Trabajo Practico Celdas: Jugador inteligente del Zelda

Martinez Gaston Alberto

Facultad de Ingenieria de la UBA
Diciembre 2 2017, Buenos Aires, Argentina

Abstract. En el presente informe se aborda el proceso de desarrollo de un agente inteligente que pueda jugar una versión reducida del juego "The legend of Zelda".

1 Introductorio

El trabajo se desarrollo en dos etapas. En la primera, el agente para ganar, debía seguir una serie de reglas predefinidas. En la segunda, el agente debía aprender un conjunto de reglas que le permitiesen ganar el juego por su propia cuenta, basado únicamente en su experiencia jugando el juego.

La primera parte del trabajo, permitió evaluar algunas decisiones de diseño que luego fueron de gran utilidad a la hora de desarrollar la segunda parte. Por esto, primero se hará un resumen de aquellas cosas que fueron importantes para el resultado final.

2 Primera Parte

Para el desarrollo de la primera versión del agente se utilizo una serie de reglas cargadas en el motor de inferencias *drools*.

2.1 Visión local del agente

En esta etapa, el agente solo tenia la capacidad de evaluar como actuar en base a las zonas sobre las cuales tenia control: las celda superior, la inferior y las que se hallaban a los lados. Esta decisión fue tomada para lograr un conjunto de reglas mas simples. Incluir mas zonas en el análisis implicaba tener que diseñar reglas mucho mas complejas y menos triviales de pensar. A su vez, el mismo funcionamiento del motor de inferencias causaba que a mayor numero de reglas, el controlador directamente fuera descalificado por timeout.

Con este modelo, si bien el agente tenia un buen desempeño, había varias situaciones en las cuales el agente tomaba caminos que eran claramente contraproducentes. Estas situaciones eran claramente cuando el agente se acercaba a las arañas y moría.

2.2 Amenazas

Acercarse directamente a un enemigo en este modelo era bastante peligroso. Si el jugador estaba frente a una pared y había un enemigo en la vecindad, el agente intentaba evitar el peligro alejándose del enemigo. El problema en esta situación radica en que para poder alejarse de la amenaza, el jugador debía rotar, acción que le consumía un turno y lo dejaba expuesto a los ataques. La misma situación se daba si intentaba atacar, ya que para posicionarse para atacar, antes debía rotar.

Con esto en mente, las reglas explicitas evitaban en todo momento la confrontación, salvo que se encontrase frente a frente con la amenaza y tuviese la capacidad de eliminarla. No obstante, si los enemigos se mueven en el turno posterior al que se mueve el jugador, este ya no tiene la capacidad de actuar de forma inteligente contra la amenaza.

2.3 Ubicación de los objetivos

En este modelo para poder simplificar la toma de decisiones, no se evalúan como objetivos distintos obtener la llave y atravesar la puerta. El agente solo sabe las ubicación de la celda a la que debe ir. De esta forma, no es necesario controlar de forma particular cada uno de estos casos.

Para lograr este efecto, se construyo un objeto denominado WorldParser, que tiene como propósito procesar la percepción del mundo y dejarle al agente solo las cosas que le importan para la toma de decisiones. De esta forma se obtiene ademas, un código mucho mas limpio ya que las operaciones de procesamiento de la percepción quedan separadas de la toma de decisiones y de la ejecución de planes.

3 Agente Inteligente

En la segunda etapa del trabajo practico, se utilizo lo aprendido de la etapa anterior para modificarlo de forma que el agente tenga la capacidad de aprender como actuar de forma autónoma.

3.1 Modelo de teorías

Para modelar las teorías, se opto por utilizar el modelo clásico y extenderlo con información que permitiese luego poder realizar de forma rápida la toma de decisiones. Las partes de las teorías son:

Condiciones y Efectos Las condiciones y efectos se representan utilizando un string que contiene tanto la observación de la vecindad del jugador, como el estado del mismo (dirección, ubicación respecto del objetivo).

Acción Las acciones posibles son solo las que producen efectos en el medio. Se descarto la posibilidad de dejar al agente no hacer nada.

Utilidad Predicha Utilidad de aplicar la acción descrita en la teoría, según la función de utilidad.

Contador de Éxitos/Aplicaciones Estos contadores se utilizan para poder calcular un indice de éxito de la teoría y así poder luego tener algún indicador que muestre si la teoría es o no fiable.

3.2 Tamaño de las causas y efectos

Para poder comenzar con el trabajo, fue necesario primero definir como representar las causas y los efectos. Ademas del como, que representar era tambien parte de la problematica. Como se puede apreciar en la figura 1, el mapa es muy grande para considerarlo completo dentro de las causas de las teorías. Mas aun, el accionar de los otros agentes presentes, hacen que haya un numero bastante alto de configuraciones posibles del estado del nivel. Esto se traduce necesariamente en teorías muy complejas y muy grandes.

Fig. 1. Mapa completo de un nivel

No obstante, el agente no tiene la capacidad de influir en las celdas que están fuera de su alcance, que son las que se muestran en la figura 2. Se podría modelar las causas y los efectos de las teorías solo en términos de esta observación. Esto es similar a la estrategia adoptada para realizar la primer parte del trabajo. De eso se había concluido que para que el agente actuase de forma "inteligente", debería poder determinar si sus acciones lo llevarían o no a ponerse en peligro, y que por ende, este modelo no era el indicado.

Con esto en mente, se analizo la situación en la vecindad de tamaño 1. Como se puede observar en la figura 3, ahora el agente tendría la capacidad de poder prever posibles amenazas. No obstante, estas no son todas las amenazas a las

Fig. 2. Área de acción del agente

cuales lo puede llevar cualquier acción. Si se analiza la vecindad de tamaño 2 (figura 4), se puede ver como aparecen mas amenazas.

Este modelo permite ahora si al agente tomar decisiones inteligentes. El problema ahora recae en si toda la información de la vecindad es completamente necesaria. La respuesta es que cualquier suceso que ocurra fuera del área encerrada por las amenazas no condiciona al agente a actuar de ninguna forma en particular. Con esto en mente, se puede suponer que a la larga, estas teorías serian generalizadas en otras que no tuviesen en cuenta estas celdas, por lo que se opto por eliminarlas desde el comienzo.

Las causas y efectos entonces se modelan como cadenas que representan las áreas antes descritas. Por ejemplo, `W . . W . . . . 2 W . W B 0 F F`, codifica el estado mostrado en la figura 5. Notar que no es necesario incluir al agente en la teoría, siendo que siempre se encontrara en la misma posición respecto del punto de observación.

Las causas y efectos ademas incluyen dos pares de símbolos que representan primero la dirección del jugador y segundo la ubicación del objetivo. Para ambos se utilizo la misma convención de símbolos:

- Primer símbolo
  f adelante (Forwards)
  b atrás (Backwards)
  0 ninguna
- Segundo símbolo
  f abajo (Forwards)
  b arriba (Backwards)
  0 ninguna

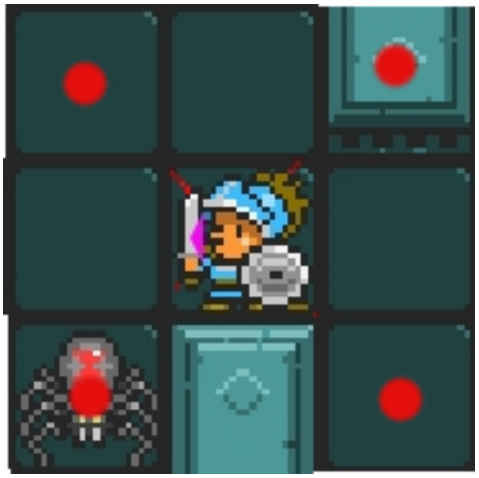

Fig. 3. Amenazas en la vecindad de tamaño 1

Se definió de forma arbitraria un símbolo comodín para simbolizar que no existe una condición sobre una determinada celda, o que no existe un efecto predicho para la misma.

3.3 Función utilidad

La función utilidad fue uno de los desafíos ocultos[1] mas difíciles que se debió afrontar en el proyecto. En las primeras etapas del desarrollo, se confió en que utilizar como función de utilidad la distancia al objetivo seria suficiente. Esto fue un grave error ya que prevenía que el agente aprendiera algo útil.

Distancia euclidiana al objetivo La primer alternativa utilizada fue definir como función utilidad la distancia en linea recta al objetivo. En el primer nivel era bastante útil para encontrar la llave, pero no para avanzar hacia la puerta.

La disposición del primer nivel coloca la llave por sobre la puerta, separadas ambas por una pared en forma de U. Con esta función utilidad, el agente una vez obtenida la llave, intentaba avanzar hacia la puerta, chocando contra la pared. Al no avanzar, intentaba descubrir una nueva teoría realizando una acción al azar. Al hacer eso, se colocaba a si mismo a mas distancia del objetivo, lo cual provocaba que volviera a la posición anterior.

Distancia real al objetivo La segunda alternativa utilizada fue reemplazar la distancia euclidiana por la distancia que calculaba el modulo de pathfinder. Esta alternativa permitió que el agente pudiese sortear finalmente el obstáculo de la pared con forma de U. En las primeras pruebas, la solución pareció ser efectiva,

[1] Oculto porque a simple vista no parecía un problema real

Fig. 4. Amenazas en la vecindad de tamaño 2

pero al entrenar al agente con unas 100 iteraciones sobre el mismo nivel, en vez de ganar mas consistentemente comenzaba a perder consistentemente. Observando la ejecución en el modo gráfico, se podía apreciar al jugador aparentemente evitando tocar la llave.

Este comportamiento fue bastante inesperado, y se atribuyo en un primer momento a otras partes del código. Mas adelante, se llego a la conclusión que el agente se comportaba de forma bastante racional, que lo que estaba mal era la función de utilidad.

Utilizando el modulo de pathfinder, al obtener la llave, la función utilidad cambia a ser la distancia a la puerta. Es decir, el agente pasaba de estar a distancia 1 a estar a 13, por lo que la utilidad bajaba dramáticamente y en consecuencia, aprendía que tomar la llave era algo necesariamente malo, mucho peor que alejarse de la llave. Esto explica entonces el comportamiento: en las primeras iteraciones el agente no tiene conciencia de que la llave es "mala" (ni "buena"), por lo que el mecanismo de generación de problemas hace que la agarre; A mayor numero de iteraciones, el agente no recurría a la generación de problemas sino a la experiencia, y por consiguiente no agarraba la llave.

Función final De lo aprendido de las estrategias anteriores, se concluyo que era mucho mas efectivo definir de forma manual la función de utilidad en base a los efectos que las acciones provocaban. Se continuo utilizando para medir la distancia el modulo de pathfinder, pero solo para poder saber si el jugador se acerca o se aleja del objetivo.

La función de utilidad definida es de la forma:

Fig. 5. Visión final del agente

$$U(i,j) = \begin{cases} 100 & \textit{Matar un enemigo} \\ 50 & \textit{Llegar al objetivo} \\ 30 & \textit{Acercarse al objetivo} \\ -1 & \textit{Cambiar de dirección} \\ -40 & \textit{Alejarse del objetivo} \\ -100 & \textit{No hacer nada} \\ -999 & \textit{Morir} \end{cases} \quad (1)$$

Los valores fueron definidos de forma de poder cumplir con las siguientes restricciones:

Evitar simetría: Si dos eventos tienen la misma utilidad con distinto signo, el agente puede optar por hacer y deshacer una acción como alternativa valida.

Castigar las acciones nulas: Para evitar que el agente se quede girando en círculos o intentando atacar/moverse contra una pared, se penaliza el no avanzar hacia el objetivo

Penalizaciones sobre beneficios: Para evitar que se tomen malas acciones, se penaliza mas fuertemente que lo que se da de beneficio. Es así que alejarse del objetivo y luego acercarse deja un saldo negativo, ya que la acción en si no es productiva.

Evitar problemas a futuro: Se incita al agente a eliminar a los enemigos. Si bien es una estrategia un tanto arriesgada, menos enemigos se traduce en menos amenazas a controlar en el futuro.

Esta función mostró ser mucho mejor que las opciones anteriores al no tener casos patológicos en los cuales su aplicación evitara el correcto aprendizaje del

agente. Las únicas situaciones en las cuales la función hace que el agente funcione de forma "poco inteligente", son aquellas en las que (dado que eliminar un enemigo es la mejor acción posible) comienza a perseguir a los enemigos poniéndose en un posible peligro y alejándose del objetivo. No obstante, estas son las menos, y de matar al enemigo al final del juego el marcador seria mayor, por lo cual no parece un comportamiento muy poco deseado.

3.4 Heuristicas

Se implementaron dos heuristicas para generar teorías mutantes:

Retracción Para generalizar los efectos predichos en las teorías
Simplicidad Para generalizar las causas de las teorías

Los objetivos de estas heuristicas son intentar reducir el numero total de teorías totales, teniendo un mayor numero de teorías generales y muchas menos teorías especificas.

La retracción se aplica constantemente al identificar que el estado actual del medio, no se condice con el estado predicho en el turno anterior. La heuristica de simplicidad se aplica cada un numero n de iteraciones y al finalizar la partida.

Es fácil ver que al generar mas teorías generales, estas tienen mayor aplicación en nuevos niveles, con lo cual es menos probable que el agente deba generar nuevas teorías para poder predecir su comportamiento. De esta forma, el numero de teorías que el agente deberá evaluar en cada caso es mucho menor y por consiguiente evita que se puedan generar timeouts.

3.5 Planificador

Para el planificador se opto por una estrategia local mas que una estrategia global. Es decir, el planificador decide entre las distintas acciones solo en base a la observación de la vecindad actual y la predicción del estado siguiente. Esta decisión esta basada en tres puntos clave:

- Dada la naturaleza del entorno, es muy factible que las condiciones cambien muy rápidamente (aparezca una araña donde no se esperaba). Un plan muy especifico rápidamente quedaría invalidado por alguno de los eventos estocásticos. Entonces, no tiene sentido gastar mucho tiempo en decidir una acción a futuro si es muy posible que debamos re-evaluarla pronto.
- Se busca tener teorías que sean lo mas genéricas posibles. Con esto en mente, es fácil ver que a medida que aplican mas predicciones sucesivas, la incertidumbre aumenta drasticamente.
- Analizar todas las posibilidades a medida que el numero de teorías aumenta, se vuelve una operación muy costosa y en cierto punto incomputable.

Para poder encontrar la acción a tomar, el planificador obtiene para cada una de las acciones posibles el estado predicho según el criterio de selección (Explicado en la sección correspondiente). Para cada uno de estos estados, aplica nuevamente el mismo mecanismo. Para cada par de acciones encadenadas, se computa ahora su utilidad, que es simplemente la suma de las utilidades predichas por cada acción particular. Finalmente, la acción a tomar es la que genera una mayor utilidad.

3.6 Generador de problemas

Hay casos en los cuales no hay ninguna teoría que aplique al estado actual censado. En estos casos, se generan teorías nuevas para que el agente pueda aprender como actuar en estas condiciones. Las teorías nuevas se marcan con una utilidad de 0, y como efectos predichos se marca el mismo estado que el inicial. Cuando se aplica por primera vez una teoría generada por este mecanismo, en vez de evaluarla de forma similar a cualquier otra, simplemente se le reemplaza el efecto predicho por el nuevo estado censado, la utilidad por la utilidad calculada y 1 en cantidad de éxitos / aplicaciones.

Existe un caso particular en el cual se utiliza el mecanismo pese a existir soluciones disponibles: si el planificador no encuentra una alternativa que tenga una utilidad positiva. Básicamente, no existen en lineas generales situaciones en las cuales todas las opciones posibles de acciones den una utilidad negativa o nula. En caso de encontrar eso, es posible que falte cierta información del entorno, y es por eso que se generan estas teorías para incitar a la exploración en esas situaciones.

3.7 Criterios de selección

Otra de las características del trabajo que mas problemas acarreo fue como determinar cual era el efecto mas probable de aplicar cada acción a cada situación dada. Se probaron sin éxito utilizar la mayor utilidad predicha, la mayor tasa de éxitos y combinaciones de estas sin buenos resultados.

Se llego luego a la conclusión que el mal desempeño de los criterios anteriores era en parte causa de las heuristicas de generalización. Al entrenar mucho al agente, se generaban muchas teorías que eran muy poco especificas. Estas teorías eran luego seleccionadas por su alto indice de éxito pero terminaban fallando al no captar algunas cosas clave del estado actual. Para mitigar este efecto, se introdujo el denominado indice de especificidad, el cual simplemente es el numero de condiciones presentes en la causa sobre el numero total de condiciones posibles dentro de la misma.

Eligiendo las predicciones utilizando el producto entre el indice de éxito y el indice de especificidad, se obtuvieron mejores resultados, permitiéndole al agente tomar decisiones mejor ajustadas a la situación.

3.8 Eliminación de Teorías

El principal problema que se encontró a la hora de desarrollar el trabajo, fue como reducir el numero de teorías. El objetivo en si, es encontrar el conjunto mas pequeño posible de las teorías lo mas genéricas posibles que le permitan al agente actuar.

Se implementaron tres mecanismos para lograr este efecto:

Eliminacion de invalidas: Las teorías que tienen utilidad 0 son aquellas que genero el generador de problemas. En algunos casos puede que se generen de mas, por lo que es necesario limpiar de forma regular la base de conocimiento para que no las tenga en cuenta para la toma de decisiones.

Compactacion: Las heuristicas generan a veces teorías duplicadas. Para evitar que haya múltiples versiones de la misma teoría y se pierda el correcto control de éxito de las teorías, se eliminan aquellas que compartan la misma causa, consecuencia, utilidad y acción, dejando solo la que tuvo mayor cantidad de aplicaciones.

Eliminacion de erróneas: Para evitar evaluar teorías que ya se probo no funcionan, se eliminan aquellas teorías que hayan sido aplicadas mas de n veces y tengan un indice de éxito menor a un umbral.

Estos mecanismos se aplican regularmente durante la ejecución de cada partida, cada un numero fijo de turnos, y una vez al terminar la partida.

4 Entrenamiento y Resultados

Para entrenar al agente se utilizo un bucle sobre el modo no interactivo de ejecución por obvias razones. Siendo que el objetivo del trabajo en si es lograr que el agente pueda ganar todos los niveles se opto por entrenarlo de forma intercalada en todos los niveles. Esta metodología previene que el agente aprenda reglas muy especificas del nivel, que hagan que luego le sea difícil jugar otros niveles. Esto ultimo se debe a que si se utiliza mucho una teoría, y esta tiene un alto margen de éxito, por el mecanismo de selección aplicado, siempre se elegirá por sobre otras, y en caso de fallar, tardaría mucho en corregir el indice de éxito. Si en cambio se ejecutan solo un numero reducido de veces, es mas fácil poder descartarlas (porque no sirven de forma indistinta para todos los niveles).

De algunas pruebas anteriores, se llego a la la conclusión de que el agente inteligente diseñado de esta forma requería de algunas iteraciones en el mismo nivel antes de poder comenzar a moverse de forma fluida. Es por esto que en vez de intercambiar nivel entre partida y partida, se juega 10 veces un nivel antes de pasar al siguiente. Esto le da la posibilidad al agente no solo de aprender algunas reglas que le permitan desplazarse por el nivel en cuestión; sino que también le permite generar teorías mas genéricas que tienen mayor capacidad de aplicación en otros niveles. Al ser también un numero bajo de iteraciones por nivel, el orden de ejecución de los mismos no influye en el resultado, así que se

eligió un recorrido del tipo round robin.

Con esta metodología, entrenando al agente unas 600 iteraciones, se logra que pueda ganar los niveles un 30% de las veces en las ultimas iteraciones. Estos resultados no son en si los esperados pero se supone que si se lo entrena un mayor numero de iteraciones se obtendrán mejores resultados. La única limitante, esta en el hecho de que si bien el mecanismo de remoción funciona, no es capaz de minimizar significativamente el numero de teorías. Dado esto, el agente en algunas ocasiones tiene una base de conocimiento que no puede llegar a procesar en los tiempos estipulados y es descalificado por timeout.

5 Conclusión

En el trabajo se pudo aplicar la teoría de agentes inteligentes a un caso concreto. Esto permitió apreciar en un caso real las influencias de ciertas decisiones de diseño que a priori parecían un tanto superfluas. El caso mas relevante fue el de la función utilidad, que en una primera etapa del análisis parecía poder ser trivial implementada como la distancia al objetivo.

A modo de lección aprendida de este trabajo queda que antes de realizar un agente inteligente autónomo, es buena idea implementarlo de forma no autónoma. En este caso, realizar la versión con reglas fijas permitió observar ciertos comportamientos del sistema que no eran visibles en la versión interactiva, por ejemplo, las rotaciones.

References

1. Agentes Inteligentes Autónomos - Garcia Martinez - (Capítulo 4)

Apéndice: Ultimas iteraciones de un entrenamiento

El código se acompaña de un archivo de teorías generado mediante un entrenamiento de 600 iteraciones. A continuación, se muestran los resultados de las ultimas iteraciones de dicho entrenamiento.

```
Iteration 500 --> Level 1
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:1.0, timesteps:64
Iteration 501 --> Level 1
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:-1.0, timesteps:75
Iteration 502 --> Level 1
Result (1->win; 0->lose): Player0:0, Player0-Score:7.0, timesteps:2000
Iteration 503 --> Level 1
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:1.0, timesteps:52
Iteration 504 --> Level 1
Result (1->win; 0->lose): Player0:1, Player0-Score:6.0, timesteps:136
Iteration 505 --> Level 1
Result (1->win; 0->lose): Player0:1, Player0-Score:2.0, timesteps:827
Iteration 506 --> Level 1
Result (1->win; 0->lose): Player0:1, Player0-Score:6.0, timesteps:1045
Iteration 507 --> Level 1
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:112
Iteration 508 --> Level 1
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:-1.0, timesteps:60
Iteration 509 --> Level 1
Too long: 0(exceeding 251ms): controller disqualified.
Result (1->win; 0->lose): Player0:-100, Player0-Score:-1000.0, timesteps:31
Iteration 510 --> Level 2
Result (1->win; 0->lose): Player0:1, Player0-Score:2.0, timesteps:265
Iteration 511 --> Level 2
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:87
Iteration 512 --> Level 2
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:110
Iteration 513 --> Level 2
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:200
Iteration 514 --> Level 2
Result (1->win; 0->lose): Player0:1, Player0-Score:2.0, timesteps:197
Iteration 515 --> Level 2
```

```
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:-1.0, timesteps:42
Iteration 516 --> Level 2
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:76
Iteration 517 --> Level 2
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:125
Iteration 518 --> Level 2
Too long: 0(exceeding 276ms): controller disqualified.
Result (1->win; 0->lose): Player0:-100, Player0-Score:-1000.0, timesteps:47
Iteration 519 --> Level 2
Result (1->win; 0->lose): Player0:1, Player0-Score:2.0, timesteps:145
Iteration 520 --> Level 3
Result (1->win; 0->lose): Player0:1, Player0-Score:4.0, timesteps:282
Iteration 521 --> Level 3
Result (1->win; 0->lose): Player0:1, Player0-Score:4.0, timesteps:151
Iteration 522 --> Level 3
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:122
Iteration 523 --> Level 3
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:99
Iteration 524 --> Level 3
Too long: 0(exceeding 332ms): controller disqualified.
Result (1->win; 0->lose): Player0:-100, Player0-Score:-1000.0, timesteps:31
Iteration 525 --> Level 3
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:125
Iteration 526 --> Level 3
Result (1->win; 0->lose): Player0:1, Player0-Score:6.0, timesteps:118
Iteration 527 --> Level 3
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:116
Iteration 528 --> Level 3
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:-1.0, timesteps:56
Iteration 529 --> Level 3
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:-1.0, timesteps:36
Iteration 530 --> Level 4
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:1.0, timesteps:9
Iteration 531 --> Level 4
I'm Dead
```

```
Result (1->win; 0->lose): Player0:0, Player0-Score:-1.0, timesteps:10
Iteration 532 --> Level 4
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:2.0, timesteps:166
Iteration 533 --> Level 4
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:94
Iteration 534 --> Level 4
Result (1->win; 0->lose): Player0:1, Player0-Score:2.0, timesteps:52
Iteration 535 --> Level 4
Result (1->win; 0->lose): Player0:1, Player0-Score:2.0, timesteps:64
Iteration 536 --> Level 4
Result (1->win; 0->lose): Player0:1, Player0-Score:2.0, timesteps:46
Iteration 537 --> Level 4
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:81
Iteration 538 --> Level 4
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:230
Iteration 539 --> Level 4
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:-1.0, timesteps:18
Iteration 540 --> Level 5
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:-1.0, timesteps:59
Iteration 541 --> Level 5
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:30
Iteration 542 --> Level 5
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:-1.0, timesteps:56
Iteration 543 --> Level 5
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:-1.0, timesteps:29
Iteration 544 --> Level 5
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:22
Iteration 545 --> Level 5
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:44
Iteration 546 --> Level 5
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:32
Iteration 547 --> Level 5
I'm Dead
```

```
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:62
Iteration 548 --> Level 5
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:2.0, timesteps:23
Iteration 549 --> Level 5
Result (1->win; 0->lose): Player0:1, Player0-Score:6.0, timesteps:176
Iteration 550 --> Level 1
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:496
Iteration 551 --> Level 1
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:1.0, timesteps:212
Iteration 552 --> Level 1
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:133
Iteration 553 --> Level 1
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:1.0, timesteps:416
Iteration 554 --> Level 1
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:-1.0, timesteps:52
Iteration 555 --> Level 1
Too long: 0(exceeding 263ms): controller disqualified.
Result (1->win; 0->lose): Player0:-100, Player0-Score:-1000.0, timesteps:15
Iteration 556 --> Level 1
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:113
Iteration 557 --> Level 1
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:248
Iteration 558 --> Level 1
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:86
Iteration 559 --> Level 1
Result (1->win; 0->lose): Player0:1, Player0-Score:4.0, timesteps:240
Iteration 560 --> Level 2
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:132
Iteration 561 --> Level 2
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:-1.0, timesteps:87
Iteration 562 --> Level 2
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:134
Iteration 563 --> Level 2
```

```
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:158
Iteration 564 --> Level 2
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:119
Iteration 565 --> Level 2
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:88
Iteration 566 --> Level 2
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:138
Iteration 567 --> Level 2
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:148
Iteration 568 --> Level 2
Result (1->win; 0->lose): Player0:1, Player0-Score:2.0, timesteps:241
Iteration 569 --> Level 2
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:126
Iteration 570 --> Level 3
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:-1.0, timesteps:64
Iteration 571 --> Level 3
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:72
Iteration 572 --> Level 3
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:-1.0, timesteps:64
Iteration 573 --> Level 3
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:2.0, timesteps:346
Iteration 574 --> Level 3
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:1.0, timesteps:84
Iteration 575 --> Level 3
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:-1.0, timesteps:23
Iteration 576 --> Level 3
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:-1.0, timesteps:60
Iteration 577 --> Level 3
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:-1.0, timesteps:41
Iteration 578 --> Level 3
I'm Dead
```

```
Result (1->win; 0->lose): Player0:0, Player0-Score:-1.0, timesteps:28
Iteration 579 --> Level 3
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:4.0, timesteps:104
Iteration 580 --> Level 4
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:202
Iteration 581 --> Level 4
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:-1.0, timesteps:18
Iteration 582 --> Level 4
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:82
Iteration 583 --> Level 4
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:182
Iteration 584 --> Level 4
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:-1.0, timesteps:50
Iteration 585 --> Level 4
Too long: 0(exceeding 3ms): controller disqualified.
Result (1->win; 0->lose): Player0:-100, Player0-Score:-1000.0, timesteps:47
Iteration 586 --> Level 4
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:-1.0, timesteps:16
Iteration 587 --> Level 4
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:-1.0, timesteps:18
Iteration 588 --> Level 4
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:2.0, timesteps:88
Iteration 589 --> Level 4
I'm Dead
Result (1->win; 0->lose): Player0:1, Player0-Score:1.0, timesteps:224
Iteration 590 --> Level 5
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:-1.0, timesteps:59
Iteration 591 --> Level 5
Result (1->win; 0->lose): Player0:1, Player0-Score:6.0, timesteps:172
Iteration 592 --> Level 5
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:33
Iteration 593 --> Level 5
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:64
```

```
Iteration 594 --> Level 5
Result (1->win; 0->lose): Player0:1, Player0-Score:6.0, timesteps:330
Iteration 595 --> Level 5
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:120
Iteration 596 --> Level 5
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:1.0, timesteps:67
Iteration 597 --> Level 5
Result (1->win; 0->lose): Player0:1, Player0-Score:2.0, timesteps:389
Iteration 598 --> Level 5
Too long: 0(exceeding 40ms): controller disqualified.
Result (1->win; 0->lose): Player0:-100, Player0-Score:-1000.0, timesteps:63
Iteration 599 --> Level 5
I'm Dead
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:52
```