Trabajo Práctico N°2 de Inteligencia Artificial I

Prof. Martín Sebastián Wain
Version 1.04

Introducción

“Cansadas del yugo opresor de los granjeros y sus agentes noctámbulos que le prohíben el sustento alimenticio, la rebelión de iguanas ha amasado un ejército pudiendo revertir el orden de dominio, avasallando con creces.
Algunos opinan que fue justificado, otros no comprenden del todo qué fue lo que sucedió. Lo que sí es seguro es que la única forma de sobrevivir para un granjero es en sigilo, robando unas pocas migas de gofres, o lamiendo el glaseado rancio de una dona descartada, en la oscuridad, dentro de las mismas fauces del demonio verde y su maquinaria fascista.”

Objetivo

El objetivo del trabajo práctico es que el alumno pueda aplicar los conceptos de grafos y su recorrido, búsqueda del camino más corto (pathfinding), generación procedimental de mapas y rutas de patrullaje, aplique comportamientos de dirección simples (steering behaviors) y entrene como crear una arquitectura para un juego del mundo real.
A su vez reforzará su conocimiento en máquinas de estados (FSM), filtrado por línea de visión (line of sight), aleatoriedad, álgebra y física vectorial, uso del lenguaje de programación C# y el funcionamiento del motor Unity.

Como encarar el proyecto

Primero léa este documento entero antes de comenzar a resolverlo.
Segundo planee todo lo que debe hacer **en papel**, que sucede en cada estado y transición, cuál es la mejor forma de manejar cómo seguir caminos de la iguana (dado que está el de patrulla y el temporal).
Si surge la duda de cómo encarar la planificación previa, preguntar al profesor en persona o por mail (no al grupo).
Luego, como recomendación, siga el orden del TP para su implementación, probando de a partes por separado pero nunca dejando la integración entre todas para último momento porque es la interacción entre sistemas la más propensa a generar errores.

Proyecto base

Para realizar este TP deberá descargar el proyecto base (39.4MB) del siguiente link: **https://bitbucket.org/2bam/w_ia1_tp2/downloads/** (“Download repository”)

O clonar el repositorio usando git:
git clone https://bitbucket.org/2bam/w_ia1_tp2.git

Criterio de evaluación

EL TRABAJO PRÁCTICO ES EN GRUPO DE DOS INTEGRANTES.
LOS GRUPOS SON ARMADOS POR EL DOCENTE.

Se tendrán en cuenta para la aprobación de este trabajo los siguientes ítems:

1. La correcta implementación y estructura.
2. Prolijidad y legibilidad del código fuente proporcionado, así como el orden del proyecto.
3. El cumplimiento de las pautas de presentación (entrega a término y formato).
4. Cumplir con los puntos de la consigna **como se pide** para sumar 10 puntos.
   No sea el Thelonious Monk del código, por favor.
5. Si aparte cumple los puntos marcados como "**Bonus**" tendrá más chances de alcanzar una mejor nota si no realiza los puntos comunes perfecto.

Pautas de presentación

La entrega se realiza por email a martin.wain@davinci.edu.ar.
Se entrega en un archivo comprimido (máxima compresión) con nombre:
IA1-TP<Número>_<Turno>_<Apellido[s]>.zip/.rar
El asunto del mail debe ser igual al nombre del archivo.
Solo incluir carpeta **Assets** y **ProjectSettings**. También incluir un archivo de texto con los nombres completos y e-mail de cada integrante.

Confirme que funciona el link al archivo comprimido descargandolo en una ventana incognito y abriéndolo de nuevo, no se aceptan prórrogas por desperfectos del mismo.

Fecha de entrega

Será pactada para la clase más cercana a 4 semanas después a la que se entrega el enunciado.
Dado que se entrega por internet (google drive y e-mail), la fecha es inamovible pese a feriados o eventualidades que impidan ir a clase.
Una entrega luego de esta fecha quedará reprobado.
Tenga en cuenta que el recuperatorio es integrador y tiene más consignas.

Consigna

Importantísimo

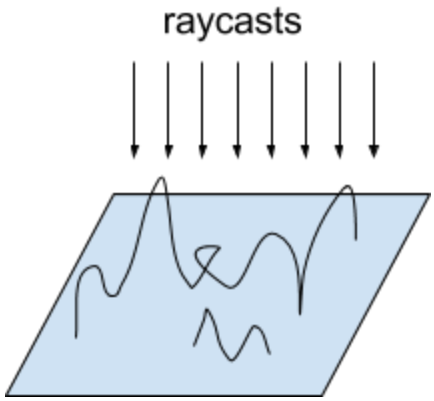
Cuando compare distancias y calcule movimientos en los personajes, recuerde hacerlo **sin el componente vertical "Y".** Mueva personajes exclusivamente usando los character controllers "SimpleMove()" (que descarta el "Y" de todos modos).
En varias partes del código se encuentran comentarios "`//ALUM:`" con indicios de lo que debe hacer.

Grafo del mapa

Clases: Debe escribir todo el siguiente comportamiento en las clases `Map` y `MapNode`, dentro del método `Configure` llamado por `GameManager`.

Nodos

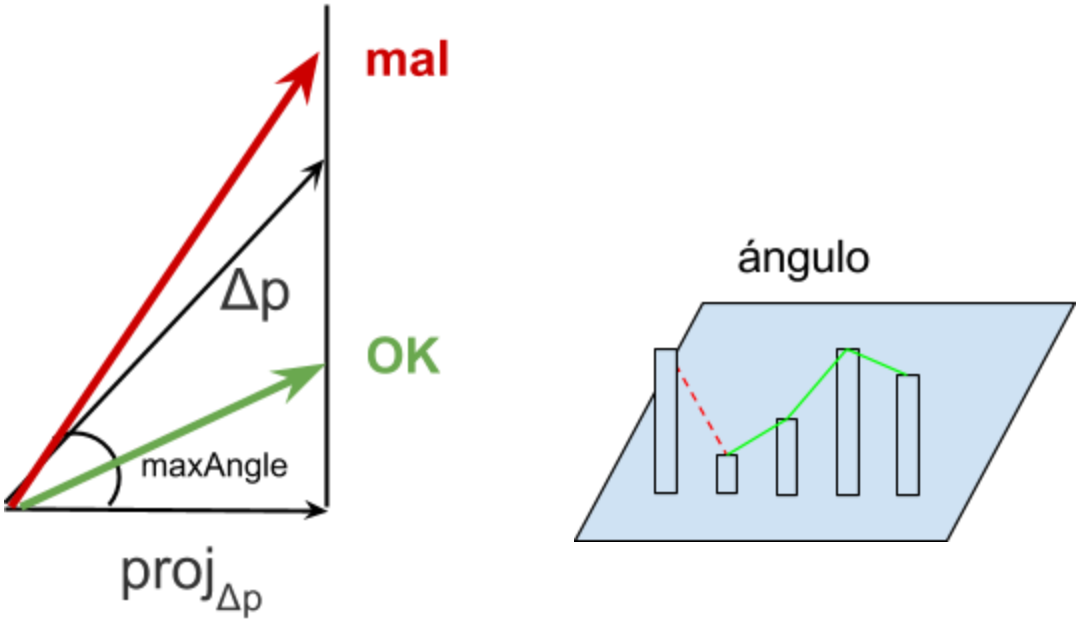
Se proporciona un terreno base (Terrain de Unity) con el cual tendrá que calcularse un grafo en forma de grilla en el plano XZ. El ancho, alto de la grilla como del tamaño de las celdas debe ser configurable (`startX, startZ, width, height, cellSize`).
Para lograrlo se debe realizar una "lluvia" de raycasts verticales desde el cielo (`raycastStartY`) hacia la tierra y determinar así en qué punto estaría cada nodo del grafo. Una vez que detecta la posición, súmele "nodeOffset".
Nótese que tanto el terreno como las paredes están ubicados en el layer "**Terrain**".
Pista: Busque qué acepción de la función Physics.Raycast toma origen, dirección y otorga información del punto de impacto.
Pista: Recuerde que existe float.MaxValue.



Aristas: Conexión de rutas

Luego para interconectar cada uno de esos puntos, se debe calcular si entre los adyacentes horizontal, vertical y diagonalmente no se supera cierto ángulo configurable (`maxAngle`) máximo respecto al plano del piso.

Para expandir utilice “vectores de movimiento”, pero de **8 direcciones**.



Para calcular este ángulo, piense que puede sacarlo entre

1. la diferencia entre un punto y otro (Δp)
2. y el vector proyectado sobre el plano XZ ($proj_{\Delta p}$)

Nota: Fijese de no repetir aristas.

Zonas inalcanzables (recorrido)

Aplicar un algoritmo de recorrido desde el nodo en [0,0] de la grilla (cerca de la casa del granjero), marque todos los nodos accesibles. El resto será inaccesible y no deberá permitir creación de iguanas ni creación de rutas de patrullaje que pasen por las mismas.

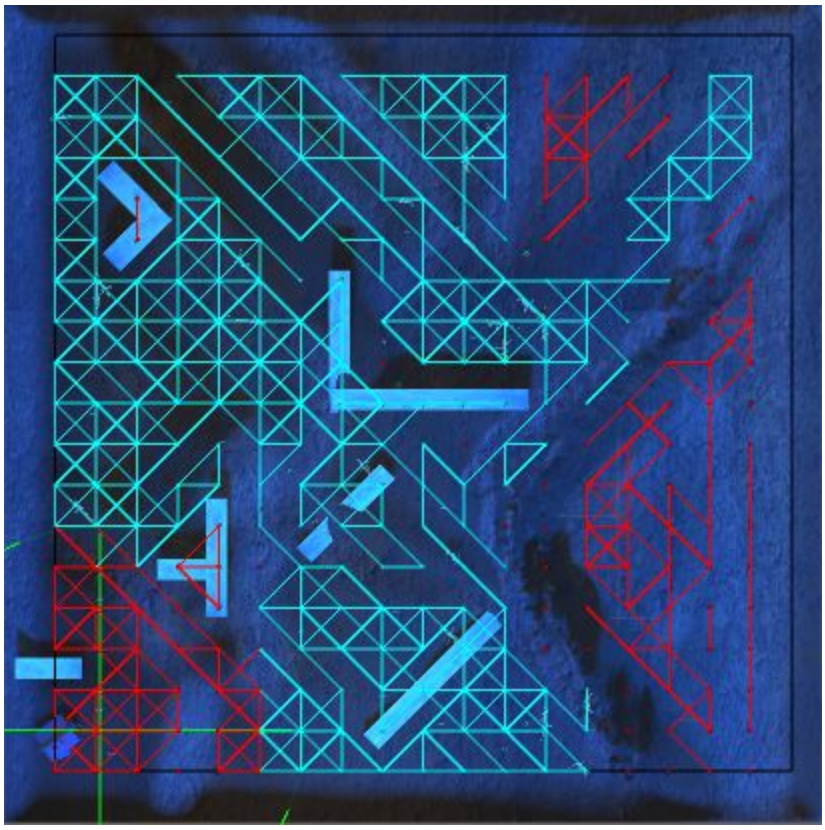
Zona segura (radio)

Marcar todos los nodos en un radio de la posición inicial del granjero como inaccesibles para evitar crear iguanas o rutas de patrullaje en los mismos (parámetro `safetyCenter` y campo `safetyRadius`)

Representación gráfica de debug (gizmos)

Implementar el dibujado de “gizmos” para representar las aristas del grafo a través de líneas y los nodos como pequeñas esferas de 0.25f en diámetro, todo en color CYAN. Si el nodo es **inalcanzable** debe dibujarlo al mismo como a sus aristas de color ROJO.

Si realizó todo correctamente se debería ver el mapa algo así:



Iguanas

Clases: `Iguana, GameManager, AStar`

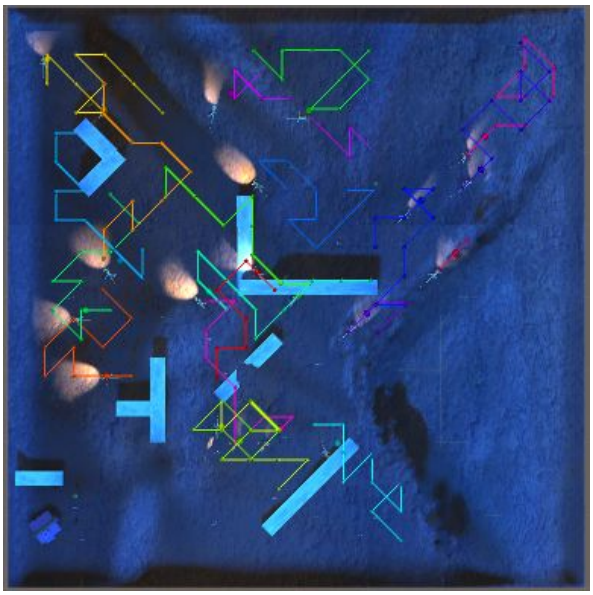
Creación

Deben aparecer en lugares al azar del mapa **siempre que estos sean accesibles**, pero lejos de un radio del player.
Si una iguana ve un player lo persigue mientras lo tenga en vista. Si oye un ruido debe buscarlo a través de los `MapNode`s hasta el último punto que lo oyó.
Si el player esta en la linea de visión debe largar un alarido y empezar a perseguirlo. Si lo pierde de vista, retoma su patrulla volviendo al último nodo en el que estaba desde el más cercano.
Hay tres rangos de iguana: 0. soldado, 1. coronel, 2. brigadier. Cada uno tiene asociado una probabilidad de aparecer en “`rankProbability`” y un casco asociado en el array

"`helmetPrefabs`". Su velocidad de movimiento y el nombre del game object son dados por el código ya incluído (no modificar). Agregar al game object "helmetPos".

Persecusión (Seeking)

La iguana debe perseguir al granjero mientras se encuentre en su campo visual.

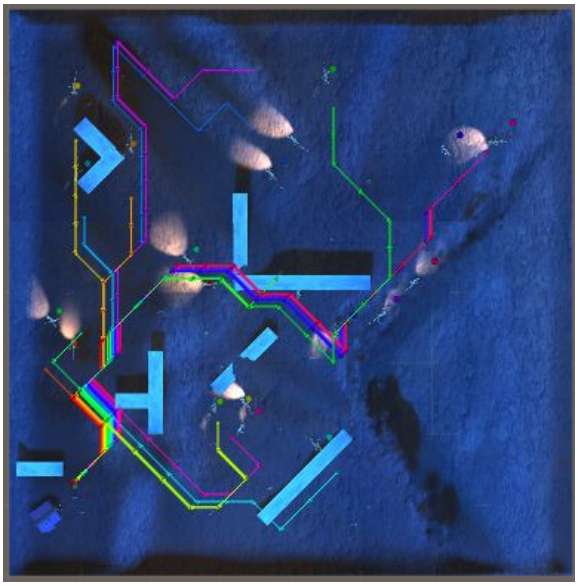


Ruta de patrullaje

Crear un camino lineal aleatorio de 10 nodos **accesibles**, pero sin repetir.
La iguana debe ir y volver por el mismo repetidamente.
Debe hacer todo lo posible por crear un camino de 10 nodos, aunque hay casos donde es inevitable (enroscarse en sí mismo y quedarse sin opciones).

Ruta temporal (ir a origen del ruido y su retorno)

Cuando se escucha un ruido, la iguana debe seguir una ruta temporal buscando el nodo más cercano a la posición del objeto que generó el ruido mediante el algoritmo A*.
La ruta de patrullaje **no debe perderse** dado que al llegar al destino temporal, se debe retornar a la misma mediante otro A* distinto (quizás en su recorrido vió al granjero y se salió de curso).
Solo en estos casos puede ir a nodos inaccesibles.



Retorno

Una vez que llego al destino del camino temporal o si deja de ver al granjero cuando estaba persiguiéndolo, debe retomar su ruta de patrullaje **desde el último nodo en el que dejó de patrullar y en la dirección que venía.**

Gizmos

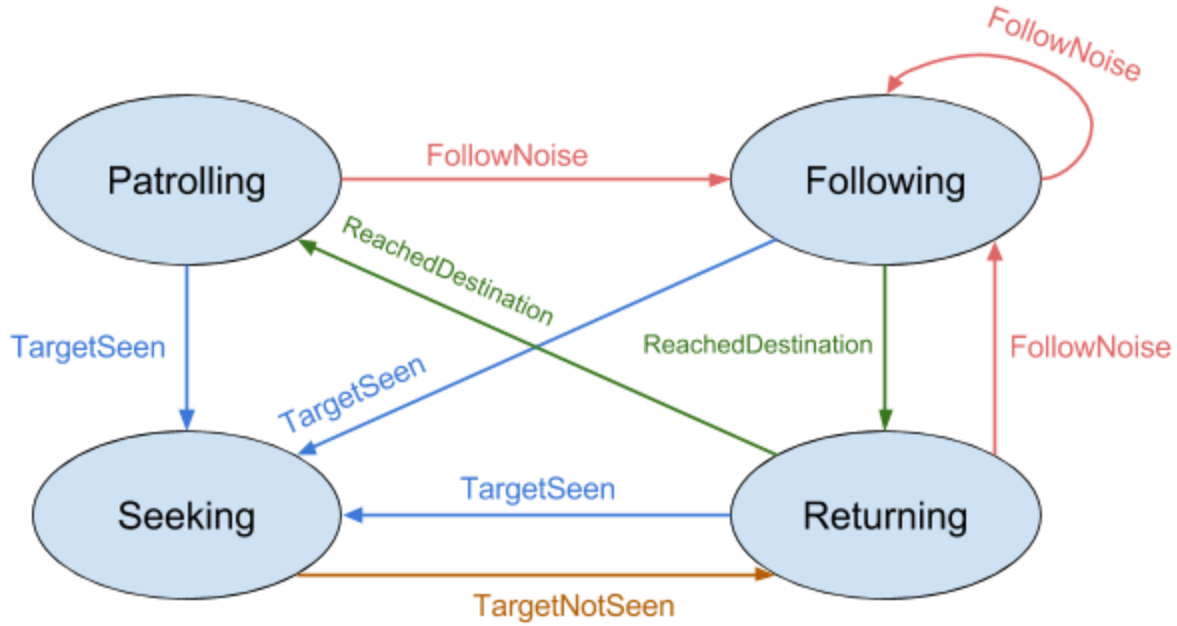
Utilizando el color _gizmosColor asignado en Configure, único por iguana, debe dibujar

1. Su ruta de patrullaje si está patrullando
2. o su ruta de seguimiento si está yendo hacia o volviendo de un sonido.

Estas deben ser esferas conectadas por líneas.

Estados de la iguana

Programe como más le guste (switch, state pattern, corrutina) la FSM de la iguana.



- **Patrolling**: Sigue su ruta de patrullaje generada, ida y vuelta.
- **Following**: Sigue un camino temporal específico hacia donde oyó un ruido (resultado de A*)
- **Returning**: Sigue un camino temporal específico hasta el último nodo en el que dejó de patrullar (resultado de A*)
- **Seeking**: Perseguir al objetivo del line-of-sight.

Entradas:

- **TargetSeen**: En el line-of-sight se ve al granjero.
- **TargetNotSeen**: En el line-of-sight **no** se ve al granjero.
- **ReachedDestination**: Se llegó al último punto del camino temporal.
- **FollowNoise**: Hay un nuevo camino temporal para seguir. Nótese que se puede “re-entrar” a Following con este, si se escucha un nuevo ruido.

Componente ruido

Clase: `NoiseSource`

Cuando el granjero colisiona con un objeto con componente `NoiseSource`, se genera un ruido aleatorio de sus clips alertando a todas las iguanas, que deberán ir a investigar. Todas las iguanas deberán converger sobre la posición originaria del ruido utilizando el algoritmo A*, cambiando temporalmente su ruta de patrullaje común por la calculada para llegar. Si bien las iguanas no deben patrullar dentro de la “zona segura”, si pueden ir a ella al escuchar un ruido.

Cosas que hacen ruido:

- Comer donas rancias
- Pisar ramas
- Una iguana al detectar al granjero (fuera del estado Seeking)

Bonus: Configurar un radio de efecto sonoro. Ejecutar un Dijkstra para la “difusión” de sonidos sobre el mismo mapa ya creado (para evitar que estos pasen por montañas o paredes) pero que vaya perdiendo intensidad a medida que el algoritmo expande. Debe utilizar el radio para calcular la intensidad proporcional que tiene cada nodo (cerca 100%, a medio radio 50%, en el perimetro del radio 0%), y por cada uno de esos nodos filtrar objetos que tenga en un radio de `Map.cellSize` utilizando cuadrados en vez de círculos para no dejar a nadie afuera.

Comida rancia

Clases: `Fly, RancidDonut`

Al ser consumidas (**Eat**) por un personaje, se desactivan y se deberá generar un ruido que llama la atención de todas las iguanas.
Pista: Una dona desactivada no puede generar sonidos.

Bonus: Crear 5 moscas (`Fly`) por cada dona rancia, que apliquen los steering behaviours “seek” y “wander” para merodear la misma.
Al comer la dona las moscas deben comenzar a seguir y merodear al granjero.

Granjero

Clase: `Farmer`

La parte de movimiento ya se encuentra incluída (teclas WASD). No la modifique.
Si un granjero toca una dona, debe comerla.
Si toca cualquier entidad que tiene NoiseSource, debe hacerlo sonar.
Si toca una iguana debe “morir” retornando a su posición inicial, pero dos unidades más arriba.

“Hágalo por ellos”

Referencia de clases/scripts

AStar

Clase para calcular el camino más corto utilizando A*.
Nótese que es una clase estática y para generar un camino se accede llamando a `AStar.Run(inicio, fin)`, y retorna un stack de nodos a del camino generado.

Farmer

Todo lo relacionado al granjero: Movimiento, colisiones, etc. se encuentra aquí.

Fly

Es una mosca que merodea un target usando steering behaviours seek y wander.

GameManager

El estado del juego en general, incluyendo la generación y mantenimiento de iguanas en posiciones aleatorias accesibles. Es dueño del mapa `Map`.
Es un **Singleton** y puede ser accedido a través de su campo estático público "**instance"**.

Iguana

Aquí va todo el código de las iguanas, incluyendo crear sus rutas de patrullaje aleatorias, seguirlas, su máquina de estados, etc.

LineOfSight

Incluido.
Componente para línea de visión. Simplificado para no usar layers, si no un único target.

Map

Esta clase contiene toda la generación y filtrado de accesibilidad del grafo del mapa.
Se puede acceder a los nodos del mismo mediante la propiedad public "grid".

MapNode

Es un nodo del mapa. Tiene posición, sus adyacencias y si es **accesible**.
Nótese que **no es** un `MonoBehavior`.

NoiseSource

Componente que genera un ruido asociado via "Play()" y debe alertar a todas las iguanas del mismo.

RancidDonut

Es una dona rancia que tiene 5 moscas. Cuando es consumida hace que las moscas persigan al que la consumió.

Utility

Aquí van funciones utilitarias que pueden ser utilizadas en varios lugares.