



UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA

TECNOLOGÍA DE COMPUTACIÓN

TRABAJO FIN DE GRADO

VAR: Visualización de Técnicas de Aprendizaje por Refuerzo

Eduardo Martín Izquierdo

Noviembre, 2017

VAR: VISUALIZACIÓN DE TÉCNICAS DE APRENDIZAJE POR
REFUERZO



UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA
Tecnologías y Sistemas de Información

**TECNOLOGÍA ESPECÍFICA DE
COMPUTACIÓN**

TRABAJO FIN DE GRADO

VAR: Visualización de Técnicas de Aprendizaje por Refuerzo

Autor: Eduardo Martín Izquierdo
Director: Dr. Luis Jiménez Linares

Noviembre, 2017

Eduardo Martín Izquierdo

Ciudad Real – Spain

E-mail: peduajo@gmail.com

Teléfono: 672 669 008

© 2017 Eduardo Martín Izquierdo

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Se permite la copia, distribución y/o modificación de este documento bajo los términos de la Licencia de Documentación Libre GNU, versión 1.3 o cualquier versión posterior publicada por la Free Software Foundation; sin secciones invariantes. Una copia de esta licencia está incluida en el apéndice titulado «GNU Free Documentation License».

Muchos de los nombres usados por las compañías para diferenciar sus productos y servicios son reclamados como marcas registradas. Allí donde estos nombres aparezcan en este documento, y cuando el autor haya sido informado de esas marcas registradas, los nombres estarán escritos en mayúsculas o como nombres propios.

TRIBUNAL:

Presidente:

Vocal:

Secretario:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENTE

VOCAL

SECRETARIO

Fdo.:

Fdo.:

Fdo.:

RESUMEN

En la actualidad, la Inteligencia Artificial cada vez va cogiendo más peso en nuestra sociedad, llegando al posible punto en el que sean capaces de sustituir a los humanos en la mayoría de los trabajos.

Esa capacidad de aprender actividades, con una eficiencia y rapidez mayor que los humanos, es gracias al Aprendizaje por Refuerzo, un área del Aprendizaje Automático.

En este trabajo se muestra una visualización de los conceptos básicos del Aprendizaje por Refuerzo, donde se espera que el usuario pueda apreciar lo potente y alucinante que puede ser el proceso de aprendizaje llevado a cabo por un agente. Por lo tanto, se demuestra que el algoritmo QL junto a un agente pueden dar lugar a aplicaciones muy fructíferas en el futuro.

ABSTRACT

Nowadays, Artificial Intelligence is increasingly taking on more weight in our society, reaching the possible point where it can replace humans in most of the jobs.

This ability to learn activities, with a greater efficiency and speed than humans, is of because Reinforcement Learning, an area of Machine Learning.

In this project, a visualization of the main concepts of Reinforcement Learning is shown, where it's expected that the user could appreciate how powerful and amazing the learning process carried out by an agent can be. Therefore, it is demonstrated that the QL algorithm together with an agent can lead to very fruitful applications in the future.

AGRADECIMIENTOS

Después de estar a punto de conseguir este título, hay que decir que la trayectoria no ha sido nada fácil, y tengo que agradecer a aquellos que siempre han intentado motivarme y evitar que tire la toalla.

Como agradecimiento más especial, es el destinado a mis padres. Sin ellos ningún camino hubiera empezado. Han sido los grandes causantes de la persistencia en mi trabajo.

A mis mejores compañeros de la facultad. Siempre son necesarias unas risas además de estar aplicado en el estudio, gente que se ayude mutuamente y que posiblemente se conviertan en amigos para toda la vida.

A todos aquellos profesores que han intentado dar lo mejor de sí mismos para ofrecer su conocimiento. En especial a Luis, mi tutor, que ya en su asignatura me motivó hacia el mundo de la Inteligencia Artificial y ahora darle las gracias por ofrecerme un trabajo, que al principio no parecía demasiado potente, pero que al acabarlo me he dado cuenta de que ha sido el trabajo más interesante y educativo que he hecho hasta ahora.

Eduardo Martín Izquierdo

A mis padres

INDICE GENERAL

RESUMEN	V
ABSTRACT	VII
AGRADECIMIENTOS	IX
INDICE GENERAL	XIII
ÍNDICE DE FIGURAS	XVII
ÍNDICE DE CUADROS	XIX
ÍNDICE DE LISTADOS	XXI
1. INTRODUCCIÓN	1
1.1 MOTIVACIÓN	1
1.2 PROBLEMA	3
1.3 ESTRUCTURA DEL DOCUMENTO	4
2. OBJETIVOS	5
2.1 OBJETIVO PRINCIPAL	5
2.2 OBJETIVOS ESPECÍFICOS	5
2.2.1 Elegir y usar una aplicación existente para la edición del mundo	5
2.2.2 Establecer las formas de mostrar el ApR	5
2.2.3 Implementación del algoritmo QL	6
2.2.4 Desarrollo de una API REST	6
2.2.5 Aplicación web	6
2.3 HERRAMIENTAS Y MEDIOS UTILIZADOS	6
2.3.1 Medios hardware	6
2.3.2 Medios software	7
3. ANTECEDENTES	11
3.1 ApR	11
3.1.1 Historia del ApR	11
3.1.2 Proceso de Decisión de Markov	13
3.1.3 ApR Activo	15
3.1.4 Algoritmo QL	19
3.2 DEEP-Q LEARNING	20
3.3 VIDEOJUEGOS BASADOS EN CELDAS	21
3.3.1 Historia	22
3.4 APLICACIÓN WEB REST	23
3.5 BASES DE DATOS RELACIONALES	25
3.6 OBJECT RELATIONAL MAPPING	26

3.6.1 SQLAlchemy	26
4. MÉTODO DE TRABAJO	29
4.1 METODOLOGÍAS ÁGILES	29
4.1.1 Diferencia entre metodologías ágiles y tradicionales	30
4.1.2 Manifiesto ágil: Valores	30
4.1.3 Manifiesto ágil: Principios	31
4.2 EXTREME PROGRAMMING	33
4.2.1 Las reglas de XP	34
4.2.2 Los valores de XP	38
4.2.3 Roles en XP	39
4.2.4 El proceso XP	42
4.2.5 Diferencias entre XP y Scrum	43
4.3 APLICACIÓN DE LA METODOLOGÍA	44
4.3.1 Asignación de los roles	44
4.3.2 Iteraciones del proyecto	44
5. RESULTADOS	49
5.1 FUNCIONALIDADES DEL SISTEMA	49
5.2 ITERACIÓN 1: LECTURA DEL MAPA DE ENTRENAMIENTO	50
5.2.1 Estudio y uso del entorno Tiled	50
5.2.2 Estudio del entorno Phaser	51
5.2.3 Edición adecuada del mapa en Tiled	53
5.2.4 Implementación de la lectura del mapa con Phaser	56
5.3 ITERACIÓN 2: IMPLEMENTAR EN PYTHON EL ALGORITMO QL	62
5.3.1 Implementación del mundo en Python	63
5.3.2 Implementación del algoritmo QL en Python	65
5.4 ITERACIÓN 3: IMPLEMENTACIÓN DEL ALGORITMO QL JUNTO A LA LIBRERÍA PHASER	73
5.4.1 Adaptación del juego Phaser al algoritmo QL	73
5.5 ITERACIÓN 4: IMPLEMENTACIÓN DEL SERVIDOR REST Y DE LAS INTERFACES GRÁFICAS	80
5.5.1 API REST	81
5.5.2 Implementación de la interfaz gráfica para iniciar sesión	83
5.5.3 Implementación de la interfaz gráfica del entrenamiento	84
5.6 ITERACIÓN 5: IMPLEMENTACIÓN DE LA BASE DE DATOS Y FINALIZACIÓN DEL SERVIDOR REST	86
5.6.1 Implementación de la interfaz gráfica del registro	87
5.6.2 Implementación de la interfaz gráfica de la carga de mapas	87
5.6.3 Implementación de la base de datos	88

5.6.4 Finalización de la implementación del servidor REST	89
6. CONCLUSIONES	97
6.1. LOGRO DE OBJETIVOS	97
6.2 MEJORAS FUTURAS	98
ANEXO A HISTORIAS DE USUARIO	99
ANEXO B MANUAL DE USUARIO	109
REFERENCIAS.....	113

ÍNDICE DE FIGURAS

Figura 1.1: Ciclo del aprendizaje.	2
Figura 1.2: Evolución del aprendizaje.	3
Figura 2.1: Información básica del equipo.	7
Figura 2.2: Logo de Phaser.	8
Figura 2.3: Logo de GitHub.	9
Figura 2.4: Logo de PyCharm.	10
Figura 3.1: El contexto del ApR.	12
Figura 3.2: Ejemplo de un MDP simple.	14
Figura 3.3: Entorno con la utilidad de cada estado.	16
Figura 3.4: error valor cuadrático medio (RMS) en los valores de la utilidad y la pérdida de la correspondiente política.	18
Figura 3.5: Capturas de pantalla de 5 juegos Atari 2600.	20
Figura 3.6: Ejemplo de conjunto de patrones.	22
Figura 3.7: Ejemplo de videojuego basado en celdas.	23
Figura 3.8: Esquema Cliente-Servidor en API REST.	24
Figura 3.9: Ejemplo de configuraciones de Frameworks web con bases de datos.	26
Figura 4.1: Comparación de ciclo de desarrollo ágil y tradicional.	32
Figura 4.2: Ejemplo de historia de usuario.	33
Figura 4.3: Desarrollo iterativo.	36
Figura 4.4: Lista de roles en XP.	39
Figura 4.5: Resumen de tareas.	40
Figura 5.1: Esquema funcional del sistema.	49
Figura 5.2: Interfaz gráfica de Tiled.	52
Figura 5.3: Mapa Tiled.	53
Figura 5.4: Capas del mapa.	55
Figura 5.5: Creación de objetos en el mapa.	55
Figura 5.6: Diagrama de clases.	57
Figura 5.7: Mapa Tiled.	63
Figura 5.8: Mapa creado con la librería TkInter.	66
Figura 5.9: Ejemplo de matriz-Q.	67
Figura 5.10: Gráficas de dos aprendizajes.	74
Figura 5.11: Esquema del cálculo de ΔQ	77
Figura 5.12: Gráficas de dos aprendizajes.	79
Figura 5.13: Interfaz gráfica del login.	84

Figura 5.14: Interfaz gráfica del entrenamiento.....	87
Figura 5.15: Ejemplo de muestra de círculos.	89
Figura 5.16: Interfaz gráfica del registro.	89
Figura 5.17: Interfaces gráficas para cargar el mapa.	90
Figura B.1: Enlace para registrarse.	109
Figura B.2: Campos a rellenar en el registro del usuario.....	109
Figura B.3: Formulario para añadir un nuevo mapa.....	110
Figura B.4: Datos a rellenar para el entrenamiento.	111
Figura B.5: Ejemplo de matriz-Q.....	111

ÍNDICE DE CUADROS

Cuadro 4.1: Diferencia entre metodologías ágiles y tradicionales.	31
Cuadro 4.2: Historias de usuario de la iteración 1.	45
Cuadro 4.3: Historias de usuario de la iteración 2.	46
Cuadro 4.4: Historias de usuario de la iteración 3	46
Cuadro 4.5: Historias de usuario de la iteración 4	46
Cuadro 4.6: Historias de usuario de la iteración 5	47
Cuadro A.1: Historia de usuario: Investigación del entorno Tiled.	99
Cuadro A.2: Historia de usuario: Investigación del entorno Phaser.	99
Cuadro A.3: Historia de usuario: Creación de mapas Tiled de forma correcta.....	100
Cuadro A.4: Historia de usuario: Implementación del mundo en JavaScript.	100
Cuadro A.5: Historia de usuario: Investigación y estudio sobre el algoritmo QL.	101
Cuadro A.6: Historia de usuario: Implementación del mundo en Python.	101
Cuadro A.7: Historia de usuario: Implementación del algoritmo QL en Python.....	102
Cuadro A.8: Historia de usuario: Adaptación del mundo Phaser para su aplicación en el algoritmo QL.	102
Cuadro A.9: Historia de usuario: Implementar el algoritmo QL junto al mundo de Phaser.....	103
Cuadro A.10: Historia de usuario: Investigación sobre la librería Flask.....	103
Cuadro A.11: Historia de usuario: Investigación sobre la librería Bootstrap.....	104
Cuadro A.12: Historia de usuario: Iniciar la implementación del servidor REST.	104
Cuadro A.13: Historia de usuario: Implementar la interfaz gráfica del inicio de sesión.....	105
Cuadro A.14: Historia de usuario: Implementar la interfaz gráfica del entorno de entrenamiento.	105
Cuadro A.15: Historia de usuario: Investigación y estudio sobre la librería SQLAlchemy.	106
Cuadro A.16: Historia de usuario: Implementar la interfaz de registro y carga de mapas.....	106
Cuadro A.17: Historia de usuario: Implementar la base de datos con SQLAlchemy.	107
Cuadro A.18: Historia de usuario: Acabar la implementación del servidor REST.	107

ÍNDICE DE LISTADOS

Listado 3.1: Pseudocódigo del algoritmo QL.....	19
Listado 5.1: Inicialización del objeto que contiene el juego.	57
Listado 5.2: Código del método addStates().	57
Listado 5.3: Código del método startPreload().	58
Listado 5.4: función prototype del estado Preload.....	58
Listado 5.5: Código para añadir el mapa Tiled y el conjunto de patrones.....	59
Listado 5.6: Código para añadir el Sprite del agente.	59
Listado 5.7: Código para saber si el agente se encuentra en el césped.....	60
Listado 5.8: Código de la función getEstadoBot().	61
Listado 5.9: Código para ajustar punto en el eje X.	61
Listado 5.10: Código de la función findObjectsByType()	62
Listado 5.11: Código para la creación del objeto mundo.	63
Listado 5.12: Código para inicializar las listas de los tipos de elementos.....	64
Listado 5.13: Código de inicialización de variables para el algoritmo QL.	68
Listado 5.14: Código de la función inicializarTablas().	69
Listado 5.15: Código de la función moverse().	69
Listado 5.16 : Código de la función maxQ().	70
Listado 5.17: Código de la función incQ().	71
Listado 5.18: Código de la función run(), donde está el algoritmo QL.	72
Listado 5.19: incrementos para las posiciones del agente.	75
Listado 5.20: Código de la función moverse().	75
Listado 5.21: Código de la función mejorValoración().	76
Listado 5.22: Código de la función difValoraciones().	77
Listado 5.23: Código de la función update()	80
Listado 5.24: Crear la instancia de Flask.	82
Listado 5.25: Código para arrancar el servidor.	82
Listado 5.26: Código para la ventana login.	83
Listado 5.27: Código para la ventana del entrenamiento.....	83
Listado 5.28: Código para importar Bootstrap con Jinja2.....	84
Listado 5.29: Código para crear la tabla Users.....	90
Listado 5.30: Código del archivo database.py.....	91
Listado 5.31: Código para cerrar sesiones de la base de datos.	91
Listado 5.32: Código para hacer la consulta y acceder a la cuenta del usuario.....	91
Listado 5.33: Código para añadir un usuario a la base de datos.....	91

Listado 5.34: Código para mostrar la interfaz de carga de mapas. 93

Listado 5.35: Código de los decoradores getMap() y getTileset(). 95

Listado 5.36: Ejemplo de uso de Jinja2 para tomar un argumento. 95

Listado 5.37: Código para pasar los argumentos con Jinja2 a archivos JavaScript. 95

1. INTRODUCCIÓN

Los seres vivos muestran siempre un tipo de comportamiento del modo que realizan una acción como respuesta a las señales que reciben del entorno en donde viven. Algunos, además, cambian su comportamiento con el tiempo, por lo que se puede decir que han aprendido del entorno. La idea de aprender a partir de interactuar con el entorno es probablemente la primera que ocurre cuando se piensa sobre la naturaleza del aprendizaje. La realización de estas actividades produce una riqueza de información sobre las relaciones causa y efecto, sobre las consecuencias de las acciones, y sobre qué hacer con el fin de alcanzar objetivos.

Aprender de la interacción con el entorno es una idea fundamental que subyace a casi todas las teorías del aprendizaje dentro del ámbito de la Inteligencia Artificial. Por lo que la utilización de un enfoque capaz de aprender del entorno, como es el Aprendizaje por Refuerzo (ApR), es un problema cuya importancia ha ido aumentando de forma progresiva en los últimos años, y mucho más ahora con el auge de los sistemas cambiantes, online, interactivos con grandes capacidades de cálculo y almacenamiento.

En un sistema estándar, en el ámbito del ApR, existe un agente situado en un entorno que puede percibir, mediante un conjunto de sensores, y transformarlo con las acciones que pueden realizar un conjunto de actuadores. Si consideramos un entorno discreto, cada interacción del agente recibe como entrada una colección de valores desde sus sensores que configurarán su estado actual ($s \in S$), y selecciona una acción ($a \in A$) de todas las posibles en dicha situación. La ejecución de una determinada acción (mediante sus actuadores) cambia el estado S' , recibiendo una señal de refuerzo o recompensa ($r \in R$). En Fig. 1.1 se ilustra el ciclo de este aprendizaje.

1.1 MOTIVACIÓN

Una política óptima es una política que maximiza la recompensa total esperada. La tarea del ApR es utilizar las recompensas observadas para aprender una política óptima para el entorno donde el agente se encuentra. En Fig. 1.2 se ilustra la evolución de la recompensa total por episodio (o experiencia), mostrando como converge en la política óptima a partir del episodio 35.

En una gran cantidad de entornos complejos, el aprendizaje por refuerzo es el único enfoque posible para entrenar un agente para que funcione de forma adecuada.

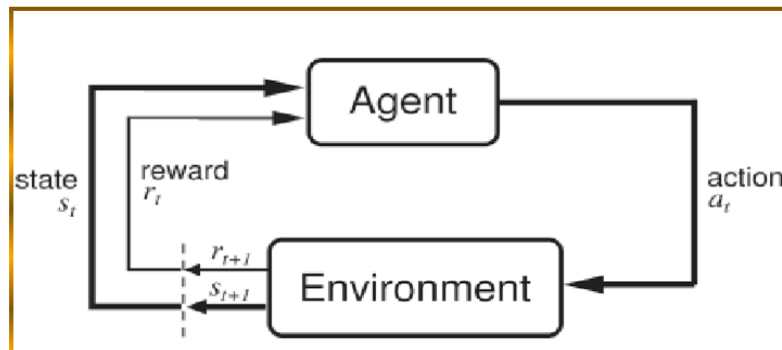


Figura 1.1: Ciclo del aprendizaje.

Hay tres diseños de agentes en este enfoque:

- **Agente basado en la utilidad:** Este agente aprende una función de utilidad que la usará para seleccionar aquellas acciones que maximizarán la utilidad.
- **Agente de QL:** Aprende una función *acción-valor*, la cual le da la utilidad esperada al realizar una acción determinada en un estado dado.
- **Agente reactivo:** Aprende una política que da lugar a una relación directa entre los estados y las acciones.

El principal inconveniente del agente basado en la utilidad es que necesita un modelo del entorno. Sin embargo, el agente de QL tiene la posibilidad de comparar los valores de sus posibles acciones sin tener que saber sus resultados, por lo que es una política que no necesita un modelo del entorno. En el aprendizaje pasivo, la política del agente está fijada y la tarea es aprender las utilidades de los estados, mientras que en el aprendizaje activo el agente aprenderá más de su entorno cuantas más experiencias tenga en él.

Por lo tanto, en el presente Trabajo Fin de Grado (TFG) se desarrolla la técnica para implementar un agente de QL, así como la visualización de su aprendizaje.

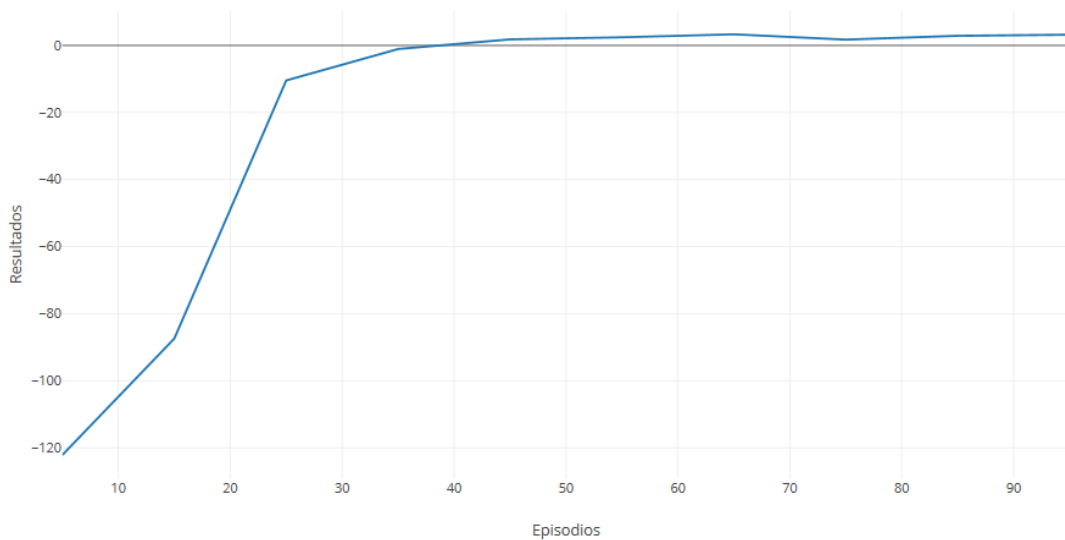


Figura 1.2: Evolución del aprendizaje.

1.2 PROBLEMA

El problema de aprender una política óptima para un entorno determinado donde se encuentre el agente pasa por calcular una matriz-Q a partir de la experiencia. Esta matriz-Q representa nada más y nada menos que a la función acción-valor que aprende el algoritmo QL.

El algoritmo QL es el principal partícipe de este cálculo con la llamada ecuación de Bellman:

$$Q(a, s) \leftarrow Q(a, s) + \alpha(R(s) + \gamma \max_{a'} Q(a', s') - Q(a, s)) \quad (1.1)$$

Siendo $Q(a, s)$ la función-Q en un determinado estado s con una determinada acción a , $R(s)$ es la función de recompensa, que especifica la recompensa de cada estado, α la velocidad de aprendizaje y γ el factor de descuento. Los parámetros α y γ se explicarán con mayor extensión más adelante, y ajustándolos podemos determinar la velocidad a la que el agente aprende y la importancia de cada acción que realiza el agente, respectivamente.

Para mostrar el aprendizaje de este agente en un entorno determinado, se propone desarrollar un entorno 2D donde se visualice a un agente aprendiendo la política óptima con el fin de encontrar un tesoro o premio que se encontrará en un punto aleatorio del mapa. Como se ha explicado antes, la principal característica brillante de este algoritmo es que se

adapta a cualquier entorno introducido por el usuario, por lo que se podrá observar la gran maleabilidad de este.

1.3 ESTRUCTURA DEL DOCUMENTO

Este TFG está estructurado con los siguientes capítulos:

Capítulo 1: Introducción

Se muestra una breve presentación del ApR, así como su motivación para su uso en este proyecto y problema general que se intenta solucionar.

Capítulo 2: Objetivos

En este apartado se muestran los objetivos principales y específicos para este TFG para solucionar de forma óptima la problemática de este TFG. También se mostrarán los medios hardware y software que han sido utilizados para el desarrollo del presente TFG.

Capítulo 3: Antecedentes

Se muestra un análisis de la situación actual o estado del arte de todos los elementos del proyecto para entender su autonomía y funcionamiento. Entre los elementos se expone: ApR, algoritmo QL, etc.

Capítulo 4: Metodología

Se explica el método de desarrollo utilizado para el desarrollo del TFG y las distintas fases llevadas a cabo para la resolución de los problemas.

Capítulo 5: Resultados

En esta sección se explica la aplicación del método de desarrollo presentado en el capítulo 4 con elementos como modelos, diagramas, especificaciones, etc. Se muestra el algoritmo QL implementado, el servidor y el cliente integrados, así como sus resultados experimentales.

Capítulo 6: Conclusiones

Se analizan y presentan todas las conclusiones obtenidas tras la completa realización del TFG. También se incluyen posibles mejoras y extensiones del mismo.

2. OBJETIVOS

En este capítulo se enumeran las diferentes metas a conseguir durante el desarrollo de este TFG, además de los medios software y hardware que han sido utilizados para el desarrollo del presente TFG. Hay que resaltar que este TFG se trata de una aplicación docente de verificación, por lo que se necesita acotar el alcance de éste.

2.1 OBJETIVO PRINCIPAL

El objetivo principal de este TFG es construir una aplicación visual que sobre un mundo 2D, un agente alcance un objetivo concreto mostrando los conceptos y acciones básicas del ApR.

Para desarrollar este algoritmo de ApR se utilizará el algoritmo QL, que es el más adecuado para mostrar la visualización de este aprendizaje ya que, como se explica con anterioridad, es un algoritmo adaptable para una amplia variedad de entornos.

Además, el usuario podrá cambiar los parámetros del aprendizaje, así como editar su propio mundo e introducirlo a la aplicación con el fin de que éste pueda observar el aprendizaje a su gusto.

2.2 OBJETIVOS ESPECÍFICOS

Para llegar a conseguir el objetivo general del TFG, será necesario llevar a cabo la consecución de los objetivos secundarios. Los objetivos específicos son los siguientes:

2.2.1 Elegir y usar una aplicación existente para la edición del mundo

Para que la aplicación pueda leer los mapas que el usuario introduce, éste necesita editarlos de forma precisa. Para ello se elige una aplicación existente para la edición de mapas formados por celdas, donde cada celda es una imagen, generando así un archivo JSON.

2.2.2 Establecer las formas de mostrar el ApR

El mapa que el usuario ha editado necesita ser leído por la aplicación. Para ello hay que elegir una librería de JavaScript existente capaz de analizar este archivo JSON. Cuando lo analiza, es capaz de mostrar todos los elementos del mapa y añadirle sprites para que cuando empiece el entrenamiento se genere una visualización acorde al ApR.

2.2.3 Implementación del algoritmo QL

Se deberá implementar el algoritmo QL para que el agente pueda aprender una política óptima en el mundo 2D. El algoritmo QL necesita unos parámetros que introduce el usuario para que éste pueda ver las variaciones del aprendizaje. Este algoritmo QL tiene que estar definido para mundos donde el agente pueda realizar únicamente 4 movimientos: arriba, abajo, izquierda y derecha.

2.2.4 Desarrollo de una API REST

Con el fin de poder utilizar la aplicación desarrollada, se implementará una API REST. A partir de esta API REST, se podrá usar el algoritmo QL para el mapa que el usuario determine. Posteriormente la API REST se desplegará en un servidor en la nube.

2.2.5 Aplicación web

En el lado del servidor se deberá definir la estructura de la página web y desarrollar la funcionalidad de la aplicación, integrando el algoritmo QL junto a la librería de JavaScript para la lectura de mapas. También se conectará a una base de datos para los usuarios de la página web, que podrán registrarse e iniciar sesión debidamente. La base de datos también deberá almacenar los mapas añadidos por el usuario y las características de éstos para así agilizar la carga cuando el usuario desee utilizarlos de nuevo.

En el lado del cliente se deberá definir el diseño web de la aplicación para que sea fácil de usar para los usuarios. En la interfaz principal de la aplicación se mostrará el mapa, así como los resultados que se vayan obteniendo y los parámetros necesarios para el algoritmo.

2.3 HERRAMIENTAS Y MEDIOS UTILIZADOS

2.3.1 Medios hardware

Ordenador de uso profesional

- Memoria RAM: 8.00 GB
- Procesador: Intel Core i7-3537U CPU 2.00 GHz 2.50 GHz
- Disco Duro: 1TB
- Tarjeta Gráfica: Nvidia GeForce 720M 2GB GDDR3

2.3.2 Medios software

Sistema Operativo Windows 10 Home

El Sistema Operativo (SO) usado para desarrollar la documentación del presente TFG ha sido el Windows 10 en su edición Home. En Fig. 2.1 se muestra la información básica del equipo.

Sistema Operativo Ubuntu Desktop 14.04

Ubuntu es una distribución del sistema operativo GNU/Linux y es distribuido como software libre. Este SO es utilizado para la implementación del software.

JavaScript

JavaScript es un lenguaje de scripting multiplataforma y orientado a objetos. Dentro de un ambiente de host, JavaScript puede conectarse a los objetos de su ambiente y proporcionar control programático sobre ellos. En el TFG se utiliza para la implementación del cliente web, así como en el uso de la librería Phaser para la lectura de mapas y la implementación del algoritmo QL.

Phaser

Phaser es un Framework gratis y sencillo para el desarrollo de videojuegos en HTML5 (Fig. 2.2). Es usado en el TFG para la lectura del mapa JSON.



Figura 2.1: Información básica del equipo.

HTML5

HTML5 es un lenguaje markup usado para estructurar u presentar contenido para la web. Es uno de los aspectos fundamentales para el funcionamiento de los sitios web. En el TFG se ha usado para la implementación de la interfaz del cliente web.

Tiled

Tiled es un editor de mapas de celdas de uso general. Está destinado a ser utilizado para editar mapas de cualquier juego basado en celdas. Se utiliza en este proyecto para la creación de mapas JSON que sirven como ejemplo para el desarrollo del proyecto global.

TkInter

Es una librería de Python que permite la edición de entornos gráficos. Ha sido utilizado en Python para la creación del prototipo de un mapa dividido en celdas.

JavaScript Object Notation

JavaScript Object Notation (JSON) es un formato para el intercambio de datos, básicamente JSON describe los datos con una sintaxis dedicada para identificar y gestionar los datos. Como se menciona antes, es el formato en el que se crean los mapas del proyecto.



Figura 2.2: Logo de Phaser.

Flask

Flask es un Framework minimalista escrito en Python que permite crear aplicaciones web. Está basado en el kit de herramientas Werkzeug y el motor de plantillas Jinja2. Es el principal partícipe para la implementación de la API REST.

Python

Python es un lenguaje de programación interpretado. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y programación funcional. En este TFG se ha usado para el desarrollo de la API REST con Flask, así como para la implementación de la base de datos.

SQLAlchemy

SQLAlchemy es el kit de herramientas de Python SQL y ORM que ofrece a los desarrolladores de aplicaciones toda la potencia y flexibilidad de SQL. Es usado en este TFG para ayudar a la implementación de una base de datos SQLite.



Figura 2.3: Logo de GitHub.

Git

Git es un sistema de control de versiones de código abierto y gratuito para manejar desde proyectos pequeños a muy grandes, con velocidad y eficiencia.

GitHub

GitHub es una plataforma de desarrollo inspirada en la forma en que se trabaja. Desde el código abierto hasta el negocio, se puede alojar y revisar códigos, administrar proyectos y crear software (Fig. 2.3).

Bootstrap

Bootstrap es un framework o conjunto de herramientas de Código abierto para el diseño de sitios y aplicaciones web. Contiene plantillas de diseño para ayudar al desarrollo de la interfaz web. En el TFG se ha usado junto a HTML5 para el desarrollo de la interfaz web.

JQuery

JQuery es una librería de JavaScript de código abierto y sirve para simplificar la tarea de programar en JavaScript. En este TFG se ha usado para el desarrollo de la funcionalidad de elementos de la interfaz, como los botones.

Cascading Style Sheets

Cascading Style Sheets (CSS) es un lenguaje de estilos que define la presentación de los documentos HTML5. En el presente TFG es usado para definir el estilo de la interfaz web.



Figura 2.4: Logo de PyCharm.

PyCharm

Es un IDE especialmente para desarrollo de Python, pero también permite el desarrollo en CSS, HTML5 y JavaScript. Es desarrollado por la compañía checa JetBrains. Se ha usado para el desarrollo del código del presente TFG (Fig. 2.4).

Microsoft Word

Microsoft Word es un procesador de textos desarrollado por Microsoft. Se ha usado en este TFG para el desarrollo de la presente documentación.

GIMP

En inglés GNU Image Manipulation Program, es un programa libre de código abierto utilizado para el retoque y edición de imagen, para el dibujo de forma libre, la conversión entre diferentes formatos de imagen, entre otras tareas. Es usado en el presente TFG para la edición de las tablas e ilustraciones de esta documentación.

3. ANTECEDENTES

En este capítulo se explican los fundamentos básicos sobre los que se ha desarrollado el proyecto. Para ello se hace un análisis detallado de todas estas tecnologías en el contexto que engloba este proyecto. Primero se hace un análisis detallado sobre el ApR, extendiéndonos en el algoritmo QL. Posteriormente se muestran avances actuales sobre el uso del algoritmo QL con redes neuronales. A continuación, se explica la tecnología para la definición y lectura de los mundos. Por último, se hará un análisis del uso de la API REST para desplegar el proyecto, así como una pequeña introducción de las bases de datos relacionales.

3.1 ApR

El Aprendizaje por Refuerzo es un área del aprendizaje automático inspirada en la psicología conductista. Trata de simular el aprendizaje de sistemas biológicos reales en un entorno determinado a partir de acciones que devuelven una recompensa o refuerzo que puede ser bueno o malo.

Las recompensas o refuerzos sirven para definir políticas óptimas en procesos de decisión de Markov (MDPs). Una política óptima es una política que maximiza la recompensa total esperada. La tarea del ApR es utilizar las recompensas observadas para aprender una política óptima para el entorno donde el agente se encuentra.

3.1.1 Historia del ApR

La Fig. 3.1 muestra las distintas disciplinas académicas que han contribuido al ApR. Las más importantes son dos:

1. Control óptimo (parte izquierda).
2. Aprendizaje animal por prueba y error (parte derecha).

Los problemas de control óptimo han sido llevados a cabo por la programación dinámica (Bellman, 1957). El aprendizaje por prueba y error tiene sus raíces en la psicología, especialmente en el condicionamiento clásico e instrumental. Como consecuencia, el control óptimo estuvo desde el principio gobernado por enfoques altamente algorítmicos o matemáticos, mientras que para el aprendizaje animal tardaron mucho más tiempo en desarrollarse los primeros modelos matemáticos [2].



Figura 3.1: El contexto del ApR.

El control óptimo y el condicionamiento instrumental trataron con problemas de control basados en retroalimentación. Sin embargo, el condicionamiento clásico trató con problemas de predicción única porque la respuesta del animal no influenciaba el experimento, es decir, no influenciaba el entorno. Un buen resumen que relaciona los enfoques algorítmicos con los experimentos reales de condicionamiento clásico fue dado por Balkenius y Moren en 1998.

A partir del estudio interdisciplinario de estos dos campos, apareció un método computacional muy influyente, llamado método de aprendizaje de Diferencia Temporal (TD) (Witten 1977, Sutton y Barto 1981). El aprendizaje TD fue originalmente asociado al aprendizaje animal, donde un refuerzo de ocurrencia temprana, también denominado estímulo condicionado, necesita ser asociado con un estímulo incondicionado que ocurre más tarde creando una situación donde las diferencias temporales de una función de valor necesitan ser evaluadas. El objetivo de esta computación es asegurar que después del aprendizaje, el estímulo condicionado se convierta en un predictor del incondicionado.

Mientras que el aprendizaje TD fue originalmente diseñado para tratar con los problemas de predicción, también fue usado para resolver problemas de control óptimo. De particular

interés es el trabajo de Watkins (1989) sobre Q-Learning, un algoritmo de control basado en la diferencia temporal.

Fue esencialmente el trabajo de Klopff (1972, 1975, 1982, 1988), que comenzó a acercar los métodos TD a las teorías de aprendizaje animal. También introdujo la diferencia entre retroalimentación evaluativa y no evaluativa, donde asoció la retroalimentación evaluativa a un entorno que no produce ninguna evaluación. La retroalimentación que llega del entorno a los sensores de la criatura solo puede ser no evaluativa. Toda evaluación, en este caso, debe ser realizada solo internamente por el mismo animal. Ya que los animales no reciben retroalimentación evaluativa, ApR parecería ser un ejemplo de aprendizaje sin supervisar. Sin embargo, esta formulación oculta el problema de cómo definir realmente el entorno.

Los métodos de aprendizaje TD necesitan predecir recompensas futuras [9]. Con el fin de alcanzar esto, el aprendizaje TD usa funciones de valor $V(t)$ que asignan valores a estados y entonces calcula el cambio de estos valores por medio de una derivada temporal. Como consecuencia, estos métodos están relacionados con los métodos de correlación basados en el aprendizaje diferencial de Hebb (en la parte derecha de la Fig. 3.1), donde un peso sináptico cambia por la correlación entre sus señales de entrada con la derivada de su salida. Tales reglas fueron principalmente discutidas por Kosco (1986) y Kopf (1986, 1988).

3.1.2 Proceso de Decisión de Markov

El agente en la toma de decisiones sigue un Proceso de Decisión de Markov (MDP). Un MDP proporciona un marco matemático para modelar la toma de decisiones en situaciones donde los resultados son parcialmente aleatorios y parcialmente están bajo el control de un responsable de la toma de decisiones.

Un MDP es un proceso de control estocástico en tiempo discreto [7]. En cada paso de tiempo el proceso está en algún estado s , y el agente que toma las decisiones puede elegir cualquier acción a que esté disponible en s . El proceso responde en el paso siguiente moviéndose aleatoriamente a un nuevo estado s' y dando al agente una recompensa correspondiente $R_a(s, s')$.

La probabilidad de que el proceso se mueva a su estado s' está influenciado por la acción elegida. Específicamente, está dado por la función de transición de estado $P_a(s, s')$. Por lo tanto, el próximo estado s' depende del estado actual s y la acción a del agente que toma las decisiones. Por lo tanto s y a son condicionalmente independientes a todos los estados y

acciones anteriores; es decir, las transiciones de estado de un MDP satisfacen la propiedad de Markov. Un proceso estocástico tiene la propiedad de Markov si la distribución de probabilidad de los estados futuros del proceso depende sólo del estado actual, y no también de la secuencia de eventos que lo precedieron.

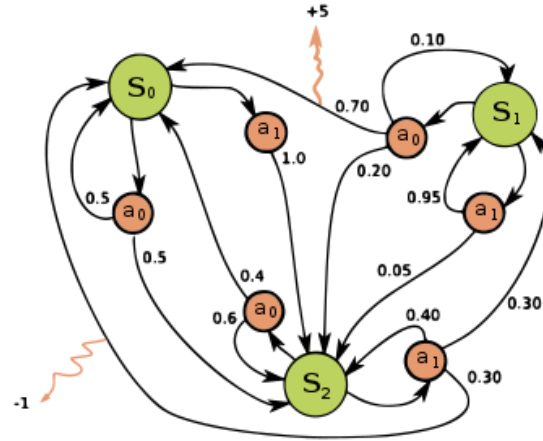


Figura 3.2: Ejemplo de un MDP simple.

En Fig 3.2 podemos observar un ejemplo de un MDP con tres estados (círculos verdes) y dos acciones (círculos rojos), con dos recompensas (flechas).

Un proceso de decisión de Markov es una 5-tupla $(S, A, P. (. , .), R. (. , .), \gamma)$ donde:

- S es un conjunto finito de estados.
- A es un conjunto finito de acciones.
- $P_a(s, s')$ es la probabilidad de que la acción a en el estado s en el momento t conduzca al estado s' en el momento $t + 1$. Donde $\sum_{s' \in S} P(s', s) = 1$.
- $R_a(s, s')$ es la recompensa inmediata recibida después de la transición del estado s al estado s' debido a la acción a .
- γ es el factor descuento, que representa la diferencia en importancia entre las recompensas futuras y las recompensas actuales.

El problema central de los MDPs es encontrar una política para el agente que toma las decisiones, es decir, una función π que especifica la acción del agente que toma las decisiones en el estado s , denotándolo como $\pi(s)$ [8].

El objetivo es elegir una política π que maximice alguna función acumulativa de recompensas denominada recompensa acumulativa. Para hacer esto el agente combina la recompensa inmediata con otras recompensas futuras. Esta recompensa acumulativa para que esté formada por recompensas mixtas, es decir futuras e inmediatas, tenemos que aplicarle un factor descuento γ ya que cuanto más futura sea una recompensa, menor valor tendrá. Entonces, a esta función V se le llama recompensa descontada:

$$V = \sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t, s_{t+1}) \quad (3.1)$$

Donde se elige $a_t = \pi(s_t)$.

3.1.3 ApR Activo

Tal y como se explica en el capítulo 1, el ApR tiene 3 tipos de agentes. En el presente TFG se toma como agente ApR el agente QL debido a que no necesita un modelo del entorno.

El agente QL tiene que aprender un modelo completo con salidas probabilísticas para todas las acciones, en lugar de tener que aprender sólo el modelo para la política fija [1].

Hay que tener en cuenta que el agente tiene que realizar la elección entre las acciones. La utilidad de un estado es el valor que tiene ese estado para que el agente lo tenga en cuenta en la toma de decisiones. En Fig. 3.3 podemos ver un entorno con las utilidades de cada estado. Las utilidades que necesita aprender son las definidas por la política óptima, que siguen la ecuación de Bellman basada en la función de utilidad:

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U(s') \quad (3.2)$$

Donde $R(s)$ es la función de recompensa, que especifica la recompensa de cada estado, $T(s, a, s')$ es el modelo de transiciones, que especifica la probabilidad de alcanzar el estado s' a partir de estado s después de realizar la acción a y γ el factor de descuento.

3	0.812	0.868	0.918	+1
2	0.762		0.660	-1
1	0.705	0.655	0.611	0.388
	1	2	3	4

Figura 3.3: Entorno con la utilidad de cada estado.

Esta ecuación se resuelva para aprender la función de utilidad U a partir de los algoritmos de iteración de valor. Cuando se obtiene tal función de utilidad U , el agente puede tomar una acción óptima a partir de un paso de mirar hacia adelante para maximizar la utilidad esperada.

Exploración

Un agente que solo sigue las indicaciones de una política óptima para el modelo aprendido en cada paso no aprende las utilidades verdaderas o la política óptima verdadera. A este agente se le denomina agente voraz, y pocas veces converge en la política óptima para este entorno mientras que otras veces converge en políticas francamente horribles.

El problema viene de que el modelo aprendido no es el mismo que el entorno real, porque el agente no sabe cuál es el entorno real, por lo que no puede calcular la acción óptima apropiada al mismo.

Este problema se soluciona cuando el agente adopta una estrategia basada en un equilibrio entre la explotación, que se basa en maximizar la recompensa, y la exploración, que tiene como objetivo maximizar su comportamiento a largo plazo[15].

Por lo tanto, un agente tendría que ser voraz en el límite de infinitas exploraciones, o GLIE (Greedy in the Limit with Infinite Exploration). El esquema GLIE es un método de exploración que se basa en que el agente intente cada acción en cada estado dado un número ilimitado de veces, para evitar tener una probabilidad finita de que una acción óptima se pierda por una mala serie de resultados atípicos. Sin embargo, también es necesario que este

esquema se convierta con el tiempo en voraz para que las acciones se conviertan en óptimas respecto al modelo aprendido.

Esquemas GLIE:

- El agente toma una acción aleatoria durante una fracción $1/t$ de tiempo y que sea voraz en el resto de los casos. Tiene el inconveniente de que el aprendizaje puede ser demasiado lento.
- Proporcionar un peso mayor a las acciones que el agente no ha intentado muy a menudo, evitando a la vez las acciones que tienen baja utilidad.

Se denota $U^*(s)$ como la estimación optimista de la utilidad del estado s siguiendo un esquema GLIE. Sea $N(a, s)$ el número de veces que la acción a se ha llevado a cabo en s , se actualiza la ecuación (3.2) para añadir la estimación optimista:

$$U^x(s) \leftarrow R(s) + \gamma \max_{a'} f(\sum_{s'} T(s, a, s') U^*(s'), N(a, s)) \quad (3.3)$$

Donde $f(u, n)$ es denominado función de exploración, donde u es la utilidad y n son las acciones que no se intentan a menudo. Esta función determina la relación entre la explotación y la exploración. Un ejemplo de esta función es la siguiente:

$$f(u, n) = \begin{cases} R^+ & \text{if } n < N_e \\ u & \text{en otro caso} \end{cases} \quad (3.4)$$

Donde R^+ es un pronóstico optimista de la mejor recompensa posible obtenible en cualquier estado y N_e es un parámetro fijo. La aplicación de esta función produce que el agente intente cada par estado-acción como mínimo N_e veces.

El uso de U^* hace que los beneficios de la exploración se propaguen recíprocamente desde los bordes de las regiones sin explorar, por lo tanto, los movimientos hacia regiones inexploradas tienen mayor valor. En Fig. 3.4 se puede observar que siguiendo esta estrategia se consigue una convergencia rápida a un rendimiento óptimo.

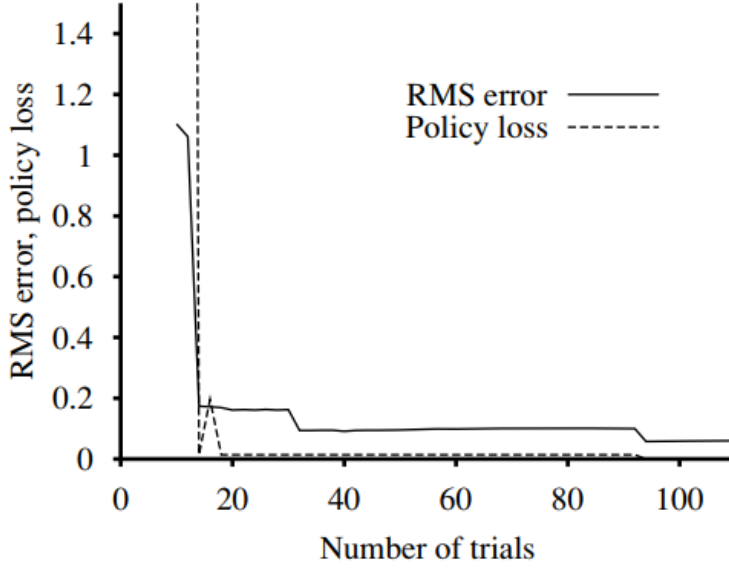


Figura 3.4: error valor cuadrático medio (RMS) en los valores de la utilidad y la pérdida de la correspondiente política.

Función Acción-Valor

El cambio más característico de un agente de aprendizaje pasivo a uno activo es que el agente no está equipado con una política fija, por lo que si aprende una función de utilidad U , tendrá que aprender un modelo que pueda elegir una acción basada en U a través de un paso de mirada hacia adelante.

Existe un método alternativo llamado aprendizaje-Q o Q-Learning que aprende una representación acción-valor en vez de aprender utilidades. Esta función-Q se denota como $Q(a, s)$ y presenta el valor de realizar una acción a en el estado s . Los valores de utilidad están relacionados de forma directa con los valores-Q:

$$U(s) = \max_a Q(a, s) \quad (3.5)$$

Por lo tanto, la función-Q es otra forma de almacenar información de utilidad y, además, el agente no necesita un modelo ni para el aprendizaje ni para la selección de acciones. Por esto se dice que el aprendizaje QL es un método libre de modelo. Habiendo introducido la función-Q, se puede actualizar la ecuación de Bellman de la siguiente forma:

$$Q(a, s) = R(s) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q(a', s') \quad (3.6)$$

Se puede usar esta ecuación de forma directa como ecuación de actualización para un proceso iterativo que calcule los valores-Q en un modelo estimado. Sin embargo, se necesita del aprendizaje del modelo, ya que en la ecuación (3.6) se usa $T(s, a, s')$. En un enfoque de

diferencia temporal no hace falta ningún modelo, por lo tanto, será necesario sustituir $T(s, a, s')$ por una velocidad de aprendizaje α actualizando la ecuación (3.6) a la introducida en el capítulo 1 (1.1). Como se menciona brevemente en el capítulo 1, α es la velocidad de aprendizaje y γ es el factor de descuento:

Velocidad de aprendizaje

Es un valor entre 0 y 1 que indica cuanto puede el agente aprender de cada experiencia. 0 significa que no aprende nada y 1 significa que olvida todo lo que sabía ahora y solo se acuerda de la nueva experiencia [16]. Para mejorar el aprendizaje se recomienda un valor dinámico donde $\alpha = 1/n$, siendo n el número de veces que se ha hecho una determinada acción en un estado dado, también definido por la función $N(s, a)$.

Factor de descuento

Es también un valor entre 0 y 1 que indica como de importante es una acción a largo plazo. 0 significa que solo importan los refuerzos inmediatos, y 1 los refuerzos a largo plazo. Por lo tanto, este factor ayuda a mezclar recompensas directas con recompensas a largo plazo dando lugar a una recompensa mixta.

3.1.4 Algoritmo QL

Uno de los primeros avances en el ApR fue el desarrollo del algoritmo QL (Watkins, 1989), definido por la ecuación (1.1).

```

1  Inicializar  $Q(s,a)$ , inicializar  $N(s,a)$ ,  $\forall s \in S$ ,  $a \in A(s)$ , y  $Q(\text{estado-terminal}, *) = 0$ 
2  Repetir (para cada episodio):
3      Inicializar  $S$ 
4      Repetir (para cada paso del episodio):
5          Elegir  $A$  de  $S$  usando una política derivada de  $Q$ 
6          tomar una acción  $A$  y observar  $R$ ,  $S'$ 
7          incrementar  $N(S,A)$ 
8           $Q(S,A) \leftarrow Q(S,A) + \alpha(N(S,A))[R + \gamma \max_{A'} Q(S',A') - Q(S,A)]$ 
9           $S \leftarrow S'$ 
10     hasta que  $S$  sea terminal

```

Listado 3.1: Pseudocódigo del algoritmo QL.

En este caso, la aprendida función- Q , aproxima directamente Q_* , la función- Q óptima, la cual es independiente de la política que es seguida. Esto simplifica dramáticamente el análisis del algoritmo y permite pruebas de convergencia temprana. La política todavía tiene

un efecto que determina cuales pares estado-acción son visitados y actualizados. Sin embargo, todo lo que se requiere para una correcta convergencia es que todos los pares continúen siendo actualizados. El pseudocódigo del algoritmo QL se puede observar el Listado 3.1 [2].

3.2 DEEP-Q LEARNING

En la actualidad, el ApR se ha llevado más allá que a la simple búsqueda de una política óptima en un entorno simple. El primer modelo de ApR con redes neuronales, llamado Deep Q-Learning en inglés, es capaz de aprender políticas de control directamente a partir de entradas sensoriales de altas dimensiones [10]. Estas entradas sensoriales de altas dimensiones son nada más y nada menos que los pixeles en bruto del entorno donde se encuentra el agente.

El modelo de redes neuronales puede ser una red neuronal convolucional, entrenado con una variante de QL cuya entrada son pixeles y la salida es una función valor estimando recompensas futuras.

Este método se ha aplicado a algunos juegos Atari 2600, como los de Fig. 3.5, superando a expertos humanos en varios de estos.



Figura 3.5: Capturas de pantalla de 5 juegos Atari 2600.

El aprendizaje de agentes de control a partir de entradas sensoriales de altas dimensiones es uno de los grandes desafíos del ApR. Las aplicaciones ApR más exitosas que operan en estos dominios se han basado en características hechas a mano con funciones de valores lineales. Por lo tanto, el rendimiento de tales sistemas depende de la calidad de la representación de las características.

Avances recientes en aprendizaje profundo han hecho posible extraer características de alto nivel a partir de datos sensoriales, conduciendo a avances en visión computacional y

reconocimiento de voz. Estos métodos utilizan un rango de arquitecturas de redes neuronales, incluyendo redes convolucionales, perceptrones multicapa, entre otros.

Sin embargo, ApR presenta distintos desafíos a partir de la perspectiva del aprendizaje profundo. Primeramente, la mayor parte de las aplicaciones exitosas en aprendizaje profundo hasta la fecha han requerido una gran cantidad de datos clasificados como conjuntos de entrenamiento. Por otra parte, los algoritmos RL deben ser capaces de aprender a partir de una recompensa escalar que es frecuentemente escasa, ruidosa y atrasada en el tiempo. Otro problema es que la mayoría de los algoritmos de aprendizaje profundo asumen que las muestras de datos son independientes, mientras que en ApR uno encuentra normalmente secuencias de estados altamente correlacionados.

El algoritmo Deep Q-Learning demuestra que una red neuronal convolucional puede llegar a sobrepasar estos desafíos aprendiendo políticas de control exitosas a partir de datos de video en entornos de ApR complejos. La red es entrenada con una variante del algoritmo Q-learning, con el Gradiente Descendente Estocástico (SGD) para actualizar sus pesos.

3.3 VIDEOJUEGOS BASADOS EN CELDAS

Es un tipo de videojuego cuya área de juego está formado por imágenes cuadradas pequeñas, aunque éstas también pueden tener forma rectangular o hexagonal.

Una característica visual esencial es que las personas que juegan no pueden distinguir la división del mapa, es decir, las celdas es básicamente una distinción técnica [17].

En el área de juego se usa un conjunto de imágenes denominado conjunto de patrones. Un ejemplo de conjunto de patrones se puede ver en Fig. 3.6.

Los juegos formados por conjuntos de celdas suelen simular una vista de arriba hacia abajo, vista lateral o 2.5D del área del juego, y son generalmente bidimensionales. Como se puede observar en el ejemplo de la figura 3.7.

Los videojuegos en el rango de la época de los años 70 a los años 90 dependían de un hardware que tenía soporte nativo para mostrar pantallas en celdas con poca participación de la CPU.

Los videojuegos basados en celdas no son un género de videojuegos específico, si no que el término se refiere a la tecnología que usa un motor de juego para su representación visual.



Figura 3.6: Ejemplo de conjunto de patrones.

Los motores basados en celdas permiten a los desarrolladores crear mundos complejos y grandes eficientemente y con recursos mínimos. Estos videojuegos suelen usar atlas de texturas, la cual es una imagen que contiene una colección de imágenes más pequeñas, por razones de eficiencia. También guardan metadatos sobre las celdas, tales como las colisiones, daño, propiedades, etc.

3.3.1 Historia

Este modelo de videojuegos fue introducido por la marca Namco en su juego Galaxian en el 1979 [6]. El tamaño del patrón más común usado en estos videojuegos era de 8 x 8 píxeles. La consola de videojuegos Intellivision, lanzada en 1979, fue diseñada para usar gráficos basados en celdas. Ya que sus juegos tenían que poder almacenarse en cartuchos de videojuegos de hasta 4K de memoria.

Los ordenadores personales tenían soporte hardware en forma de caracteres ASCII, generalmente con el propósito de mostrar texto, pero los juegos también podrían escribirse usando letras y signos de puntuación como elementos del juego. Los personajes de los juegos podrían ser cualquier gráfico del juego que encajase en una celda de 8x8 píxeles.

Este modelo de videojuegos se utilizó principalmente en géneros como plataformas y rol, llegando a su auge durante la era de las consolas de 8 y 16 bits.

desarrollo web, para identificar los problemas que existían en la web y así poder comparar soluciones alternativas a esos inconvenientes y por lo tanto certificar que extensiones de protocolo no serían dañinas para las restricciones principales que aseguran el éxito de la web.

Las APIs REST contienen las siguientes características:

- Identificación de recursos: Los recursos deben ser totalmente accesibles, es decir deben poseer un URI de identificación única
- Manipulación de recursos: Uso de recursos HTTP a partir del acceso gracias a sus métodos Get, Post, Put, Delete, Patch.
- Metadatos para describir nuevos recursos: Usa métodos o tecnologías estándar como HTTP, URI, XML, JSON, etc.
- Comunicación sin estado: no guardas las transacciones entre el cliente y el servidor. Tiene como finalidad disminuir el tiempo de respuesta de invocación al servicio web.



Figura 3.8: Esquema Cliente-Servidor en API REST.

Mediante el uso de un protocolo sin estado y operaciones estándar, los sistemas REST apuntan a un rendimiento rápido, confiabilidad y la capacidad de crecer, al reutilizar componentes que se pueden administrar y actualizar sin afectar al sistema en su conjunto, incluso mientras se está ejecutando. En Fig. 3.8 se puede ver el esquema Ciente-Servidor en API REST.

En la implementación de una API REST, las URLs permiten acceder a cada página, sección o documento del sitio web. Cada página, sección o documento se denomina como recurso [5]. Por lo tanto, el recurso es la información a la que queremos acceder independientemente de su formato.

Las URL son un tipo de Uniform Resource Identifier (URI) en inglés, que permiten identificar únicamente cada recurso.

Para manipular los recursos, HTTP concede los siguientes métodos para operar:

- GET: Consulta y lectura de recursos.
- POST: Creación de recursos.
- PUT: Editar recursos.
- DELETE: Eliminar recursos.
- PATCH: Editar partes determinadas de un recurso.

Sin embargo, durante los últimos años los desarrolladores web han usado únicamente los métodos GET y POST para realizar todas estas acciones. Esto ha podido ser debido al soporte de ciertos navegadores o al desconocimiento de los desarrolladores. Esto puede generar problemas en la API REST a la hora de crear las URLs.

Como último aspecto básico de la API REST, está Hypermedia. La definición y objetivo de Hypermedia es conectar mediante vínculos las aplicaciones clientes con las APIs, permitiendo a dichos clientes despreocuparse por conocer de antemano como acceder a los recursos [14]. Hypermedia es útil por ejemplo para que el cliente no necesite conocer las URLs de los recursos, evitando realizar el mantenimiento de cada uno si en el futuro se producen cambios en la URLs.

3.5 BASES DE DATOS RELACIONALES

Una base de datos relacional es una base de datos que es utilizada como un conjunto de tablas operándola conforme con el modelo de datos relacional. Está formada por un conjunto de objetos que se utilizan para el almacenamiento, gestión y acceso a los datos. Como ejemplos de objetos hay tablas, vistas, índices, funciones, etc.

Una base de datos relacional particionada es aquella donde los datos se manejan repartidos en múltiples particiones o nodos. Esta separación de los datos es transparente para los usuarios de gran parte de las sentencias de SQL.

En estas bases de datos cada tabla representa una relación, cada fila de esta tabla una tupla y cada columna un atributo. Estas bases de datos deben contener una clave primaria que no puede ser nula y pueden tener una clave foránea que sirve para establecer relaciones con otra tabla.

3.6 OBJECT RELATIONAL MAPPING

Es un modelo de programación que consiste en transformar las tablas de una base de datos en una serie de objetos que simplifiquen las operaciones de las bases de datos por parte del programador. SQL ha sido sin duda el lenguaje más usado para acceder a las bases de datos relacionales.

Con el uso de ORM se mejora mucho más la productividad ganando importancia a los datos de la aplicación y dejando de lado los datos de persistencia. Como ventaja tenemos un aumento de seguridad ante el ataque contra las bases de datos. Sin embargo, es negativo en aplicaciones dependientes del rendimiento ya que su curva de aprendizaje es muy elevada. Como ejemplo de ORM a continuación se menciona a SQLAlchemy.

3.6.1 SQLAlchemy

SQLAlchemy es un conjunto de herramientas de código abierto en SQL y un ORM para el lenguaje de programación Python. SQLAlchemy proporciona una interfaz general para crear y ejecutar código independiente de la base de datos sin necesidad de escribir sentencias SQL [11].

SQLAlchemy se puede usar con o sin las características de ORM. En Fig 3.9 se pueden ver algunos ejemplos de configuraciones de bases de datos con Frameworks web.

Entre las ventajas de SQLAlchemy, la más característica es que permite a los programadores usar Python en su proyecto para mapear desde el esquema de la base de datos hasta los objetos de Python de la aplicación. Otra ventaja es que no son necesarios conocimientos en SQL para manejar la base de datos.





web framework	None	Flask	Flask	Bottle
Database interaction	SQLAlchemy Core	SQLAlchemy Core	SQLAlchemy Core + ORM	SQLAlchemy Core + ORM
database connector	(built into Python stdlib)	psycopg	psycopg	psycopg
relational database	 SQLite	 PostgreSQL	 PostgreSQL	 PostgreSQL

Figura 3.9: Ejemplo de configuraciones de Frameworks web con bases de datos.

Una gran idea es usar SQLAlchemy en una aplicación web como backend de base de datos. Además, en la creación de aplicaciones web con Flask o Bottle existen librerías de Python que sirven como puente entre la librería SQLAlchemy y estos dos frameworks.

4. MÉTODO DE TRABAJO

En el presente capítulo se explicará y analizará la metodología de gestión escogida para el desarrollo del TFG. Primero se hablará sobre las metodologías ágiles y a continuación se describe la metodología eXtreme Programming (XP), que es la que se ha usado para el desarrollo de software. Por último, se explica como se ha utilizado XP en el proyecto.

4.1 METODOLOGÍAS ÁGILES

Las metodologías ágiles describen un conjunto de valores y principios para el desarrollo de software bajo los cuales los requisitos y las soluciones evolucionan a través del esfuerzo colaborativo de equipos organizados multifuncionales.

Usan una planificación adaptativa, el desarrollo evolutivo, la entrega temprana y la mejora continua fomentando una respuesta rápida y flexible al cambio.

Fue desarrollado como evolución de las clásicas metodologías de gestión. En 2001 varios desarrolladores de software se reunieron en Utah para analizar estos métodos ágiles, publicando así el Manifiesto para el desarrollo de software ágil [12]. Este documento está dividido en doce principios y cuatro valores generando así la filosofía de las metodologías ágiles.

Entre los distintos métodos o técnicas basadas en metodología de desarrollo ágil, los más populares son:

- eXtreme Programming. Es el método utilizado en el presente TFG y se detalla más adelante.
- Scrum. Es una metodología ágil que proporciona un marco de trabajo para la gestión de proyectos. En la actualidad es una de las más populares. Tiene como objetivo obtener resultados tempranos adaptándose a cambios en las necesidades de los clientes. Entre sus principios más característicos se encuentran el desarrollo software por medio de iteraciones incrementales y las reuniones a lo largo del proyecto.
- Dynamic Systems Development Method (DSDM). Este método se apoya en la continua interacción con el usuario. Esto da lugar a un sistema mucho más adaptable a los requisitos cambiantes del usuario y a las necesidades de la empresa.
- Agile Unified Process (AUP). Muestra un enfoque simple para el desarrollo de aplicaciones comerciales por medio de técnicas y conceptos ágiles. Estas técnicas

incluyen el desarrollo basado en pruebas, el modelado ágil, gestión ágil de cambios y refactorización de bases de datos para mejorar la productividad.

4.1.1 Diferencia entre metodologías ágiles y tradicionales

Generalmente las metodologías tradicionales tienen como prioridad el uso exhaustivo de documentación y tienen como meta cumplir con la planificación del proyecto dejando de lado la capacidad de respuesta a los cambios, así como la importancia de la confianza en las habilidades del equipo y las relaciones con el cliente.

En Fig 4.1 se puede observar una comparación de los ciclos de desarrollo software entre la metodología tradicional y la ágil. Como principal diferencia, se puede observar la existencia de iteraciones en la metodología ágil, ocupando una gran parte del ciclo de desarrollo.

Habiendo visto la comparativa en los ciclos de desarrollo, en el Cuadro 4.1 se muestran las diferencias respecto al contexto del equipo y organización.

4.1.2 Manifiesto ágil: Valores

El Manifiesto Ágil propuso un conjunto de valores y principios para todas las metodologías ágiles de aquel momento como contraposición a las clásicas metodologías de gestión desarrolladas hasta ese momento. Estos valores se disponen como una contraposición. Estos valores son los siguientes:

- Individuos e interacciones sobre procesos y herramientas. Se refiere al hecho de que lo más importante en el desarrollo de software son las personas y las interacciones entre ellas, por lo que es prioritario su bienestar individual y común. Este valor es clave para mejorar la productividad en equipo.
- Software de trabajo sobre documentación extensiva. Este valor define la tendencia a documentar de manera excesiva los desarrollos software de manera que se emplea más tiempo en la documentación que en el propio desarrollo de software.
- Colaboración del cliente sobre negociación contractual. Se refiere al hecho de crear contratos de requisitos menos cerrados dando lugar a una interacción constante entre el cliente y el equipo de desarrollo.

- Respondiendo ante el cambio sobre seguir un plan. Este valor define la inviabilidad de una planificación al uso debido a que las metodologías ágiles necesitan cambios constantes en los requisitos durante el desarrollo de software.

Metodología Ágil	Metodología Tradicional
Pocos Artefactos. El modelado es prescindible, modelos desechables.	Más Artefactos. El modelado es esencial, mantenimiento de modelos
Pocos Roles, más genéricos y flexibles	Más Roles, más específicos
No existe un contrato tradicional, debe ser bastante flexible	Existe un contrato prefijado
Cliente es parte del equipo de desarrollo (además in-situ)	El cliente interactúa con el equipo de desarrollo mediante reuniones
Orientada a proyectos pequeños. Corta duración (o entregas frecuentes), equipos pequeños (< 10 integrantes) y trabajando en el mismo sitio	Aplicables a proyectos de cualquier tamaño, pero suelen ser especialmente efectivas/usadas en proyectos grandes y con equipos posiblemente dispersos
La arquitectura se va definiendo y mejorando a lo largo del proyecto	Se promueve que la arquitectura se defina tempranamente en el proyecto
Énfasis en los aspectos humanos: el individuo y el trabajo en equipo	Énfasis en la definición del proceso: roles, actividades y artefactos
Basadas en heurísticas provenientes de prácticas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Se esperan cambios durante el proyecto	Se espera que no ocurran cambios de gran impacto durante el proyecto

Cuadro 4.1: Diferencia entre metodologías ágiles y tradicionales.

4.1.3 Manifiesto ágil: Principios

Los valores del apartado anterior se sustentan en 12 principios señalando así la diferencia entre un desarrollo ágil y un desarrollo clásico. Los principios son los siguientes:

- La mayor prioridad es satisfacer al cliente a partir de la entrega temprana y continua de software con valor.
- Se acepta que los requisitos cambien, en cualquier etapa del desarrollo. Los procesos ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.

- Se entrega software funcional de forma frecuente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
- Los responsables de negocio y los desarrolladores trabajan juntos durante todo el proyecto.
- Los proyectos se desarrollan en torno a un personal motivado. Hay que ofrecerles el entorno y el apoyo que necesitan, y confiar en ellos para finalizar el trabajo.
- El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es el diálogo cara a cara.
- El software funcionando es la medida principal de progreso.
- Los procesos ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios deben ser capaces de mantener un ritmo constante de forma continua.
- La atención continua a la calidad técnica y al buen diseño mejora la agilidad.
- La simplicidad es esencial.
- Las mejores arquitecturas, requisitos y diseños surgen de equipos organizados por sí mismos.
- En intervalos regulares, el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento.



Figura 4.1: Comparación de ciclo de desarrollo ágil y tradicional.

4.2 EXTREME PROGRAMMING

Extreme Programming o eXtreme Programming (XP) es una metodología de desarrollo ágil que tiene como objetivo principal aumentar la productividad a la hora de desarrollar un proyecto software [3]. Muestra una mayor importancia a los trabajos que dan un resultado directo y en los cuales se reduce la burocracia que exista en el entorno de trabajo. Hay que añadir que también añade buenas prácticas que tienen un límite mayor que la gestión de proyectos software.

XP es exitosa porque enfatiza la satisfacción del cliente. En lugar de ofrecerle todo lo que pueda desear en fecha lejana, este proceso le ofrece el software necesario a medida que lo necesita. XP les permite a sus desarrolladores responder con confianza a los requisitos cambiantes de los clientes, incluso a finales del ciclo de vida.

Investigación y estudio del entorno Tiled	
Número: 1	Prioridad: Media
Riesgo en desarrollo: Bajo	Iteración: 1
Programador responsable: Eduardo M.	Estimación temporal: 4 horas
Descripción: Se investiga el entorno de desarrollo de mapas basados en conjuntos de patrones o celdas.	
Validación: La tarea finaliza cuando se consiga crear un mapa adecuado para su futuro uso en el proyecto, y se hayan aprendido las distintas funcionalidades del editor.	

Figura 4.2: Ejemplo de historia de usuario.

XP prioriza el trabajo en equipo. Los gerentes, clientes y desarrolladores son socios iguales en un equipo colaborativo. También implementa un entorno simple y eficaz que permite a los equipos ser altamente productivos. El equipo se organiza a sí mismo en torno al problema para resolverlo eficientemente.

Los programadores que usan XP se comunican constantemente con sus clientes y otros programadores. Obtienen una retroalimentación al probar su software desde el primer día. Su objetivo es entregar lo antes posible el sistema a los clientes para los cambios que estos consideren necesarios. Con estos principios los programadores extremos pueden responder valientemente a los requisitos y la tecnología cambiantes.

4.2.1 Las reglas de XP

Planificación

- Las historias de usuario tienen que ser escritas: Las historias de usuario siguen el mismo propósito que los casos de uso, pero no son lo mismo. Se usan para crear estimaciones de tiempos para la reunión de planificación de la versión. Las historias de usuario deben ser escritas por los clientes como cosas que el software debe hacer para ellos. Son similares a los casos de uso, pero no están limitados a describir una interfaz de usuario. En Fig. 4.2 se muestra un ejemplo de historia de usuario.
- La planificación de lanzamiento crea el cronograma de lanzamiento: El plan de lanzamiento se usa para crear planes de iteración para cada iteración individual. La planificación de versiones tiene un conjunto de reglas que permite que todo el personal involucrado en el proyecto tome sus propias decisiones. Las reglas definen un método para negociar un cronograma con el que todos puedan comprometerse.
- Hacer pequeñas pero frecuentes entregas: El equipo de desarrollo necesita entregar versiones iterativas del sistema a los clientes de forma frecuente. El cliente pedirá un software nuevo cada semana o dos. Al final de cada iteración, el cliente tendrá un software funcional y listo para producción.
- El proyecto se divide en iteraciones: El desarrollo iterativo agrega agilidad al proceso de desarrollo. Un ejemplo básico es dividir el cronograma de desarrollo en una docena de iteraciones, de 1 a 3 semanas de duración. Hay que intentar mantener la longitud de la iteración constante durante todo el proyecto. Esta es la clave que hace medir el progreso y que la planificación sea simple y confiable. El esquema iterativo de XP se puede observar en Fig. 4.3.
- La planificación de la iteración comienza cada iteración: Se convoca una reunión de planificación de la iteración al comienzo de cada una para producir el plan de tareas de esa iteración. Cada iteración debe durar de 1 a 3 semanas.
- Semana de 40 horas: Una propiedad importantísima dentro de la productividad es que la semana tiene 40 horas de trabajo. Si no se acaba el trabajo planificado en ese tiempo, significa que algo va mal.

Gestión

- Ofrecer al equipo un espacio de trabajo abierto y específico: La comunicación es muy importante para un equipo de XP. Es importantísimo derribar las barreras que limitan a la comunicación entre los distintos individuos que forman este equipo.
- Establecer un ritmo sostenible: Para mantener el ritmo hay que tomar un gran interés en cada iteración. Hay que obtener el software más completo, probado, integrado y listo para producción en cada iteración. El software incompleto o defectuoso representa una cantidad desconocida de esfuerzo futuro.
- Dispersar al personal: Mover a las personas para evitar la pérdida de conocimiento y la obtención de cuellos de botella. Si solo una persona en el equipo trabaja en un área determinada y esa persona tiene un problema, la productividad en el proyecto se desestabilizará bastante.
- Tener en cuenta la velocidad del proyecto: La velocidad del proyecto es una medida que mide cuanto trabajo se está haciendo en el proyecto. Para realizar esta medición, se suman las estimaciones de las historias de los usuarios que se completaron durante la iteración.

Diseño

- Simplicidad: Construir un diseño lo más simple posible para dar respuesta a las necesidades y requisitos actuales del sistema. Un diseño simple permite ejecutar todas las pruebas donde no haya lógica duplicada y que muestre las prioridades del desarrollador. También no debe tener un gran número de clases y métodos. XP permite utilizar lenguajes de modelado como Unified Modeling Language (UML). El uso de diagramas puede ayudar a mantener una dirección única en el proyecto. Los diagramas tienen que ser pequeños, por lo tanto, los elementos analizados por cada persona son limitados. Estos diagramas también deben girar en torno a los detalles importantes. La información específica está incluida únicamente en el código, mientras que la información en alto nivel estará inmersa en los diagramas.
- Elegir una metáfora del sistema: La metáfora del sistema es en sí misma una metáfora para un diseño simple con ciertas cualidades. La cualidad más importante es poder explicar el diseño del sistema a personas nuevas sin tener que recurrir a entregarles una gran cantidad de documentación. Esta metáfora debe ser compartida entre el grupo de desarrolladores y el cliente para ayudar al diálogo. Una buena metáfora es

aquella que es fácil de comprender por el cliente y contiene la información necesaria para guiar la estructura del proyecto.

- Usar tarjetas CRC para el diseño: Las Tarjetas de Clase-Responsabilidad-Colaboración sirven para diseñar el sistema como un equipo. El mayor valor de las tarjetas CRC es permitir que las personas abandonen el modo de pensamiento basado en procedimientos y aprecien más la tecnología basada en objetos. Las tarjetas CRC permiten a equipos enteros participar en el diseño. Cuanto mayor sea la participación en el diseño del sistema, mayor será la cantidad de mejores ideas.
- Refactorización siempre que sea posible: Los desarrolladores tienen a su cargo reestructurar el sistema, sin modificar su funcionalidad. De esta forma se conseguirá una eliminación del código duplicado, se reducirá el acoplamiento aumentando la cohesión y la flexibilidad. Cuando el diseño necesita ser simplificado para facilitar su mantenibilidad se realizan estas refactorizaciones.

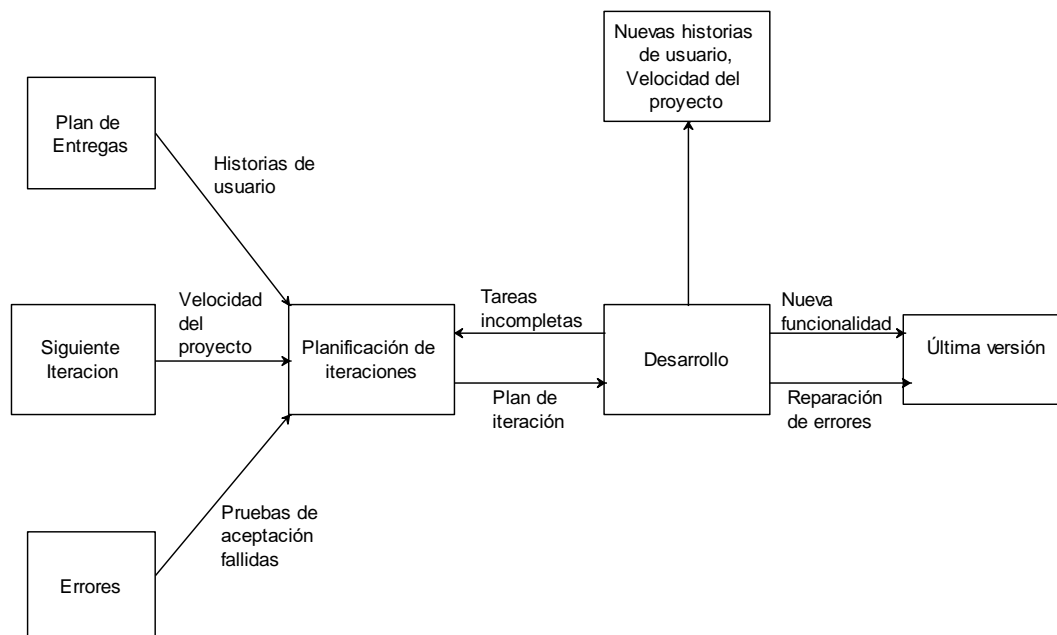


Figura 4.3: Desarrollo iterativo.

Codificación

- El cliente siempre está disponible: Esto es necesario para ayudar al equipo de desarrollo y para formar parte de este. Todas las fases de desarrollo en XP requieren comunicación con el cliente. Por lo general, se debería asignar uno o más clientes al equipo de desarrollo.
- El código se debe implementar de acuerdo con los estándares: Los estándares de código lo mantienen constante y fácil de leer y rehacer para todo el equipo.
- Implementar primero las pruebas unitarias: Cuando las pruebas unitarias se crean antes del código, resulta mucho más fácil crear el código del programa. El tiempo combinado que lleva crear una prueba unitaria y un código que apruebe esta prueba es casi lo mismo que simplemente codificar el programa directamente.
- Todo el código es programado en pares: El par de desarrolladores está formado por dos identidades. Uno es el programador del código fuente y las pruebas; mientras que el otro revisa ese código y pruebas. Por lo tanto, el segundo componente de este par será el encargado de revisar el código desde otro enfoque, incluyendo nuevas pruebas unitarias y revisando la simplicidad del diseño.
- Integración continua: Los desarrolladores deberían integrar y subir código al repositorio de código cada pocas horas. La integración continua a menudo evita esfuerzos de desarrollo divergentes, donde los desarrolladores no se comunican entre sí sobre lo que se puede reutilizar o compartir. Todos deben trabajar con la última versión de código.
- El código es una propiedad colectiva: La propiedad colectiva considera que el código pertenece al proyecto y al equipo, en lugar de a la persona que programó cada parte. Por lo tanto, cualquier participante del proyecto puede añadir nuevas funcionalidades, solucionar errores o refactorizar en cualquier parte del proyecto.

Pruebas

- El código debe tener pruebas unitarias: Una prueba unitaria es la verificación de una unidad de código determinado dentro del sistema. Las pruebas unitarias son una de las piedras angulares de XP. Las pruebas unitarias nos aseguran que un determinado módulo cumpla con un comportamiento esperado en forma aislada antes de ser integrado al sistema. En los últimos años se han creado un conjunto de frameworks llamados xUnit que permiten la automatización de las pruebas unitarias. Además,

estos frameworks permiten definir las pruebas unitarias y ejecutarlas de forma reiterada.

- El código debe pasar todas las pruebas unitarias antes de ser lanzado: En XP los programadores deben escribir pruebas unitarias para cada módulo antes de escribir el código. Después de eso, los programadores ejecutan las pruebas unitarias, las cuales tienen que tener una efectividad del 100% para que ese código pueda integrarse al sistema.
- Crear nuevas pruebas al encontrar errores: Cuando se encuentra un error, se crean nuevas pruebas para evitar que regrese. Un error en la producción requiere una prueba de aceptación para evitarlo. Crear una prueba de aceptación antes de la depuración ayuda a los clientes a definir el problema y comunicárselo a los programadores.

4.2.2 Los valores de XP

XP se basa en valores. Las reglas anteriores son una extensión o consecuencia de maximizar los presentes valores [13]. XP es una forma de trabajar en armonía con sus valores personales y corporativos. Los valores son los siguientes:

- Simplicidad: Se desarrolla únicamente lo necesario y nada más que eso. Por lo tanto, se maximizará el valor de la inversión realizada. Se toman pequeños pasos hacia el objetivo, evitando los errores que se produzcan. Una ventaja importante es que no se invierte esfuerzo en futuros requerimientos que podrían cambiar, o simplemente no se vayan a necesitar.
- Comunicación: Todos son parte del equipo y la comunicación cara a cara es imprescindible. Todo el equipo trabaja en colaboración siempre. Esta comunicación se hace por medio de transferencia de conocimientos en reuniones de manera frecuente, entre los clientes y los desarrolladores.
- Retroalimentación: Cuando el cliente está integrado en el proyecto, se tiene la ventaja de saber su opinión en cada momento. Al llevarse a cabo breves ciclos de desarrollo, se minimiza el tener que rehacer partes que no cumplen con los requisitos aumentando así la productividad de los programadores.
- Valor: Se comunicará la verdad sobre el progreso y las estimaciones. No hay excusas para el fracaso porque en el horizonte siempre se visualiza el éxito. El equipo se

adapta a los cambios cuando sucedan y el trabajo se acaba hoy y no se deja para mañana.

- Respeto: Todos los miembros de equipo se muestran y se ofrecen el mismo respeto entre ellos. Todos aportan valor, incluso si únicamente es entusiasmo. Los desarrolladores respetan la experiencia de los clientes y viceversa. Los miembros del equipo respetan su propio trabajo porque siempre se lucha por una mayor calidad en el producto y por un diseño eficiente y óptimo para la solución a través de la refactorización de código.

4.2.3 Roles en XP

En XP, el énfasis está puesto en la colaboración de todo el equipo y en la comunicación continua. A pesar de eso, se necesitan ciertos roles para que un proyecto de programación extrema funcione y las personas que asuman estos roles tengan sus responsabilidades correspondientes. Es aconsejable asignar a las personas adecuadas a cada rol en lugar de tratar de cambiar a las personas para que se ajusten a esos roles. En Fig. 4.4 se puede ver una lista de roles en XP.



Figura 4.4: Lista de roles en XP.

Iteración	Sub-objetivo	Tarea	Tiempo	Fecha	Descripción
1	1	-	-	25-09-17	Reunión iteración 1
		1	4h	-	Investigación y estudio del entorno Tiled
		2	8h	-	Investigación y aprendizaje del entorno Phaser
		3	4h	-	Creación de mapas en Tiled apropiadamente
		4	12h	-	Implementación del mundo en JavaScript
2	2	-	-	9-10-17	Reunión iteración 2
		5	8h	-	Investigación y estudio sobre el algoritmo QL
		6	6h	-	Implementación del mundo en Python
		7	30h	-	Implementación del algoritmo QL en Python
3	3	-	-	23-10-17	Reunión iteración 3
		8	4h	-	Adaptar la implementación del mundo en JavaScript
		9	10h	-	Implementar el algoritmo QL en Phaser
4	4-5	-	-	30-10-17	Reunión iteración 4
		10	6h	-	Investigación y estudio sobre la librería Flask
		11	3h	-	Investigación y estudio sobre Bootstrap
		12	15h	-	Iniciar la implementación del servidor REST
		13	2h	-	Implementar la interfaz del inicio de sesión
		14	10h	-	Implementar la interfaz del entrenamiento
5	4-5	-	-	6-11-17	Reunión iteración 5
		15	5h	-	Investigación sobre la librería Flask-SQLAlchemy
		16	10h	-	Implementar la interfaz de registro y carga de mapas
		17	8h	-	Implementar la BBDD con SQLAlchemy
		18	15h	-	Acabar la implementación del servidor REST

Figura 4.5: Resumen de tareas.

Roles esenciales

- Programador: Posiblemente es el rol más importante de XP. Es el encargado de implementar el código del proyecto y las pruebas unitarias. También formula las tareas de cada historia de usuario, así como la estimación de tiempo que se requerirá en cada una.
- Cliente: En XP el rol del cliente es tan importante como el de desarrollador, ya que es el cliente el que debe saber qué programar, mientras que el desarrollador debe saber cómo programar. Es el encargado de escribir las historias de usuario y las pruebas de aceptación o funcionales. También establece las prioridades entre las historias de usuario, decidiendo así cuales se implementan en cada iteración.
- Encargado de pruebas: Es el encargado de ayudar al cliente en formular las pruebas funcionales, ejecutándolas de forma constante y difundiendo los resultados al equipo. También tiene como responsabilidad la gestión de las herramientas de soporte para pruebas.
- Tracker: Tiene como responsabilidad realizar el seguimiento del proyecto proporcionando realimentación al equipo. Es también el encargado de validar el éxito entre las estimaciones realizadas y el tiempo real gastado.
- Entrenador: Es el responsable del proyecto en general. Debería tratarse de un experto en XP, concediendo su conocimiento a los miembros del equipo en forma de guías para aplicar las normas de XP. También tiene como responsabilidad determinar la tecnología y metodologías a usar por el programador.
- Jefe de proyecto: Es el dueño tanto del equipo como de sus respectivos problemas. Debe ser experto en labores de gestión y a la tecnología asociada al proyecto. Tiene como esencial labor la coordinación del proyecto con los clientes. También es el encargado de administrar las reuniones de planes de iteración y de las agendas de compromisos.

Roles opcionales

- Consultor: El consultor es normalmente alguien exterior del equipo y debería tener como mínimo determinados conocimientos sobre algún tema necesario para el proyecto. Estas cualidades le permiten conducir al equipo del proyecto para resolver problemas determinados.

- Doomsayer: Tiene como responsabilidad transmitir al equipo los riesgos existentes al proyecto, así como la comunicación de las malas noticias. Tiene que tener una investigación continua en cuanto a la existencia de riesgos en el proyecto, así como comunicar al equipo las posibles soluciones a estos.

4.2.4 El proceso XP

El proceso XP se basa en una serie de iteraciones. Cada iteración se considera como un ciclo de desarrollo en el que se siguen los siguientes pasos:

1. El cliente define el valor de negocio a implementar.
2. Se hace una estimación por parte del programador del esfuerzo que va a necesitar para esa implementación.
3. El cliente elige lo que quiere implementar.
4. El programador se encarga de construir ese valor de negocio.
5. Se vuelve al paso 1.

El ciclo de vida ideal de XP consiste en las siguientes fases:

Fase de exploración

En esta fase, el cliente determina de forma general las historias de usuario que son necesarias, mientras que el equipo de desarrollo empieza a entrar en contacto con las tecnologías y herramientas que serán necesarias para el desarrollo del proyecto. La fase de exploración toma de varias semanas a pocos meses, dependiendo del tamaño del proyecto y la familiaridad con estas tecnologías.

Fase de planificación

El cliente establecerá la prioridad a cada una de las historias de usuario, y a su vez, los programadores tendrán que realizar un estudio del esfuerzo necesario para cada historia de usuario. Hay un consenso entre el cliente y los programadores por la determinación del cronograma. La primera entrega no debería hacerse en más de tres meses.

Fase de iteraciones

En esta fase se realizan las diferentes iteraciones antes de la entrega del sistema. Las iteraciones no deben durar más de tres semanas. Una buena práctica es en la primera iteración

establecer una estructura del sistema que sirva de referencia para el resto del proyecto. Cuando se acabe la última iteración, el sistema debería estar listo para la fase de producción.

Fase de producción

La fase de producción consta de la realización de pruebas y validaciones antes de que el sistema sea entregado al cliente. Algunos cambios en esta fase pueden producir que se tomen decisiones para añadir nuevas características a la versión actual del sistema.

Fase de mantenimiento

Al mismo tiempo que la primera versión está en producción, el sistema tiene que estar en funcionamiento a la vez que se desarrollan nuevas iteraciones. Por lo tanto, se requiere de tareas de soporte para el cliente, produciendo así un decremento de la velocidad de desarrollo después de dejar el sistema en producción.

Fase de muerte del proyecto

Cuando el cliente no tiene más historias que incluir en el sistema, se puede decir que se ha llegado a esta fase. En esta fase el cliente puede pedir más requisitos como aquellos basados en el rendimiento y confiabilidad del sistema. Por último, se produce la documentación final del sistema y no se producirán más cambios en la arquitectura.

4.2.5 Diferencias entre XP y Scrum

Estas dos metodologías ágiles están muy alineadas, pero no son iguales, por eso se cree importante describir las diferencias entre estas. Las diferencias pueden ser bastante sutiles, pero son importantes. Estas son las diferencias más destacables:

- Los equipos de XP suelen trabajar en iteraciones que duran una o dos semanas, mientras que los equipos Scrum generalmente trabajan en iteraciones, llamadas Sprints, que duran entre dos semanas y un mes.
- Los equipos de Scrum no pueden hacer cambios en sus Sprints, es decir, cuando se completa la reunión de planificación de un Sprint, los elementos que conforman ese Sprint permanecen inmutables hasta el final del Sprint. Sin embargo, en XP los cambios en las iteraciones son mucho más concebibles.
- Los equipos de XP trabajan en un orden de prioridad estricto. Las características desarrolladas son priorizadas por el cliente y, por lo tanto, el equipo ve la necesidad de trabajar con estas características en ese orden. Por el contrario, el propietario del

producto Scrum da prioridad a la acumulación de productos y el equipo es el encargado de formular la secuencia en la que desarrollarán los elementos atrasados.

- Scrum no realiza ninguna práctica de ingeniería, mientras que XP sí. Entre las prácticas de ingeniería de XP se encuentran las pruebas, el enfoque de pruebas automatizadas, la programación en pares, la refactorización, etc.

4.3 APLICACIÓN DE LA METODOLOGÍA

Haber tomado XP como metodología de desarrollo se debe principalmente a la necesidad de una metodología para desarrollar un código complejo de implementar, así como el uso de la refactorización de código. XP también simplifica mucho los diseños, haciendo que el presente TFG sea mucho más ameno y fácil de desarrollar.

Al tratarse de una aplicación docente de verificación, XP se adapta bien a la función de este proyecto. Pero dado que este proyecto es una aplicación docente, no se han seguido a la perfección cada una de las normas de XP. En este proyecto se ha partido de una base formada a partir de los elementos de XP con los que se ha formado la metodología necesaria.

4.3.1 Asignación de los roles

Dado que el equipo de desarrollo está únicamente formado por el autor y director del presente TFG, todos los roles mostrados anteriormente se repartirán entre estos dos individuos. La asignación es la siguiente:

- El autor: Al presente redactor de estas palabras se le ha asignado el rol de programador y encargado de pruebas. Es el encargado de toda la implementación del código, así como el encargado de desarrollar todas las pruebas para verificar el software.
- El director: El director del TFG tiene asignado los roles de jefe de proyecto, entrenador y tracker. Esto es debido a que el director es experto en la tecnología asociada al proyecto, así como de la gestión de este y organizador de las reuniones de planes de iteración. Es también el encargado de validar el éxito de las estimaciones encargadas y proporciona realimentación al equipo.

4.3.2 Iteraciones del proyecto

En este apartado se describen brevemente las diferentes iteraciones que se han llevado a cabo en este proyecto, mientras que en el siguiente capítulo se explicará detalladamente

cada iteración, así como los resultados que se han obtenido. Las primeras iteraciones han durado dos semanas mientras que las últimas han durado una semana. Se ha reducido la duración al final ya que se tenía que aumentar la velocidad del proyecto para entregar el proyecto en el plazo establecido. Se realizaban las reuniones de iteración en el despacho del director del proyecto donde se especificaba qué hacer en la siguiente iteración. A continuación, se describen brevemente las distintas iteraciones:

Iteración 1

En esta iteración se realiza la primera toma de contacto con el entorno Tiled para la edición de mapas por celdas. Se estudia cómo crear un mapa para que el Framework Phaser pueda leerlo, por lo tanto el mapa creado debe ser preciso para que esta librería pueda entenderlo. Por ello, también se realiza una investigación sobre Phaser para la creación de juegos por celdas. Finalmente, cuando se han aprendido estas técnicas, se implementa un prototipo en JavaScript con un mapa que servirá como ejemplo para próximas iteraciones. En el cuadro 4.2 se pueden observar las historias de usuario de esta iteración.

Historias de usuario	Pronóstico
Investigación y estudio del entorno Tiled	4 horas
Investigación y aprendizaje del entorno Phaser	8 horas
Creación de mapas en Tiled apropiadamente	4 horas
Implementación del prototipo del mundo en JavaScript	12 horas

Cuadro 4.2: Historias de usuario de la iteración 1.

Iteración 2

En la segunda iteración se implementa el algoritmo QL con el prototipo de un mundo en el lenguaje Python. Se ha tenido que investigar la creación de un mundo en Python para que se pueda visualizar el aprendizaje del agente. También se hace un estudio sobre el algoritmo QL para adaptarlo al proyecto. Una vez hecho esto se implementa el algoritmo QL en Python. Las historias de usuario de esta iteración están en el cuadro 4.3.

Iteración 3

La tercera iteración se basa en pasar la implementación de la iteración anterior a JavaScript. Por lo tanto, lo primero que hay que hacer es adaptar el mundo al algoritmo QL. Cuando se han hecho los respectivos cambios, se añade el algoritmo QL compaginando así

el Framework Phaser con el algoritmo. Las historias de usuario respectivas a esta iteración se muestran en el cuadro 4.4.

Historias de usuario	Pronóstico
Investigación y estudio sobre el algoritmo QL	8 horas
Implementación del prototipo del mundo en Python	6 horas
Implementación del algoritmo QL con el prototipo del mundo	30 horas

Cuadro 4.3: Historias de usuario de la iteración 2.

Historias de usuario	Pronóstico
Adaptar la implementación del prototipo del mundo en JavaScript	4 horas
Implementar el algoritmo QL junto con la librería Phaser	10 horas

Cuadro 4.4: Historias de usuario de la iteración 3

Iteración 4

En esta iteración se realiza un estudio e investigación sobre el Framework Flask para implementar el servidor REST. También se hace una investigación de la librería Bootstrap para la ayuda en la creación de las interfaces. Se crea la interfaz gráfica del inicio de sesión y del entorno de entrenamiento. Por último, se inicia la implementación del servidor en Flask, pero no se acaba hasta la siguiente iteración donde se incluye la base de datos. Se pueden observar las historias de usuario respectivas en el cuadro 4.5.

Historias de usuario	Pronóstico
Investigación y estudio sobre la librería Flask	6 horas
Investigación y estudio sobre Bootstrap	3 horas
Iniciar la implementación del servidor REST	15 horas
Implementar la interfaz gráfica del inicio de sesión	2 horas
Implementar la interfaz gráfica del entorno de entrenamiento	10 horas

Cuadro 4.5: Historias de usuario de la iteración 4

Iteración 5

Se hace una investigación sobre un Framework adecuado para integrar bases de datos a Flask. Al final, se toma la decisión de usar SQLAlchemy ya que facilita mucho el manejo de bases de datos relacionales. Se crea la interfaz del registro del usuario, así como la interfaz

de carga de mapas. Para terminar el proyecto, se implementa la gestión de las bases de datos con SQLAlchemy en Flask y se termina la implementación del servidor REST con la conexión de todas las interfaces entre sí. En el cuadro 4.6 están las historias de usuario de esta última iteración.

Historias de usuario	Pronóstico
Investigación y estudio sobre la librería SQLAlchemy en Flask	5 horas
Implementar la interfaz de registro y carga de mapas	10 horas
Implementar la base de datos con Flask-SQLAlchemy	8 horas
Acabar la implementación del servidor REST	15 horas

Cuadro 4.6: Historias de usuario de la iteración 5

Todas las historias de usuario mencionadas anteriormente se muestran detalladamente en el Anexo A del presente TFG.

Por último, en Fig. 4.5 se muestra un resumen de tareas. Se muestra la línea temporal en el desarrollo del proyecto, orientado por la fecha de cada reunión para la planificación de cada iteración. También se puede observar el transcurso de la consecución de los objetivos.

5. RESULTADOS

En este capítulo se muestran los resultados obtenidos a lo largo del desarrollo del proyecto, describiendo cada una de sus iteraciones. Por otro lado, se explicará el algoritmo diseñado, los experimentos realizados y la aplicación gráfica.

5.1 FUNCIONALIDADES DEL SISTEMA

En este punto vamos a mostrar una visión general del software final relacionando los conceptos más importantes. El software final es una aplicación web donde el usuario puede visualizar el aprendizaje por refuerzo de un agente en un mundo introducido por el propio usuario. El esquema funcional del sistema se muestra en Fig. 5.1.

En los siguientes apartados se muestran detalladamente el trabajo, desarrollo y análisis de cada una de las iteraciones que se han tenido que realizar para el desarrollo de este software final.

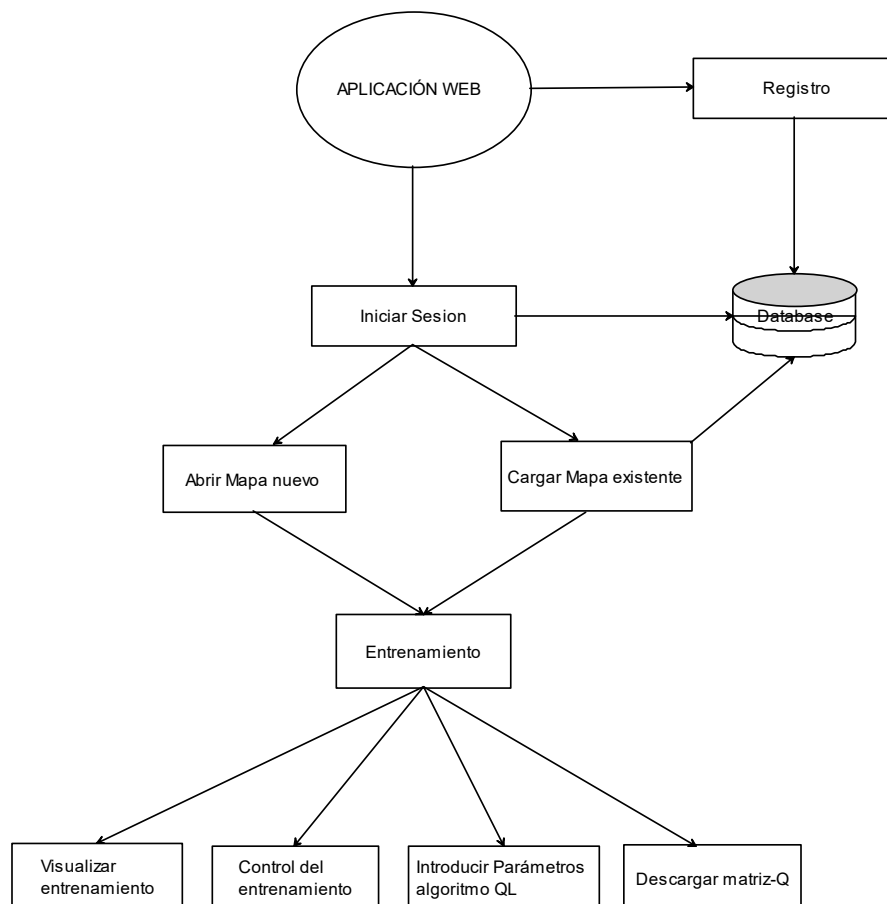


Figura 5.1: Esquema funcional del sistema.

5.2 ITERACIÓN 1: LECTURA DEL MAPA DE ENTRENAMIENTO

La primera iteración del desarrollo del sistema se basa en la primera toma de contacto con los entornos para editar el mapa y su correspondiente lectura en JavaScript. Se va a dividir este apartado en cada una de las historias de usuario que se han llevado a cabo en esta iteración.

5.2.1 Estudio y uso del entorno Tiled

En la reunión de planificación para esta iteración se discutió el Framework que se utilizaría para que el usuario de la aplicación pudiera crear sus mapas o mundos. Al final se llegó a la conclusión de que el mejor Framework sería Tiled, ya que es una de las herramientas más famosas para editar mapas basados en celdas. Las características más importantes por las que se ha elegido Tiled se enumeran a continuación:

- Es posiblemente el Framework más flexible para la edición de mapas.
- No hay restricciones en el tamaño del mapa ni de la celda.
- No hay límite en el número de capas o conjuntos de patrones.
- Permite establecer propiedades a los elementos del mapa, cosa más que imprescindible para poder reconocer en cada momento donde se sitúa el agente.
- Permite exportar el formato en formato Translation Memory Exchange (TMX), JSON o CSV. Esta variedad de formatos ayuda mucho a que luego se pueda leer ese mapa en nuestro sistema.

Una vez hecho esto, se dispuso a la descarga e instalación de Tiled. Tiled puede ser instalado tanto en Windows como en distribuciones de Linux. Hay que añadir que el presente TFG se ha desarrollado en torno a la versión 0.17.2 de Tiled ya que es una de las versiones más apropiadas que nos permite crear mapas leíbles en el futuro. Con el objetivo de probar un poco la herramienta, se hicieron algunas pruebas sobre su funcionamiento. A continuación, se describirá como se ha creado un mapa básico en Tiled a partir de un conjunto de patrones:

- Primero se ha seleccionado la opción de menú *Archivo > Nuevo* donde se ha especificado el tamaño del mapa en celdas y el tamaño de cada celda. Finalmente se ha pulsado el botón aceptar.
- Después se ha añadido el conjunto de patrones para formar el mapa a partir de los patrones que se ha querido que formen el mapa. Se ha seleccionado la opción *Nuevo*

conjunto de patrones, que se encuentra en el conjunto de opciones de abajo a la derecha, y se le ha dado al botón explorar para añadir la imagen correspondiente. El tamaño de cada celda se ha modificado al gusto para que luego se añada al mapa. Hay que señalar que el conjunto de patrones ha tenido que ser una imagen en formato PNG o JPG, cuyos patrones han tenido que estar divididos en celdas iguales para que Tiled haya podido reconocerlo.

- Posteriormente se muestra toda la interfaz con todas las opciones disponibles para la edición del mapa. En Fig.5.2 se puede observar esta interfaz gráfica. En la parte de arriba está la barra de herramientas con las opciones para gestionar imágenes y objetos. En el recuadro de arriba a la izquierda se muestran las diferentes capas de nuestro mapa, mientras que en el recuadro de abajo se puede observar el conjunto de patrones añadido previamente. En la columna de la izquierda se pueden observar cada uno de los atributos del mapa, como los de cada cerda u objeto. Y finalmente, en el centro está el mapa que se está editando. Para añadir más capas, ya sean de patrones u objetos, se ha pulsado la opción Añadir Capa, que se encuentra debajo del listado de capas.
- Una vez conocidas estas opciones, se han arrastrado patrones o celdas al mapa, formando así el mapa deseado. Después de crear la capa o las capas de patrones, se ha añadido una capa de objetos para especificar las partes de colisiones y para especificar el tipo de objeto que se trata. Esto es una característica de esencial importancia para la futura lectura del mapa, como se detallará próximamente.
- Por último, en Fig. 5.3 se puede ver el ejemplo de un mapa creado. Este es un mapa adecuado para su futura lectura por Phaser, ya que entre otras cosas se le ha añadido objetos. En un punto futuro se detalla la forma de crear un mapa exactamente para que Phaser lo lea. Cuando tenemos este mapa, finalmente hay que exportarlo en formato JSON.

5.2.2 Estudio del entorno Phaser

Phaser es el entorno o librería que se ha utilizado en JavaScript para leer el mapa que se ha editado. Phaser es probablemente el motor de juegos de HTML5 más popular. Las razones por las que se ha tomado la decisión de elegir a Phaser para la lectura de mapas son las siguientes:

- Al ser el motor de juegos más popular, tiene una excelente documentación, así como un número enorme de ejemplos para poder aprender del entorno.
- Tiene un excelente rendimiento ya que usa Pixi.js como motor de renderizado. Pixi.js usa WebGL o Canvas dependiendo del dispositivo utilizado.
- Carga los recursos por adelantado, reduciendo mucho la complejidad del código.
- Tiene tres motores físicos integrados y gratis para usar: Arcade Physics, AABB y Ninja Physics.
- Permite crear Sprites para añadirlos al mapa.
- Puede crear grupos de Sprites para facilitar la edición grupal permitiendo así una reducción de tiempo de desarrollo y complejidad de código.
- Tiene una cámara genial para permitir el enfoque que se desee.
- Permite la lectura de mapas Tiled. Esta es una de las características más importantes por las que se ha elegido Phaser. Tiene un montón de funciones para manejar y gestionar mapas formados por celdas, pero estos deben tener el formato apropiado que se mostrará a continuación.
- Es un entorno muy probado y utilizado, por lo tanto, asegura que es un sistema robusto con la documentación y solución de dudas suficiente.

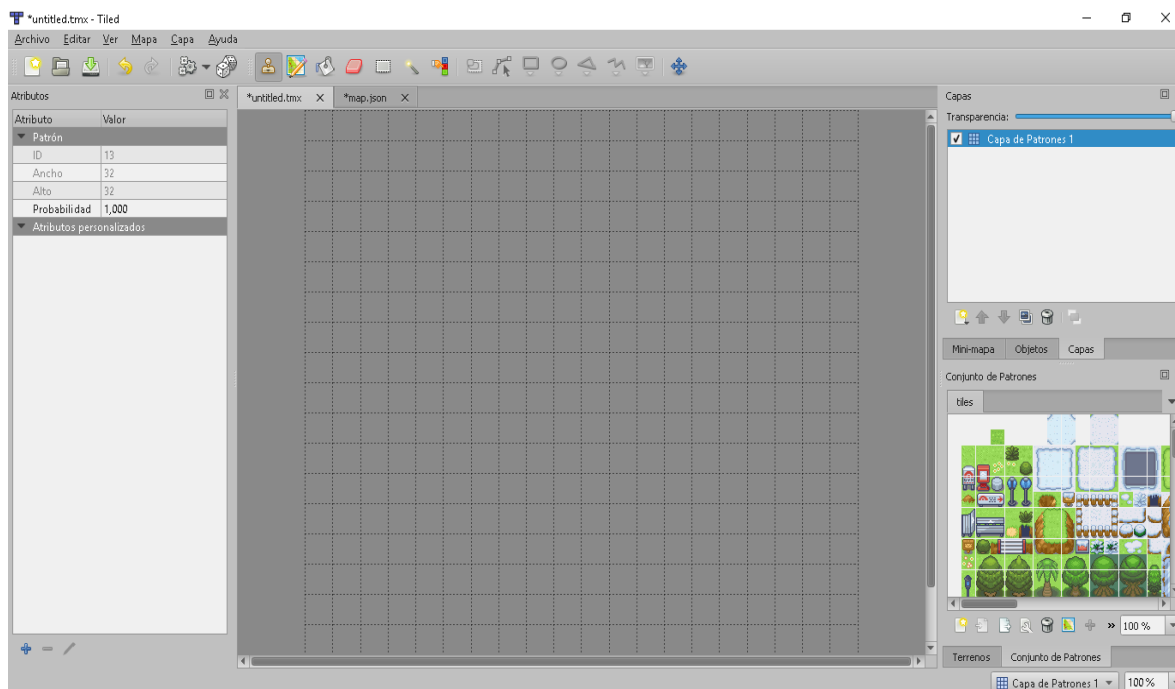


Figura 5.2: Interfaz gráfica de Tiled.

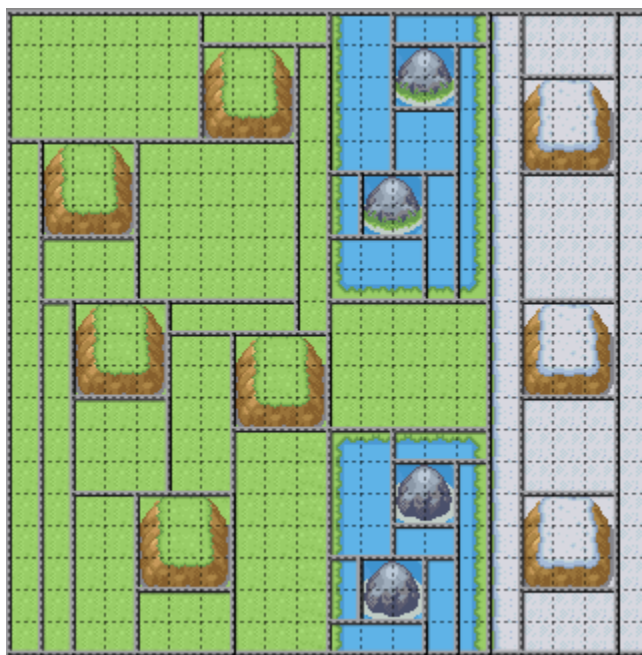


Figura 5.3: Mapa Tiled.

5.2.3 Edición adecuada del mapa en Tiled

En este apartado se explica como se ha editado el mapa en Tiled para que Phaser pueda leerlos. Hay que remarcar que los pasos seguidos han tenido que ser precisos, ya que de otra forma Phaser lanzaba errores de lectura.

Como se menciona anteriormente, la versión usada de Tiled ha sido la 0.17.2. Esta es una de las versiones que es capaz de exportar archivos JSON adecuados para Phaser. Se ha empezado usando la ultima versión, que es la 1.0.3, pero se ha visto que los archivos JSON exportados no eran soportados por Phaser. Por lo tanto, se ha tenido que reducir drásticamente la versión para lograr la aceptación de Phaser.

Aparte de la versión, se han realizado unas ligeras modificaciones o añadidos a la lista de pasos para haber creado el mapa del apartado 5.2.1. Los pasos añadidos son los siguientes:

- Al darle a *Archivo > Nuevo* y salir la ventana emergente de *Crear un nuevo mapa*, hay que dejar seleccionado el formato CSV para el conjunto de patrones.
- Al abrir un nuevo conjunto de patrones, en la versión de Tiled de Ubuntu, cuando sale la ventana emergente para elegir el archivo, se ha tenido que marcar la opción del CheckBox de *Adjuntar al mapa*. Esta opción es importante ya que en el JSON creado se muestra el conjunto de patrones adjunto con el que se ha creado ese mapa,

y Phaser tiene que saber donde se encuentra el archivo del conjunto de patrones que forma el mapa.

- Tiled permite añadir más de un conjunto de patrones para crear un mapa, sin embargo, el sistema que se ha creado solo permite la lectura de un conjunto de patrones adjunto al mapa. Si se quiere crear un mapa con más de un conjunto de patrones, el usuario es el encargado de crear un conjunto de patrones que contenga todos los patrones necesarios; y Tiled es una herramienta que permite fácilmente esa edición.
- Los mapas creados han tenido que tener una única capa de objetos y una o varias capas de patrones. Phaser es capaz de leer más de una capa de objetos, pero para el presente TFG solo es necesario una capa de objetos para especificar el tipo de entorno en cada celda del mapa. Un ejemplo de estructura de capas se puede observar en Fig. 5.4 donde se puede observar una capa de objetos y otra capa de patrones.
- Para que el agente de nuestro sistema pueda saber donde se encuentra en cada momento, se ha tenido que crear una capa de objetos, donde cada celda del mapa está sobre un objeto. Por lo tanto, para añadir objetos se ha tenido que seleccionar la capa objetos, pulsar la opción *Insertar rectángulo* y arrastrar con el cursor para cubrir la parte del mapa con el rectángulo. En Fig. 5.5 se puede observar la opción marcada y como el cuadrado cubre una montaña. Todos los mapas han tenido que estar cubiertos por objetos, como se puede observar en Fig. 5.3, donde el mapa está cubierto por rectángulos.
- A continuación, se ha tenido que añadir una propiedad *Type* a cada objeto creado. Esta propiedad ha sido asignada de tipo *String* y hay que ponerle el tipo de elemento que es, para que luego nuestra aplicación de JavaScript sepa el tipo de elemento de cada patrón. Para terminar, se ha tenido que exportar el mapa en formato JSON y dejado el archivo de conjunto de patrones en el directorio asignado en el archivo JSON. Este último paso no es estrictamente necesario ya que Phaser leerá correctamente el archivo JSON, pero para tener todo ordenado se ha decidido dejar los conjuntos de patrones en la carpeta correspondiente.

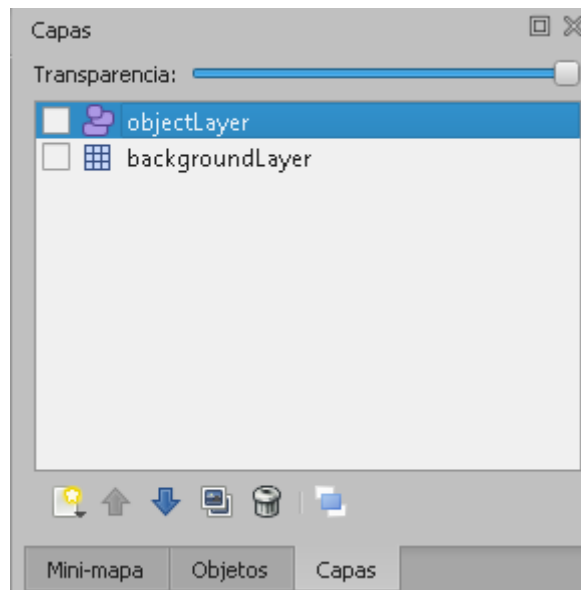


Figura 5.4: Capas del mapa.

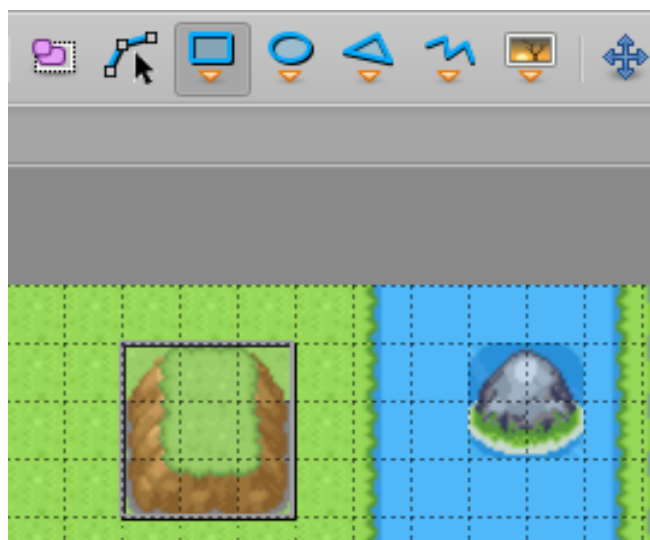


Figura 5.5: Creación de objetos en el mapa.

Para terminar este apartado, se ha tenido que analizar la estructura del archivo JSON creado. Este mapa JSON contiene toda la información de cada capa de patrones y de objetos, así como su conjunto de patrones asociado. No es necesario explicar detalladamente cada uno de los atributos del archivo, ya que Phaser hace todo el trabajo de lectura. Por lo tanto, se van a explicar los atributos del archivo JSON que son necesarios de analizar:

- Height: Se trata del número de celdas a lo alto que tiene el mapa.
- Width: Es el número de celdas que tiene el mapa a lo ancho.
- Tileheight: Son los pixeles que forman la celda a lo alto.

- **Tilewidth:** Son los pixeles que forman la celda a lo ancho
- **Tilesets:** Se trata de una lista con los archivos de conjuntos de patrones que forman el mapa. Como en esta aplicación solo se permite uno, la lista tendrá un único elemento. Este conjunto de patrones muestra sus características, como las columnas que tiene, su directorio, el nombre del archivo, los pixeles a lo ancho y lo alto, el tamaño de la celda, etc.
- **Layers:** Es una lista con todas las capas que forman el mapa. En el ejemplo creado hay una capa de patrones y una capa de objetos. La capa de patrones tiene el atributo *Data*, que es una lista con el identificador de cada uno de los patrones que forman la capa, así como el tamaño del mapa y el nombre de la capa. Mientras que la capa de objetos tiene como atributos su nombre y una lista de objetos que lo forman. Cada objeto de la lista muestra su tamaño en pixeles, su posición en el mapa y sus propiedades. En estas propiedades es donde se puede observar el tipo de elemento que es ese objeto.

5.2.4 Implementación de la lectura del mapa con Phaser

Para leer los mapas creados con Tiled se ha tenido que crear un prototipo en JavaScript para probar el funcionamiento de Phaser con Tiled, así como para ver como compaginan entre sí.

Como ejemplo de mapa se ha utilizado el creado por los puntos anteriores. La estructura de carpetas empleada ha sido por un lado el archivo HTML y por otro los directorios *Assets* y *Js*. En *Assets* se encuentran las imágenes empleadas y el mapa Tiled, mientras que en *Js* se encuentran los archivos y librerías de JavaScript. Mantener estructurado el directorio es un aspecto muy importante en el desarrollo de videojuegos web.

En cuanto al archivo HTML, se han importado cada una de las clases del juego Phaser. En la etiqueta *Body* se ha llamado a la clase *Main*, mientras que en la etiqueta *Head* se han importado la librería Phaser y las clases *Preload* y *Game*. Los estados en Phaser son partes separadas de la lógica del juego, por lo que las clases *Preload* y *Game* son los dos estados del juego. En *Preload* se cargan los gráficos utilizados mientras que en *Game* comienza el juego. La transición de un estado a otro es inapreciable cuando se ha implementado el juego ya que no hay ningún menú de juego ni nada por el estilo, únicamente se cargan las imágenes y el mapa en *Preload* y en *Game* se lanza el Juego.

Para analizar mejor la relación y contenido de estos estados del juego, se puede observar la Fig. 5.6, donde se muestra el Diagrama de clases asociado.

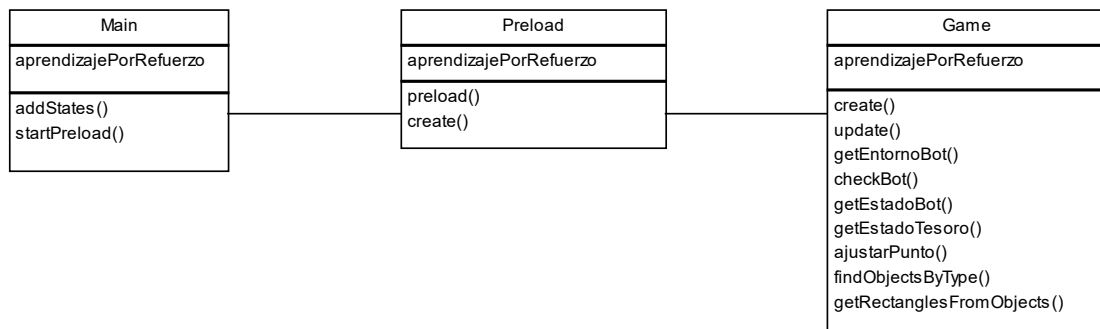


Figura 5.6: Diagrama de clases.

Como se puede observar, las tres clases tienen un único atributo, que es el objeto del juego. Este objeto es utilizado en cada estado ya que contiene los diferentes estados del juego. En el Listado 5.1 muestra la forma de inicializar este objeto en todas las clases.

```
1 var AprendizajePorRefuerzo = AprendizajePorRefuerzo || {};
```

Listado 5.1: Inicialización del objeto que contiene el juego.

En la clase *Main* tiene dos métodos. Los métodos se describen a continuación:

- **AddStates():** En este método se inicializa el juego con la librería Phaser. Se crea un objeto *Game* cuyos parámetros son la altura y anchura del juego en píxeles y el tipo de renderizado. Cuando se ha inicializado el juego en Phaser se han añadido los estados *Preload* y *Game* gracias a la función *state.add()*, cuyos parámetros son el nombre en clave del estado y la función donde está. En el Listado 5.2 se puede ver el código del método.
- **StartPreload():** Con este método únicamente se pasa al estado inicial *Preload*. Para ello se utiliza la función *state.start()*, cuyo parámetro es el nombre en clave del estado a donde se quiere realizar la transición. En el Listado 5.3 se puede ver la línea de código de este método.

```
3 AprendizajePorRefuerzo.game = new Phaser.Game(320, 320, Phaser.AUTO, '');
5 AprendizajePorRefuerzo.game.state.add('Preload', AprendizajePorRefuerzo.Preload);
6 AprendizajePorRefuerzo.game.state.add('Game', AprendizajePorRefuerzo.Game);
```

Listado 5.2: Código del método *addStates()*.

```
8 AprendizajePorRefuerzo.game.state.start('Preload');
```

Listado 5.3: Código del método *startPreload()*.

La clase o estado *Preload*, como se ha mencionado antes, se encarga de la carga de los gráficos del juego y la transición al estado *Game* para que empiece el juego. En este estado se crea su función y se le añade la propiedad *prototype* para que esa función pueda ser llamada como un objeto. En esa función hay dos métodos cuyo código se puede ver en el Listado 5.4. Estos métodos son los siguientes:

- **Preload():** En este método cargamos el mapa Tiled con la función *load.tilemap()*, cuyos parámetros son el nombre en clave del mapa, la URL del mapa y el formato de este. También se cargan las imágenes con la función *load.image()*, donde sus parámetros son el nombre en clave de la imagen y la URL de la imagen.
- **Create():** Este método ha desempeñado la función de indicar el motor físico del juego, indicando el tipo Arcade. Y finalmente se ha pasado al estado *Game*.

El estado *Game* es el estado principal del juego, ya que contiene prácticamente toda la lógica de este. Como en el estado *Preload*, se ha creado su función y se ha añadido la propiedad *prototype*. Dentro de la función hay nueve métodos, de los cuales se describen los aspectos más importantes.

```
14 AprendizajePorRefuerzo.Preload.prototype = {
15     preload: function() {
16         this.load.tilemap('map', urlMapa, null, Phaser.Tilemap.TILED_JSON); 17
18         this.load.image('gameTiles', urlTileset);
19         this.load.image('tesoro', urlTesoro);
20         this.load.image('bot', urlBot);
21         this.load.image('circuloAmarillo', urlCirculo);
22     },
23     create: function() {
24         this.game.physics.startSystem(Phaser.Physics.ARCADE);
25         this.game.state.start('Game');
26     }
27 };
```

Listado 5.4: función *prototype* del estado *Preload*.

Create()

En esta función se han inicializado la mayoría de los atributos del juego y se han añadido los elementos respectivos al juego. En el Listado 5.5 se puede ver la forma de añadir el mapa

Tiled y el conjunto de patrones al juego. También se ha tenido que especificar la capa de patrones del mapa Tiled, por lo tanto se ha añadido con la función *map.createLayer()*, donde el argumento es el nombre de la capa que se le ha asignado en Tiled.

En el Listado 5.6 se muestra la forma de añadir la imagen o Sprite del agente al juego. Por lo tanto, esto sirve para añadir el Sprite del agente y del tesoro. Para ello se usa la función *add.tileSprite()*, donde los parámetros son la posición X e Y en el mapa en píxeles, donde se ha usado la función *ajustarPunto()* que se explica más adelante, el tamaño del Sprite en píxeles y el nombre en clave de la imagen que usará como Sprite. Este es el nombre en clave cargado en el estado *Preload*. Una vez creados los Sprites hay que ajustarlos en el mapa, por lo que se ha escalado la imagen en un 80% con la función *scale.setTo()* y se ha ajustado el centro del eje de coordenadas del propio Sprite en el centro con la función *anchor.setTo()*, donde se le ha pasado el parámetro 0.5.

```
7  this.map = this.game.add.tilemap('map');
8  this.map.addTilesetImage('tiles2', 'gameTiles');
9  this.backgroundlayer = this.map.createLayer('backgroundLayer');
10 this.backgroundlayer.resizeWorld();
```

Listado 5.5: Código para añadir el mapa Tiled y el conjunto de patrones

También, se ha tenido que habilitar el motor físico a los Sprites creados con la función *physics.enable()*, cuyos parámetros son los Sprites a los que se quiere añadir el motor y el tipo de motor. También hay que especificar que los Sprites no pueden salir del límite del juego, eso se hace asignando *True* a la propiedad del Sprite *body.collideWorldBounds*.

Por último, hay que crear una lista para cada tipo de elemento del mapa. Para ello se han utilizado las funciones *findObjectsByType()* y *getRectanglesFromObjects()*, las cuales se explican más adelante. Los elementos del entorno de este mapa son las distintas propiedades añadidas anteriormente en Tiled. En este ejemplo los elementos son césped, agua, hielo, islas, montaña helada y montaña normal.

```
11 this.inicioBotX = this.ajustarPunto("x");
12 this.inicioBotY = this.ajustarPunto("y");
13 this.bot = this.game.add.tileSprite(this.inicioBotX, this.inicioBotY, 16, 16, 'bot');
14 this.bot.anchor.setTo(0.5);
15 this.bot.scale.setTo(0.8);
```

Listado 5.6: Código para añadir el Sprite del agente.

Update()

En este método se crea la lógica del propio juego ejecutandose constantemente. Por ejemplo, comprueba si se ha pulsado una tecla o si se ha chocado contra un objeto. En este ejemplo se creó un cursor para manejar el agente con el teclado y una detección constante por si el agente había encontrado el tesoro con la función *checkBot()*, que se verá más adelante. No se necesario mostrar el código completo de esta función ya que no tiene relación con la aplicación futura, solo se hizo para probar el motor físico y la colisión con el tesoro.

GetEntornoBot()

Este método tiene como finalidad comunicar el elemento del entorno donde está el agente. Para ello se utiliza la lista de objetos creados en el método *create()* y la función de phaser *Rectangle.intersects()* para ver si el sprite del agente se encuentra sobre tal objeto. En el Listado 5.7 se puede ver el código para ver la intersección entre los objetos de césped y el agente.

```
65 for (var i = 0; i < this.cespedRectangles.length; i++) {  
66     if (Phaser.Rectangle.intersects(this.bot.getBounds(), this.cespedRectangles[i])) {  
67         console.log("cesped");  
68     }  
69 }
```

Listado 5.7: Código para saber si el agente se encuentra en el césped.

CheckBot()

Esta función lo único que hace es saber si el estado del agente se encuentra en el estado del tesoro. Si fuera así notificaría de que el agente ha encontrado el tesoro. Para que esto funcione se necesita el estado actual del tesoro y del agente, para ello se utilizan las funciones *getEstadoBot()* y *getEstadoTesoro()*, que se explican ahora.

GetEstadoBot y GetEstadoTesoro()

Estas funciones devuelven la celda del mapa donde se cuenta el agente y el estado, respectivamente. Para ello se usa la función de Phaser *getTileWorldXY()*, cuyos argumentos son las posiciones en pixeles X e Y del Sprite y su tamaño. En el Listado 5.8 se puede ver el código de la función *getEstadoBot()*.


```

101  getEstadoBot: function() {
102      return this.map.getTileWorldXY(this.bot.x, this.bot.y, this.map.tileWidth,
103      this.map.tileHeight, this.backgroundlayer);
103  },

```

Listado 5.8: Código de la función *getEstadoBot()*.

AjustarPunto(eje)

Esta función se ha implementado para que devuelva una posición aleatoria del mapa. Pero esta posición no es una cualquiera, sino que tiene que ser el centro exacto de una celda del mapa, ya que el agente debe empezar centrado a entrenarse. Por lo tanto, devuelve aleatoriamente uno de los centros de las celdas del mapa.

El argumento eje puede ser *X* o *Y* por lo que el punto devuelto es en el eje correspondiente. Hay que añadir que el eje de coordenadas del mapa empieza arriba a la izquierda, por lo tanto, las *X* son crecientes hacia la derecha y las *Y* son crecientes hacia abajo.

Esta función simplemente calcula un número aleatorio entre el rango del mapa con la función de Phaser *world.randomX* o *world.randomY*. Con el cálculo del Listado 5.9 se puede observar el cálculo para ajustar el punto al centro de la celda. El cálculo observado está enfocado al eje de las *X*. Los pasos del cálculo se explican a continuación:

1. Si el número aleatorio es el último píxel a lo ancho, únicamente se le resta la mitad del ancho de la celda.
2. Si el número aleatorio es un número distinto al último píxel a lo ancho, el punto hay que ajustarlo al borde izquierdo de la celda, por lo que al número aleatorio hay que restarle la diferencia de su posición al borde izquierdo.
3. Para ajustar al punto del borde izquierdo al centro de la celda hay que sumarle la mitad de la celda.

```

48  var rx = this.game.world.randomX;
49  if (rx !== this.map.widthInPixels) {
50      var x = (rx - (rx % this.map.tileWidth) + (this.map.tileWidth / 2));
51  }
52  else {
53      var x = this.map.widthInPixels - (this.map.tileWidth / 2);
54  }
55  return x;

```

Listado 5.9: Código para ajustar punto en el eje *X*.

FindObjectByType(type)

Esta función devuelve la lista de los objetos de un tipo. El argumento *type* tiene que ser exactamente igual al nombre de la propiedad *type* introducida en el mapa Tiled. Como se puede observar en el Listado 5.10, se implementa un bucle con todos los objetos de la capa *objectLayer* para añadirlos a la lista.

```
130 findObjectsByType: function(type, map) {  
131     var result = [];  
132     map.objects['objectLayer'].forEach(function(element) {  
133         if (element.properties.type === type) {  
134             result.push(element);  
135         }  
136     });  
137     return result;  
138 },
```

Listado 5.10: Código de la función *findObjectsByType()*

GetRectanglesFromObjects(objects)

Este método tiene como objetivo devolver una lista de objetos *Phaser.Rectangle*. Para ello, se le ha pasado la lista de objetos creados en el método *findObjectByType()*. Este objeto *Phaser.Rectangle* tiene como parámetros la posición exacta del objeto y su tamaño en píxeles, para así formar el rectángulo en el mapa.

Una vez explicadas todas las clases junto con sus métodos, se muestra el resultado de la implementación en Fig. 5.7. Se puede observar el agente representado como el Sprite del robot y el tesoro como el Sprite de la bolita roja.

5.3 ITERACIÓN 2: IMPLEMENTAR EN PYTHON EL ALGORITMO QL

En esta iteración se ha desarrollado un prototipo de aprendizaje QL de un agente en un mapa similar al creado en Tiled. Hay que remarcar que esta iteración ha sido la más larga, ya que se ha tardado bastante en conseguir que el agente aprenda de la forma correcta, por ello se han llevado a cabo varias correcciones en el algoritmo QL.

Se ha implementado primero el algoritmo QL en un prototipo en Python porque el desarrollo y las pruebas serían más sencillas que implementarlo directamente en el sistema final.



Figura 5.7: Mapa Tiled.

5.3.1 Implementación del mundo en Python

Se ha tenido que crear un mapa similar a los creados en Tiled, por eso se ha usado la librería de Python TkInter. Este entorno permite crear un mapa basado en celdas, por lo que a continuación se explican los aspectos más importantes de la implementación. Esta implementación ha seguido unos pasos muy parecida a la implementación en Phaser, por lo que no se va a detallar demasiado. El mundo se ha creado en una clase llamada *mundo.py* que se ha importado desde la clase donde se ha implementado el algoritmoQL.

Lo primero de todo ha sido crear un objeto TkInter con la función *Tk()*. Este objeto ha servido como argumento para crear el objeto *mundo*. En el Listado 5.11 se puede ver la inicialización de ese objeto. Este objeto es de tipo *Canvas* y tiene como parámetros el objeto Tkinter anteriormente mencionado y el tamaño del mapa en píxeles.

```
8 mundo = Canvas(master, width = x * width, height = y * width)
```

Listado 5.11: Código para la creación del objeto mundo.

Se ha tenido que crear un punto para el agente y uno para el tesoro, por lo que aleatoriamente se toma ese punto en los rangos del mapa. También se han tenido que crear unas variables con la recompensa de cada elemento del mundo y la recompensa del tesoro.

Además, ha sido necesario crear una lista de estados, donde cada estado es una celda del mapa. Después de eso se ha creado una lista de estados para cada elemento del mundo.

En el Listado 5.12 se puede ver como se han añadido los elementos a cada lista dependiendo de la clave que tengan.

RenderizarMundo()

Una vez creadas estas listas se ha implementado esta función para renderizar continuamente el mapa. Con las listas anteriores, creamos cada celda del mundo a partir de la función de TkInter *create_rectangle()*. Los argumentos de esta función son las coordenadas de la esquina izquierda superior y la esquina derecha inferior. También

```
68  for estado in estados:
69      if keyEstados1[c] == "c":
70          estadosCesped.append(estado)
71      elif keyEstados1[c] == "a":
72          estadosAgua.append(estado)
73      elif keyEstados1[c] == "m":
74          estadosMontana.append(estado)
75      elif keyEstados1[c] == "h":
76          estadosHielo.append(estado)
77      elif keyEstados1[c] == "i":
78          estadosIsla.append(estado)
79      elif keyEstados1[c] == "mh":
80          estadosMontanaHielo.append(estado)
```

Listado 5.12: Código para inicializar las listas de los tipos de elementos.

tendría como argumentos el color de relleno de la celda, entre otros. Tendría la forma *create_rectangle(x1, y1, x2, y2, **kwargs)*. Entonces, recorreremos con bucles *for* todas las listas creando Sprites de cada tipo. También se crea el Sprite del agente en esta función, para ello se utilizará la coordenada del agente.

Moverse(dx,dy)

Esta función básicamente mueve un sprite a una nueva coordenada. Sirve para mover el Sprite del agente dependiendo de la acción que realice, y devuelve la recompensa dependiendo del nuevo estado donde acabe el agente a partir de ese movimiento. Por lo que dependiendo de si ese estado es un tipo de entorno u otro, se devuelve la recompensa asignada para cada tipo de entorno o la recompensa del tesoro, si lo encuentra.

Inicialmente se establece la nueva coordenada del agente sumando los argumentos dx y dy a la anterior coordenada del agente. Estos argumentos dependerán del movimiento de la acción que se quiera tomar en el algoritmo QL, que ya se verán cuales son.

A continuación, se utiliza la función de TkInter *coords()* para mover el Sprite del agente, donde sus argumentos son, el objeto del Sprite del agente, y las coordenadas de la esquina izquierda superior y la esquina derecha inferior. Mejor visto, la función sería *coords(spriteBot, x1, y1, x2, y2)*. Hay que añadir que el origen del eje de coordenadas es la esquina superior izquierda del mapa, como en Phaser.

Una vez hecho esto, a partir de condiciones *if-else* se devuelve la recompensa correspondiente de la nueva coordenada del agente.

ReiniciarJuego()

En cada episodio del algoritmo QL se tiene que reiniciar el entrenamiento. Por lo que esta función se encarga de poner el agente en el estado inicial y poner el sumatorio de resultados al valor inicial.

HaReiniciado()

Este método simplemente devuelve una variable de tipo *Boolean* para saber si el agente ha encontrado el tesoro o no.

Con esta clase se ha conseguido que el mapa Tiled de Fig. 5.8 sea creado con la librería TkInter. En Fig. 5.8 se puede ver el mapa, donde el Sprite amarillo pequeño es el agente y el Sprite amarillo grande es el tesoro.

5.3.2 Implementación del algoritmo QL en Python

Esta ha sido la parte del desarrollo del proyecto que más esfuerzo ha conllevado, ya que, como se explica anteriormente, se ha tenido que corregir el algoritmo implementado hasta que el agente tuviera un aprendizaje adecuado.

Lo primero que se ha tenido que hacer para realizar la implementación del algoritmo QL, ha sido pensar como llevar la teoría del algoritmo a la práctica. A continuación se explica paso por paso como se ha interpretado la teoría.

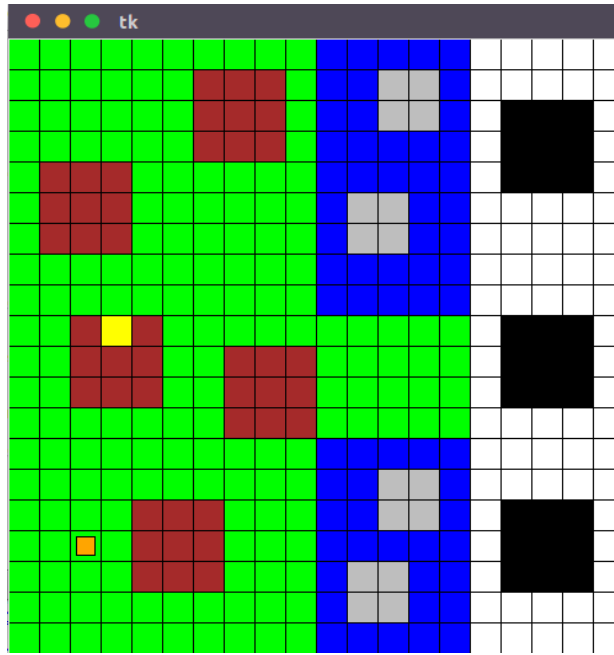


Figura 5.8: Mapa creado con la librería TkInter.

Estados

Los estados posibles donde puede ir el agente son cada una de las celdas que forman el mapa.

Acciones

Las acciones posibles que puede realizar el agente son cuatro: arriba, abajo, izquierda y derecha.

Función acción-valor

La función *acción-valor* que presenta el valor de realizar una acción a en el estado s es representada por una matriz-Q. Esta matriz se basa en una tabla, cuyas filas son los estados del mapa y las columnas son las acciones que puede realizar el agente. En Fig. 5.9 se puede observar un ejemplo de matriz-Q de este algoritmo.

Velocidad de aprendizaje

La velocidad de aprendizaje α viene definida por el valor dinámico $\alpha = 60/(59 + N(s, a))$, donde $N(s, a)$ es el número de veces que se ha hecho una determinada acción en un estado. En el programa es una matriz donde las filas son los estados y las acciones son las columnas, por lo que cada elemento de la matriz tendrá el número de veces que se realizó la acción en ese estado. Esto también ayuda a la exploración, ya que al principio el agente

prácticamente solo se acuerda de la nueva experiencia, y cada vez va aprendiendo menos con el tiempo.

Factor descuento

El factor de descuento es un valor que será introducido por el usuario por lo que no es necesario especificarlo.

$$R = \begin{matrix} & \text{Action} \\ \text{State} & \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{matrix}$$

Figura 5.9: Ejemplo de matriz-Q.

Elección de la acción

La elección de la acción futura ha sido el mayor problema, ya que posiblemente ha sido la causa del defectuoso aprendizaje que se tenía al principio. Al principio se intentó seguir el proceso de Decisión de Markov para la toma de decisiones. La idea era que cada acción en un estado tuviera una probabilidad directamente proporcional a su valor en la matriz-Q, y la suma de esas probabilidades sumara 1. Por lo que las acciones con mayor valor, tienen más probabilidades de llevarse a cabo.

Este modelo es genial para incentivar la exploración al principio. Pero por desgracia, durante su implementación en la práctica no salían los resultados esperados ya que el cambio de los valores de la matriz-Q no era el apropiado y por lo tanto, las acciones tampoco eran correctas y el aprendizaje era defectuoso.

Entonces, se cambió la forma de tomar las acciones a simplemente tomar siempre la acción que tuviera el mayor valor en ese estado. De ese modo, siempre se tomaba la mejor acción, es decir, siempre se llevaba una estrategia voraz. Sin embargo, esta voracidad se compensó con la exploración incorporando recompensas negativas, menos la del tesoro que era positiva. Esto hacía que al principio siempre se exploraran los sitios por donde el agente no había pasado, ya que debido a las recompensas negativas los sitios más transitados deberían tener menos valor que los sitios más transitados.

El uso de recompensas negativas, hace que se puedan llamar penalizaciones en vez de recompensas. Se vió desde una perspectiva en la que cada acción llevaba una penalización, a no ser que encontrase el tesoro. Entonces esa penalización puede ser interpretada como la energía que pierde el agente con cada acción, por lo tanto, el agente al final buscará la política óptima que le permita gastar menos energía para llegar al tesoro.

El agente con esta estrategia hace un aprendizaje que le debería permitir conocer el camino más óptimo desde cualquier punto del mapa hacia el tesoro. Por eso en cada episodio del aprendizaje se muestra la suma de refuerzos que se han recogido.

Algoritmo QL

El algoritmo QL mostrado en el Listado 3.1 es el mismo al usado en el algoritmo de este programa. Se utiliza la ecuación (1.1) para actualizar los valores de la matriz-Q.

Más adelante se hará la explicación detallada del algoritmo QL con el código en Python.

La implementación del algoritmo QL se ha realizado en una clase llamada *RL.py*. Ahora se va a explicar detalladamente lo implementado en esta clase y su relación con los puntos anteriores.

Inicialización de variables y tablas

Como se muestra en el Listado 5.13, hay que inicializar las tablas, las acciones y los estados del agente y el tesoro. Las acciones, y los estados son referenciados de la clase *mundo.py* ya que ahí es donde se crea el mapa y se implementa el movimiento del agente. También se inicializan la matriz-Q y la tabla N como diccionarios para facilitar el acceso a sus valores.

```
6  discount = 0.75
7  acciones = Mundo.acciones
8  estados = Mundo.estados
9  final = Mundo.final
10 Q = {}
11 N = {}
```

Listado 5.13: Código de inicialización de variables para el algoritmo QL.

PosiblesAcciones(s)

Este método tiene como argumento un estado *s* del mapa. Por lo tanto, tiene como objetivo devolver las posibles acciones del agente en ese estado. Por ejemplo, si el estado es la última celda por la derecha, las acciones posibles serán todas menos la derecha.

InicializarTablas()

En este método se inicializan la matriz-Q y la tabla N. Como se puede observar en el Listado 5.14, se recorren todos los estados del mapa y se usa la función *posiblesAcciones()* para saber las acciones posibles en ese estado. Entonces, si la acción es posible se inicializa a 0, de otra manera, para especificar que es una acción imposible, se inicializa a -999. En cuanto a la Tabla N, todos los valores se inicializan a 0.

```
40 def inicializarTablas():
41     for estado in estados:
42         temp = {}
43         tempA = {}
44         posAcc = posiblesAcciones(estado)
45         for acc in acciones:
46             tempA[acc] = 0
47             if acc in posAcc:
48                 temp[acc] = 0.1
49             else:
50                 temp[acc] = -999
51
52         Q[estado] = temp
53         N[estado] = tempA
```

Listado 5.14: Código de la función *inicializarTablas()*.

```
55 def moverse(accion):
56     if accion == acciones[0]:
57         r = Mundo.moverse(0, -1)
58     elif accion == acciones[1]:
59         r = Mundo.moverse(0, 1)
60     elif accion == acciones[2]:
61         r = Mundo.moverse(-1, 0)
62     elif accion == acciones[3]:
63         r = Mundo.moverse(1, 0)
64     return r, s2
```

Listado 5.15: Código de la función *moverse()*.

Moverse(accion)

En este método, básicamente se implementa un paso o experiencia del agente. Dependiendo del argumento *accion*, el agente se moverá de un estado a otro, por lo tanto,

este movimiento conlleva la obtención de la recompensa y el estado donde se ha movido. El agente solo podrá moverse a sus estados adyacentes en un paso o experiencia. En el Listado 5.15 se puede observar como se obtiene la recompensa simplemente comparando con condiciones *if-else*.

MaxQ(s)

Esta es la función utilizada para tomar la acción con mayor valor en cada estado. En el Listado 5.16 se puede observar como funciona. Simplemente se toma el valor mayor de la fila de la matriz-Q correspondiente al estado s y se devuelve conjuntamente con la acción a que produce ese valor.

```

66  def maxQ(s):
67      val = None
68      acc = None
69      for a, q in Q[s].items():
70          if val is None or (q > val):
71              val = q
72              acc = a
74      return acc, val

```

Listado 5.16 : Código de la función *maxQ()*.

IncQ(s, a, alpha, inc)

Esta función implementa exactamente la ecuación (1.1) para actualizar los valores de la matriz-Q. Los argumentos introducidos son el estado actual s , la acción elegida a , la velocidad de aprendizaje α y el incremento.

El incremento es la suma de la recompensa r con el valor máximo que toma el agente en el estado próximo s' con la acción próxima a' que consigue ese valor, reducido por el factor descuento γ . Esta reducción con γ es debida a que, como se explica anteriormente en el MDP, los valores de las acciones futuras tienen menos importancia que los valores de las acciones actuales. En la ecuación (5.1) se puede ver el incremento como parte de la ecuación (1.1). En el Listado 5.17 se puede observar la ecuación implementada.

$$R(s) + \gamma \max_{a'} Q_{a'}(a', s') \quad (5.1)$$

```
75 def incQ(s, a, alpha, inc):  
76     Q[s][a] = Q[s][a] + alpha*(inc - Q[s][a])
```

Listado 5.17: Código de la función *incQ()*.

Run()

Este es el método principal, donde se implementa el algoritmo QL utilizando los métodos y variables anteriores. En este prototipo de python el agente empieza siempre en el mismo punto al empezar cada episodio. Cada episodio empieza cuando en el anterior se ha encontrado el tesoro.

El entrenamiento dura hasta que la suma de las recompensas o los resultados de cada episodio son iguales durante 10 episodios, esto significa que el agente ya ha aprendido lo suficiente. En el Listado 5.18 se puede ver el código del algoritmo QL.

Cuando se llegan a las 10 repeticiones seguidas de resultados, el tiempo de ejecución entre paso y paso es un poco mayor para poder ver bien como el agente ha aprendido, ya que durante el entrenamiento los movimientos son difíciles de apreciar debido al mínimo tiempo de ejecución entre cada paso. En el entrenamiento el tiempo es necesariamente mínimo, ya que si fuera mayor el agente tardaría más en aprender. También se podría haber puesto como límite de entrenamiento el número de episodios, ya que cuanto más episodios se hayan llevado a cabo por el agente, mayor habrá sido su aprendizaje.

En relación con el algoritmo QL del Listado 3.1, se va a explicar detalladamente la implementación de este algoritmo QL donde cada iteración del bucle *while* es una experiencia nueva para el agente. Por lo tanto, los pasos de cada iteración son los siguientes:

1. Se crea una variable *s* que especifica el estado actual del agente.
2. Se selecciona la acción que tiene que tomar el agente para conseguir el mayor valor en el estado actual. Eso se ha hecho con la función *maxQ()*.
3. Se utiliza esa acción para mover al agente al nuevo estado con la función *moverse()*. Con ese movimiento se recoge la recompensa *r* y el estado siguiente *s2*.
4. A continuación se incrementa el valor correspondiente en la tabla N.
5. Con la tabla N actualizada, se usa para inicializar la variable *alpha* para la velocidad aprendizaje con la ecuación explicada anteriormente.
6. Con la función *maxQ()* se recoge el mayor valor posible a tomar en el estado siguiente *s2*.

7. Se aplica la ecuación (1.1) con la función *incQ()*. Para ello el parámetro incremento es la ecuación (5.1) con la recompensa *r*, el factor descuento γ y el mayor valor tomado en el paso 6.
8. Se controla si el agente ha encontrado el tesoro, de ser así se reestablecerían los valores del episodio con la función *Mundo.reiniciarJuego()*.
9. Si se han repetido los resultados de los últimos 10 episodios, se descarga la matriz Q y se disminuye el tiempo de ejecución entre cada experiencia con la función *time.sleep()*.

Una vez creado correctamente el algoritmo QL en Python, se muestran varios experimentos de aprendizajes. En la figura 5.10, se pueden observar dos gráficas de un aprendizaje cada una donde se muestra el resultado por episodio. Se puede observar que en los dos experimentos los puntos de la gráfica cada vez son más estables formando una línea recta de puntos paralela al eje X en un único resultado, es decir, convergen en una política óptima.

```

101 def run():
102     global discount
103     time.sleep(1)
104     inicializarTablas()
105     t = 1
106     pruebas = 0
107     sleep = 0.005
108     scores = []
109     while True:
110         s = Mundo.bot
111         accion, _ = maxQ(s)
112         r, s2 = moverse(accion)
113         N[s][accion] += 1
114         alpha = float(60)/float(59 + N[s][accion])
115         _, maxVal = maxQ(s2)
115         incQ(s, accion, alpha, r + discount * maxVal)
117         t += 1.0
118         if Mundo.haReiniciado():
119             scores.append(Mundo.score)
120             Mundo.reiniciarJuego()
121             time.sleep(0.01)
122             t = 1.0
123             pruebas += 1
124         if Mundo.repeticiones == 10 and t == 1.0:
125             imprimirQmatrix()
126             sleep=0.1
127             time.sleep(sleep)

```

Listado 5.18: Código de la función *run()*, donde está el algoritmo QL.

5.4 ITERACIÓN 3: IMPLEMENTACIÓN DEL ALGORITMO QL JUNTO A LA LIBRERÍA PHASER

Esta ha sido la iteración más corta debido a que se ha adaptado el código de la iteración 2 al código de la iteración 1. Esta iteración ha durado una semana, desde que se hizo la reunión de planificación de esta iteración hasta la reunión de planificación de la iteración 4. Esta duración de una semana fue porque se iba un poco justo de tiempo con la entrega, ya que se había perdido mucho tiempo en la iteración 3.

5.4.1 Adaptación del juego Phaser al algoritmo QL

Respecto al código que se había implementado con Phaser, solo se ha modificado y añadido código en la clase *Game.js*. A continuación se explican los cambios que se han llevado a cabo.

Create()

En la función *create()* se han inicializado las variables necesarias para el algoritmo QL. Estas son las inicializaciones realizadas:

- Se ha creado una lista de acciones con los Strings de *arriba*, *abajo*, *izquierda* y *derecha*.
- Se ha creado un diccionario u objeto de JavaScript para asociar las acciones a los respectivos incrementos en la posición del agente. En el Listado 5.19 se puede ver el código.
- Se ha creado una lista de estados con la función *inicializarEstados()*.
- Se ha inicializado la matriz-Q y la tabla N exactamente como si hizo con el código en Python. Son dos diccionarios y se inicializan con la función *inicializarTablas()*.
- Se han creado variables contadoras del número de episodios y de los resultados de cada episodio. También se ha creado una variable para contar las iteraciones o pasos de cada episodio. Por último, se han creado dos variables para especificar el número máximo de episodios que tendrá el entrenamiento y el número máximo de iteraciones por episodio.
- Se han creado nuevas variables para calcular ΔQ . Dependiendo de ΔQ se sabe si se debe parar el entrenamiento. Ya se explicará más adelante como funciona y por qué ya no se usan las repeticiones de resultados por episodio para parar el entrenamiento.

Estas nuevas variables creadas son una lista con las mejores valoraciones de cada estado, un lista igual pero del paso o experiencia siguiente y el valor del ΔQ mínimo.

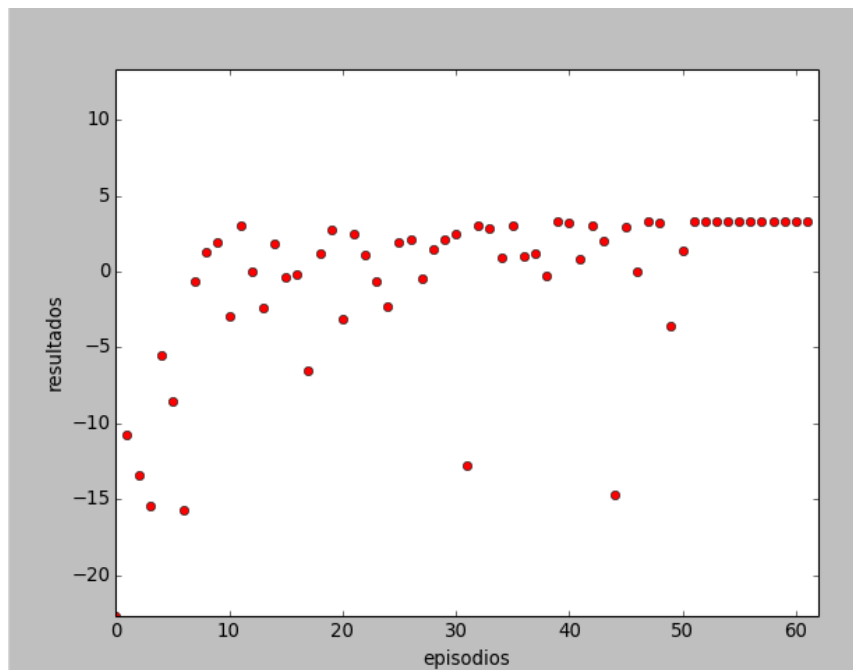
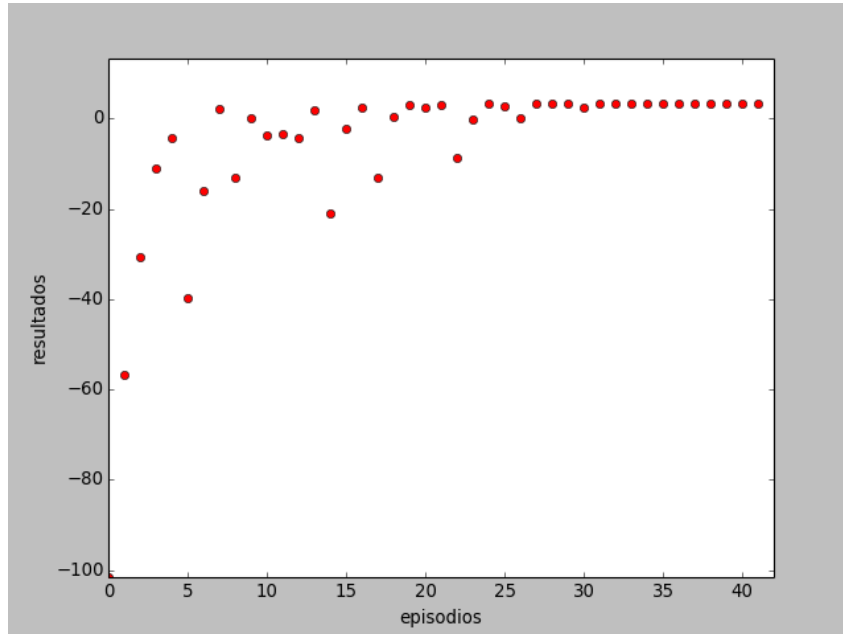


Figura 5.10: Gráficas de dos aprendizajes.

IncQ(s,a,alpha,inc)

Esta función es exactamente igual a la implementada en Python.

```
138  this.inc = {  
139      "arriba": [0, -1],  
140      "abajo": [0, 1],  
141      "izquierda": [-1, 0],  
142      "derecha": [1, 0]  
143  };
```

Listado 5.19: incrementos para las posiciones del agente.

MaxQ(s)

Esta función tiene la misma funcionalidad que la implementada en Python pero es un poco diferente ya que JavaScript no tiene las mismas funciones para recorrer diccionarios que Python.

GetPosiblesAcciones(s)

Es la misma función a la implementada en Python. Es usada en la función *inicializarTablas()* para la matriz-Q y la tabla N.

Moverse(accion)

Este método no es exactamente igual al implementado en Python ya que no devuelve la recompensa de la acción ni el siguiente estado. Únicamente se desplaza el Sprite del agente a la nueva posición. Esto se hace con la suma de la posición actual del agente al incremento asociado a la acción multiplicado por el ancho o largo de la celda, dependiendo si es movimiento en el eje de las X o de las Y. En el Listado 5.20 se puede ver el código de este método.

```
384  moverse: function(accion) {  
385      this.spriteBot.body.x += this.inc[accion][0] * this.map.tileWidth;  
386      this.spriteBot.body.y += this.inc[accion][1] * this.map.tileHeight;  
387  },
```

Listado 5.20: Código de la función moverse().

InicializarEstados()

En este método simplemente se devuelve una lista con los estados del mapa. Para ello, simplemente se recorre con dos bucles *for* la matriz de celdas del mapa y se añaden las posiciones de cada celda a la lista *estados*, que se devuelve posteriormente.

InicializarTablas()

Este método es exactamente igual al creado en el código de Python. Se inicializa la matriz-Q y la tabla N.

MejorValoración()

Como se menciona anteriormente, el entrenamiento del agente en esta implementación termina dependiendo de ΔQ . Este valor se calcula en cada paso o experiencia del agente. Entonces para calcular este valor, hay que realizar unos cálculos en diferentes métodos. Este método es uno de ellos y se calcula la mejor valoración de cada estado, por lo tanto, se utiliza la función $\text{max}Q()$. En el listado 5.21 se muestra el código de este método.

```
318 mejorValoracion: function() {  
319     var mejorValEstados = {};  
320     for (var i = 0; i < this.estados.length; i++) {  
321         var valAcc = this.maxQ(this.estados[i]);  
322         mejorValEstados[this.estados[i]] = valAcc[1];  
323     }  
324     return mejorValEstados;  
325 },
```

Listado 5.21: Código de la función *mejorValoración()*.

DifValoraciones()

En este método se termina el cálculo empezado por la función *mejorValoración()* para obtener el valor de ΔQ . La lista de pasos para calcular este valor es la siguiente:

1. Crear una lista *diferencial* que tendrá las diferencias entre las listas de las mejores valoraciones para cada estado. Estas listas son las creadas en el método *create()*.
2. Se recorre la lista de estados del mapa.
3. Para cada estado se crea la diferencia de sus mejores valoraciones en el paso actual y en el próximo. Para ello se utilizan las listas mencionadas tomando el valor absoluto de sus diferencias en ese estado.
4. Se añade esa diferencia a la lista *diferencias*.
5. Se toma el valor máximo de la lista *diferencias* y se devuelve como valor de ΔQ . El valor es el máximo porque es la diferencia más característica, que servirá para compararla con ΔQ mínimo.

En el Listado 5.22 se muestra el código de esta función. Para entender mejor este proceso, se puede observar Fig. 5.11, donde se muestra gráficamente el proceso para el cálculo de ΔQ .

```

311 difValoraciones: function() {
312     var diferencial = [];
313     for (var i = 0; i < this.estados.length; i++) {
314         diferencial.push(Math.abs(this.mejorValEstados[this.estados[i]] - this.mejor
ValEstadosProx[this.estados[i]]));
315     }
316     return Math.max.apply(null, diferencial);
317 },

```

Listado 5.22: Código de la función *difValoraciones()*.

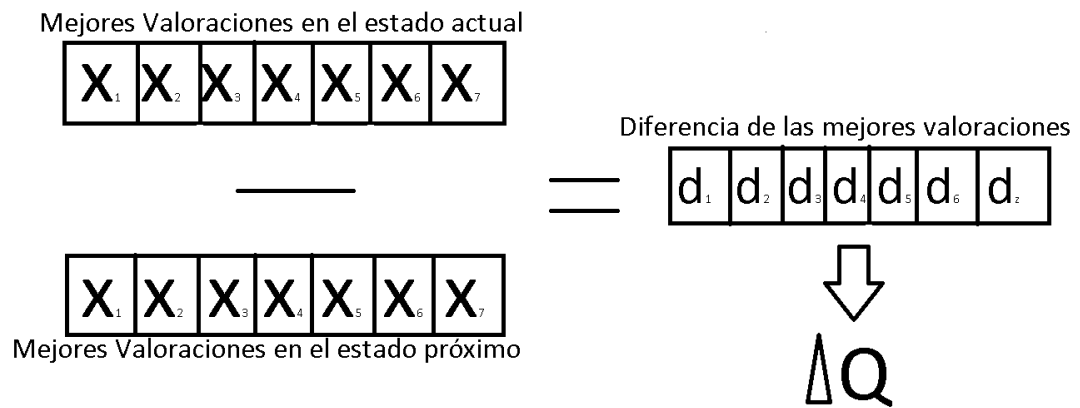


Figura 5.11: Esquema del cálculo de ΔQ .

Update()

Este es el metodo principal del algoritmo QL donde se aplican los métodos anteriores. El método *update()* es ejecutado por Phaser continuamente, es decir, cada fotograma que es renderizado equivale a una llamada al método. Por ejemplo, si la velocidad de renderizado es de 60 Fotogramas por segundo (FPS), pues el método se llamará 60 veces en un segundo.

Cada vez que se llama al método, se realiza un paso o experiencia del agente. Por lo tanto, si comparamos la función *update()* con la función *run()* del Listado 5.18, cada iteración del bucle *while* equivale a una llamada al método *update()*.

Con esto sabido, se presupone que la velocidad de cálculo o aprendizaje depende de los FPS del juego, por lo que no se ha podido sobrecargar mucho la complejidad del mundo. Se ha podido eliminar el movimiento del agente, pero para visualizar el ApR se ha visto necesario mostrarlo.

En el Listado 5.23 se puede ver el código de la función. Las diferencias respecto a la función *run()* del código en Python son las siguientes:

- Se tiene una variable booleana *parar* para saber cuando parar de entrenar al agente.
- Cada vez que se empieza un nuevo episodio, el agente empieza en un estado aleatorio, por lo tanto, no se empieza siempre desde el mismo estado. Esto se hace para que se pueda aprender la política óptima desde todos los estados del mapa, entonces al final del entrenamiento el agente debería conocer el camino óptimo al tesoro desde cualquier estado. También se hace para mejorar la exploración del mapa, ya que de la otra forma hay partes del mapa que son poco explorados. El único problema de esto es que el aprendizaje tardará más en llevarse a cabo, pero el resultado será mejor.
- Debido a que el agente empieza en estados distintos en cada episodio, la condición para acabar el entrenamiento es distinta ya que no se pueden dar repeticiones en los resultados por episodio. Esto es debido a que el camino óptimo del agente en cada episodio es siempre distinto. Por ello se utiliza el valor ΔQ . Entonces para que el entrenamiento acabe, el valor de ΔQ debe de ser menor que el ΔQ mínimo un número de iteraciones consecutivas. En este ejemplo, el número de veces consecutivas es 10. Si nunca se llega a este número de veces consecutivas, el entrenamiento para en un número máximo de episodios. Además, para que se contabilice el valor de ΔQ , se tienen que haber superado un mínimo de pasos, ya que al principio del entrenamiento el valor de ΔQ suele ser 0. ΔQ al principio suele ser 0 porque como las recompensas son negativas, las acciones sin intentar serán las que mejor valoración tengan. Por lo tanto, el valor de ΔQ se estabiliza cuando se han intentado la mayoría de las acciones un número considerable de veces.
- Si no se encuentra el tesoro antes del número máximo de iteraciones por episodio, se empieza un nuevo episodio.

Una vez explicada la implementación del algoritmo QL en el sistema junto a sus respectivas diferencias a la implementación en Python, se muestran dos experimentos. En Fig. 5.13 se muestran estos dos experimentos.

Se puede observar que ahora también convergen en una política óptima. Sin embargo, la convergencia ya no es una línea recta paralela al eje X en un único resultado como en Fig. 5.12 debido a que el agente empieza en cada episodio en un estado distinto. Esto hace que

la convergencia cueste más y se den mayores picos en la gráfica. Por ejemplo, el agente podría empezar en una zona del mapa con refuerzos muy malos, produciendo resultados bastante malos.

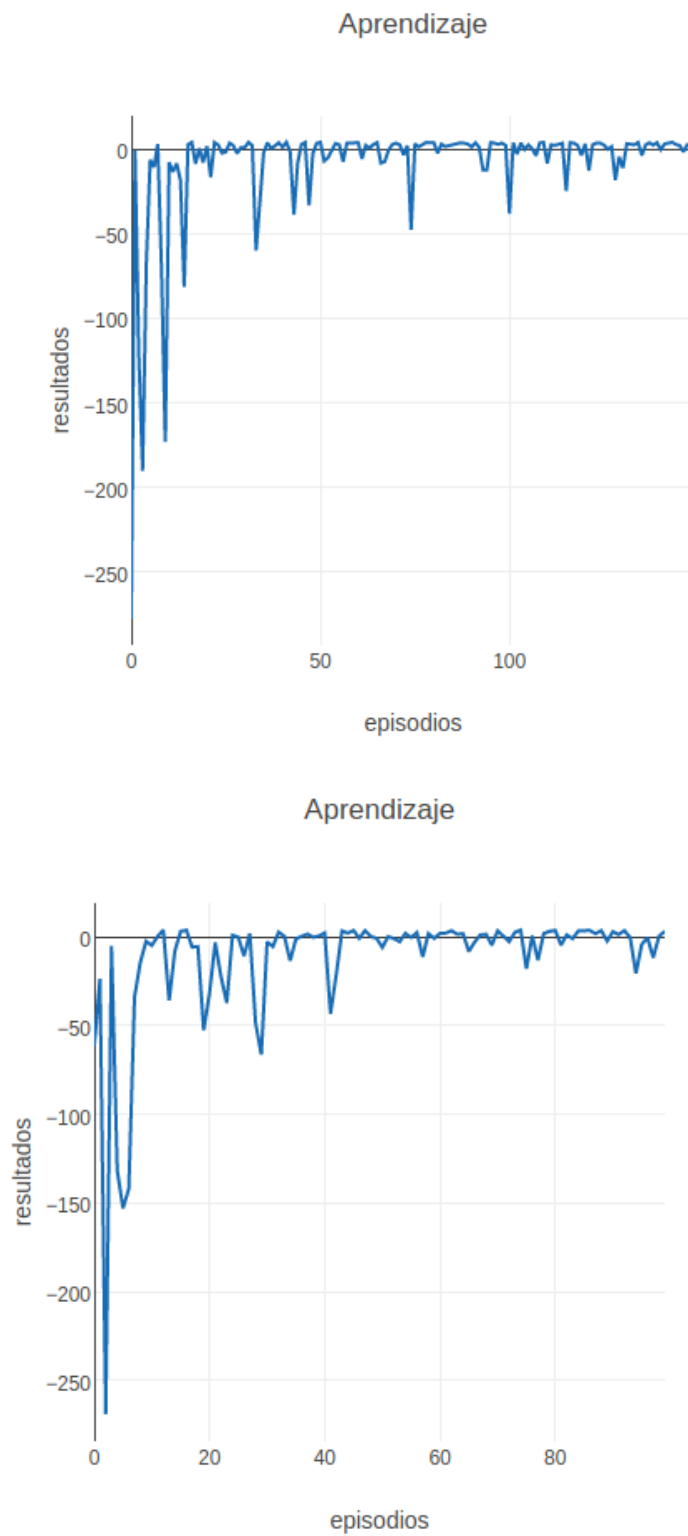


Figura 5.12: Gráficas de dos aprendizajes.

5.5 ITERACIÓN 4: IMPLEMENTACIÓN DEL SERVIDOR REST Y DE LAS INTERFACES GRÁFICAS

En esta iteración se comienza la implementación del servidor REST, así como algunas de las interfaces gráficas de la aplicación. El desarrollo de la API REST no se realiza en su totalidad porque al no haberse implementado la base de datos todavía, no tenía sentido implementar todo a su totalidad. Además, la base de datos es con la librería Flask-SQLAlchemy, por lo tanto, la base de datos y el servidor REST son muy dependientes entre sí.

En cuanto a las interfaces gráficas, se utiliza el mismo archivo CSS para todos los archivos HTML. También se utiliza la librería de Bootstrap para ayudar en la interfaz gráfica y la librería de JQuery para facilitar la implementación de la web dinámicamente.

```
218 if (!parar) {
219     if (!this.done && this.iteraciones < iteracionesMaximas) {
220         var estado = this.getEstadoBot();
221         var accVal = this.maxQ(estado);
222         var accion = accVal[0];
223         this.moverse(accion);
224         this.checkBot();
225         var refuerzo = this.getRefuerzoBot();
226         this.N[estado][accion] += 1;
227         var alpha = 60 / (59 + this.N[estado][accion]);
228         var proximoEstado = this.getEstadoBot();
229         var accValProx = this.maxQ(proximoEstado);
230         var inc = refuerzo + discountFactor * accValProx[1];
231         this.mejorValEstados = this.mejorValoracion();
232         this.incQ(estado, accion, alpha, inc);
233         this.mejorValEstadosProx = this.mejorValoracion();
234         this.difQ = this.difValoraciones();
235         this.iteraciones += 1;
236     } else {
237         this.score = 1;
238         this.iteraciones = 0;
239         this.pasos++;
240         this.spriteBot.x = this.ajustarPunto("x");
241         this.spriteBot.y = this.ajustarPunto("y");
242         this.done = false;
243     }
244     if (this.difQ < this.difQmin && this.pasos > 30) {
245         this.repeticiones++;
246     } else {
247         this.repeticiones = 0;
248     }
249     if (this.repeticiones === repDifQ || this.pasos === maxPasos) {
250         parar = true;
251     }
252 }
253 }
```

Listado 5.23: Código de la función *update()*

5.5.1 API REST

Al haber acabado la implementación del algoritmo QL en nuestra aplicación con Phaser, se implementa un servidor de tipo Transferencia de Estado Representacional (REST). Con este servidor se aceptan peticiones HTTP, con las que el usuario solicita la aplicación del algoritmo QL sobre el mapa que éste haya introducido.

Esta implementación se lleva a cabo con Flask, por lo tanto, la estructura de directorios de la aplicación es algo esencial para el correcto funcionamiento. La estructura es la siguiente:

- /static
 - /js
 - /css
 - /assets
- /templates
- App.py

El directorio *static* está formado a su vez por tres directorios. El directorio *js* contiene los archivos de JavaScript, el directorio *css* contiene los archivos CSS y el directorio *assets* contiene las imágenes del juego Phaser.

Los directorio *static* y *templates* deben llamarse así para que Flask puede acceder a ellos mediante la función *url_for()*, que se verá más adelante. En este caso solo se van a incluir en *templates* los archivos HTML de la ventana para Iniciar Sesión y de la ventana para el entrenamiento y visualización del agente.

Una vez explicada la estructura de la API, se explica a continuación la implementación del archivo *app.py*, ya que es donde se implementa el servidor REST.

Lo primero ha sido importar la librería Flask y crear una instancia de Flask, como se muestra en el Listado 5.24. El argumento `__name__` del objeto Flask es el nombre del módulo de la aplicación.

```
9   from flask import Flask
10  app = Flask(__name__)
```

Listado 5.24: Crear la instancia de Flask.

Después, hay que saber arrancar el servidor. Para ello se utiliza el código del Listado 5.25.

```
178  if __name__ == "__main__":
179      app.secret_key = os.urandom(12)
180      app.run(debug=True)
```

Listado 5.25: Código para arrancar el servidor.

Como se puede apreciar en la línea 179, se ha creado una clave secreta aleatoria. Esto se hace para que no se puedan editar las cookies de los usuarios.

Ahora se procede a implementar la parte más importante de la aplicación ya que representa la forma en la que se trabaja en esta librería. Se usa el decorador *route()* para decirle a Flask qué URL va a llamar a esta función. Como primer argumento, se indica la URL que va a llamar a esa función.

En el Listado 5.26 se puede ver la implementación para la ventana de Iniciar sesión. En el decorador *route()* la URL es */login* y el segundo argumento es la lista de métodos HTTP que acepta la función. En este caso, *login* acepta *GET* y *POST*. *GET* es para mostrar la ventana del *login* al usuario y *POST* es para recibir el formulario que el usuario rellene cuando inicie sesión.

Para saber cual de los dos métodos se aplica a la función, se utiliza la función de Flask *request.method*. En este caso, si se trata de *GET*, se devuelve el HTML del *login* con la función de Flask *render_template()*, cuyo argumento es el nombre del archivo HTML que se encuentra en el directorio *templates*.

Si es el caso de *POST*, se toman los datos del formulario con la función de Flask *request.form*. Así, se puede ver de la línea 15 a la 18 como se recogen los datos del usuario y se comparan para ver si son correctos. Si se introducen unos datos correctos, se accederá a la cuenta del usuario con la función de Flask *redirect()*. Esta función redirecciona a la URL indicada por la función de Flask *url_for()*, donde sus argumentos son el nombre de la función asociada a un decorador *route()* y los argumentos que se pasan como parámetros a ese

decorador. Si los datos son incorrectos se manda un mensaje de error con la función *flash()* y se devuelve de nuevo el HTML del *login*.

Después se ha implementado la función para el decorador *route()* de la cuenta de los usuarios. Sin embargo, como la implementación de la API REST hasta entonces ha sido simple, únicamente se muestra la ventana del entrenamiento cuando el usuario entre en su cuenta. En el Listado 5.27 se muestra la implementación para la ventana del entrenamiento.

La URL creada tiene asociado el nombre del usuario como se puede ver en la línea 27. Se usa */home/<email>* indicando que *email* es el argumento recibido por la llamada a la función asociada al decorador *route()*. De esta manera se crea una URL única para cada usuario. Posteriormente se devuelve el HTML del entrenamiento.

Una vez hecho esto, no se implementa nada más en la API REST hasta la creación de la base de datos en la siguiente iteración.

```
12 @app.route('/login', methods=['GET', 'POST'])
13 def login():
14     if request.method == 'POST':
15         POST_USUARIO = str(request.form['email'])
16         nombre = POST_USUARIO.split('@')[0]
17         POST_PWD = str(request.form['password'])
18         if POST_USUARIO == 'admin@gmail.com' and POST_PWD == 'admin':
19             return redirect(url_for('home', email = nombre))
20         else:
21             flash('CAMPOS INCORRECTOS')
22             return render_template('login.html')
23     else:
24         return render_template('login.html')
```

Listado 5.26: Código para la ventana *login*.

```
27 @app.route("/home/<email>")
28 def home(email):
29     return render_template('index.html', name = email)
```

Listado 5.27: Código para la ventana del entrenamiento.

5.5.2 Implementación de la interfaz gráfica para iniciar sesión

La implementación de esta interfaz gráfica se realiza en el archivo *login.html*. Se usa la librería de Bootstrap. Esta librería es importada como se puede ver en el Listado 5.28. Se puede observar que se utiliza la función *url_for()* de Flask entre doble corchete. Para que HTML reconozca funciones de Flask se utiliza el Framework Jinja2, que viene incluido en la librería de Flask. Por lo tanto, se usa el doble corchete para indicar que es una función de Flask. Todas las importaciones se tienen que hacer con esta función.

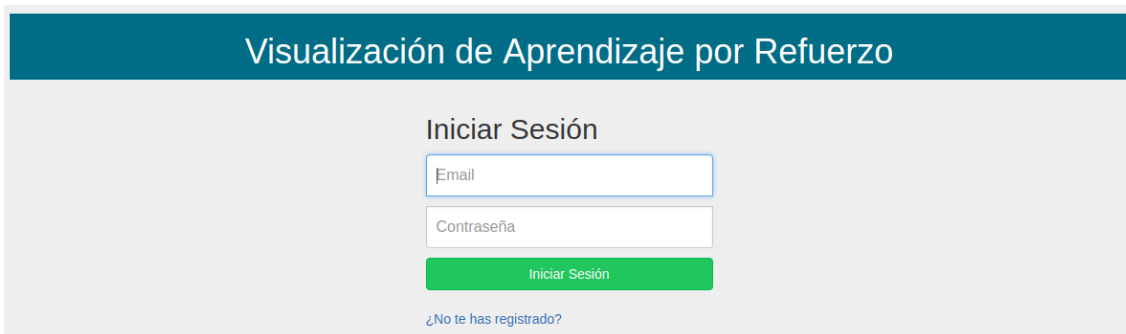
```
8 < script src = "{{ url_for('static', filename='js/lib/bootstrap.min.js') }}" > < /script>
```

Listado 5.28: Código para importar Bootstrap con Jinja2.

En cuanto a la implementación del *body*, se ha usado Bootstrap para dividir la interfaz en 2 filas. Estas filas contienen lo siguiente:

- **Fila 1:** Contiene el título de la aplicación. Esto se expone con la etiqueta *h1*.
- **Fila 2:** Contiene un formulario para iniciar sesión y un controlador de mensajes de error de Flask. Primero se implementa un básico formulario. Posteriormente se manda el formulario a la URL */login*. Hay que añadir que el botón del formulario es de una clase de Bootstrap. Debajo del formulario, añade un código de Jinja2 para controlar dinámicamente los mensajes enviados por la función *flash()* de Flask. Para ello se usa la función *get_flashed_messages()*.

El resultado de la anterior implementación da lugar a la interfaz gráfica que se puede observar en Fig. 5.13.



Visualización de Aprendizaje por Refuerzo

Iniciar Sesión

Email

Contraseña

Iniciar Sesión

[¿No te has registrado?](#)

Figura 5.13: Interfaz gráfica del *login*..

5.5.3 Implementación de la interfaz gráfica del entrenamiento.

En esta interfaz se usa la librería Bootstrap y la librería JQuery para la funcionalidad de los controles del entrenamiento. La implementación de esta interfaz gráfica se realiza en el archivo *index.html*.

Se ha reutilizado el código del archivo HTML creado en la iteración 3. Todos los archivos de JavaScript han sido importados, así como el archivo CSS. La interfaz implementada se puede observar en Fig. 5.14.

La implementación del *body* se compone de dos filas. La primera es un simple menú de usuario donde se puede cerrar sesión. La segunda se divide en dos columnas. Cada columna de la segunda fila se explica en los siguientes puntos:

- La primera columna se divide en tres filas. La primera fila contiene una caja de cuatro botones para el control y visualización del entrenamiento. Los botones son *entrenar*, *parar*, *reiniciar* y *mostrar/quitar valoraciones de los estados*. La segunda fila contiene un indicador con el estado del agente y el mapa Phaser. Los estados son *parado*, *entrenando* y *entrenado*. La última fila contiene los valores de un estado del mapa cuando se deja el cursor sobre ese estado en el mapa.
- La segunda columna se divide en tres filas. La primera fila se divide a su vez en dos columnas. La primera de esas columnas es un formulario donde el usuario tiene que introducir los parámetros para el aprendizaje; y en la segunda se muestran los datos de ese aprendizaje. La segunda fila tiene la lista de tipos de refuerzos del mapa junto a los valores que introduce el usuario. La tercera fila tiene dos botones, uno para poner valores por defecto a los parámetros del aprendizaje, y otro para descargar la matriz-Q en un archivo CSV.

Hay que señalar que los botones del panel de control del entrenamiento son de una clase de Bootstrap. En cuanto al evento de tipo *click* de cada botón del panel de control del entrenamiento, se han creado las siguientes funciones en la clase *Game.js*.

Botón entrenar

Se ha implementado la función *empezarJuego()*. Tiene como objetivo recoger los parámetros para empezar el entrenamiento y avisar al juego Phaser para comenzar a entrenar. Para ello se usa una variable global de tipo *Boolean* llamada *parar*, con el fin de que la función *update()* permita al agente generar experiencias o no. También se cambia el estado del agente a *entrenando*.

Botón parar

Se ha implementado la función *pararJuego()*. En esta función únicamente se cambia el estado del agente a *parado* y la variable *parar* se asigna a *true*. Por lo tanto, el agente no entrena en estos momentos.

Botón reiniciar

Aquí se cambia el estado del agente a *parado* y se empieza el juego Phaser de nuevo, por lo tanto, se llama a la función de Phaser `game.state.start('Game')`.

Botón mostrar o quitar valoraciones

Se implementa en la función `círculos()`. Es una nueva funcionalidad que se le añade al usuario para que pueda visualizar mejor las valoraciones de cada estado. Entonces cuando se pulsa el botón se muestra, o se deja de mostrar si se estaba mostrando, un círculo encima de cada estado, cuyo tamaño es directamente proporcional a la máxima valoración de ese estado.

Se crea otra variable global de tipo *Boolean* llamada *mostrar* para controlar la muestra de los círculos en el método `update()`. Para mostrar los círculos, eliminarlos y ajustarlos se utilizan las funciones `crearCírculos()`, `ajustarCírculos()` y `destruirCírculos()`. Estas funciones se explican en los siguientes puntos:

- **CrearCírculos():** Se crea un diccionario *círculos* en el que se asigna a cada estado el Sprite de un círculo.
- **AjustarCírculos():** Posteriormente se escala el tamaño de estos círculos dependiendo del máximo valor de cada estado. Para hacer de forma correcta el escalado, se toma este valor y se normaliza entre 0 y 1. Sin embargo, los valores que sean menores que 0.1, se dejan en 0.1 porque si no el círculo es inapreciable. Entonces, cada Sprite se escala con la función `scale.setTo()` de Phaser.
- **DestruirCírculos():** Se eliminan los Sprites de cada círculo. Para eliminar Sprites en Phaser se usa la función `destroy()`.

Un ejemplo del resultado de pulsar el botón *Mostrar/Quitar valoraciones* se muestra en Fig. 5.15. En este ejemplo se puede apreciar la distribución de las valoraciones en el tamaño de los círculos, siendo mayores los círculos que se encuentran cerca del tesoro.

5.6 ITERACIÓN 5: IMPLEMENTACIÓN DE LA BASE DE DATOS Y FINALIZACIÓN DEL SERVIDOR REST.

Esta es la última iteración del desarrollo de la aplicación, donde se añaden las interfaces gráficas que faltan, se implementa la base de datos y se acaba la implementación del servidor REST con Flask.

5.6.1 Implementación de la interfaz gráfica del registro

Para la interfaz gráfica del registro se ha creado otro archivo HTML llamado *registro.html*. Se utiliza el archivo CSS y Bootstrap, como en las otras interfaces. Esta interfaz es prácticamente igual a la del *login*, solo se diferencia brevemente en el formulario. El formulario al ser completado es enviado a la URL */registro*. Además, este formulario pide al usuario confirmar su contraseña y que no añada un correo electrónico ya existente.

En Fig. 5.16 se puede observar el resultado de la implementación de la interfaz. Hay que añadir que también se han usado botones predeterminados de Bootstrap.

Figura 5.14: Interfaz gráfica del entrenamiento.

5.6.2 Implementación de la interfaz gráfica de la carga de mapas

Esta interfaz se crea en la clase *carga.html*. Se utilizan dos filas de Bootstrap. La primera es la misma barra de menú de usuario que la interfaz del entrenamiento. La segunda fila es un complejo formulario explicado en los siguientes puntos:

- El formulario tiene dos formas, dependiendo de si selecciona la opción *abrir mapa nuevo* o *cargar mapa*. Estas dos formas quieren decir que algunos elementos del mapa aparecerán o se esconderán dependiendo de la opción. Para ocultar o enseñar los elementos se ha utilizado el atributo de CSS *style.display*.

- Si la forma es *abrir mapa nuevo*, se le pide al usuario introducir el mapa en formato JSON y un conjunto de patrones. Esta opción es seleccionada cuando el usuario quiere introducir en su cuenta un mapa por primera vez.
- Cuando el usuario quiera seleccionar un mapa que ha introducido con anterioridad, seleccionará la opción *cargar mapa*. Entonces únicamente le aparecerá un listado con los mapas que ya ha cargado en su cuenta, de donde deberá seleccionar uno.

Para ver el resultado de esta implementación, en Fig. 5.17 se muestran las dos formas de esta interfaz. La interfaz de arriba es la opción *abrir nuevo mapa* y la de abajo es la opción *cargar mapa*.

5.6.3 Implementación de la base de datos

Como se explica con anterioridad, la base de datos se crea con la librería SQLAlchemy, que viene incluida en la librería Flask-SQLAlchemy para una mejor compenetración entre ellos.

La finalidad de la base de datos es almacenar la lista de usuarios registrados con sus mapas asociados, permitiendo así que los usuarios puedan registrarse, iniciar sesión, así como cargar y añadir mapas.

Para crear la base de datos correctamente, se han implementado los archivos *models.py* y *database.py*. Estos archivos luego son importados en *app.py* para añadirlo a Flask. En el archivo *models.py* se han creado las dos tablas y la relación entre ellas. Cada tabla es una clase de Python. Las dos tablas se forman de la forma siguiente:

- La tabla *Users* contiene los usuarios con sus respectivas contraseñas. La clave primaria es el ID de cada usuario. También se establece una relación con la tabla *Mapas* para saber los mapas que tiene cada usuario. En el Listado 5.29 se puede ver el código para implementar esta tabla. Hay que especificar el tipo que es cada elemento y en la línea 11 se puede observar como se hace la relación con la función *relationship*.
- La tabla *Mapas* contiene cada mapa que han añadido los usuarios con sus propiedades. Se añade una clave foránea, que es el ID del usuario que añadió ese mapa, estableciendo así la relación. El resto de propiedades o atributos de los mapas son algunos como el archivo JSON, el nombre del archivo, el conjunto de patrones, la lista de refuerzo, etc.

En cuanto al archivo *database.py*, se muestra su código en el listado 5.30. En la línea 5 se especifica que la base de datos es de tipo SQLite y el directorio donde está la base de datos. Posteriormente en la línea 6 se crea una sesión de tipo *scoped* en la base de datos con las funciones *scoped_session()* y *sessionmaker()*. Esta sesión podrá instanciarse en la clase *app.py* para acceder a la base de datos. Por último se crea el método *init_db()*, donde en la línea 11 se importan las tablas creadas en *models.py*. Este método es el llamado por *app.py* para crear una sesión, creando una nueva base de datos si no estaba creada antes.

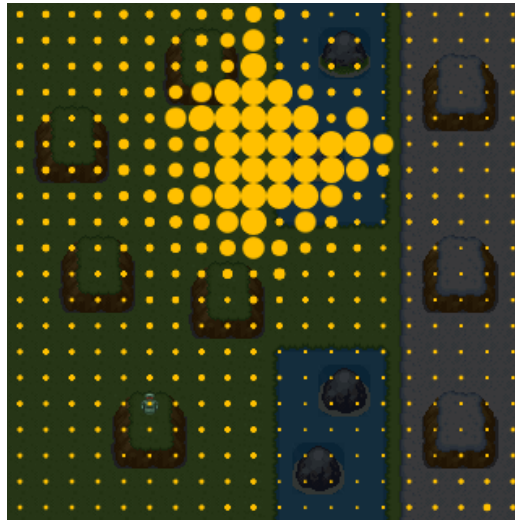


Figura 5.15: Ejemplo de muestra de círculos.

Figura 5.16: Interfaz gráfica del registro.

5.6.4 Finalización de la implementación del servidor REST

Una vez implementada la base de datos, hay que saber gestionarla con Flask en el archivo *app.py*, es decir, en el archivo donde se despliega el servidor REST.

Lo primero de todo es llamar al método *init_db()* de la clase *database.py* para crear una sesión de la base de datos. Posteriormente hay que hacer que Flask elimine las sesiones de

la base de datos automáticamente, por ejemplo, cuando la aplicación se cierra. Para ello se puede ver el Listado 5.31, donde se crea un decorador *teardown_appcontext*. La función asociada a este decorador eliminará la sesión con la función *db_session.remove()*.

Una vez hecho esto se van a explicar los cambios y añadidos que se han hecho a cada uno de los decoradores *route()* para terminar con la implementación del servidor REST.

The figure consists of two screenshots of a web application interface. Both screenshots show a dark header bar with 'VAR' and 'Home' on the left, and 'Bienvenido, pepe' on the right. The main content area is titled 'CARGAR MAPA'. In the top screenshot, there is a dropdown menu 'Selecciona el tipo de carga' with 'Cargar mapa' selected, and another dropdown 'Elige un mapa' with 'map2.json' selected. A blue 'Cargar' button is at the bottom. In the bottom screenshot, the 'Selecciona el tipo de carga' dropdown now shows 'Abrir nuevo mapa'. Below this, there are two sections: 'Abre un mapa JSON' and 'Abre un conjunto de patrones'. Each section has a 'Seleccionar archivo' button and the text 'Ningún archivo seleccionado'. A blue 'Cargar' button is at the bottom of the second screenshot.

Figura 5.17: Interfaces gráficas para cargar el mapa.

```
5 class User(Base):
6     __tablename__ = "users"
7
8     id = Column(Integer, primary_key=True)
9     username = Column(String)
10    password = Column(String)
11    maps = relationship("Map")
12
13    def __init__(self, username, password):
14        self.username = username
15        self.password = password
```

Listado 5.29: Código para crear la tabla *Users*.

```

5  engine = create_engine('sqlite:///data/usuarios.db', convert_unicode=True)
6  db_session
=scoped_session(sessionmaker(autocommit=False, autoflush=False, bind=engine))
7  Base = declarative_base()
8  Base.query = db_session.query_property()
9
10 def init_db():
11     import models
12     Base.metadata.create_all(bind=engine)

```

Listado 5.30: Código del archivo *database.py*.

Login()

En esta función se reutiliza el código del Listado 5.25, añadiendo una consulta a la base de datos para el usuario y contraseña. Para realizar la consulta con SQLAlchemy, se hace de forma muy sencilla con la función *query*. Como lo que se quiere es realizar una instrucción SELECT para el usuario y contraseña, se usa la función *query.filter()*, donde sus argumentos son los atributos para esa consulta.

Posteriormente, se quiere saber si en la base de datos está ese usuario con la contraseña correspondiente, por lo que a la consulta anterior hay que aplicarle la función *first()*. En el Listado 5.32 se puede ver el código de la consulta. También se muestra como acceder a la cuenta del usuario con la función *redirect()* y *url_for()*, poniendo como parámetros el nombre del usuario y su ID.

```

172 @app.teardown_appcontext
173 def shutdown_session(exception=None):
174     db_session.remove()

```

Listado 5.31: Código para cerrar sesiones de la base de datos.

```

20 query = User.query.filter(User.username==POST_USUARIO, User.password==POST_PWD)
21 result = query.first()
22 if result:
23     return redirect(url_for('home', email = nombre, idUser = result.id))

```

Listado 5.32: Código para hacer la consulta y acceder a la cuenta del usuario.

```

45 user = User(POST_USUARIO, POST_PWD)
46 db_session.add(user)
47 db_session.commit()
48 return redirect(url_for('login'))

```

Listado 5.33: Código para añadir un usuario a la base de datos.

Registro()

Esta función es nueva y sirve para gestionar el registro de los usuarios. Es muy parecida a la función *login()*. Acepta *GET* y *POST* como métodos HTTP. Si el método es *GET*, simplemente se devuelve el archivo HTML del registro. De la otra forma, cuando es *POST*, se recibe el formulario del usuario para registrarse y se vuelve a hacer otra consulta para ver si el usuario existe. Si el usuario no existe y se ha verificado la contraseña correctamente, se añade el usuario y contraseña a la base de datos. En el Listado 5.33 se muestra el código para añadir un usuario en SQLAlchemy. En la línea 45 se instancia un objeto de tipo Usuario. Posteriormente, en la línea 46 se añade a la base de datos con la función *db_session.add()*. Y por último, en la línea 47 se confirma el cambio a la base de datos con la función *db_session.commit()*.

Home()

Esta es la función principal del servidor REST ya que gestiona la cuenta de los usuarios. Este método controla la carga del mapa Tiled y posteriormente se encarga de mostrar correctamente la interfaz del entrenamiento.

También acepta los métodos HTTP *GET* y *POST*. Cuando es una petición *GET*, se cargan todos los mapas del usuario y se muestra la interfaz de carga. En el Listado 5.34 se muestra ese código. En la línea 60 se hace una consulta para todos los nombres de los mapas de ese usuario con la función *all()*. Posteriormente se añaden esos mapas a una lista y en la línea 63 se devuelve la interfaz de carga con el nombre del usuario, el ID del usuario y la lista de mapas como argumentos.

Si es una petición *POST*, es para recibir el formulario de carga del usuario. Esta parte contiene una gran cantidad de control de errores por si el usuario carga algún mapa Tiled erróneamente.

Como se dijo anteriormente, este formulario tiene las formas de *abrir mapa nuevo* y *cargar mapa*.

En cuanto a *abrir mapa nuevo*, se realiza una gran cantidad de control de excepciones, como si el formato del archivo no es JSON, si ya existe ese mapa en la base de datos, etc.

Si todo es correcto, se añade ese mapa a la base de datos junto con sus propiedades. Las propiedades dependen del mapa que se haya introducido, ya que se leen y guardan automáticamente en este método. Los detalles sobre las propiedades son las siguientes:

- **Archivo JSON:** Este archivo se añade a la base de datos como un String, ya que SQLAlchemy no acepta columnas de archivos JSON en SQLite. Por lo tanto, se almacena como un String.
- **Nombre del Archivo JSON:** Sirve para mostrar al usuario la lista de mapas que tiene cargados.
- **Conjunto de patrones:** Es necesario para la correcta carga del mapa Tiled en Phaser.
- **Nombre de la capa de objetos:** Es necesario saber el nombre para que Phaser pueda cargar la lista de objetos de esta capa.
- **Lista de nombres de las capas de patrones:** Sirven para añadir las distintas capas al Mapa Tiled en Phaser.
- **Lista de refuerzos:** Es necesaria para mostrar en la interfaz de entrenamiento el listado de los refuerzos a los que el usuario asigna los valores.
- **Tamaño del mapa en píxeles:** Esta información es necesaria para renderizar el juego Phaser correctamente respecto a su tamaño, y también para controlar si es demasiado grande.
- **ID del usuario:** También es imprescindible guardar el ID del usuario que carga el mapa para establecer la relación entre los mapas y los usuarios.

```

58  if request.method == 'GET':
59      mapasUsuario = []
60      mapas = Map.query.filter(Map.idUser==id).all()
61      for mapa in mapas:
62          mapasUsuario.append(mapa.nombreMapa)
63      return render_template('carga.html',name=email,idUser =
id,mapasUsuario = mapasUsuario)

```

Listado 5.34: Código para mostrar la interfaz de carga de mapas.

Después de añadir el mapa junto a sus propiedades a la base de datos, se toman las URL de las imágenes de los círculos, del tesoro y del agente con la función *url_for()*. Una

vez hecho esto, se devuelve la interfaz de entrenamiento, donde los argumentos enviados son el nombre del usuario, el mapa y sus propiedades.

Cuando se trata de *cargar mapa*, se toma del formulario el nombre del mapa que ha seleccionado el usuario y se toma el mapa y sus propiedades de la base de datos.

En estos dos tipos de formularios, cuando se quiere devolver la interfaz gráfica de entrenamiento no se puede enviar como argumentos el archivo JSON y la imagen del conjunto de patrones directamente, ya que Phaser solo acepta una URL para cargarlos. Por ello se crean dos decoradores *route()* más con el fin de tener una URL específica para cada cosa.

En el Listado 5.35 se puede ver el código de estos dos decoradores. En la línea 160 se realiza una consulta para tomar el mapa correspondiente. En la línea 161, como el mapa almacenado en la base de datos es un String, se tiene que convertir a formato JSON con la función *json.loads()*. Y finalmente, para devolver el archivo JSON correctamente se usa la función *jsonify()*. En cuanto a la función *getTileset()*, se consulta el conjunto de patrones correspondiente en la línea 167. Y por último, se devuelve la imagen con la función *send_file()*.

Argumentos con Jinja2

Una vez desplegado el servidor REST en el archivo *app.py*, hay que tomar los argumentos enviados a cada archivo HTML. Los argumentos se pueden acceder con Jinja2. Por ejemplo, en la barra de menú de usuario hay que mostrar el nombre del usuario, el cual es pasado como argumento. En el Listado 5.36 se muestra cómo acceder a ese argumento con Jinja2. Se puede ver que el acceso se realiza con el doble corchete al argumento *name*.

Sin embargo, Jinja2 no se puede usar desde un archivo de JavaScript externo al HTML, por lo tanto, hay que pasar esos valores como parámetros a través de una función de esos archivos externos. En el Listado 5.37 se muestra cómo se pasan esos valores en el archivo *index.html*.

Por último, hay que explicar algunos cambios importantes que se han hecho en la clase *Game.js*. Estos cambios son debidos a la obligada generalización del código para el entrenamiento de cualquier mapa introducido por el usuario. El cambio principal ha sido la obligada automatización para recoger los tipos de refuerzos y sus valores, para ello se han implementado las siguientes funciones:

- **GetControlFormulario():** Esta función carga los parámetros del entrenamiento que el usuario ha introducido almacenándolos en una variable global llamada *listaInputs*. Si algún valor no se ha introducido devuelve un error.

- **GetRectángulos():** Se crea un diccionario para asociar una lista de objetos del entorno a cada tipo de refuerzo. Esta lista de objetos se obtiene a partir de las funciones *getRectanglesFromObjects()* y *findObjectsByType()*.
- **GetRefuerzoBot():** Este método tiene la misma funcionalidad que *getEntornoBot()* y se ha simplificado más para poder automatizar la lectura del diccionario de la función *getRectángulos()*. Hay que recordar que esta función devuelve la recompensa que recibe el agente por alcanzar ese estado. En ella simplemente se recorre cada una de las listas de objetos de ese diccionario y se compara si el rectángulo que forma el Sprite del agente colisiona con el rectángulo de los objetos de cada una de las listas.

```

157 @app.route("/getMap/<int:id>")
158 def getMap(id):
159     nombreMapa = request.args.get('nombreMapa')
160     mapa =
Map.query.filter(Map.idUser==id,Map.nombreMapa==nombreMapa).first()
161     mapaJson = json.loads(mapa.mapa)
162     return jsonify(mapaJson)
163
164 @app.route("/getTileset/<int:id>")
165 def getTileset(id):
166     nombreMapa = request.args.get('nombreMapa')
167     mapa =
Map.query.filter(Map.idUser==id,Map.nombreMapa==nombreMapa).first()
168     nameFileTileset = mapa.nombreTileSet + ".png"
169     return
send_file(io.BytesIO(mapa.tileSet),attachment_filename=nameFileTileset,mime
type='image/png')

```

Listado 5.35: Código de los decoradores *getMap()* y *getTileset()*.

```

31 data-toggle="dropdown">Bienvenido, {{name}} <b class="caret"></b>

```

Listado 5.36: Ejemplo de uso de Jinja2 para tomar un argumento.

```

14 <script type="text/javascript">
15     initURL("{{ urlCirculo }}","{{ urlTesoro }}","{{ urlBot
}}","{{ urlMapa }}","{{ urlTileset }}");
16     initGame("{{ nombreTileSet }}","{{ capaObjetos
}}","{{ capasBackground }}","{{ x }}","{{ y }}");
17 </script>

```

Listado 5.37: Código para pasar los argumentos con Jinja2 a archivos JavaScript.

6. CONCLUSIONES

En este capítulo se muestran las conclusiones a las que se han llegado tras el desarrollo del proyecto. Se muestra un análisis del logro de los objetivos del proyecto y por último se mencionan las posibles mejoras y aplicaciones futuras del proyecto.

6.1. LOGRO DE OBJETIVOS

El objetivo principal del presente TFG se ha alcanzado con éxito. Se ha logrado construir una aplicación visual que muestre los conceptos básicos del ApR. Para lograr esta visualización del ApR, se ha implementado el algoritmo QL, donde un agente es capaz de aprender en un entorno la política óptima para encontrar un tesoro o premio.

También se ha conseguido desarrollar una aplicación que acepte una gran variedad de mundos introducidos por el usuario, dando lugar a un gran dinamismo con el algoritmo QL.

El usuario también es capaz de visualizar distintas variaciones del aprendizaje cambiando los parámetros de este, logrando así un mejor entendimiento del algoritmo QL.

En cuanto a los objetivos específicos, se han conseguido todos sin ningún problema. Hay que resaltar que se ha tenido suerte en la existencia de herramientas como Tiled o Phaser, ya que su funcionamiento es sorprendente y permiten crear juegos o mundos geniales. La consecución de la API REST ha sido también muy interesante de desarrollar. La herramienta Flask es sorprendentemente potente y te hace ver una forma sencilla de implementar servidores REST.

Por último, el algoritmo QL es especialmente efectivo para el aprendizaje de agentes en una gran variedad de entornos. Es verdad que su implementación no es fácil, ya que es difícil saber cuando está aprendiendo correctamente el agente. Sin embargo, sorprende el hecho de cómo un algoritmo con un código tan relativamente corto puede llegar a ser tan potente.

6.2 MEJORAS FUTURAS

La aplicación implementada obtiene muy buenos resultados, pero esto no quiere decir que se pueda mejorar aún más. Las mejoras pensadas son las siguientes:

- Permitir al usuario crear mapas mucho más grandes y complejos, ya que este está bastante limitado en cuanto a tamaño y capas de objetos. Por ejemplo, el usuario podría añadir más elementos al mapa, como colisiones, obstáculos, etc.
- Dar al usuario la opción de elegir el algoritmo para el aprendizaje del agente. Se podría añadir algún algoritmo de ApR que tenga un uso de redes neuronales y ver la comparación entre ellos.
- El usuario podría meter las imágenes del agente y el tesoro a su placer, para mejorar su visibilidad en el mapa, o para otro interés de este.

ANEXO A HISTORIAS DE USUARIO

Las historias de usuario llevadas a cabo en el desarrollo del proyecto se mencionan a continuación:

Investigación y estudio del entorno Tiled	
Número: 1	Prioridad: Media
Riesgo en desarrollo: Bajo	Iteración: 1
Programador responsable: Eduardo M.	Estimación temporal: 4 horas
Descripción: Se investiga el entorno de desarrollo de mapas basados en conjuntos de patrones o celdas.	
Validación: La tarea finaliza cuando se consiga crear un mapa adecuado para su futuro uso en el proyecto, y se hayan aprendido las distintas funcionalidades del editor.	

Cuadro A.1: Historia de usuario: Investigación del entorno Tiled.

Investigación y estudio del entorno Phaser	
Número: 2	Prioridad: Media
Riesgo en desarrollo: Bajo	Iteración: 1
Programador responsable: Eduardo M.	Estimación temporal: 8 horas
Descripción: Se basa en la investigación del entorno para la lectura de mapas basados en conjuntos de patrones.	
Validación: Se completa la tarea cuando se hayan hecho algunos ejemplos de juegos en Phaser con mapas Tiled, demostrando así el aprendizaje del entorno.	

Cuadro A.2: Historia de usuario: Investigación del entorno Phaser.

Creación de mapas en Tiled de forma correcta	
Número: 3	Prioridad: Media
Riesgo en desarrollo: Medio	Iteración: 1
Programador responsable: Eduardo M.	Estimación temporal: 4 horas
Descripción: La tarea consiste en averiguar la forma de implementar mapas en Tiled, de una forma que el entorno Phaser sea capaz de leerlos sin problemas.	
Validación: La historia de usuario se da por terminada cuando se crea un mapa que luego se pueda leer con Phaser apropiadamente.	

Cuadro A.3: Historia de usuario: Creación de mapas Tiled de forma correcta.

Implementación del mundo en JavaScript	
Número : 4	Prioridad: Alta
Riesgo en desarrollo: Alto	Iteración: 1
Programador responsable: Eduardo M.	Estimación temporal: 12 horas
Descripción: Se basa en la implementación del juego Phaser para el mapa creado en el entorno Tiled.	
Validación: Se completa la tarea cuando se pueda crear un juego Phaser con un mundo Tiled apropiado para implementar posteriormente el algoritmo QL.	

Cuadro A.4: Historia de usuario: Implementación del mundo en JavaScript.

Investigación sobre el algoritmo QL	
Número: 5	Prioridad: Baja
Riesgo en desarrollo: Bajo	Iteración: 2
Programador responsable: Eduardo M.	Estimación temporal: 8 horas
Descripción: Esta tarea requiere la investigación y estudio del algoritmo QL y su aplicación a la práctica.	
Validación: La historia de usuario es finalizada cuando se ha estudiado la teoría del algoritmo y se conoce como aplicarlo en el presente TFG.	

Cuadro A.5: Historia de usuario: Investigación y estudio sobre el algoritmo QL.

Implementación del mundo en Python	
Número: 6	Prioridad: Media
Riesgo en desarrollo: Bajo	Iteración: 2
Programador responsable: Eduardo M.	Estimación temporal: 6 horas
Descripción: Se basa en la creación de un mundo visual en Python para que posteriormente se pueda usar para el algoritmo QL.	
Validación: Se termina esta historia de usuario cuando se ha implementado correctamente este mundo, para que próximamente se pueda aplicar el algoritmo QL.	

Cuadro A.6: Historia de usuario: Implementación del mundo en Python.

Implementación del algoritmo QL en Python	
Número: 7	Prioridad: Media
Riesgo en desarrollo: Medio	Iteración: 2
Programador responsable: Eduardo M.	Estimación temporal: 30 horas
Descripción: En esta tarea hay que implementar el algoritmo QL para el mundo implementado en la historia de usuario anterior.	
Validación: Esta historia de usuario finaliza cuando el aprendizaje del agente con el algoritmo QL funciona de forma correcta.	

Cuadro A.7: Historia de usuario: Implementación del algoritmo QL en Python.

Adaptación del mundo Phaser para su aplicación en el algoritmo QL	
Número: 8	Prioridad: Alta
Riesgo en desarrollo: Alto	Iteración: 3
Programador responsable: Eduardo M.	Estimación temporal: 4 horas
Descripción: Se debe adaptar el programa implementado en la historia de usuario número 4 para que se pueda aplicar al algoritmo QL.	
Validación: Esta historia de usuario se da por concluida cuando se ha modificado el mundo de una forma correcta para poder ser aplicado el algoritmo QL.	

Cuadro A.8: Historia de usuario: Adaptación del mundo Phaser para su aplicación en el algoritmo QL.

Implementar el algoritmo QL junto al mundo de Phase	
Número: 9	Prioridad: Alta
Riesgo en desarrollo: Alto	Iteración: 3
Programador responsable: Eduardo M.	Estimación temporal: 10 horas
Descripción: Esta tarea se basa en implementar el algoritmo QL creado en la historia de usuario número 7 en JavaScript junto al mundo de la historia de usuario anterior.	
Validación: Se finaliza cuando el aprendizaje del agente es el apropiado conforme al algoritmo QL en ese mundo de Phaser.	

Cuadro A.9: Historia de usuario: Implementar el algoritmo QL junto al mundo de Phaser.

Investigación sobre la librería Flask	
Número: 10	Prioridad: Baja
Riesgo en desarrollo: Bajo	Iteración: 4
Programador responsable: Eduardo M.	Estimación temporal: 6 horas
Descripción: Esta tarea se basa la investigación y estudio del entorno Flask, aprendiendo a usarlo y entender su funcionalidad.	
Validación: Esta historia de usuario termina cuando se cree haber aprendido a usar Flask básicamente para aplicarlo al desarrollo del software.	

Cuadro A.10: Historia de usuario: Investigación sobre la librería Flask.

Investigación sobre la librería Bootstrap	
Número: 11	Prioridad: Baja
Riesgo en desarrollo: Bajo	Iteración: 4
Programador responsable: Eduardo M.	Estimación temporal: 3 horas
Descripción: Se basa en un estudio sobre Bootstrap, aprendiendo su funcionalidad y uso sobre un programa en HTML e implementando algunos ejemplos.	
Validación: Se da por concluida esta historia cuando se conozca lo suficiente la librería como para aplicarla en el desarrollo de la interfaz gráfica de la aplicación.	

Cuadro A.11: Historia de usuario: Investigación sobre la librería Bootstrap.

Iniciar la implementación del servidor REST	
Número: 12	Prioridad: Alta
Riesgo en desarrollo: Alto	Iteración: 4
Programador responsable: Eduardo M.	Estimación temporal: 15 horas
Descripción: Esta tarea trata de empezar la implementación de la API REST hasta poder conectar la ventana de iniciar sesión con la ventana de la cuenta del usuario.	
Validación: La historia de usuario finaliza cuando se consigue desplegar satisfactoriamente el <i>login</i> y la cuenta del usuario.	

Cuadro A.12: Historia de usuario: Iniciar la implementación del servidor REST.

Implementar la interfaz gráfica del inicio de sesión	
Número: 13	Prioridad: Alta
Riesgo en desarrollo: Medio	Iteración: 4
Programador responsable: Eduardo M.	Estimación temporal: 2 horas
Descripción: Se basa en la implementación de una interfaz gráfica para iniciar sesión con su formulario correspondiente.	
Validación: Finaliza cuando se haya implementado correctamente esta interfaz gráfica y el formulario sea recibido correctamente por el servidor.	

Cuadro A.13: Historia de usuario: Implementar la interfaz gráfica del inicio de sesión.

Implementar la interfaz gráfica del entorno de entrenamiento	
Número: 14	Prioridad: Alta
Riesgo en desarrollo: Medio	Iteración: 4
Programador responsable: Eduardo M.	Estimación temporal: 10 horas
Descripción: Esta tarea trata de implementar correctamente la interfaz gráfica donde se visualizarán los conceptos básicos del ApR.	
Validación: Finaliza cuando se haya implementado correctamente esta interfaz gráfica y se crea válida para la visualización de los conceptos básicos del ApR.	

Cuadro A.14: Historia de usuario: Implementar la interfaz gráfica del entorno de entrenamiento.

Investigación y estudio sobre la librería SQLAlchemy	
Número: 15	Prioridad: Baja
Riesgo en desarrollo: Bajo	Iteración: 5
Programador responsable: Eduardo M.	Estimación temporal: 5 horas
Descripción: Se basa en aprender los conceptos básicos para implementar una base de datos y manejarla con Flask.	
Validación: Esta historia de usuario termina cuando se cree que se ha aprendido a usar SQLAlchemy para implementar una base de datos en Flask correctamente.	

Cuadro A.15: Historia de usuario: Investigación y estudio sobre la librería SQLAlchemy.

Implementar la interfaz de registro y carga de mapas	
Número: 16	Prioridad: Alta
Riesgo en desarrollo: Medio	Iteración: 5
Programador responsable: Eduardo M.	Estimación temporal: 10 horas
Descripción: La tarea es implementar la interfaz gráfica del registro del usuario y la carga de mapas para pasar a la fase de entrenamiento.	
Validación: Se acaba cuando la interfaz de registro funciona correctamente, enviando el formulario de forma correcta y también cuando la interfaz de carga de mapas es también completada y su funcionalidad es correcta.	

Cuadro A.16: Historia de usuario: Implementar la interfaz de registro y carga de mapas.

Implementar la base de datos con SQLAlchemy	
Número: 17	Prioridad: Alta
Riesgo en desarrollo: Alto	Iteración: 5
Programador responsable: Eduardo M.	Estimación temporal: 8 horas
Descripción: Se basa en implementar la base de datos para la aplicación con SQLAlchemy para que posteriormente se pueda gestionar desde Flask.	
Validación: La historia de usuario termina cuando se ha implementado correctamente la base de datos con SQLAlchemy y se ha comprobado el correcto acceso desde Flask.	

Cuadro A.17: Historia de usuario: Implementar la base de datos con SQLAlchemy.

Acabar la implementación del servidor REST	
Número: 18	Prioridad: Alta
Riesgo en desarrollo: Alto	Iteración: 5
Programador responsable: Eduardo M.	Estimación temporal: 15 horas
Descripción: La tarea es acabar la API REST que se empezó en la historia de usuario número 12, y finalizar así el desarrollo de la aplicación.	
Validación: Se acaba cuando el servidor REST se ha desplegado completamente y correctamente en todos sus aspectos.	

Cuadro A.18: Historia de usuario: Acabar la implementación del servidor REST.

ANEXO B MANUAL DE USUARIO

A continuación, se muestra una guía básica del uso de la aplicación del presente TFG. La guía se resume en los siguientes pasos:

1. Si el usuario está en la ventana del *login* y no está registrado, debe pulsar en el enlace que hay debajo del botón *iniciar sesión* para registrarse. En Fig. B.1 se muestra ese enlace claramente.

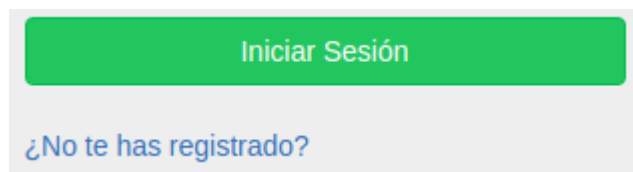


Figura B.1: Enlace para registrarse.

2. Cuando el usuario esté en la ventana de registro, debe introducir su correo electrónico, la contraseña, confirmar esa contraseña repitiéndola de nuevo y pulsando el botón *registrarse*. En caso de que no quiera registrarse, tiene que pulsar el botón *cancelar*. En Fig. B.2 se muestra de forma simple los campos a rellenar y los botones.

Figura B.2: Campos a rellenar en el registro del usuario.

3. Cuando el usuario está registrado ya y se encuentre en la ventana del *login*, debe introducir su usuario y contraseña. Posteriormente debe pulsar el botón *iniciar sesión*.
4. Si el usuario ha entrado en cuenta, se le mostrará el formulario de Fig. B.3. Debe de seleccionar la opción que desee para cargar el mapa. Si desea añadir un mapa nuevo que no ha añadido antes, debe de seleccionar el archivo del mapa en formato JSON y la imagen del conjunto de patrones. A continuación, tiene que pulsar el botón *cargar*. Pero cuando el usuario haya pulsado la opción *cargar mapa*, debe seleccionar el nombre del mapa que quiere cargar y pulsar el botón *cargar*.

CARGAR MAPA

Selecciona el tipo de carga

Abrir nuevo mapa ▼

Abre un mapa JSON

Seleccionar archivo Ningún archivo seleccionado

Abre un conjunto de patrones

Seleccionar archivo Ningún archivo seleccionado

Cargar

Figura B.3: Formulario para añadir un nuevo mapa.

5. Si el usuario desea cerrar sesión cuando está en su cuenta, tiene que ir a la barra de menú superior y pulsar en su nombre. Posteriormente debe pulsar en el desplegable que dice *cerrar sesión*.
6. Cuando el usuario haya entrado a la fase de entrenamiento, lo primero que tiene que hacer es introducir los datos para el entrenamiento del agente. Estos datos se muestran en Fig. B.4. Una vez hecho esto, si se quiere empezar a entrenar o reanudar el entrenamiento se debe pulsar el botón *entrenar*. Para parar el entrenamiento, hay que pulsar el botón *parar*. Si se quiere reiniciar el entrenamiento para que el agente y el robot aparezcan en otro punto aleatorio, se debe pulsar el botón reiniciar. Cuando se haya comenzado el entrenamiento, el usuario puede visualizar la distribución de

valoraciones en el mundo pulsando el botón *Mostrar/Quitar valoraciones*. Una vez que el usuario haya pulsado el botón *Mostrar/Quitar valoraciones*, puede poner el cursor encima de la celda que desee para ver las valoraciones de ese estado. Estas valoraciones se muestran debajo del mapa Phaser.

7. En la ventana del entrenamiento, se mostrarán los resultados recientes del entrenamiento en los datos de la parte derecha de la ventana.

Figura B.4: Datos a rellenar para el entrenamiento.

Estado	arriba	abajo	izquierda	derecha
0-0	-999	-0,105	-999	-0,102
1-0	-999	-0,106	-0,102	-0,105
2-0	-999	-0,111	-0,107	-0,111
3-0	-999	-0,114	-0,112	-0,114
4-0	-999	-0,107	-0,115	-0,111
5-0	-999	-0,111	-0,109	-0,103
6-0	-999	-0,097	-0,104	-0,102
7-0	-999	-0,098	-0,104	-0,094
8-0	-999	-0,096	-0,092	-0,101
9-0	-999	-0,105	-0,099	-0,103
10-0	-999	-0,092	-0,093	-0,097
11-0	-999	-0,094	-0,089	-0,092
12-0	-999	-0,093	-0,097	-0,082

Figura B.5: Ejemplo de matriz-Q.

8. El usuario puede descargar la matriz-Q de la iteración actual del entrenamiento pulsando el botón *descargar Q-Matrix*. En Fig. B.5 se puede ver un fragmento de un ejemplo de matriz-Q.
9. El usuario puede introducir unos valores predefinidos para el entrenamiento pulsando el botón *valores por defecto*. Hay que fijarse que a los valores de la lista de refuerzos se le introducirán unos valores aleatorios.

REFERENCIAS

- [1] Peter Norvig y Stuart J. Russell, *Inteligencia Artificial: Un Enfoque Moderno*. Prentice Hall, 1994.
- [2] Richard S. Sutton y Andrew G. Barto. *Reinforcement Learning: An Introduction*. Bradford Book, 2014.
- [3] Cynthia Andres y Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, 2004.
- [4] Pautasso, Wilde y Alarcon. *REST: Advanced Research Topics and Practical Applications*. Springer, 2014.
- [5] Ferreira y Otavio. *Semantic Web Services: A RESTful Approach*. IADIS, 2009.
- [6] Mark J.P. Wolf. *Before the Crash: Early Video Game History*. Wayne State University Press, 2016.
- [7] Burnetas y Katehakis. *Optimal Adaptive Policies for Markov Decision Processes*. INFORMS, 1997.
- [8] *Artificial Intelligence, Foundations of Computational Agents, Decision Processes*. Disponible en: http://artint.info/html/ArtInt_224.html [Accedido en: 20- Nov-2017]
- [9] C. Touzet. *Q-learning and Robotics*. IJCNN'99, 2001.
- [10] *Playing Atari with Deep Reinforcement Learning*. DeepMind Technologies. Disponible en: <https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf> [Accedido en: 21-Nov-2017]
- [11] *SQLAlchemy Wiki*. Disponible en: <https://bitbucket.org/zzzeek/sqlalchemy/> [Accedido en: 24-Nov-2017]
- [12] M. Poppendieck y T. Poppendieck, *Learn Software Development: An Agile Toolkit*. Addison-Wesley Professional, 2003.
- [13] *Extreme Programming: A gentle introduction*. Disponible en: <http://www.extremeprogramming.org/> [Accedido en: 22-Nov-2017]
- [14] S. Aggarwal. *Flask Framework Cookbook*. Packt Publishing Ltd, 2014.

- [15] N. J. Nilsson, *Principles of Artificial Intelligence*. Springer Science & Business Media, 1982.
- [16] Shai Ben-David, Shai Shalev-Shwartz. *Understanding Machine Learning: From Theory to Algorithms*. 2014.
- [17] Juul Jesper. *A history of matching Tile Games*. Disponible en: <https://www.jesperjuul.net/text/swapadjacent/> [Accedido en: 28-Nov-2017]