100 partidas vs agentes random y se apuntaron el fitness de cada individuo

-Se seleccionaron por ranking los mejores individuos usando el fitness de este individuo que inicialmente dejamos como base y comparación. De esta forma no solo estamos escogiendo los mejores, si no que estamos escogiendo los que han obtenido una mayor mejoría respecto a un individuo base.

-Se realizaron operaciones de mutación swap, ya que otros tipos de operación podrían darnos individuos inválidos debido al hecho de que los valores de una lista deben sumar 1, si hacemos un crossover la suma podría resultar diferente al no tener control total sobre el resultado de esa operación.

-Creamos una nueva población sustituyendo los peores padres por los nuevos hijos

-Se realizaron los pasos anteriores hasta que la diferencia de fitness entre unas generaciones y otras sea muy pequeña o se sospeche que ésta diferencia se deba al azar por la aleatoriedad de la selección.

## Lógica de las funciones

La lógica de este agente tomo bastante inspiración del archivo **PabloAleixAgent.py**, aquí se explican las principales funciones:

**On\_start\_game():** Se creó un nuevo código inspirándose en el de este archivoya mencionado, que sigue con un if una de las dos estrategias, o comprueba que los terrenos tengan la mayor diversidad de recursos, o comprueba que el numero del terreno sea el más probable.

**On\_thief\_move():** Esta función también tomó como inspiración el archivo ya mencionado adoptando la forma en la que se obtiene los datos sobre si hay un nodo del agento o de los enemigos en un terreno dado, el resto es original siguiendo, al igual que en la función anterior, una de entre 2 estrategias, o busca un terreno con la mayor cantidad de enemigos en sus nodos, o busca los terrenos con un número más probable de salir de los dados.

**On\_build\_phase():** Toma prestada la funcionalidad de detectar el mejor nodo para colocar la construcción del archivo ya mencionado. Con esto la función decide, según el resultado de la selección de prioridad, qué edificación debería priorizar, si prioriza una ciudad, un pueblo, una carretera o cartas de desarrollo. El hecho de priorizar una construcción antes que otra no indica que solo pueda hacer ese tipo de construcción.

En esta práctica se usó chatGPT para ayudarnos a detectar errores que teníamos en el código mientras trabajábamos y para ayudarnos con posibles estrategias a tener en cuanta para el cromosoma.