Proyecto Monopoly Deal

Simulación + Inteligencia Artificial + Compilación

Kevin Talavera Díaz C-311

Facultad de Matemática y Computación Universidad de La Habana

Curso 2021-2022

Link: https://github.com/KevinTD15/Monopoly_Deal-Project

Introducción

El juego de cartas Monopoly Deal reúne toda la diversión del juego de mesa Monopoly en un rápido juego de cartas. Está compuesto por 106 cartas que incluyen cartas de Propiedad, cartas de Renta, cartas de Acción, cartas de Casa y Hotel, Cartas de Dinero y cartas de Comodín de Propiedad. Las cartas de Comodín de Propiedad permiten a los jugadores formar grupos de propiedades. Las cartas de Acción permiten a los jugadores cobrar rentas y hacer hábiles negocios. Las cartas de Casa y Hotel suben el valor de la renta. Y las cartas de Dinero sirven para pagar las deudas. Para ganar completa 3 grupos de Propiedades de distinto color antes que los demás. (Ver reglas en pdf adjunto)

Simulación

Está basada en agentes donde estos son los jugadores y pueden tener comportamientos tanto proactivos como reactivos en dependencia de estado del juego donde se encuentren. El medio ambiente es el tablero de juego, del cual los agentes pueden obtener la información necesaria para la toma de decisiones. Para un jugador, se encuentran implementados 3 tipos de comportamiento:

1- Jugar la carta (o cartas) que decida en dependencia de una heurística en su propio turno. Este es un comportamiento proactivo.
2- Reaccionar ante el efecto de cartas de adversarios en dependencia de una heurística, este es un comportamiento reactivo ya que es en respuesta a un estímulo externo, que, en este caso, viene de otro agente.
3- Descartar cartas de la mano.

Como resultado de realizar estas simulaciones, variando la cantidad de jugadores por partidas y los propios tipos de estos, así como la cantidad de posibles jugadas generadas para ser evaluadas (ver más en detalle en Inteligencia Artificial), se pudo mejorar las heurísticas.

Inteligencia Artificial

Este proyecto cuenta con dos de sus las aplicaciones: **heurísticas** y **procesamiento de lenguaje natural.**

El juego consta de 4 tipos de jugadores, de los cuales 3 tienen funciones heurísticas y uno es totalmente aleatorio. De los 3 jugadores "inteligentes" implementados, uno hace uso de probabilidades basadas en qué tan probable es que cartas que tenga algún contrario que pueda llegar a perjudicarlo junto a qué tan buena sería hacer la jugada, otro calcula que tan buena es una jugada pero sin tomar en cuenta acciones que pueda realizar el rival, este jugador fue tomado como base debido a su buen rendimiento en las pruebas que se realizaron para probar diferentes heurísticas e ir variando sus valores, y el mejor resultado obtenido fue el último de los jugadores inteligentes, el cual funciona justo como el anteriormente explicado pero con valores más certeros y un rendimiento aún mejor.

Las **heurísticas** fueron desarrolladas como parte de un problema de búsqueda donde se generan aleatoriamente una cantidad (definida por parámetros) de posibles jugadas y estas escogen la mejor de todas y son las que se ejecutan. Específicamente son 3, una que se centra en elegir la mejor jugada que le toca hacer a un jugador en su turno, otra que devuelve cuales serían las cartas que al descartarlas afecta menos al jugador (ver en las reglas cuando un jugador debe descartar cartas), y por último tenemos una que devuelve cuál es la mejor acción a tomar cuando tenemos que reaccionar ante la jugada de algún adversario.

El **procesamiento de lenguaje natural** (PLN) esta implementado para poder añadir nuevas cartas al juego de cualquiera de los tipos predefinidos con sus respectivos parámetros. Esto se logra mediante la implementación de una gramática para poder procesar los textos de forma eficiente y elegante. Luego al obtener el AST (Árbol de Sintaxis Abstracta) que este devuelve se verifica si es una carta válida, se crea y se añade al mazo de cartas para poder usarse en la simulación.

Gramática

Implementada para permitir al usuario crear cartas de todo tipo, exceptuando cartas de acción rápida.

Para añadir cartas nuevas el usuario puede escribir en lenguaje natural sus especificaciones, pero con las siguientes restricciones:

1- Para crear una carta de Propiedad válida, el usuario debe definirle un nombre, este tiene que tener la letra inicial en mayúscula; también debe de especificarse el color, de cuantas propiedades de ese color estará conformado el grupo (Ej. Las propiedades azules tienen grupo de tamaño 2), cuales son las rentas, las cuales son el resultado de tener varias propiedades de un mismo color en el tablero, estas tienen que estar en consonacia con lo dicho en el punto anterior (Ej. Como las azules tienen grupo de tamaño 2, este grupo solo podrá tener 2 rentas, estos deben ser ingresados por coma de la forma: <num><,><' '><num> …), y por último se de debe definir un valor a la carta, para esto de debe teclear la palabra "valor" o "valga" seguido de cualquier palabra o grupo de palabras y luego el valor que queremos asignarle seguido de la letra "M" (esta hace referencia a la moneda del juego). Todo esto puede ser tecleado en cualquier orden.

2- Para crear una carta de Comodín válida, el usuario debe definirle un nombre, este tiene que tener la letra inicial en mayúscula; también debe definir los colores por los que se podrá jugar esta carta, y en el caso que desee crear el comodín maestro, no escribir ningun color específico o escribirlos todos, y por último definirle un valor, análogo a como se hace en la creación de propiedades.

3- Para crear una carta de Dinero basta con poner el valor desead.o seguido de "M".

4- Para crear una carta de Acción Renta el usuario debe definirle un nombre, este tiene que tener la letra inicial en mayúscula; también debe teclear todos los colores para los que sea válida esa renta y en caso de no definir un color se tomará como si valiera por todos, también hay que especificar si es un efecto contra todos los jugadores o no, en caso de ser contra todos hay que escribir "todos" o "cada" en caso contrario es teclear la cantidad la cantidad de jugadores contra los que se usa la carta de la palabra "jugadores" o "participantes" y por último el valor de esta carta, análogo a como se hace en la creación de propiedades.

5- Para añadir una carta de Acción Construcción el usuario debe definirle un nombre, este tiene que tener la letra inicial en mayúscula; el tipo de construcción ya sea "casa" u "hotel", luego definir el monto por el que nos tendrán que pagar tecleando el valor seguido de "M" y por último el valor de esta carta, para esto de debe teclear la palabra "valor" o "valga" seguido de cualquier palabra o grupo de palabras y luego el valor que queremos asignarle seguido de la letra "M".

6- Para crear una carta de Accion para tomar cartas del mazo el usuario debe definirle un nombre, este tiene que tener la letra inicial en mayúscula; luego debe poner la cantidad de cartas que desea tomar seguido de la palabra "carta", "tarjeta" o "naipe", y en caso de ser más de una poner esas palabras en plural y por último el valor de esta carta, análogo a como se hace en la creación de propiedades.

7- Para crear una carta de Acción que robe propiedades, el usuario debe definirle un nombre, este tiene que tener la letra inicial en mayúscula; también debe decidir si será una carta con la cual 2 jugadores intercambien cartas o si las roba directamente, y esto le logra poniendo la palabra “intercambio” o “intercambiar” y en caso que lo que desee sea robar, no teclear ninguna de estas 2 palabras. Luego hay que definir cuantas cartas se van a tomar del rival tomar seguido de la palabra “carta”, “tarjeta” o “naipe”, y en caso de ser más de una poner esas palabras en plural y en caso de que no se defina una cantidad se definira que la carta roba un grupo completo de color y por último el valor de esta carta, análogo a como se hace en la creación de propiedades.

8- Para crear una carta de Acción que robe dinero, el usuario debe definirle un nombre, este tiene que tener la letra inicial en mayúscula; luego se debe definir si se va a usar contra todos o contra 1 solo jugador, en caso de ser contra todos hay que escribir “todos” o “cada” en caso contrario es teclear la cantidad la cantidad de jugadores contra los que se usa la carta de la palabra “jugadores” o “participantes”, también tiene que definir el monto que el/los participantes deberán pagar seguido de la letra “M” y por último el valor de esta carta, análogo a como se hace en la creación de propiedades.

Compilación

Para implementación del DSL se utilizaron las librerías lex (para el analizador léxico) y yacc (para el análisis sintáctico) de PLY y se partió un link[i] que tenía desarrollado un compilador básico que ejecutaba las tablas de multiplicar, en el estaban implementadas las instrucciones, creación de ciclos while, bloques if-else, creación de números (enteros) e impresión por consola (print). Su lenguaje incluía declaración de variables y el símbolo ‘;’ al final de cada instrucción al estilo de otros lenguajes.

El compilador desarrollado interpreta y ejecuta código fuente del lenguaje de programación diseñado para la aplicación, la extensión de los programas de este lenguaje es ‘mpd’ (Monopoly Deal).

Las funcionalidades incluidas son las siguientes:

1. Declaración de funciones (con y sin retorno). Permite incluir lista de argumentos.
2. Ejecución de funciones, tanto las definidas en el DSL, como en el juego de monopolio desarrollado en Python.
3. Tratamiento de listas (crear lista, añadir elementos, asignar valor a un elemento, obtener el valor de un elemento). Se reconocen también listas definidas en un módulo del juego de monopolio de Python, pero solo se gestiona, en este caso, la asignación de un valor, añadir un elemento y obtener valor.
4. Permite gestionar los atributos de una clase declarada en el módulo de juego diseñado.
5. Gestión de variables definidas en módulos del juego.
6. Permite añadir jugadores a la lista de jugadores del juego, cuyos elementos son instancias de clases en dependencia del tipo de jugador.
7. Permite crear cartas nuevas haciendo uso internamente del módulo de procesamiento de lenguajes implementado en Python.
8. Las funciones pueden formar parte de las expresiones.
9. Permite crear una instancia del juego.
10. Permite obtener la longitud de una cadena o lista
11. Se implementó tratamiento de errores. En el intérprete muestra mensaje de error y la fila de la instrucción que provocó el error.

En el análisis léxico y sintáctico se realizó en el módulo gramática.py. En cuanto al léxico, se ajustó a los términos del DSL del juego de monopolio; ejemplo, se eliminó el punto y coma al final de cada instrucción, se dejaron las llaves como forma de definir bloque de instrucciones y se excluyó la declaración de número. Se añadieron palabras reservadas que fueron llevadas a expresiones regulares para validar la entrada y el AST se fue construyendo mediante instanciación de clases, permitiendo así, una vez construido el árbol, en el módulo interprete.py, ejecutar la secuencia de instrucciones generadas en dependencia de la clase instanciada. Para el tratamiento de los objetos asociados a los módulos del juego en cuestión, se creó en cada situación una nueva clase que identificara que estaba en una instrucción externa para ejecutarla a partir de la instancia del juego creada que también se crea mediante una instrucción del DSL .

Se añadieron palabras reservadas siguientes:

'func' : 'FUNC',
'retorno' : 'RETURN',
'len' : 'LEN',
'crear' : 'CREAR',
'juego' : 'JUEGO',
'agregar' :'AGREGAR',
'jugador' : 'JUGADOR',
'aleatorio' : 'ALEATORIO',
'inteligentebasico' : 'INTELIGENTE1',
'inteligenteprobabilistico' : 'INTELIGENTE',
'inteligentemejorado' : 'INTELIGENTE2',
'ganador' : 'GANADOR',
'notificaciones' : 'NOTIFICACIONES',
'jugadores' : 'JUGADORES',
'reestablecer' : 'REESTABLECER',
'ejecutarturno' : 'EJECUTARTURNO',
'carta' : 'CARTA',
'finaljuego' : 'FINALJUEGO'

Sintaxis del lenguaje:

- **Imprimir**:
  <*imprimir*> < (> < expresión de cadena > <) >
  Se permite concatenas expresiones de cadena con “&”.

- **Asignación**:
  < variable > < = > < expresión de cadena >

- **Mientras**:
  <*mientras*> < (> < expresión lógica > <) > < { > < instrucciones > < } >

- **Si**:
  < *si* > < (> < expresión lógica > <) > < { > < instrucciones > < } >

- **Sino**:
  < *sino* > < { > < instrucciones > < } >

- **Declaración de funciones**:
  < *func* > < identificador > < (> < parámetros > <) > < { > < instrucciones > < } >

- **Llamado de función**:
< identificador > < (> < parámetros > <) >

- **Retorno**:
< *retorno* > < expresión de cadena >

- **Agregar elemento a una lista**:
< *agregar* > < expresión de cadena > < identificador >

- **Crear el juego**:
< *crear* > < *juego* >

- **Crear jugador**:
< *crear* > < *jugador* > < tipo > < identificador >

- **Crear carta**:
< *crear* > < *carta* > < cadena >

- **Reestablecer parámetros del juego**:
< *reestablecer* > < *juego* >

- **Ejecutar turno del juego**:
< *ejecutar* > < *turno* >

Variable: nombre, un elemento de una lista.
Identificador: nombre.
Expresión de cadena: números, cadenas, elementos de listas, funciones.
Expresión lógica: < expresión de cadena > < símbolo de comparación > < expresión de cadena >
Tipo: aleatorio, inteligenteProbabilistico, inteligenteBasico, inteligenteMejorado

[i] https://ericknavarro.io/2020/03/15/26-Interprete-sencillo-utilizando-PLY/