Nombre del Alumno:

Examen Resuelto de Sistemas Inteligentes

29 de mayo de 2023

Instrucciones generales para realizar el examen:

- **Pon tu nombre en todas las hojas del examen**.
- Escribe con letra clara y sin faltas de ortografía
- No se permiten dispositivos electrónicos salvo una calculadora
- **Contesta a las preguntas en el hueco que se proporciona**. Puedes apoyarte en tantas hojas en sucio adicionales como quieras, pero las respuestas deben ir en el examen para evitar que se pierdan. Las hojas de respuesta pueden estar escritas por las dos caras y podéis utilizar el reverso del enunciado del examen en caso de que hiciese falta.
- El tiempo para la realización del examen es de **2.5h**.

Problema 1 (2,5 puntos)

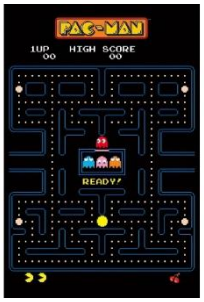
Queremos modelar un sistema de agentes autónomos que controle a los fantasmas del videojuego Pac-Man. El jugador en este videojuego controla un personaje en forma de esfera amarilla con una abertura que simula la boca del personaje. Su movimiento es automático en la última dirección presionada por el jugador. Si llega a una pared el jugador se detiene. El objetivo del juego es comerse todas las bolitas amarillas que hay en el escenario para tener más puntuación, evitando que te toquen los fantasmas que son tus enemigos. Hay unas bolitas amarillas más grandes que otorgan invulnerabilidad al jugador durante cierto tiempo. Durante ese tiempo, el jugador si toca a alguno de los enemigos se lo puede comer. Una parte del escenario no tiene límite en el borde, lo que permite salirse del mismo y entrar por el lado contrario del escenario.

El comportamiento que realizarán los fantasmas que tenemos que modelar es el siguiente.

- El fantasma intentará moverse preferentemente en la dirección que haga aproximarse más al jugador.
- Si choca con una pared, cambiará de dirección de forma aleatoria
- Si el jugador tiene invulnerabilidad (se ha comido la bola amarilla grande), se moverá en la dirección contraria a la del jugador.
- Si el jugador se come al fantasma, este desaparece durante unos segundos y reaparece en el centro del escenario.
- En esta versión de Pac-Man los fantasmas pueden comerse unas frutas que le dan poderes adicionales. Si el fantasma está cerca de la fruta (a menos de 8 casillas) intentará comerse la fruta antes que el jugador.

Nombre del Alumno:

- Al comerse la fruta, el fantasma durante segundos es invulnerable a Pac-Man incluso con la bola amarilla. Cuando el fantasma está en este modo también aumenta ligeramente su velocidad. En este modo el jugador vuelve a ir a por Pac-Man preferentemente y sólo se ve una fruta cerca (o no puede continuar en dicha dirección) se desviará de su trayectoria.



Se pide:

1. ¿Qué tipo de agente es: simple, basado en modelo, basado en objetivo…? Razona la respuesta. (0.5 p)

Es un agente basado en modelo ya que dispone de una memoria que guarda el estado en el que se encuentra el agente y en base a esa memoria y a la percepción toma una decisión. No puede ser simple debido a que el agente debe recordar si ha tomado la fruta o no. Es algo que no puede deducir directamente de la percepción. Tampoco puede ser basado en objetivos. Aunque pueda ir puntualmente hacia la fruta o hacia Pacman. En realidad, el objetivo final siempre es comerse a Pacman. Además, los agentes basados en objetivos. No son agentes reactivos si no agentes “deliberativos”. Crean un plan para conseguir sus metas. Para implementarlos se necesita planificación o algún tipo de búsqueda de los pasos que deben ejecutarse para conseguir la meta. Por lo que si fuera deliberativo basado en objetivos, entonces no se podría resolver con una máquina de estados.

2. Suponiendo que hay 4 fantasmas y que el objetivo de todos ellos es comerse a Pac-Man ¿Qué tipo de sistema multi agente es en base a su colaboración? Razona la respuesta. (0.5 p)

Es un sistema multiagente de colaboración con Independencia ya que los objetivos son compatibles (hay que comerse a Pacman y vale con que cualquiera de ellos se lo coma) la habilidad del agente es suficiente y los recursos que disponen son suficientes. Podríamos argumentar obstrucción si consideramos que la fruta es un recurso escaso, pero es un resultado muy rebuscado.

3. Formaliza el agente usando REAS (0,5p)

Sensores:

- Detector de fruta cerca: si hay una fruta a menos de 8 casillas avisa al agente.
- Detector del jugador: El agente sabe en todo momento la posición del jugador

- Jugador invulnerable: El Agente sabe si el jugador es invulnerable o no en todo momento.
- Choque de pared: El agente detecta una pared solo si se choca con ella.

Variables:

- Modo Fruta: booleano.

Actuadores:

- Posición del Agente: Vector2
- Dirección: Vector2
- Velocidad: real.

Medida de rendimiento: Veces que he comido el jugador. Podríamos tener en cuenta también otros criterios en caso de empate, como por ejemplo el número de bolitas que haya comido el jugador (cuantas menos mejor) o el número de veces que ha sido comido el fantasma.

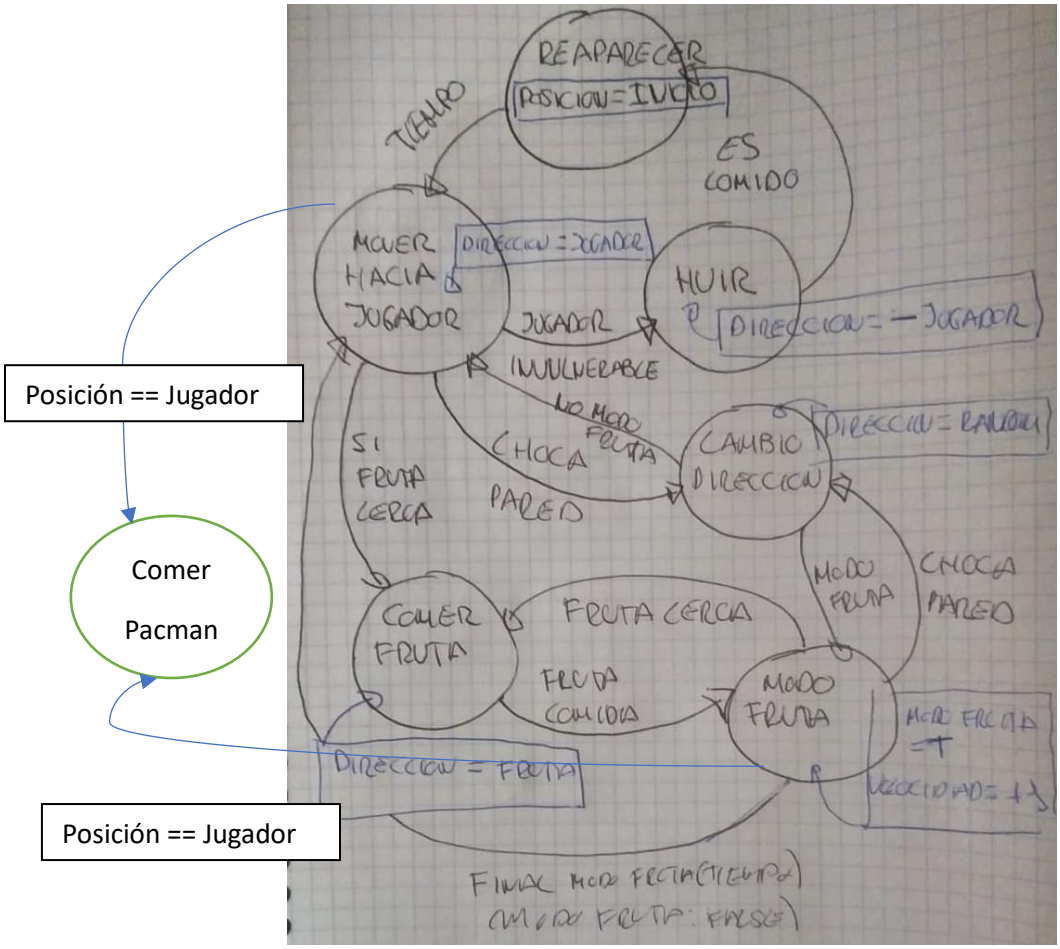
Entorno:

- Jugador
- Frutas
- Bolas grandes que hacen al jugador invulnerable
- Paredes
- Posición de inicio del agente
- Otros fantasmas
- Para el agente las bolitas normales que come Pacman a priori no son relevantes.

4. Modela el agente con el formalismo que consideres más adecuado (1 p)

Usaremos el formalismo de máquina de estados para modelar el agente.

Nombre del Alumno:



Problema 2 (2.5 p)

Queremos configurar un procesador de tipo System On A Chip (SOC) de forma modular para un dispositivo de alto rendimiento multimedia y juegos. Estos procesadores incluyen la mayoría de los componentes que tiene un computador moderno. Disponemos de un área máxima disponible de 8 x 8 casillas (64 casillas) para colocar estos componentes que son los siguientes:

- CPU: Ocupa 2x2 = 4 casillas
- Interfaz de memoria: Ocupa 3 x 1 = 3 casillas
- Controladora de IO: Ocupa 1 x 1 = 1 casilla
- GPU: Ocupa 3x3 = 9 casillas
- Memoria Cache L3 compartida entre CPU y GPU ocupa 1x2 = 2 casillas

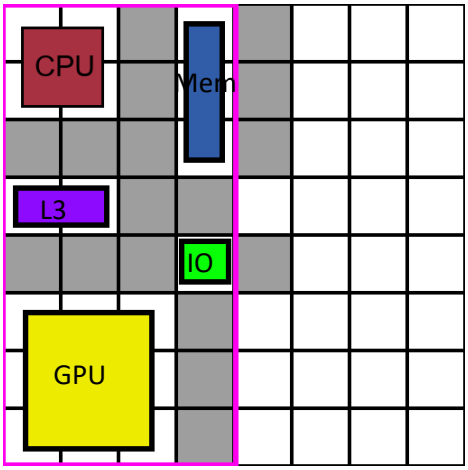
Pero hay que cumplir ciertas restricciones:

- La memoria cache L3 es deseable que esté lo más cerca posible de CPU y GPU.
- Tiene que haber una casilla de separación entre cada componente para poder colocar el bus de conexión entre ellos y para que el calor que genere cada componente no se propague al resto. Las diagonales no cuentan, solo en horizontal y vertical.

El objetivo es minimizar el área ocupada efectiva para poder construir procesadores lo más pequeños posibles y así poder obtener más chips por oblea. El área efectiva **es el área**

Nombre del Alumno:

rectangular mínima que engloba a todos los componentes del SOC. En el ejemplo esta representado como un cuadrado rosa. **Los componentes pueden colocarse en horizontal o en vertical pero no en diagonal.**



Se pide:

1. Elegir uno de los métodos vistos en clase para resolver el problema y explica razonadamente por qué lo has elegido (1p)

El formalismo que mejor encaja sería algoritmos genéticos porque es un problema de optimización, pero también podría plantearse como un algoritmo de búsqueda. Si lo planteamos como algoritmo de búsqueda sería:

- Estado inicial: la matriz vacía
- Estado final: la matriz con todos los elementos
- Operadores: colocar CPU, GPU, L3, etc. cada uno tendría un coste que sería su área más el espacio alrededor suyo sin contar las diagonales.

2. Modelar el problema usando el método o algoritmo seleccionado. Recordad que, en función del tipo de algoritmo, se necesita una modelización diferente. (1.5P)

Modelado usando genéticos:

La representación de los individuos sería la posición (X, Y) de cada una de las piezas, además de la rotación en el caso de la de L3 y Mem (0 en horizontal y 1 en vertical). Habría que fijar el punto donde se establece la posición en cada una de ellas. Por ejemplo, en la esquina superior izquierda.

CPU{X,Y}	GPU{X,Y}	IO{X,Y}	MEM{X,Y,O}	L3{X,Y,O}

El ejemplo del enunciado sería:

[0,0] [0,5] [3,4][3,0,1] 0,3,0]

O se hace una codificación en binario o si se mantiene una codificación decimal, entonces hay que tener en cuenta que la mutación de los genes que representan la dirección solo puede tomar los valores 0-1. En los cruces no hay que descolocar los alelos para evitar problemas ya que no todos significan lo mismo.

¿Cuál es la función de fitness? pues habría que comprobar el área que engloba cada pieza, por ejemplo, marcándola en la matriz y después quedándonos con los valores más altos y bajos de x e y, y calcular su resta para saber el tamaño del rectángulo en horizontal y vertical. Después aplicando el área de un rectángulo podremos saber el área ocupada por los componentes de la gpu. Para premiar que la cache L3 se sitúe a 1 casilla de distancia de la GPU y CPU (que es lo mínimo permitido) aumentaremos el fitness de aquellos que tengan la Cache L3 a más distancia de 1 de forma proporcional a la distancia a ambos componentes. Otra opción es usar este criterio como desempate. Priorizando el área mínima primero.

¿Qué pasa si generamos ejemplos inválidos?

Debemos gestionar que sucede si colocamos en la mutación o en el cruce a dos elementos que no pueden ser colocados. Hay diferentes alternativas, por un lado

- Podemos penalizar con un fitness muy bajo a los individuos inválidos para que así no sean seleccionados por el algoritmo
- Podemos intentar que en la mutación no se generen valores inválidos si llevamos la cuenta de en qué casillas se permite la colocación de los chips y en el caso de elegir un valor inadecuado (no quepa el chip en esa área o caiga en una casilla contigua) repetir la elección hasta que quepa en alguna ubicación.

Otra representación posible seria numerar las casillas de la 1 a la 64 e indicar la casilla en la que va la pieza correspondiente.

Representaciones como usar la matriz de 64 casillas para representar un individuo o usar todas las casillas que ocupan los chips (4 en la cpu por ejemplo) para representar el individuo, tiene como problema un mayor número de individuos inválidos. Podríamos generar en el último caso, individuos que tengan una forma no adecuada al componente o en el caso de usar como individuo la matriz de 64 casillas, poner 2 cpus. Con la codificación elegida esto no sucede. Aun así, no parece que exista una codificación más o menos obvia que no de ningún tipo de error.

Problema 3 (2.5 p)

Se desea implementar un sistema de predicción de **notas de examen** para la asignatura de Sistemas Inteligentes y para ello se dispone de un histórico con valores de 400 alumnos en un dataset. Los atributos almacenados por cada alumno son:

- Edad: entero de 18 a 100
- Género: Masculino, Femenino, Otros.
- Nota práctica 1: real de 0 a 10. Peso en la nota de evaluación continua de 0.1
- Nota práctica 2: real de 0 a 10. Peso en la nota de evaluación continua de 0.1
- Nota práctica 3: real de 0 a 10. Peso en la nota de evaluación continua de 0.1
- Nota práctica 4 grupal: real de 0 a 10. Peso en la nota de evaluación continua de 0.6
- Nota práctica 5: real de 0 a 10. Peso en la nota final de ordinaria de 0.1
- Porcentaje de asistencia a clase: real de 0 a 1.

Nombre del Alumno:

- Número de dudas preguntadas en clase: entero mayor o igual que 0.

La idea es predecir el resultado **de la nota del examen** con valores numérico (2.5, 6, 7.25) o bien con valores discretos (suspenso, aprobado, notable, sobresaliente, matrícula de honor)

Se pide:

1. ¿Qué trasformaciones harías a los datos para poder procesarlo con una Red de neuronas y para maximizar sus resultados? Razona la respuesta (0.5p)

Normalizar todos los valores a 0-1 o bien usando una normalización restando la media y dividiendo por la desviación típica o bien teniendo en cuenta los rangos. En el caso del género lo representaría en binario con One-Hot-encoding. También es correcto eliminar alguna de las entradas por considerarlas poco representativas (como el género)

2. Si usamos un perceptrón multicapa. ¿Qué configuración de capas de entrada, capas de salida y función de activación elegirías si quisiéramos predecir la nota con valores discretos? (0.5p)

Si usamos one-hot-encoding tendríamos 3 variables para el género. Más las 8 entradas nos daría un número de neuronas de entrada de 11. El número de neuronas de salida para predecir la nota numérica sería de 5, cada salida representaría una clase y nos quedaríamos con la clase con un valor mayor. La función de activación de las capas ocultas podría ser la que quisiéramos. Para la capa de salida podría ser softmax (nos daría la probabilidad de suspender). Si quitas el género nos quedarían 8 valores de entrada.

3. Define una medida de similitud para el algoritmo KNN que sea adecuada para este problema. (0.5p)

Usaría la distancia euclídea ponderada por los valores del peso de las notas (pero cualquier otra medida podría valer, siempre que sea ponderada). Quitaría el género porque no creo que aportase valor. Si no lo quitase le daría menos peso que las notas. Por ejemplo, la suma de las notas sería un 60% y el resto (edad, género, % de asistencia, dudas en clase) los daría un 10%. Del 60% calcularía el peso de cada práctica para calcular su % de contribución a la distancia.

4. Queremos implementar un sistema apoyado por un modelo de machine learning donde se realicen tutorías extra a los alumnos con mayor riesgo de suspender la asignatura. Se han evaluado los modelos usando cross-validation con K = 4 y un 25% de validación. No se dispone de muchos profesores así que el modelo debe ser suficientemente preciso para no malgastarlos.

KNN=0.7	Aprobados reales	Suspensos reales
Aprobados Modelo	40	10
Suspensos Modelo	20	30

Perceptrón Multicapa=0.8	Aprobados reales	Suspensos reales
Aprobados Modelo	55	15
Suspensos Modelo	5	25

Árbol de Decisión=0.8	Aprobados reales	Suspensos reales
Aprobados Modelo	50	10
Suspensos Modelo	10	30

Viendo estos resultados, con que modelo te quedarías. Razona la respuesta (1 p)

Elegiría Árbol de decisión porque es la que más alumnos suspensos cubre y menos profesores me hace desperdiciar. KNN me hace poner profesor a 20 alumnos que habrían aprobado el examen final, mientras que el árbol solo pondría a 10 profesores. El perceptrón tiene más acierto en los aprobados, pero falla más en los suspensos, por lo que dejaríamos a más alumnos suspensos sin profesor de apoyo, a pesar de que desperdiciaríamos a menos profesores con alumnos que aprobarían igualmente.

Problema 04 (2,5p)

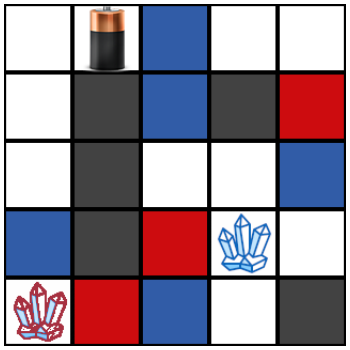
Queremos utilizar A* para resolver el siguiente problema: La Agencia Espacial Europea ha diseñado un robot que es capaz de tomar muestras de la superficie de la luna y hacer pruebas online para luego enviarlas a la tierra. Pero para llevarlo a cabo necesita tener energía en sus baterías. Inicialmente el robot parte con 20 unidades de energía y puede realizar las siguientes acciones con su coste de energía asociado:

- Moverse 1 metro en cualquier dirección de izquierda, derecha, arriba o abajo (coste base 1)
- Tomar una muestra (coste base 2 unidades)
- Analizar la muestra (coste 1 unidad)
- Enviar los datos a la tierra (coste 3 unidades)

En función del terreno el coste base de movimiento puede ser mayor, ya que el terreno lunar es muy accidentado.

Partimos de una configuración del robot de como decimos 20 unidades de energía y queremos conseguir que el robot analice una roca situada donde indica el mapa que se muestra a continuación. La meta del robot es enviar la información y volver a la estación de recarga con la mayor cantidad de energía posible. En el gráfico podemos ver los siguiente:

- La posición de la batería de recarga representada como una pila.
- La posición de posibles minerales a recoger. Los Azules tienen el coste de extracción base y los rojos se le suma una unidad más de energía a su coste base.
- Casillas negras que determinan regiones por donde el robot no puede moverse
- Casillas azules que implican un coste de 2 unidades de energía al moverte por ellas.
- Casillas rojas que implican un coste de 3 unidades de energía al moverte por ellas.



La idea es resolver cualquier problema de este tipo, pero el ejemplo que aportamos sirve para ilustrar el problema.

Se pide:

Nombre del Alumno:

1. Modela el problema como un problema de búsqueda definiendo el estado inicial, las metas, la función de coste y los operadores. (1p)

Estado inicial:

- Posición del robot: en la batería
- Muestra enviada: NO
- Muestra recogida: NO
- Muestra analizada: NO

Meta:

- Posición del Robot: en la batería
- Muestra enviada: SI
- Muestra recogida: SI
- Muestra analizada: SI

Operadores:

Mover: Cambia la posición del robot. El coste depende de la casilla a la que se mueva. En realidad son 4 operadores, mover arriba, mover abajo, mover izquierda y mover derecha.

Tomar Muestra: Muestra Enviada = SI. El coste depende del tipo de mineral. Solo puede ejecutarse si la posición del robot = a la del mineral.

Analizar Muestra: Muestra analizada: SI, la precondición de esta acción es que la muestra esté recogida

Enviar Muestra: Muestra enviada: SI, la precondición de esta acción es que la muestra esté analizada

Podemos agrupar las tres acciones de recoger, analizar y enviar la muestra en una sola ya que son consecutivas.

Si nos quedamos sin energía por un camino, esa rama no generará ningún sucesor más.

2. Define una función heurística admisible. La puntuación será mejor cuanto más informada sea. (1p)

La función heurística será la distancia de manhattan hacia el mineral más cercano que tenga el agente en un momento dado, más la distancia de manhattan de ese mineral a la batería (hay que volver) más el coste de procesar el mineral mínimo (Es decir asumir que el mineral escogido sea siempre azul) que será de 2+1+3 = 6. También valdría la euclídea, pero es menos informada.

Una heurística mejor sería elegir de entre las dos posible no solo la más cercana si no la que menor coste acumulado tenga. Si sabemos si el mineral es rojo o azul, entonces podemos elegir de entre los dos minerales, el azul si está más cerca o a igual distancia y solo si está más cerca el rojo elegir el rojo ya que procesar el rojo cuesta una unidad más.

Nombre del Alumno:

3. Si al ejecutar el algoritmo vemos que el tiempo que tarda es demasiado largo y no encuentra una solución en un tiempo razonable. ¿Qué harías para conseguir resolver el problema? Razona la respuesta (0.5p)

Podemos intentar hacer una heurística no admisible, por ejemplo, asumir que es mejor elegir el mineral más prometedor, es decir el que tenga menos coste y distancia como objetivo siempre. Esto probablemente pueda darnos soluciones no optimas, pero reducirá la expansión de nodos.

Otra alternativa es usar búsqueda local heurística como escalada que tampoco garantiza la solución pero que tiene un coste menor o cualquier otro algoritmo de búsqueda salvo la búsqueda a ciegas por ser un paso atrás muy evidente sobre A*. Por ejemplo Genéticos o Q-Learning o cualquiera de los algoritmos que no hemos visto en profundidad en clase pero que están en la bibliografía de la asignatura.