

# Inteligencia Artificial 1

Este es el planteo del proyecto de curso necesario para aprobar el curso de *Inteligencia Artificial 1* del primer semestre de 2019, para las carreras de Licenciatura e Ingeniería en Informática de la UCU, a cargo de los docentes Agustín Castillo y Leonardo Val.

El proyecto de curso es un trabajo obligatorio a realizarse en equipo, dentro y fuera de clase.

## Planteo

El proyecto consiste en implementar un jugador artificial para un juego estrategia por turnos sobre un tablero.

El juego se realiza sobre un damero de 5x9 donde cada uno de los 2 jugadores tiene 3 piezas de 3 tipos distintos cada una. Las formas de movimiento y formas de ataque viene dados por el tipo de pieza. También sobre el tablero se encuentran obstáculos que limitan el movimiento de las piezas.

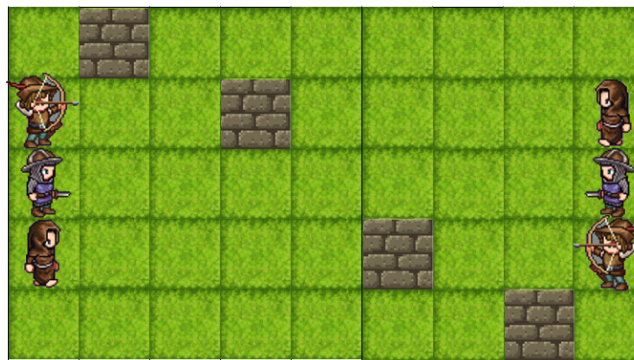




Figure 1: mapa

En cada turno el jugador puede realizar una de las acciones posibles con una de sus piezas. Las acciones implican movimientos dentro del tablero y/o ataques a piezas enemigas. Los movimientos siempre se realizan antes de los ataques, de hacerse. Los movimientos siempre se realizan de forma ortogonal, es decir horizontal o verticalmente. Las piezas siempre deben mover a casillas que no estén bloqueadas.

Las piezas tienen una cantidad de puntos de vida (**hp**) asociados. Cuando una pieza ataca a otra, le hace una cantidad de daño que es descontada de los puntos de vida de la pieza objetivo. Cuando la pieza pierde todos sus puntos de vida, es removida del tablero. Un jugador pierde cuando queda sin piezas. Cuando un jugador pierde, el otro jugador gana.

Los tipos de piezas y sus reglas son los siguientes. Muchas propiedades de las piezas y sus acciones se dan definidas como parámetros (e.g. **k\_hp**), cuyos valores deberán ser asignados por la implementación del juego.

-  *Knight*: El caballero tiene **k\_hp** puntos de vida inicialmente. Puede mover hasta 2 casillas por turno.
  - Si se mueve 2 casillas en línea, puede atacarse a un enemigo en la tercera casilla alineada, con un daño de **k\_dmg\_1**.
  - Con cualquier otro movimiento, puede atacar a una de las piezas enemigas adyacentes ortogonalmente con daño **k\_dmg\_2**.
  - Si no mueve, puede atacar a todas las piezas enemigas adyacentes ortogonalmente con daño **k\_dmg\_3**.
-  *Mage*: El mago tiene **m\_hp** puntos de vida inicialmente. Puede mover hasta 3 casillas por turno.
  - Si mueve, cura a todas las piezas aliadas adyacentes ortogonalmente por **m\_dmg\_1** puntos de vida. La cantidad de puntos de vida nunca debe superar la cantidad de puntos de vida inicial de cada pieza. El mago no se cura a sí mismo.

- Si no mueve, puede atacar a una pieza oponente que esté a una distancia de Manhattan mayor de  $m\_dist\_1$  con un daño  $m\_dmg\_2$ .



- **Archer:** El arquero tiene  $a\_hp$  puntos de vida inicialmente. Puede mover hasta 3 casillas por turno.
  - Siempre puede atacar a todas las piezas oponentes visibles que estén en la misma diagonal u ortogonal, a una distancia mayor que  $a\_dist\_2$ .
  - Si no mueve, el daño de su ataque es de  $a\_dmg\_1$ .
  - Si mueve, el daño de su ataque es de  $a\_dmg\_2$ .

Además de implementar la lógica del juego y los jugadores artificiales, también deben encontrarse los valores apropiados para los 12 parámetros del juego (puntos de vida, puntos de daño, y distancias). Los parámetros y sus dominios son:

- **Knight:**  $k\_hp \in [1-30]$ ,  $k\_dmg\_1 \in [1-30]$ ,  $k\_dmg\_2 \in [1-30]$ ,  $k\_dmg\_3 \in [1-30]$ .
- **Mage:**  $m\_hp \in [1-30]$ ,  $m\_dmg\_1 \in [1-30]$ ,  $m\_dmg\_2 \in [1-30]$ ,  $m\_dist\_1 \in [1-9]$ .
- **Archer:**  $a\_hp \in [1-30]$ ,  $a\_dmg\_1 \in [1-30]$ ,  $a\_dmg\_2 \in [1-30]$ ,  $a\_dist\_2 \in [1-9]$ .

Los valores para dichos parámetros se deben optimizar de forma tal de lograr los siguientes objetivos:

1. La duración promedio de las partidas sea de 18 turnos.
2. El juego este nivelado, osea que ambos jugadores tengan una probabilidad similar de ganar.

## Entrega

La entrega se compone de lo siguiente:

1. Implementación en Python de la lógica del juego utilizando la última versión disponible del framework provisto por la cátedra.
2. Un componente que optimice los parámetros del juego, utilizando alguna de las metodologías vistas en el curso:
  - a. *Hillclimbing* o *simulated annealing*.
  - b. *Computación evolutiva*.
3. Documentación detallando análisis y diseño de la solución, problemas encontrados, decisiones y aclaraciones que se crean pertinentes. No se espera que incluyan en la documentación explicaciones sobre los algoritmos utilizados que ya estén dadas en la bibliografía del curso.

## Cronograma

La entrega podrá realizarse hasta el miércoles 3 de Julio a las 23 horas, vía Webasignatura. La entrega debe incluir:

- todo el código fuente programado y necesario para ejecutar el juego y la optimización,
- la bibliografía utilizada,
- un breve reporte (formato PDF) documentando el trabajo realizado,
- opcionalmente *logs* de ejecuciones que consideren interesantes.

La entrega *no* debe incluir:

- código compilado (e.g. archivos `.pyc`),
- archivos temporales,
- archivos de manejo de proyecto, de IDEs o del entorno del sistema operativo.

Las entregas cuyas pruebas no puedan reproducirse (por el motivo que sea) serán calificadas con 0 puntos. El respeto a las normas de entrega forma parte de la calificación final.

## Referencias

Arte tomado de [opengameart.org](https://opengameart.org) bajo licencia CC-BY-SA 3.0.