UNIVERSIDAD DE GRANADA

/ UGR / decsai

Departamento de Ciencias de la Computación e Inteligencia Artificial

Técnicas de los Sistemas Inteligentes

Curso Académico 2020-21

Práctica 1: Desarrollo de un agente basado en búsqueda heurística para el entorno GVGAI

La Práctica 1 consiste en desarrollar un controlador, basado en A* o alguna de sus variantes, dentro del entorno GVGAI[1] que guíe a un avatar a resolver un juego en distintos niveles. El juego escogido es el juego con índice 11 en los tipos de juego "singleplayer", que se pueden encontrar en el fichero "examples/all_games_sp.csv" de la distribución de GVG_AI, denominado Boulder Dash.

1. Descripción general del juego

Boulder Dash es un videojuego comercializado en 1984 para la familia de computadores Atari de 8 bits. En dicho juego, el protagonista/avatar debe cavar en una cueva con una pala, para encontrar al menos 9 diamantes, de entre un conjunto mayor de ellos inicialmente distribuidos en diferentes posiciones, antes de salir por una puerta (véase Figura de más abajo). Algunas rocas pesadas pueden caer mientras cava, matando al jugador si es golpeado desde arriba. Hay enemigos en la cueva que pueden matar al jugador, pero si dos enemigos diferentes chocan, se genera un nuevo diamante.

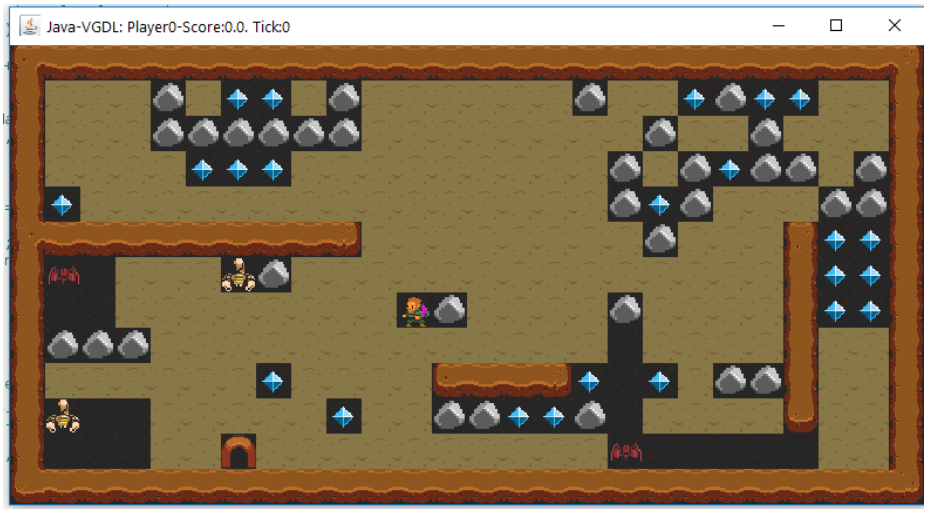

Figura 1: Mapa del primer nivel (levelIdx=0) de Boulder Dash en GVGAI

1 Se puede descargar desde https://github.com/GAIGResearch/GVGAI/archive/master.zip

Acciones: se pueden ejecutar 4 acciones de Movimiento (IZQUIERDA, DERECHA, ARRIBA, ABAJO) y una acción de USO de la pala (que en esta práctica no utilizaremos). También se pueda realizar la acción NIL (acción nula).

Para ver más información sobre cómo instalar el entorno y cómo desarrollar un controlador básico para el juego, se pueden consultar las transparencias de la presentación de la práctica y el documento Tutorial de GVGAI. En las transparencias de presentación de la práctica hay instrucciones sobre cómo instalarlo en Eclipse.

El juego 11, *Boulder Dash* es "casi" determinista. Hay enemigos confinados en "habitáculos" y mientras éstos no se abran no suponen amenaza. No obstante, si se abren los habitáculos (es decir, si el avatar excava un túnel hasta ellos), el juego es no-determinista porque, en cada *tick* del juego (es decir, en cada ciclo de ejecución de una acción del avatar), el enemigo se mueve aleatoriamente a una posición contigua, aunque no necesariamente persigue al avatar.

2. Descripción de la tarea a realizar

El objetivo de la práctica es que los estudiantes se familiaricen con los comportamientos deliberativos de las técnicas de búsqueda heurística, así como comportamientos reactivos. Para ello, se proponen 5 problemas/tareas/niveles de progresiva complejidad:

NIVEL 1. Comportamiento deliberativo simple: consistente en la búsqueda del camino **óptimo** hasta el portal. En este nivel no hay enemigos, pero sí puede haber muros que compliquen la búsqueda del camino óptimo. Se trata de planificar la mejor ruta para ir del punto en el que se encuentra el avatar hasta el punto en que se encuentra el portal.

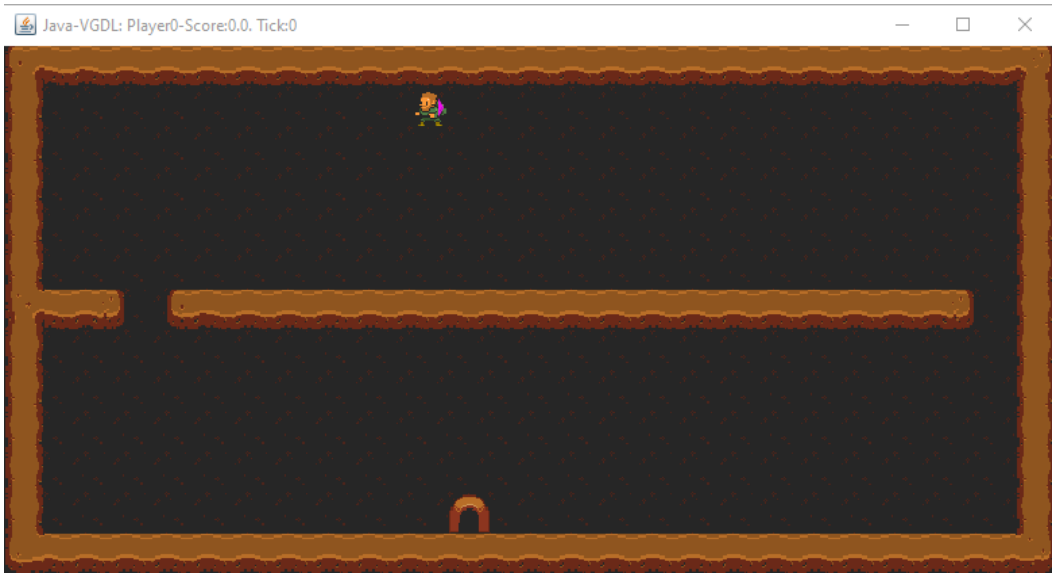

Figura 2: Ejemplo de mapa de nivel 1

Nota: Para que el juego termine sin recoger ninguna gema y así poder medir el número de *ticks* empleado, es necesario modificar la definición del juego como se explica en la presentación de la práctica.

NIVEL 2. Comportamiento deliberativo compuesto: consistente en la búsqueda de 9 gemas (en un mapa con un número igual o mayor de gemas) y, una vez se tengan todas, alcanzar el portal. Tampoco se presentan enemigos en el mapa, de modo que el comportamiento a implementar es, de nuevo, puramente deliberativo. El **orden en el que se recogen las gemas** puede escogerse siguiendo muchas estrategias diferentes (mismamente, A* podría emplearse también para escoger dicho orden), pero en esta práctica, por simplicidad, se debe seguir una **estrategia greedy usando la distancia Manhattan** (es decir, la más cercana según esta métrica), mientras que **la ruta entre dos puntos** debe ser **óptima** (usando A* o alguna de sus variantes).

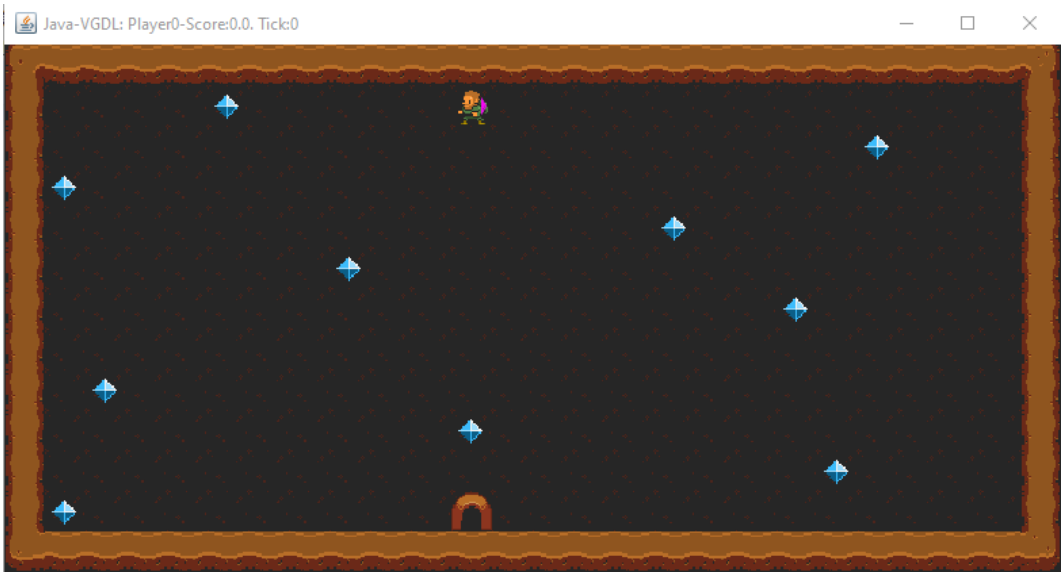

Figura 3: Ejemplo de mapa de nivel 2

NIVEL 3. Comportamiento reactivo simple: consistente en mantenerse alejado de un enemigo durante un tiempo predeterminado (los 2000 *ticks* que dura el juego). Es decir, el mapa presenta el avatar y un enemigo con movimientos pseudo-aleatorios. El objetivo es mantenerse alejado del enemigo sin que este toque al avatar y lo elimine.

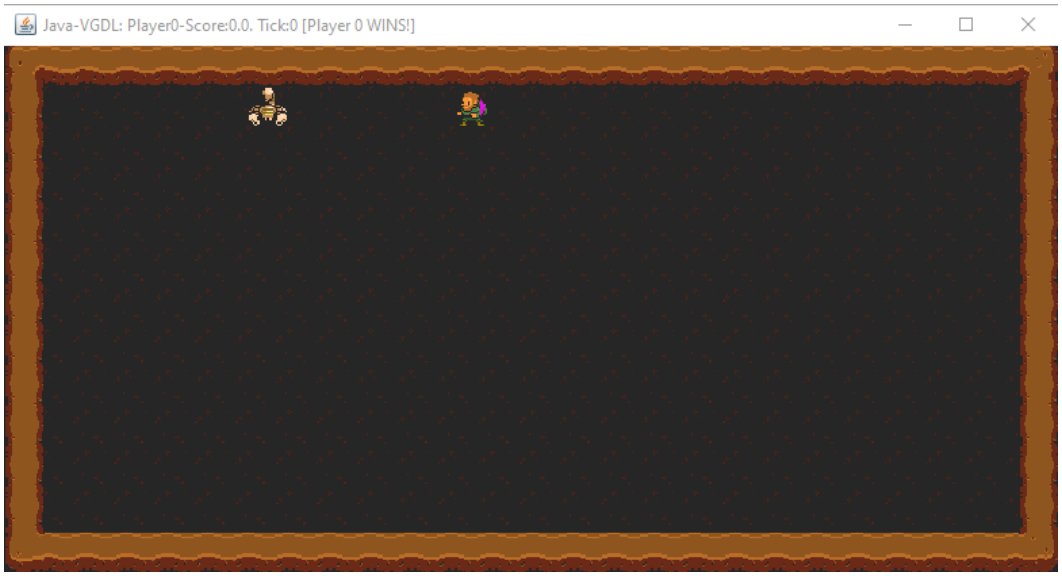

Figura 4: Ejemplo de mapa de nivel 3

NIVEL 4. Comportamiento reactivo compuesto: consistente en el mismo problema anterior, pero implicando a varios enemigos simultáneamente (este nivel se evaluará con dos enemigos, de modo que con que funcione en ese caso es suficiente).

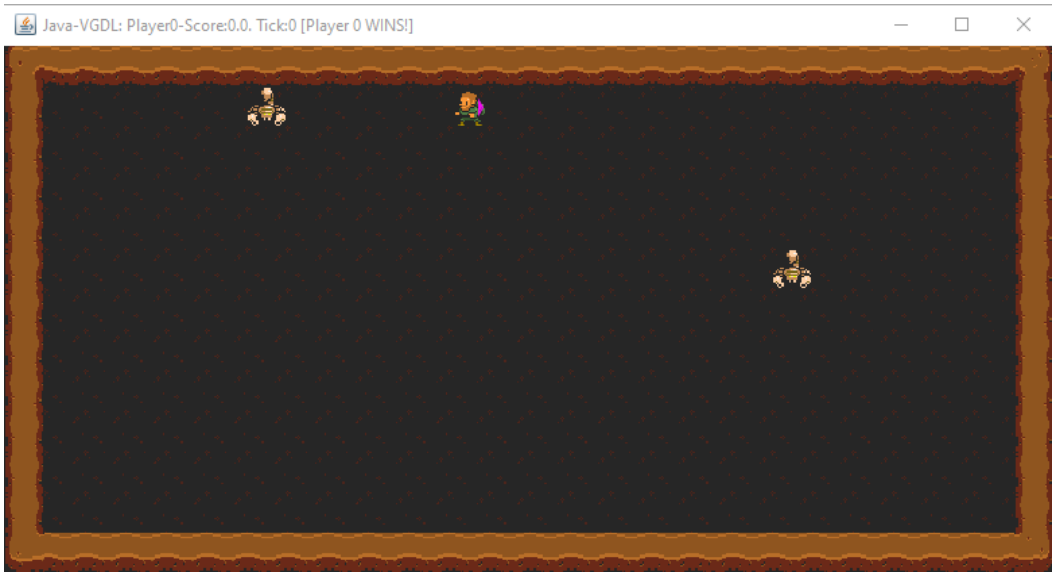

Figura 5: Ejemplo de mapa de nivel 4

NIVEL 5. Comportamiento reactivo-deliberativo: consistente en la búsqueda de 9 gemas (en un mapa con un número igual o mayor de gemas), evitando el enemigo presente en el mapa, y una vez se tengan todas alcanzar la puerta dentro de los límites de tiempo predeterminados.

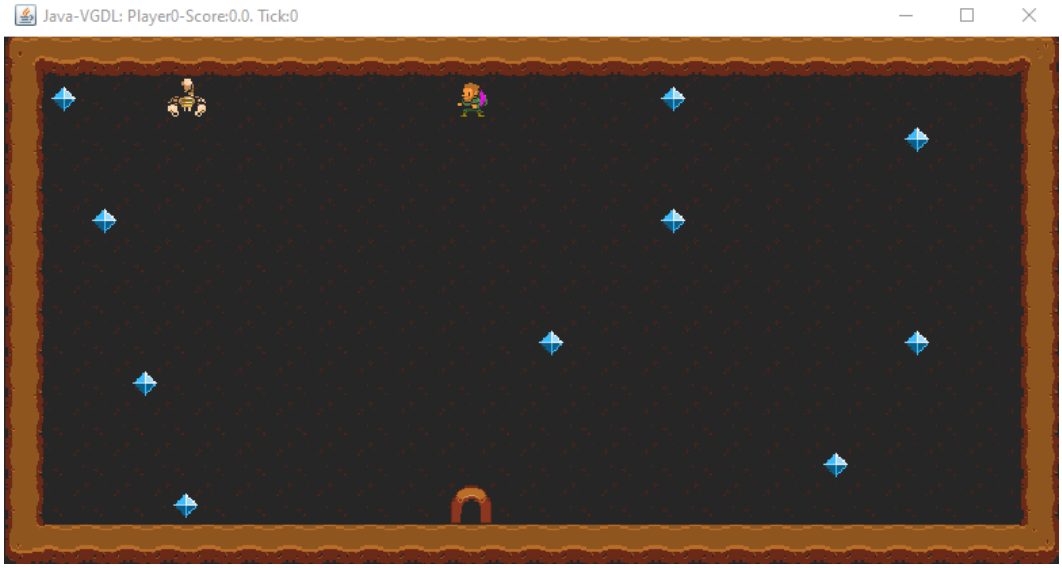

Figura 6: Ejemplo de mapa de nivel 5

Teniendo en cuenta los 5 problemas (de progresiva complejidad) anteriormente propuestos, es importante tener presente que el juego en su conjunto tiene varias características que lo convierten en un desafío desde el punto de vista de la búsqueda heurística. Un agente que resuelva el juego completo, con los cinco problemas o niveles propuestos, tiene que tener la capacidad de plantearse varios objetivos (determinar cuáles son las 9 gemas a recoger) que deben ser priorizados (recoger las gemas en el menor tiempo posible). Además, la consecución de cada objetivo es un problema de búsqueda heurística en el que no solo hay que considerar la distancia al objetivo, pues otros aspectos del estado influyen también en la evaluación de un nodo y en la determinación de cómo de prometedor es un nodo (estimación de la distancia a la que un estado se encuentra del objetivo). Por ejemplo, el número de enemigos o distancia del avatar a éstos.

Por otro lado, el agente implementado debe integrar los comportamientos reactivo y deliberativo. Reactivo porque hay que considerar el estado actual del mundo y ejecutar en cada ciclo (en cada *tick* del juego) la acción más adecuada para conseguir un objetivo inmediato (que puede variar desde ejecutar directamente una acción de un plan ya elaborado, a ejecutar una acción para evitar una situación de peligro). Deliberativo porque hay que considerar el estado actual del mundo para elaborar un plan con el fin de alcanzar objetivos a medio/largo plazo. También es interesante observar que el plan elaborado puede fallar porque ha surgido una situación inesperada y puede ser necesario replanificar. Incluso la nueva situación no prevista puede obligar a replantearse el conjunto de gemas a recoger.

A cada nivel/problema/tarea le corresponde un mapa.[2] No obstante, la entrega constará de **un único agente** que sea capaz de resolver cualquiera de los niveles.

Restricciones en la ejecución. Es importante tener en cuenta que cada ciclo de decisión está limitado de tal forma que la decisión de qué acción ejecutar en **cada *tick* debe tomarse en un tiempo no superior a 40ms**[3]. Además, **cada nivel debe superarse en menos de 2000 *ticks***.

Restricciones en la implementación. Se pide que la estrategia de búsqueda de la parte deliberativa esté basada en A* o alguna extensión del algoritmo A*. Importante remarcar que A* podría ser un algoritmo perfectamente válido para resolver el problema, y no es obligatorio recurrir a métodos más complejos. Adicionalmente, la estrategia *greedy* debe estar basada en la distancia Manhattan.

Pasos a seguir:

1) Descargar e instalar el entorno GVGAI (seguir indicaciones de las *slides* de la presentación de esta misma práctica).
2) Probar varios juegos y niveles para familiarizarse con el *framework*.
3) Consultar y revisar los materiales proporcionados con esta práctica, así como la documentación de GVGAI:
   - Presentación de la práctica, que incluye comentarios sobre GVGAI y una guía de su instalación en Eclipse.
   - Tutorial de GVGAI, en el que se describe cómo instalar el entorno, cuáles son las funciones de la API de GVGAI más relevantes para la implementación del controlador basado en A*, y cómo desarrollar un controlador básico para un juego.
   - La estructura del código y documentación básica está en https://github.com/EssexUniversityMCTS/gvgai/wiki/Code-Structure
4) Explorar y ejecutar el juego de la carrera de camellos (*Camel Race*), del que se proporciona un script sencillo (comentado en el tutorial y en las diapositivas de presentación de la práctica).
5) Comenzar la práctica implementando los niveles propuestos.

[2] Importante tener presente que los mapas con que los profesores evaluarán las soluciones de los estudiantes serán diferentes a los empleados por estos en su desarrollo de la práctica. De modo que se deben desarrollar solucionar generalistas que, dentro de la definición dada de la tarea en cuestión, permita resolver otro mapa que represente la misma casuística.

[3] Si la acción se decide entre 40-50ms GVGAI devuelve acción nula; si el tiempo es mayor a 50ms se pierde la partida.

3. Material a entregar

El material a entregar será un fichero ZIP con el siguiente contenido:

- Una carpeta denominada "src_<apellido1>_<apellido2>_<nombre>" (en caso de nombres y/o apellidos compuestos, eliminar los espacios) que incluya el código fuente en Java cumpliendo las siguientes restricciones:
  a) Debe ser un único paquete Java cuyo nombre sea el mismo que el de la carpeta, es decir "src_<apellido1>_<apellido2>_<nombre>".
  b) Debe contener al menos un fichero "Agent.java"[4] en el que se defina la clase que implementa el controlador, tal y como se describe en los tutoriales que se entregan como material de la práctica o en los tutoriales del entorno. Podrán entregarse otros ficheros fuente adicionales si así lo considera el alumno, todos ellos dentro del mismo paquete.
  c) El **código** debe estar adecuadamente presentado y **comentado**, explicando los distintos aspectos de la solución propuesta por el estudiante de una forma clara: qué se hace, cómo se hace y por qué.
  d) El código **no puede imprimir nada**. El motivo es que, de cara a la corrección de los ejercicios, realizaremos una evaluación y generación de informes automáticos. Si el código imprime algo, dificulta el análisis de dichos informes.

4. Criterios de evaluación

La evaluación consistirá en ejecutar el **software entregado** para comprobar su efectividad y eficiencia en la resolución de distintos escenarios, es decir, niveles del juego. Durante la ejecución de la implementación de búsqueda heurística se tendrán en cuenta todos los niveles descritos en el guion de la práctica (pero usando mapas distintos, es decir, por ejemplo para el nivel 1 se empleará una posición del portal distinta a la que tiene en el mapa proporcionado al estudiante y con una disposición de los muros diferente).

La calificación de la práctica se realizará teniendo en cuenta los siguientes criterios:

- 1.5 puntos: Nivel 1 (Comportamiento deliberativo simple) superado.
- 1.5 puntos: Nivel 2 (Comportamiento deliberativo compuesto) superado.
- 1.5 puntos: Nivel 3 (Comportamiento reactivo simple) superado.
- 1.5 puntos: Nivel 4 (Comportamiento reactivo compuesto) superado.
- 3 puntos: Nivel 5 (Comportamiento reactivo-deliberativo) superado.

[4] El código del agente debe contener el código para, de modo automático, funcionar con cada uno de los mapas. Es decir, debe detectar si hay gemas y/o enemigos, y actuar en consecuencia. Dicho de otro modo, no hay que entregar un "Agente.java" por cada apartado/nivel de la práctica.

- 1 punto restante (calificación 9-10): se calculará considerando el tiempo empleado en la resolución del nivel final, ordenando los trabajos entregados por cuartiles de menor a mayor tiempo. Para ello es necesario haber superado todos los niveles anteriores (con la puntuación máxima). Se considera que este punto extra valora la eficiencia de la solución proporcionada por el estudiante.

¿Cómo se define superar un nivel?

- Para el deliberativo, el número mínimo de ticks según la estrategia deliberativa de cada nivel (A* para el deliberativo simple y A*+greedy para el deliberativo compuesto). Si no se consigue, 0 puntos.
- Para el reactivo simple: toda la puntuación si se sobrevive en >=9 ejecuciones (sobre 10); 0 si es 6 o menos; calificación lineal de 6 a 9:
  - 6 veces (0 ptos)
  - 7 veces (0.5 ptos)
  - 8 veces (1 pto)
  - 9 veces (1.5 ptos)
- Para el reactivo compuesto: toda la puntuación si se sobrevive en >=7 ejecuciones (sobre 10); 0 si es 4 o menos; calificación lineal de 4 a 7:
  - 4 veces (0 ptos)
  - 5 veces (0.5 ptos)
  - 6 veces (1 pto)
  - 7 veces (1.5 ptos)
- Para el reactivo-deliberativo: toda la puntuación si se sobrevive + se recogen todas las gemas necesarias + se llega al portal en >=7 ejecuciones (sobre 10); 0 si es 4 o menos; calificación lineal de 4 a 7:
  - 4 veces (0 ptos)
  - 5 veces (1 pto)
  - 6 veces (2 ptos)
  - 7 veces (3 ptos)

Este año, a diferencia de otros, no se pide entregar memoria con esta práctica. A cambio, se hace énfasis en comentar adecuadamente el código: un código cuyos comentarios sean, a criterio del profesor, extremadamente deficientes podrán invalidar total o parcialmente aquellos niveles/tareas/problemas a los que afecte. Por ejemplo, si la implementación de A* no contiene ningún comentario o si estos se consideran extremadamente inadecuados, aunque el código funcione correctamente, implicará que la puntuación correspondiente a los niveles deliberativos no será tenida en cuenta.