UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Modelo de propagación de malware basado en aprendizaje por refuerzo

Presentado por Diego García Muñoz
en Universidad de Burgos — 22 de septiembre de 2022
Tutores: Bruno Baruque Zanón y Roberto Carlos Casado Vara

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

D. Bruno Baruque Zanón, profesor del departamento de Ingeniería Informática, área de Ciencia de la Computación e Inteligencia Artificial, y D. Roberto Carlos Casado Vara, profesor del departamento de Matemáticas y Computación, área de Matemática Aplicada.

Expone:

Que el alumno D. Diego García Muñoz, con DNI 71296810F, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado Modelo de propagación de malware basado en aprendizaje por refuerzo.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 22 de septiembre de 2022

Vº. Bº. del Tutor:	Vº. Bº. del co-tutor:
D. nombre tutor	D. nombre co-tutor

Resumen

En este trabajo se propone utilizar métodos de aprendizaje por refuerzo aplicados a la búsqueda de rutas de navegación dentro de una red de ordenadores. El uso de estos métodos ofrece una mayor flexibilidad frente a cambios en el entorno, además de la posibilidad de adaptación a diferentes objetivos. Gracias a estas características, se han considerado métodos apropiados para utilizar en un agente malware que pretenda propagarse por una red de ordenadores.

Se ha desarrollado un algoritmo de aprendizaje que simula la situación propuesta y realiza dicha búsqueda, además de un sitio web que implementa dicho algoritmo y ofrece una forma fácil y rápida de utilizarlo para usuarios con pocos conocimientos de programación.

Descriptores

Aprendizaje automático, Aprendizaje por refuerzo, redes de ordenadores, malware, búsqueda de ruta.

Abstract

In this work, we have proposed the use of reinforcement learning methods in the context of pathfinding within a computer network. The applicacion of these methods in contrast with other alternatives results in a higher flexibility to changes in the environment, in addition to being easily adaptable to various objectives. Thanks to these traits, we have considered these learning methods to be useful within the context of a malware agent attempting to propagate within a computer network.

We have developed a learning algorithm capable of simulating the aforementioned situation and carrying out said pathfinding. In addition, we have developed a website which implements said algorithm and offers a fast and simple way of utilizing the algorithm, for users with limited knowledge of programming.

Keywords

Machine learning, reinforcement learning, computer networks, malware, pathfinding.

Índice general

Índice de figuras

Índice de tablas

Introducción

En la actualidad, cada vez existen más amenazas a los equipos informáticos, y las organizaciones tienen que invertir más recursos en proteger sus redes privadas y todos los dispositivos que se encuentran en ellas.

Para poder asegurarse de que su nivel de defensa es apropiado, la perspectiva del desarrollador que ha creado o configurado los servicios no siempre es suficiente, por lo que es común que las organizaciones contraten la ayuda de los llamados hackers éticos. Estos hackers éticos son individuos – o equipos de individuos – que se encargan de infiltrarse en una red, dispositivo o servicio, sin causar daño real, pero encontrando sus vulnerabilidades y puntos débiles. De este modo, al tener la perspectiva de alguien externo a la organización, y utilizar las técnicas y herramientas que un atacante real utilizaría, pueden encontrar áreas con seguridad insuficiente que podrían haber sido pasadas por alto.

Una situación que podría suceder en una organización es que alguna persona introdujese algún tipo de malware en alguno de los dispositivos de la red, y éste comenzase a propagarse por el resto de los equipos, posiblemente teniendo algún objetivo, como acceder a archivos confidenciales y enviarlos a alguna dirección externa. Ni la organización ni los hackers éticos podrán predecir qué objetivos tendrán los atacantes, ni qué métodos utilizarán, así que tendrán que anticiparse a todos los posibles, para así poder reducir el riesgo a valores aceptables.

Lo que se propone en este proyecto es el uso de aprendizaje automático para simular esta situación, la introducción de un malware a una red de ordenadores, y observar qué rutas tomaría para conseguir un objetivo con el menor riesgo. Sabiendo eso, se podrían introducir medidas de defensa para evitar que el malware tomase esas rutas, haciendo así más difícil el trabajo

de los hackers maliciosos. La principal ventaja que ofrece este método es que puede encontrar rutas y puntos débiles que a un ser humano no se le habría ocurrido proteger, especialmente en redes grandes con gran cantidad de dispositivos.

Se ha optado por utilizar aprendizaje por refuerzo, pues al ser un modelo semi-supervisado no será necesario definir qué rutas se consideran exitosas para el malware y cuales no, sino que será suficiente con asignar valores de recompensa a las diferentes acciones que se pueden tomar. Además, es más robusto frente a variaciones en el entorno que otros modelos de aprendizaje, dando lugar a que, aunque cambie la situación de la red, el algoritmo pueda encontrar la ruta óptima igualmente.

Para facilitar su uso, este algoritmo de aprendizaje por refuerzo que se ha desarrollado también ha sido implementado dentro de una página web.

1.1. Estructura de la memoria

La estructura de esta memoria es la siguiente:

- **Introducción:** contiene una descripción del proyecto, además de una explicación de la estructura de la memoria y de los materiales complementarios.
- **Objetivos del proyecto:** listado de los objetivos, tanto generales como específicos.
- **Conceptos teóricos:** explicación de los principales conceptos teóricos necesarios para la comprensión del proyecto
- **Técnicas y herramientas:** descripción de las técnicas, metodologías y herramientas utilizadas, indicando por qué han sido elegidas frente a sus alternativas.
- **Aspectos relevantes del desarrollo del proyecto:** explicación de las diferentes decisiones tomadas durante el transcurso del proyecto, además de otras cuestiones que se consideren importantes.
- **Trabajos relacionados:** visión general de otros trabajos en el campo de la propagación de malware y aprendizaje automático en redes, y su relación al proyecto actual.

- **Conclusiones y líneas de trabajo futuras:** observaciones obtenidas después de haber completado el trabajo, y áreas en las que se puede profundizar o mejorar en caso de continuar el trabajo en el proyecto.

Además de la memoria, se cuenta con los siguientes apéndices:

Apéndice A - Plan de proyecto software: contiene tanto la planificación temporal del proyecto como los estudios de viabilidad realizados.

Apéndice B - Especificación de requisitos: lista los objetivos, requisitos y casos de uso del trabajo.

Apéndice C - Especificación de diseño: cubre las decisiones tomadas a la hora de diseñar los datos y procedimientos del sistema, además de una explicación detallada de su situación final.

Apéndice D - Documentación técnica de programación: incluye todos los aspectos que se consideren relevantes para los programadores, desde la estructura de directorios o las instalaciones necesarias hasta las características especiales de los ficheros fuente.

Apéndice E - Documentación de usuario: contiene un conjunto de explicaciones orientadas a los usuarios finales para que sean capaces de utilizar la aplicación correctamente y sin problemas.

1.2. Materiales adicionales

Junto con la memoria se proporciona un repositorio con el código fuente del algoritmo de aprendizaje por refuerzo, así como la página web lista para ser desplegada en cualquier ordenador para su uso.

Dicho repositorio también se encuentra en el siguiente enlace: https://github.com/dgm1003/TFG-RL-malware

Se incluye también un archivo con la imagen Docker de la página web con el nombre `tfg-website-image.tar`

Objetivos del proyecto

Este proyecto cuenta con varios objetivos:

2.1. Objetivos Generales:

- Diseñar y desarrollar un algoritmo de aprendizaje por refuerzo que simule la propagación de un malware por una red de ordenadores, buscando la ruta óptima para llegar a un equipo objetivo.
- Crear una interfaz gráfica que permita a un usuario con conocimientos mínimos de programación configurar el algoritmo y utilizarlo.

2.2. Objetivos Específicos:

1. Obtener conocimientos sobre aprendizaje por refuerzo, a través de documentación técnica y de investigación, y aplicar dichos conocimientos en el trabajo, centrándose especialmente en los Procesos de Decisión de Markov (MDP) y sus elementos: estados, acciones y recompensas.
2. Facilitar un entorno que permita representar mediante un grafo los datos de una red de ordenadores, que sea adaptable a diferentes topologías, sea capaz de representar los posibles estados de la red, y tenga definidas las diferentes acciones que puede tomar el agente malware introducido en la red, con sus recompensas asociadas.
3. Ofrecer una forma de visualizar el estado del entorno, representando tanto la topología de la red como el recorrido tomado por el malware y los dispositivos que han sido infectados.

4. Incluir también un agente de aprendizaje por refuerzo que trabaje sobre dicho entorno para entrenar la propagación del malware en la red, centrándose en la infección de un nodo objetivo.

5. Proporcionar una interfaz, como puede ser una página web, que permita a un usuario trabajar con el entorno y agente existentes, personalizándolos a sus necesidades, y obteniendo los resultados del entrenamiento y búsqueda de ruta consiguientes.

6. Realizar diferentes pruebas y estudios sobre el algoritmo creado para demostrar su eficiencia y adaptabilidad.

Conceptos teóricos

Este proyecto utiliza un modelo de aprendizaje automático llamado aprendizaje por refuerzo. Se explican los conceptos más importantes para comprender el proyecto a continuación.

3.1. Aprendizaje por refuerzo

Conceptos básicos

A diferencia de otros modelos de aprendizaje automático, el aprendizaje por refuerzo no es supervisado, pues no cuenta con conjuntos de entrenamiento en los cuales se indican las respuestas correctas a diferentes situaciones, ni tampoco es no supervisado, ya que sí cuenta con recompensas que le informan sobre como de aceptable es una solución concreta.

Se encuentra en un punto intermedio entre esas dos ramas: Se cuenta con un agente, un programa que decide qué acciones tomar y cuyo propósito es aprender. A este agente, al realizar acciones, se le ofrecen recompensas a corto y largo plazo, de modo que intentará maximizar el valor obtenido con las recompensas, descubriendo qué acciones son óptimas en cada momento.

Componentes del Aprendizaje por Refuerzo

Para poder resolver un problema de Aprendizaje por Refuerzo necesitaremos varios elementos esenciales. El primero de ellos será un **agente**, que como ha sido explicado anteriormente, es quien realiza el aprendizaje,

Por otro lado, también será necesario un **entorno** en el cual se realice el propio aprendizaje. Este entorno se representará mediante un conjunto de

estados en los que puede estar el agente, y un conjunto de **acciones** que podrá tomar en cada uno de los diferentes estados.

El agente además contará con una **política**. Esta política representará el comportamiento del agente, indicando de algún modo qué acciones tomará dependiendo de en qué estado se encuentre en cada momento, con el objetivo de maximizar una recompensa.

Esta señal de **recompensa** también es un elemento indispensable del problema, siendo uno de los componentes que más influyan sobre los resultados finales. Cada vez que el agente cambie de estado, o cuando pase una unidad de tiempo, se le enviará un valor numérico de recompensa, que indicará como de óptima es la acción tomada si se pretende llegar al objetivo. Por ello, si se diseñan dos algoritmos de Aprendizaje por Refuerzo que tengan todos sus componentes idénticos excepto sus funciones de recompensa, se verá como los agentes pueden llegar a cumplir objetivos completamente diferentes.

Otra parte que está muy fuertemente relacionada con la señal de recompensa será la señal de **valor** (o calidad). Al igual que las señales de recompensa indican como de buena es la acción o el estado actual, una señal de valor indicará la posible recompensa futura a partir de dicho estado. Esta señal es importante, pues permitirá comportamientos en los que el agente ignora recompensas altas inmediatas, para obtener recompensas mayores en un futuro. Sin embargo, el valor de un estado no es algo que se pueda saber al principio, por lo que es algo que tendrá que ir calculando y aprendiendo el agente a medida que progrese.

Por ejemplo, en el ejemplo siguiente (Figuras X y X), tenemos una red simple, siendo el inicio el nodo 0 y el objetivo el nodo 2. Después de realizar un entrenamiento hemos obtenido una tabla con valores de calidad para cada pareja estado-acción (siendo el estado el nodo en el que se encuentre el agente, y la acción moverse al nodo indicado o infectar el nodo actual).Se puede ver como, empezando en el estado 0, y seleccionando la acción con un valor más alto de calidad, la ruta óptima será: `0 -> 1 -> 2 -> Infectar` en vez de otras rutas más largas.

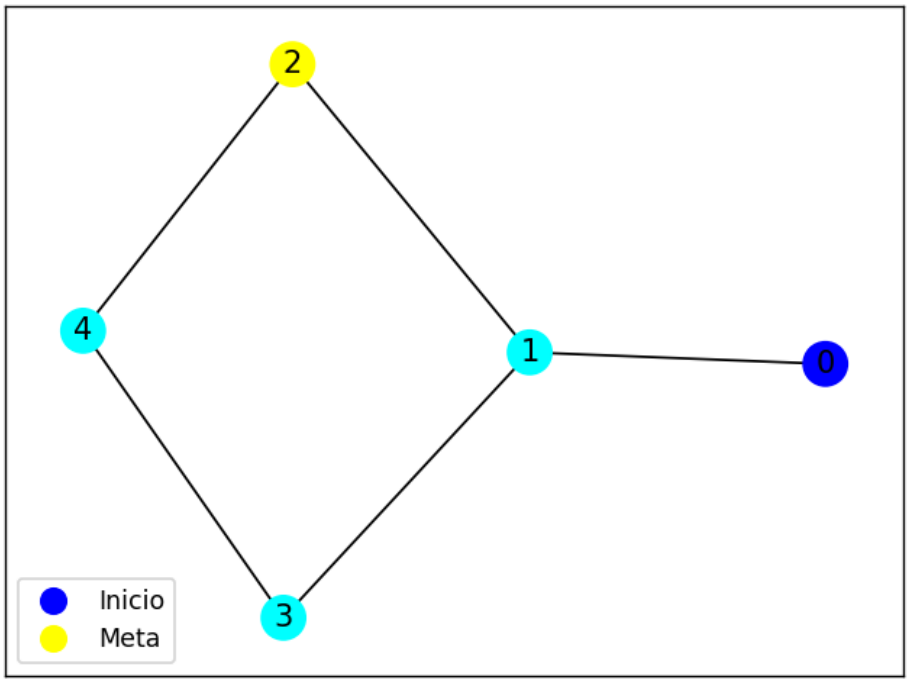

Figura 3.1: Ejemplo de red simple para demostración de valor de calidad

Tabla de calidades (normalizada)

Estado/Accion	0	1	2	3	4	Infectar
0	[0.34233919	0.4894891	0.	0.	0.	0.34113799]
1	[0.34234207	0.48948499	0.69969914	0.34233193	0.	0.4882879]
2	[0.	0.48947462	0.69969327	0.	0.48947462	1.]
3	[0.	0.48948499	0.	0.34233193	0.48947462	0.34113073]
4	[0.	0.	0.6996997	0.34233919	0.48948499	-0.0015015]

Figura 3.2: Ejemplo de tabla de calidades para la red anterior

Por último, en algunas situaciones, se puede contar con un **modelo**. Este modelo consistirá en conocimiento, total o parcial, del entorno de aprendizaje. Esto ayudará al agente a predecir situaciones futuras, y poder tomar decisiones más óptimas con más frecuencia. Sin embargo, no siempre es posible contar con este modelo, por lo que es un elemento opcional.

Exploración y explotación

Un tema a tener en cuenta al realizar algoritmos de Aprendizaje por Refuerzo es la necesidad de balancear la exploración y la explotación. Al iniciar el aprendizaje, el agente no tendrá información sobre los valores y recompensas de los diferentes estados, por lo que tendrá que probar

acciones e ir recogiendo los valores que se devuelven para ir descubriendo esas recompensas y valores. A este comportamiento se le llama exploración. Una vez explorado el entorno, el agente podrá seleccionar las acciones que le devuelvan mayor recompensa o valor, pues son las que más aumentarán la recompensa total. Este otro comportamiento se denomina explotación.

Sin embargo, aunque parezca que, una vez explorado inicialmente el entorno ya no sea necesario volver a hacerlo, y sea más óptimo realizar explotación durante el resto del tiempo para maximizar la recompensa, no siempre será verdad este caso. Es muy probable que no se hayan explorado todos los posibles comportamientos, o que la calidad de algún estado no sea cercana a la realidad al haber sido explorado pocas veces, lo cual resultaría en situaciones en las que alguno de los caminos sin recorrer acabe otorgando mayor recompensa. También existen problemas de aprendizaje por refuerzo en los que el entorno va cambiando a lo largo del tiempo, en los cuales sería esencial realizar exploración a lo largo de todo el proceso de aprendizaje. Por lo tanto, será esencial mantener un equilibrio entre ambos tipos de comportamiento.

Procesos de decisión de Markov

Existen varios tipos de problemas que se pueden resolver mediante aprendizaje por refuerzo, pero el que se va a tratar en este trabajo es un Proceso de Decisión de Markov (MDP).

En un MDP, el agente y el entorno se comunican de forma cíclica. Al empezar, el entorno le proporcionará al agente una representación de su situación actual, la cual llamaremos estado. Dicho estado se puede entender como el conjunto de características observables por el agente. Con esta información, el agente realizará una acción, notificando al entorno de ello. Al ocurrir dicha acción, el estado del entorno cambiará, siendo enviado al agente junto con una recompensa numérica.

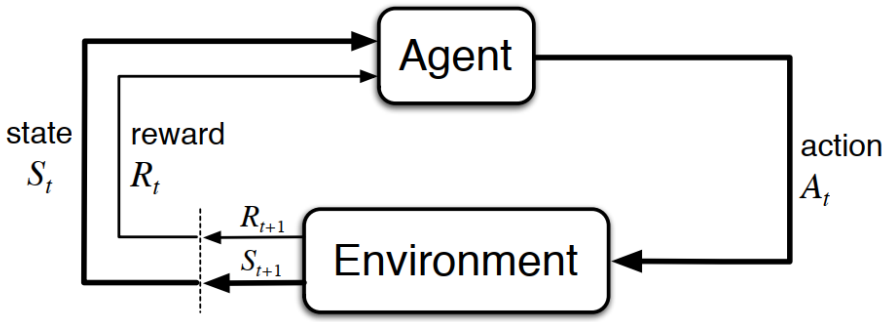

Figura 3.3: Relación entre el entorno y el agente en un proceso de decisión de Markov [25]

La característica principal de los Procesos de Decisión de Markov es que sus estados cumplen la propiedad de Markov. Dicha propiedad es[23]:

$$P[S_t + 1|S_t] = P[S_t + 1|S_1, S_2, ..., S_t]$$

Lo cual representa que la probabilidad de llegar a un estado desde un estado anterior es igual a la probabilidad de llegar a dicho estado teniendo en cuenta el camino completo recorrido hasta su estado anterior. Es decir, que cada estado tiene toda la información relevante para tomar decisiones, y no es necesario mirar a los estados anteriores para seleccionar una acción.

Los MDP pueden ser tanto estocásticos como deterministas. En los procesos deterministas, realizar una acción a partir de un estado llevará siempre al mismo estado siguiente, con la misma recompensa asociada. Mientras tanto, en un modelo estocástico, realizar una acción a partir de un estado nos podrá llevar a varios estados destino, cada uno con una recompensa y probabilidad asociada. La suma de las probabilidades de los diferentes estados destino para una pareja estado-acción siempre será 1. En nuestro caso, el problema se representará mediante un Proceso de Decisión de Markov determinista

Aprendizaje de diferencia temporal

En el método de aprendizaje por refuerzo de diferencia temporal, el objetivo es obtener una estimación acertada del valor de cada estado, es decir, la posible recompensa futura que se podrá obtener si se visita dicho estado.

Un ejemplo del cálculo de la estimación del valor de un estado (llamado $V(S_t)$) será el siguiente[25]:

$$V(S_t) = V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

Lo cual indica que, dado un estado, y una estimación anterior de dicho estado, el incremento de valor entre la nueva estimación y la antigua consistirá en la diferencia entre la suma de la recompensa del estado inmediatamente posterior R_{t+1} y su valor $V(S_{t+1})$ (multiplicado por una tasa de descuento γ) y la estimación del valor del estado actual $V(S_t)$, multiplicado por la tasa de aprendizaje α.

Interpretando esta ecuación paso a paso, podemos ver como $R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ será la diferencia entre dós términos:

- La estimación del valor actual $V(S_t)$
- Una nueva estimación $R_{t+1} + \gamma V(S_{t+1})$, que se considera más exacta.

Podemos afirmar que la segunda será más exacta porque, aunque no podamos saber con certeza el valor de un estado, sí podemos descubrir con exactitud la recompensa después de realizar una acción, y si tenemos en cuenta que el valor de un estado es una estimación de todas las recompensas futuras, al saber una de ellas, estaremos más cerca del valor real. Por lo tanto, podemos considerar $R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ como una aproximación al error entre el valor real y el valor estimado, el cual toma el nombre de *error TD*.

Conociendo este error TD, podremos acercar nuestra estimación al valor real, y eso es lo que hace la operación mostrada anteriormente: suma al valor estimado el error TD, después de haberlo multiplicado por una tasa de aprendizaje α.

Este método de aprendizaje proporciona ventajas en comparación a otros, siendo la ventaja principal el hecho de que calcula las estimaciones de los valores basándose en otras estimaciones, y por lo tanto, tiene que esperar menos tiempo para actualizar sus valores, y estos cambiarán con más frecuencia, lo que hará que converjan a los valores reales más rápidamente. En comparación, otros métodos como el aprendizaje Monte Carlo tienen que esperar a que, dentro del entrenamiento, se alcance el objetivo para poder actualizar sus valores.

Otra ventaja es el uso del factor de descuento γ, que permite definir la importancia que tienen los estados y situaciones futuras en la toma de decisiones. A valores de γ muy bajos, se tomará como valor estimado principalmente la recompensa inmediata, haciendo que el valor futuro influya muy poco en el resultado.

Q learning

Sin embargo, en la mayoría de sistemas, un estado tendrá varias acciones disponibles, que a su vez llevarán a diferentes estados. Por lo tanto, obtener el valor de un estado concreto no será muy útil, pues nos indicará por qué estados hay que pasar, pero no como se puede llegar a ellos. Considerando esto, se puede ver como es mejor calcular el valor para cada pareja estado-acción, es decir, $V(S_t, A_t)$. De este modo, habrá que modificar el cálculo del valor, y obtendremos la siguiente operación[25]:

$$V(S_t, A_t) = V(S_t, A_t) + \alpha[R_{t+1} + \gamma * max_a V(S_{t+1}, a) - V(S_t, A_t)]$$

Como se puede ver, en vez de utilizar el valor del estado siguiente, ya que se necesita una pareja estado-acción, se seleccionará la acción que nos devuelva el mayor valor estimado.

Esta variante del aprendizaje por diferencia temporal es llamada **Q-Learning**, y es el método utilizado en este proyecto.

3.2. Redes de ordenadores

Este algoritmo de aprendizaje por refuerzo que se ha desarrollado está diseñado para navegar por una red de ordenadores. Por **red de ordenadores** se entiende a un conjunto de dispositivos informáticos, o computadoras, que son autónomas (es decir, no necesitan estar conectadas a otras computadoras para funcionar) y que están interconectadas, independientemente de la forma que tome esta conexión, ya sea mediante bluetooth, wifi, o directamente a través de cableado.[26]

Los dispositivos conectados a las redes de ordenadores pueden ser muy variados, pudiendo convivir ordenadores, servidores, switches, impresoras, firewalls, escáneres, y otra multitud de dispositivos en la misma red. No se va a explicar las diferencias entre los comportamientos de estos dispositivos, ni las características de las diversas conexiones posibles, pues para la realización de este trabajo se han considerado todos los dispositivos como iguales, y todas las conexiones como idénticas. Se deja la ampliación de complejidad del entorno como una posible línea de trabajo futura.

Para la organización de todos estos dispositivos, la práctica más común consiste en distribuirlos en una topología de red. Algunos ejemplos de topologías son la **topología en anillo**, en la cual cada equipo se conecta

únicamente con otros dos dispositivos, formando un bucle; la **topología en estrella**, que cuenta con un dispositivo central al que se conectan todos los demás; o la **topología en árbol**, que sigue una distribución jerárquica.

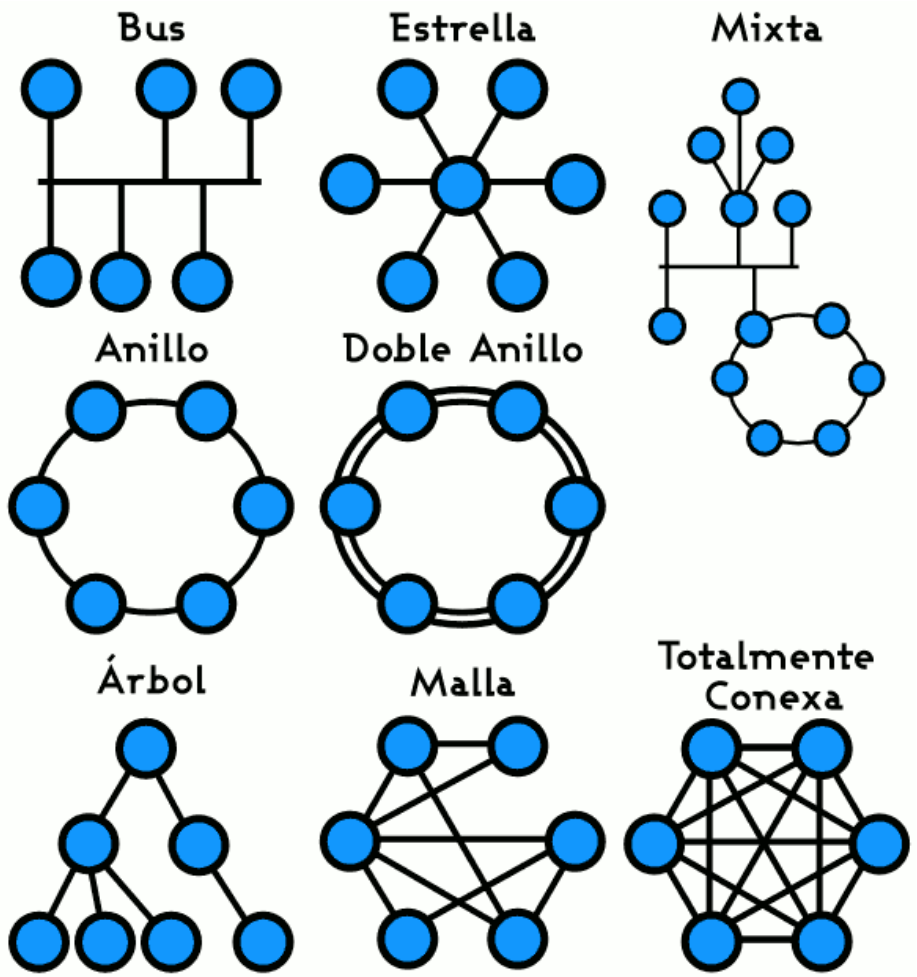

Figura 3.4: Principales topologías de redes informáticas, obtenido de [3]

En la página web desarrollada se hace uso de una topología mixta, contando con varias redes en forma de árbol, cuyas raíces forman parte de una malla. Esto se decidió así pues, en la mayor parte de situaciones reales en organizaciones o empresas, se cuenta con dispositivos conectados en estructuras jerárquicas, teniendo varios dispositivos conectados a un mismo switch o router, o varios switches conectados a un switch central, por ejemplo. Se incluyó la topología en malla para ofrecer más rutas al agente que una topología en árbol pura, y a la vez simular la redundancia

de dispositivos y conexiones que es común en redes grandes. Sin embargo, el agente funciona igualmente aplicado a redes de otras topologías.

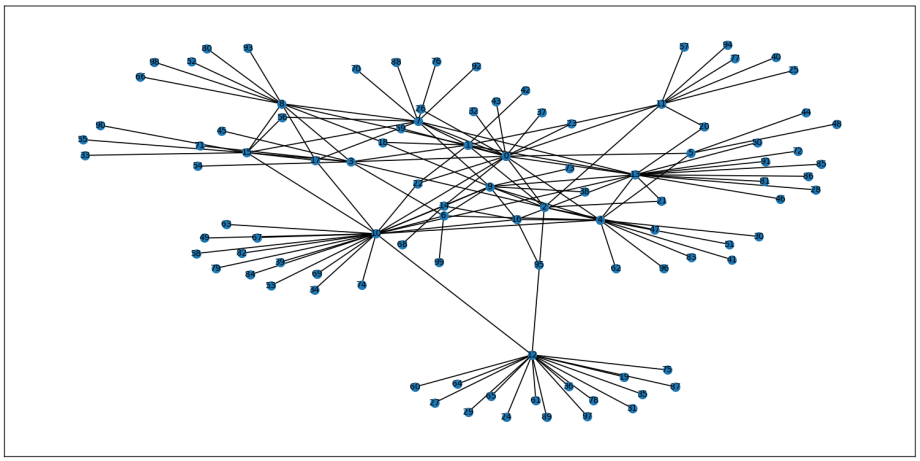

Figura 3.5: Ejemplo de red utilizada en el proyecto

Técnicas y herramientas

En el siguiente capítulo se presentan las diferentes metodologías empleadas en la realización del proyecto, además de todas las herramientas de las que se ha hecho uso. Se habla también de las posibles alternativas que fueron consideradas, y las razones por las cuales se optó por una alternativa en vez de las demás.

4.1. Metodologías de trabajo

Metodología ágil: Scrum

Para la realización del Trabajo Fin de Grado se ha optado por tomar como base las metodologías ágiles, principalmente Scrum[22], en vez de utilizar otras más tradicionales como la metodología Waterfall.

La principal característica de la metodología Scrum es que el trabajo se realiza de manera iterativa e incremental, de modo que al final de cada iteración (también llamadas Sprints) se obtenga una versión del producto funcional, con un mayor número de características que en la iteración anterior. Para ello, los requisitos se dividen en tareas, se priorizan, y se van ordenando en el denominado Product Backlog. Al inicio de cada sprint se seleccionarán los requisitos que se ha decidido trabajar en dicho sprint, y se añadirán al Sprint Backlog. En Scrum se definen varios roles, siendo los más importantes el Product Owner, que se comunica con el cliente o usuario final para definir los requisitos y el backlog, y el Scrum Master, que se encargará del seguimiento y coordinación del modelo Scrum, además de solucionar aquellos problemas que tenga el equipo de desarrollo. Por último, para asegurar el avance fluido de los proyectos, se realizan varios tipos de reuniones: Una al inicio de un sprint, definiendo sus objetivos, reuniones

diarias de seguimiento, y una al finalizar cada sprint, para evaluar el trabajo realizado en la iteración.

Para este trabajo se ha seguido una idea similar, realizando Sprints de 1-2 semanas, pero con varias modificaciones:

- Los primeros sprints fueron destinados a adquirir conocimientos necesarios para la realización del trabajo, y por lo tanto, no resultaron en un producto funcional al finalizar dichos sprints.
- Debido al reducido número de personas que participan en el trabajo, no se ha realizado una división en roles, sino que es el alumno quien se encarga de todas las responsabilidades de cada rol, con asistencia de los tutores, en ocasiones actuando de Product Owner.
- Se han descartado las reuniones diarias, debido a que no sería posible coordinar el tiempo necesario entre el alumno y los tutores cada día. En vez de eso, el alumno trabajó por su cuenta, contactando con los tutores mediante correo electrónico en caso de que surgiese algún problema que bloquease el trabajo, o, en casos excepcionales, realizando reuniones extraordinarias.
- Las reuniones inicial y final de cada sprint se fusionaron, por la misma razón que el punto anterior, de modo que solamente se realizaba una reunión conjunta en la que se finalizaba un sprint y se planificaba el siguiente.

Se decidió abarcar el proyecto con esta metodología frente a otras tradicionales como la metodología en cascada porque ofrecía una mayor flexibilidad, priorizando la obtención de un programa funcional de manera rápida sin sacrificar calidad, y permitiendo su ampliación y modificación de forma fluida a lo largo del tiempo, dos puntos que se consideraron importantes para la realización de un Trabajo Fin de Grado

Gestión del repositorio: GitHub con ZenHub

Por otro lado, para realizar un seguimiento del proyecto por parte del alumno y de los profesores a la vez, y para tener los archivos y sus versiones accesibles fácilmente en internet, se decidió utilizar una página de control de versiones y repositorios. Se consideraron diferentes alternativas, como GitLab, CodeCommit, SourceForge o BitBucket; pero finalmente se decidió optar por **GitHub**[10].

Como el propósito del repositorio era ser público, se descartaron opciones como GitLab, CodeCommit o BitBucket, que, aunque ofrezcan un mayor número de características, son más indicadas para repositorios privados.

GitHub, sin embargo, está centrada en repositorios públicos, y al ser una página muy popular cuenta con una gran cantidad de extensiones y aplicaciones que aportan muchas herramientas y opciones adicionales. Además, el alumno ya se ha familiarizado con el uso de dicha página durante el curso académico,

Por otro lado, para poder implementar la metodología Scrum, los issues y milestones de GitHub eran un buen primer paso, pero no ofrecían todas las opciones necesarias para ello. Por eso, como adición a GitHub, se decidió utilizar la extensión **ZenHub**[28], que permite un mejor control del proceso de integración contínua y gestión de proyecto Scrum.

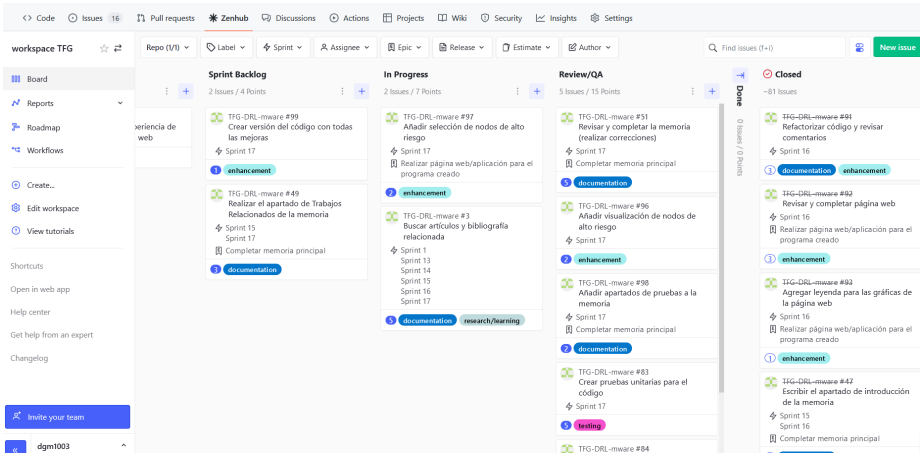

Figura 4.1: Ejemplo de control de issues y seguimiento Scrum con ZenHub

Control de calidad: Codacy, CodeClimate

Para poder controlar la calidad del código que se va creando, existe una gran cantidad de aplicaciones compatibles con GitHub que analizan los archivos de los repositorios indicados, buscando diferentes tipos de problemas o malas prácticas. A menor número de estos problemas, el código será más fácil de comprender, mantener y utilizar, por lo que su calidad será mayor.

Se ha optado por utilizar dos de estas aplicaciones: **Codacy**[2] y **CodeClimate Quality**.[1] Codacy detecta un mayor número de problemas que CodeClimate, y es capaz de analizar una mayor variedad de tipos de

archivo, como CSS, pero existen problemas como la complejidad cognitiva (dificultad de comprensión del código) que solo se muestran en CodeClimate, por lo que se consideraron relevantes los análisis de ambas aplicaciones.

4.2. Herramientas

A continuación se muestran las herramientas utilizadas durante el desarrollo.

Lenguaje de programación: Python

El proyecto se propuso inicialmente con la intención de desarrollarse en **Python**[21], debido a que existían librerías muy apropiadas para los objetivos definidos, e incluso para líneas de trabajo futuras como la inclusión de aprendizaje profundo por refuerzo. Las librerías utilizadas fueron las siguientes:

- **Random:**[19] librería simple para la generación de secuencias de números aleatorios
- **Numpy:**[16] librería enfocada a trabajos matemáticos y con matrices. En nuestro caso, para que el algoritmo seleccionado resultase efectivo era necesario el uso de tablas, y esta librería facilitaba mucho la creación de dichas tablas y la búsqueda dentro de ellas.
- **Matplotlib:**[12] librería para la generación de elementos gráficos. Se utilizó para crear imágenes representativas de la red de ordenadores, facilitando la comprensión de dichas redes y su situación en cada momento.
- **NetworkX:**[15] librería para la creación y el manejo de grafos. Las redes de ordenadores se tenían que estructurar y guardar de algún modo, y esta librería era ideal para ello. Además, ofrecía una gran cantidad de opciones para guardar diferentes atributos en los nodos y conexiones, agilizando la ejecución del algoritmo y simplificando la generación de imágenes con Matplotlib. También incluía herramientas para la generación de redes de forma aleatoria, reduciendo mucho el tiempo de desarrollo.
- **Flask (incluye Jinja2):**[6] framework para la creación de aplicaciones web. Fue indispensable para el desarrollo del sitio web.

- **Unittest:**[20] framework de automatización de pruebas, utilizado en la creación de pruebas unitarias.

Además de Python, también se tuvieron que usar otros lenguajes de programación. HTML y CSS fueron necesarios para el sitio web, y como también se utilizó **Docker**[9] para empaquetar el programa en una imagen fácil de ejecutar, se tuvo que utilizar su lenguaje propio para los archivos de configuración de la imagen.

IDE: Visual Studio Code

Teniendo en cuenta que el proyecto se va a realizar con el lenguaje de programación Python, las IDEs consideradas fueron:

- Spyder
- Visual Studio Code
- Jupyter Notebook
- Eclipse + PyDev

Se optó por **Visual Studio Code**[13] por varias razones:

1. No requiere de la instalación ni manejo de extensiones adicionales para el trabajo básico con Python, al contrario que Eclipse, aunque cuenta con una gran variedad de extensiones con funciones adicionales.
2. Se trabaja con ficheros .py, en vez de .ipynb como en Jupyter, lo cual facilitará su uso en una página web en caso de que se decida desarrollar una, al no tener que exportar el código con cada modificación. Aún así, es compatible con ficheros .ipynb en caso de ser necesario trabajar con ellos.
3. Cuenta con integración con Git, lo cual permite realizar commits de forma rápida y sencilla. Aunque Spyder también cuenta con funcionalidades similares, están menos desarrolladas y son más difíciles de usar, lo cual conllevaría un aumento del tiempo necesario para hacer commits u operaciones de push/pull.

Sin embargo, en ocasiones se utilizó **Jupyter Notebook**[11] y **Google Colab**[7] para la realización de algunos tutoriales, además de **Spyder**[24] durante el estudio de valores óptimos, pues permitía un manejo más fácil de las gráficas de Matplotlib.

Documentación: Overleaf

Los programas que se consideraron fueron:

- Microsoft Word
- OpenOffice Writer
- Overleaf
- Texmaker

Por un lado, se decidió utilizar el lenguaje **LaTeX**[18], pues es un lenguaje ofrece una gran cantidad de opciones para textos académicos, permitiendo el manejo fácil de aspectos como referencias, fórmulas matemáticas, o índices, aligerando la carga de trabajo en esos aspectos, y permitiendo al alumno centrarse en escribir la documentación en sí.

Dentro de los programas de edición de documentos LaTeX, el alumno se decantó por **Overleaf**[17], pues aunque es posible que ofrezca menos opciones que otros editores que se instalen en un ordenador, permite la edición colaborativa de documentos, lo cual es útil a la hora de mostrar el progreso realizado a los profesores y facilitarles el proceso de corrección.

Aspectos relevantes del desarrollo del proyecto

En este capítulo se recogen los aspectos más importantes del proceso de desarrollo de este proyecto, cubriendo desde los procesos de selección de tema y planificación hasta los problemas que surgieron durante el desarrollo, y explicando por qué se tomaron las decisiones que se tomaron.

5.1. Inicio del proyecto

La idea del proyecto surgió del trabajo de uno de los tutores en el área del aprendizaje por refuerzo, y otros trabajos fin de grado en los que había participado anteriormente, y el alumno se interesó en dicha idea, pues tanto el aprendizaje por refuerzo como la propagación de malware y el hacking ético son áreas que no se suelen tratar en la carrera pero están ganando cada vez más importancia en el ámbito profesional.

Originalmente, el trabajo se propuso con la idea de utilizar aprendizaje profundo por refuerzo, pero rápidamente se llegó a la conclusión de que sería mejor dejar el aprendizaje profundo a un lado y centrarse en el aprendizaje por refuerzo – para empezar desde cero y comprender mejor los elementos involucrados y cómo afectan al contexto seleccionado – dejando así el uso de aprendizaje profundo para una posible ampliación, como un trabajo fin de máster o una continuación del proyecto fuera de la universidad.

5.2. Primeros pasos y obtención de conocimientos

Después de haber decidido el tipo de aprendizaje que se utilizaría en el proyecto, y después de unas reuniones iniciales, se definieron los objetivos que se quería alcanzar más allá de la idea inicial, como el diseño de una visualización de los resultados y una interfaz en modo de página web, se seleccionaron los programas y metodologías a utilizar a lo largo del proyecto, y se pasó a la fase de formación y obtención de conocimientos. Estos dos primeros apartados se muestran en más profundidad en las secciones de "Objetivos del proyecto" y "Técnicas y herramientas" respectivamente.

En cuanto a la formación, para la realización del proyecto se necesitaban conocimientos no solamente sobre el aprendizaje por refuerzo y las diferentes variedades que se podrían implementar, sino también sobre qué librerías de python servirían para programar dicho aprendizaje y cómo funcionan, además de los elementos y topologías de redes de ordenadores.

Los recursos utilizados a la hora de ampliar mis conocimientos sobre el aprendizaje por refuerzo y las redes de ordenadores fueron:

- Libro *Reinforcement Learning: An Introduction* (Richard S.Sutton y Andrew G. Barto)[25]
- Presentación *Descubriendo información en una red compleja usando aprendizaje por refuerzo*
  https://youtu.be/vu3rruSrtTo
- Curso de DeepMind y UCL *Introduction to Reinforcement Learning with David Silver*[23]
- Artículo *Playing Atari with Deep Reinforcement Learning* (Volodymyr Mnih, Koray Kavukcuoglu et al)[14]
- Seminarios ofrecidos por los tutores.

Los tutoriales que se siguieron a la hora de aprender las diferentes librerías de python que se pretendía utilizar fueron los enumerados a continuación. No todos se llegaron a realizar al completo, pero al menos fueron leídos y sirvieron para entender algunos de los conceptos:

- Numpy:

  - Primera toma de contacto, ejemplo simple: http://firsttimeprogrammer.blogspot.com/2016/09/getting-ai-smarter-with-q-learning.html
  - Situación más compleja, junto con varias explicaciones de conceptos: https://blog.floydhub.com/an-introduction-to-q-learning-reinforcement-learning/
  - Otra situación: https://www.analyticsvidhya.com/blog/2021/04/q-learning-algorithm-with-step-by-step-implementation-using-python/
  - Ejemplo de uso de grafos para aprendizaje por refuerzo: https://www.viralml.com/video-content.html?v=nSxaG_Kjw_w

- Gym, Keras:

  - Ejemplo simple de uso en Q-learning: https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/
  - Tutoriales de tanto aprendizaje por refuerzo como aprendizaje profundo por refuerzo: https://rubikscode.net/2021/07/13/deep-q-learning-with-python-and-tensorflow-2-0/
  - Tutorial sobre entornos de Gym: https://colab.research.google.com/github/araffin/rl-tutorial-jnrr19/blob/master/5_custom_gym_env.ipynb
  - Vídeo de Nicholas Renotte sobre problema de CartPole de Gym, versión resumida de 20 minutos: https://www.youtube.com/watch?v=cO5g5qLrLSo
  - Curso completo de Nicholas Renotte de aprendizaje por refuerzo en 3 horas: https://www.youtube.com/watch?v=Mut_u40Sqz4
  - Vídeo de Nicholas Renotte sobre entornos de gym: https://www.youtube.com/watch?v=bD6V3rcr_54
  - Otro tutorial del entorno de CartPole de Gym: https://keon.github.io/deep-q-learning/

- NetworkX:

  - Tutorial de la documentación: https://networkx.org/documentation/stable/tutorial.html

5.3. Diseño del problema

Una vez se tenía los suficientes conocimientos como para empezar a diseñar el problema, se empezó a realizar tanto el aprendizaje como el desarrollo en tándem. También se fueron proponiendo puntos en los que sería posible la ampliación del alcance del problema, aunque no se fuesen a desarrollar dentro del proyecto.

Problema

El problema se definió de la siguiente manera:

El usuario tendría conocimiento de una red de ordenadores, gracias a un proceso anterior a la introducción del malware a dicha red. De esta red, se conocerían las interconexiones entre los diferentes dispositivos, además de una estimación de como de seguro sería cada uno de los dispositivos (obtenida a partir de las características de cada dispositivo, como el tipo de dispositivo, modelo, o importancia dentro de la empresa). Esta estimación de la seguridad de un dispositivo, por lo tanto, sería proporcional a la probabilidad de que el malware fuera detectado si intentase desplazarse por ese dispositivo. Por lo tanto, se tendría un nivel de información parcial sobre el entorno en el que actuaría el agente. Gracias a esta información sobre la red, el informático que fuese a liberar al malware en el sistema podría definir un dispositivo como objetivo para la infección.

De este modo, se daría por completada la misión del malware si, una vez entrase en la red por algún método, consiguiese acceder al dispositivo objetivo e infectarlo sin ser detectado.

Para poder obtener resultados satisfactorios, se optó por realizar el enrutado mediante un algoritmo de aprendizaje por refuerzo, de modo que aprendiese las características de la red y encontrase la ruta más corta y segura hasta su destino. Para el entrenamiento del agente se decidió utilizar un método de tablas Q.

Para que este algoritmo obtuviese los resultados esperados, se definió un conjunto de estados, acciones y recompensas, que se explican en profundidad en el anexo C.

Red

La red se decidió representar como un grafo, siendo los nodos los diferentes dispositivos, permitiendo el entrenamiento del algoritmo con diferentes redes

y topologías, sin afectar a su rendimiento. Para este trabajo, se tomará una visión simplificada, definiendo ciertas restricciones. Se representarán en forma de grafos, siendo los nodos los diferentes dispositivos y las aristas las conexiones entre dichos dispositivos.

Por un lado asumiremos que, entre dos dispositivos concretos, solamente habrá una conexión directa única, y no se tendrá en cuenta el tipo de conexión (dará igual que sea por un cable USB entre un ordenador y un teléfono móvil, o mediante conexión inalámbrica a un router, o cable ethernet a un switch). Lo que le importará a nuestro algoritmo no será de qué modo moverse entre dos nodos conectados, sino decidir por qué nodos pasar en su recorrido hasta el objetivo.

Por otro lado, los nodos podrán ser una gran variedad de dispositivos: tanto routers con acceso a otras redes, como switches (con o sin firewall), además de ordenadores de sobremesa y portátiles, impresoras, escáneres, teléfonos móviles, u otros dispositivos conectados a la red. Sin embargo, el algoritmo no decidirá en base a qué tipo de dispositivo sea cada nodo, sino en base a dos criterios:

- **Si es un nodo hoja o no**: si solamente tiene una conexión (es una hoja), desplazarse a ese nodo no aportará nada al algoritmo, pues tendrá que retroceder para poder continuar su camino hacia el objetivo. Se preferirán nodos con más de una conexión, sin importar el número mientras sea mayor de uno.

- **Grado de seguridad del nodo**: dispositivos como ordenadores sobremesa de posiciones críticas o switches con firewalls tendrán un mayor número de medidas de seguridad, que puedan detectar al malware y realizar acciones para detenerlo. Por lo tanto, el algoritmo intentará evitar dichos dispositivos.

Estos dos criterios serán los atributos de cada nodo, representados como un valor numérico.

Las redes se generarán de forma relativamente aleatoria, una vez se definan varios parámetros como el número de nodos de la red, para probar el algoritmo en una gran cantidad de situaciones. Se podrá probar con diferentes topologías, como topologías de árbol o en anillo. Para la generación de redes se hará uso de librerías de Python, las cuales se pueden ver en el apartado de Técnicas y Herramientas.

5.4. Desarrollo del algoritmo

Siguiendo la metodología SCRUM, se enfocó el programa de modo que se empezase con una versión simple pero funcional del código, y en cada iteración se iría aumentando el número de características, o mejorando su funcionamiento. Se pasó por las siguientes iteraciones:

1. Versión inicial: Se comenzó desarrollando un producto mínimo viable, que encontrase una solución para una red relativamente simple, comprobando que se había entendido cómo programar una instancia del aprendizaje por refuerzo. Consistía en el código necesario para poder definir la red y entrenar un agente hasta que encontrase una ruta hasta el objetivo.

2. División en estructura de clases: A continuación, se aplicaron los conocimientos obtenidos en el grado respecto a la Ingeniería de Software y Programación Orientada a Objetos para dividir el código inicial en un conjunto de funciones, incluidas en una clase. De este modo, la utilización del programa se simplificó considerablemente, pues el usuario tendría que crear una nueva instancia de la clase y llamar a unas pocas funciones, en vez de buscar diferentes valores en el propio código y modificarlos. Además, la transformación del código en una clase permitía también que la implementación en otros contextos fuese más fácil.

3. Función de recompensa: Más adelante, se consideró que, en vez de obtener los valores de recompensa desde una tabla de $2n * 2n$ posiciones (siendo n el número de nodos de la red), sería más eficiente obtener dicha recompensa de una función que tomase como entrada el estado actual y la acción a realizar. De este modo, no solo se ahorraría espacio en memoria (el cual sería considerable con redes grandes), sino que también, aunque ejecutar la función tardase más tiempo que acceder a la correspondiente posición de la tabla, se ahorraría el tiempo necesario en construir la tabla completa. Además, resulta en un método más fácil de entender de cara a nuevos programadores que decidan trabajar en el proyecto en el futuro.

4. Implementación de NetworkX: El problema con las versiones anteriores surgía de que, para definir la red de ordenadores, se utilizaba un vector con tuplas de dos números, cada tupla representando una conexión entre dos nodos. Esto resultaba muy engorroso a la hora de definir redes con números elevados de nodos y conexiones, por lo que se pasó

a utilizar una estructura de grafos para representar dicha red. Se optó por la librería de NetworkX, pues era relativamente fácil de aprender, pero ofrecía una gran cantidad de opciones a la hora de generar y manipular grafos. Después de un breve periodo de aprendizaje, y de decidir en la forma de generación de dichos grafos, se implementó la librería en el código, sustituyendo a la antigua representación de las redes.

5. Separación de algoritmo y red: La situación que surgió al incluir NetworkX era que el código del algortimo estaba inevitablemente relacionado con la representación de la red, cosa que se hizo evidente al ver la cantidad de cambios que hubo que realizar para adaptar el uso de Networkx para nuestro algoritmo. Por ello, el siguiente paso consistió en separar la programación del algoritmo de la representación de la red, de modo que si se quisiera cambiar uno de los dos elementos en algún momento, sería mucho menos probable que hubiese que modificar el otro elemento. Esto aumentó la posible reutilización del código en otros contextos de manera considerable, además de simplificar su mantenimiento de cara a otros programadores que fueran a utilizarlo.

6. Separación entre agente y entorno: Expandiendo en la idea anterior, se decidió ir un paso más allá, separando el agente del entorno. De este modo, el agente no sería el responsable de modificar la situación de los componentes de la red, ni obtener la recompensa de una acción ni otras cuestiones similares, delegando todas estas tareas al entorno (además de la gestión de la red como se explica en el apartado anterior), y centrándose solamente en entrenar su modelo (o tabla Q), y en buscar las rutas óptimas.

7. Entrenamiento mediante episodios: Esta separación entre agente y entorno dio lugar a más oportunidades de expansión. La que se consideró más interesante para este trabajo fue el cambio a entrenar mediante episodios.

8. Más opciones para nodos de riesgo: Por último, se realizaron mejoras varias, enriqueciendo la representación gráfica del entorno e incluyendo la posibilidad de redefinir los nodos de riesgo.

5.5. Desarrollo de la página web

Para la creación de la página web, se optó por diseñarla utilizando Flask, e incluirla dentro de una imagen de Docker para facilitar su uso e instalación.

Estas herramientas fueron seleccionadas debido a la familiaridad que se tenía con ellas, además de que se adaptaban a las necesidades del proyecto.

Originalmente se quiso crear una aplicación Docker, que se preparase con el comando docker-compose, simplificando el trabajo de los usuarios. Sin embargo, debido a varios problemas que se encontraron durante el desarrollo, se acabó descartando esta opción por una más simple, que consistía en crear solamente una imagen Docker, y pedir a los usuarios que la construyeran y ejecutasen a continuación un contenedor con ella.

En cuanto a la estructura, se decidió dejar el código del algoritmo separado del código de la página web. Para ello, se dividieron en backend y frontend, respectivamente.

Diseño del backend

El backend consistía simplemente en los dos ficheros python del agente y del entorno. Sin embargo, se realizaron algunos cambios y adiciones al código para adaptarlo a las necesidades de la página web. En el agente no hubo modificaciones, excepto la eliminación del método main, al no ser necesario. Sin embargo, en el entorno sí que hubo varias diferencias importantes.

- La primera adición fue la posibilidad de crear un entorno sin especificar los nodos inicial y final. Como consecuencia, se tuvo que crear un nuevo método para asignar inicio y meta al entorno. Este cambio ocurrió porque se quería que el usuario de la página seleccionase el origen y destino una vez hubiese visualizado la red de ordenadores que se había generado.

- Por otro lado, se añadió la opción de guardar el renderizado del entorno como una imagen en una carpeta determinada. De ese modo, se podrían mostrar las imágenes más adelante en la página web de forma mucho más sencilla.

- Otra nueva funcionalidad que se añadió fue la opción de transformar al entorno a un formato JSON, y viceversa, para poder guardarlo como variable de sesión en la página web. Esto conllevó a tener que crear dos métodos nuevos, además de modificar el constructor de la clase.

- Por último, debido a que se guardaban las redes como variables de sesión del sitio web, tenían un límite de tamaño impuesto sobre ellas para evitar que algunos navegadores las bloqueasen. Por ello, hubo

que modificar una de las redes predefinidas, reduciendo su número de nodos, para así disminuir el tamaño de su versión JSON.

Diseño del frontend

Se diseñó un frontend relativamente simple, haciendo uso de Flask para el manejo de endpoints, y Jinja2 para las plantillas HTML. El archivo de Python con la aplicación de Flask gestiona las llamadas a los endpoints que hace el usuario en su navegador web, y prepara la plantilla HTML correspondiente. Haciendo uso de formularios HTML, la aplicación Flask recoge los datos que introduzca el usuario, y llama a los métodos necesarios del backend para preparar el entorno, realizar el entrenamiento y buscar la ruta óptima.

5.6. Realización de pruebas

Una vez desarrollado el algoritmo, se procedió a crear varias pruebas para comprobar su funcionamiento, e intentar arreglar aquellos posibles problemas que surgieran, u optimizar su ejecución.

Pruebas unitarias

Por un lado, se realizó un conjunto de pruebas unitarias sobre los archivos del código fuente. Se creó un fichero de tests unitarios para cada uno de los archivos de código, y se realizaron pruebas sobre todos los métodos existentes en dichos archivos, consistiendo en un total de 59 casos de prueba.

Para ello se hizo uso del framework de automatización de pruebas Unittest, que permitía la rápida ejecución de los tests después de modificar algún archivo fuente, pudiendo así probar el código una gran cantidad de veces habiendo escrito las pruebas una única vez. La mayoría de entornos de desarrollo tienen funcionalidades o extensiones que facilitan aún más la ejecución de tests y la visualización de los problemas encontrados.

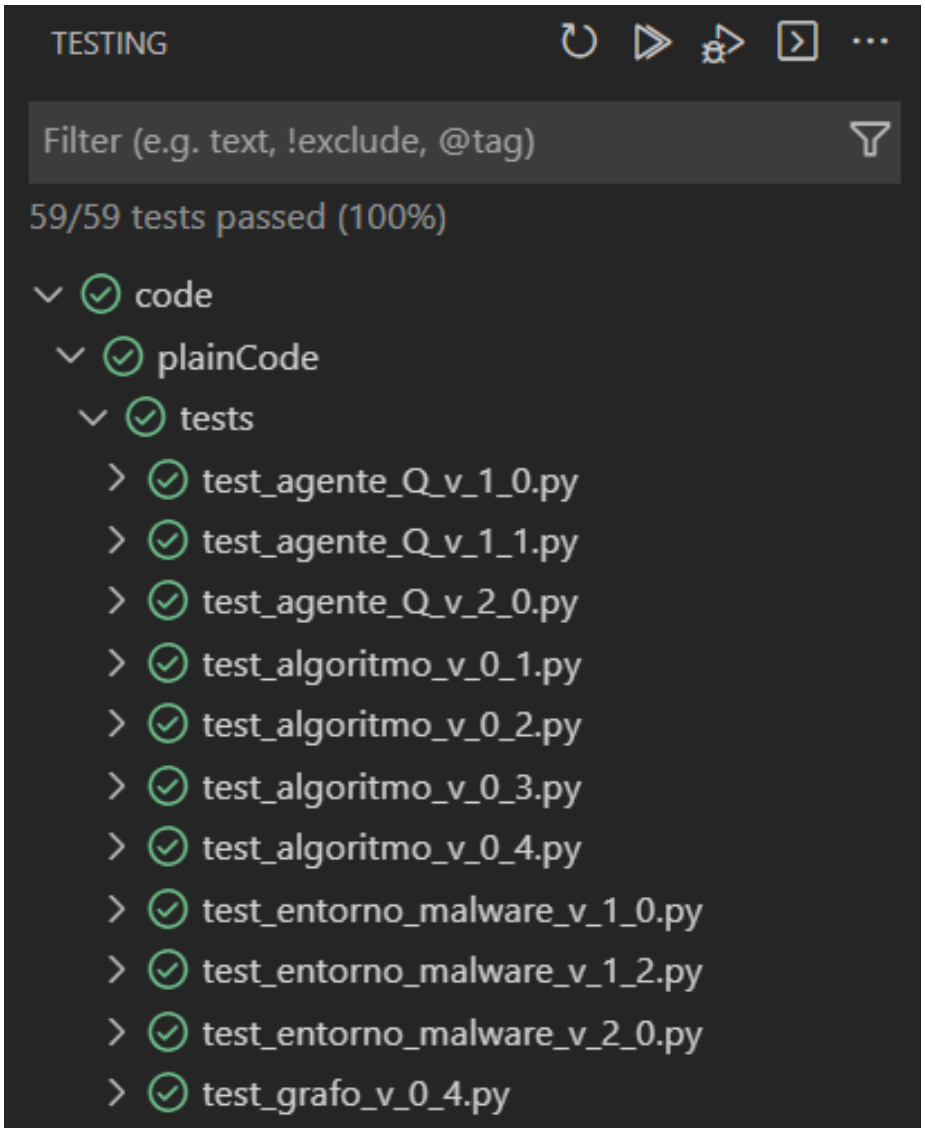

Figura 5.1: Tests ejecutados en Visual Studio Code

Estudio de valores de entrenamiento

Por otro lado, también se realizó un estudio de los valores de entrenamiento, para observar la influencia que tenían sobre los resultados obtenidos. Para ello, se definió un conjunto de valores para cada parámetro, y se probó con todas las combinaciones de esos valores en un conjunto de redes de diferentes tamaños.

Para poder comparar el rendimiento de los diferentes experimentos, se decidió registrar cuánto se había aprendido a lo largo de los episodios. Para poder representar esto, se realizaba una búsqueda al finalizar cada episodio, y se guardaba la recompensa obtenida. Al finalizar el entrenamiento se generaba una gráfica con el progreso de las recompensas, pudiendo ver así su variación a lo largo del entrenamiento.

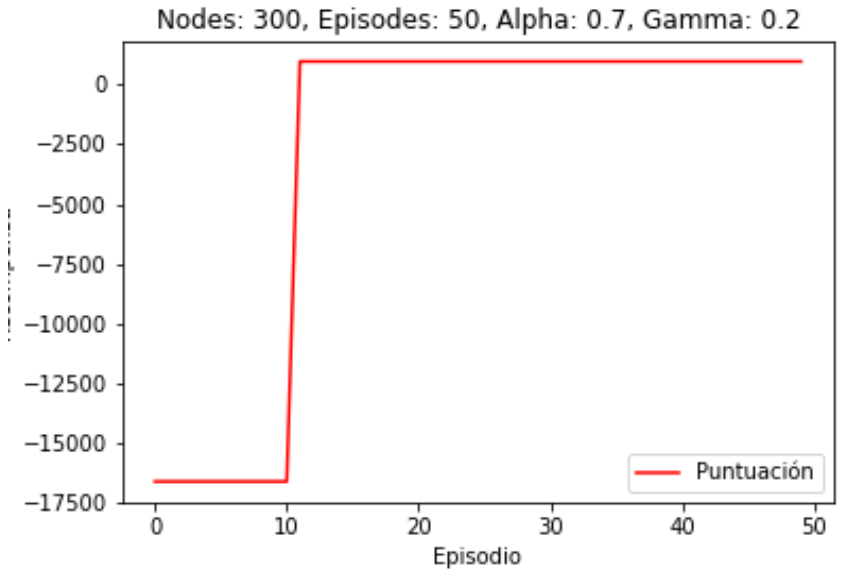

Figura 5.2: Ejemplo de gráfica obtenida al realizar uno de los experimentos

Sin embargo, debido al proceso de búsqueda de ruta y las diferentes recompensas definidas en el algoritmo, cuando no se encontraba una ruta hasta el nodo objetivo la recompensa obtenida era un valor negativo de una magnitud considerablemente mayor a la recompensa obtenida al encontrar una ruta óptima. Por lo tanto, en las gráficas lo que se observa es un salto muy elevado e instantáneo en el momento en el que se encuentra un camino válido hasta el nodo objetivo. Por ello, se realizó una comparación entre los experimentos del tiempo que se tardaba en encontrar una ruta al nodo objetivo. Detalles sobre los experimentos realizados y sus resultados pueden verse en el Apéndice D.

Trabajos relacionados

En este capítulo se presentan varios trabajos realizados que comparten campo con el proyecto, en caso de que se desee profundizar más en los diferentes temas.

Debido a las limitaciones impuestas sobre el alcance del proyecto, tanto sobre el entorno como sobre el algoritmo utilizado, y el contexto específico que se ha seleccionado, no se han encontrado trabajos similares, que permitiesen realizar comparativas entre sus resultados. Por eso, esta sección está enfocada con la idea de ofrecer diferentes puntos de partida, dado el caso de que se desee aprender más sobre las aplicaciones de la inteligencia artificial en el área de ciberseguridad, o sobre las estrategias de propagación de malware, entre otros asuntos.

Los artículos científicos que han sido seleccionados son los siguientes:

Malware Detection and Prevention using Artificial Intelligence Techniques [5]

Este artículo tuvo como propósito la investigación sobre técnicas de Inteligencia Artificial (IA) para la detección de malware. Se realizó una revisión bibliográfica en busca de artículos científicos en varias bases de datos científicas, además de bases de datos on-line como IEEE o ScienceDirect, que tratasen sobre el tema del uso de inteligencia artificial en la detección y prevención de ataques malware.

A continuación se explicaron las técnicas encontradas, hablando primero sobre las tres ramas de técnicas de detección de malware: signature-based (basado en firmas), anomaly-based (basado en anomalías) o heuristic-based (basado en heurísticas). Después, se pasa a explicar trabajos que utilizan

inteligencia artificial para dicha detección, ya sea mediante redes convolucionales gráficas, mediante machine learning extrayendo características, o incluso utilizando máquinas virtuales para la detección de intrusiones. Por último, el artículo destaca las limitaciones que tienen estos sistemas propuestos.

Se ha considerado relevante este artículo, pues, al haber realizado una búsqueda sobre trabajos en este campo, y referenciándolos en el texto, sirve como lanzadera para investigación adicional.

Impact of Network Structure on Malware Propagation: A Growth Curve Perspective [8]

En este trabajo, se toma un alcance diferente. En vez de buscar métodos o técnicas para detectar ataques de malware, el objetivo del trabajo fue analizar diferentes topologías de red para observar cuales eran más seguras frente a la propagación de un agente malware.

Para ello, se definieron primero las mecánicas de propagación de malware, indicando que el objetivo sería infectar al mayor número de dispositivos posible, pudiendo replicarse a si mismo el agente. Se explicaron las funciones y modelos que describían la propagación del malware. También se realizó un estudio previo de topologías de redes, diferenciando entre redes sociales – en concreto MySpace – y redes tecnológicas, es decir, redes propias de organizaciones.

Una vez hecho esto, se pasó a configurar y modelar las simulaciones de propagación y, con los resultados obtenidos, se creó un modelo de riesgo estructural, y se crearon estrategias de defensa basadas en dicho modelo de riesgo estructural. Al compararlas con otras estrategias de defensa, se observó como eran capaces de detener la propagación de una forma más rápida.

Reinforcement learning based stochastic shortest path finding in wireless sensor networks [27]

En el siguiente artículo, se utiliza aprendizaje por refuerzo de Q-learning para encontrar rutas dentro de redes, al igual que en este proyecto, pero el artículo decide centrarse específicamente en redes de sensores inalámbricos, y su objetivo no tiene nada que ver con malware, sino que se enfoca en evitar retrasos en el tiempo y congestiones. Igualmente, se considera que

las similitudes con el proyecto actual son lo suficientemente relevantes como para incluirse.

En el artículo, se propone un modelo estocástico, en el cual la longitud de los enlaces entre nodos va definida por una función de probabilidad. De este modo, no se puede saber con claridad cual será la recompensa por moverse a través de un enlace. Se diseñan algoritmos de Q-learning y SARSA para la navegación por este entorno, y se ejecutan. A continuación, se compara su eficiencia con otros algoritmos de enrutamiento de paquetes, midiendo el valor de regret, que consiste en la diferencia entre la recompensa obtenida y la recompensa óptima teórica.

Advanced malware propagation on random complex networks [4]

Se incluye también un último trabajo, que propone el uso de autómatas celulares para la propagación de malware dentro de redes complejas aleatorias. Primero, se diseña el autómata celular, y a continuación se definen las redes, indicando que tendrán un número fijo de dispositivos, y estos dispositivos tendrán cuatro posibles estados: susceptible, infectado, atacado y recuperado. Después de esto se realiza un estudio de la efectividad del autómata dentro de redes aleatorias complejas, variando diferentes características del entorno, como los valores de probabilidad utilizados en la generación de la red, o la selección de nodos iniciales.

Conclusiones y Líneas de trabajo futuras

En este apartado se exponen observaciones y conclusiones obtenidas al completar el trabajo, además de las áreas por las que se puede continuar el desarrollo en caso de seguir con el trabajo en el proyecto.

7.1. Conclusiones

Al finalizar el proyecto, hemos llegado a las siguientes conclusiones:

Por un lado, la mayoría de los objetivos que se definieron inicialmente han sido cumplidos satisfactoriamente. Se ha creado un entorno que representa una red de ordenadores, y el malware localizado en ella, basándose en las características definidas como necesarias para una simulación correcta. También se ha programado un agente que actúa sobre dicho entorno utilizando aprendizaje por refuerzo para encontrar la ruta óptima hasta un nodo objetivo. Además se ha diseñado una página web que sirve como interfaz para la fácil utilización de dicho agente.

Sin embargo, el objetivo de realizar pruebas y estudios sobre el funcionamiento del algoritmo no se ha podido llegar a realizar por completo, debido principalmente a una falta de tiempo. Se realizó un estudio de convergencia, pero inicialmente se pretendió también examinar otras capacidades del algoritmo, como la robustez frente a cambios en el entorno.

Este proyecto ha sido muy útil a la hora de aplicar una gran cantidad de los conocimientos obtenidos a lo largo del curso universitario, a la vez que se iban adquiriendo más conocimientos concretos para el trabajo. Se han podido aplicar conocimientos de metodologías ágiles, programación

estructurada, pruebas de código o diseño de páginas web, entre otras cosas. Para algunas de las áreas, como el uso de contenedores Docker, se han ampliado los conocimientos adquiridos en el grado para poder ajustarse a las necesidades del proyecto.

Por otro lado, el realizar este proyecto me ha ayudado a tener una primera toma de contacto con el desarrollo de trabajos a largo plazo, desde su planificación hasta su desarrollo y consecuente mejora. Ha servido para obtener experiencia a la hora de estimar el esfuerzo necesario en las diferentes tareas que formaban parte del trabajo, y el esfuerzo que se era capaz de realizar en los diferentes plazos de tiempo. Esto era algo con lo que se ha tenido dificultad, pero se ha acabado con un mayor conocimiento tanto del proceso de planificación de proyectos como de los límites de uno mismo.

No obstante, se habría preferido incluir más elementos al proyecto, y extenderlo algo más, pero no ha sido posible dentro del tiempo del que se disponía. Estas ampliaciones se exponen en el siguiente apartado.

7.2. Líneas de trabajo futuras

A la hora de ampliar el alcance del proyecto, e incluir funciones adicionales, se sugieren las siguientes avenidas de mejora:

- **Adición de fuentes de incertidumbre:** Para mostrar la adaptabilidad de los algoritmos de aprendizaje por refuerzo, sería una buena idea añadir fuentes de ruido o incertidumbre durante el entrenamiento del agente, como la variación de la peligrosidad de los nodos o la modificación de conexiones entre ellos.

- **Estudio de adaptabilidad:** Después de realizar el punto anterior, sería interesante realizar un estudio de la capacidad de adaptarse que tendría el agente programado, observando su respuesta a diferentes niveles de ruido, y si sería capaz de ajustarse apropiadamente.

- **Ampliación de los objetivos:** El objetivo actual, de infección de un único nodo concreto, es útil a la hora de proteger ese dispositivo, pero la mayoría de malware suelen tener objetivos más complejos. Por lo tanto, se propone la necesidad de infectar múltiples dispositivos objetivo, o la posibilidad de infectar equipos a lo largo del camino para reducir su riesgo en caso de tener que pasar otra vez por ellos, o ampliaciones similares al objetivo del malware.

- **Publicación de la página web:** Actualmente, la página se encuentra dentro de un contenedor Docker, requiriendo su descarga y ejecución en un dispositivo. Una posibilidad para que el proyecto sea accesible a más gente sería la publicación de la página en internet, registrando un dominio para que se pudiese acceder desde cualquier navegador. Junto a esto, se podrían incluir mejoras a la propia página, como una gestión de usuarios, mayor libertad a la hora de generar las redes, u otras formas de representar visualmente los resultados del entrenamiento.
- **Adición de complejidad al entorno:** el entorno presente en el proyecto considera todos los dispositivos conectados a la red como iguales, diferenciándose únicamente por su riesgo, y todas las conexiones iguales también. Se podría ampliar la complejidad de esta red, añadiendo diferentes características a los equipos, como switches o firewalls, y a las conexiones, que afecten al comportamiento del malware.
- **Algoritmo de Deep Reinforcement Learning:** Por último, se podría ir un paso más allá, y crear un nuevo agente que, en vez de Q-Learning, utilice aprendizaje profundo por refuerzo y redes neuronales para realizar el entrenamiento y búsqueda de rutas. Una vez creado, se podrían realizar estudios comparando ambos agentes para observar sus diferencias, y cuál obtiene mejores resultados.

Bibliografía

[1] Code Climate. Quality. https://codeclimate.com/quality/. [Internet].

[2] Codacy. The devops intelligence platform. https://www.codacy.com/. [Internet].

[3] Wikimedia Commons. File:topología de red.png — wikimedia commons, the free media repository. https://commons.wikimedia.org/w/index.php?title=File:Topolog%C3%ADa_de_red.png&oldid=544565082, 2021. [Internet].

[4] A Martín del Rey, Guillermo Hernández, A Bustos Tabernero, and A Queiruga Dios. Advanced malware propagation on random complex networks. *Neurocomputing*, 423:689–696, 2021.

[5] Md Jobair Hossain Faruk, Hossain Shahriar, Maria Valero, Farhat Lamia Barsha, Shahriar Sobhan, Md Abdullah Khan, Michael Whitman, Alfredo Cuzzocrea, Dan Lo, Akond Rahman, et al. Malware detection and prevention using artificial intelligence techniques. *2021 IEEE International Conference on Big Data (Big Data)*, pages 5369–5377, 2021.

[6] Flask. Welcome to flask. https://flask.palletsprojects.com/en/2.2.x/. [Internet].

[7] Google. Google colab. https://colab.research.google.com/. [Internet].

[8] Hong Guo, Hsing Kenneth Cheng, and Ken Kelley. Impact of network structure on malware propagation: A growth curve perspective. *Journal of Management Information Systems*, 33(1):296–325, 2016.

[9] Docker Inc. Docker. https://www.docker.com/. [Internet].

[10] GitHub Inc. Github. https://github.com/. [Internet].

[11] Jupyter. Project jupyter. https://jupyter.org/. [Internet].

[12] Matplotlib. Visualization with python. https://matplotlib.org/. [Internet].

[13] Microsoft. Visual studio code. https://code.visualstudio.com/. [Internet].

[14] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. [presentado en NIPS Deep Learning Workshop].

[15] NetworkX. Network analysis in python. https://networkx.org/. [Internet].

[16] NumPy. Numpy. https://numpy.org/. [Internet].

[17] Overleaf. Online latex editor. https://www.overleaf.com/. [Internet].

[18] The LaTeX Project. Latex – a document preparation system. https://www.latex-project.org/. [Internet].

[19] Python. random - generate pseudo-random numbers. https://docs.python.org/3/library/random.html. [Internet].

[20] Python. Unittest - unit testing framework. https://docs.python.org/3/library/unittest.html. [Internet].

[21] Python. Welcome to python.org. https://www.python.org/. [Internet].

[22] Scrum.org. What is scrum? https://www.scrum.org/resources/what-is-scrum. [Internet].

[23] David Silver. Introduction to reinforcement learning. lecture 2: Markov decision processes. https://www.deepmind.com/learning-resources/introduction-to-reinforcement-learning-with-david-silver, 2015. [Lecture].

[24] Spyder. Spyder ide. https://www.spyder-ide.org/. [Internet].

[25] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2020. [Segunda edición].

[26] Andrew S Tanenbaum. *Computer Networks*. Pearson, 1981. [Cuarta edición].

[27] Wenwen Xia, Chong Di, Haonan Guo, and Shenghong Li. Reinforcement learning based stochastic shortest path finding in wireless sensor networks. *IEEE Access*, 7:157807–157817, 2019.

[28] ZenHub. Productivity management for software teams. https://www.zenhub.com/. [Internet].