UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Creación de tareas competitivas multi-agente en Minecraft

Trabajo Fin de Máster

Máster Universitario en Ingeniería Informática

Autor: Carlos Javier Martínez Pedrón

Tutor: José Hernández Orallo

2017/2018

Resumen

El objetivo de este trabajo es crear tareas competitivas multiagente en el mundo de Minecraft. Para ello, se utiliza la plataforma Project Malmo, que nos permite crear entornos en este mundo y lanzar agentes que tengan como objetivo cumplir una misión. Mediante XML se crean los escenarios y con Python las dinámicas de las tareas y los algoritmos. Posteriormente, se implementa el algoritmo Q-Learning y adapta a cada una de las tareas. Además, se desarrollan algoritmos de estrategia fija para poder compararlos con el Q-Learning. Con estas comparaciones, se comprueba el beneficio de los algoritmos de aprendizaje automático frente a otros de estrategia fija y la correcta implementación de las tareas.

Palabras clave: Inteligencia Artificial, Malmo, aprendizaje por refuerzo, multiagente, Minecraft.

Abstract

The aim of this paper is create multi-agent competitive tasks in the world of Minecraft. For it, we use the platform “Project Malmo”,wich allows us to create environments in this world and launch agents which objective is achieve a mission. Through XML, the environments are created and with Python the dynamics of the tasks and the algorithms. Subsequently, the Q-Learning algorithm is implemented and adapted to each of tasks. In addition, fixed strategy algorithms are developed to be able to compare them with Q-Learning. With these comparisons, the benefit of automatic learning algorithms is tested against others of fixed strategy and the correct implementation of the tasks.

Keywords : Artificial Intelligence, Malmo, reinforcement learning, multi-agent, Minecraft.

.

Tabla de contenidos

1. Introducción

En la actualidad, cada vez escuchamos más las palabras Inteligencia Artificial (IA), y es que a diario, sin darnos cuenta, estamos en contacto con la IA casi en todas partes. Aunque cuando pensamos en IA parece que pensemos en robots inteligentes supercomplejos que pueden convivir y actuar con el ser humano, realmente la IA se encuentra en objetos o máquinas que usamos a diario como pueden ser los ordenadores, teléfonos móviles, robots de limpieza, en diferentes sectores como la industria, medicina, automoción…

La IA trata de construir sistemas computacionales que demuestren comportamientos inteligentes, que sean capaces de observar el entorno y tomar decisiones para lograr cumplir ciertos objetivos y aprender de las acciones tomadas. Por lo que en la mayoría de los casos tratan de hacer la vida más cómoda y sencilla al ser humano, delegando este algunas responsabilidades y toma de decisiones a estos sistemas computacionales.

Para fomentar la investigación en IA, un equipo de Microsoft ha creado una plataforma llamada Project Malmo, con la que investigadores pueden desarrollar y probar algoritmos de IA de forma sencilla y divertida, ya que en el mundo de Minecraft se pueden crear infinidad de tareas, ya sean sencillas o complejas. Además permite crear sistemas multiagente en los que diferentes agentes pueden colaborar y competir para lograr sus objetivos.

1.1 Motivación

Dada la importancia de la IA en la actualidad, la motivación principal de este trabajo es poder colaborar con el proyecto de Microsoft y ofrecer diferentes desafíos para investigadores del sector. Se utilizará la plataforma Project Malmo para crear tareas competitivas multiagente, en las que se puedan probar algoritmos de manera sencilla y evaluarlos. Estas podrán servir como ejemplo para otros desarrolladores y como ayuda para poder implementar sus propios desafíos. Además de crear las tareas, se pretende desarrollar un algoritmo de IA y otro de estrategia fija para cada una ellas, de forma que una vez compartido el código, otros desarrolladores puedan comparar sus algoritmos con estos o mejorarlos.

1.2 Objetivos

El objetivo principal de este proyecto es crear diferentes tareas competitivas multiagente, para posteriormente implementar un algoritmos que nos permitan resolver los problemas planteados y evaluar los resultados obtenidos. Este objetivo principal lo podemos dividir en una serie de objetivos secundarios a realizar:

- Pensar en tres tareas competitivas multiagente que puedan ser implementados en la plataforma Project Malmo.
- Elegir un lenguaje de programación de los disponibles para la plataforma Project Malmo, con el que nos resulte más sencillo implementar las tareas y el algoritmo para resolverlas.
- Implementar las tres tareas, algoritmos de estrategia fija y un algoritmo de aprendizaje automático para cada una de ellas, que permitirán a los agentes realizar las tareas.
- Realizar pruebas en cada tarea para comparar si alguno de los agentes tiene cierta ventaja frente al otro utilizando los mismos algoritmos.
- Enfrentar a dos agentes en las tres tareas, uno con un algoritmo de estrategia fija y el otro con el de aprendizaje automático.
- Publicar código y resultados para que otros desarrolladores puedan usarlo y mejorarlo.

1.3 Estructura de la memoria

En esta memoria se van a tratar diversos temas relacionados con el trabajo. En primer lugar, la introducción, contexto y objetivos con el fin de situar al lector en contexto del trabajo. En segundo lugar, se desarrollará una sección sobre el marco teórico con el fin de explicar brevemente los conceptos básicos necesarios para entender el trabajo realizado, como son los agentes inteligentes, aprendizaje por refuerzo, el Q-Learning y la evaluación de la IA. En tercer lugar, se describirán las herramientas utilizadas para realizar los objetivos propuestos. En cuarto lugar, se explicarán las tres tareas desarrolladas y los algoritmos implementados. Se realizarán distintas evaluaciones orientadas a tareas, comparando los algoritmos entre sí y se analizarán los resultados obtenidos y las dificultades encontradas. En último lugar, se detallarán las conclusiones y el trabajo futuro de este proyecto.

2. Marco teórico

En este apartado se definen los conceptos teóricos requeridos para el desarrollo del trabajo. En la sección 2.1 se definen los sistemas multiagente y la clasificación de estos dependiendo de las características de los agentes. La sección 2.2 trata del aprendizaje por refuerzo y de uno de los algoritmos más conocidos para esta categoría: Q-Learning. Por último en la sección 2.3 se describen los tipos de evaluación de la IA.

2.1 Sistemas multiagente

Russell y Norvig (1995) definen de forma simple un agente como cualquier cosa que pueda verse percibiendo el entorno a través de sensores y actuar sobre el entorno a través de actuadores.

Una definición más completa es la que define Maes (1995):

“Los agentes autónomos son sistemas computacionales que habitan un entorno complejo, dinámico, sensitivo y actúan de manera autónoma en ese entorno, y realizando así un conjunto de metas o tareas para las que están diseñados”.

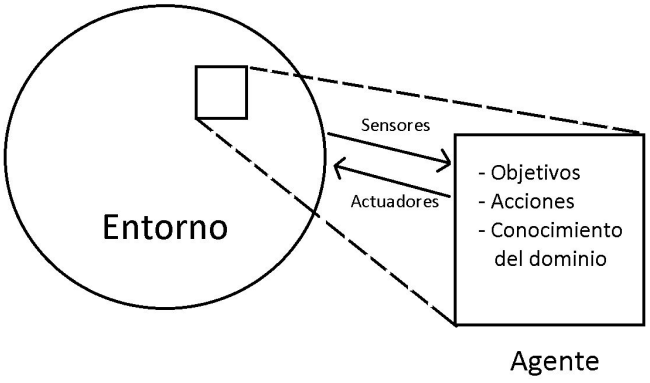


Figura 2.1. Esquema de agente

Una vez definido el concepto de agente podemos definir un sistema multiagente. Stone y Veloso (2000) dan la siguiente definición:

"Un sistema multiagente es una red poco acoplada de entidades (agentes) de resolución de problemas que trabajan juntos para encontrar respuestas a problemas que están más allá de las capacidades individuales o conocimiento de cada entidad (agente)"

En los sistemas multiagente podemos clasificar en diferentes tipos los agentes involucrados dependiendo de distintos criterios: colaboración, roles y dominio del conocimiento.

Según el criterio de colaboración podemos distinguir entre si los agentes tienen un objetivo común, en cuyo caso se pueden definir como agentes colaborativos, que colaboran entre ellos para lograr obtener el mayor número de recompensas o si por el contrario tienen un objetivo individual, pertenecen al grupo de agentes competitivos, en el que los agentes compiten por lograr una mayor recompensa individual (Hoen et al. 2006).

Según el rol que tengan los agentes podemos clasificar un sistema multiagente como homogéneo si todos los agentes tienen el mismo rol o heterogéneo en caso de ser diferentes (Liu y Wu 2001).

Según el dominio del conocimiento que tengan los agentes podemos clasificarlos como conocimiento parcial en caso de que los agentes no conozcan toda la información o conocimiento total si los agentes conocen todos los datos del entorno. También puede darse el caso de que los agentes tengan diferentes dominios de conocimiento por lo que podrían clasificarse como mixtos (Ishii et al. 2001).

2.2 Aprendizaje por refuerzo

El aprendizaje por refuerzo trata de determinar las acciones que debe realizar un agente en un entorno dado para maximizar su recompensa (Murphy K. 1998). Al agente no se le indican las acciones que tiene que escoger, sino que se le ofrecen un conjunto de acciones permitidas para el estado actual y mediante ensayo y error el agente va aprendiendo cual es la que mayores recompensas le ofrece. En la mayoría de los problemas definidos las acciones tomadas por el agente pueden conducir a dos tipos de recompensas, que son las inmediatas y las de a largo plazo.

En el aprendizaje por refuerzo existen dos elementos fundamentales: el agente que es el que aprende y toma las decisiones y el entorno que es el medio con el que interactúa el agente y sobre el que repercuten las acciones tomadas, ofreciendo en forma de estado la situación actual y recompensas por cada acción realizada. El esquema de esta interacción entre el agente y el entorno puede verse en la Figura 2.2.

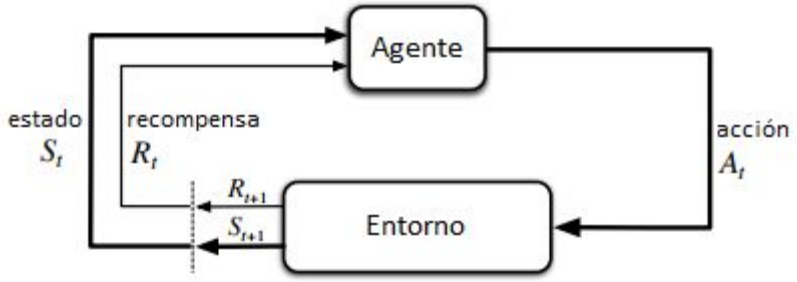


Figura 2.2. Interacción del agente con el entorno en el aprendizaje por refuerzo (Sutton y Barto 2012).

Uno de los principales problemas del aprendizaje automático es encontrar el equilibrio entre exploración y explotación (Tokic y Palm 2011). Para obtener una mayor recompensa el agente debe elegir entre las acciones que mayor recompensa le han dado en el pasado, pero para encontrar estas acciones, el agente debe explorar nuevas acciones con el fin de encontrar las mayores recompensas. El dilema está en explorar el terreno para poder encontrar la mayor recompensa posible y no quedarse atascado en una solución parcial que ofrezca una recompensa elevada pero que no sea la óptima. (Sutton y Barto 2012).

2.2.1 Q-Learning

Q-Learning es una técnica de aprendizaje por refuerzo sin modelo, lo que significa que el agente no puede predecir cuál será el siguiente estado y recompensa que obtendrá antes de realizar una acción (Huys et al. 2014). Esta técnica permite a los agentes aprender para tomar las acciones adecuadas para resolver problemas de forma óptima en dominios de Markov, observando las consecuencias que tienen las acciones tomadas (Watkins y Dayan 1992).

El proceso de aprendizaje establecido por esta técnica es el siguiente: el agente prueba a realizar una acción en un determinado estado y recibe una recompensa o penalización y un nuevo estado mediante las observaciones. Probando todas las acciones disponibles en cada uno de los estados existentes, el agente aprende cuales son las acciones óptimas gracias a las recompensas y penalizaciones obtenidas para cada una de ellas.

Algoritmo

El modelo del problema está compuesto por un agente, un conjunto de estados S y un conjunto de acciones permitidas por estado A. Mediante las acciones el agente puede moverse de un estado a otro y cada vez que el agente realiza una acción obtiene una recompensa o penalización. El objetivo del agente el obtener el mayor número de recompensas, para ello aprende con qué acción obtiene una mayor recompensa dependiendo de su estado actual. La acción óptima para cada estado es la que mayor recompensa ofrece a largo plazo, que es una suma ponderada de las recompensas de las acciones futuras a partir del estado actual. El peso de las recompensas para las acciones futuras es γ, cuyo valor está comprendido entre 0 y 1, llamado factor de descuento, que se utiliza para dar mayor o menor importancia a las recompensas futuras.

El algoritmo está compuesto por una función que calcula la cantidad de una combinación de estado-acción:

$$Q : S \times A \rightarrow R$$

Antes del aprendizaje, Q devuelve un valor fijo elegido por el desarrollador, posteriormente para cada par acción estado se actualiza el valor de Q dependiendo de la recompensa obtenida, la acción tomada y el estado actual y futuro.

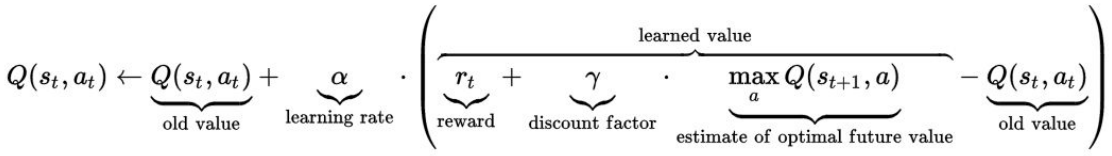


Figura 2.3. Función Q del Q-Learning. (Q-Learning, s.f.)

En la Figura 2.3 podemos ver la función Q, donde r es la recompensa obtenida y α la tasa de aprendizaje, que será un valor comprendido entre 0 y 1.

Para adaptar el algoritmo a los diferentes problemas podemos modificar los valores de las variables: tasa de aprendizaje (α), factor de descuento (γ) y condiciones iniciales (Q_0).

La tasa de aprendizaje determina la importancia de la recompensa adquirida recientemente con respecto a la anterior. De forma que si establecemos esta variable a 0 conseguiremos que el agente no aprenda nada, mientras que si la ponemos a 1 indicaríamos al agente que solo tenga en cuenta la información más reciente.

El factor de descuento determina la importancia de las recompensas futuras. Establecer la variable a 0 hará que el agente sólo considere las recompensas actuales, mientras que ponerla a 1 hará que el agente se esfuerce por obtener un mayor número de recompensas a largo plazo.

Las condiciones iniciales son los valores que se establecen antes de que se produzca la primera actualización del valor de Q. Pueden utilizarse valores altos para estimular la exploración. Con unos valores altos las acciones que no hayan sido tomadas serán candidatas a ser elegidas ya que las que ya hayan sido elegidas habrán dejado un valor menor al establecido inicialmente. (Q-Learning, s.f.)

2.3 Evaluación de la IA

Una vez creados los algoritmos de IA debemos evaluar los mismos para compararlos con otros y ver cuales ofrecen mejores resultados. La evaluación de la IA trata de evaluar la inteligencia de los artefactos creados. Podemos distinguir entre dos tipos de evaluación que se explicarán con más detalle a continuación: la evaluación orientada a tareas y la orientada a la habilidad (Hernández 2016).

2.3.1 Evaluación orientada a tareas

Vamos a distinguir tres tipos de evaluación del comportamiento: por la discriminación humana (realizando una comparación contra o por los seres humanos), los puntos de referencia de problemas (un repositorio o generador de problemas) y por la confrontación entre pares (‘partidos’ multiagente o 1-vs-1).

2.3.1.1 Evaluación por discriminación humana

En esta categoría se incluyen las evaluaciones que se realizan mediante una comparación con o por seres humanos. El test de Turing es uno de los más famosos

métodos de evaluación comprendidos en esta categoría, en el que se realizan ambas comparaciones (Hernández 2016).

2.3.1.2 Evaluación a través de los puntos de referencia de problemas

En este tipo de evaluación se define M como un conjunto de problemas. Por lo que la calidad de la evaluación depende de M y de cómo de exhaustiva sea la exploración del conjunto de problemas. Aunque existen otras condiciones en las que se puede ver comprometida la calidad de este tipo de evaluación. Una de estas condiciones puede ser que M sea un repositorio de problemas público no muy largo, lo que podría provocar que los sistemas puedan especializarse en ese conjunto de problemas. Para que no se de esta condición podemos utilizar los generadores de problemas.

Esta solución también nos plantea otro problema y es que no podemos evaluar los sistemas de IA con todo el conjunto M, por lo que debemos hacer un muestreo de M. Para realizar este muestreo hay dos opciones que pueden resultar útiles(Hernández 2016):

- Muestreo basado en información: suponiendo que tenemos una función que calcula la similitud entre dos problemas de M, podemos comparar los problemas coger aquellos que no sean similares y que mayores probabilidades tengan.

- Muestreo basado en la dificultad: un conjunto de problemas M puede tener desafíos difíciles y fáciles. Si utilizamos los desafíos complejos para sistemas malos o los desafíos fáciles para sistemas buenos no estaremos realizando una evaluación correcta. La idea es escoger un rango de dificultades para las cuales los resultados de la evaluación pueden ser informativos. Para ello es necesario definir una función de dificultad.

2.3.1.3 Evaluación por confrontación entre pares

La evaluación por confrontación entre pares consiste en evaluar un sistema dejándolo competir contra otro. Este tipo de evaluación es usual para juegos y también es común en la evaluación de sistemas multiagente. Los resultados de cada partida en la que participen los diferentes agentes puede servir como estimación sobre qué sistema es mejor. El inconveniente de este tipo de evaluación es que los resultados son relativos al oponente (Hernández 2016).

Este es el tipo de evaluación que se realizará para evaluar el comportamiento de los agentes creados en este proyecto, dado que es el más indicado para evaluar agentes que compiten en juegos.

2.3.2 Evaluación orientada a la habilidad

Existen determinados sistemas de IA que sobre los que la evaluación orientada a tareas no es muy apropiada, como podrían ser robots cognitivos, mascotas artificiales, avatares..., ya que de ellos se espera que sean personalizados por el usuario para realizar una variedad de tareas, y no están diseñados para realizar una tarea

determinada. Para estos tipos de sistemas podemos utilizar la evaluación orientada a la habilidad, que trata de evaluar las habilidades que posee un sistema, como pueden ser, razonamiento, habla, movimiento...

Este tipo de evaluación no está muy asentada en la actualidad pero tendrá un rol más relevante en el futuro (Hernández 2016).

3. Herramientas utilizadas

En este punto se describen las herramientas utilizadas para realizar los objetivos propuestos. Al inicio, la plataforma Project Malmo, sobre la que se desarrollan las tareas y se lanzan los agentes para que las resuelvan. Posteriormente las utilizadas para el desarrollo del código: Python y XML, como lenguajes de programación, Atom como editor de código y Github para compartir el código y llevar un control de versiones.

3.1 Malmo

La plataforma Project Malmo está diseñada para soportar una amplia gama de necesidades de experimentación y puede apoyar la investigación en robótica, visión por computadora, aprendizaje de refuerzo, planificación, sistemas multiagente y áreas relacionadas (Johnson et al. 2016).

Como puede observarse en la Figura 3.1, Malmo está compuesto por un mod para la versión Java, y un código que ayuda a los agentes de inteligencia artificial a observar y actuar dentro del entorno de Minecraft. Los dos componentes pueden ejecutarse en Windows, Linux o Mac OS, y los investigadores pueden programar sus agentes en diferentes lenguajes de programación (Project Malmo, 2015).

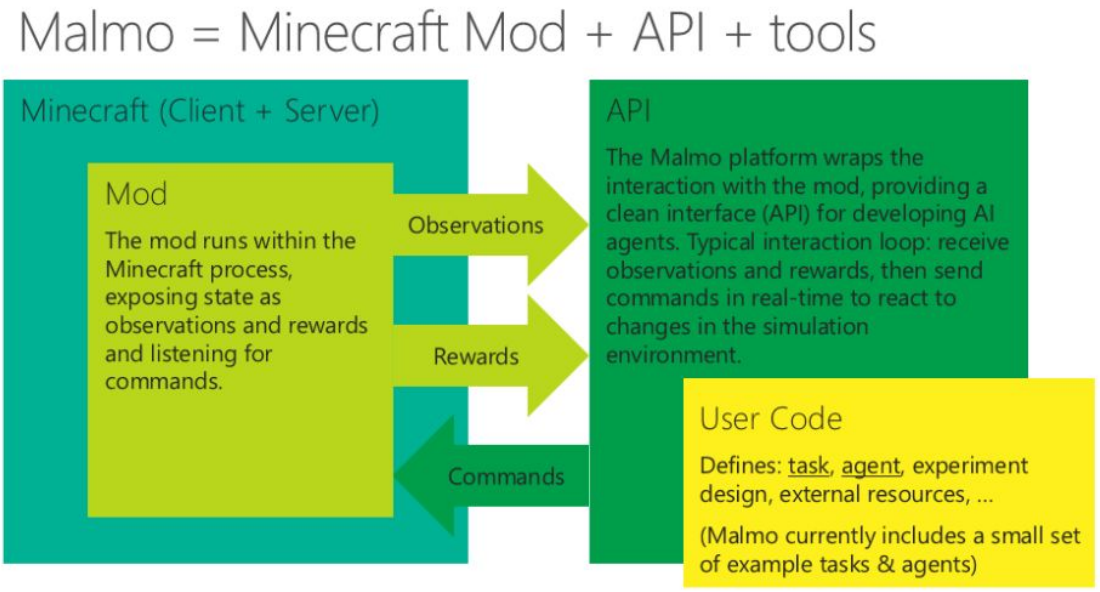


Figura 3.1. Malmo. Por Holfmann (2016)

En su github podemos encontrar el código de la plataforma, guías de instalación para los diferentes sistemas operativos, enlaces a su API y al esquema del XML para la definición de la misión, así como diferentes ejemplos y un sencillo tutorial que nos

pueden servir para aprender y posteriormente desarrollar nuestros propios desafíos. Como soporte también ofrece un blog y un chat de Gitter para poder ayudar a los desarrolladores interesados.

Además en marzo de 2017 propusieron un desafío de IA colaborativa en el que un agente debe aprender a colaborar con el ser humano para conseguir capturar un cerdo. Este desafío se propuso con el fin de desarrollar tecnología que sea capaz de colaborar con el ser humano para conseguir sus objetivos, ya que es uno de los objetivos a largo plazo de los investigadores de la IA.

3.1.1 Creación de la misión

La misión o tarea puede incluir la creación del mapa, recompensas, ítems a recoger y los diferentes tipos de observaciones, comandos y acciones permitidas para los agentes. Para la creación de la misión la plataforma Malmo nos ofrece dos opciones que pueden combinarse entre sí: usar un fichero XML para especificar la misión y utilizar la API proporcionada (Johnson et al. 2016).

La definición de la misión la podemos dividir en dos secciones importantes: una relacionada con el servidor y la otra relacionada con los agentes.

En la sección del servidor tenemos por un lado las condiciones iniciales, que permiten especificar el tiempo, paso del tiempo, meteorología, así como los generadores de monstruos y monstruos permitidos. Por otro lado tenemos los manejadores del servidor, en cuya sección podemos definir los aspectos relacionados con la creación del terreno, diferentes tipos de decoración del mismo y cuando finalizará la misión, pudiendo establecer el tiempo de misión y/o que la misión finalice cuando algún agente acabe.

En la sección de los agentes, al igual que en la sección del servidor tenemos por una parte las condiciones iniciales, que incluyen la posición inicial del agente y los objetos que dispondrá en su inventario. Por otra parte tenemos los manejadores del agente que incluyen los tipos de observaciones permitidas, movimientos y condiciones para las recompensas y finalización de la misión.

3.1.2 Creación de un agente

Para la creación de los agentes la plataforma Project Malmo nos ofrece una API con la que podemos definir el agente y el comportamiento que tendrá. La metodología a seguir sería:

En primer lugar, se inicializan el agente y la misión y se indica dónde se guardarán los datos de la misión (videos, recompensas, comandos…), para posteriormente asignar la misión al agente creado.

Una vez establecida la misión se procede a realizar un bucle mientras esta no haya finalizado en el que el agente obtiene las observaciones y recompensas y dependiendo de éstas se le envían los comandos para realizar las acciones necesarias y así finalizar la misión de manera exitosa.

Para consultar los comandos disponibles a enviar al agente podemos consultar la documentación disponible en el github de Malmo.

En la Figura 3.2 podemos ver el código de un ejemplo de agente que simplemente da vueltas y salta. Como podemos observar el código es bastante intuitivo y se necesitan pocas líneas para crear un agente y lanzar la misión que hayamos creado.

```
import MalmoAgent
import time

agent_host = MalmoPython.AgentHost()
my_mission = MalmoPython.MissionSpec("my_mission.xml")

my_mission_record = MalmoPython.MissionSpec("save.tgz")
my_mission_record.recordRewards()

agent_host.startMission( my_mission, my_mission_record )
// TO DO: wait for mission to start + check for errors

world_state = agent_host.peekWorldState()
while world_state.is_mission_running:
    // interpret world state
    world_state = agent_host.getWorldState()
    for reward in world_state.rewards:
        print "Summed reward:",reward.value
    for observation in world_state.observations:
        print "Observation:",observation.text
    // act
    agent_host.sendCommand( "move 1" )
    agent_host.sendCommand( "turn 0.5" )
    agent_host.sendCommand( "jump 1" )
    time.sleep(0.1)
```

Figura 3.2 Ejemplo simple en python. Por Holfmann (2016)

3.1.3 Ejemplos

Algunos de los ejemplos en Python ofrecidos en el github de Malmo relacionados con las tareas implementadas en este trabajo son:

- Reward_for_items_test.py: misión en la que un agente recoge distintos ítems.
- Tabular_q_learning.py: misión en la que un agente trata de alcanzar un bloque, en el que hay un diamante, sin caer a la lava.
- Multi_agent_test.py: misión en la que cuatro agentes deben coger manzanas y a su vez matar a todos los zombies.

Estos ejemplos han servido como base para implementar las tareas propuestas utilizando código de parte de ellos, adaptándolo y agregando nuevas funcionalidades.

3.1.3.1 Reward_for_items_test.py

En esta misión, en un terreno de 100x100 bloques, se crean 400 objetos en posiciones aleatorias cuyos premios varían en el rango [-2,2]. El agente, mediante movimientos continuos va recogiendo objetos durante 15 segundos. El movimiento que realiza depende de las recompensas recibidas, por lo que no se trata de ningún comportamiento inteligente.

Figura 3.3. Ejecución del ejemplo reward_for_items_test.py

3.1.3.2 Tabular_q_learning.py

En este ejemplo, para superar la misión, el agente debe alcanzar un bloque de lapislázuli en el que se encuentra un diamante. El terreno consiste en una serie de bloques de arena rodeados de lava y con algún hueco en los que el agente puede caer y morir, finalizando la misión insatisfactoriamente.

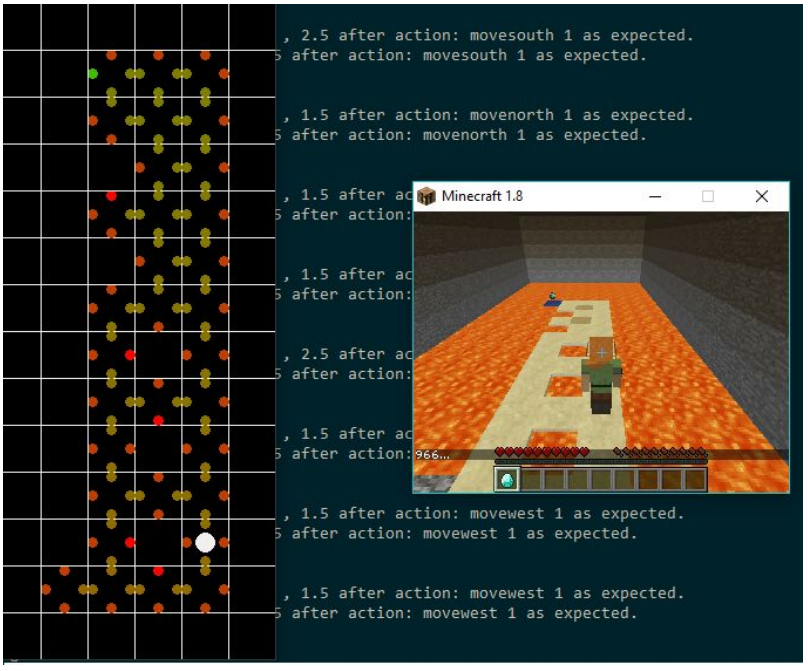


Figura 3.4 Ejecución del ejemplo tabular_q_learning.py

Para que el agente sea capaz de aprender a no caer en la lava y llegar hasta el lapislázuli se implementa una clase de agente con el algoritmo Q-Learning para el aprendizaje automático y la toma de decisiones. Así mediante ensayo y error el agente consigue aprender los movimientos óptimos para alcanzar su objetivo.

Además, como puede verse en la Figura 3.3, se muestra una ventana en la que se puede apreciar los movimientos realizados por el agente en todas las repeticiones y los resultados negativos (rojo) o positivos (verde) producidos por cada una de ellas.

3.1.3.3 Multi_agent_test.py

En esta misión los agentes deben conseguir el mayor número de puntuaciones posible para ganar. El terreno de juego es un cuadrado cerrado en que se crean el doble de manzanas y zombies que de agentes y se sitúan al azar.

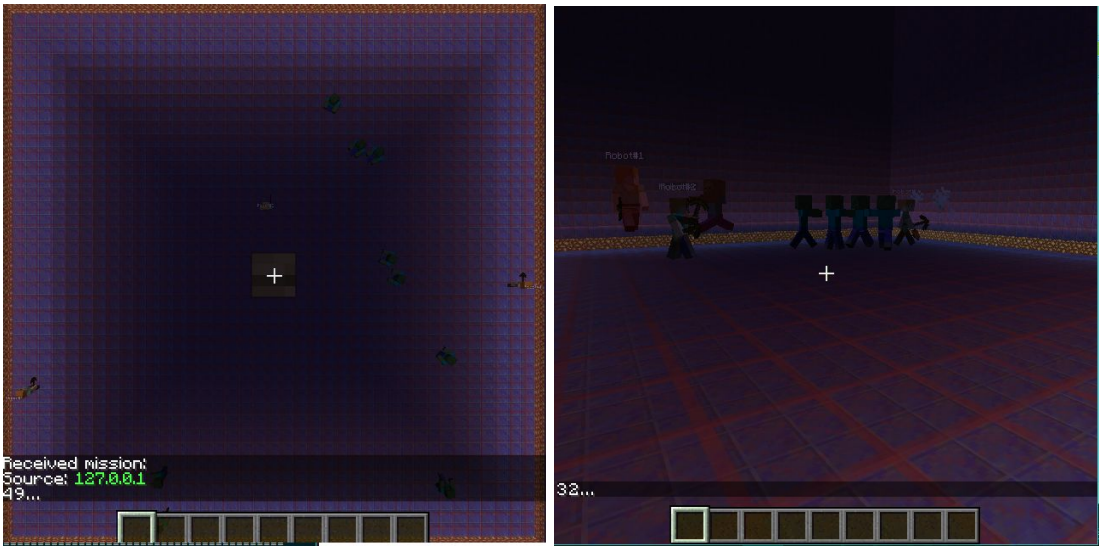


Figura 3.5 Ejecución del ejemplo multi_agent_test.py

Mientras la misión no haya finalizado y quede algún agente vivo a parte del observador, se recorren los agentes y dependiendo de las observaciones, se calcula el ángulo que debe girar y la velocidad para alcanzar un objeto o un zombie y se envían los comandos oportunos. Si en el punto de mira el agente encuentra un zombie se le envían comandos para atacar.

3.2 Lenguajes de programación

Para el desarrollo de las tareas, como lenguaje de programación se ha elegido Python por las siguientes razones, además de por la experiencia obtenida durante los años de estudio:

- Facilidad de escribir código a partir de un algoritmo, así como la legibilidad del mismo.

- Se trata de un lenguaje bastante poderoso, con el que se pueden diseñar grandes algoritmos sin la necesidad de crear tanto código como en otros lenguajes.
- Se importan librerías de forma rápida y sencilla.
- Dado que es un lenguaje dinámico no es necesario declarar las variables, lo que reduce la probabilidad de errores y hace más ágil la programación.

Para la definición de las misiones de los agentes ha sido necesario utilizar el lenguaje de marcas extensible (XML) ya que era el único disponible en la plataforma Project Malmo.

3.3 Atom

Para el desarrollo del código se ha utilizado el editor de textos Atom. Este permite una edición de manera sencilla y rápida gracias a su gran número de módulos, que ayudan a la edición y a la integración con git, mostrando las diferencias entre el código actual y el subido a la plataforma.

3.4 Github

Para la publicación del código y el control de versiones se ha utilizado github. Se trata de una plataforma de desarrollo en la que se pueden crear proyectos y almacenar el código de forma pública, permitiendo así la colaboración entre desarrolladores. Además también ofrece la posibilidad de almacenar el código de forma privada mediante una cuenta de pago.

Con el objetivo de colaborar con la plataforma Project Malmo y ofrecer ejemplos a aquellos desarrolladores que quieran comenzar a utilizar la plataforma se ha creado un proyecto público llamado Malmo en el que se ha subido el código de las 3 tareas y videos de ejecuciones para cada una de ellas, así como una breve descripción del mismo.

4. Tarea 1: Recolección de ítems

En esta sección se detallarán la implementación y pruebas de la tarea Recolección de ítems. En primer lugar, se describirán los aspectos importantes de la definición de la misión. En segundo lugar, se detallará el comportamiento de los agentes, basado en las percepciones del entorno, representadas en estados. En tercer lugar, se describirá la dinámica de la tarea. En cuarto lugar, se analizarán las pruebas realizadas para la evaluación de la IA y el correcto funcionamiento de la tarea. En último lugar, se describirán las dificultades encontradas.

Esta tarea consta de 2 a 4 agentes, posicionados en las esquinas de un terreno cuadrado, de dimensiones a especificar con el parámetro “dim”, rodeado de lava, que compiten para recoger el mayor número de ítems.

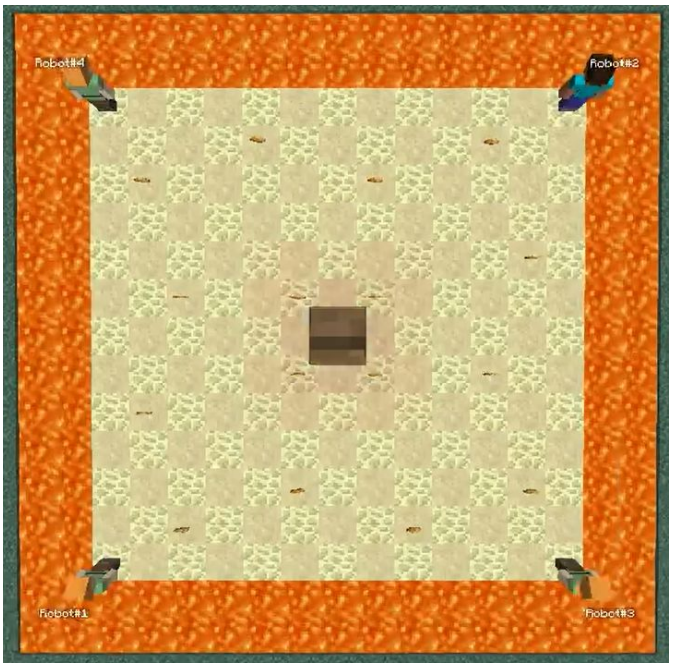


Figura 4.1 Instante inicial con 4 agentes.

Para implementar esta tarea se han realizado dos subtareas. Por una parte la creación de la misión, especificada en xml y por otra parte el comportamiento de los agentes, desarrollado en python.

4.1 Definición de la misión

Inicialmente, se especifica la creación del terreno, en esta parte se crea el mundo en el que van a actuar los agentes. En este caso está compuesto por un terreno cuadrado, de dimensiones a especificar en el parámetro “dim”, rodeado de lava. Además de una serie de patatas posicionadas aleatoriamente en el terreno, tantas como

especifiquemos en el parámetro “items”. Finalmente, un cubo de tipo “fence” donde se situará el agente vigilante, cuya función es mostrar lo que sucede.

Posteriormente, se crean los agentes que participarán en la tarea, posicionados en las esquinas contrarias del terreno, tantos como se indiquen en el parámetro “agents” (cuatro máximo), como podemos ver en la Figura 4.1. En la configuración de estos agentes se pueden destacar los parámetros referentes a las observaciones y los referentes a premios a obtener.

Referentes a las observaciones:

- “ObservationFromGrid”, de lado 3, que permite al agente saber de qué tipo son los bloques que tiene a un paso hacia cualquier dirección para tenerlo en cuenta a la hora de tomar una decisión y no caer en la lava.
- “ObservationFromNearByEntities”, cuyo rango en x y z es la dimensión del terreno. Este parámetro permitirá al agente saber en qué posición se encuentran los diferentes objetos a recoger, así como la posición del resto de agentes.
- “ObservationFromFullStats”, que en este caso se utilizará para saber la posición actual del agente.

Y los referentes a los premios a obtener:

- “RewardForCollectingItem”, con este parámetro se establece un premio de 10 puntos por recoger las patatas.
- “RewardFromTouchingBlockType”, con el que establecemos un premio de -100 por tocar la lava.
- “RewardForSendingCommand”, con el que establecemos un premio de -1 por cada comando enviado al agente, que nos servirá para que el agente realice el menor número de movimientos.

Finalmente, se crea el agente vigía, que será el que aportará la visión desde arriba para poder observar al resto de agentes.

4.2 Comportamiento de los agentes

Para permitir a los agentes resolver esta tarea se han implementado dos algoritmos: el Q-Learning como algoritmo de aprendizaje y otro de estrategia fija, basado en coger el ítem más cercano.

4.2.1 Q-Learning

Se ha implementado el algoritmo de Q-learning explicado anteriormente para dotar a los agentes de aprendizaje y permitirles resolver la tarea de la forma más óptima posible. Además se ha añadido el parámetro épsilon, que deberá ser un valor comprendido entre el 0 y 1, que se utiliza para realizar movimientos aleatorios con

probabilidad del valor de épsilon. El estado utilizado para representar el entorno está compuesto por una lista de 16 números cuyo significado es el siguiente:

- 0..3: En las 4 primeras posiciones del array se indica al agente si hay un bloque de tipo lava en las posiciones N,S,W,E respectivamente mediante los valores 1, cuando el bloque es de tipo lava y 0 cuando no.

- 4...7: En las siguientes cuatro posiciones del array se indica el número de ítems posicionados al N,S,W,E, respectivamente, cuya distancia máxima sea menor o igual que 2 en el eje 'x'(N-S) o 'z'(W-E) y su mínima sea menor que 2, lo que quiere decir que son alcanzables realizando un único movimiento, ya que no es necesario estar en la misma posición que el objeto para recogerlo, sino que puede estar a un bloque de distancia. Los valores de estas posiciones del array pueden ir desde 0 hasta el número de ítems existentes. La orientación a la que está el objeto es representada por el eje sobre el que más lejos está el ítem del agente, y en caso de ser iguales, se representará en la orientación sobre el eje 'x' (N-S). Por ejemplo, si el objeto está 3 bloques al norte y 2 al este se representará como situado al norte.

- 8...11: En estas cuatro posiciones se representan aquellos objetos que no están lo suficientemente cerca del agente para ser alcanzados mediante un movimiento.

- 12...15: En las últimas cuatro posiciones representamos el número de agentes situados al N,S,W,E. En este caso sí indicamos la orientación con respecto a los 2 ejes, de forma que si un agente está situado al norte y al este del agente se sumará 1 a los valores pertenecientes a las orientaciones N y E. Los valores en estas posiciones del array pueden ser desde 0 hasta el número de agentes.

Las acciones disponibles para cada agente son: moverse al norte, sur, este y oeste. Para que el tiempo de aprendizaje sea menor se ha modificado el algoritmo eliminando de las acciones disponibles aquellas que hagan caer en la lava al agente, comprobando en cada iteración si hay algún 1 en las cuatro primeras posiciones del array que representa el estado.

4.2.2 Ítem más cercano

Para que el agente siga una estrategia fija de recoger los objetos más cercanos se ha implementado un algoritmo sencillo cuya metodología es la siguiente:

Se recorren las observaciones, se busca el objeto que está más cerca del agente y se guardan las distancias en X y Z que hay que recorrer hasta llegar al el.

Una vez tenemos el objeto más cercano, se van realizando movimientos en el eje sobre el que hay mayor distancia hacia él.

Para este algoritmo no es necesario contemplar la posibilidad de caer en la lava ya que sus movimientos siempre van a ser acercarse al objeto, por lo que sería imposible caerse dado que no hay objetos situados en la lava.

4.3 Dinámica de la tarea

La dinámica de esta tarea es bastante sencilla, se establece un número de mapas, cada uno de ellos con objetos posicionados de forma aleatoria, y un número de misiones a realizar en cada uno de los mapas.

Una vez inicializada se recorre un bucle hasta que finalice la misión, pudiendo finalizar si un agente cae a la lava, por el tiempo de la misión establecido o si se han recogido todos los ítems. En este bucle se van recorriendo los agentes una y otra vez para que realicen el movimiento oportuno dependiendo de las observaciones recibidas y de los algoritmos utilizados.

Una vez se han obtenido todos los ítems los agentes abandonan la misión y son teletransportados a su posición inicial para iniciar la siguiente.

4.4 Pruebas realizadas

Para la evaluación de esta tarea se ha realizado una evaluación orientada a tarea en la que se han enfrentado a dos agentes, que deberán obtener el mayor número de objetos posible. Se han realizado dos tipos de pruebas:

En primer lugar, se han realizado pruebas con el objetivo de comprobar si la tarea ofrece algún tipo de ventaja a un agente frente al otro y si se ha implementado correctamente.

En segundo lugar, se ha enfrentado a dos agentes con algoritmos diferentes, uno con el algoritmo Q-Learning y otro con el Ítem más cercano, con el objetivo de comprobar cual de ellas ofrece mejores resultados y observar la curva de aprendizaje para el algoritmo Q-Learning.

4.4.1 Comprobar ventajas

Para poder comparar algoritmos que siguen estrategias diferentes y enfrentar a los agentes es necesario comprobar que el entorno en el que se van a enfrentar no ofrece ninguna ventaja a ningún agente. Para llegar a este punto se han realizado diferentes versiones.

En esta tarea los aspectos que pueden dar ventaja a un agente frente al otro son la disposición de los objetos y el turno.

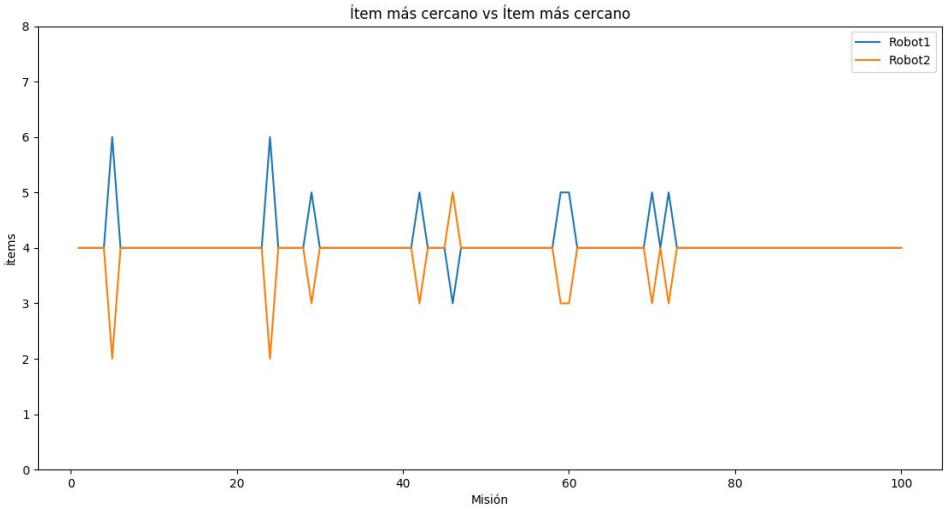
En cuanto a la disposición de los objetos se han realizado 3 versiones:

1. Se planteó posicionar los elementos de forma totalmente aleatoria en el tablero, pero tras realizar diferentes ejecuciones y ver que los resultados de los agentes no estaban muy igualados se decidió hacer otra versión.
2. Para que la disposición de los objetos estuviera más igualada se dividió el tablero en dos zonas, dejando la diagonal intermedia vacía, sobre las que se posicionaron el mismo número de objetos de forma aleatória. Tras probar esta

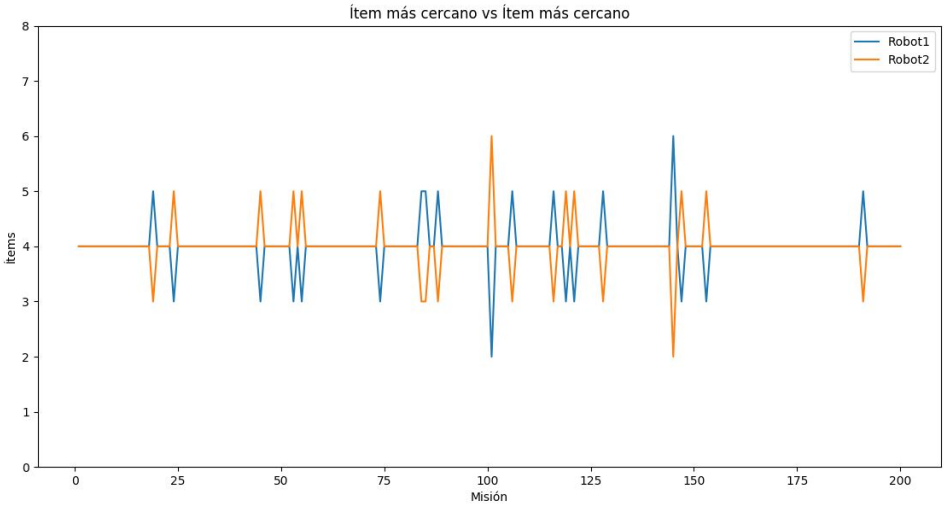
versión se comprobó que tampoco puede proporcionar datos fiables a no ser que se realicen números muy elevados de partidas.

3. Finalmente, se mejoró esta última versión, poniendo en una parte del terreno los objetos de forma aleatoria y en la otra posicionados a la inversa. De esta forma tenemos un terreno simétrico y más fiable a la hora de comparar estrategias.

Tras tener un tablero simétrico en el que comparar las estrategias de dos agentes se realizaron 100 partidas en las que el Robot 1 comenzaba primero, utilizando ambos una estrategia similar. Los resultados para esta prueba se muestran en la Gráfica 4.1. En ella podemos ver que el Robot 1 ha ganado 8 veces frente a 1 que ha ganado el Robot 2.



Gráfica 4.1. Comparación de dos agentes con el mismo algoritmo empezando siempre el Robot 1.



Gráfica 4.2. Comparación de dos agentes con el mismo algoritmo empezando cada vez uno.

Para corregir esta desventaja se ha modificado el algoritmo de la tarea para que en las partidas pares comience el Robot 1 y en las impares el Robot 2. Con esta modificación se ha realizado una prueba de 200 partidas en las que se enfrentan dos agentes con la misma estrategia cuyo resultado se muestra en la Gráfica 4.2. En ella podemos ver que el Robot 1 ganó 9 partidas, el Robot 2 ganó 10 y empataron en 181 ocasiones, por lo que podemos decir que ya no existe una clara ventaja del Robot 1 frente al Robot 2.

4.4.2 Comparación de estrategias

Para evaluar los distintos algoritmos implementados se ha realizado una prueba en la que se enfrentan dos agentes, uno con el algoritmo basado en una estrategia fija de recoger el ítem más cercano (Robot 1) y el otro con el algoritmo de Q-Learning (Robot 2), ambos explicados anteriormente.

Los parámetros referentes a la tarea son:

- Dimensión del terreno: 9
- Número de objetos: 8
- Número de agentes: 2

Los valores de los parámetros del Q-Learning para el agente 1 son: 0.1 para el valor de la tasa de aprendizaje, 1 para el valor del factor de descuento y 0.02 para épsilon.

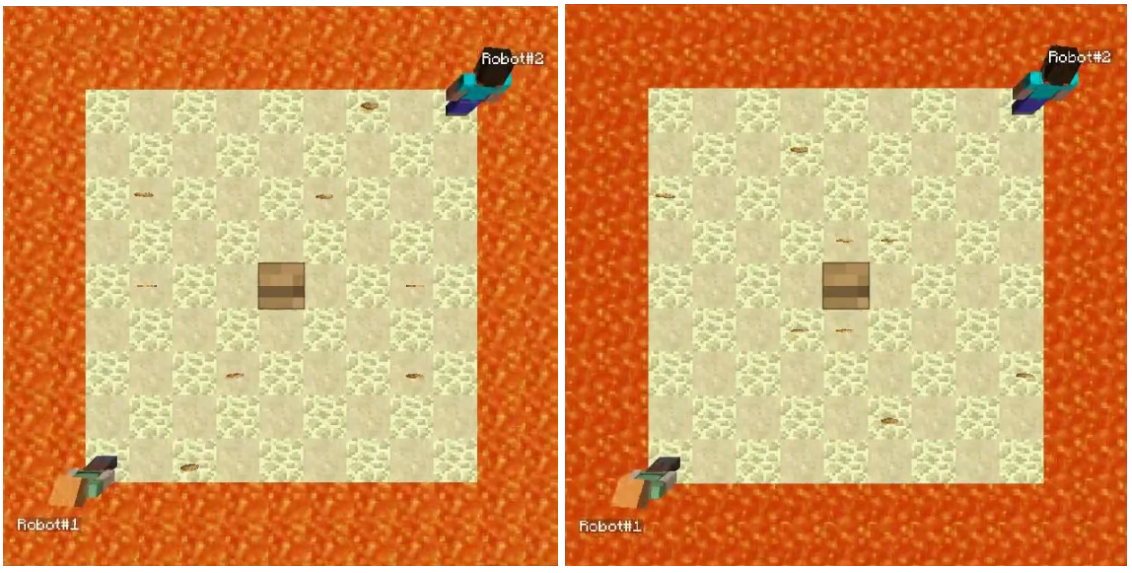
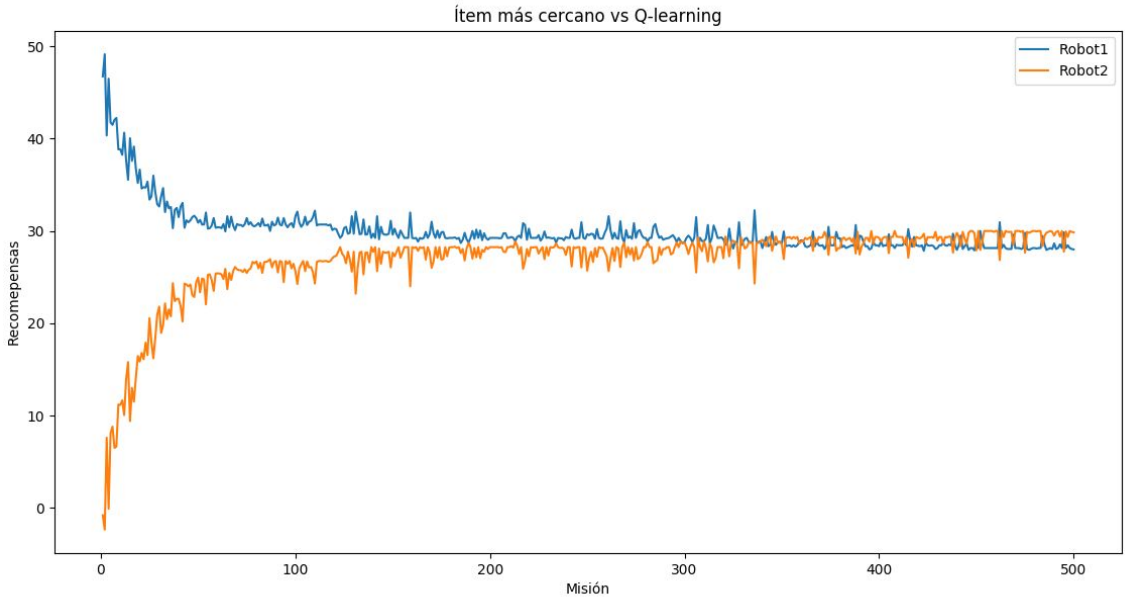


Figura 4.1. Instante inicial de la tarea en dos ejecuciones diferentes.

Como la generación de objetos en el terreno es aleatoria se han realizado 30 misiones diferentes para tener variedad en la disposición de los objetos y poder observar así en qué casos un agente recoge más ítems que otro. Estas 30 misiones se han realizado con 500 repeticiones cada una para el mostrar el aprendizaje del agente con el algoritmo Q-Learning.

Tras las 15000 ejecuciones se han procesado los resultados obtenidos para mostrarlos en forma de gráfica, calculando la media de las recompensas obtenidas, dando lugar a la Gráfica 4.3.



Gráfica 4.3. Comparación de 2 agentes similares con el algoritmo Q-Learning.

Como puede observarse en la Gráfica 4.3, El Robot 1 obtiene muchas más recompensas que el Robot 2 en la fase inicial del aprendizaje, pero en la fase final es el Robot 2 el que obtiene un número mayor de recompensas. En las primeras 150-200 iteraciones es cuando podemos ver una curva de aprendizaje mayor pronunciada, lo que significa que a partir de estas el Robot 2 ya ha encontrado una solución óptima o está muy cerca de encontrarla. Pero es a partir de 300 cuando recibe mayores recompensas que el Robot 1. Por lo que podemos decir que en general el Robot 2, tras la fase de aprendizaje ha encontrado mejores soluciones.

4.5 Dificultades encontradas

Al implementar esta tarea no se han encontrado muchas dificultades ya que es una tarea sencilla, en la que únicamente hay que colocar ítems de forma aleatoria. Aunque sí que se han encontrado algunos aspectos a tener en cuenta:

- En la implementación de la función aleatoria que genera los objetos se han quitado de las casillas disponibles para poner los objetos aquellas en las que está situado un agente y las casillas adyacentes, ya que para obtener el objeto no es necesario situarse en la misma casilla, sino que es suficiente con estar en una casilla adyacente.

- Al finalizar la tarea y reiniciar la misión los objetos son puestos en primer lugar y luego los agentes son situados en sus respectivas posiciones. Para que los agentes no queden situados en medio del terreno y les caigan los objetos,

variando así las características de la misión se ha decidido teletransportar a los agentes a sus posiciones iniciales antes de finalizar la misión.

5. Tarea 2: Cazador vs. Presa

En esta sección se detallarán la implementación y pruebas de la tarea Cazador vs. Presa. En primer lugar, se describirán los aspectos importantes de la definición de la misión. En segundo lugar, se especificará el comportamiento de los agentes, relacionado con el estado que representa el entorno. En tercer lugar, se describirá la dinámica de la tarea. En cuarto lugar, se analizarán las pruebas realizadas para la evaluación de la IA y el correcto funcionamiento de la tarea. En último lugar, se describirán las dificultades encontradas a la hora de implementar esta tarea.

La tarea Cazador vs. Presa consta de 2 agentes, posicionados en el centro de los lados del terreno, de dimensiones a especificar con el parámetro “dim”, uno enfrente del otro. Los agentes pueden tener los roles de cazador y presa, que alternan cada 15 pasos, siendo el cazador el que consigue 25 puntos si se mantiene a la misma distancia de la presa , 50 puntos si se acerca y 500 puntos por cada vez que caza a la presa, mientras que la presa recibe 25 puntos por mantenerse a la misma distancia, 50 por alejarse y -500 puntos si es cazado. Una vez se han dado caza, los agentes son teletransportados a su posición inicial, así hasta realizar 3 cambios de rol para que ambos agentes hayan sido cazador y presa durante 2 turnos.

Podemos distinguir los roles de los agentes ya que cuando un agente tiene el rol de cazador lleva una espada, mientras que cuando su rol es el de presa no lleva nada.

Para implementar esta tarea, al igual que en la anterior, se han realizado dos tareas principales, la definición de la misión y el comportamiento de los agentes, que se detallarán a continuación.

5.1 Definición de la misión

Los aspectos importantes en la definición de la misión son los siguientes:

Por una parte, la creación del terreno, en esta parte se crea el mundo en el que van a actuar los agentes, en este caso está compuesto por un terreno cuadrado, de dimensiones a especificar en el parámetro “dim”, rodeado de lava. Finalmente, un cubo de tipo “fence” donde se situará el agente vigilante, cuya función es mostrar lo que sucede en el terreno.

Por otra parte, se crean los dos agentes que participarán enla tarea, equipados con una espada en el inventario y posicionados en el centro de los lados del terreno, como se muestra en la Figura 5.1. En cuanto a la configuración de estos agentes se pueden destacar los parámetros referentes a las observaciones y los referentes a premios a obtener.

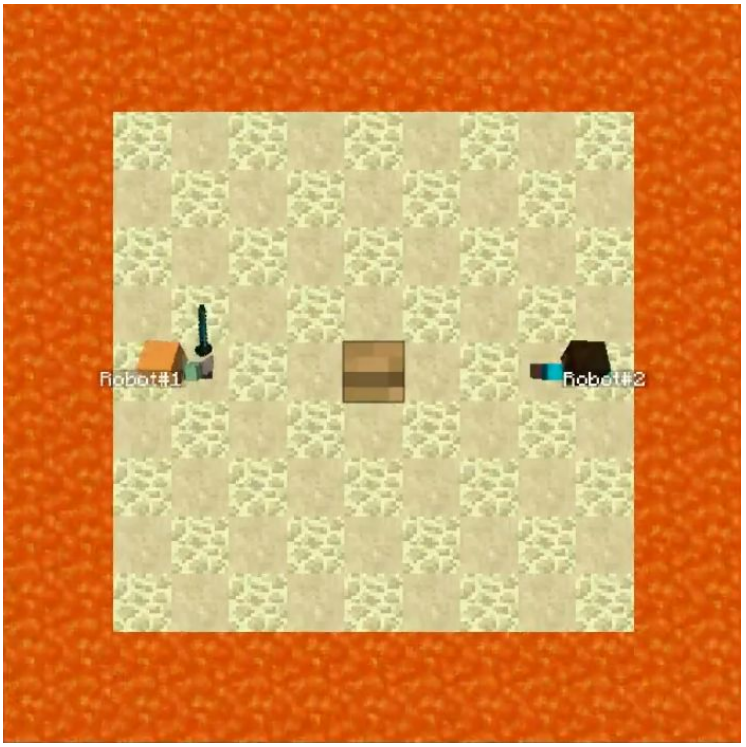


Figura 5.1. Situación inicial de los agentes en la tarea Cazador vs. Presa

Referentes a las observaciones:

- “ObservationFromGrid”, de lado 3, que permite al agente saber de qué tipo son los bloques que tiene a un paso hacia cualquier dirección para tenerlo en cuenta a la hora de tomar una decisión y no caer en la lava.
- “ObservationFromNearByEntities”, cuyo rango en x y z es la dimensión del terreno. Este parámetro permitirá al agente saber en qué posición se encuentra el agente rival.
- “ObservationFromFullStats”, que en este caso se utilizará para saber la posición actual del agente.

Y los referentes a los premios a obtener:

- “RewardFromTouchingBlockType”, con el que establecemos un premio de -1000 por tocar la lava.
- “RewardForSendingCommand”, con el que establecemos un premio de -1 por cada comando enviado al agente, que nos servirá para que el agente realice el menor número de movimientos.

En último lugar, se crea el agente vigía, que será el que aportará la visión desde arriba para poder observar al resto de agentes.

5.2 Comportamiento de los agentes

Para esta tarea se han implementado dos algoritmos: el Q-Learning que dota al agente de aprendizaje y otro de estrategia fija en el que el agente persigue cuando tiene el rol de cazador y huye cuando no lo tiene o está a punto de dejar de tenerlo.

5.2.1 Q-Learning

Se ha implementado el algoritmo de Q-learning explicado anteriormente para dotar a los agentes de aprendizaje y permitirles resolver la tarea de la forma más óptima posible. Además, como en la tarea anterior, se ha añadido la variable épsilon para que el agente pueda realizar movimientos aleatorios, lo que le permitirá explorar más posibles soluciones. El estado utilizado para representar el entorno está compuesto por una lista de 10 números cuyo significado es el siguiente:

- 0..3: En las 4 primeras posiciones del array se indica al agente si hay un bloque de tipo lava en las posiciones N,S,W,E respectivamente mediante los valores 1, cuando el bloque es de tipo lava y 0 cuando no.
- 4...7: En las 4 siguientes posiciones se indica la distancia a la que está el agente sobre cada orientación, N,S,W,E.
- 8: En esta posición se establece el valor 0 en caso de que el rol del agente sea cazador o 1 si es presa.
- 9: Si quedan menos de 3 pasos para el cambio de rol se pone a 1, siendo 0 en el caso contrario

Las acciones disponibles para cada agente son: moverse al norte, sur, este y oeste. Al igual que para la Tarea 1, se ha modificado el algoritmo para que las acciones permitidas sean aquellas que no lleven al agente a caer en la lava y así disminuir el tiempo de aprendizaje.

5.2.2 Perseguir y huir

El algoritmo implementado de estrategia fija trata de hacer que el agente huya cuando puede ser cazado y persiga en caso de ser el cazador. Para ello se sigue la siguiente metodología:

Si el agente tiene el rol de presa o tiene el rol de cazador pero quedan menos de 3 pasos para dejar de serlo, se recorren las observaciones para coger la posición del cazador y se comprueba en qué eje está más cerca. En caso de poder ir a la dirección contraria en ese eje porque no hay lava se realiza el movimiento para alejarse. Si no es posible porque caería en la lava, se comprueba la distancia sobre el otro eje para saber donde está situado el cazador y realizar un movimiento hacia el lado contrario si no hay lava o hacia el mismo ya que no hay otra opción. Si la distancia es la misma para los dos ejes se realiza un movimiento aleatorio que lo aleje del cazador entre los movimientos disponibles (aquellos que no conduzcan a la lava).

Si por el contrario, el agente tiene un rol de cazador, persigue a la presa. Para ello, igual que en el caso anterior, se recorren las observaciones para ver la distancia a la que está el otro agente y se aproxima realizando un movimiento hacia él en el eje sobre el que más lejano esté. En caso de estar a igual distancia en los dos ejes se selecciona uno al azar.

5.3 Dinámica de la tarea

En primer lugar, se establece un número de misiones a realizar para entrenar al agente y se inicia un bucle para ejecutarlas. En cada una de ellas, se establecen los roles iniciales de los dos agentes, siendo el Robot 1 cazador y el 2 presa para las misiones pares y al revés para las misiones impares. Cuando el rol ya se ha establecido se le envía un comando al agente para que seleccione la espada o no dependiendo de si es cazador o presa.

Una vez inicializada la tarea se recorre un bucle hasta que finalice la misión, pudiendo finalizar si un agente cae a la lava o cuando ambos hayan sido cazador dos veces . En este bucle se van recorriendo los agentes una y otra vez para que realicen el movimiento oportuno dependiendo de las observaciones recibidas y el rol establecido. Además, se van contando los pasos de los agentes para cambiar el rol de los mismos cuando hayan realizado 15 pasos. Cuando ambos han sido cazador dos veces se les envía un comando para que finalicen la misión. También se comprueba si los agentes se han alcanzado para que su siguiente acción sea teletransportarse a la posición inicial y procesar los diferentes premios.

5.4 Pruebas realizadas

Las pruebas realizadas para esta tarea son del mismo estilo que para la tarea anterior, se ha realizado una evaluación orientada a tareas en la que se enfrentan dos agentes entre sí.

Por una parte tenemos las pruebas para comprobar si la dinámica ofrece ventaja a un agente frente al otro. En caso de ser claramente ventajosa para algún agente se realizarán las modificaciones necesarias para igualar lo máximo posible la tarea y poder así comparar los algoritmos implementados de forma correcta.

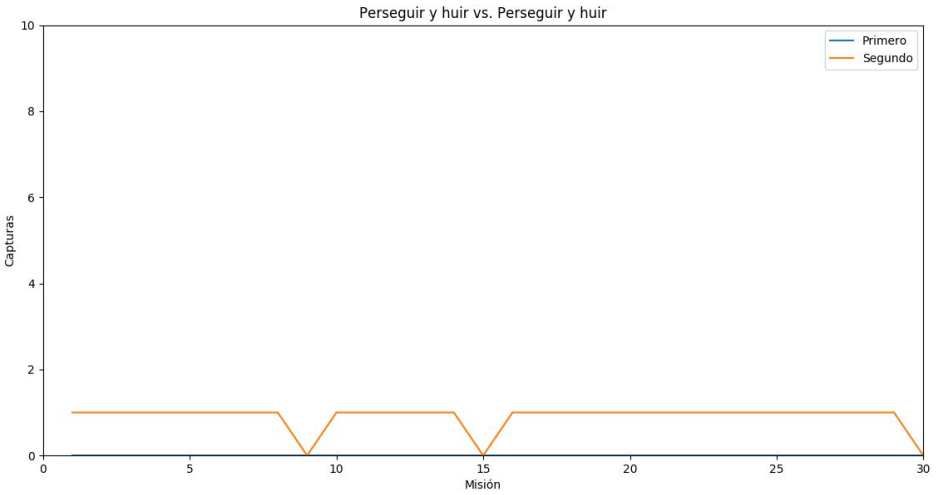
Por otra parte, una vez se ha encontrado la dinámica que ofrece la mínima ventaja, se enfrentan dos agentes con los algoritmos descritos en el apartado 5.2. Mediante este enfrentamiento se comprueba si han sido implementados correctamente y se comparan los resultados obtenidos.

5.4.1 Comprobar ventajas

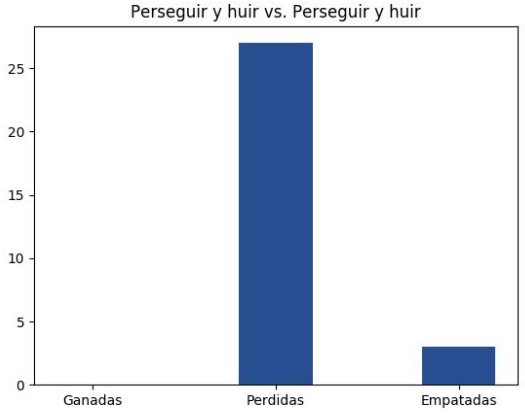
Como para la tarea anterior, resulta necesario comprobar si la dinámica de la tarea ofrece alguna ventaja a un agente frente al otro antes de comparar los algoritmos.

En esta tarea el único aspecto que puede dar ventaja es el turno, por lo que se ha realizado una prueba en la que los agentes se mueven por turnos y otra en la que se mueven simultáneamente.

En la prueba por turnos se realizaron 30 ejecuciones en un terreno cuadrado de 9 bloques de lado, en las que el agente que empezaba primero fue capturado en 27 de ellas mientras que el segundo no fue capturado en ninguna. Los resultados pueden verse en las Gráficas 5.1 y 5.2.

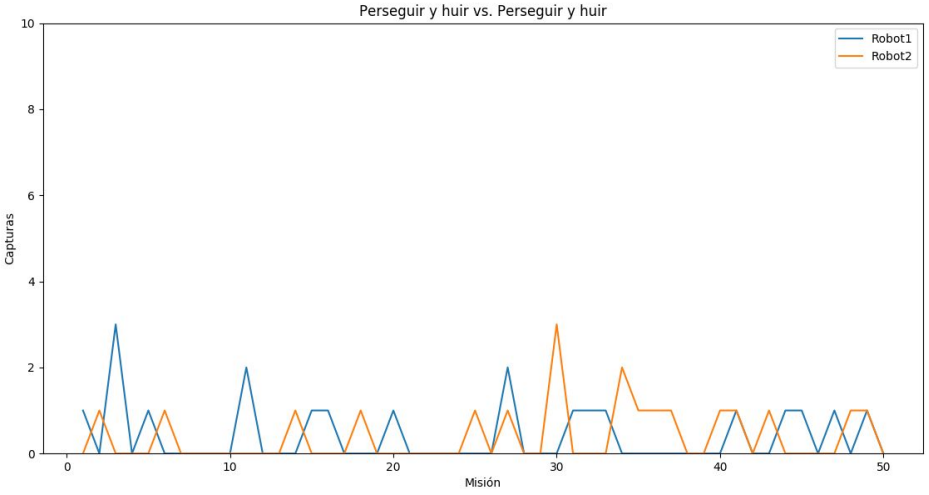


Gráfica 5.1. Comparación de dos agentes con el mismo algoritmo con movimientos por turnos.

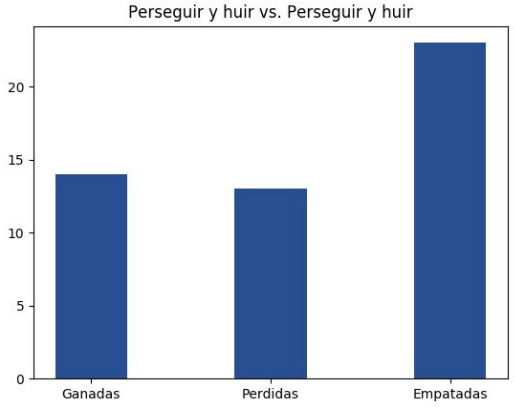


Gráfica 5.2. Partidas ganadas perdidas y empatadas del agente que empieza primero.

Observando los videos de las ejecuciones se ha comprobado que en un entorno con movimientos por turnos el que los realiza en segundo lugar puede acorralar al primero, por lo que captura a este en una gran mayoría de ocasiones. Para corregir esta ventaja se ha cambiado la dinámica, de forma que los movimientos de los agentes sean simultáneos. Con este cambio, se ha realizado una prueba de 50 ejecuciones, cuyos resultados se muestran en las Gráficas 5.3. y 5.4. En esta los agentes empatan en 23 ocasiones, el Robot1 gana en 14 partidas y el Robot2 en 13, por lo que podemos concluir que no existe una ventaja aparente con esta dinámica en un terreno de lado 9.



Gráfica 5.3. Comparación de dos agentes con el mismo algoritmo con movimientos simultáneos.



Gráfica 5.4. Partidas ganadas perdidas y empatadas del Robot1.

5.4.2 Comparación de estrategias

Para evaluar los algoritmos implementados para esta tarea se han realizado dos pruebas. En la primera interactúa un agente con el algoritmo Q-Learning con un parámetro fijo de épsilon de 0.02 y otro con el algoritmo de estrategia fija basado en perseguir cuando es cazador y huir cuando no lo es o va a dejar de serlo. En la segunda se varía el parámetro épsilon del Q-Learning, comenzando en 0.1 hasta llegar a 0 una vez realizadas el 90% de las ejecuciones, de manera que en el último 10% no realiza ningún movimiento aleatorio.

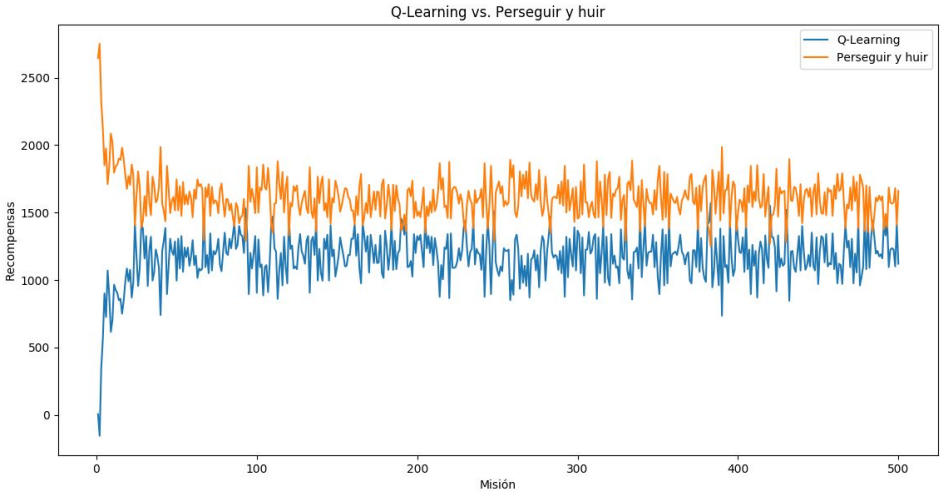
5.4.2.1. Q-Learning con épsilon fija vs. Perseguir y huir

La primera prueba realizada para comparar los algoritmos de Q-Learning y Perseguir y huir explicados anteriormente se ha ejecutado en un terreno de 9 bloques de lado. Los parámetros utilizados para el Q-Learning son: 0.1 para el valor de la tasa de aprendizaje , 0.5 para el factor de descuento, ya que se trata de un entorno no

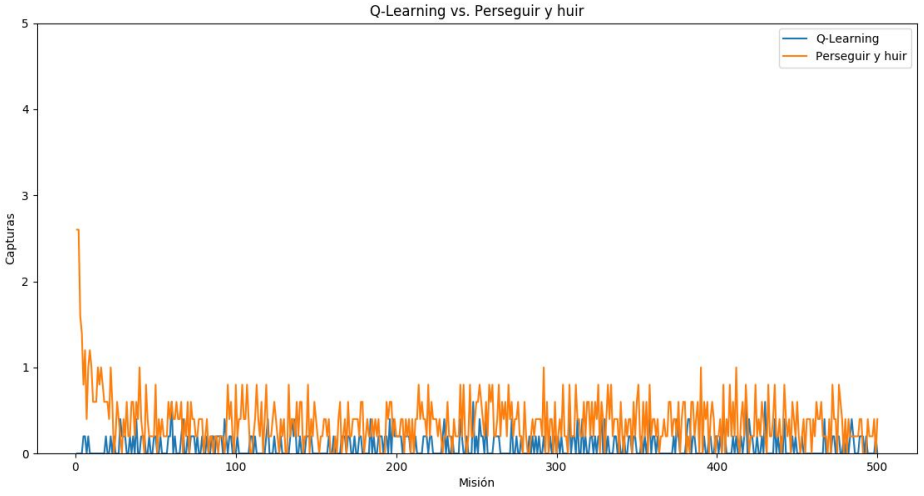
determinista y las recompensas dependen de las acciones que realice el otro agente, y 0.02 para épsilon.

Se han lanzado 5 ejecuciones, cada una de ellas con 500 iteraciones para poder observar el aprendizaje del agente que utiliza el Q-Learning.

En la Gráfica 5.5. se muestra la media de las recompensas obtenidas durante las 5 ejecuciones por ambos agentes. En ella se puede apreciar que tras 100 iteraciones el agente del Q-Learning llega a una fase de estancamiento y no logra superar al de estrategia fija, por lo que con estos parámetros podemos decir que no ha encontrado una solución que permita cazar al otro agente cuando tiene el rol de cazador y no ser cazado cuando es presa.



Gráfica 5.5. Media de las recompensas obtenidas por cada uno de los agentes.



Gráfica 5.6. Media de las capturas realizadas por cada uno de los agentes.

En la Gráfica 5.6. se muestran las capturas que han realizado cada uno de los agentes y como puede verse el agente del algoritmo Perseguir y huir consigue en casi todo momento estar por encima del agente del Q-Learning en cuanto a capturas, al igual que en recompensas.

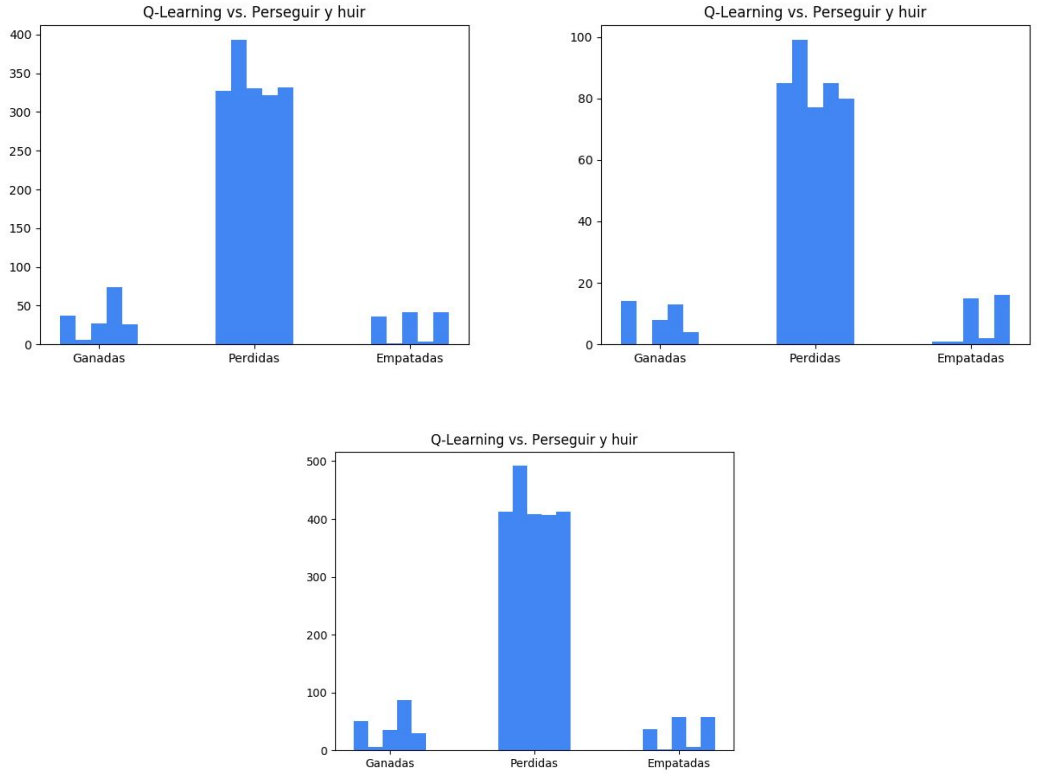


Figura 5.2. Partidas ganadas, perdidas y empatadas del agente con Q-Learning frente al de Perseguir y huir en las 400 primeras iteraciones, 100 últimas y total respectivamente.

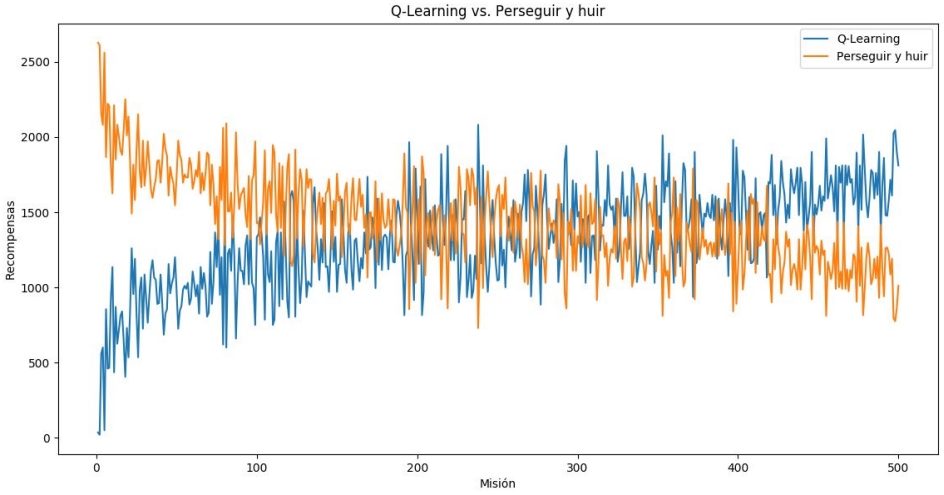
Por último, en la Figura 5.2 se pueden ver las partidas ganadas, perdidas y empatadas del agente del Q-Learning. En ella podemos ver que tanto en el conjunto global de las partidas como en la fase inicial y final del aprendizaje, pierde muchas más veces de las que gana o empata. Por lo que una vez analizadas todas las gráficas podemos decir que el agente del Q-Learning no ha conseguido dar con una solución mejor que la ofrecida por el algoritmo Perseguir y huir.

5.4.2.2. Q-Learning con épsilon variable vs. Perseguir y huir

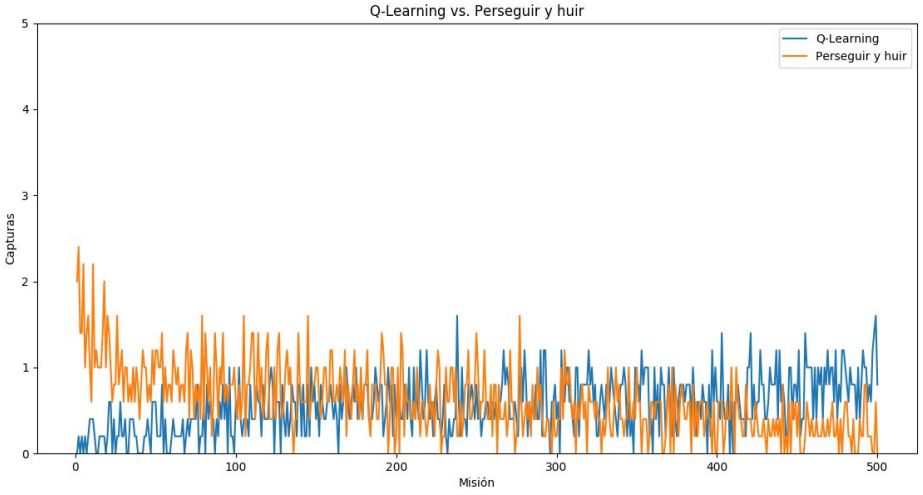
Como los resultados de la prueba anterior no han sido satisfactorios se ha modificado el algoritmo del Q-Learning de forma que la variable épsilon vaya disminuyendo su valor hasta llegar a 0 una vez realizadas el 90 % de las iteraciones. Mediante esta modificación se pretende que el agente realice una mayor exploración para encontrar una solución que le permita cazar y huir en los momentos oportunos. En esta prueba se enfrenta de nuevo a un agente con el algoritmo Q-Learning modificado con épsilon variable contra otro que utiliza el Perseguir y huir en un entorno similar al anterior.

Los parámetros del Q-Learning son los mismos que en la anterior prueba excepto el valor de épsilon que para esta es 0.1. Se han realizado 5 ejecuciones de 500 iteraciones cada una, al igual que en la prueba anterior.

En la Gráfica 5.7. podemos ver la media de las recompensas obtenidas durante las 5 ejecuciones por ambos agentes. Como puede apreciarse, a partir de las 200 iteraciones es cuando el agente del Q-Learning comienza a obtener mayores recompensas en algunas iteraciones y a partir de las 400 cuando supera en recompensas al agente del algoritmo Perseguir y huir. Por lo que podemos decir que al final de la fase de aprendizaje, con el valor de épsilon a 0, es cuando el agente del Q-Learning es capaz de realizar los movimientos oportunos para cazar y no ser cazado de mejor manera que el agente que utiliza el algoritmo Perseguir y huir.



Gráfica 5.7. Media de las recompensas obtenidas por cada uno de los agentes.



Gráfica 5.8. Media de las capturas realizadas por cada uno de los agentes.

En la Gráfica 5.8. se muestran las capturas realizadas por los agentes. En ella se puede observar como el número de capturas del agente del Q-Learning se va incrementando, mientras que el que utiliza el algoritmo Perseguir y huir va disminuyendo, llegando al punto en el que el primero caza en una gran mayoría de veces más de lo que es cazado.

Finalmente, en la Figura 5.3 se pueden ver las partidas ganadas, perdidas y empatadas del agente del Q-Learning. En ella podemos ver que en el conjunto global de las partidas pierde más veces de las que gana o empata, pero si dividimos las partidas en las 400 primeras y las últimas 100 podemos ver la diferencia entre la fase de aprendizaje y la fase final de este. En la primera gráfica de la figura vemos que el agente pierde más veces de las que empata o gana y esto es porque todavía está explorando distintas soluciones. En la segunda se observa todo lo contrario, y es que el agente gana más veces de las que empata o pierde por lo que podemos decir que ha aprendido la manera de actuar para cazar y no ser cazado por el otro agente.

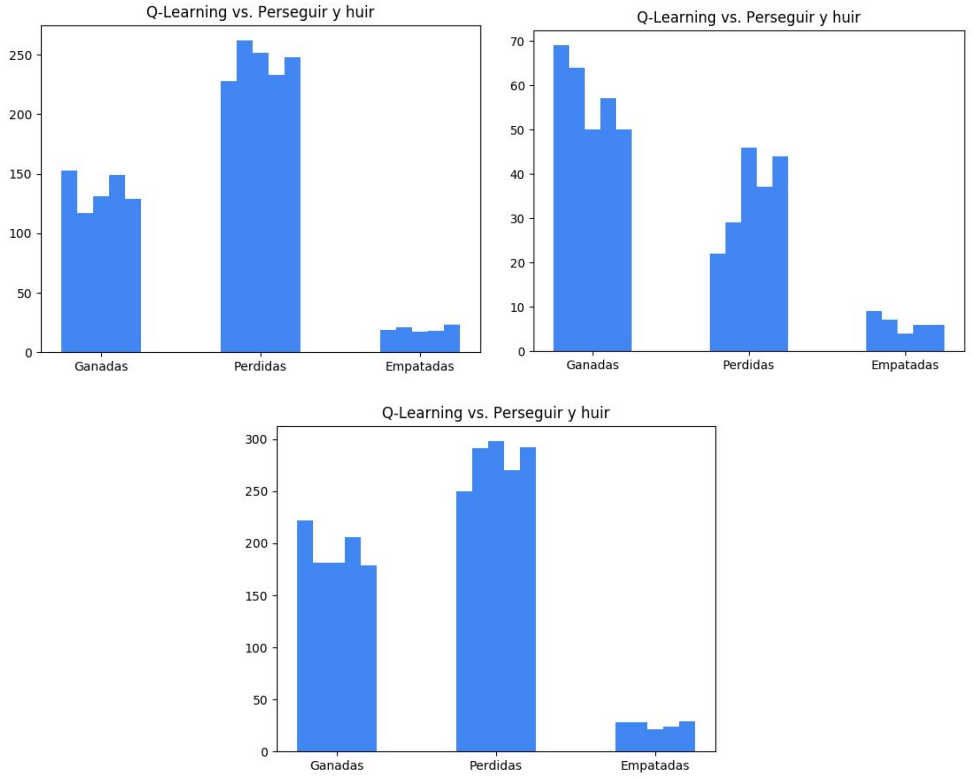


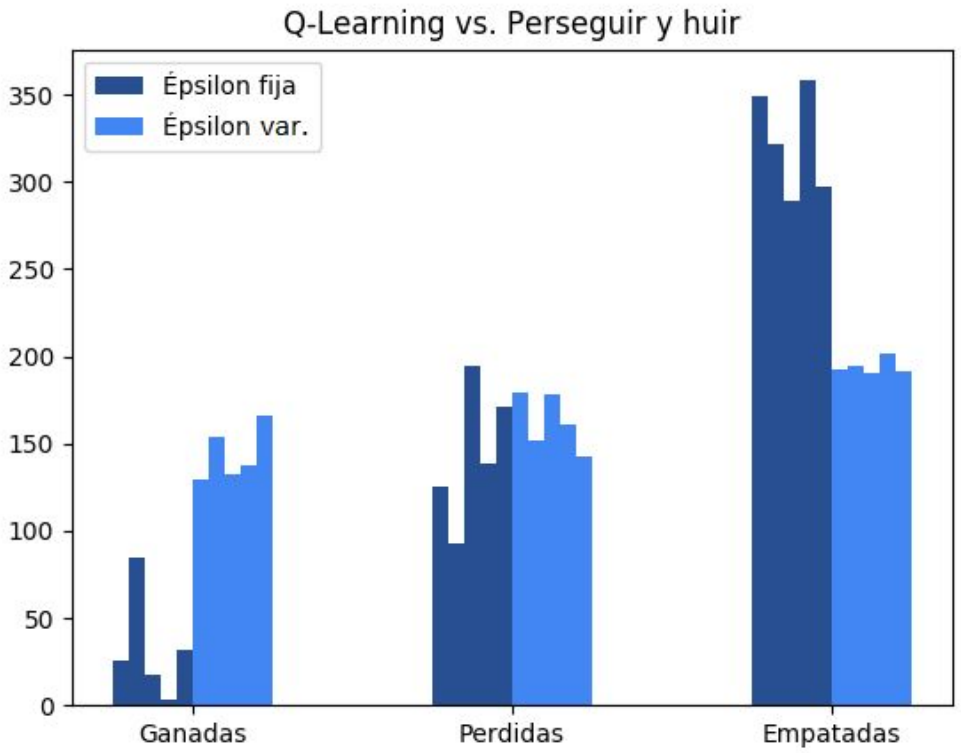
Figura 5.3. Partidas ganadas, perdidas y empatadas del agente con Q-Learning con epsilon variable frente al de Perseguir y huir en las 400 primeras iteraciones, 100 últimas y total respectivamente.

5.4.2.3. Q-Learning con épsilon variable vs. Q-Learning con épsilon fija

Una vez realizadas las pruebas en las que se comparan los algoritmos Q-Learning con épsilon fija y épsilon variable con el algoritmo Perseguir y huir se ha realizado una comparación entre los resultados obtenidos en las anteriores pruebas de las dos variantes del Q-Learning.

Como puede verse en la Gráfica 5.9 los resultados obtenidos con el algoritmo Q-Learning con épsilon variable son mucho mejores que los de épsilon fija. Aunque las partidas perdidas totales son bastante similares, el uso de épsilon variable le ha permitido al agente ganar más del triple de partidas de las que ha ganado con épsilon fija.

Finalmente, podemos concluir que la variación del valor de épsilon, disminuyendo su valor de manera lineal hasta llegar a 0 ha permitido al agente explorar un mayor número de soluciones que con una épsilon fija. Gracias a esa mayor exploración el agente ha podido encontrar los movimientos que le generan mayores recompensas, lo que se traduce a mayores capturas y ser menos veces capturado.



Gráfica 5.9. Comparación entre los Q-Learning con épsilon fija y variable.

5.5 Dificultades encontradas

Durante la implementación de esta tarea se han encontrado una serie de dificultades relacionadas con los premios a obtener y el rol de los agentes:

- La plataforma Project Malmo no nos ofrece ningún comando para dar un premio al tocar un agente, aunque si lo hubiera tampoco podríamos utilizarlo ya que el premio obtenido depende del rol de los agentes.

  Para solucionar este problema se ha tenido que comprobar las posiciones de cada uno de los agentes en cada iteración del bucle. Si los agentes están en la misma posición se le indica mediante variables a la función que calcula la

acción de cada agente que ha sido cazado o que ha cazado, dependiendo del rol de cada uno, por lo que su movimiento debe ser teletransportarse hacia su posición inicial. Además se le suman los correspondientes valores a las recompensas.

- Para el manejo del rol simplemente se ha utilizado una variable para indicarlo y otra como contador, inicializada a 15, que se ha ido decrementando con los pasos realizados, de forma que cuando el agente llega a 15 pasos se cambian los roles.

- Para poder diferenciar de forma visual los roles de los agentes y comprobar así que el juego está implementado correctamente se ha tomado la decisión de poner en el inventario de cada agente una espada y mediante el envío de comandos ofrecidos por la plataforma se le indica al agente si debe seleccionarla o no dependiendo de su rol.

6. Tarea 3: Conquista

En esta sección se detallarán la implementación y pruebas de la tarea Conquista. En primer lugar, se describirán los aspectos importantes de la definición de la misión. En segundo lugar, se detallará el comportamiento de los agentes, relacionado con el estado que representa el entorno. En tercer lugar, se describirá la dinámica de la tarea. En cuarto lugar, se analizarán las pruebas realizadas para la evaluación de la IA y el correcto funcionamiento de la tarea. En último lugar, se describirán las dificultades encontradas.

En la tarea Conquista pueden participar de 2 a 4 agentes, que serán posicionados en las cuatro esquinas del terreno cuadrado, de forma similar a la tarea Recolección de recoger ítems, de dimensiones a especificar con el parámetro “dim”. Cada uno de los agentes tiene en su inventario un material distinto, que utilizará para marcar el terreno por el que va, de forma que cuando acabe la misión, ganará el que más bloques tenga marcados con su material.

Para implementar esta tarea, al igual que en las anteriores, se han realizado dos tareas principales, la definición de la misión y el comportamiento de los agentes, que se detallarán a continuación.

6.1 Definición de la misión

Inicialmente, se especifica la creación del terreno, en esta parte se crea el mundo en el que van a actuar los agentes, en este caso se ha creado un terreno similar al de la tarea anterior (Cazador vs. Presa) que se puede observar en la Figura 6.1.

Figura 6.1. Instante inicial de la tarea Conquista.

Posteriormente, se crean los agentes que participarán en la tarea, equipados con sus respectivos materiales para marcar el terreno y posicionados en las esquinas contrarias, de igual forma que en la primera tarea implementada (tarea de recoger ítems), como podemos ver en la Figura 6.1. En la configuración de estos agentes se pueden destacar los parámetros referentes a las observaciones y los referentes a premios a obtener.

Referentes a las observaciones:

- "ObservationFromGrid", en este caso tenemos 3 observaciones de este tipo:
    - 'floor3x2x3': Nos indicará el tipo de bloques cercanos que tenemos a un paso en dos alturas, para y=-1 nos indicará el tipo de los bloques que están por debajo del agente, que serán los arena o lava, y para y=0 nos indicará el tipo de los bloques que están a la altura del agente, que serán los puestos por los demás agentes o aire si todavía nadie ha conquistado ese terreno.
    - 'floorBigArea': Nos muestra el tipo de los bloques situados a la altura del usuario (y=0) en un cuadrado de lado igual a la dimensión del terreno, estando el agente en el medio, de forma que verá todos los bloques si está situado justo en el centro del terreno.
    - 'subFloorBigArea': Es de igual tamaño que el anterior pero comprende los bloques situados en y=-1, nos servirá para saber si los bloques mostrados por la observación anterior que sean de tipo aire están situados en el terreno.
- "ObservationFromNearByEntities", cuyo rango en x y z es la dimensión del terreno. Este parámetro permitirá al agente saber en qué posición se encuentran los diferentes agentes.
- "ObservationFromFullStats", que en este caso se utilizara para saber la posición actual del agente.

Y los referentes a las recompensas a obtener:

- "RewardFromTouchingBlockType", de este tipo tenemos los siguientes premios:
    - -100 por tocar la lava.
    - 10 por tocar el terreno.
    - 20 por tocar cada uno de los materiales del resto de agentes.
- "RewardForSendingCommand", con el que establecemos un premio de -1 por cada comando enviado al agente.

Finalmente, se crea el agente vigía, que será el que aportará la visión desde arriba para poder observar al resto de agentes. A este se le ha añadido el parámetro “ObservationFromGrid” para que sea el que calcule los bloques que hay de cada agente al final de la partida.

6.2 Comportamiento de los agentes

Para esta tarea se han implementado dos algoritmos: el Q-Learning para dotar de aprendizaje al agente y otro que permite realizar al agente una estrategia de barrido de ida y vuelta, recorriendo todo el terreno en forma de ‘S’ como se muestra en la Figura 6.2.

6.2.1 Q-Learning

Al igual que para la tares anteriores se ha implementado el algoritmo de Q-learning explicado anteriormente para dotar a los agentes de aprendizaje y permitirles resolver la tarea de la forma más óptima posible. Además se le ha añadido una variable épsilon como en las tareas anteriores. El estado utilizado para representar el entorno está compuesto por una lista de 17 números cuyo significado es el siguiente:

- 0..3: En las 4 primeras posiciones del array, como en las tareas anteriores, se indica al agente si hay un bloque de tipo lava en las posiciones N,S,W,E respectivamente mediante los valores 1, cuando el bloque es de tipo lava y 0 cuando no.
- 4...7: En las cuatro posiciones siguientes representamos el número de agentes situados al N,S,W,E, está representación es similar a la realizada en la tarea de recoger ítems.
- 8-11: En estas posiciones se indica el tipo de bloque situado al N,S,W,E del agente respectivamente, siendo el valor 0 para el tipo aire, 1 para el material del agente y 2 para los materiales del resto de agentes.
- 12-15: En estas cuatro posiciones del array indicarán el número de bloques que tiene el agente en las direcciones N,S,W,E que son diferentes al suyo. De forma que si un bloque está situado a la misma distancia en el eje N-S que en el eje W-E, se considera como orientado al N-S.
- 16: En la última posición se almacena un indicador del tiempo que lleva el agente realizando la tarea, que servirá como multiplicador de recompensas, dándole mayor importancia a las obtenidas finales.

Las acciones disponibles para cada agente son: moverse al norte, sur, este y oeste.

6.2.2 Barrido

Para que un agente siga una estrategia de barrido continuo de ida y vuelta se ha implementado un algoritmo con la siguiente metodología:

Primero se inicializa una lista para insertar las acciones tomadas, que servirá para llevar la cuenta de el número de acciones realizadas y posteriormente se calculan tres listas:

- Cambio de dirección hacia el norte: en esta lista se insertan el número de movimientos tras los que el agente tiene que cambiar su dirección hacia el norte. Por ejemplo para un agente situado en la esquina inferior derecha de un escenario de lado 5 serían 0,10,20.

- Movimientos hacia el norte: en ella se insertan el número de movimientos tras los cuales el agente tendrá que moverse hacia el norte. Estos se calculan dependiendo de la lista anterior.

- Acciones hacia el este: número de movimientos tras los cuales el agente se mueve al este.

Una vez tenemos estas tres listas primero comprobamos si estamos en la ida o en la vuelta. Si estamos en la ida y el número de acciones realizadas está en alguna de ellas se realiza el movimiento adecuado, si no lo está se realiza un movimiento al sur. Si estamos en la vuelta se realiza el movimiento contrario.

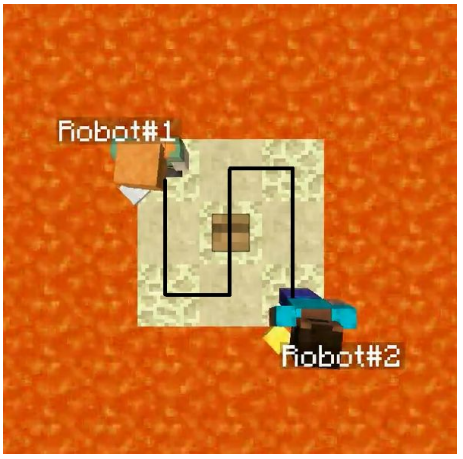


Figura 6.2 Recorrido del agente con estrategia de barrido.

6.3 Dinámica de la tarea

Como en las tareas anteriores, se establece un número de misiones para entrenar al agente y se inicia un bucle para realizarlas.

Una vez inicializada la tarea se recorre un bucle hasta que finalice la misión, pudiendo finalizar si un agente cae a la lava o si han realizado tantos pasos como la cantidad de bloques del terreno más la mitad. En este bucle se van recorriendo los agentes una y otra vez para que realicen el movimiento oportuno dependiendo de las observaciones recibidas. Al realizar un movimiento, para marcar el suelo con sus materiales los agentes saltan, pican, en caso de haber un bloque de otro agente y lanzan su bloque.

6.4 Pruebas realizadas

Para esta tarea únicamente se han realizado pruebas de comparación de algoritmos, ya que no se ha considerado necesaria realizar una prueba para comprobar ventajas, dado que el único elemento que puede dar ventaja es el turno. Para contrarrestar esta

desigualdad se han intercambiado los turnos de los agentes para cada partida, empezando cada vez uno.

Al igual que en la tarea de Cazador vs. Presa se han modificado las acciones permitidas para que los agentes no caigan en la lava y así minimizar los tiempos de ejecución de las pruebas.

En las pruebas de comparación de algoritmos se enfrenta a un agente Q-Learning con otro que realiza una estrategia de barrido. En la primera se utiliza un valor para épsilon fijo y en la segunda uno variable, para permitir una mayor exploración en el Q-Learning. Con ellas comprobaremos si los algoritmos se han implementado correctamente y cual es el que mejor resultados ofrece.

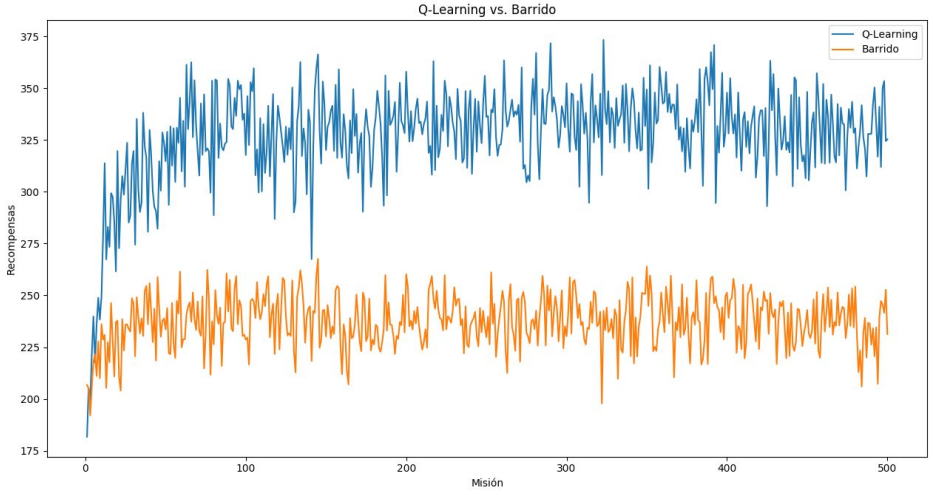
6.4.1 Q-Learning con épsilon fija vs. Barrido

La primera prueba realizada para esta tarea enfrenta a dos agentes. El primer agente, llamado Robot1, emplea el Q-Learning para la toma de decisiones y el segundo, llamado Robot 2, utiliza el Barrido.

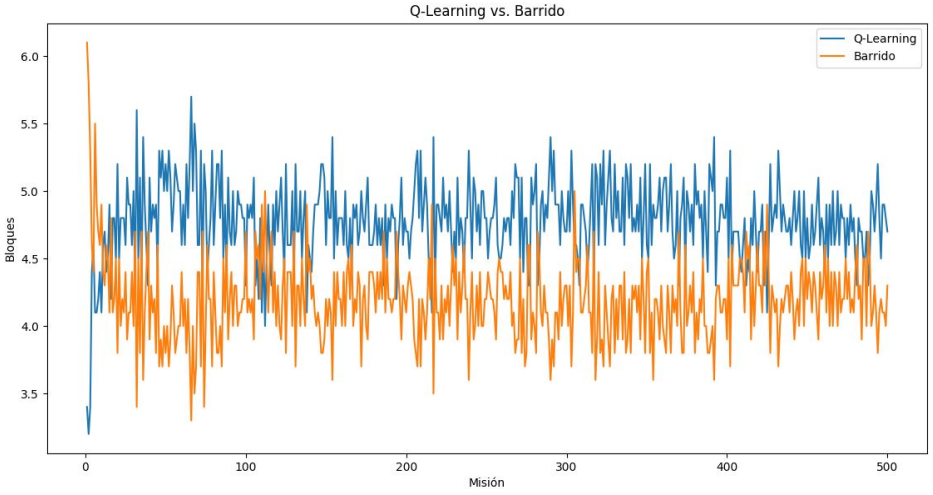
Este enfrentamiento se ha realizado en un terreno cuadrado de lado 3 y se han realizado 10 ejecuciones de 500 iteraciones cada una de ellas.

Los parámetros utilizados en el Q-Learning son: 0.1 para el factor de aprendizaje, 1 para el de descuento y 0.02 para épsilon.

En la Gráfica 6.1. se muestra la media de las recompensas obtenidas durante las 10 ejecuciones de cada uno de los agentes. Se puede apreciar la curva de aprendizaje desde la iteración 0 hasta la 150, después las recompensas recibidas van variando en menor forma, por lo que el agente está estancado y no consigue mejorar de forma significativa.



Gráfica 6.1. Media de las recompensas obtenidas por cada uno de los agentes.



Gráfica 6.2. Media de los bloques de cada uno de los agentes.

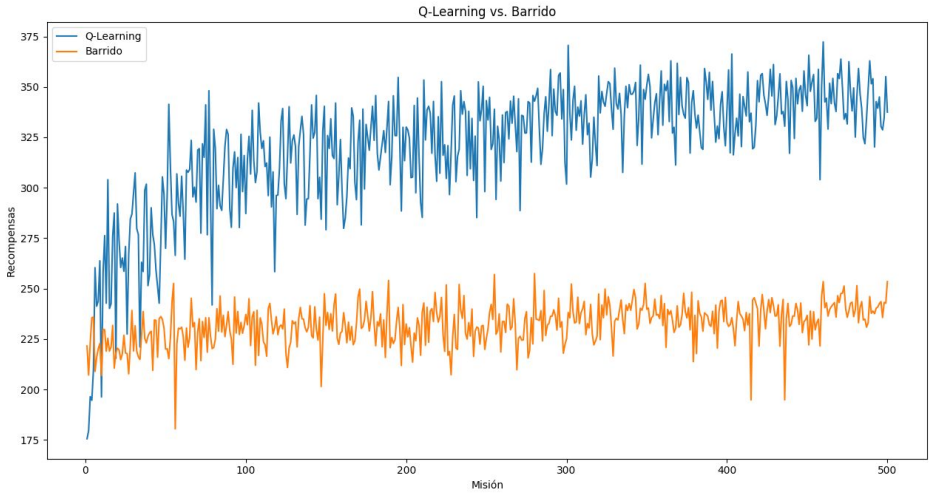
En la Gráfica 6.2 se muestra el número de bloques que quedan de cada agente al final de la partida. Mediante esta gráfica podemos ver que no le cuesta mucho más de 25 iteraciones al agente del Q-Learning superar al del algoritmo de barrido. En la fase de estancamiento vemos que la mayoría de las partidas acaban con 5 bloques para el Q-Learning y 4 para el Barrido. Aunque el Q-Learning haya dado mejores resultados, no son mucho mejores a los del Barrido por lo que se realizará otra prueba variando el parámetro épsilon del Q-Learning, intentando de esta forma mejorar los resultados y encontrar una diferencia algo mayor entre los algoritmos.

6.4.2 Q-Learning con épsilon variable vs. Barrido

Como pasó en la tarea Cazador vs. Presa, tras una primera prueba comparando ambos algoritmos, se ha visto que la diferencia entre ellos no ha sido muy significativa, por lo que se ha modificado el parámetro épsilon del Q-Learning para permitir una mayor exploración. En esta prueba se disminuye linealmente el valor de épsilon hasta llegar a 0 al 90% de las iteraciones.

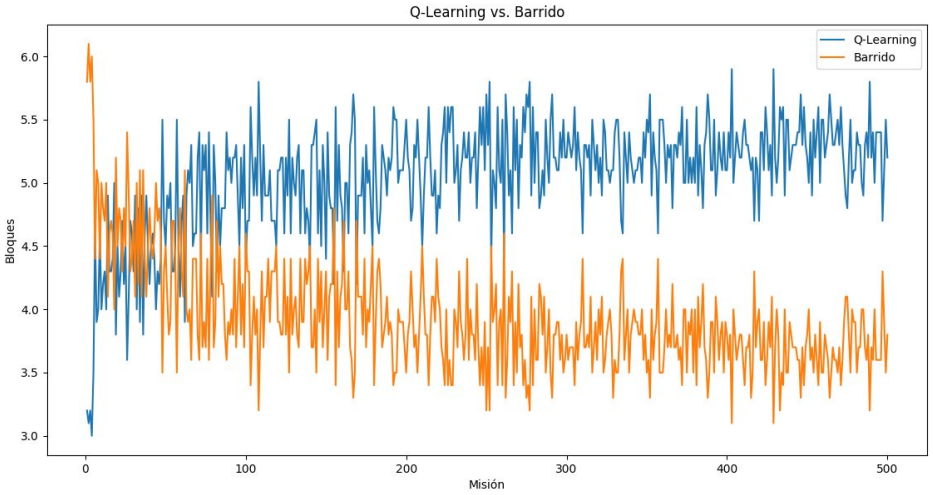
Los parámetros del Q-Learning son los mismos que en la anterior prueba excepto el valor de épsilon que para esta es 0.1. Se han realizado 10 ejecuciones de 500 iteraciones cada una, tal y como se hizo en la prueba anterior.

En la Gráfica 6.3. podemos ver la media de las recompensas obtenidas en las 500 iteraciones de las 10 ejecuciones. En este caso vemos una curva más prolongada, sin una zona de estancamiento clara, por la variación de épsilon, siendo a partir de la iteración 450 cuando obtiene las recompensas más altas. El agente del Q-Learning supera en una fase muy temprana las recompensas obtenidas por el agente del Barrido, aumentando cada vez más sus diferencias y llegando a una ventaja considerable.



Gráfica 6.3. Media de las recompensas obtenidas por cada uno de los agentes.

En la Gráfica 6.4 se muestra el número de bloques que quedan de cada agente al final de la partida. En este caso es a partir de las 50 iteraciones cuando el Q-Learning da mejores resultados que el Barrido. Durante la fase final del aprendizaje, con valores de épsilon muy bajos o nulos, es cuando vemos ya una clara diferencia entre los bloques puestos por ambos agentes. Por lo que como era de esperar, podemos decir que el Q-Learning ha dado mejores resultados en esta tarea que el algoritmo de barrido.



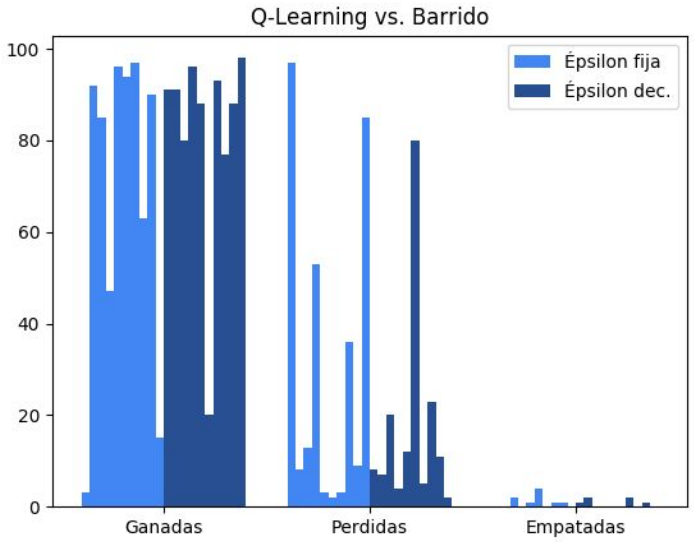
Gráfica 6.4. Media de los bloques de cada uno de los agentes.

6.4.3 Q-Learning con épsilon variable vs. Q-Learning con épsilon fija

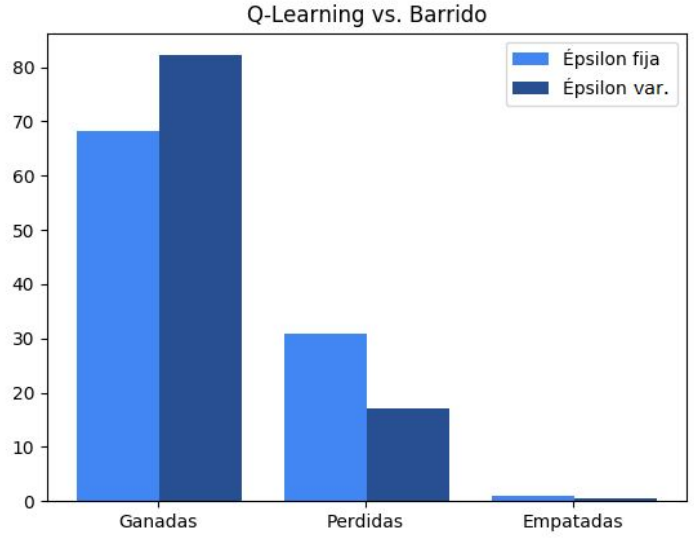
Tras realizar las comparaciones entre las versiones del Q-Learning y el Barrido se ha realizado una comparación entre los resultados obtenidos por las dos variantes del Q-Learning.

Como puede verse en las Gráficas 6.5 y 6.6 los resultados obtenidos por el Q-Learning con épsilon variable son mejores a los obtenidos con épsilon fija. Debido a una mayor exploración de las posibles soluciones a la tarea, mientras que con épsilon fija el agente pierde más partidas de las que gana en 3 de las 10 ejecuciones, con épsilon variable únicamente pasa en una ejecución.

Con estos resultados podemos concluir que para esta tarea, igual que para la tarea anterior, el algoritmo Q-Learning funciona mejor utilizando un valor variable para el parámetro épsilon. Este era el resultado esperado ya que al explorar más opciones es más probable que se encuentre una mejor solución y el agente no se quede estancado en una solución que no sea óptima.



Gráfica 6.5. Partidas ganadas, perdidas y empatadas con épsilon fija y variable en las últimas 100 iteraciones.



Gráfica 6.6. Media de partidas ganadas, perdidas y empatadas con épsilon fija y variable en las últimas 100 iteraciones.

6.5 Dificultades encontradas

Las dificultades encontradas a la hora de implementar esta tarea están principalmente relacionadas con marcar el suelo.

La primera idea fue utilizar diferentes bloques de tamaño normal, de forma que el agente reemplazara el situado a sus pies por el suyo. Para realizarlo se enviaban al agente los comandos de saltar, picar y poner el bloque (en caso de no estar encima de un bloque propio). Una vez implementado, se observó el comportamiento de los agentes y se comprobó que no funcionaba correctamente en todas las iteraciones. Para corregirlo se insertaron sleeps entre los envíos de los comandos, pero tras cambiar los valores de estos varias veces y ver que no funcionaban siempre se optó por pensar en otra solución.

Tras investigar los tipos de bloques existentes en Minecraft, se encontraron unos tipos de bloques llamados placas de presión, en los que podíamos encontrar 4 variedades para asignarlas a los 4 agentes y así poder distinguir cada uno de ellos. Estos bloques son de tamaño mucho menor, por lo que resultó más fácil utilizarlos para marcar el terreno.

Una vez solucionado el problema de poder poner los bloques de cada agente correctamente se observó que en determinadas ocasiones el agente gastaba los 64 bloques, que son el máximo disponible en cada hueco del inventario, y por lo tanto dejaba de marcar el terreno. Para solucionar este problema se llenó también el segundo hueco. De esta forma, se ha utilizado un comando para mandar los objetos del segundo hueco al primero para que el agente no se quede sin bloques.

7. Conclusiones y trabajo futuro

En este apartado se darán una serie de conclusiones a las que se han llegado tras la realización del trabajo y se describirán diferentes tareas como trabajo futuro.

7.1 Conclusiones

El trabajo realizado en este proyecto se enfocó en la creación de tareas competitivas multiagente en Minecraft y posteriormente en implementar un algoritmo que permitiera a los agentes aprender, mediante el uso de recompensas, para realizar las acciones que le lleven a resolver cada una de las tareas de la manera más óptima posible. Se eligió el algoritmo Q-learning ya que aunque es de los más antiguos, es uno de los más conocidos y ofrece buenos resultados, siendo menos complejo que otros algoritmos, como podrían ser las redes neuronales. Además, fue necesario el desarrollo de otros algoritmos de estrategia fija para compararlos con el Q-Learning.

Para realizar el objetivo principal, se pensaron tres tareas competitivas que pudieran realizarse en el mundo de Minecraft, que fueron: Recolección de ítems, Cazador vs. Presa y Conquista.

Una vez definidas las tareas a realizar se pasó a la fase de implementación. Para ello ha sido necesario aprender a utilizar la plataforma Project Malmo y mejorar los conocimientos de Python. Tras este aprendizaje se consiguió implementar las tareas planteadas de forma satisfactoria.

Posteriormente, se implementaron diferentes versiones del algoritmo Q-Learning adaptadas a cada tarea y otros de estrategia fija para compararlos con este.

Finalmente, con las tareas y algoritmos implementados se han realizado una serie de pruebas orientadas a tarea para evaluar el funcionamiento de los algoritmos y las características de cada tarea.

Tras analizarlas podemos concluir que los algoritmos de aprendizaje automático, en este caso el Q-Learning, funcionan mejor o igual que los de estrategia fija diseñados, ya que con los parámetros adecuados son capaces de encontrar mejores soluciones o iguales a las dadas por los algoritmos de estrategia fija. Además, ofrecen la ventaja al desarrollador de no tener que pensar una estrategia para cada tarea, únicamente necesitamos saber representar la información del entorno y conocer las acciones que nos acerquen a una solución para recompensar al agente cuando las realice.

Como aspecto negativo podemos decir que el mayor problema del Q-Learning es la exploración. Como este algoritmo funciona en base a ensayo y error, puede darse el caso, sobretodo en tareas muy complejas, en el que el agente no explore todas las

soluciones o tarde mucho tiempo en hacerlo, por lo que puede no encontrar la solución óptima.

7.2 Trabajo futuro

A lo largo del proyecto se han ido descubriendo diferentes mejoras y trabajos a realizar en el futuro.

Las posibles mejoras y trabajos futuros similares para cada tarea podrían ser:

- Cambiar las posiciones iniciales de los agentes y dimensiones del terreno.
- Hacer diferentes pruebas cambiando los valores de los parámetros del Q-Learning, el estado que define el entorno, así como los valores de las recompensas para encontrar aquellos que ofrezcan mejores resultados.
- Implementar más algoritmos para poder compararlos entre sí.

Sobre la tarea Recolección de ítems:

- Realizar las pruebas con más agentes.

Sobre la tarea Cazador vs. Presa:

- Hacer variaciones en el comportamiento cuando el cazador caza a la presa, actualmente cada uno de ellos se teletransporta a su posición inicial. Podría teletransportarse a uno de ellos o a ninguno.
- Cambiar el número de pasos por rol.

Sobre la tarea conquista:

- Realizar las pruebas con más agentes.
- Aumentar el número de pasos para finalizar la tarea.

8. Bibliografía

Atom. (2017). A hackable text editor for the 21st Century. Disponible en: https://atom.io/ [Consulta: 5 Julio 2018].

Hernández J. (2016). Evaluation in artificial intelligence: from task-oriented to ability-oriented measurement. Springer Science+Business Media Dordrecht.

Hoen P., Tuyls K., Panait L., Luke S., y La Poutré J. (2006). An Overview of Cooperative and Competitive Multiagent. Descargado de: <https://cs.gmu.edu/~sean/papers/LAMAS05Overview.pdf>

Holfmann K. (2016). Project Malmo. Descargado de <https://upvedues-my.sharepoint.com/personal/jorallo_upv_edu_es/_layouts/15/WopiFrame.aspx?docid=0a947d6bded684dfa8798febb46d14aba&authkey=AaBa_qxhDNEjsp4Z2MPtiZo&action=view>

Huys Q., y Cruickshank A., Seriès P. (2014) Reward-Based Learning, Model-Based and Model-Free. Translational Neuromodeling Unit, Institute of Biomedical Engineering, ETH Zürich and University of Zürich, Zürich, Switzerland. Department of Psychiatry, Psychotherapy and Psychosomatics, Hospital of Psychiatry, University of Zürich, Zürich, Switzerland.b Department of Psychiatry, Psychotherapy and Psychosomatics, Hospital of Psychiatry, University of Zürich, Zürich, Switzerland.

Ishii S., Fujita H., Yamazaki T., Matsuda J. y Matsuno Y. (2001) A Reinforcement Learning Scheme for a Partially-Observable Multi-Agent Game. Machine Learning, 59, 31–54.

Johnson M., Hofmann K., Hutton T., Bignell D. (2016) *The Malmo Platform for Artificial Intelligence Experimentation.* Proc. 25th International Joint Conference on Artificial Intelligence, Ed. Kambhampati S., p. 4246. AAAI Press, Palo Alto, California USA. <https://github.com/Microsoft/malmo>

Leyton-Brown K. y Shoham Y. (2008). Essentials of Game Theory. Morgan & Claypool.

Liu J. and Wu J. (2001). Multi-Agent Robotic Systems. CRC Press.

Maess P. (1995). Artificial life meets entertainment: Life like autonomous agents, Communications of the ACM, vol. 38, no. 11, pp. 108-114. Descargado de <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.471.5463&rep=rep1&type=pdf>

Murphy K. (1998). A brief introduction to reinforcement learning.

Project Malmo, (2015). Descargado de <https://www.microsoft.com/en-us/research/project/project-malmo/>

Q-Learning. (Sin fecha). En Wikipedia. <https://en.wikipedia.org/wiki/Q-learning> [Consula: 10 de mayo 2018]

Stone P., Veloso M. (2000). Multiagent systems: A survey from a machine learning perspective", Autonomous Robots, vol. 8, no. 3, pp. 345–383

Sutton R., Barto A. (2012). Reinforcement Learning: An Introduction. Cambridge, Massachusetts, London, England.

Tokic M., Palm G. (2011). Value-Difference based Exploration: Adaptive Control between epsilon-Greedy and Softmax. Institute of Neural Information Processing, University of Ulm, 89069 Ulm, Germany. Institute of Applied Research, University of Applied Sciences Ravensburg-Weingarten, 88241 Weingarten, Germany