# Inteligencia Artificial 2024

Tercer Proyecto

01.mayo.2024

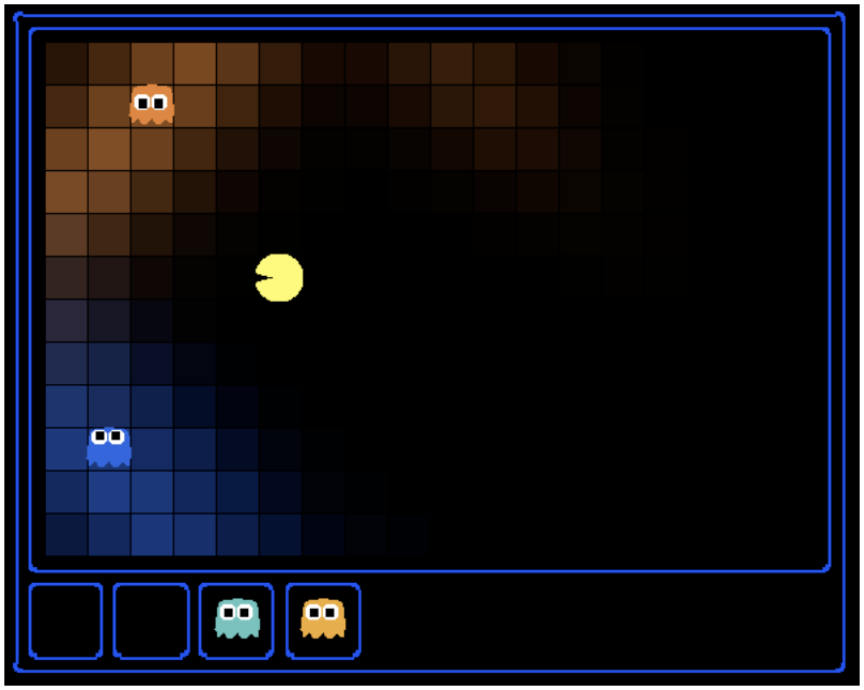
Antes de comenzar el desarrollo del proyecto conviene recordar la documentación general del proyecto, así como la guía y recomendaciones: `http://ai.berkeley.edu/project_overview.html`.
`http://ai.berkeley.edu/project_instructions.html`.

**Tercer Proyecto: Ghostbusters**.
PacMan se pasa la vida huyendo de fantasmas, pero las cosas no siempre fueron así. Cuenta la leyenda que hace muchos años, el bisabuelo de PacMan, Grandpac, aprendió a cazar fantasmas por deporte. Sin embargo, estaba cegado por su poder y sólo podía rastrear a los fantasmas por sus golpes y ruidos metálicos.
En este proyecto, diseñarás agentes PacMan que utilizan sensores para localizar y comerse fantasmas invisibles. Pasarás de localizar fantasmas individuales e inmóviles a cazar manadas de múltiples fantasmas en movimiento con una eficiencia despiadada.
De nuevo, mi sugerencia es que conviene comenzar una instalación nueva, en lugar de mezclar archivos del proyecto 1 o proyecto 2.
Al igual que en el proyecto 1, este proyecto incluye un *autograder* para que usted pueda calificar sus respuestas en su máquina. No se usará este autocalificador para efectos de evaluación (úselo únicamente a modo de test y verificación de sus algoritmos).

Archivos a editar:

- `bustersAgents.py` Archivo donde se guarda la información de los agentes para jugar la variante *Ghostbusters* de PacMan.
- `inference.py` Archivo donde reside la clase de inferencia y los algoritmos probabilísticos que se usarán para rastrear a los fantasmas.

Revisar el sitio del proyecto *Ghostbusters*, `http://ai.berkeley.edu/tracking.html`
Asegurarse de leer las especificaciones y los *hints* incluidos en el proyecto.



1. **Problema 1**: Inferencia Exacta (Q1 del proyecto de Berkeley).

   En esta pregunta se debe actualizar el método de observación en la clase **ExactInference** de `inference.py` para actualizar correctamente la distribución de creencias del agente sobre las posiciones de los fantasmas dada una observación de los sensores de Pacman.

   Una implementación correcta también debería manejar un caso especial: cuando un fantasma es devorado, debes colocarlo en su celda de prisión, como se describe en los comentarios de observar.

   Para ejecutar el autocalificador para esta pregunta y visualizar el resultado:
   `python autograder.py -q q1`

   Mientras se observan los casos de prueba, asegúrese de comprender cómo convergen los cuadrados hasta su coloración final. En los casos de prueba en los que PacMan está encajonado (es decir, no puede cambiar su punto de observación), ¿por qué PacMan a veces tiene problemas para encontrar la ubicación exacta del fantasma?

2. **Problema 2**: Inferencia Exacta con Tiempo Transcurrido (Q2 del proyecto de Berkeley).

   En la pregunta anterior implementaste actualizaciones de creencias para PacMan basadas en sus observaciones. Afortunadamente, las observaciones de PacMan no son su única fuente de conocimiento sobre dónde puede estar un fantasma. PacMan también tiene conocimiento sobre las formas en que puede moverse un fantasma; es decir, que el fantasma no puede atravesar una pared o más de un espacio en un solo paso de tiempo.

   Para entender por qué esto es útil para PacMan, considere el siguiente escenario en el que hay PacMan y un Fantasma. PacMan recibe muchas observaciones que indican que el fantasma está muy cerca, pero luego una que indica que el fantasma está muy lejos. Es probable que la lectura que indica que el fantasma está muy lejos sea el resultado de un sensor defectuoso. El conocimiento previo de PacMan sobre cómo se puede mover el fantasma disminuirá el impacto de esta lectura, ya que PacMan sabe que el fantasma no podría moverse tan lejos con un solo movimiento.

   En esta pregunta, implementará el método **elapseTime** en **ExactInference**. Su agente tiene acceso a la distribución de acciones de cualquier GhostAgent. Para probar su implementación de elapseTime por separado de su implementación de observación en la pregunta anterior, esta pregunta no utilizará su implementación de observación.

   Dado que PacMan no utiliza ninguna observación sobre el fantasma, esto significa que PacMan comenzará con una distribución uniforme en todos los espacios y luego actualizará sus creencias de acuerdo con cómo sabe que el Fantasma puede moverse. Dado que PacMan no observa al fantasma, esto significa que las acciones del fantasma no afectarán las creencias de PacMan. Con el tiempo, las creencias de PacMan llegarán a reflejar lugares del tablero donde él cree que es más probable que a los fantasmas se les dé la geometría del tablero y lo que PacMan ya sabe sobre sus movimientos válidos.

   Para las pruebas de esta pregunta, a veces usaremos un fantasma con movimientos aleatorios y otras veces usaremos GoSouthGhost. Este fantasma tiende a moverse hacia el sur, por lo que con el tiempo y sin ninguna observación, la distribución de creencias de PacMan debería comenzar a centrarse en la parte inferior del tablero. Para ver qué fantasma se utiliza para cada caso de prueba, puede buscar en los archivos .test.

   Para ejecutar el autograder para esta pregunta y visualizar el resultado:

   `python autograder.py -q q2`

   Como ejemplo de GoSouthGhostAgent, puede ejecutar

   `python autograder.py -t test_cases/q2/2-ExactElapse`
   y observe que la distribución se concentra en la parte inferior del tablero.

   Mientras observa el resultado del autograder, recuerde que los cuadrados más claros indican que PacMan cree que es más probable que un fantasma ocupe esa ubicación, y los cuadrados más oscuros indican que es menos probable que un fantasma ocupe esa ubicación. ¿En cuál de los casos de prueba observa que surgen diferencias en el sombreado de los cuadrados? ¿Puedes explicar por qué algunos cuadrados se vuelven más claros y otros más oscuros?

3. **Problema 3**: Inferencia Exacta Full Test (Q3 del proyecto de Berkeley).

   En esta pregunta utilizará sus implementaciones de **observe** y **elapseTime** juntas, junto con una estrategia de caza *greedy* simple que implementará para esta pregunta. En la estrategia *greedy* y simple, PacMan asume que cada fantasma está en su posición más probable según sus creencias, luego se mueve hacia el fantasma más cercano. Hasta este punto, PacMan se ha movido seleccionando aleatoriamente una acción válida.

   Implemente el método **ChooseAction** de la clase **GreedyBustersAgent** en `bustersAgents.py`. Tu agente primero debe encontrar la posición más probable de cada fantasma restante (no capturado) y luego elegir una acción que minimice la distancia hasta el fantasma más cercano. Si se implementa correctamente, su agente debería ganar el juego en `q3/3-gameScoreTest` con una puntuación superior a 700 al menos 8 de cada 10 veces. Nota: el autograder también verificará directamente la exactitud de su inferencia, pero el resultado de los juegos es una verificación de cordura razonable.

   Para ejecutar el autograder para esta pregunta y visualizar el resultado:
   `python autograder.py -q q3`

   Nota: Si desea ejecutar esta prueba (o cualquiera de las otras pruebas) sin gráficos, puede agregar el siguiente flag:
   `python autograder.py -q q3 --no-graphics`

4. **Problema 4**: Inferencia Aproximada y Filtro de Partículas (Q4 del proyecto de Berkeley).

   A continuación, implementarás un algoritmo de filtrado de partículas para rastrear un solo fantasma.

   Implemente las funciones **initializeUniformly**, **getBeliefDistribution** y **observe** la clase **ParticleFilter** en `inference.py`. Una implementación correcta también debería manejar dos casos especiales. (1) Cuando todas sus partículas reciban peso cero según la evidencia, debe volver a tomar muestras de todas las partículas anteriores para recuperarlas. (2) Cuando se come un fantasma, debes actualizar todas las partículas para colocar ese fantasma en su celda de prisión, como se describe en los comentarios de la observación. Cuando esté completo, debería poder rastrear fantasmas casi con tanta eficacia como con la inferencia exacta.

   Para ejecutar el autocalificador para esta pregunta y visualizar el resultado:
   `python autograder.py -q q4`

---

**Evaluación : 5 puntos cada uno de los cuatro problemas, para un total de 20.**
**Fecha de Entrega: semana del 27 del mayo al 1 de junio.**