UNIVERSIDAD DE ORIENTE

NÚCLEO ANZOÁTEGUI

ESCUELA DE INGIENERIA Y CIENCIAS APLICADAS

DEPARTAMENTO DE COMPUTACION

TALLER DE INTELIGENCIA ARTIFICIAL

JUEGO DEL ZORRO Y LOS CAZADORES

Profesor:

Claudio Cortinez

Integrantes:

Brazon Eulises

Báez Josue

Barcelona 13 de marzo del 2023

Tipo de Agente: Basado en Utilidad

Un agente basado en la utilidad es un tipo de agente inteligente que utiliza una función de utilidad (valuar_tablero) para medir la calidad de las diferentes acciones que puede realizar y elegir (max_evaluacion = max(max_evaluacion, evaluacion)) la acción que maximiza su utilidad. Una función de utilidad es una medida numérica de qué tan bien se valora un resultado determinado y se utiliza para calificar la bondad de las acciones de un agente hacia una meta.

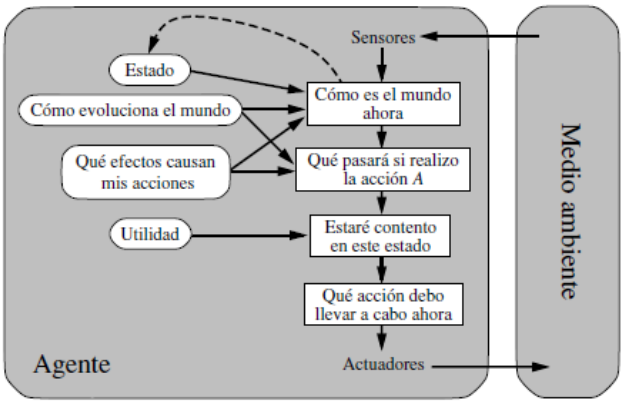
Los agentes basados en la utilidad siguen un enfoque "racional" para la toma de decisiones. Elige las acciones que espera que produzcan los mejores resultados con base en la información disponible en ese momento (la percepción del entorno). Una función de utilidad es una herramienta importante para implementar este enfoque racional, ya que proporciona una forma de comparar diferentes acciones posibles.

Para utilizar un agente basado en utilidades, primero debe definirse las funciones de utilidades que sean apropiadas para el problema que se está resolviendo (ruta más corta hacia la meta, no dejar que los cazadores se acerquen mucho, mientras más cerca se encuentre de la meta recibirá mejor puntuación). Esta función puede ser muy simple o muy compleja y puede depender de muchos factores, tales como: Metas del agente, restricciones ambientales, recursos disponibles y riesgos o costos potenciales asociados con varias acciones.

Una vez definida la función de utilidad, el agente puede usarla para evaluar diferentes acciones posibles y elegir la acción que maximiza su utilidad esperada. Este proceso de elegir el mejor curso de acción puede implicar la consideración de múltiples opciones y el uso de técnicas de búsqueda y optimización para encontrar la mejor solución (minimax).

Un agente basado en la utilidad es un tipo de agente inteligente que utiliza una función de utilidad para medir la calidad de diferentes acciones posibles, en este caso al implementar la técnica de minimax elige la acción que maximiza su utilidad al evaluar los desplazamientos del zorro y minimiza la utilidad de los cazadores en sus posibles movimientos, la función de utilidad es una herramienta importante para implementar un enfoque racional para la toma de decisiones y puede depender de varios factores que se deben considerar en la toma de decisiones.

Describir los elementos asociados al Agente como son las Percepción, y actuadores.



En esta imagen se puede visualizar cuales son los elementos asociados al agente basado en utilidad

Los **Sensores**, en este caso al tratarse de una recreación digita, no hay presencia de sensores físicos, pero si existe a la interacción con el juego a través de eventos (pulsaciones del mouse), que correspondes con coordenadas en una matriz de pixeles que tienen relación con el tablero.

El **entorno**, está representado en una matriz 8x8, y cada ficha posee una coordenada asignada.

Con la representación del entorno se evalúa las posibles acciones, y como estas van a influir en las **medidas de rendimiento** (que tan lejos se encuentra de la meta, que tan cerca se encuentran los cazadores, cual es la ruta más corta), según estos resultados, se buscara aquella medida que retorne una mayor utilidad, para decidir la acción a realizar (a cuál casilla se debe de mover el zorro).

Los **actuadores,** al tratarse de un entorno digital no hay actuadores físicos, pero si hay acciones a realizar, y que son representados a través de una interface gráfica, sonde se visualiza la posición de las fichas, y se observa cómo actúa cada elemento atreves de la interface.

Describir la estrategia ganadora del Agente.

La estrategia para que el agente inteligente pueda tomar sus propias decisiones y elegir el movimiento sea más conveniente, se utilizan principalmente dos técnicas:

Minimax: El método minimax es un algoritmo que utilizan los agentes inteligentes a la hora de tomar decisiones en presencia de adversarios. El objetivo de este algoritmo es encontrar la mejor estrategia para el agente dada la estrategia del oponente, asumiendo que el oponente está tratando de minimizar el resultado del agente.

El algoritmo funciona evaluando todos los movimientos posibles que puede realizar un agente, y para cada uno de ellos simula los movimientos del oponente y evalúa el resultado final. El resultado final tiene la forma de una función de utilidad que mide la ganancia o pérdida del agente.

Luego, el agente elige el movimiento que maximiza el mejor resultado que puede lograr, es decir, la estrategia que produce el mejor resultado posible. Esto se debe a que el oponente elige el movimiento que tiene el peor resultado posible para el agente. El método minimax se utiliza en muchas áreas, incluidos los juegos de estrategia como el ajedrez y el Go, los sistemas de recomendación, la toma de decisiones financieras y el control de procesos industriales.

Es una técnica cuyo código se divide principalmente en tres partes, por una parte, se define un **caso base**, es en este caso base que se evaluar las condiciones actuales del ambiente para para retornar un valor que sirve como punto de comparación entre las posible jugadas (valuar_tablero), por otra parte, se encuentra el bloque asociado a **es_maximizador,** en este bloque de código se evalúa todas las posibles jugadas que puede realizar el zorro, y se selecciona aquella que maximice las posibilidades de ganar, según el resultado que se retorne desde el caso base

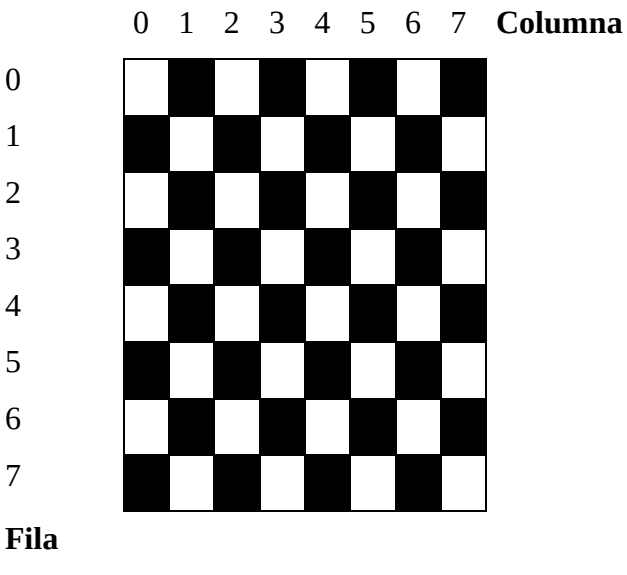
Luego se encuentra el bloque contrario a maximizador, que corresponde con la **minimización**, donde se evalúa las posibles jugadas que puede realizar cada uno de los jugadores, y se selecciona la jugada, que sea más desfavorable para los cazadores.

Una vez que se define como ser realizaran las evaluaciones de las distintas posibilidades de las jugadas, pasemos a el código base **valuar_tablero,** este método recibe como parámetros tanto la posición del zorro como de los cazadores, y crea una puntuación para distintas condiciones, que definen el comportamiento de la IA, estas puntaciones son:

- Si se encuentra posicionado en una fila lejana de la meta, se le resta puntos, mientras más lejos este de la meta mayor será la penalización
- Si la posición del zorro corresponde con alguna de las metas, se le da una recompensa bastante alta, para que se elija esta opción por sobre cualquier otra condición.
- Si el zorro se encuentra en la fila más alejada de la meta recibe una penalización alta.
- Entre las posibles jugadas, se calcula la ruta más corta hacia la meta (dijkstra), mientras más larga sea la ruta que debe recorrer, mayor será la penalización.
- Mientas los cazadores estén más cerca del zorro, recibe una mayor penalización.

Cada una de estas condiciones poseen su propia puntuación, unas mayores que otras (según la importancia de la condición), luego de evaluar todas estas condiciones se tiene una puntuación general que resulta de la suma de todas las condiciones, es esta puntuación que ser retorna hacia el minimax en el caso base, para poder así optimizar la utilidad, eligiendo aquella opción que genere mejor puntuación.

Documentar brevemente las estructuras de datos (TDA) empleadas para el algoritmo.



Para representar el entrono dentro de la agente se definió un arreglo bidimensional, donde la primera coordenada corresponde con la fila, y la segunda con la columna (fila, columna).

Inicialmente a cada posición se le coloca un símbolo distinto '-' para las casillas libres y 'X' para las casillas bloqueadas, esta representación se utilizó como entrada para crear un grafo con ayuda de la librería networkx (pip install networkx),

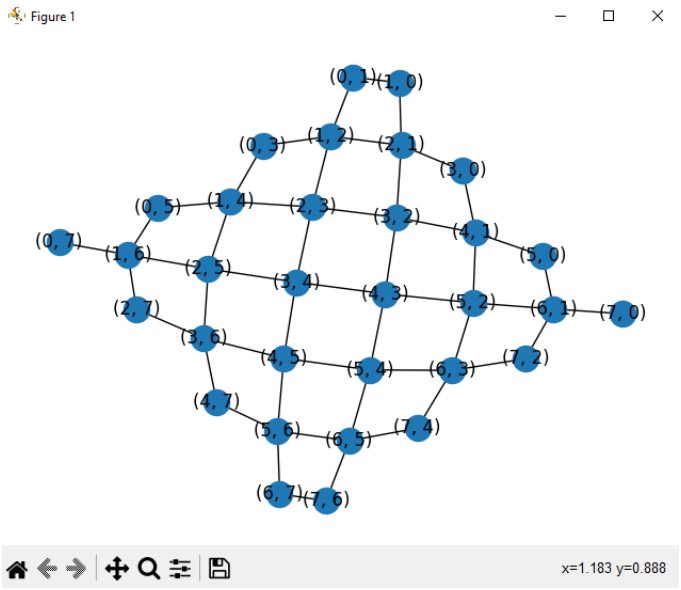
```
tablero = [['X', '-', 'X', '-', 'X', '-', 'X', '-'],
           ['-', 'X', '-', 'X', '-', 'X', '-', 'X'],
           ['X', '-', 'X', '-', 'X', '-', 'X', '-'],
           ['-', 'X', '-', 'X', '-', 'X', '-', 'X'],
           ['X', '-', 'X', '-', 'X', '-', 'X', '-'],
           ['-', 'X', '-', 'X', '-', 'X', '-', 'X'],
           ['X', '-', 'X', '-', 'X', '-', 'X', '-'],
           ['-', 'X', '-', 'X', '-', 'X', '-', 'X']]
```

EL grafo que se crea contiene los distintos nodos identificados con las coordenadas a la cual corresponde, y los vértices, representan las casillas con las cuales tiene comunicación, esto servirá de base, para implementar el algoritmo de Dijkstra.

```
def show(self):
    nx.draw(self.grafo, with_labels=True)
    plt.show()
```

Con ayuda de la libreria matplotlib (pip install matplotlib), se visualiza como queda estructurado.

Es este grafo que forma parte de la base de unas de las evaluaciones dentro de valuar_tableto, que se utiliza para tomar en cuenta la ruta más corta desde la posición del zorro, hacia las coordenadas que corresponden a las metas, el cual influye en la toma de decisiones dentro de la IA. En las posiciones donde se encuentren los cazadores, se eliminan los nodos, ya que las rutas asociadas a ellas, no son válidas.



Para representar a los jugadores se definieron, atributos por separado del tablero, dado que esto simplifica la reutilización del grafo base, al no tener que recalcularlo por completo en cada iteración, y contar con las coordenadas directamente, en lugar de recorrer el tablero para encontrar la posición de cada ficha.

```
#(fila, columna)
self.zorro_posicion = zorro_posicion
self.cazadores_posiciones = cazadores_posiciones
self.movimientos_cazadores = [(-1, 1), (-1, -1)]
self.movimientos_zorro = [(-1, 1), (1, 1), (-1,-1),(1,-1)]
```

Desarrollar el algoritmo como estrategia ganadora del Agente.

Algoritmo de minimax

```
función minimax(nodo, profundidad, esJugadorMaximizador)
    si la profundidad es 0 o el nodo es un nodo terminal
        retornar valuar_tablero #el valor de evaluación del nodo
    si esJugadorMaximizador
        mejorValor := menos infinito
        para cada hijo del nodo #cada posible jugada del zorro
            valor := minimax(hijo, profundidad - 1, FALSO)
            mejorValor := max(mejorValor, valor)
        retornar mejorValor
    sino
        mejorValor := más infinito
           #cada combinación posible de jugadas de los cazadores
        para cada hijo del nodo
            valor := minimax(hijo, profundidad - 1, VERDADERO)
            mejorValor := min(mejorValor, valor)
        retornar mejorValor
```

En este algoritmo, 'nodo' representa el nodo actual en el árbol de búsqueda, 'profundidad' representa la profundidad máxima de la búsqueda, el ¨esJugadorMaximizador¨ indica si el jugador actual está tratando de maximizar la puntuación.

La función minimax utiliza llamadas recursivas para explorar todos los movimientos posibles y determinar el mejor movimiento posible. Con cada llamada recursiva, los roles del jugador se intercambian, lo que permite evaluar todos los movimientos posibles.

El algoritmo minimax es una técnica de inteligencia artificial popular para la toma de decisiones en los juegos y se ha utilizado en muchos juegos exitosos como el ajedrez y el Go.

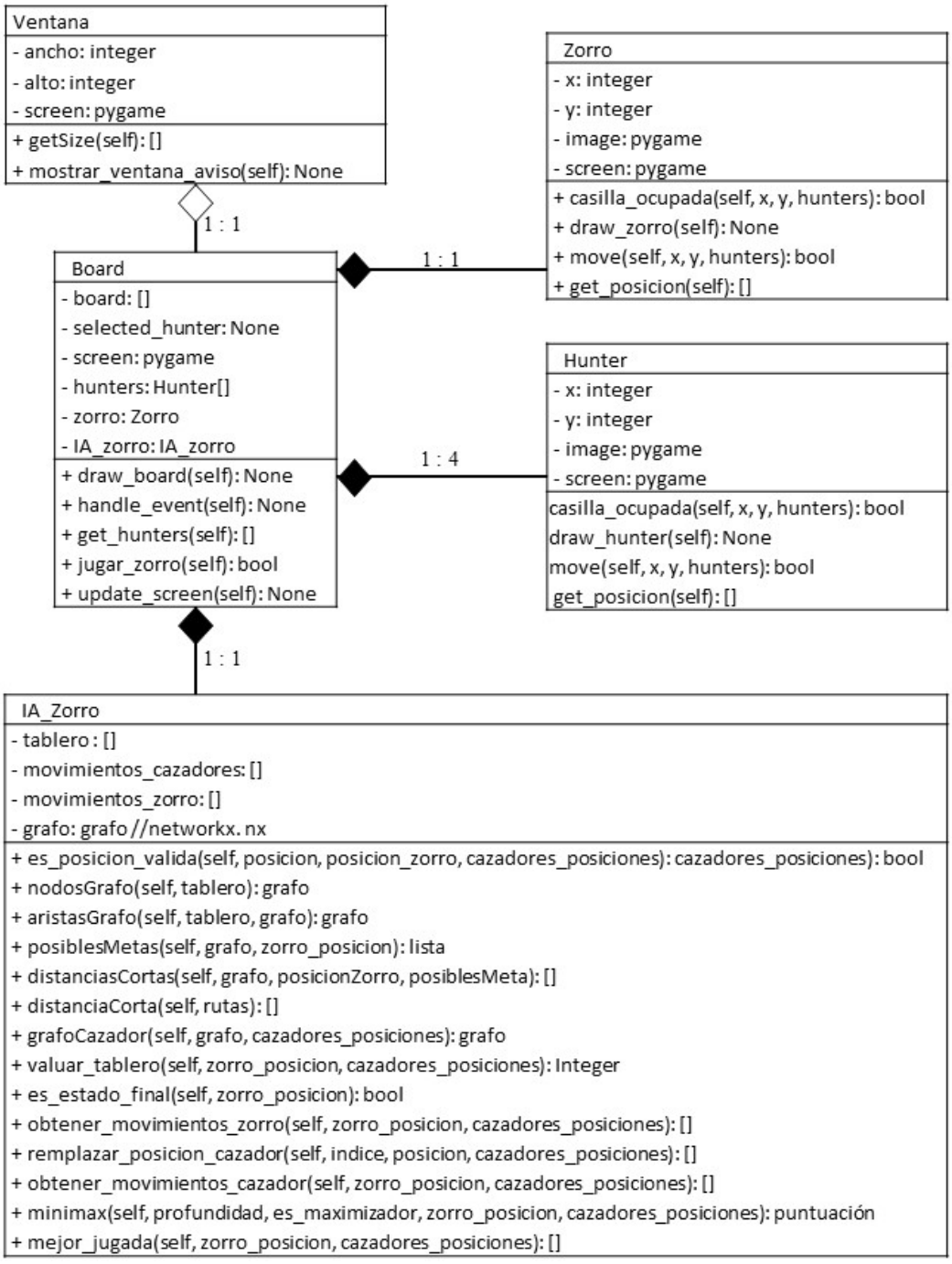
Algoritmo valuar_tablero

```
def valuar_tablero(self, zorro_posicion, cazadores_posiciones):
        puntaje = 0

       #si la posicion es la meta, se le da una recompenza muy alta para que
       #elija esa opcion por sobre todas las demas
        if self.es_estado_final(zorro_posicion):
            puntaje += 10000
        #Penalizar si el zorro está en una fila antes de la fila 8
        if zorro_posicion[0] < 7:
            puntaje -= 10000
        #mientras más lejos este de la meta menos punto recibirá
        puntaje -= 30*(8-zorro_posicion[0])

        #Puntaje basado en la ruta más corta
        #cuantas casillas tiene que recorrer, en la ruta más corta para
        #llegar a la meta una de las metas
        #mientras más distancia tenga que recorrer mayor será la
        #penalización
        #grafo que toma en cuenta la poscicion de los cazadores
        grafoCazador = self.grafoCazador(self.grafo, cazadores_posiciones)
        distancia_zorro = len(self.distanciaCorta(self.distanciasCortas(
                        grafoCazador, zorro_posicion, self.posiblesMetas(
                        grafoCazador, zorro_posicion))))
        puntaje = puntaje -(20*distancia_zorro)
        #mientras se encuentre más alejado de los cazadores mejor
        distancias = []
        for cazador in cazadores_posiciones:
            distancia_horizontal = abs(cazador[1] - zorro_posicion[1])
            distancia_vertical = abs(cazador[0] - zorro_posicion[0])
            distancia_maxima = max(distancia_horizontal, distancia_vertical)
            distancias.append(distancia_maxima)
        distancia_cazadores = sum(distancias)
        puntaje = puntaje + (10*distancia_cazadores)

        return puntaje

def mejor_jugada(self, zorro_posicion, cazadores_posiciones):

        movimientos_zorro = self.obtener_movimientos_zorro(zorro_posicion,
        cazadores_posiciones)

        contador = 0
```

```
        for movimiento in movimientos_zorro:
            if contador == 0:
                #almaceno el primer movimiento, y el valor que este arroja
                mejor_movimeinto = movimiento
                valor_movimiento = self.minimax(self.profundidad, True,
                                        movimiento, cazadores_posiciones)
            else:
                valor_jugada = self.minimax(self.profundidad, True,
                                    movimiento, cazadores_posiciones)
                #si existe una jugada con mejor valoracion
                if valor_movimiento < valor_jugada:
                    mejor_movimeinto = movimiento
                    valor_movimiento = valor_movimiento
            contador += 1

        return mejor_movimeinto
```

El algoritmo de mínimax por sí solo, retorna una valoración, considerando la mayor cantidad de factores posible, pero es solo eso, una valoración, es necesario contar con una función adicional que se encargue de comparar todas esas estimaciones, y al conocer la mejor entre todas, indicar cuál es la acción a realizar que para este caso son las coordenadas hacia donde se debe desplazar el zorro.

Describir el diagrama de diseño de clases asociado al Proyecto (Agente-Ambiente).



la IA_Zorro está pensada para que al recibir la posición actual del zorro y la posiciones de los cazadores, retorna una coordenada, que es utilizara la clase Board para mover al zorro, con ayuda de la clase ventana se representara gráficamente las interacciones que se produzcan en el entorno de juego,

Describir las características del Ambiente presentes.

El **medio ambiente** en este caso se trata de un tablero de ajedrez de que puede ser representado a través de una matriz de 8x8, este posee cuatro cazadores, y un zorro cada uno con su respectiva coordenada correspondiente a la fina y columna.

Se trata de un medio ambiente **totalmente observable**, donde el agente tiene acceso completo a toda la información relevante en el entorno en el momento que lo necesita (cuando es su turno de jugar).

El medio es **semideterminista** porque, las acciones que se producen por la IA del zorro están determinado por las condiciones actuales (cuando es su turno), pero el próximo movimiento que se realizará con los cazadores, existe cierta incertidumbre, aun así, se pueden evaluar las distintas circunstancias que se pueden presentar para que el agente optimice su comportamiento para maximizar su utilidad esperada dentro del entorno.

En entorno es **secuencial** dado que el juego se desarrolla en múltiples turnos, y cada jugada se ve afectado por el turno anterior. El agente debe tomar una serie de decisiones basadas en el estado actual del juego y como afectaran sus movimientos en el futuro para maximizar sus posibilidades de ganar.

El ambiente se considera **semidinámico** ya que, aunque el tablero de ajedrez es estático (no cambia durante el juego, el turno del zorro), se considera semidinámico porque las estrategias y decisiones de la IA dependen del estado actual del juego, que varía dependiendo de la jugada de los cazadores, lo que lo convierte en un sistema dinámico. La IA debe actualizar constantemente su estimación de la posición actual y ajustar su estrategia en consecuencia, convirtiéndolo en un sistema semidinámico.

El medio se considera con estados **discretos** ya que puede verse como un conjunto finito de estados y acciones discretos. Cada turno el jugador elige una de las piezas del tablero y la mueve a otra casilla. Esta perspectiva se usa comúnmente en juegos de mesa y puede abordarse mediante algoritmos de búsqueda como el algoritmo minimax.

El agente se considere que es un **multiagente,** el juego de ajedrez es muy similar a este, en donde se sabe que es entorno donde cada jugador busca mejorar su propia condición con cada jugada, para esta caso del cazador y el zorro un agente viene representado por la inteligencia artificial, que implementando diversas técnicas elige la que considera la mejor opción, y el jugador humano también se considera un agente inteligente, que posee la capacidad de razonar y tomar decisiones para actuar, en base a lo que percibe del entorno, la IA lo percibe a través de representaciones en matrices, y el ser humano lo percibe a través de una interface.

Describir brevemente las herramientas empleadas para la implementación.

Python es un lenguaje de programación interpretado, dinámico y de alto nivel. Es un lenguaje multiplataforma y puede ejecutarse en diferentes sistemas operativos como Windows, macOS y Linux. Este es un lenguaje de programación muy popular debido a su simplicidad y facilidad de uso, tiene una sintaxis clara y concisa que hace que el código sea fácil de leer y comprender. Además, cuenta con numerosas bibliotecas y herramientas para crear aplicaciones y soluciones a diversos problemas.

Pygame es una biblioteca de Python para crear videojuegos y otras aplicaciones multimedia interactivas. Ofrece una variedad de herramientas y funciones para crear gráficos, animaciones, efectos de sonido y música en juegos y aplicaciones. Es una herramienta muy útil para crear videojuegos interactivos y aplicaciones multimedia en Python, muy utilizada por programadores, educadores y entusiastas de los juegos de todo el mundo.

NetworkX es una biblioteca de Python para crear, manipular y explorar estructuras de red complejas. Está diseñado para funcionar en cualquier tipo de red, como redes sociales, de transporte, de telecomunicaciones, biológicas, etc. Ofrece una amplia gama de herramientas para crear, editar y analizar redes. Las funciones incluyen la creación y edición de diagramas, la adición de nodos y aristas, el cálculo de medidas y estadísticas de red, y el dibujo de diagramas fácilmente personalizables.

Matplotlib es una biblioteca de visualización de datos de Python que permite crear diagramas, gráficos y visualizaciones en 2D y 3D. Es una de las bibliotecas más utilizadas en Python para crear gráficos y visualizaciones, especialmente en los campos de la ciencia y el análisis de datos.

sys es una biblioteca integrada de Python que brinda acceso a algunas variables y funciones específicas del intérprete, como lo es sys.exit(), que se utiliza para salir del programa.