UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Diseño y desarrollo de un juego de navegador para comenzar a aprender construcciones básicas de programación.

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Lapeña Navarro, Aitor
Tutor: Pastor López, Óscar
Co-Tutor: Vos, Tanja
Director Experimental: Marín, Beatriz

Curso 2021-2022

Resum

Les persones i la societat s'estan tornant cada vegada més dependents del software, ja que aquest determina cada vegada més les nostres activitats de la vida diària tant en l'àmbit social com en l'empresarial. Amb l'ús cada vegada major de software, existeix una creixent necessitat de programadors que desenvolupen aqueix programari. Aprendre a programar és un procés complex que va més enllà de l'aprenentatge de la sintaxi d'un llenguatge de programació. Els estudiants han d'aprendre a resoldre problemes a través de computacions, és a dir, pensament computacional. Un procés de conceptualització eficient es converteix en un requisit essencial. La pràctica del raonament lògic i la resolució de problemes estan en el centre de les activitats realitzades pels programadors.

La millor manera d'aprendre aquestes habilitats és practicant molt, i una de les formes més divertides de fer-ho és usant jocs. La ludificació en la docència ha mostrat molts beneficis en tota mena d'àrees, per exemple salut, construcció, enginyeria i en programació. En aquest treball final de grau, s'ha desenvolupat un joc de navegador simple que els estudiants que estan començant a aprendre programació poden usar per a practicar amb habilitats de pensament computacional. El joc consta de 12 nivells, i consisteix en un robot que ha d'aconseguir una plataforma final per a superar cada nivell. Aquest objectiu l'aconseguirà mitjançant una sèrie d'instruccions que respecten la sintaxi del llenguatge Python.

Per a avaluar aquest joc, s'ha creat un formulari i s'han seleccionat 30 persones de diferents perfils per a veure l'impacte del joc. Pels resultats obtinguts, el joc ha ensenyat a les persones inexpertes en pensament computacional a desenvolupar solucions per a superar els nivells. Encara que els participants sense experiència en programació no han sigut capaces de resoldre tots els nivells, han sigut capaços de resoldre més de la meitat d'ells. En el que coincideixen pràcticament tots els participants és que han passat un bon temps jugant al joc.

Per tant, es conclou que els jocs com a forma d'aprenentatge són útils i creen més interés en el context de la programació, tant en persones sense experiència com en persones amb experiència.

Paraules clau: Python, programació, joc, ludificació, pensament computacional, aprenentatge, codi obert

---

Resumen

Las personas y la sociedad se están volviendo cada vez más dependientes del software, ya que éste determina cada vez más nuestras actividades de la vida diaria tanto en el ámbito social como en el empresarial. Con el uso cada vez mayor de software, existe una creciente necesidad de programadores que desarrollen ese software. Aprender a programar es un proceso complejo que va más allá del aprendizaje de la sintaxis de un lenguaje de programación. Los estudiantes deben aprender a resolver problemas a través de computaciones, es decir, pensamiento computacional. Un proceso de conceptualización eficiente se convierte en un requisito esencial. Es por esto que la práctica del razonamiento lógico y la resolución de problemas están en el centro de las actividades realizadas por los programadores.

La mejor manera de aprender estas habilidades es practicando mucho, y una de las formas más divertidas de hacerlo es usando juegos. La gamificación en la docencia ha mostrado muchos beneficios en todo tipo de áreas, por ejemplo salud, construcción, ingeniería y en programación. En este trabajo de final de grado, se ha desarrollado un juego

de navegador simple para que los estudiantes que están comenzando a aprender programación puedan usar para practicar las habilidades de pensamiento computacional. El juego consta de 12 niveles, y consiste en un robot que debe alcanzar una plataforma final para superar cada nivel. Este objetivo lo va a conseguir mediante una serie de instrucciones que respetan la sintaxis del lenguaje Python.

Para evaluar este juego, se han seleccionado a 30 personas de diferentes perfiles para usar el juego y valorar su posible impacto mediante un cuestionario. Los resultados obtenidos evidencian que el juego ha enseñado a las personas inexpertas en pensamiento computacional a desarrollar soluciones para superar los niveles. Aunque los participantes sin experiencia en programación no han sido capaces de resolver todos los niveles, ellos sí han sido capaces de resolver más de la mitad de los niveles. En lo que coinciden prácticamente todos los participantes es que han pasado un rato agradable jugando al juego.

Por lo tanto, se concluye que los juegos como forma de aprendizaje son útiles y crean más interés en el contexto de la programación, tanto en personas sin experiencia cómo en personas con experiencia.

Palabras clave: Python, programación, juego, gamificación, pensamiento computacional, aprendizaje, código abierto

Abstract

People and society are getting more and more dependent on software since it increasingly determines our daily live activities on social and business contexts. With the increasing use of software, comes an increasing need for programmers that develop that software. Learning how to program is a complicated process that goes beyond the learning of the syntax of a programming language. Students need to learn how to solve problems with computations, i.e. learn computational thinking. Having a sound conceptual process becomes a need. For that reason, practicing logical reasoning and problem-solving are at the center of the activities performed by programmers.

The best way to learn these skills is by practicing a lot, and one of the most fun way to do that is by using games. Gamification has shown many benefits in all kind of areas, as health, construction sector, engineering and programming. In this final degree work we will develop a simple browser game that students that are starting to learn programming can use it to practice with computational thinking skills. The game has 12 different levels, and in order to complete these levels, a robot must reach the final platform in each level. This objective will be accomplished by executing some instructions by using Python syntax.

To evaluate the game, we have selected 30 people with different profiles to use the game and evaluate its impact by a questionnaire. The results obtained provide evidence about that the game was useful for inexpert people to start developing a computational thinking in order to accomplish some levels. Altough users not experimented in programming weren't able to complete all the levels, they could solve more than a half of the levels. All players agreed that they spent a funny time playing the game.

Finally, we can conclude that games as a learning technique are useful and they allow to generate more interest in programming concepts, for experimented and not experimented people.

Key words: Python, programming, game, gamification, computational thinking, learning, open source

Índice general

Índice de figuras

CAPÍTULO 1
Introducción

Hoy en día, la programación es un concepto que prácticamente todo el mundo conoce, ya que la tecnología avanza cada vez más en un mundo donde se pretende automatizar el trabajo a través de programas con código. Podemos definir la programación como el proceso utilizado para idear y ordenar las acciones necesarias para realizar una determinada tarea en un lenguaje concreto.

Con el aumento del uso de tecnologías que se observa conforme pasan los años, se espera disponer de un conocimiento básico de programar sea algo imprescindible en cualquier currículum. Para poder empezar a programar, es necesario desarrollar un pensamiento computacional (computational thinking) [22]. Este pensamiento no es nada fácil de adquirir y no se obtiene de manera inmediata, sino que con el paso del tiempo se va desarrollando más y se pueden ir asentando las bases de la programación. A través del pensamiento computacional, se puede reconocer el problema que se presenta, buscar una solución adecuada, y estudiar las posibles salidas que se van obteniendo para sacar conclusiones.

Debido a la dificultad que conlleva el aprendizaje de la programación a los nuevos estudiantes, y a la gran cantidad de lenguajes diferentes que existen, se ha querido orientar este trabajo de fin de grado al diseño de un juego para las personas que quieren comenzar a programar de manera sencilla e interactiva. Para el diseño del juego, se va a tratar el concepto de "Gamification"(Gamificación) [23], que aplicada a la educación, es la incorporación de elementos de juegos para conseguir motivar y estimular a los alumnos de una forma divertida, mediante sistemas de puntuación, aumento de la dificultad mediante niveles, entrega de premios al superar niveles, entre otros; y que además propone un desafío y motivación a los jugadores de superarse (vea la figura 1.1).

Figura 1.1: Bases principales de la gamificación

El concepto de juego tiene un propósito principalmente de entretenimiento y ocio, mientras que los aprendizajes tradicionales, por lo general no provocan especial diversión en los estudiantes. Hay gran cantidad de investigaciones que demuestran que la gamificación aplicada a la enseñanza provoca mejores resultados en los estudiantes, por ejemplo [24] [25] [26] [27].

Si lo llevamos al terreno de la programación, un juego con elementos gamificados puede ser especialmente útil, ya que la abstracción para resolver problemas se puede representar perfectamente a través de un juego que presente la siguiente estructura: un problema ->un camino ->una solución.

La clase de juego cuyo propósito es la enseñanza de unos conceptos determinados a los jugadores del mismo, se define como *Juego Serio* [28] [29]. Para conseguir atraer y enganchar a los usuarios, se introducen elementos de gamificación en estos juegos.

La limitación principal de estos juegos es que son de código cerrado y están escritos en un único lenguaje. Una vez se sube la versión final de los juegos, no se pueden modificar de manera sencilla los desafíos de estos juegos serios. Normalmente se diseñan específicamente para un lenguaje de programación y en un idioma determinado, lo que limita el uso de los mismos a contextos determinados (una carrera, un grado, etc).

Para lograr atraer a los nuevos estudiantes de programación, en este trabajo se ha desarrollado un juego serio mecánicamente sencillo y fácil de comprender para sentar las bases de la programación en lenguaje Python. Este juego se ha desarrollado a través de una interfaz HTML5 y una lógica en JavaScript.

El juego consta de una serie de niveles que el jugador debe ir superando, cada nivel con mayor dificultad que el anterior. Todos los niveles se superan de la misma manera: consiguiendo que el robot alcance el final del nivel. Para ello, el jugador debe pensar la solución, evitando los diferentes diseños de obstáculos que impiden llegar de manera sencilla. La solución se debe introducir en un campo de texto a través de instrucciones específicas que el robot entiende. Estas instrucciones y estructuras siguen la sintaxis de Python.

Se ha tenido en cuenta la dificultad que conlleva la programación para las personas que no han tenido ningún contacto con ella antes. Por lo tanto, el juego desarrollado tiene un diccionario de instrucciones corto y muy cercano al lenguaje humano que permite realizar únicamente las acciones necesarias para superar el juego.

Para conseguir ampliar el alcance del juego implementado, el código es Open Source, de forma que tras la entrega de este trabajo se pueda seguir manteniendo el juego. En cuanto a los idiomas que se han implementado, hay una versión disponible en castellano y en inglés, de forma que podrá ser usado en más sitios que los juegos existentes.

1.1 Motivación

Como principal motivación del trabajo, existe una motivación técnica al conocer la dificultad que conlleva el aprendizaje de la programación. En la mayoría de ingenierías, con intención de desarrollar el pensamiento computacional, los alumnos deben cursar cursos de programación en los primeros años. Así, la programación se introduce de manera progresiva, y todos los conceptos que se enseñan son fundamentales para seguir avanzando en su comprensión y conocimiento. Algunos conceptos pueden ser más difíciles de comprender, y no entenderlos puede dificultar bastante el aprendizaje del resto.

Por otro lado, existe un problema respecto a la falta de preparación e información en la mayoría de institutos de los que provienen los alumnos que pretenden hacer una

carrera tecnológica. En la mayoría de los casos, no existe ninguna asignatura en su plan de estudios relacionada con la programación ni con el pensamiento lógico necesario para ello. En resumen, muchos estudiantes parten de cero cuando entran a la carrera, y eso no es algo fácil de afrontar (hablando desde mi experiencia propia).

La idea de este juego está inspirada en un juego muy similar, previamente desarrollado para aprender los conceptos básicos del lenguaje C (con un tablero, obstáculos, un robot, etc). El juego del robot en lenguaje C se verá más adelante en la sección 2.1. Entonces, existe también una motivación de desarrollar un juego con la misma arquitectura, pero esta vez para el lenguaje Python.

Por último, existe otra motivación respecto al lenguaje elegido para enseñar con este trabajo. El lenguaje Python es un lenguaje muy potente y que se usa en multitud de grandes empresas, por lo que dominarlo es un punto positivo a la hora de encontrar trabajo, además que su uso está en constante crecimiento. Además de la importancia del lenguaje, no hay prácticamente juegos serios orientados al aprendizaje de las bases del lenguaje Python, tal y como se verá en la sección 2.1.

1.2 Objetivos

Los objetivos de este trabajo son los siguientes:

1. Facilitar los primeros pasos con la programación a los estudiantes de primer año a través de un juego desarrollado en HTML5 y Javascript para atraer más su interés hacia la resolución de problemas.

2. Introducir los conceptos más básicos del lenguaje Python y su sintaxis a los nuevos estudiantes de programación de forma interactiva.

3. Explicar el concepto de testing de código a los alumnos cuando empiezan a aprender a programar.

1.3 Impacto esperado

El impacto de este trabajo es considerable. Prevemos que cerca de 400 alumnos van a usar el juego en los cursos de programación de la UPV en 2022/2023 para iniciar el aprendizaje de Python. El alcance del juego puede llegar a ser incluso mayor si se usa cada año, o si alcanza a otras universidades interesadas en la enseñanza de Python de manera sencilla e interactiva.

Además de la UPV en España, una universidad de Holanda ya conoce el juego y ha demostrado su interés por usarlo en sus cursos de programación, así que aproximadamente otros 100 alumnos van a jugar la versión inglesa del juego en el siguiente curso.

Con el uso del juego serio también se espera mejorar la actitud de los estudiantes frente a los cursos de programación en Python, ya que jugar al juego les va a motivar más a seguir aprendiendo. Esto también podría impactar positivamente su estilo de aprendizaje para los siguientes cursos de sus carreras.

1.4 Metodología

La metodología que se ha empleado para el desarrollo de este trabajo ha sido la metodología *action research* [2] (investigación-acción), como se explica a continuación.

Inicialmente, hay un problema que se busca solucionar, referido a la situación actual respecto al aprendizaje del lenguaje Python, y es que no hay medios de aprendizaje atractivos para los estudiantes, lo que provoca que no estén motivados, no practiquen suficiente y finalmente no comprendan bien los conceptos tratados en clase. La acción que se propone para resolver el problema es introducir conceptos básicos de Python de una forma más agradable para las personas interesadas, consistiendo esto en un juego con diferentes niveles que supone un reto para sus jugadores. Por último, se ha evaluado el impacto de este trabajo en los interesados para poder sacar conclusiones sobre la solución que se ha planteado.

Además, para el diseño, implementación y validación del juego se han sostenido reuniones semanales con los tutores para revisar el estado del arte, aclarar conceptos, analizar las mejores estrategias de diseño, validar el diseño y la implementación, y discutir sobre los resultados obtenidos y futuras mejoras.

1.5 Estructura del documento

El contenido de este trabajo se estructura de la siguiente manera:

1. El capítulo 2 presenta el contexto del trabajo, explicando la elección del lenguaje Python.
2. El capítulo 3 presenta el contenido que aporta este trabajo y los requisitos específicos.
3. El capítulo 4 muestra el diseño del juego propuesto y la explicación de la arquitectura del mismo.
4. El capítulo 5 presenta la parte lógica del juego, es decir, el código que da funcionalidad al juego implementado. Además, se presenta la manera acceder al juego y el repositorio GitHub que se ha creado para mantener el código fuente.
5. El capítulo 6 presenta la evaluación, validación y resultados del juego en casos reales.
6. Por último, el capítulo 7 presenta las principales conclusiones de este trabajo.

CAPÍTULO 2
Contexto

En la actualidad y cada vez más, la educación requiere de nuevas formas de atraer a los alumnos en su proceso de aprendizaje. Como se ha explicado en la Introducción, el juego serio es una de las maneras más efectivas de conseguirlo. La necesidad de superar un desafío es una de las bases principales de estos juegos, que cuentan con elementos de gamificación para hacerlos todavía más divertidos.

En el sector de la programación existen gran cantidad de juegos que enseñan las bases de los lenguajes de programación más importantes. En la sección 2.1 se van a dar ejemplos de juegos de algunos de los lenguajes de programación más importantes.

Hay una variedad enorme de lenguajes de programación (se conocen aproximadamente 680 lenguajes diferentes), y al igual que los diferentes idiomas que se pueden hablar en el mundo, cada uno tiene sus particularidades, lo que resulta que sea prácticamente imposible entender y programar a la perfección todos ellos.

Debido a esta dificultad, cada empresa, universidad o negocios del sector de las ingenierías deben elegir uno o varios lenguajes para desarrollar su código, que sean compatibles y se integren sin problemas con el resto de herramientas que ellos usan.

Los lenguajes, de acuerdo con su finalidad, pueden agruparse en tres:

1. **Lenguaje máquina:** Aquí se encuentran los lenguajes que son directamente programables e interpretables por los microcontroladores. Los códigos se basan en secuencias binarias de 0s y 1s. Una variante es el lenguaje ensamblador, que consiguió reducir los códigos fuentes al abstraer un poco estas secuencias de bits.

2. **Lenguaje de bajo nivel:** Se definen como los lenguajes en los que sus instrucciones ejercen un control directo sobre el hardware y por lo tanto están condicionados por la estructura física de las computadoras que soportan el lenguaje. Son muy eficientes al trabajar a un nivel de abstracción tan bajo. Un ejemplo es el lenguaje C (en su nivel más bajo), que permite de manera sencilla los accesos a memoria para lecturas y escrituras. Es el más usado por las empresas de electrónica.

3. **Lenguaje de alto nivel:** Los lenguajes de alto nivel no están unidos a microcontroladores ni ningún tipo de hardware, es decir, son portables. La gran diferencia que presentan frente al bajo nivel es que estos lenguajes son mucho más entendibles y se han creado teniendo en cuenta las capacidades cognitivas de los seres humanos, lo que permite una mayor facilidad de comprensión y aprendizaje. En esta categoría encontramos la mayoría de lenguajes más usados hoy en día: C, C#, Java, Javascript, Python, etc.

Dominar los lenguajes de programación de alto nivel es importante, ya que se usan en la mayoría de empresas de desarrollo de software (Javascript para el desarrollo de páginas web, Python y Java en el backend, etc.). Como podemos ver en la figura 2.1, que refleja la popularidad de los lenguajes de programación de alto nivel más usados, el lenguaje Python es actualmente el más empleado por los usuarios [1], razón principal de que este trabajo haya sido enfocado hacia el aprendizaje de Python.

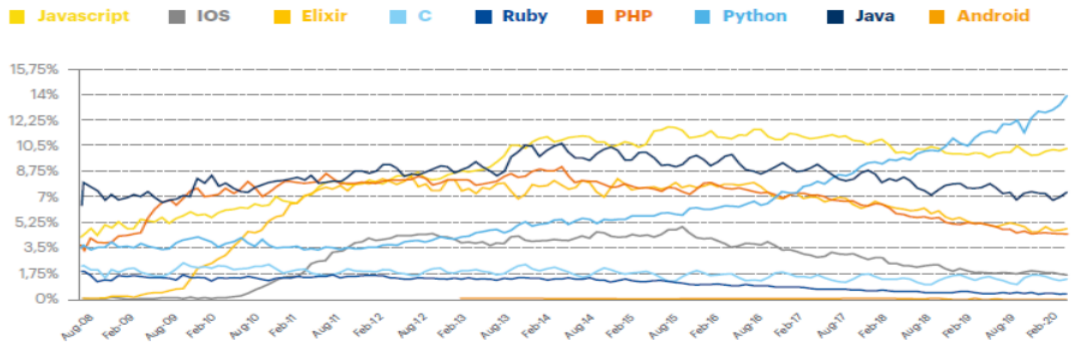


Figura 2.1: Uso de lenguajes de programación desde 2008

2.1 Estado del arte: juegos serios en programación

En esta sección se presenta cuál es el estado actual sobre los juegos serios orientados a la programación. Para ello, se va a pasar por los diferentes tipos de juegos que existen, y en el caso de las aplicaciones y videojuegos, se va a dividir la explicación por el lenguaje de programación que pretenden enseñar.

2.1.1. Juegos serios de mesa

Los juegos de mesa buscan ilustrar algún conocimiento en concreto, y están pensados para los niños más pequeños. Existen multitud de juegos de mesa que pretenden mostrar los aspectos más básicos de la programación de manera muy sencilla y general (sin enseñar un lenguaje concreto, mostrando el "pseudo-código").

El objetivo de estos juegos suele ser esquivar obstáculos y llegar a una meta mediante una secuencia lineal de movimientos del personaje.

Algunos ejemplos de juegos de mesa orientados a programación son RobotTurtles [6] y CoderBunnyz [7], que como se ve en la figura 2.2, muestran a los jugadores los conceptos básicos en pensamiento lógico, secuencias de instrucciones, bucles, condiciones, funciones, etc.

Figura 2.2: Coder Bunnyz

2.1.2. Videojuegos y aplicaciones

Los videojuegos/aplicaciones van dirigidos a usuarios de una edad normalmente mayor a los juegos serios de mesa. Pueden jugarlos tanto estudiantes, adultos, etc.

En este apartado se exponen videojuegos y aplicaciones para aprender a programar, y qué lenguajes se pretende enseñar mediante ellos. Para ello, se ha generado una clasificación por lenguajes y se va a documentar con ejemplos de juegos existentes:

2.1.2.1. Java

Java es uno de los lenguajes básicos que se enseña en gran cantidad de universidades para la introducción a la programación. Por ello, se han desarrollado algunos juegos para su aprendizaje y práctica, aunque no se han encontrado demasiados juegos dedicados a su aprendizaje.

- Robocode [10]: Es un juego de programación cuyo objetivo es codificar en Java un tanque robot para luchar contra otros robots en la arena de combate (vea la figura 2.3). Este juego requiere de un conocimiento previo del lenguaje, ya que el juego hay que instalarlo como un proyecto en el ordenador del jugador para poder usarlo, y la API que se proporciona con el juego está pensada para jugadores con algo de experiencia con Java.

El juego tiene mucha fama, existen bastantes herramientas relacionadas con el juego: un repositorio con ejemplos de robots, una comunidad de jugadores con una Wiki propia, desafíos, etc.

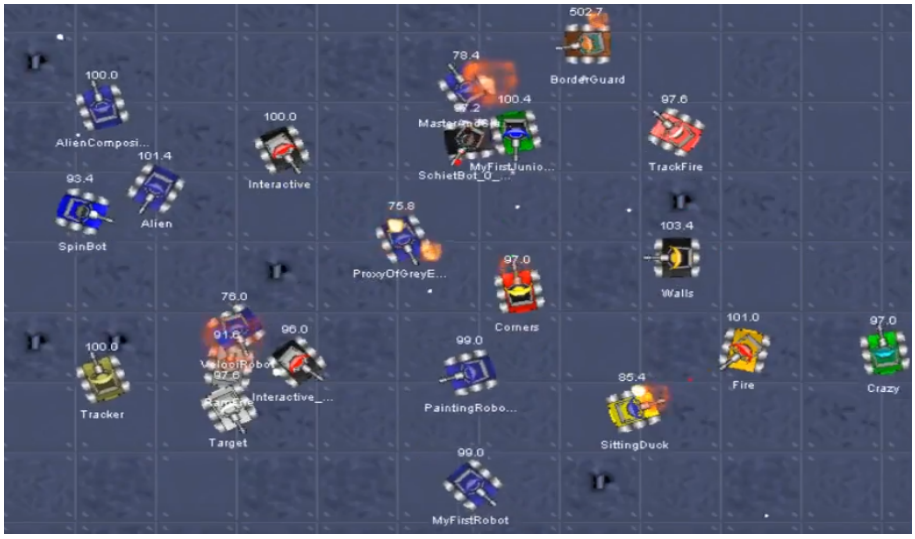


Figura 2.3: Combate en tiempo real en Robocode

2.1.2.2. Javascript

Se han encontrado una gran cantidad de juegos diseñados para la enseñanza de programación en Javascript, ya que es un lenguaje muy importante en el desarrollo de páginas webs. Al existir tanta variedad de juegos, las temáticas son diferentes: plantear problemas sencillos que se resuelven usando Javascript, introducir la sintaxis del lenguaje, desarrollar el pensamiento computacional, etc. Se van a explicar 2 juegos serios de Javascript que se han tomado como ejemplo:

1. Elevator Saga [8]:

   El juego Elevator Saga tiene unos gráficos bastante minimalistas y sencillos. El objetivo es resolver los 19 desafíos que se plantean, los cuales tratan de subir a un número de personas en un tiempo que va variando según el nivel. Para ello se puede usar un número limitado de ascensores. En la figura 2.4 se muestra la interfaz y el código Javascript que se tiene que modificar para conseguir superar los desafíos.

Figura 2.4: Interfaz de Elevator Saga

2. Untrusted [9]:

   Es un juego que requiere algo de experiencia en Javascript. Se indican una serie de desafíos para guiar al Dr.Eval a través de una máquina que va alterando la realidad (diferentes niveles). La interfaz que presenta al iniciar el juego es un mapa sencillo (vea la figura 2.5), que va aumentando su dificultad con obstáculos (vea la figura 2.6), y a través de los retos en Javascript se puede llegar al final del nivel.

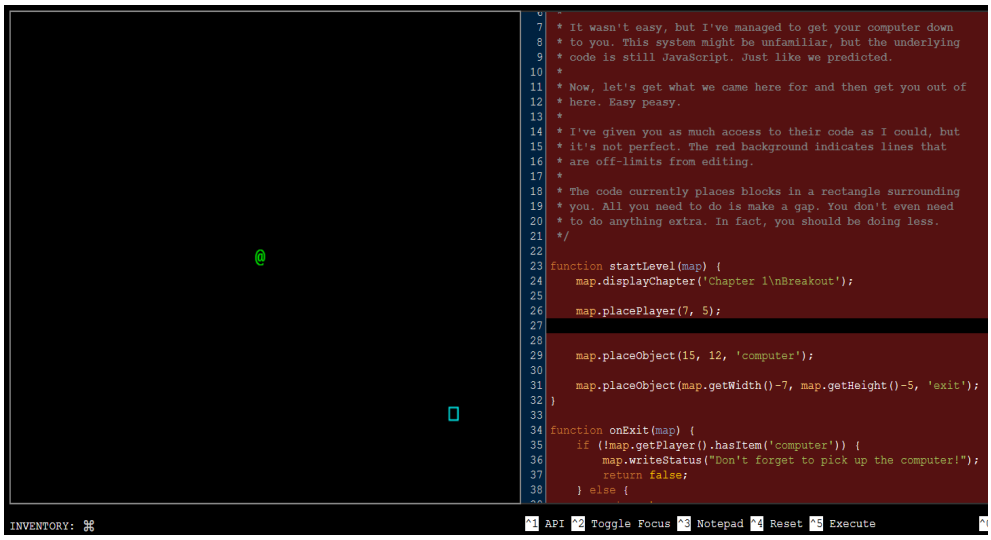


Figura 2.5: Nivel inicial Untrusted

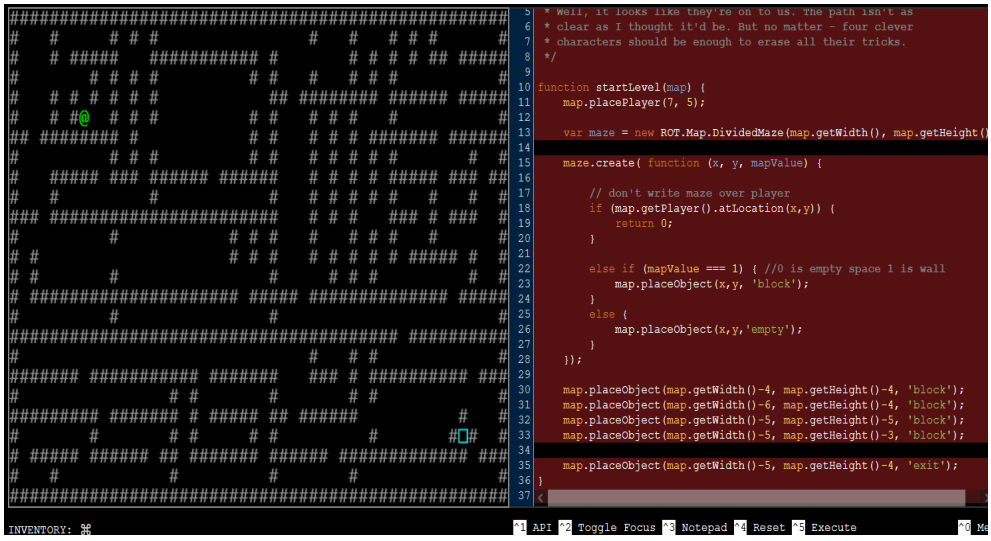


Figura 2.6: Nivel 2 Untrusted

2.1.2.3. SQL

Existen juegos que buscan enseñar las bases del lenguaje SQL, que es un lenguaje de programación empleado para realizar operaciones sobre bases de datos. Estos juegos buscan enseñar conceptos como las operaciones básicas sobre bases de datos: lecturas, inserciones, modificaciones, claves primarias y foráneas, etc.

Un ejemplo es SQL Muder Mistery [11] (vea la figura 2.7), donde se ayuda a un detective a resolver los misterios de un crimen, lo que se consigue a través de los conoci-

mientos que aporta el juego. Está orientado a principiantes del lenguaje SQL, ya que hay una documentación aplicada a los conceptos necesarios para superar el juego. El problema del juego es que no dispone de muchos elementos de gamificación y requiere que los jugadores se lean una documentación para poder ir avanzando en el juego.

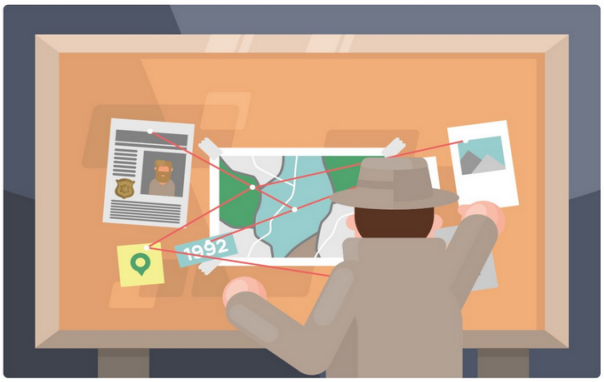


Figura 2.7: SQL Murder Mistery

2.1.2.4. C

El lenguaje C es considerado como un lenguaje base para el resto de lenguajes (Java, C#, C++, Python, etc). Su uso actual está más orientado a la programación de código a bajo nivel, por lo tanto, casi no se han encontrado juegos serios cuyo objetivo sea enseñar la programación en C, al ser un lenguaje orientado más al uso profesional.

Por esta misma razón, en la UPV se diseñó un juego, sobre el que está basado el juego desarrollado en este TFG, que enseña las bases de la programación en C. Este juego se puede encontrar en un enlace del servidor de personales UPV [13], el cual ha servido como inspiración para el desarrollo de este juego, cuyo objetivo en enseñar las bases de programación en Python, un lenguaje con pocas herramientas gamificadas de aprendizaje.

Figura 2.8: Nivel 1 del juego del robot para C

El juego consta de 10 niveles, cada uno más difícil que el anterior, y el objetivo es conseguir que el robot quede en la casilla que se encuentra debajo de la puerta. De manera progresiva se introducen nuevas instrucciones, los bucles, condicionales, etc.

2.1.2.5. CSS

El lenguaje CSS (Cascading Style Sheets) es el lenguaje base de estilos empleado para la presentación de los documentos HTML. Por la importancia del desarrollo web hoy en día, muchas personas quieren aprender a programar con el lenguaje CSS.

Un ejemplo de juego diseñado para el aprendizaje de CSS es **CSS Diner** [12], en el cual se enseña CSS de manera bastante sencilla. El juego dispone de 32 niveles, cada nivel más díficil que el anterior. Se muestran diferentes elementos encima de una mesa, y un código sencillo HTML que referencia cada elemento. El desafío que presenta cada nivel es ir buscando la manera de aplicar un estilo mediante CSS a cada elemento. En la figura 2.9 se ve que el objetivo del nivel 4 es aplicar un estilo a un elemento dentro de otro (objeto dentro del plato).

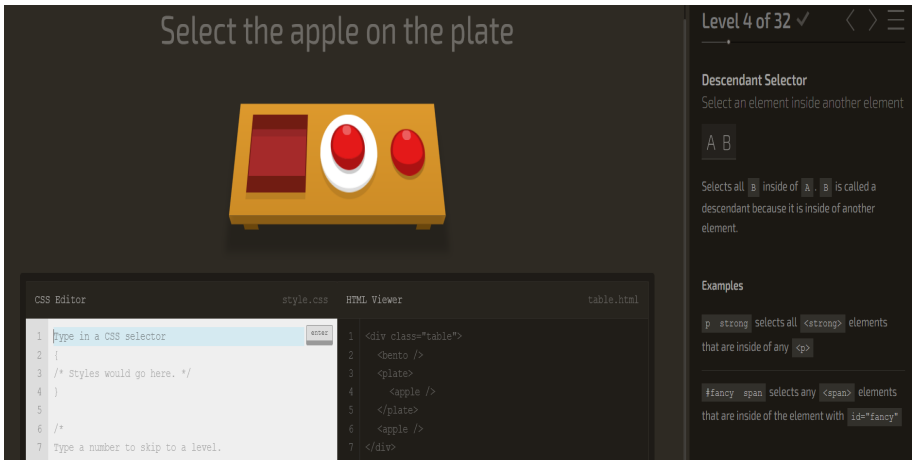


Figura 2.9: CSS Diner nivel 4

2.1.2.6. Python

En cuanto al lenguaje Python, que es el que realmente nos interesa para este trabajo, se han encontrado muy pocos juegos serios orientados al aprendizaje de las bases de Python. Esto y el auge del lenguaje han sido las razones de peso que se han tenido en cuenta para la realización de este trabajo.

Un juego que se ha encontrado es **CodeMonkey**, que es un juego orientado a estudiantes de 6 a 14 años, por lo tanto se empieza desde el mínimo conocimiento de programación. Realmente el juego no enseña las bases de Python, si no que enseña conceptos básicos de programación general. En CodeMonkey, como se puede ver en la figura 2.10, las instrucciones y la sintaxis no pueden realmente asociarse con Python, por lo que es una muy buena herramienta para empezar a programar y a desarrollar el pensamiento computacional.

Figura 2.10: Interfaz del nivel 5 en CodeMonkey

El juego se basa en superar todos los niveles, donde el objetivo es que el mono consiga el plátano. Cada nivel introduce información nueva (instrucciones, herramientas, obstáculos, etc). Como se puede observar en la figura 2.10, la interfaz y la jugabilidad están diseñadas para alumnos sin experiencia, por eso este juego tiene mucha popularidad en las clases, dando opción a crear grupos de jugadores en las aulas. A diferencia de CodeMonkey, el juego del robot que se ha diseñado para este TFG sigue al pie de la letra las restricciones de la sintaxis Python.

2.2 Lenguaje Python

Python es un lenguaje de programación de alto nivel que se utiliza para multitud de aplicaciones de todo tipo. Se trata de un lenguaje interpretado, es decir, no es necesario compilarlo para ejecutar las aplicaciones Python, sino que se ejecutan directamente (a través de un intérprete), sin necesidad de traducirlo a lenguaje máquina para que el ordenador pueda procesarlo.

Como se expuso en la figura 2.1, el lenguaje Python está en constante crecimiento sobre su número de usuarios. Actualmente, Python es uno de los lenguajes de programación más demandados en todo el mundo, habiendo aumentado más de un 49 % la demanda de programadores Python con respecto a 2019.

Las razones que explican este crecimiento son las siguientes:

- Es un lenguaje **sencillo**, con gran similitud con el lenguaje humano, de forma que es fácil de entender, aprender y escribir.

- Python es multiplatafoma (compatible con varios sistemas operativos) y de código abierto (open source), lo que permite un desarrollo de software ilimitado para los usuarios.

- Es un lenguaje **interpretado**, no compilado.

- Es un lenguaje **estructurado**, y por lo tanto mucho más legible que otros lenguajes no estructurados. En Python es muy importante la **identación** (sangría), ya que mediante las tabulaciones se estructuran los bloques de código ->funciones, condicionales, bucles, etc. (vea la figura 2.11).

```
import random

def buscarElemento(lista, elemento):
    for i in range(0,len(lista)):
        if(lista[i] == elemento):
            return i

def imprimirLista(lista,nombre):
    for i in range(0,len(lista)):
        print nombre + "[" + str(i) + "]=" + str(lista[i])

def leerLista():
    lista=[]

    i=0
    while i < 10:
        lista.append(int(random.randint(0, 10)))
        i=i+1
    return lista
```

Figura 2.11: Ejemplo de código identado Python

2.2.1. Usos del lenguaje Python

Python es usado en varios sectores. En la figura 2.12 se ve como se distribuye el uso de Python según una encuesta realizada por los usuarios de la plataforma StackOverflow en 2020 [3]:

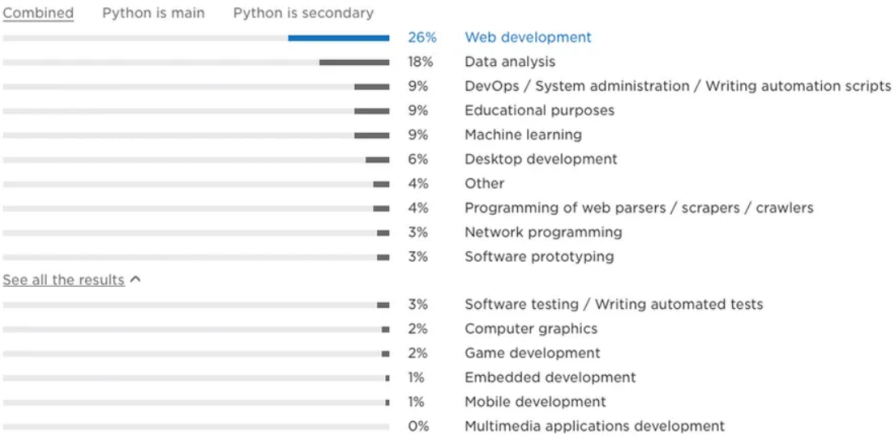


Figura 2.12: Usos de Python en 2020 según StackOverflow

1. **Análisis de datos y Big Data:** Python es un lenguaje muy potente, por lo que es bastante empleado para el tratamiento de grandes cantidades de datos. Python dispone de una gran cantidad de bibliotecas, y al ser tan simple y permitir desarrollar soluciones complejas en pocas líneas de código, es el lenguaje ideal para trabajar con los análisis de datos y Big Data. Entre otras grandes empresas, Amazon es una de las que utiliza Python para Big Data (vea la figura 2.13).

Figura 2.13: Amazon usa Python para Big Data

2. **Data mining:** La minería de datos es el proceso de predecir información mediante patrones a través del análisis de enormes bases de datos. El Data mining al fin y al cabo es un análisis de datos, por lo que como se ha comentado antes para el análisis de datos general, Python es un lenguaje muy apropiado y preparado para el tratamiento y estudio de grandes cantidades de datos, por su simplicidad y cercanía al lenguaje humano.

3. **Inteligencia artificial / Machine Learning:** El Machine Learning o aprendizaje automático es un método de análisis de datos mediante el cual los sistemas pueden aprender datos e identificar patrones para poder funcionar normalmente sin intervención humana. De nuevo, la sencillez y el fácil aprendizaje de Python hacen que sea el lenguaje más empleado para Machine Learning [4].

4. **Juegos y gráficos 3D:** Pese a no ser el uso principal de Python, el lenguaje dispone de herramientas para el desarrollo de videojuegos. Es un buen lenguaje para la programación de videojuegos, ya que permite programar cosas complejas de manera sencilla, pero el problema que tiene es que Python es un lenguaje interpretado, no compilado, lo cual hace que los juegos pierdan rendimiento. La librería más famosa de Python para los juegos es Pygame. Se conocen más de 1000 proyectos registrados usando Pygame [5]. Algunos ejemplos son Space Way y Urban Champion remake.

5. **Desarrollo web**: Es el uso principal de Python actualmente (como se puede observar en la figura 2.12). Estas páginas se caracterizan por tener menos líneas de código y ser más ligeras y optimizadas. Normalmente se utiliza el framework Django (ver figura 2.14). Algunos ejemplos del uso de Django en el desarrollo web son Instagram, National Geographic, Pinterest, etc.

Figura 2.14: Framework Django para desarrollo web en Python

CAPÍTULO 3
Requisitos del juego

Para el diseño y la implementación de este juego, el procedimiento que se ha seguido viene de unos requisitos. El juego tiene que cumplir una serie de funcionalidades y enseñar unos conceptos necesarios para superar los niveles. En este capítulo, previo a la explicación del Diseño y de la Implementación, se van a explicar estos requisitos.

3.1 Conceptos de Python que aporta el trabajo

En esta sección se presentan las estructuras del lenguaje Python que son consideradas en este trabajo. La manera en la que se tratan estas estructuras y los niveles se verán más adelante en el capítulo 4, donde se presenta la propuesta de juego.

El objetivo de este trabajo es introducir las bases de la programación en lenguaje Python a los nuevos alumnos, por lo que los conceptos que aporta este trabajo son sencillos, pero ya requieren un desarrollo del pensamiento lógico. De hecho, como se ve en la figura 3.1, según la organización IEEE (Institute of Electrical and Electronics Engineers) [30], el orden recomendado para aprender a programar de manera continua es la siguiente:

Figura 3.1: Pasos de aprendizaje en programación

En este trabajo, se pretende enseñar de manera muy sencilla el paso 1 (tipos numéricos y expresiones), el paso 5 (condicionales), el paso 6 (bucles) y el paso 8 (funciones).

1. **Tipos numéricos y expresiones:** Los tipos numéricos, como indica su nombre, representan a los números, y por ello se puede operar matemáticamente con ellos (suma, resta, producto, etc). En programación, estos tipos numéricos se representan a través de bits. Para definir la cantidad de bits que se van a usar para representar los números, se crearon diferentes tipos.

Los conceptos que aporta este trabajo sobre los tipos numéricos son únicamente su uso como parámetros para llamar a funciones, sin asignar un tipo ni definir una variable, simplemente pasar el valor directamente a las funciones.

2. **Condicionales:** Los condicionales en programación son estructuras que provocan que el flujo de código se divida en dos. Si la condición se cumple, el código sigue un camino, mientras que si no se cumple seguirá otro camino.

   En el nivel 9 del juego, se introduce el concepto de condicional para los alumnos, de forma que si una condición booleana se cumple el robot hará unos movimientos, mientras que si no se cumple hará otros movimientos. Esto se ha implementado a través de la estructura *si(orientación_robot):*, donde la orientación del robot hará que se ejecuten unas instrucciones u otras. Se explica más adelante, en los capítulos 4 y 5, como se ha implementado esta funcionalidad.

3. **Bucles:** Los bucles son secuencias de instrucciones de código que se repiten un número determinado de veces. Ese número de veces va determinado por una o varias condiciones, de tal forma que se ejecutará la secuencia de instrucciones hasta que la guarda (conjunto de condiciones) del bucle deje de satisfacerse.

   Respecto a los bucles en este trabajo, se muestra la sintaxis de los bucles en Python, aunque de una manera bastante intuitiva. La instrucción repite(x):"sirve como estamento de inicio de un bucle, de forma que el parámetro x es el número de veces que se ejecuta la secuencia dentro del bucle.

4. **Funciones:** Una función en programación es un bloque de código diseñado para realizar ciertas acciones de manera independiente al resto de código, aunque puede llamar a otras funciones para completar esas acciones. Las funciones pueden tener opcionalmente parámetros de entrada, que son usados en la misma función, y un valor de retorno, que suele ser usado en otras funciones.

   La enseñanza de funciones en este juego es bastante breve. Se enseña el concepto de función, de forma que cuando los alumnos escriban en el panel una función que sea compilable para el robot, el robot ejecutará unas acciones determinadas. El concepto de parámetro de entrada también se introduce ya que las funciones, como se ha explicado antes, necesitarán un valor en el parámetro para indicar el número de movimientos, iteraciones, etc.

5. **Testing de código:** Aunque los tests de código no aparecen en la imagen 3.1, es un concepto muy importante que los alumnos deben empezar a conocer desde el principio por su importancia.

   Los tests unitarios en Python se pueden hacer de manera manual o automática.

   - Los tests manuales son pruebas que hace el usuario a mano generando distintas combinaciones para probar una funcionalidad.

   - Los tests automáticos se tratan de código que valida otro código y que pueden cubrir una mayor parte de la funcionalidad. Estos son los tests que realmente nos interesan y que se han implementado en el juego. Para ello, en Python se usa la instrucción assert, tal y como se ve en la figura 3.2.

```
assert(calcula_media([3, 7, 5]) == 5.0)
assert(calcula_media([30, 0]) == 15.0)
```

Figura 3.2: Ejemplo de la instrucción assert en Python

En este juego, se ha implementado una manera sencilla de realizar tests unitarios sobre la ejecución de las instrucciones disponibles en el tablero. Se explicará con más detalle el funcionamiento en el capítulo 5.

3.2 Requisitos específicos del juego

En esta sección se explican los requisitos que son específicos de este juego. Estos requisitos son:

1. Entender que hay que traducir los programas para que el ordenador los pueda comprender.
2. Diferenciar entre Escribir, Traducir y Ejecutar un programa.
3. Asimilar la sintaxis de Python: bloques de código, identación, comentarios, ausencia de puntos y comas, etc.
4. Comprender el concepto de testing de código.
5. Mantener el juego dejando el código Open Source, que permite seguir aumentando las funciones del juego en un futuro.

CAPÍTULO 4
Diseño del juego

A lo largo de este capítulo, se explica la propuesta de juego que se ha desarrollado. Se va a empezar por las herramientas que se han usado y el diseño del juego, y por último se explicará como interaccionan los elementos del juego en la Arquitectura del juego.

El objetivo del juego es lograr que el robot avance a lo largo del tablero llegando a la casilla final esquivando los obstáculos que impiden al robot avanzar en la mayoría de ocasiones.

4.1 Diseño de la Interfaz Gráfica

El juego se ha diseñado usando HTML5 para la estructura del contenido, y luego para el estilo de la interfaz se ha alternado entre CSS y Bootstrap.

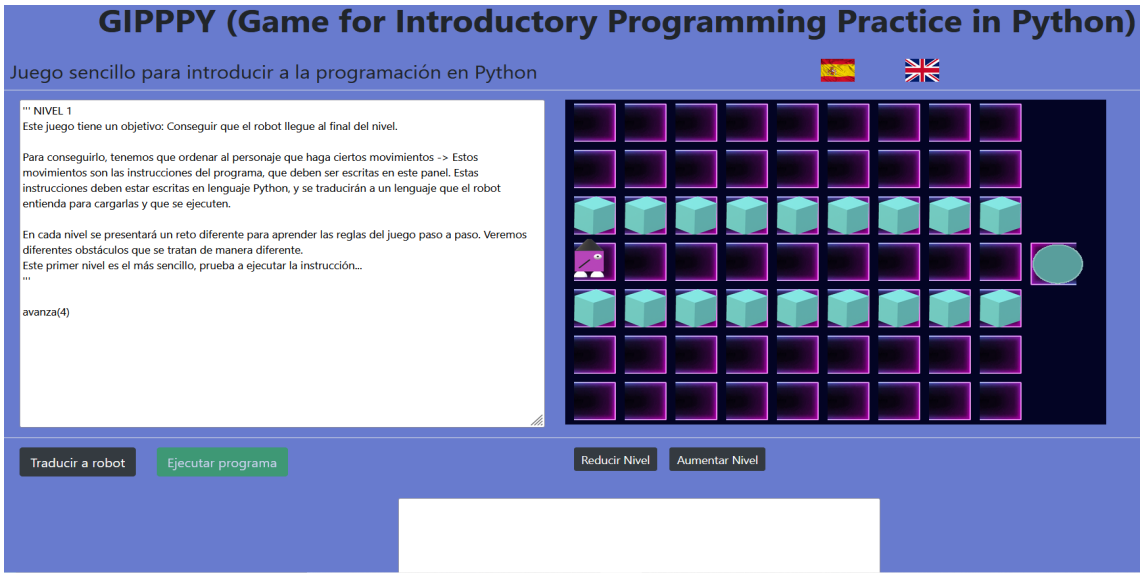


Figura 4.1: Interfaz Principal del juego

En la interfaz principal que se muestra en la figura 4.1, vemos los siguientes elementos html:

1. **Cuerpo:** El body HTML es el nodo padre de la estructura del contenido de un archivo HTML. A partir de él salen todos los subnodos que forman el diseño. Vemos en la figura 4.2 el estilo CSS que da color al fondo, a parte de la llamada a la función cargar_nivel(), cuando suceda el evento on_load, es decir, cuando se cargue

la página. La lógica de las funciones se explicará más adelante en el capítulo de Implementación.

```
<body style="background-color:rgb(104, 123, 206) " onload= "cargar_nivel(1,true)">
    <h1 class="text-center font-weight-bold">GIPPPY (Game for Introductory Programming Practice in Python)</h1>
```

Figura 4.2: Body y h1 del HTML

2. **<h1>y <h4>:** El h1 es una etiqueta de HTML para destacar cierto contenido en la interfaz, por ello el contenido GIPPPY (Game For Introductory Programming Practice in Python) sirve como título para el juego. Se ve en la figura 4.2 que el h1 tiene una clase de estilo mediante Bootstrap, simplemente para situarlo en el centro de la página y en negrita. La etiqueta h4 en este caso expone un contenido menos prioritario, que es una breve descripción del juego ("Juego Sencillo para introducir a la programación con lenguaje Python").

```
<table>
    <tr>
        <td>
            <h4 id="h4_inicio" style="margin-left: 40px; margin-top: 20px;">Juego sencillo para introducir a la programación con lenguaje Python</h4>
        </td>
        <td>
            <input type="image" src="bandera_spn.gif" style="margin-left: 300px; margin-top: 10px; width: 40px; height: 30px;" onclick="cambiarIdioma('esp')">
        </td>
        <td>
            <input type="image" src="bandera_eng.webp" style="margin-left: 30px; margin-top: 10px; width: 40px; height: 30px;" onclick="cambiarIdioma('eng')">
        </td>
    </tr>
</table>
```

Figura 4.3: Tabla h4 e inputs

3. **Inputs tipo imagen**: Las banderas de España e Inglaterra se han creado como inputs de tipo imagen, de forma que se puede hacer click sobre ellas para cambiar el idioma del juego. Se puede ver en la figura 4.3 que tanto el h4 como estos inputs van dentro de un elemento table para dar estructura al contenido. Se ve también que al activar el evento on click en los inputs se llama a las funciones del cambio de idioma, que se explicarán más adelante.

4. **Panel de código**: El panel de código es un elemento <textarea>de HTML modificable por los jugadores. Tiene cargados unos valores por defecto para cada nivel, aunque de eso se ocupa la parte lógica en Javascript.

   El contenido del textarea debe estar en lenguaje Python. Como se ve en la figura 4.1, hay gran parte del contenido que corresponde con un comentario multilínea en Python, y fuera del comentario hay una instrucción que será compilada.

   El panel también soporta los comentarios simples de Python, que se inician con el carácter "#", y ocupan una única línea.

5. **Escenario**: El escenario corresponde con el tablero donde están todos los elementos que dan lugar al juego: el robot, las baldosas, los obstáculos y la plataforma final. El escenario está diseñado mediante un elemento de HTML con imágenes dentro, situadas y ordenadas en una determinada manera para dar formato al tablero. En la figura 4.4, podemos ver parte del lenguaje html que da forma al escenario, ya que no se ven todas las imágenes que lo conforman.

```
<div id="escenario" style= "position:absolute; width:640px; height: 420px; z-index: 0; background-color:rgb(3, 4, 36);">

    <img id="robot" src = "robot_lateral.gif" style="position:absolute; left: 10px; top: 180px; z-index: 3; width: 40px; height: 55px;">
    <img src = "suelo_inicio.png" style="position:absolute; left: 10px; top: 5px; z-index: 0; width: 50px; height: 50px;">
    <img src = "suelo_inicio.png" style="position:absolute; left: 10px; top: 65px; z-index: 0; width: 50px; height: 50px;">
    <img src = "suelo_inicio.png" style="position:absolute; left: 10px; top: 125px; z-index: 0; width: 50px; height: 50px;">
    <img src = "suelo_inicio.png" style="position:absolute; left: 10px; top: 185px; z-index: 0; width: 50px; height: 50px;">
    <img src = "suelo_inicio.png" style="position:absolute; left: 10px; top: 245px; z-index: 0; width: 50px; height: 50px;">
    <img src = "suelo_inicio.png" style="position:absolute; left: 10px; top: 305px; z-index: 0; width: 50px; height: 50px;">
    <img src = "suelo_inicio.png" style="position:absolute; left: 10px; top: 365px; z-index: 0; width: 50px; height: 50px;">

    <img src = "suelo_inicio.png" style="position:absolute; left: 70px; top: 5px; z-index: 0; width: 50px; height: 50px;">
    <img src = "suelo_inicio.png" style="position:absolute; left: 70px; top: 65px; z-index: 0; width: 50px; height: 50px;">
    <img src = "suelo_inicio.png" style="position:absolute; left: 70px; top: 125px; z-index: 0; width: 50px; height: 50px;">
    <img src = "suelo_inicio.png" style="position:absolute; left: 70px; top: 185px; z-index: 0; width: 50px; height: 50px;">
    <img src = "suelo_inicio.png" style="position:absolute; left: 70px; top: 245px; z-index: 0; width: 50px; height: 50px;">
    <img src = "suelo_inicio.png" style="position:absolute; left: 70px; top: 305px; z-index: 0; width: 50px; height: 50px;">
    <img src = "suelo_inicio.png" style="position:absolute; left: 70px; top: 365px; z-index: 0; width: 50px; height: 50px;">

    <img src = "suelo_inicio.png" style="position:absolute; left: 130px; top: 5px; z-index: 0; width: 50px; height: 50px;">
    <img src = "suelo_inicio.png" style="position:absolute; left: 130px; top: 65px; z-index: 0; width: 50px; height: 50px;">
    <img src = "suelo_inicio.png" style="position:absolute; left: 130px; top: 125px; z-index: 0; width: 50px; height: 50px;">
    <img src = "suelo_inicio.png" style="position:absolute; left: 130px; top: 185px; z-index: 0; width: 50px; height: 50px;">
    <img src = "suelo_inicio.png" style="position:absolute; left: 130px; top: 245px; z-index: 0; width: 50px; height: 50px;">
    <img src = "suelo_inicio.png" style="position:absolute; left: 130px; top: 305px; z-index: 0; width: 50px; height: 50px;">
    <img src = "suelo_inicio.png" style="position:absolute; left: 130px; top: 365px; z-index: 0; width: 50px; height: 50px;">

    <img src = "suelo_inicio.png" style="position:absolute; left: 190px; top: 5px; z-index: 0; width: 50px; height: 50px;">
    <img src = "suelo_inicio.png" style="position:absolute; left: 190px; top: 65px; z-index: 0; width: 50px; height: 50px;">
    <img src = "suelo_inicio.png" style="position:absolute; left: 190px; top: 125px; z-index: 0; width: 50px; height: 50px;">
    <img src = "suelo_inicio.png" style="position:absolute; left: 190px; top: 185px; z-index: 0; width: 50px; height: 50px;">
    <img src = "suelo_inicio.png" style="position:absolute; left: 190px; top: 245px; z-index: 0; width: 50px; height: 50px;">
    <img src = "suelo_inicio.png" style="position:absolute; left: 190px; top: 305px; z-index: 0; width: 50px; height: 50px;">
    <img src = "suelo_inicio.png" style="position:absolute; left: 190px; top: 365px; z-index: 0; width: 50px; height: 50px;">
```

Figura 4.4: Diseño del escenario de juego

Lo más importante de las imágenes es el campo style, ya que mediante CSS le estamos asignando a cada elemento la posición (absolute, left, y top), el tamaño (width y height) y la profundidad (z-index).

6. **Botones:** Los botones del juego controlan el funcionamiento de los niveles. El estilo de los botones se ha añadido mediante Boostrap, como se puede ver en la figura 4.5.

```
<td>
    <input id="traducirprog" class="btn btn-dark" type="button" style="margin-left: 40px;" value="Traducir a robot" onclick="traducirCodigo()">
    <input id="ejecutarprog" class="btn btn-success" type="button" style="margin-left: 20px;" disabled value="Ejecutar programa" onclick="ejecutarCodigo()">
</td>
<td>
    <input class="btn btn-dark btn-sm" id="rnivel" type="button" style="margin-left: 10px;" value="Reducir Nivel" onclick="cambio_nivel(0)">
    <input class="btn btn-dark btn-sm" id="anivel" type="button" style="margin-left: 10px;" value="Aumentar Nivel" onclick="cambio_nivel(1)">
</td>
```

Figura 4.5: Botones HTML

Los botones *Traducir a robot* y *Ejecutar programa* son complementarios. El primero de ellos comprueba la validez del contenido del panel de código, y en caso de ser correcto, activa el botón *Ejecutar programa*, el cual provoca los movimientos del robot. Esta activación y desactivación del botón se hace mediante Javascript, pero por defecto está desactivado mediante la propiedad **disabled** (ver figura 4.5).

Respecto a los botones de *Reducir Nivel* y *Aumentar Nivel*, su uso es bastante fácil de entender. Cuando se hace click en *Reducir Nivel*, se carga el nivel anterior, y al revés con *Aumentar nivel*. Si se intenta reducir el nivel en el nivel 1 o si se intenta aumentar el nivel estando en el nivel 12, se muestran las alertas de error que se ven en las figuras 4.6 y 4.7.

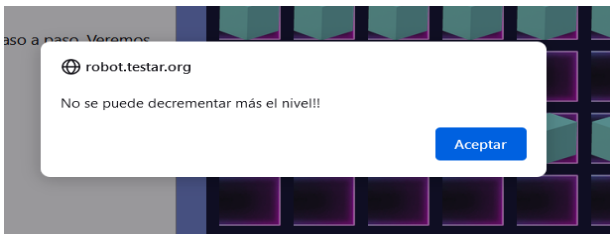


Figura 4.6: Alerta cuando se decrementa nivel 1

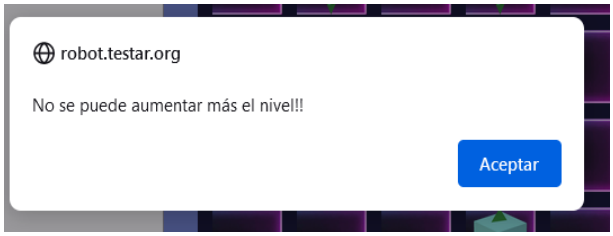


Figura 4.7: Alerta cuando se aumenta nivel 12

7. **Consola de salida:** La consola es un elemento textarea de HTML de sólo lectura. Esto se ha conseguido mediante el parámetro readonly (ver figura 4.8). No se permite la escritura ya que la consola sirve para informar al usuario de los estados del juego: compilación correcta, errores de compilación, colisión con pinchos, final del nivel, etc.

```
<textarea readonly id="output" style="position:relative; left: 500px;" rows="8" cols="60"></textarea>
```

Figura 4.8: Consola HTML

Un ejemplo de interacción del juego con la consola se puede ver en las figuras 4.13 (donde se ve una colisión con pinchos) y 4.30 (donde el robot cae al vacío y a parte se muestra el resultado de un testcase).

4.1.1. Nivel 1

El nivel 1 se ha diseñado como introducción al juego. En el panel de código se ve bastante texto, ya que se explican las mecánicas básicas del juego y el objetivo de todos los niveles: conseguir que el robot llegue a la plataforma final, que corresponde con la casilla que contiene el círculo.

Como se ha dicho antes, este nivel es introductorio, por lo que se ve en la figura 4.9 que el camino que lleva al robot al final del nivel es trivial, simplemente con una instrucción se puede conseguir el objetivo. Como ayuda, se propone a los jugadores que ejecuten directamente la instrucción **avanza(4)**, para verificar que el robot va a avanzar el número de casillas que se le indiquen como parámetro a la instrucción avanza(x). De esta forma se introduce el concepto de llamada a función con Python.

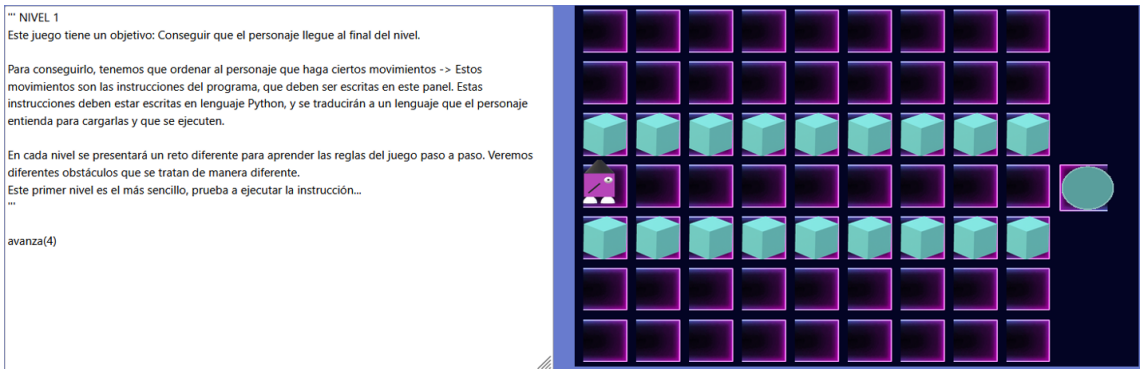


Figura 4.9: Nivel 1

Los obstáculos presentes en el nivel no influyen de momento, ya que no se ha presentado la instrucción gira(x), de forma que es imposible que el robot colisione con los obstáculos. Se explicará su funcionamiento en el nivel 2.

En este nivel, se pretende introducir el concepto de instrucción y de llamada a función con un parámetro. Los jugadores van a observar que, si siguen el formato función(parámetro) de manera correcta, se traduce esto a un lenguaje que entiende el robot y que eso tiene una consecuencia (que el robot avance el número de casillas indicadas como parámetro).

4.1.2. Nivel 2

En el segundo nivel, ya se complican las cosas. Hay menos cantidad de obstáculos que en el nivel 1, pero en este caso, hay un obstáculo en forma de cubo (iguales que los del nivel 1) que impide avanzar en línea recta hacia la meta final del nivel (ver figura 4.10). Por lo tanto, el objetivo de este nivel, es introducir la nueva instrucción gira(x), donde x es el número de giros que va a dar el robot. El sentido de los giros viene dado dependiendo de si x es positivo o negativo:

Si x >0 ->robot gira en sentido contrario a las agujas del reloj.
Si x <0 ->robot gira en sentido de las agujas del reloj.

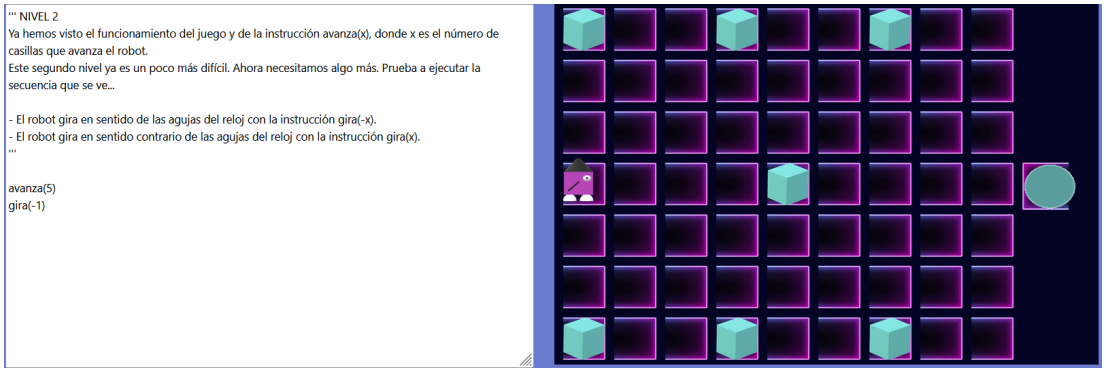


Figura 4.10: Nivel 2

Respecto al obstáculo en forma de cubo, es un obstáculo sencillo, cuyo funcionamiento es impedir que el robot avance a través de él. Si el robot colisiona con este obstáculo, se seguirán ejecutando las instrucciones restantes, pero nunca atravesando el obstáculo sin rodearlo. Podemos ver en la figura 4.11 el resultado de ejecutar la secuencia de instrucciones que se ofrece a los jugadores (avanza(5), gira(-1)), donde el robot ha chocado con el obstáculo pero aún así ha realizado el giro.

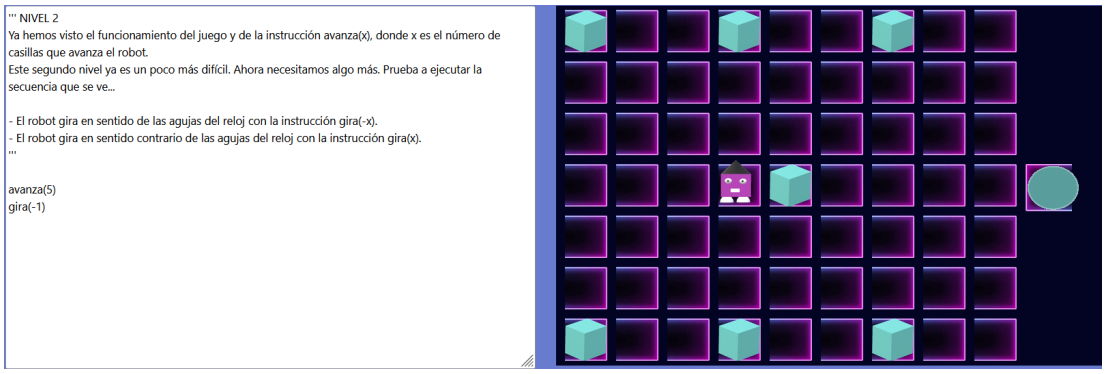


Figura 4.11: Resultado de ejecutar el Nivel 2

El concepto que se pretende que transmita este nivel es la existencia de más funciones que puede haber implementadas en un proyecto, y que pueden seguir la misma sinta-

xis que otras instrucciones. También los alumnos pueden darse cuenta del concepto de secuencia, de forma que las instrucciones se ejecutan en el orden de escritura, una a una.

4.1.3. Nivel 3

El nivel 3, como se puede ver en la figura 4.12, es igual que el nivel 2 respecto a la colocación de los obstáculos, pero la diferencia principal se encuentra en el obstáculo central que impide al robot llegar al final de forma sencilla. En este nivel 3, se introduce el obstáculo con pinchos, el cuál tiene una función bastante restrictiva. Si el robot colisiona con el obstáculo con pinchos, la ejecución del nivel se termina.

Figura 4.12: Nivel 3

El objetivo del nivel es obligar a los jugadores a introducir el número exacto de casillas que el robot debe avanzar para esquivar el obstáculo, es decir, los parámetros de las instrucciones. Si el obstáculo fuera sencillo como en el caso del nivel 2, el nivel se puede superar con las instrucciones correctas, pero sin necesidad de que los parámetros sean exactos.

En la figura 4.13, podemos ver la ejecución por defecto del nivel 3, donde el robot colisiona con el obstáculo con pinchos, y además, se muestra por la consola una alerta a los jugadores para avisar sobre el problema de colisionar con este obstáculo.

Figura 4.13: Ejecución nivel 3 por defecto

Figura 4.14: Ejecución nivel 3, secuencia de instrucciones cancelada

Lo más importante que se deduce en este nivel es que una secuencia de instrucciones, bajo unas condiciones determinadas, puede cancelarse y no terminar. En este caso, la colisión con el obstáculo de pinchos provoca que el robot no siga ejecutando nada más de las órdenes que hay en la cola (ver figura 4.14).

4.1.4. Nivel 4

En el nivel 4 se va a introducir algo que se va a encontrar en muchos niveles a lo largo del juego: la aleatoriedad de la posición de los obstáculos.

Respecto al tipo de obstáculos, no se van a introducir obstáculos nuevos, pero se complica a partir de ahora el planteamiento de una solución para el nivel, ya que, al añadir las posiciones aleatorias, cada vez que carguemos el nivel 4, la solución va a ser diferente.

En este caso, únicamente la primera fila de obstáculos es aleatoria, en concreto el hueco por el que el robot debe pasar para alcanzar el final del nivel. En las figuras 4.15 y 4.16 se ven dos posibles estructuras del nivel, pero en total hay siete diferentes estructuras dependiendo del número aleatorio generado.

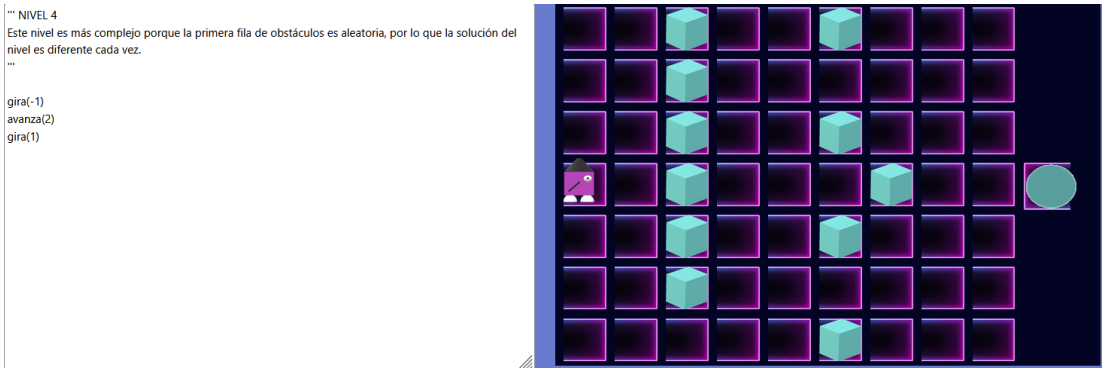


Figura 4.15: Nivel 4 hueco aleatorio primer ejemplo

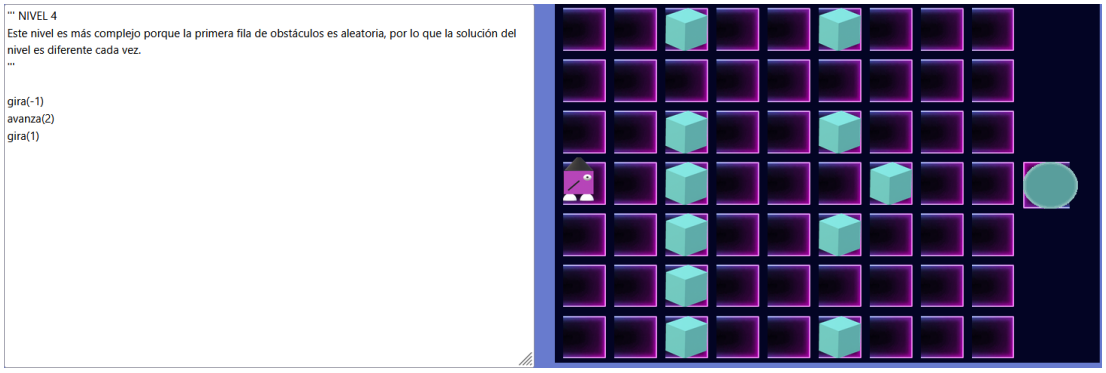


Figura 4.16: Nivel 4 hueco aleatorio segundo ejemplo

Tampoco se introduce en este nivel nada nuevo respecto a la ejecución de instrucciones, se da como ayuda a los jugadores algunas instrucciones que pueden servir para comenzar (dependiendo de la posición aleatoria). Realmente, si un alumno supera este nivel en más de 1 ocasión, es porque va teniendo clara la mecánica del juego y ya comienza poco a poco un desarrollo del pensamiento lógico.

4.1.5. Nivel 5

En el nivel 4 se introdujeron los niveles aleatorios, así que el nivel 5 presenta también algo de aleatoriedad. Para no complicar mucho el nivel todavía, solamente se ha hecho aleatoria la posición del último obstáculo con pinchos, ya que los jugadores ya se enfrentan por primera vez a un nivel con gran cantidad de obstáculos con pinchos (en este caso, todos tienen pinchos), los cuales ya dificultan bastante la resolución.

Figura 4.17: Nivel 5 obstáculo aleatorio primera posibilidad

Figura 4.18: Nivel 5 obstáculo aleatorio segunda posibilidad

Figura 4.19: Nivel 5 obstáculo aleatorio tercera posibilidad

Como se ve en las figuras 4.17, 4.18 y 4.19, las tres posibilidades para la posición del obstáculo se han pensado para no coincidir con las posiciones de la fila de obstáculos anterior.

La intención en este nivel es dificultar el juego un poco más, siendo ya cada vez más complicado superar los niveles. El aumento de dificultad provoca que los alumnos quieran seguir con el juego, ya que supone la solución para los niveles posteriores suponen un desafío mayor. Esto repercute en su aprendizaje, pues mientras más tiempo pasen jugando, más claro serán los conceptos de Python que están aplicando.

4.1.6. Nivel 6

El nivel 6 presenta un reto diferente. Hasta ahora, se podían obtener bastantes soluciones para cada nivel, ya que los obstáculos restringían el camino directo, pero aún así, dan opción a ser esquivados de muchas maneras diferentes.

En el nivel 6 solo existen 2 soluciones directas (sin que el robot camine hacia atrás). Como se puede ver en la figura 4.20, los caminos pasan por evitar que el robot entre dentro del 'callejón sin salida'. Se puede pasar por encima o por debajo de los obstáculos, pero solo hay 2 secuencias de instrucciones que son correctas para superar el nivel.

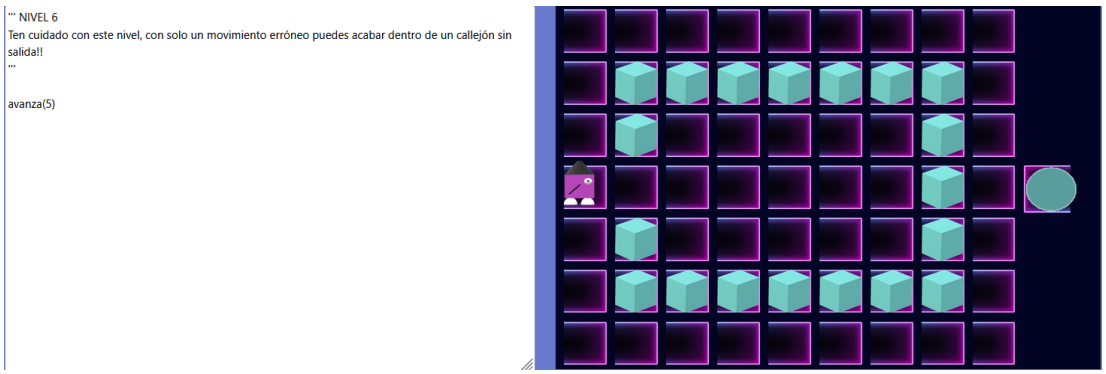


Figura 4.20: Nivel 6

Este nivel no aporta conocimientos extra a los alumnos respecto de las instrucciones, sino que les permite identificar que para la resolución de cualquier problema existen muy pocas soluciones eficientes y muchas soluciones ineficientes. Se puede hacer que el robot entre al callejón sin salida y dé la vuelta, pero se sobreentiende que eso es un malgasto de instrucciones, por lo que estas soluciones son ineficientes. Para afianzar ese conocimiento, se ha implementado un análisis para las soluciones de los usuarios, de

forma que la consola les avisa si han superado el nivel de manera eficiente o no. Esto se explicará con más detalle en el apartado 5.

4.1.7. Nivel 7

A estas alturas, se espera que los jugadores ya tengan experiencia con las secuencias de instrucciones usando avanza(x) y gira(x), así que en este nivel 7 se introduce la primera estructura implementada en el juego: los bucles.

Figura 4.21: Nivel 7

Se puede ver en la figura 4.21 que el nivel tiene bastantes obstáculos, algunos de ellos con pinchos, así que es un nivel con mayor dificultad. Las instrucciones que se ofrecen de ayuda al jugador en el nivel 7 constan de una estructura repite(x), donde se ejecutan 4 veces las instrucciones que están identadas correctamente (en este caso, tanto avanza(2) como gira(1) se van a repetir).

Tras la ejecución del código, el robot vuelve a la posición inicial del nivel, de forma que los alumnos puedan entender el funcionamiento del repite(x). Por lo tanto, se espera que en este nivel se entienda como funcionan los bloques de código en Python. En este caso, el bloque lo inicia una estructura iterativa, seguida del carácter ':', que indica el inicio del bloque. A partir del carácter ':', todo lo que vaya a formar parte del bloque de código debe dejar un espaciado obligatorio. En este juego, se ha respetado la identación de Python [14], siendo necesario dejar 2 espacios exactamente para que el texto compile de manera correcta.

4.1.8. Nivel 8

El nivel 8 es el primer nivel completamente aleatorio. En cuanto a dificultad, es uno de los niveles más difíciles de resolver, ya que hay que calcular con mucho cuidado los pasos que el robot debe dar para llegar al final sin inconvenientes.

En las figuras 4.22 y 4.23, se muestran 2 posibles diseños del nivel 8. El nivel puede ser más o menos complicado dependiendo de los números aleatorios generados. En el caso de la figura 4.22, los espacios por donde puede pasar el robot coinciden en la segunda y la tercera fila, por lo que el nivel va a ser más sencillo (requiere menor número de instrucciones). Sin embargo, en la figura 4.23, ningún hueco coincide, así que el nivel es más complejo (requiere mayor número de instrucciones).

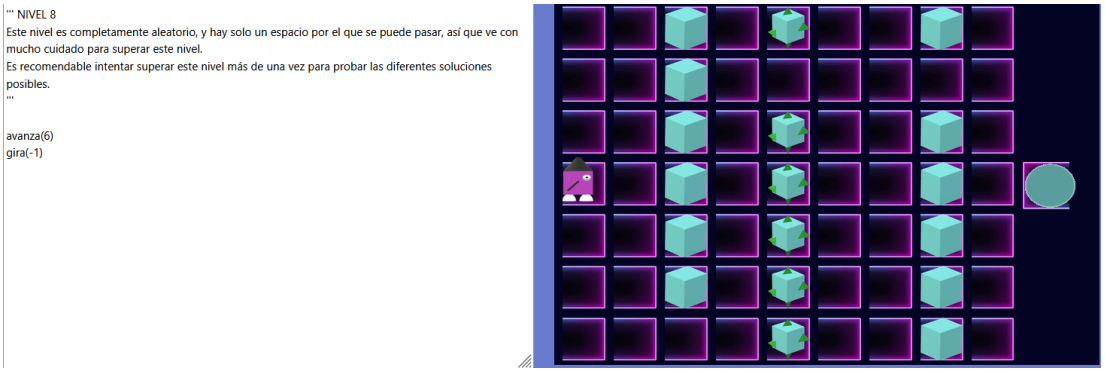


Figura 4.22: Nivel 8 primer ejemplo

Figura 4.23: Nivel 8 segundo ejemplo

No se aporta ningún concepto nuevo de programación en este nivel, simplemente se empieza a complicar el juego cada vez más para proponer un desafío mayor con el paso de los niveles.

4.1.9. Nivel 9

Respecto al nivel 9, el diseño del tablero es difícil. La tercera fila y la quinta fila de obstáculos son aleatorias, además la quinta fila tiene obstáculos con pinchos.

Lo más importante de este nivel es la introducción de los condicionales. La estructura creada es iniciada por la instrucción si(condicion):, y a continuación las instrucciones que van dentro del condicional, que deben estar identadas correctamente (como se ve en figura 4.24).

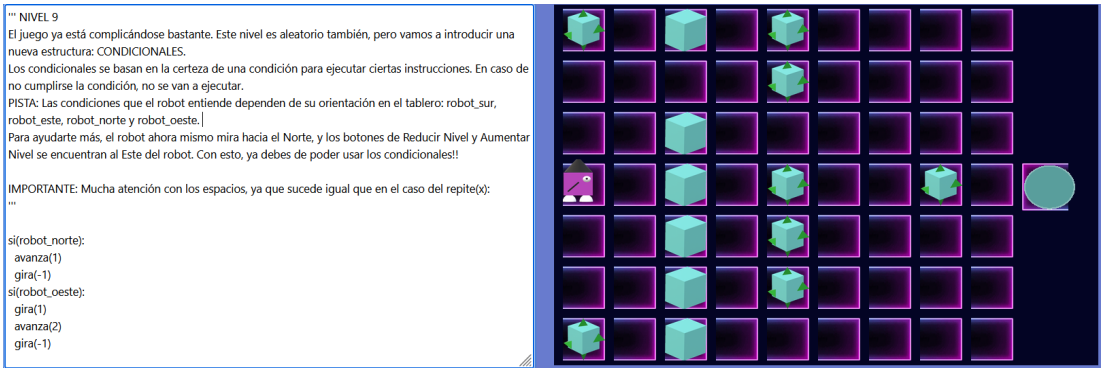


Figura 4.24: Nivel 9

Las condiciones disponibles dependen de la orientación del robot en el momento en que se llegue a la estructura condicional. El robot entiende las condiciones robot_norte, robot_sur, robot_oeste y robot_este. La orientación del robot se tiene en cuenta respecto al tablero (norte del tablero, sur del tablero, etc). Como se explica en el comentario del nivel, el robot comienza orientado al norte, por lo que al norte se encuentra la plataforma final, y los botones de *Reducir Nivel* y *Aumentar Nivel* se encuentran al este del robot.

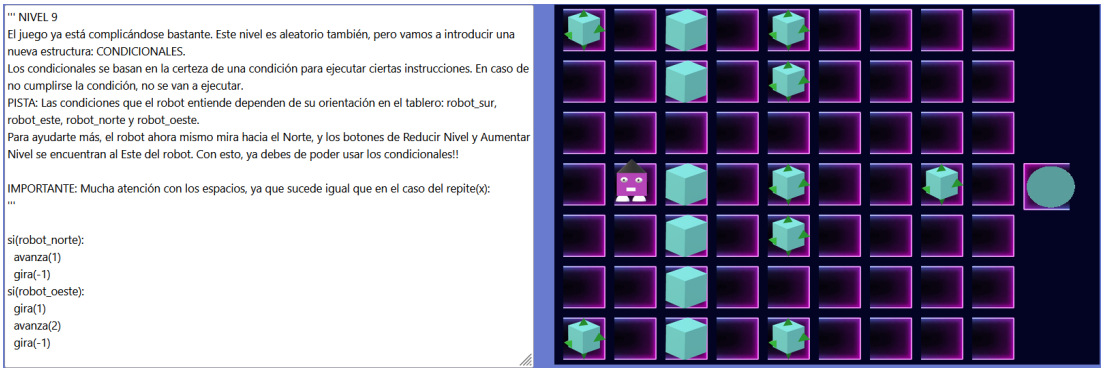


Figura 4.25: Nivel 9 ejecución por defecto

El resultado de la ejecución del código de ayuda para el nivel 9 se encuentra en la figura 4.25, donde el robot ejecuta solamente las instrucciones dentro de la condición robot_norte (el robot siempre empieza orientado al norte). La condición robot_oeste no se cumple (el robot acaba orientado hacia el este tras gira(-1)).

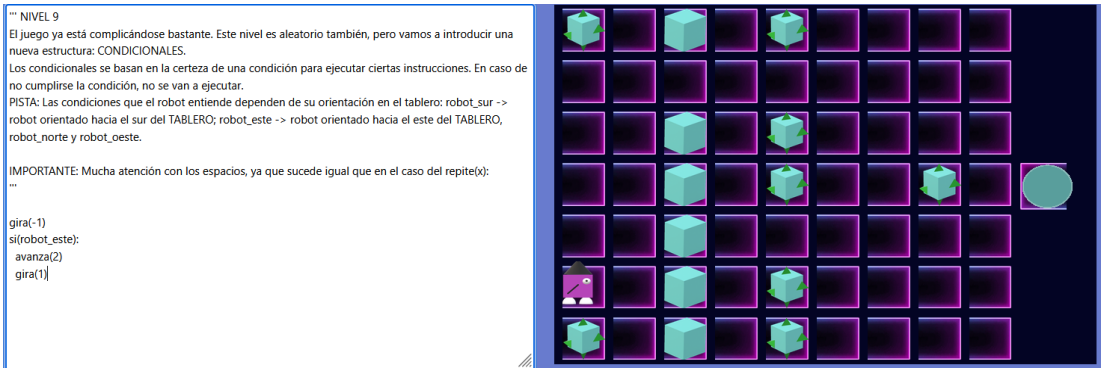


Figura 4.26: Nivel 9 condición robot_este

Tal y como se ve en la figura 4.26, si hay instrucciones antes que modifiquen la orientación del robot, la condición tiene en cuenta ese cambio. En este caso, gira(-1) deja al robot orientado al este, por lo que la condición se cumple y el robot ejecuta las instrucciones avanza(2), gira(1).

En este nivel se introducen las estructuras condicionales. Se explica mediante el comentario por defecto del nivel cómo funcionan los condicionales, es decir, que las instrucciones dentro de la estructura se ejecutan sólo si se cumple la condición o guarda. También los usuarios vuelven a encontrarse con los bloques de código en Python, igual que pasaba con el nivel 7 en el caso de los bucles.

4.1.10. Nivel 10

Este nivel, junto al nivel 12, son los más difíciles de resolver de todo el juego. El diseño es totalmente aleatorio, excepto la última fila. La última fila no puede ser aleatoria, ya que si el hueco no está justamente en el medio de la fila el nivel es insuperable.

Respecto a los obstáculos, todos son obstáculos de pinchos, por lo que la dificultad es todavía mayor (ver figuras 4.27 y 4.28).

Figura 4.27: Nivel 10 diseño aleatorio 1

Figura 4.28: Nivel 10 diseño aleatorio 2

Este nivel aporta más dificultad al juego, esto ha sido diseñado para intentar que el juego atraiga aún más a los alumnos que hayan conseguido superar todos los niveles anteriores. Además, a estas alturas, se pueden usar tanto los bucles como los condicionales para intentar superar el nivel.

4.1.11. Nivel 11

El nivel 11 introduce el último concepto que pretende enseñar este juego: el testing de código con Python.

Respecto al diseño gráfico del nivel, es un nivel fijo sin elementos aleatorios (ver figura 4.29). Por un lado, la primera fila de pinchos solo se puede atravesar por debajo, mientras que la segunda fila de pinchos se puede pasar únicamente por arriba, por lo que se requiere un buen manejo de las instrucciones avanza() y gira().

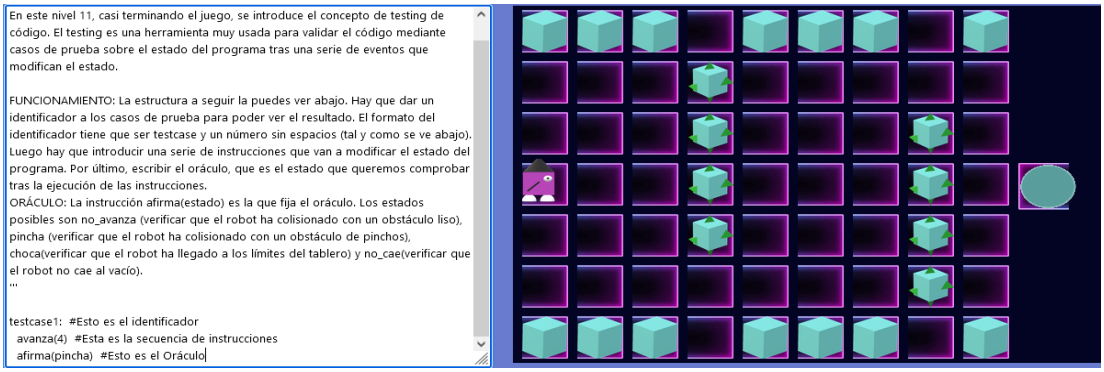


Figura 4.29: Diseño del nivel 11

El comentario que se ha diseñado para el nivel es el más largo hasta ahora, pero se necesita explicar con detalle la forma en la que se puede realizar el testing en este juego. En resumen, hay que seguir la estructura que se da como ejemplo para ejecutar: asignar un identificador al caso de prueba (mediante la palabra clave testcase y un número justo detrás), acabado con el carácter ':'; después, indicar la secuencia de instrucciones que se debe ejecutar para el caso de prueba; por último, mediante la instrucción afirma() se indica el oráculo del test. El oráculo es el estado del programa que se pretende verificar tras la ejecución de la secuencia de instrucciones.

Los posibles oráculos son:

- no_avanza (not_forwards en la versión en inglés): Sirve para verificar que tras ejecutar la secuencia de instrucciones indicada, el robot ha chocado con un obstáculo liso que no le ha permitido avanzar.

- pincha (stabbed en la versión en inglés): Mediante el oráculo 'pincha' podemos verificar si tras ejecutar las instrucciones, el robot ha colisionado con un obstáculo de pinchos.

- choca (collides en inglés): El oráculo 'choca' verifica si el robot ha chocado con los límites del tablero. Es decir, si el robot ha ejecutado alguna instrucción que le ha hecho chocarse con los límites del tablero.

- no_cae (not_falls en inglés): Mediante 'no_cae' podemos comprobar si el robot cae o no cae al vacío tras ejecutar las instrucciones indicadas. El robot caerá al vacío en caso de salirse por el límite del eje x (positivo) del tablero (ver figura 4.30).

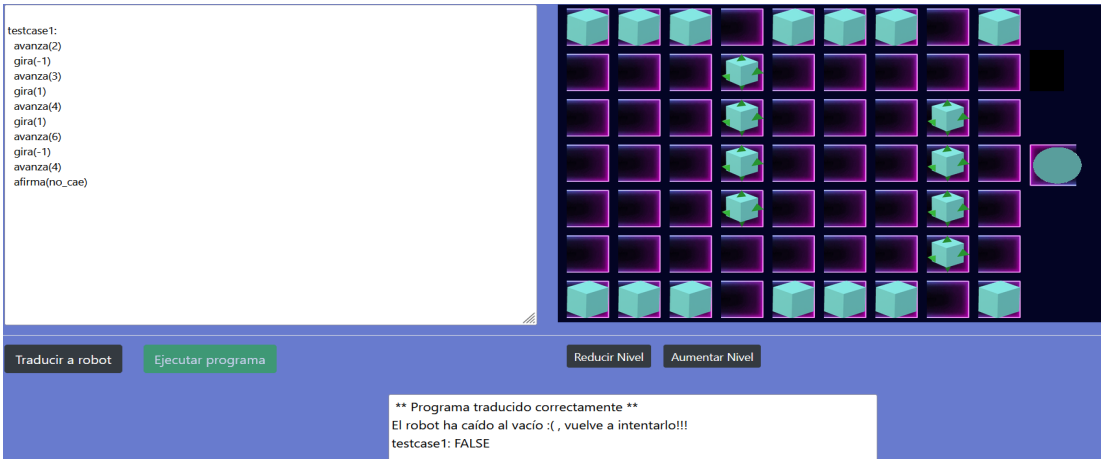


Figura 4.30: Caso de prueba en el nivel 11, el robot cae al vacío

Para aclarar mejor los oráculos y la nueva estructura, se ha añadido en la figura 4.31 una ejecución donde se combinan varios casos de prueba con diferentes oráculos para ver un uso más completo de la estructura. Se muestra en la consola que el testcase6 y el testcase8 son TRUE, ya que el robot ha colisionado con un obstáculo sencillo tras la instrucción avanza(5) del testcase6, y luego tras la instrucción avanza(2) del testcase8, el robot ha chocado con el límite inferior del tablero. Por último, el testcase1 es FALSE debido a que con las instrucción avanza(2), desde la casilla inicial, el robot no llega a colisionar con ningún obstáculo de pinchos.

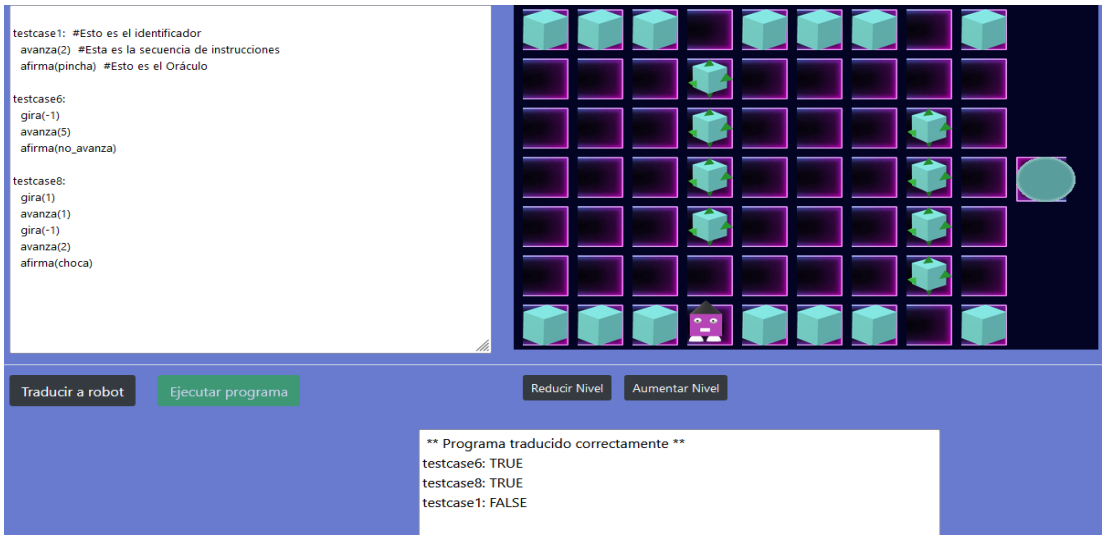


Figura 4.31: Nivel 11 ejecución con varios casos de prueba

Como ya se ha explicado, el concepto que transmite este nivel el juego es el testing de código con Python. La estructura que se ha diseñado en este juego no es exactamente la estructura que sigue Python para el testing, pero la instrucción afirma (en la versión inglesa assert) si que respeta el testing de Python (ver figura 3.2).

4.1.12. Nivel 12

Para concluir el juego, el nivel 12 contiene el diseño más complicado hasta el momento. Es el nivel más restrictivo en cuanto a movimientos posibles, y a parte es el nivel con más obstáculos en el tablero. Las 4 filas de obstáculos intermedias son aleatorias, mientras que las filas horizontales de arriba son fijas y restringen mucho el movimiento.

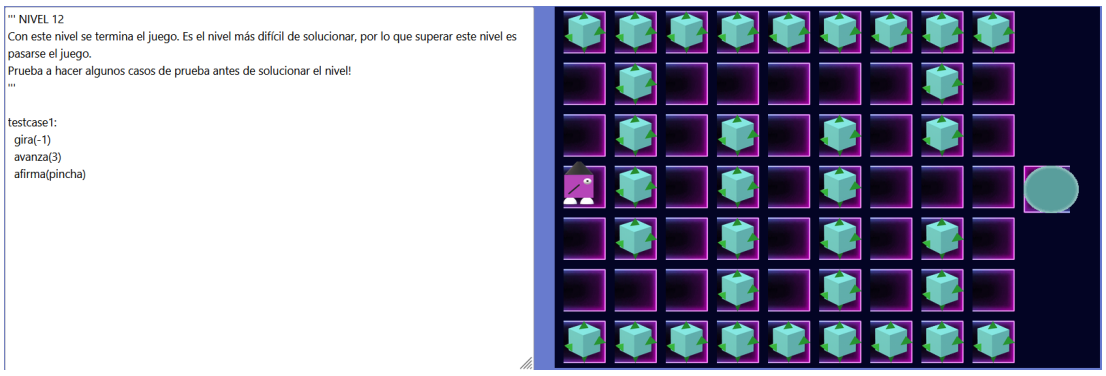


Figura 4.32: Diseño del nivel 12

En el comentario del nivel se recomienda a los usuarios que realicen varios casos de prueba sobre el nivel para afianzar la estructura de los casos de prueba, ya que si han

sido capaces de superar todos los niveles anteriores, este nivel deberían ser capaces de superarlo sin demasiados problemas.

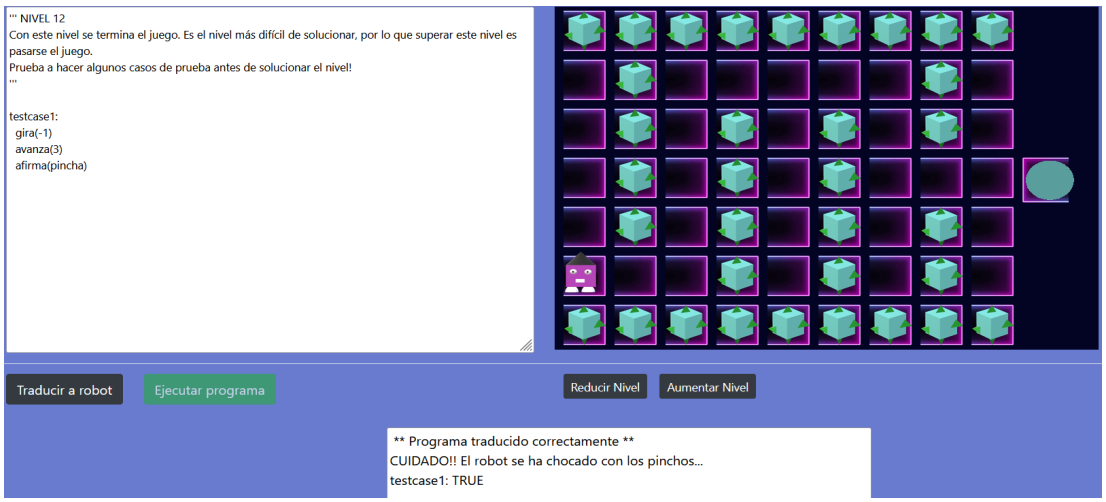


Figura 4.33: Nivel 12 ejecución por defecto

El resultado de la ejecución por defecto del nivel se puede ver en la figura 4.33, donde el robot colisiona con un obstáculo de pinchos tras la instrucción avanza(3), por lo que el resultado del testcase1 es TRUE.

4.1.13. Estilo (CSS+Bootstrap)

Para concluir con el diseño, se va a resumir el estilo que da forma al diseño del juego. Para empezar, algunos elementos de la interfaz tienen únicamente estilo CSS, mientras que otros elementos combinan clases de Bootstrap con estilo CSS.

En este trabajo no se ha creado ningún fichero de estilo .css, si no que se el estilo se ha dado usando la propiedad style de los elementos HTML (ver figura 4.3).

```
img_player.style.left = "10px";
img_player.style.top = "180px";
img_player.src = "robot_lateral.gif";
```

Figura 4.34: Lógica del programa modificando el estilo CSS

Mediante Javascript se modifica gran parte del estilo CSS del juego (ver figura 4.34). La mayoría de modificaciones son posiciones de imágenes, el source de la imagen del robot, etc.

Para el uso de las clases de Boostrap, es importante importarlo en el fichero HTML del proyecto, tal y como se hace en la figura 4.35.

```
<title>TFG Aitor Lapeña Navarro</title>
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">
```

Figura 4.35: Importar clases de Bootstrap en el proyecto

Una vez importado, se puede usar cualquier clase de Bootstrap para dar estilo al HTML (por ejemplo, el estilo de los inputs tipo botón en la figura 4.5). Estas clases son aplicables según el tipo de elemento HTML, y se pueden encontrar en la página de Bootstrap [15].

4.2 Arquitectura del juego

La arquitectura de un juego se define como el conjunto de elementos software que dan forma y estructura al juego. Se ha diseñado un diagrama para entender de manera ordenada la arquitectura del juego:

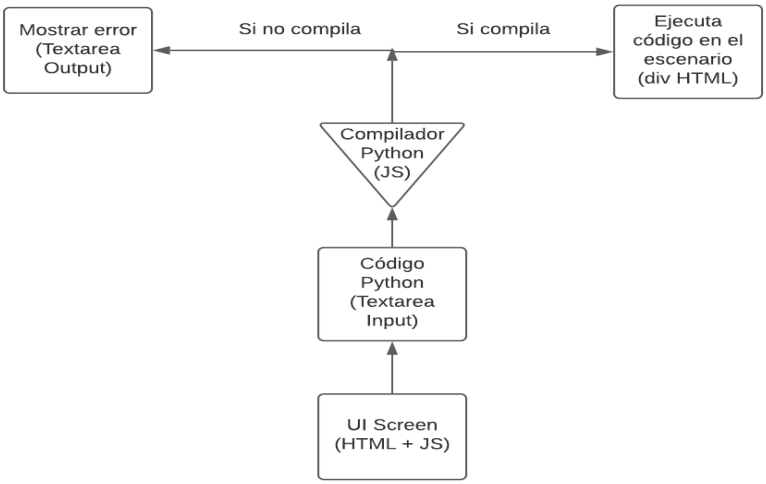


Figura 4.36: Arquitectura del Juego

Se van a explicar ahora los elementos que conforman la arquitectura del juego desarrollado:

1. **UI Screen:** Lo primero que se observa al iniciar el juego es la pantalla principal con la que van a interactuar los jugadores. En ella se muestra directamente el nivel 1 cargado y los elementos de la arquitectura que se mantienen en todos los niveles. Por lo tanto, la UI Screen es el contenedor de todos los elementos de la arquitectura. En la figura 4.1, que se muestra en el apartado de diseño, se ve la interfaz principal del juego.

2. **Código Python:** Para poder ejectuar el código e ir superando los niveles, se necesita introducir el código a traducir en el textarea modificable que actúa como el panel donde el jugador introduce las instrucciones. El panel contiene unos comentarios predefinidos para cada nivel que no son ejecutables y que dan información sobre los niveles. Se usa la estructura de comentarios multilínea en Python (''' comentario '''), aunque cualquier comentario de una línea tampoco se ejecutan (#comentario). A parte, se da un pequeño avance de las instrucciones que pueden ser útiles para el nivel concreto, aunque los jugadores pueden introducir cualquier secuencia válida. En la figura 4.37 se ve el panel del nivel 1 donde se ve el comentario inicial que explica el juego junto a una instrucción de ayuda para los usuarios.

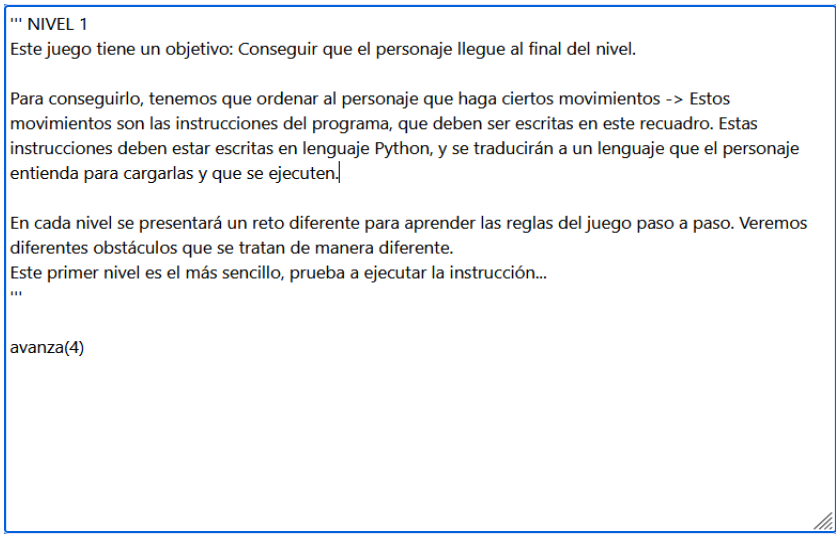


Figura 4.37: Panel de código Python

3. **Compilador Python:** Como el objetivo de este trabajo es enseñar las bases del lenguaje Python, el texto que se escriba en el panel de código va a ser entendido como código en Python, por lo que se deben seguir las reglas del lenguaje.

   Ese código tiene que traducirse de alguna manera para que el robot pueda entender las instrucciones que los jugadores han introducido en el panel de código. Para ello, se ha implementado un parser para traducir a lenguaje robot". Este parser recorre todos los caracteres del panel de código Python y los analiza, para ver si se trata de un comentario multilínea, comentario sencillo, o directamente son instrucciones. No se ha implementado un parser de Python completo con todas las funcionalidades del lenguaje, ya que solo se requieren los comentarios y las instrucciones que entiende el robot.

   Para compilar el código que se ha introducido, simplemente hay que hacer click en el botón Traducir a Robot. Si la compilación fue bien, se habilita el botón de Ejecutar código y se muestra en la consola un mensaje para informar sobre el éxito en la compilación (ver figura 4.38).

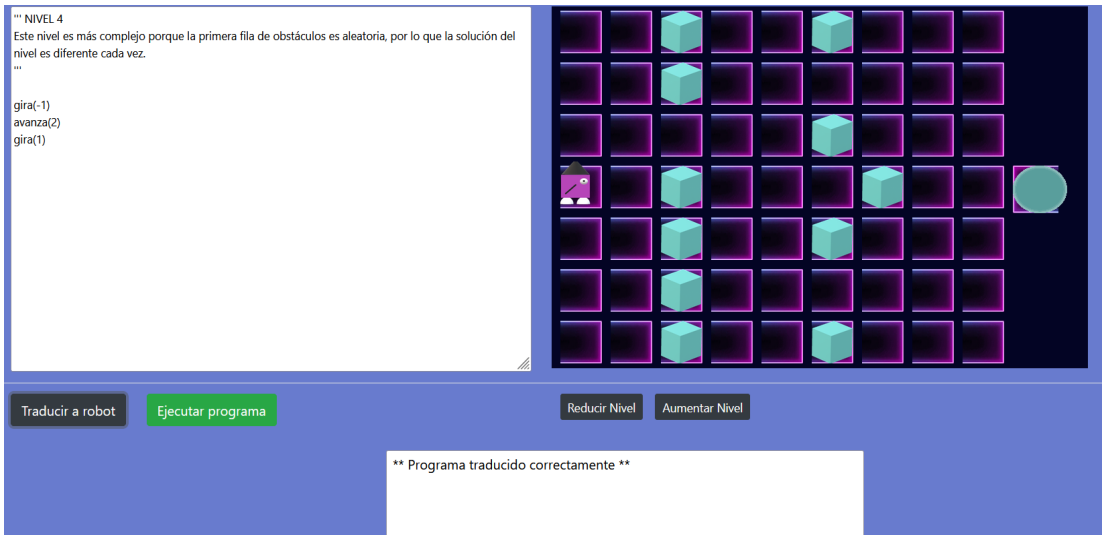


Figura 4.38: Éxito en la compilación de código

Sin embargo, si el código introducido no compila bien, se analiza el error que ha podido cometer el usuario para informar del error a través de la consola, y el botón de ejecución no se habilita. No se pueden implementar todos los tipos de errores sintácticos posibles, así que se han depurado los más comunes: falta de paréntesis, parámetro incorrecto, falta de parámetro, instrucciones no existentes, y en los casos de estructuras como el repite (bucle) o si (condicionales), comprobar la identación, los símbolos de puntuación (el carácter ':'), condición no existente, etc.

En la figuras 4.39, 4.40 y 4.41 se pueden ver algunos errores junto a los mensajes de error que se muestran.

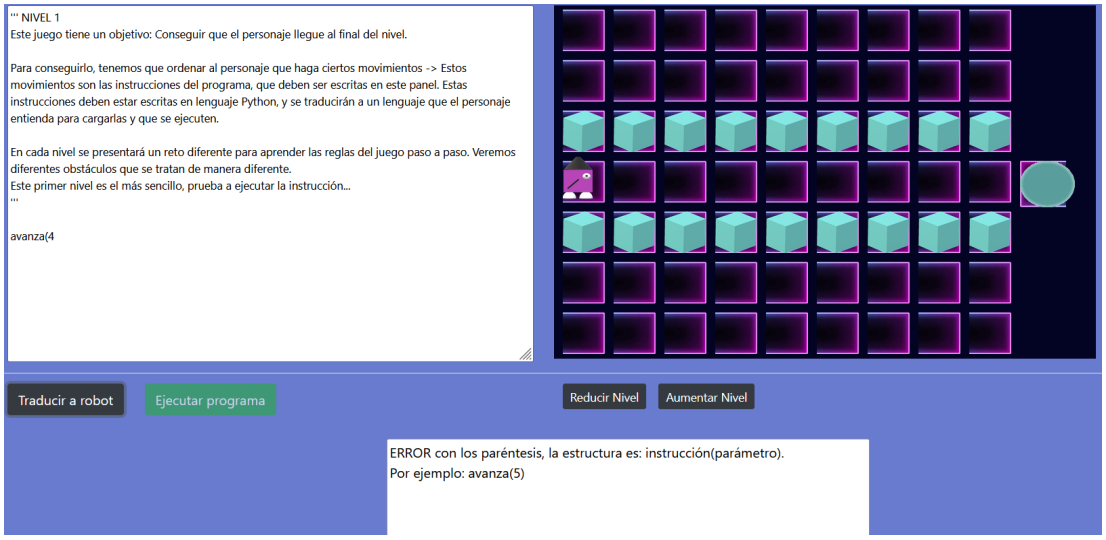


Figura 4.39: Mensaje de error con paréntesis

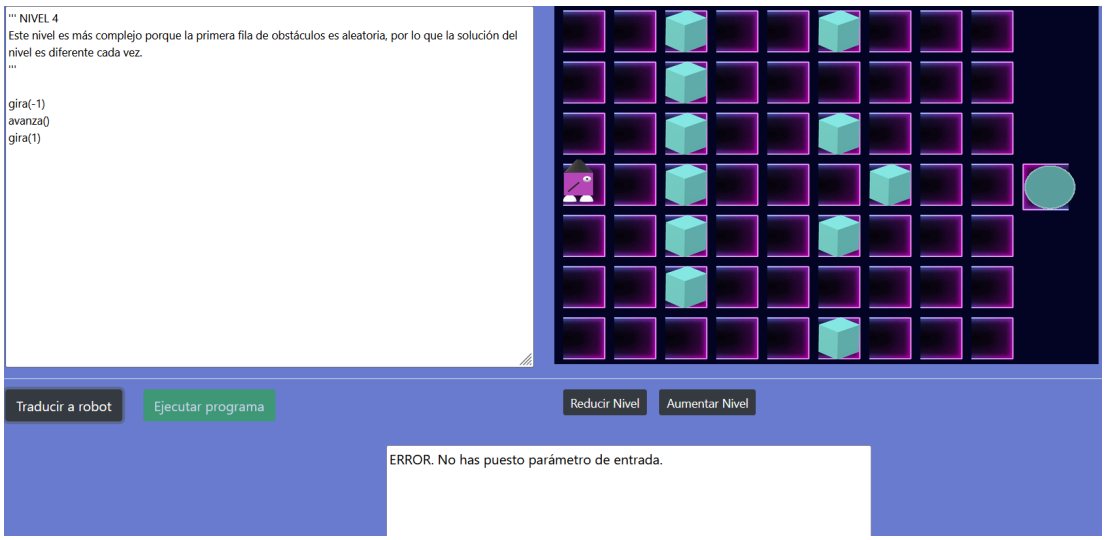


Figura 4.40: Mensaje de error falta de parámetro

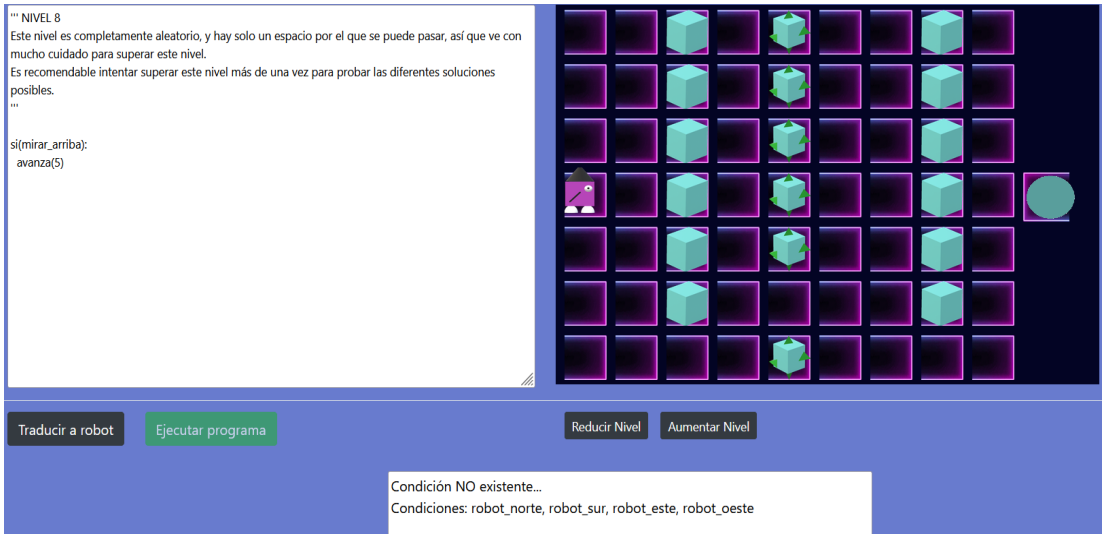


Figura 4.41: Mensaje de error condición no existente

4. **Ejecución del código:** Una vez el texto del panel haya sido compilado correctamente y traducido al lenguaje del robot, se comienza con la ejecución del código.

   La ejecución se activa haciendo click en el botón Ejecutar programa, que como se explicó arriba, es un botón que está deshabilitado hasta que se traduce el código y se compila de manera correcta (ver figura 4.40, donde ante un error de compilación, el botón sigue inhabilitado).

   Las instrucciones disponibles que activan al robot y provocan movimientos en el robot son:

   - avanza(x): La instrucción avanza(x) hace que el robot avance x casillas hacia la dirección donde está orientado (se podría decir que va recto hacia donde está mirando). La ejecución de esta instrucción se puede ver afectada por las colisiones con los obstáculos, como se ha explicado en el diseño de los niveles anteriormente, y también por los límites del escenario, es decir, el robot no puede salir por los laterales del tablero.

   - gira(x): La instrucción gira(x) hace que el robot gire x veces, sin avanzar de la casilla en la que se encuentra en ese momento. Como se ha explicado en apartados anteriores, el sentido del giro del robot depende de si el parámetro x es positivo o negativo. La ejecución de esta instrucción no se ve afectada, siempre se va a ejecutar, excepto en caso de alcanzar el final del nivel, ya que en este caso se termina la ejecución de cualquier instrucción.

```
gira(-1)
avanza(2)
gira(1)
```

Figura 4.42: Secuencia de instrucciones

   Estas instrucciones se pueden ejecutar individualmente (figura 4.42) o añadirlas dentro de estructuras: bucles y condicionales:

   - repite(x): Los bucles se han implementado a través de la estructura repite(x):, donde x es el número de iteraciones que se hacen las instrucciones que estén dentro de la estructura (figura 4.43). El bucle solamente se ve interrumpido por una colisión con el obstáculo de pinchos o por el final del nivel.

```
repite(4):
  avanza(2)
  gira(1)
```

Figura 4.43: Instrucciones dentro de una estructura repite

Como se ha explicado en el diseño del nivel 7, son muy importantes los espacios para que se tomen las instrucciones dentro de las estructuras, ya que esto es una característica propia del lenguaje Python (ver figura 2.11).

- si (condición): Este es el caso de los condicionales, que en este juego se han implementado a través de la estructura si(condición):. En cuanto a las condiciones, se han creado 4 condiciones dependiendo en la orientación del robot. La funcionalidad y las condiciones posibles se explicarán en el capítulo 5.

En el nivel 9 se introducen estas estructuras, y al igual que en los bucles con repite(x), es necesario identar de manera correcta las instrucciones dentro de los condicionales, ya que si no se dejan 2 espacios, las instrucciones no se considerarán parte de la estructura condicional (ver figura 4.44).

```
si(robot_norte):
  avanza(1)
  gira(-1)
si(robot_oeste):
  gira(1)
  avanza(2)
  gira(-1)
```

Figura 4.44: Estructura condicional

Los condicionales se anidan, es decir, se ejecutan las instrucciones dentro del primer condicional, y si al terminar de ejecutarlas se cumple la siguiente condición, se ejecutarán estas instrucciones también.

- testcase + afirma(estado): Se introduce el concepto de testing de código en los últimos niveles. Los casos de prueba tienen un formato especial que hay que cumplir. En la figura 4.45, se muestra la estructura que se da como ayuda en el nivel 11, donde además, a través de comentarios, se explica con detalle el proceso de testing en el juego.

```
testcase1:  #Esto es el identificador
  avanza(4)  #Esta es la secuencia de instrucciones
  afirma(no_pincha)  #Esto es el Oráculo
```

Figura 4.45: Estructura casos de prueba

La secuencia de instrucciones del testcase provoca que el robot se mueva, y cuando esa secuencia de instrucciones termina, se verifica si el estado del juego corresponde con el oráculo que se especifica en la instrucción afirma (en Python, assert).

En el diseño del nivel 11 y en el capítulo de 5 se puede leer con detalle el funcionamiento y los posibles oráculos.

5. **Mostrar error:** Por último, en caso de que el texto introducido en el panel de código Python pase por el compilador Python, pero no compile de manera correcta, se va

a mostrar un mensaje de error en el Textarea de salida (en la figura 4.1, el textarea que se encuentra abajo de la interfaz).

Este textarea es solamente de lectura, y la lógica del programa analiza el estado del programa para ir mostrando diferentes mensajes.

Se han implementado gran cantidad de errores. Estos errores se reportan comparando cada línea del panel de código con ciertas expresiones regulares (regex). Sólo se reporta un error por compilación, que corresponde con el primero detectado.

En el capítulo 5 se verá la manera de detectar errores y los errores que se han implementado en el juego.

CAPÍTULO 5
Implementación

En el capítulo de Implementación se explica la lógica que da funcionalidad al juego, en especial las funciones más importantes, los lenguajes usados y la manera de testear el código.

5.1 Funciones

En este apartado, se van a explicar con detalle las funciones más importantes del juego. Hay bastantes más funciones que las que se van a explicar, pero con estas se puede entender la lógica básica del juego.

1. **cambiarIdioma:** Como su nombre indica, esta función es la encargada de cargar los valores adecuados en la interfaz gráfica del juego dependiendo del idioma que se seleccione. Esta función es llamada al hacer click en las imágenes de las banderas en el HTML (ver figura 4.3).

   Al haber gran cantidad de cadenas de texto necesarias para el juego, se ha creado un archivo separado para almacenar los textos en arrays de idiomas. Dependiendo del parámetro que le llegue a esta función, se carga el array del idioma indicado en la variable array_idioma (ver figura 5.1).

```
function cambiarIdioma(lang) {
    var h4 = document.getElementById("h4_inicio");
    var btn_trad = document.getElementById("traducirprog");
    var btn = document.getElementById('ejecutarprog');
    var btn_anivel = document.getElementById("anivel");
    var btn_rnivel = document.getElementById("rnivel");
    var texto_py = document.getElementById('tarea');

    switch (lang) {  // preparado para añadir los lenguajes que haya diseñados.
        case "esp":
            array_idioma = array_esp;
            idioma = "español";
            break;
        case "eng":
            array_idioma = array_eng;
            idioma = "ingles";
            break;
        default:
            break;
    }
```

Figura 5.1: Código cambiarIdioma para cargar el array de idioma

   Lo siguiente es cargar el valor de los elementos fijos del HTML, esto es el contenido del h4 y el contenido de los botones (figura 5.2).

```
// Texto del h4 y los botones cambiado al idioma apropiado.
h4.innerHTML = array_idioma[0];
btn_trad.value = array_idioma[1];
btn.value = array_idioma[2];
btn_rnivel.value = array_idioma[3];
btn_anivel.value = array_idioma[4];
```

Figura 5.2: Código cambiarIdioma para cargar el valor de los elementos fijos HTML

Por último, en esta función hay que tener en cuenta el nivel donde se cambia el idioma. Como el usuario puede cambiar el idioma en cualquier momento, esta función debe cargar los valores del panel de código para el nivel actual dependiendo del idioma (ver figura 5.3).

```
// iniciar el contenido en el idioma apropiado para el panel de código cuando hagamos click.
switch(nivel_actual) {
    case 1:
        texto_py.value = array_idioma[5];
        break;
    case 2:
        texto_py.value = array_idioma[6];
        break;
    case 3:
        texto_py.value = array_idioma[7];
        break;
    case 4:
        texto_py.value = array_idioma[8];
        break;
    case 5:
        texto_py.value = array_idioma[9];
        break;
    case 6:
        texto_py.value = array_idioma[10];
        break;
    case 7:
        texto_py.value = array_idioma[11];
        break;
    case 8:
        texto_py.value = array_idioma[12];
        break;
    case 9:
            texto_py.value = array_idioma[13];
            break;
```

Figura 5.3: Código cambiarIdioma para iniciar el panel

2. **cargar_nivel:** La función cargar_nivel es la función que inicializa la interfaz gráfica para cada nivel. En resumen, se ocupa de posicionar los obstáculos en el tablero de una manera determinada.

   Como cada nivel es diferente, hay una variable global nivel_actual que va a controlar a esta función mediante un *switch* (ver figura 5.4).

```
switch(nivel) {
    case 1:
        // NIVEL 1
```

Figura 5.4: Switch que controla la función cargar_nivel

Hay 2 tipos de niveles: fijos y aleatorios. Los niveles fijos se generan creando imágenes del obstáculo en posiciones fijas. En algunos niveles con pocos obstáculos, cada obstáculo se genera por separado (ver figura 5.5).

```
case 2:
    // NIVEL 2
    continuar = limpiar_nivel();
    while (!continuar) {
        continuar = limpiar_nivel();
    }

    texto_py.value = array_idioma[6];

    var obs = new Image();
    obs.src = "obs.gif";
    obs.style.height = '50px';
    obs.style.width = '50px';
    obs.style.position = 'absolute';
    obs.style.top = '185px';
    obs.style.zIndex = 6;
    obs.style.left = '250px';
    array_obstaculos.push(obs);
    escenario.appendChild(obs);
```

Figura 5.5: Crear un obstáculo simple en cargar_nivel

En la figura 5.5 se ve también la llamada a limpiar_nivel(). Esta función se llama al inicio de cada nivel, y más adelante se va a explicar su funcionamiento.

En los obstáculos fijos, también se pueden encontrar patrones para añadir muchos obstáculos. Para ello, hay que hacer un bucle y jugar con el style.top y el style.left de las imágenes (ver figura 5.6).

```
for (var i=0; i<7; i++) {
    var img = new Image();
    img.src = "obs.gif";
    img.style.height = '50px';
    img.style.width = '50px';
    img.style.position = 'absolute';
    img.style.top = '65px';
    img.style.zIndex = 6;
    img.style.left = 70 + 60*i + 'px';
    array_obstaculos.push(img);
    escenario.appendChild(img);
}
```

Figura 5.6: Bucle para crear obstáculos en cargar_nivel

Respecto a los obstáculos aleatorios, se generan de manera similar a los obstáculos fijos, pero hay que generar varios números aleatorios, dependiendo de la aleatoriedad que busquemos. Por ejemplo, en la figura 5.7, se están creando 2 números aleatorios para una fila de obstáculos que tiene 2 huecos.

```
// w,x -> primera fila de obstáculos
var w = Math.floor((Math.random() * 7) + 1);
var x = Math.floor((Math.random() * 7) + 1);

// y -> segunda fila
var y = Math.floor((Math.random() * 7) + 1);
// z -> tercera fila
var z = Math.floor((Math.random() * 7) + 1);

var i = 0;
var j = 0;
texto_py.value = array_idioma[14];

while (i<5){
    // los valores de 'w' y 'x' deciden aleatoriamente donde habrá un hueco
    if (w == j || x == j) {
        j++;
    } else {
        var img = new Image();
        img.src = "pinchos.gif";
        img.style.height = '50px';
        img.style.width = '50px';
        img.style.position = 'absolute';
        img.style.top = 5 + 60*j + 'px';
        img.style.zIndex = 6;
        img.style.left = '130px';
        array_obstaculos.push(img);
        escenario.appendChild(img);
        i++;
        j++;
    }
```

Figura 5.7: Generación de filas de obstáculos aleatorias

El resultado de este código se puede ver en la figura 4.27, en concreto corresponde con la primera fila de obstáculos (que tiene 2 huecos aleatorios).

Esta función también carga el contenido del panel de código. En todos los niveles, se carga el contenido que se ha creado para cada nivel con instrucciones en un comentario (ver capítulo 4). El valor del comentario en cada nivel está guardado en el array del idioma correspondiente (del que se ha hablado antes en la función cambiarIdioma), en una posición concreta (por ejemplo, la figura 5.5 corresponde al nivel 2, cuya posición en el array es la 6).

3. **parsearCodigo:** La función parsearCodigo es la encargada de entender y clasificar el contenido del panel de código. Recibe como parámetro el contenido del *textarea* (ver figuras 5.8 y 5.9).

```
var texto_python = document.getElementById("tarea");
var img_player = document.getElementById('robot');

var texto = texto_python.value;
```

Figura 5.8: Guardado del contenido del panel de código en la variable texto

```
parsearCodigo(texto).then(array => {
```

Figura 5.9: Llamada a la función parsearCodigo()

Ya se ve también en la figura 5.9 que la función parsearCodigo devuelve una Promise para garantizar la ejecución ordenada de la lógica del programa.

Entrando ahora en detalle sobre esta función, recorre caracter a caracter el contenido de la variable texto mediante un bucle *for*, y dependiendo del valor del caracter, va a hacer unas cosas u otras (esto lo controla el *switch* de la figura 5.10).

```
for (var i = 0; i < codigo.length; i++) {
    var caracter = codigo.charAt(i);
    switch (caracter) {
        case '#':
            if (!es_Comentario_linea) {
                es_Comentario_linea = true;
            }
            if (instruccion != "") {
                if (instruccion.charAt(0) == ' ') {  // Eliminar espacios en caso de
                    var inst_temp = instruccion;
                    inst_temp = inst_temp.trim();
                    instruccion = "  " + inst_temp;
                } else {   // Eliminar espacios sin importar la identación de Python.
                    instruccion = instruccion.trim();
                }
                var añadir = array_instrucciones.push(instruccion);
                instruccion = "";
            }
            break;
        case '\n':
            if (es_Comentario_linea) {
                es_Comentario_linea = false;
            } else if (es_Comentario_largo) {
                /* do nothing */
            } else {
                if (instruccion != "") {
                    añadir = array_instrucciones.push(instruccion);
                    instruccion = "";
                }
            }
            break;
```

Figura 5.10: Switch de la función parsearCodigo

Como se puede ver, el caracter # inicia un comentario de una línea (igual que en Python). Sin embargo, si se encuentra un caracter # y ya hay algo en la variable *instruccion*, es porque el comentario empieza a mitad de línea, como por ejemplo se puede ver en el comentario del nivel 11 (en la figura 4.29). Por lo tanto, hay que añadir la instrucción al array (obviando el comentario).

El salto de línea (/n) también es un caracter que hay que tener en cuenta, ya que un salto de línea debe finalizar un comentario sencillo de una línea. También el salto de línea sirve como límite para añadir el valor de la variable *instruccion* al array de instrucciones.

Falta el caracter ', que también hay que tenerlo en cuenta, ya que en Python, el comentario multilínea se implementa con 3 comillas seguidas.

Por último, el resto de caracteres se deben tratar de la misma manera. Si están dentro de un comentario, se ignoran; sin embargo, si no lo están, se añade la línea entera (concatenando caracter a caracter para formar un string) al array de instrucciones. Por lo tanto, si los comentarios multilínea se han cerrado de manera correcta o si la línea de código no corresponde con un comentario simple, el contenido se debe analizar (esta función no es la encargada de hacerlo). La lógica se puede ver en la figura 5.11.

```
                break;
            default:
                if (!es_Comentario_linea && !es_Comentario_largo) {
                    instruccion += caracter;
                    if (i == (codigo.length - 1) ) {
                        array_instrucciones.push(instruccion);
                    }
                }
                break;
        }
    }
    if (es_Comentario_largo) {
        texto_salida.value = array_idioma[18];
        resolve(null);
    }
    resolve(array_instrucciones);
})
```

Figura 5.11: Resto de caracteres en el caso default del switch

Esta función hace un resolve() para finalizar la *promise* cuando no quedan más caracteres, y puede resolver con null (en caso de no haber cerrado el comentario multilínea) o con el array de instrucciones en caso de éxito.

4. **avanza:** Como su nombre indica, la función avanza() se encarga de controlar el movimiento del robot. Se llama igual que la función disponible en el juego por simplicidad. La documentación de la función se puede ver en la figura 5.12: el parámetro es el número de casillas que el robot debe avanzar, y la función devuelve una *Promise* para garantizar una ejecución ordenada.

```
// Función para mover al robot un número de casillas indicado por parámetro. Funciona tanto horizontalmente como verticalmente.
/**
 * @param {Número de casillas a avanzar por el robot} npasos
 * @returns Promise resolve()/reject()
 *          resolve() -> true: termina la ejecución de avanza de manera natural (sin pararlo)
 *                       false: termina la ejecución de avanza por colisión con pinchos o caída al vacío.
 *                       2: termina la ejecución de avanza por finalizar el nivel.
 */
function avanza(npasos) {
    return new Promise(function(resolve, reject) {
```

Figura 5.12: Documentación y definición de la función avanza()

Como el movimiento puede ser tanto horizontal como vertical, dentro de la función avanza hay 2 subfunciones para avanzar horizontalmente o verticalmente (ver figura 5.13). Si el robot está orientado para moverse en el eje x, se llamará a la subfunción framehorizontal(), y si se orienta para avanzar en el eje y, se llama a framevertical(). Es importante destacar también, que para simular los pasos del robot, se llama a las subfunciones usando setInterval, con una frecuencia de 275 milisegundos.

```
if (img_src == "robot_lateral.gif" || img_src == "robot_lateral_atras.gif"){
    id = setInterval(framehorizontal, 275);
}

if (img_src == "robot_frente.gif" || img_src == "robot_espalda.gif") {
    id = setInterval(framevertical, 275);
}
```

Figura 5.13: Llamadas framehorizontal y framevertical

La diferencia entre las funciones framehorizontal y framevertical, es que la horizontal modifica el style.left del personaje para que se mueva en el eje x, y la vertical modifica el style.top para que el personaje se mueva en el eje y. En las figuras 5.14 y 5.15, que corresponden con framehorizontal() y framevertical() respectivamente, se puede ver el avance del robot en caso de no haber colisión con obstáculos.

```
function framehorizontal(){
    if ( x < maxPos_left && i < npasos) {
        if (img_src == "robot_lateral.gif") {
            x = x + 60;
        }
        if (img_src == "robot_lateral_atras.gif" && x > minPos_left) {
            x = x - 60;
        }
        hayColision(x,1).then(colision => {
            switch (colision) {
                case 0:  // no hay colisión
                    img_player.style.left = x + "px";
                    i+=1;
                    break;
```

Figura 5.14: Avance horizontal del robot si no hay colisión

```
function framevertical() {
    if (j < npasos) {
        if (img_src == "robot_espalda.gif" && y > maxPos_top) {
            y = y - 60;
        }
        if (img_src == "robot_frente.gif" && y < minPos_top) {
            y = y + 60;
        }
        hayColision(y,2).then(colision => {
            switch (colision) {
                case 0:  // no hay colisión
                    img_player.style.top = y + "px";
                    j+=1;
```

Figura 5.15: Avance vertical del robot si no hay colisión

Como se puede ver, en la función avanza (tanto en framehorizontal como framevertical) se llama a la función hayColision(), la cual analiza si hay colisiones en cada paso que da el robot. La detección de colisiones se realiza recorriendo un array que contiene los obstáculos y comprobando sus propiedades offsetLeft/offsetTop (dependiendo de si el movimiento es horizontal o vertical). En la figura 5.16, se muestra la documentación para la función hayColision() y la parte de colisiones horizontales. La parte de las colisiones verticales es prácticamente igual y por eso no se muestra en la figura.

```
// Función que detecta colisiones del robot con obstáculos.
/**
 * @param {Coordenada actual del robot en el tablero} pos_actual
 * @param {Número que indica el eje donde se está moviendo actualmente el robot} direccion
 *        direccion == 1: el robot se está moviendo en el eje x
 *        direccion == 2: el robot se está moviendo en el eje y
 * @returns Promise resolve()/reject()
 *              resolve() -> 1: colisión con obstáculo sencillo
 *                           2: colisión con obstáculo de pinchos
 */
function hayColision(pos_actual, direccion) {
    return new Promise(function(resolve, reject) {
        var img_player = document.getElementById('robot');
        var pos_posible_colisiony = img_player.offsetTop + 5;
        var pos_posible_colisionx = img_player.offsetLeft;

        switch (direccion) {
            case 1:
                // comprobar colision horizontal
                array_obstaculos.forEach(obstaculo => {
                    var obs_src = obstaculo.getAttribute("src");
                    if (pos_actual == obstaculo.offsetLeft && pos_posible_colisiony == obstaculo.offsetTop) {
                        switch (obs_src) {
                            case "obs.gif":
                                resolve(1);
                                break;
                            case "pinchos.gif":
                                resolve(2);
                                break;
                            default:
                                /* do nothing */
                        }
                    }
                });
                resolve(0);
                break;
```

Figura 5.16: Código de la función hayColision() en movimiento horizontal

La ejecución de la función termina en cuanto se detecta una colisión o cuando ya se ha recorrido todo el array, por lo que el robot puede avanzar sin problemas. En el switch(obs_src), se distinguen las colisiones, ya que no se trata de la misma manera el caso de una colisión con un obstáculo liso y el caso de una colisión con un obstáculo de pinchos.

Por último, la función avanza también detecta las caídas al vacío del robot, y por ello comprueba si el usuario ha introducido algún testcase con la condición no_cae; y también el final del nivel (ya que sólo se pueden producir tras una llamada a la función avanza, en concreto tras una llamada a framehorizontal). La lógica se muestra en la figura 5.17, donde también se comprueba que la solución del usuario ha sido eficiente o no (para mostrarlo por la consola).

```
if (x == maxPos_left) {
    if (img_player.offsetTop != 180) {  // el robot se ha caído al vacío
        texto_salida.value += '\n' + array_idioma[20];
        img_player.src = "suelo_negro.png";
        if (array_oraculos.includes(array_idioma[86])) {
            var indice = array_oraculos.indexOf(array_idioma[86]);
            texto_salida.value += '\n' + casos_prueba[indice] + ' FALSE';
            array_oraculos.splice(indice, 1);
            casos_prueba.splice(indice, 1);
        }
        resolve(false);
        clearInterval(id);
    } else {  // NIVEL COMPLETADO
        if (eficiente == 0) {  // nivel aleatorio, no se puede comprobar la eficiencia
            texto_salida.value = array_idioma[21];
        } else if (eficiente == 2) {  // solución eficiente
            texto_salida.value = array_idioma[79];
        } else texto_salida.value = array_idioma[78];  // solución ineficiente
        resolve(2);
    }
}
```

Figura 5.17: Caída al vacío y final del nivel con la función avanza()

5. **gira:** Como indica su nombre, la función gira() se ejecuta cuando el usuario introduce una llamada a la función con el mismo nombre.

```
// Función que provoca que el robot gire mediante cambios de imagen.
/**
 * @param {Número de giros que va a dar el robot} ngiros
 *        Si ngiros > 0, el robot gira en sentido contrario a las agujas del reloj.
 *        Si ngrios < 0, el robot gira en sentido de las agujas del reloj.
 * @returns Promise resolve()/reject()
 */
function gira(ngiros) {

    return new Promise(function(resolve,reject) {
        var i = 1;
        var j;
        var id = null;
        var rotaciones = 0;
        var invertir = false;
        var img_player = document.getElementById('robot');
        var img_src = img_player.getAttribute("src");

        if (ngiros < 0) {
            invertir = true;
            ngiros = -ngiros;
        }

        clearInterval(id);
        id = setInterval(girar, 350);
```

Figura 5.18: Documentación y análisis del parámetro en la función gira())

Lo primero a tener en cuenta es el parámetro de entrada, ya que puede ser un número positivo o negativo. En caso de ser negativo, el robot va a girar en sentido contrario a las agujas del reloj, y si es positivo, girará en sentido de las agujas del reloj (ver figura 5.18).

Si el parámetro *ngiros* es negativo, se quita el signo para hacerlo positivo, ya que interesa el valor absoluto del número de giros, y se pone la variable local invertir a true para tenerlo en cuenta.

Respecto a la lógica del giro, cabe destacar, como se ve en la figura 5.18, que hay una subfunción girar que se ejecuta con una frecuencia de 350ms usando setInterval().

La subfunción girar() implementa el giro, el cuál se hace a través de un *switch* usando una variable que el campo src de la imagen del robot, que se debe ir modificando para simular el giro (el switch se puede ver en la figura 5.19).

```
if (i <= ngiros) {
    j = i - (4*rotaciones);
    switch (j) {
        case giro_espalda:
            img_player.src = "robot_espalda.gif";
            break;
        case giro_izq:
            img_player.src = "robot_lateral_atras.gif";
            break;
        case giro_frente:
            img_player.src = "robot_frente.gif";
            break;
        case giro_der:
            img_player.src = "robot_lateral.gif";
            rotaciones++;
            break;
        default:
            break;
        }
    i++;
} else {
    resolve(true);
    clearInterval(id);
}
```

Figura 5.19: Switch en la subfunción girar()

6. **limpiar_nivel:** La función limpiar_nivel(), que ya ha sido mencionada antes al explicar la función cargar_nivel(), es la que limpia el escenario de obstáculos y resetea el resto de elementos que influyen para el inicio de otros niveles (ver figura 5.20).

```
// Función para vaciar el nivel de obstáculos y resetear al robot, llamada al cargar cada nivel.
/**
 * @returns true si el nivel está limpiado completamente // false si todavía faltan obstáculos por limpiar
 */
function limpiar_nivel() {
        var id = null;
        var escenario = document.getElementById('escenario');
        var texto_salida = document.getElementById('output');
        var img_player = document.getElementById('robot');
        var btn = document.getElementById('ejecutarprog');
        btn.disabled = true;
        casos_prueba = [];
        array_oraculos = [];
        img_player.style.left = "10px";
        img_player.style.top = "180px";
        img_player.src = "robot_lateral.gif";
        texto_salida.value = "";
        for (var i = 0; i < array_obstaculos.length; i++) {
            id = array_obstaculos.pop();
            escenario.removeChild(id);
        }

        if (array_obstaculos.length == 0) {
            return true;
        }
        return false;
}
```

Figura 5.20: Código de limpiar_nivel()

Las acciones que realiza esta función son:

- Deshabilitar el botón de Ejecutar Código.

- Resetear arrays de los testcases.

- Resetear posición del robot, poniendo sus posiciones iniciales en sus propiedades de estilo .left y .top.

- Vaciar la consola.

- Recorrer el array de obstáculos, y eliminar cada elemento del array y del escenario.

7. **comprobarArray:** Es una de las funciones más importantes de todo el código. Implementa toda la parte del compilador, es decir, comprueba si el array de instrucciones que se ha obtenido en la función parsearCodigo() es válido, y si es válido, lo traduce para ser ejecutado por el robot. En la figura 5.21 se muestra la documentación y definición de la función.

```
// Función que valida cada línea de código del panel de código.
/**
 * @param {Array que contiene cada línea del panel de código} array
 * @returns Promise resolve()/reject()
 */
async function comprobarArray(array) {
    return new Promise(function(resolve,reject) {
        var texto_salida = document.getElementById('output');
        var terminado = false;
        var es_bucle = false;
        var es_condicional = false;
        var repeticiones = 0;
        var condicion = "";
        var es_tc = false;

        const start = async () => {
            await asyncForEach(array, async (instruccion) => {
                await new function() {
                    // si la instruccion no hace Match con ninguna de estas expresiones, ERROR DE COMPILACIÓN.
                    var concuerda_avanza = matchExact(array_idioma[33], instruccion);
                    var concuerda_gira1 = matchExact(array_idioma[34], instruccion);
                    var concuerda_gira2 = matchExact(array_idioma[35], instruccion);
                    var concuerda_repite = matchExact(array_idioma[36], instruccion);
                    var concuerda_si_sur = matchExact(array_idioma[37], instruccion);
                    var concuerda_si_norte = matchExact(array_idioma[38], instruccion);
                    var concuerda_si_este = matchExact(array_idioma[39], instruccion);
                    var concuerda_si_oeste = matchExact(array_idioma[40], instruccion);
```

Figura 5.21: Documentación de la función comprobarArray()

Para validar el contenido del array pasado como parámetro, se han definido expresiones regulares para comprobar si cada línea del panel de código corresponde con alguna de las instrucciones definidas. Las expresiones regulares están en la variable array_idioma (ver figuras 5.22 y 5.23).

```
//POSICIÓN 33 ES LA PRIMERA REGEX
/avanza\(\d+\)/, /gira\(\d+\)/, /gira\(-\d+\)/, /repite\(\d+\):/, /si\(robot_sur\):/, /si\(robot_norte\):/, /si\(robot_este\):/, /si\(robot_oeste\):/,
/  repite\(\d+\):/, /  avanza\(\d+\)/, /  gira\(\d+\)/, /  gira\(-\d+\)/, /  si\(robot_sur\):/, /  si\(robot_norte\):/, /  si\(robot_este\):/, /  si\(robot_oeste\):/,
```

Figura 5.22: Expresiones regulares en castellano

```
// POSICIÓN 33 REGEX
/forward\(\d+\)/, /turn\(\d+\)/, /turn\(-\d+\)/, /repeat\(\d+\):/, /if\(south_robot\):/, /if\(north_robot\):/, /if\(east_robot\):/, /if\(west_robot\):/,
/  repeat\(\d+\):/, /  forward\(\d+\)/, /  turn\(\d+\)/, /  turn\(-\d+\)/, /  if\(south_robot\):/, /  if\(north_robot\):/, /  if\(east_robot\):/, /  if\(west_robot\):/,
```

Figura 5.23: Expresiones regulares en inglés

La función comprobarArray() está estructurada dependiendo de si hay alguna estructura o no. La función se comporta de manera diferente dependiendo de si se ha iniciado alguna estructura (bucle, condicional o testcase), o si no hay ninguna estructura. En la figura 5.24, se muestra el comportamiento de la función si se ha iniciado una estructura repite() para un bucle.

```
if (es_bucle) {
    // instrucciones dentro de la estructura iterativa repite():
    if (concuerda_avanza_ident || concuerda_gira1_ident || concuerda_gira2_ident) {
    instruccion = instruccion.substring(2); // hay que filtrar los espacios (2 espacios antes de la instrucción)
    array_bucle.push(instruccion);
    } else if (concuerda_repite_ident)  { // no soporta bucles dentro de otro bucle
        texto_salida.value = array_idioma[22];
        resolve(false);
    } else if (concuerda_si_norte_ident|| concuerda_si_este_ident || concuerda_si_sur_ident || concuerda_si_oeste_ident) {  // no soporta co
        texto_salida.value = array_idioma[30];
        resolve(false);
    } else if (concuerda_avanza || concuerda_gira1 || concuerda_gira2) {  // instrucciones sin espacios, fuera del bucle
        es_bucle = false;
        crearArrayBucle(array_bucle, repeticiones).then(() => {
            array_bucle = [];
            array_traducido.push(instruccion);
        })
    } else if (concuerda_si_norte|| concuerda_si_este || concuerda_si_sur || concuerda_si_oeste) {  // fin de bucle, inicio de condicional
        es_bucle = false;
        crearArrayBucle(array_bucle, repeticiones).then(() => {
            array_bucle = [];
            es_condicional = true;
            condicion = instruccion.substring(3);
        })
    } else if (concuerda_repite) {  // inicio de otro bucle
        crearArrayBucle(array_bucle, repeticiones).then(() => {
            array_bucle = [];
            repeticiones = instruccion.match(/\d+/);
        })
```

Figura 5.24: comprobarArray() caso de bucle

La condición if(es_bucle) implica que se ha encontrado un repite() previamente. Todos los *if* encadenados de después sirven para controlar qué se debe hacer al encontrar algo en la siguiente línea de código:

- Si encuentra una instrucción avanza/gira identada (cumpliendo el espaciado de Python), se aplica un subtring para quitar los espacios y se añade al array que se encarga de generar el bucle.

- Si encuentra un repite() identado, muestra un error, ya que no se soportan los dobles bucles.

- Si encuentra un condicional (estructura si():) identado, muestra un error, ya que no se soportan combinaciones de estructuras.

- Si se encuentra una instrucción avanza/gira sin cumplir la identación, se cierra el bucle, pero no se muestra error, ya que se interpreta que el usuario no quiere que la instrucción pertenezca al bucle.

- Si encuentra un repite() o un condicional sin identación, se cierra el bucle actual y se da inicio a la siguiente estructura (si es un condicional, se pone la variable booleana es_condicional a true, y la estructura es similar a la del caso del bucle).

La implementación que se ha hecho para las estructuras condicionales y los casos de prueba es similar, pero falta ver el caso en el que no estamos dentro de ninguna estructura (ver figura 5.25). En ese caso, hay que analizar las siguientes instrucciones:

- Si viene una instrucción avanza/gira, se añade directamente al array para su ejecución.

- Si viene un repite():, se debe poner a TRUE la variable booleana es_bucle.

- Si viene un si():, se debe poner a TRUE la variable booleana es_condicional.

- Si viene un testcase+numero, se debe poner a TRUE la variable booleana es_tc.

- Si no viene ninguna de las anteriores, hay que reportar un error. Para ello, se ha implementado la función depurarError(), que se va a explicar a continuación.

```
if (concuerda_avanza || concuerda_gira1 || concuerda_gira2) {  // instrucción para pushear directamente al array
    array_traducido.push(instruccion);
} else if (concuerda_repite) {  // es un bucle
    es_bucle = true;
    repeticiones = instruccion.match(/\d+/);
} else if (concuerda_si_norte|| concuerda_si_este || concuerda_si_sur || concuerda_si_oeste) {  // es un condicional
    es_condicional = true;
    condicion = instruccion.substring(3);
} else if (concuerda_testcase) {  // inicio de un testcase
    es_tc = true;
    array_tcases.push(instruccion);
} else {  // ERROR
    terminado = depurarError(instruccion);
    while (!terminado) {
        terminado = depurarError(instruccion);
    }
    resolve(false);
}
```

Figura 5.25: comprobarArray() si no hay estructuras

8. **depurarError:** La función depurarError() se llama en caso de que una línea de código no concuerde con ninguna expresión regular de las instrucciones que entiende el robot. Concretamente, lo que hace esta función es analizar el error que ha cometido el usuario y avisarle con un mensaje por la consola (ver figura 5.26 que explica la función).

```
// Función para dar información del error que hay en el código respecto a lo que ha introducido el usuario.
/**
 * @param {String con la línea de código que causó el error} instruccion_actual
 * @returns True para indicar la finalización del análisis del error.
 */
function depurarError(instruccion_actual) {

    var error_parentesis1_av = instruccion_actual.match(array_idioma[53]);
    var error_parentesis2_av = instruccion_actual.match(array_idioma[54]);
    var error_parentesis1_gi = instruccion_actual.match(array_idioma[55]);
    var error_parentesis2_gi = instruccion_actual.match(array_idioma[56]);
    var error_parentesis3_gi = instruccion_actual.match(array_idioma[57]);
    var error_parentesis4_gi = instruccion_actual.match(array_idioma[58]);
    var error_parametro_av = instruccion_actual.match(array_idioma[59]);
    var error_parametro_gira = instruccion_actual.match(array_idioma[60]);
    var error_parametro_repite = instruccion_actual.match(array_idioma[61]);
    var error_parametro_si = instruccion_actual.match(array_idioma[62]);
    var error_parametro_general = instruccion_actual.match(array_idioma[70]);
    var error_instruccion_nostruct_pos = instruccion_actual.match(array_idioma[63]);
    var error_instruccion_nostruct_neg = instruccion_actual.match(array_idioma[69]);
    var error_instruccion_nostruct_str = instruccion_actual.match(array_idioma[100]);
    var error_instruccion_struct_num = instruccion_actual.match(array_idioma[64]);
    var error_instruccion_struct_str = instruccion_actual.match(array_idioma[65]);
    var error_repite_puntos = matchExact(array_idioma[66], instruccion_actual);
    var error_si_puntos = matchExact(array_idioma[67], instruccion_actual);
    var error_condicion = matchExact(array_idioma[68], instruccion_actual);
    var error_assert_noident_1 = instruccion_actual.match(array_idioma[91]);
    var error_assert_noident_2 = instruccion_actual.match(array_idioma[92]);
```

Figura 5.26: Documentación y definición de depurarError()

En la figura 5.26 se ve el uso de expresiones regulares de nuevo. Es la manera más segura de detectar el error concreto que ha cometido el usuario. Para ello, se han definido bastantes expresiones regulares que detectan todo tipo de errores. En la figuras 5.27 y 5.28 se muestran algunas de las expresiones regulares en castellano y en inglés, respectivamente (hay más errores implementados).

```
//POSICIÓN 53 REGEX ERRORES
/avanza\d+\)/g, /avanza\(\d+/g, /gira\d+\)/g, /gira\(\d+/g, /gira-\d+\)/g, /gira\(-\d+/g, /avanza\(\)/, /gira\(\)/,
/repite\(\d+\)/, /si\(\w+\)/, /si\(\w+\):/, /\w+\(-\d+\)/, /\w+\(\)/,
```

Figura 5.27: Expresiones regulares errores en castellano

Se han implementado los siguientes errores:

```
//POSICIÓN 53 REGEX ERRORES
/forward\d+\)/g, /forward\(\d+/g, /turn\d+\)/g, /turn\(\d+/g, /turn-\d+\)/g, /turn\(-\d+/g, /forward\(\)/, /turn\(\)/,
/repeat\(\d+\)/, /if\(\w+\)/, /if\(\w+\):/, /\w+\(-\d+\)/, /\w+\(\)/,
```

Figura 5.28: Expresiones regulares errores en inglés

- Errores de paréntesis ->Falta abrir o cerrar paréntesis (figura 4.39).

- Error de parámetro ->Falta el parámetro (figura 4.40).

- Error estructuras ->Falta el carácter ':' para iniciar una estructura.

- Error tipo de parámetro ->El tipo del parámetro no es el que la función espera (figura 5.29).

- Error condición ->Condición en los condicionales no existente.

- Error identación ->En un testcase o condicional, las instrucciones no están identadas.

- Error oráculo ->El oráculo en la instrucción afirma no existe.

Para mostrar el resultado de cometer estos errores, se han diseñado mensajes para el usuario. En el apartado de Arquitectura, se han mostrado los mensaje de error para algunos errores (figuras 4.39, 4.40 y 4.41). En este apartado se ha añadido también el error de tipo de parámetro (vea la figura 5.29).

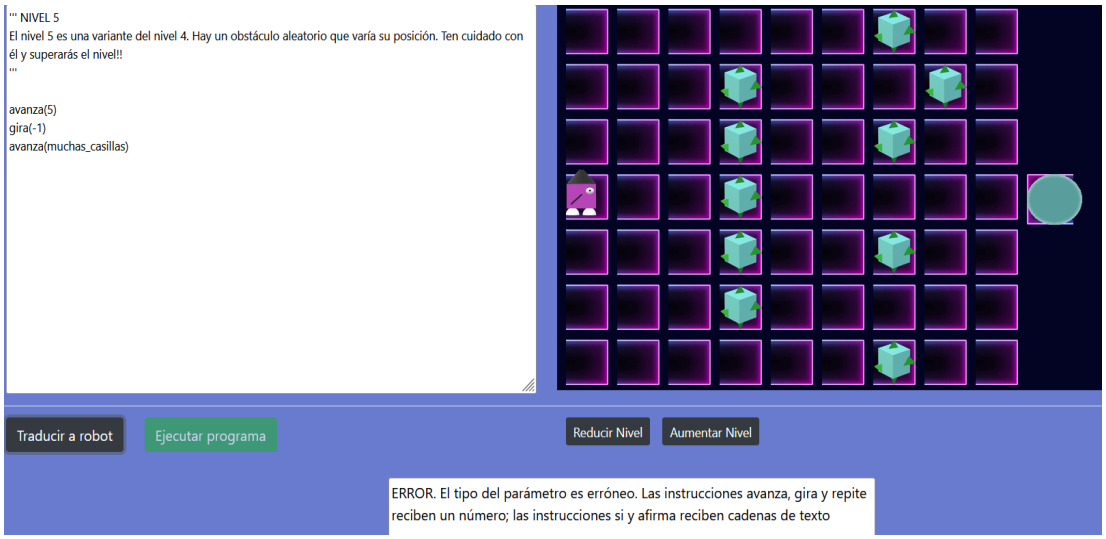


Figura 5.29: Mensaje de error tipo de parámetro

5.2 Versión inglesa

En esta sección se va a explicar con más detalle las características de la versión inglesa de GIPPPY, ya que todas las imágenes que se han mostrado del diseño del juego han sido en la versión castellana.

En la sección de Funciones, se ha explicado el funcionamiento de cambiarIdioma(), que es la función que se activa al hacer click en las imágenes de las banderas de la interfaz principal. La función cambia todos los elementos que contienen texto al idioma seleccionado.

Estas cadenas de texto se almacenan en unos arrays que están en un archivo separado del principal creado a propósito para no llenar demasiado este archivo: array_idiomas.js, donde se almacenan los arrays para cada idioma (ver figuras 5.30 y 5.31).

```
/* Fichero para los arrays con los string necesarios para el juego en diferentes idiomas
 * Si se quiere añadir otro idioma al juego, simplemente crear un nuevo array para el idioma. IMPORTANTE respetar los índices del array.
 */

const array_esp = ["Juego sencillo para introducir a la programación en Python",
"Traducir a robot", "Ejecutar programa", "Reducir Nivel", "Aumentar Nivel",
"''' NIVEL 1" + '\n' + 'Este juego tiene un objetivo: Conseguir que el robot llegue al final del nivel.' + '\n' + '\n' +
'Para conseguirlo, tenemos que ordenar al personaje que haga ciertos movimientos -> Estos movimientos son las instrucciones del programa, que deben ser escritas en este panel.'
+ ' Estas instrucciones deben estar escritas en lenguaje Python, y se traducirán a un lenguaje que el robot entienda para cargarlas y que se ejecuten.'
+ '\n' + '\n' + 'En cada nivel se presentará un reto diferente para aprender las reglas del juego paso a paso. Veremos diferentes obstáculos que se tratan de manera diferente.'
+ '\n' + 'Este primer nivel es el más sencillo, prueba a ejecutar la instrucción...' + '\n' + "'''" + '\n' + '\n' + 'avanza(4)',
"''' NIVEL 2" + '\n' + 'Ya hemos visto el funcionamiento del juego y de la instrucción avanza(x), donde x es el número de casillas que avanza el robot.' + '\n' +
'Este segundo nivel ya es un poco más difícil. Ahora necesitamos algo más. Prueba a ejecutar la secuencia que se ve...' + '\n' + '\n' + '- El robot gira en sentido de las agujas
+ '\n' + '- El robot gira en sentido contrario de las agujas del reloj con la instrucción gira(x). ' + '\n' + "'''" + '\n' + '\n' + 'avanza(5)' + '\n' + 'gira(-1)',
"''' NIVEL 3" + '\n' + 'Introducimos ahora un nuevo obstáculo con pinchos. Es más pequeño pero más peligroso!!. Ejecuta para ver qué pasa si chocas con él...'
+ '\n' + "'''" + '\n' + '\n' + 'avanza(6)',
"''' NIVEL 4" + '\n' + 'Este nivel es más complejo porque la primera fila de obstáculos es aleatoria, por lo que la solución del nivel es diferente cada vez.' + '\n'
+ "'''" + '\n' + '\n' + 'gira(-1)' + '\n' + 'avanza(2)' + '\n' + 'gira(1)',
"''' NIVEL 5" + '\n' + 'El nivel 5 es una variante del nivel 4. Hay un obstáculo aleatorio que varía su posición. Ten cuidado con él y superarás el nivel!!' + '\n' + "'''" + '\n'
'\n' + 'avanza(5)',
"''' NIVEL 6" + '\n' + 'Ten cuidado con este nivel, con solo un movimiento erróneo puedes acabar dentro de un callejón sin salida!!' + '\n' + "'''" + '\n' + '\n' + 'avanza(5)',
"''' NIVEL 7" + '\n' + 'En este nivel se introducen las repeticiones de movimientos (bucles). Para evitar escribir mucho texto, se pueden crear repeticiones.' + '\n'
+ 'Prueba a ejectuar el código que te ofrecemos, y observa con detenimiento los espacios que se dejan antes de las instrucciones dentro de la repetición.' + '\n' +
"OJO: Si no dejas 2 espacios ESTRICTAMENTE, no se va a repetir lo que pongas debajo del repite(x):" + '\n' + "'''" + '\n' + '\n' + 'repite(4):' + '\n'
+ "  avanza(2)" + '\n' + "  gira(1)",
```

Figura 5.30: Array en castellano en el archivo array_idiomas.js

```
const array_eng = ["Easy game to introduce programming in Python",
"Translate to robot", "Execute program", "Reduce Level", "Increase Level",
"''' LEVEL 1" + '\n' + 'This game has only one objective: Help the robot to reach the end of the level by going through the door.' + '\n' + '\n' +
'To do it, we must give the robot instructions to make certain movements. These instructions make up our program and should be written in this panel.'
+ ' The instructions must be written using the language Python, when you click the button -Translate to robot- they will be translated to a language that the robot can understand an
+ '\n' + '\n' + 'This is the first level of this game. At each level, we will face a different challenge and learn the game rules step by step. We will face different types of obsta
+ '\n' + 'This first level is the easiest one, try to execute the instruction below...' + '\n' + "'''" + '\n' + '\n' + 'forward(4)',
"''' LEVEL 2" + '\n' + 'We have already seen how the game and the instruction forward(x) work. When calling forward(x), x is the number of steps the robot moves on x axis.' + '\n' +
'This second level is a bit more difficult. We will add an instruction to turn the robot around.' + '\n' + '\n' + '- Robot turns clockwise with instruction turn(-x). '
+ '\n' + '- Robot turns counter clockwise with instruction turn(x).' +'\n' + 'Try to execute the instructions sequence...\n'  + "'''" + '\n' + '\n' + 'forward(5)' + '\n' + 'turn(-1)
"''' LEVEL 3" + '\n' + 'In this level we introduce a new spiked obstacle. It is smaller but more dangerous!!. Execute the given sequence to see what happens when you collide with it
+ '\n' + "'''" + '\n' + '\n' + 'forward(6)',
"''' LEVEL 4" + '\n' + 'This level is even more difficult because the first obstacles row is placed in random order, so the solution for this level is different each time you play tl
+ "'''" + '\n' + '\n' + 'turn(-1)' + '\n' + 'forward(2)' + '\n' + 'turn(1)',
"''' LEVEL 5" + '\n' + 'Level 5 is a modification of level 4. There is one random obstacle that changes its position. Stay alert and you will succeed!!' + '\n' + "'''" + '\n' +
'\n' + 'forward(5)',
"''' LEVEL 6" + '\n' + "Take care with this level, by executing only one wrong movement you may find yourself in a dead end!!" + '\n' + "'''" + '\n' + '\n' + 'forward(5)',
"''' LEVEL 7" + '\n' + 'In this level we are introducing loops. Repetitions are usefull when we need to repeat the same instruction several times. A loop is a control structure whicl
+ 'Try to execute the given code, be aware of the spaces (identation) before the instructions that are in the loop body.' + '\n' +
"IMPORTANT: If you forget to put EXACTLY 2 spaces in from of the instructions, they are not part of the loop body and hence they will not be repeated." + '\n' + "'''" + '\n' + '\n' +
+ "  forward(2)" + '\n' + "  turn(1)",
```

Figura 5.31: Array en inglés en el archivo array_idiomas.js

Lo más importante que hay que tener en cuenta de los arrays es que la posición de los elementos debe coincidir. Es decir, si las instrucciones del nivel 5 están en la posición 16 en el array_esp, en el resto de arrays que se creen para otros idiomas deben respetarse los índices.

Ahora se van a mostrar algunas imágenes de la interfaz del juego en su versión inglesa, para ver cómo se ha portado absolutamente todo al inglés (ver figuras 5.32 y 5.33 que contienen ejemplos de la versión inglesa del juego).

Figura 5.32: Interfaz nivel 1 en inglés

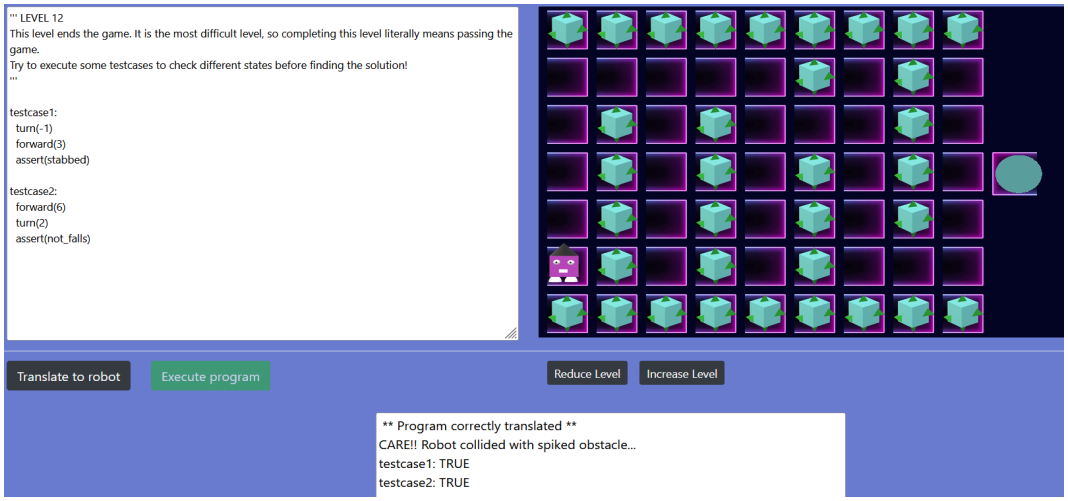


Figura 5.33: Testcases y colisión con pinchos en inglés

5.3 Lenguajes usados

En esta sección se presentan de manera resumida los lenguajes usados para la parte de la implementación de este proyecto.

5.3.1. Javascript

El lenguaje empleado para la lógica, como se ha visto en el apartado de funciones, es Javascript.

Javascript es un lenguaje de programación ligero, de compilación just-in-time (o compilación dinámica, es decir, el código se compila en tiempo de ejecución). Es el único lenguaje de programación que funciona de forma nativa (sin necesidad de compilación) en los navegadores, por ello, su uso principal es el desarrollo de página webs, combinado con HTML y CSS.

Javascript sigue un modelo asíncrono y no bloqueante, con un loop de eventos implementado con un único thread para sus interfaces de entrada/salida. Es decir, la fina-

lización de una operación es notificada al programa principal, pero el procesado de la respuesta se hará en otro momento posterior.

En este trabajo se necesita que la respuesta sea procesada inmediatamente después de la finalización de la ejecución, por eso se han empleado las *Promises* de Javascript para garantizar la sincronía de estas operaciones (ver figura 5.16, donde se usa una *Promise* para esperar correctamente a la finalización de la función).

Actualmente, Javascript es un lenguaje popular por su sencillez y por su gran cantidad de librerías (jQuery, AJAX, Node.js, etc.) [16]. Se ha elegido este lenguaje para la implementación del juego debido al mayor conocimiento del este lenguaje gracias a la formación adquirida en la carrera. Al tener más experiencia con Javascript en el desarrollo web, se han enfocado los esfuerzos en mejorar las mecánicas del juego al máximo y en transmitir de la mejor manera los conceptos de Python.

5.3.2. Jest

Jest es un framework de testeo para Javascript. Es sencillo y está diseñado para el testeo de proyectos web con gran cantidad de líneas de código. Facebook es una de las empresas más grandes que valida su código usando el framework Jest [17].

Configurar Jest es bastante sencillo, no requiere nada más que una instalación (figura 5.34) y ya es posible usarlo en un proyecto.

```
$ npm install --save-dev jest
```

Figura 5.34: Comando para la instalación del framework Jest

Una vez instalado, podemos crear un archivo de test para validar cualquier función con entradas y salidas. La extensión soportada para los archivos de test es .test.js.

Una vez se tengan unos tests correctos para una función, hay que ejecutarlos en la consola con el comando *jest*. Si queremos también un informe con resultados de cobertura, podemos ejecutar el comando *jest –coverage*. En la consola veremos el resultado de los tests y una tabla con medidas, tal y como se ve en la figura 5.35.

Las métricas que genera el informe de Jest son las siguientes:

- **%Stmts (Staments):** Representa el porcentaje de estamentos ejecutables que se han cubierto con los tests.

- **%Branch (Ramas):** Representa el porcentaje de ramas del código que han cubierto con los tests. Las ramas del código se generan cuando hay condiciones, lo que divide el flujo del programa en ramas.

- **%Funcs (Functions):** Representa el porcentaje de llamadas a funciones que se cubren con los tests. Si la función delega cierto trabajo a otras funciones, hay que evaluar todas las posibilidades para cubrir esas llamadas.

- **%Lines (Líneas):** Corresponde con el porcentaje de líneas del código que se han cubierto con los tests. En caso de que no haya un 100 % de *Lines coverage*, se muestran al lado las líneas que no se ha cubierto con los tests.

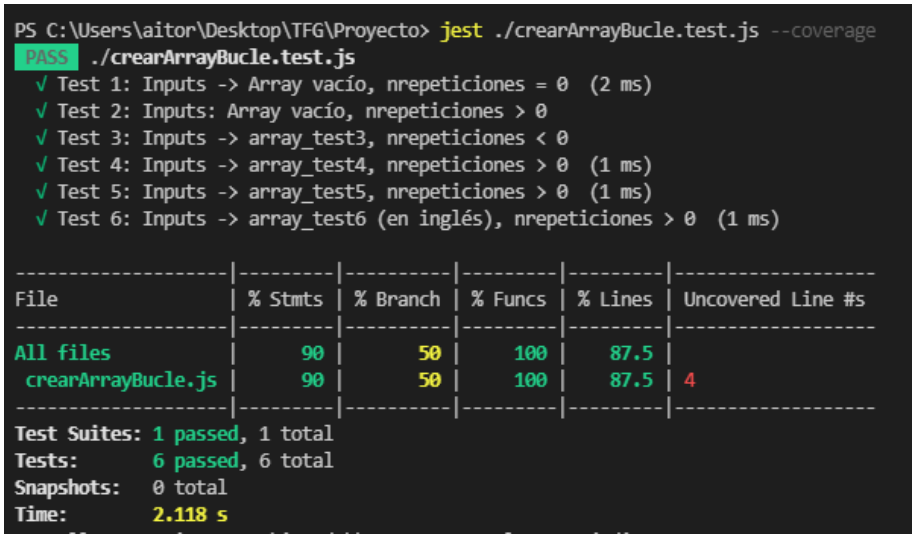


Figura 5.35: Informe de tests con Jest

En la sección de Testeo de código se explicará su uso en este trabajo, ya que Jest ha sido elegido para realizar los test unitarios de las funciones de este proyecto, de forma que se han encontrado algunos errores que han sido corregidos gracias a estos tests.

5.4 Testeo del código

En esta sección se presentan los procedimientos que se han seguido para validar el código que da funcionamiento a GIPPPY.

Antes se explicó el Framework Jest, que es un framework diseñado para testear código Javascript. Jest se ha usado para testear de manera unitaria algunas funciones. No ha sido posible testear todas las funciones mediante Jest, ya que Jest no puede validar las funciones que tienen impacto visual, es decir, las funciones cuyo resultado es modificar la interfaz gráfica del juego (añadir, borrar o modificar elementos). Para estas funciones, la validación se ha hecho a través de recargar la página web (para ver los cambios) y las herramientas del navegador para desarrolladores web, principalmente mediante escrituras de variables en la consola.

5.4.1. Tests con Jest

La mayoría de funciones de este juego tienen impacto visual (es decir, sobre elementos del HTML), por lo que como se ha explicado arriba, no se puede usar Jest para testearlas.

Aún así, hay algunas funciones que se han podido adaptar para tener unas entradas y salidas y poder realizar unos test unitarios con Jest.

1. **cambioNivel():** En la figura 5.36, se muestra el código adaptado de la función cambio_nivel() para poder ser testeada mediante Jest. Es importante fijarse en la última línea de código que está fuera de los límites de la función. Mediante module.exports = cambio_nivel permitimos que el archivo de test pueda llamar de manera remota a esta función con el nombre indicado.

   Tras esto, hay que crear el archivo de test, cuya extensión es archivo.test.js. Una vez creado, hay que importar la función a testear (ver figura 5.37). Esto se hace a través de la instrucción require (./nombre_fichero_javascript).

```
// nivel_actual --> Nivel actual del juego.
// mas_menos = 0 (decrementar nivel) ;  mas_menos = 1 (aumentar nivel)
function cambio_nivel(nivel_actual, mas_menos) {

    const num_niveles = 12;

    switch(mas_menos) {
        case 0:
            //cargar nivel anterior
            if (nivel_actual == 1){
                return -1;
            } else {
                nivel_actual--;
                //cargar_nivel(nivel_actual, false);
                return nivel_actual;
            }
        case 1:
            //cargar nivel siguiente
            if (nivel_actual == num_niveles){
                return -1;
            } else {
                nivel_actual++;
                //cargar_nivel(nivel_actual, false);
                return nivel_actual;
            }
    }
}

module.exports = cambio_nivel;
```

Figura 5.36: Código de la función cambio_nivel antes de los tests

```
const cambio_nivel = require('./cambioNivel');
const aumentar = 1;
const decrementar = 0;

test('Se espera que devuelva -1 al intentar decrementar de nivel estando en el primero', () => {
    const nivel_actual = 1;
    expect(cambio_nivel(nivel_actual, decrementar)).toBe(-1);
});

test('Va a devolver 3 porque estamos en el nivel 4 y se ha solicitado un decremento de nivel', () => {
    const nivel_actual = 4;
    expect(cambio_nivel(nivel_actual, decrementar)).toBe(3);
});

test('Va a devolver 9 porque estamos en el nivel 8 y se ha solicitado un aumento de nivel', () => {
    const nivel_actual = 8;
    expect(cambio_nivel(nivel_actual, aumentar)).toBe(9);
});

test('Va a devolver -1 porque estamos en el nivel 12, que es el último, y se ha solicitado un aumento de nivel', () => {
    const nivel_actual = 12;
    expect(cambio_nivel(nivel_actual, aumentar)).toBe(-1);
});

Debug
test('Va a devolver -1 porque estamos en un nivel fuera del número de niveles que se han diseñado', () => {
    const nivel_actual = 13;
    expect(cambio_nivel(nivel_actual, decrementar)).toBe(-1);
});
```

Figura 5.37: Tests de la función cambio_nivel con fallo

A partir de este paso, se puede proceder a crear los tests. Para ello, hay que usar la función test(identificador, función). El identificador sirve para identificar los resultados de los tests. En la figura 5.37, se pueden ver los primeros tests para la función cambio_nivel.

Respecto a los tests que se muestran en la figura 5.37, el primer test verifica que si se intenta reducir el nivel estando en el nivel 1, el resultado es -1. El segundo test verifica que estando en el nivel 4, si se decrementa el nivel el resultado es 3; el tercer test verifica que si aumentamos el nivel estando en el nivel 8 el resultado es 9, y así sucesivamente con el resto de tests. Se ha buscado probar todos los casos posibles con los tests para conseguir un coverage del 100 % sobre el código de la función.

Es importante destacar el último test, que sale en rojo porque Jest ha detectado que el test es erróneo antes de ejecutarlo. Si el nivel no entra dentro de los límites (el juego tiene 12 niveles), y se intenta decrementar el nivel, la función no devuelve error. De hecho, si se analiza el código de la función (figura 5.36), los límites superiores e inferiores del número de niveles no están contemplados en el código.

Realmente, en el juego no se puede estar nunca fuera de los límites de los niveles, pero para conseguir que la función sea testeable, se ha modificado la función de la siguiente manera (ver figura 5.38):

```
// nivel_actual --> Nivel actual del juego.
// mas_menos = 0 (decrementar nivel) ;  mas_menos = 1 (aumentar nivel)
function cambio_nivel(nivel_actual, mas_menos) {

    const num_niveles = 12;

    switch(mas_menos) {
        case 0:
            //cargar nivel anterior
            if (nivel_actual <= 1 || nivel_actual >= num_niveles){
                return -1;
            } else {
                nivel_actual--;
                //cargar_nivel(nivel_actual, false);
                return nivel_actual;
            }
        case 1:
            //cargar nivel siguiente
            if (nivel_actual <= 1 || nivel_actual >= num_niveles){
                return -1;
            } else {
                nivel_actual++;
                //cargar_nivel(nivel_actual, false);
                return nivel_actual;
            }
    }
}

module.exports = cambio_nivel;
```

Figura 5.38: Código de la función cambio_nivel tras detectar un error con Jest

Al cambiar las comparaciones exactas (==) por comparaciones de rango (<=, >=), los niveles fuera de rango quedan contemplados, pudiendo testear ahora la función completa.

```
const cambio_nivel = require('./cambioNivel');
const aumentar = 1;
const decrementar = 0;

test('Se espera que devuelva -1 porque estamos en un nivel por debajo del número de niveles diseñado', () => {
    const nivel_actual = -2;
    expect(cambio_nivel(nivel_actual, aumentar)).toBe(-1);
});

test('Va a devolver -1 porque estamos en un nivel por encima del número de niveles que se han diseñado', ()  => {
    const nivel_actual = 13;
    expect(cambio_nivel(nivel_actual, decrementar)).toBe(-1);
});

test('Se espera que devuelva -1 al intentar decrementar de nivel estando en el primero', () => {
    const nivel_actual = 1;
    expect(cambio_nivel(nivel_actual, decrementar)).toBe(-1);
});

test('Va a devolver 3 porque estamos en el nivel 4 y se ha solicitado un decremento de nivel', ()  => {
    const nivel_actual = 4;
    expect(cambio_nivel(nivel_actual, decrementar)).toBe(3);
});

test('Va a devolver 9 porque estamos en el nivel 8 y se ha solicitado un aumento de nivel', ()  => {
    const nivel_actual = 8;
    expect(cambio_nivel(nivel_actual, aumentar)).toBe(9);
});

test('Va a devolver -1 porque estamos en el nivel 12, que es el último, y se ha solicitado un aumento de nivel', ()  => {
    const nivel_actual = 12;
    expect(cambio_nivel(nivel_actual, aumentar)).toBe(-1);
});
```

Figura 5.39: Test unitarios corregidos para la función cambio_nivel

En la figura 5.39 se muestran los tests finales para la función cambio_nivel. Se ha añadido un test más para verificar el límite inferior (niveles por debajo de 1) para cubrir todas las ramas del código. Ahora ningún test sale con errores.

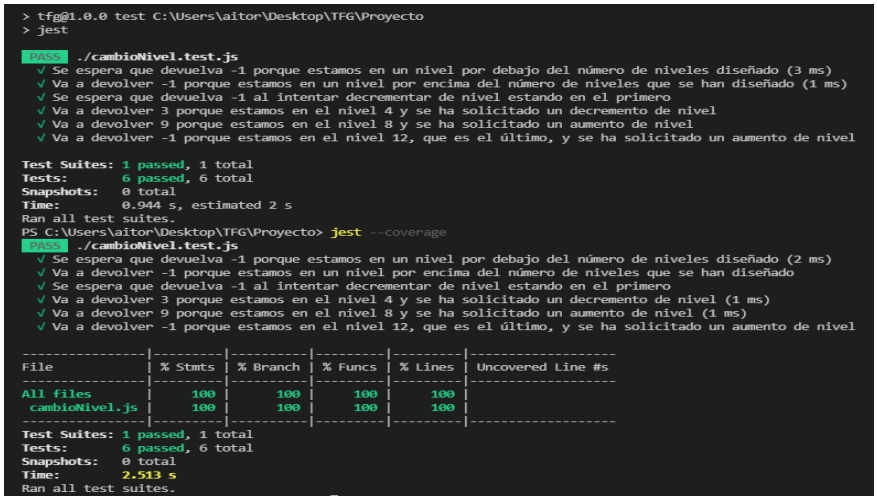


Figura 5.40: Informe test unitarios para cambio_nivel

Para finalizar el proceso de test, en la figura 5.40 se adjunta el informe que ofrece Jest con estadísticas para los tests unitarios. En el informe se muestran los resultados de los tests (en este caso todos), y una tabla con las estadísticas de coverage, que ya se han explicado antes en el apartado 5.3.2.

2. **crearArrayBucle**: La función crearArrayBucle se ha podido validar usando Jest. Esta función se llama cuando aparece una estructura repite(x) en el panel de código, y el objetivo de la función es multiplicar el array de instrucciones por el número de repeticiones indicado como parámetro en la función repite(x). En la figura 5.41 se muestra el código adaptado para poder validar la función con Jest.

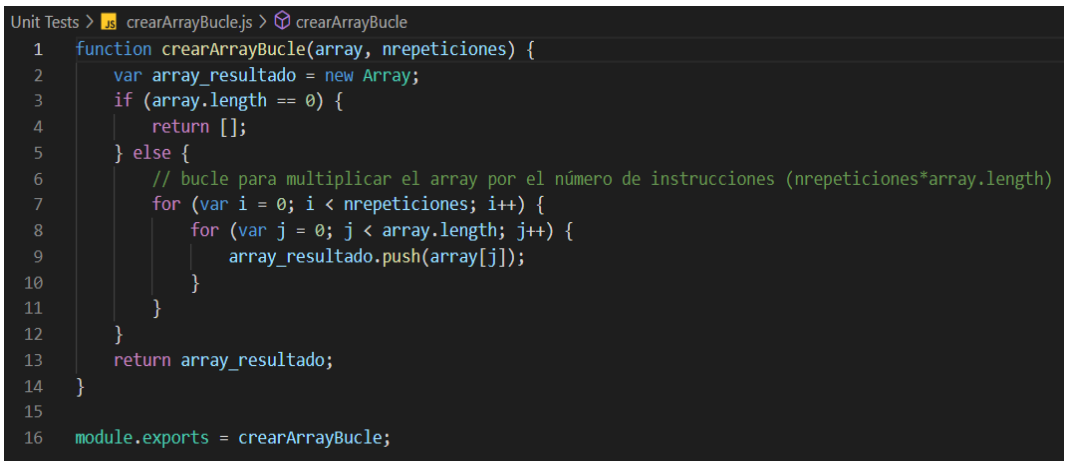


Figura 5.41: Código adaptado para la función crearArrayBucle

Los tests unitarios que se han diseñado para validar la función están en la figura 5.42. Se han probado todas las ramas de código posibles. En los 2 primeros tests, el array de instrucciones está vacío, de forma que validamos el primer return (devuelve array vacío). El resto de tests (menos el tercero, que prueba un valor negativo de repeticiones), validan la funcionalidad esperada.

```
crearArrayBucle.test.js > ...
1   const crear_array_bucle = require('./crearArrayBucle');
2   var nrepeticiones;
3   const array_test3 = ["avanza(3)", "gira(1)"];
4   const array_test4 = ["avanza(10)"];
5   const array_test5 = ["gira(-3)", "gira(2)", "avanza(1)"];
6   const array_test6 = ["forward(12)", "turn(-4)"];
7
8
9   test('Test 1: Inputs -> Array vacio, nrepeticiones = 0 ', () => {
10      nrepeticiones = 0;
11      expect(crear_array_bucle([], nrepeticiones)).toStrictEqual([]);
12  });
13
14  test('Test 2: Inputs: Array vacio, nrepeticiones > 0', () => {
15      nrepeticiones = 5;
16      expect(crear_array_bucle([], nrepeticiones)).toStrictEqual([]);
17  });
18
19  test('Test 3: Inputs -> array_test3, nrepeticiones < 0 ', () => {
20      nrepeticiones = -5;
21      expect(crear_array_bucle(array_test3, nrepeticiones)).toStrictEqual([]);
22  });
23
24  test('Test 4: Inputs -> array_test4, nrepeticiones > 0 ', () => {
25      nrepeticiones = 3;
26      expect(crear_array_bucle(array_test4, nrepeticiones)).toStrictEqual(["avanza(10)", "avanza(10)", "avanza(10)"]);
27  });
28
29  test('Test 5: Inputs -> array_test5, nrepeticiones > 0 ', () => {
30      nrepeticiones = 2;
31      expect(crear_array_bucle(array_test5, nrepeticiones)).toStrictEqual(["gira(-3)", "gira(2)", "avanza(1)", "gira(-3)", "gira(2)", "avanza(1)"]);
32  });
33
34  test('Test 6: Inputs -> array_test6 (en inglés), nrepeticiones > 0 ', () => {
35      nrepeticiones = 3;
36      expect(crear_array_bucle(array_test6, nrepeticiones)).toStrictEqual(["forward(12)", "turn(-4)", "forward(12)", "turn(-4)", "forward(12)", "turn(-4)"]);
37  });
```

Figura 5.42: Tests unitarios para la función crearArrayBucle

El resultado de estos tests ha sido exitoso (ver figura 5.43). No ha sido necesario modificar nada de la función.

```
PS C:\Users\aitor\Desktop\TFG\Proyecto> jest ./crearArrayBucle.test.js --coverage
 PASS  ./crearArrayBucle.test.js
  √ Test 1: Inputs -> Array vacío, nrepeticiones = 0  (2 ms)
  √ Test 2: Inputs: Array vacío, nrepeticiones > 0 (1 ms)
  √ Test 3: Inputs -> array_test3, nrepeticiones < 0  (1 ms)
  √ Test 4: Inputs -> array_test4, nrepeticiones > 0  (1 ms)
  √ Test 5: Inputs -> array_test5, nrepeticiones > 0
  √ Test 6: Inputs -> array_test6 (en inglés), nrepeticiones > 0

--------------------|---------|----------|---------|---------|-------------------
File                | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
--------------------|---------|----------|---------|---------|-------------------
All files           |     100 |      100 |     100 |     100 |
 crearArrayBucle.js |     100 |      100 |     100 |     100 |
--------------------|---------|----------|---------|---------|-------------------
Test Suites: 1 passed, 1 total
Tests:       6 passed, 6 total
Snapshots:   0 total
Time:        0.838 s, estimated 2 s
```

Figura 5.43: Informe de coverage de la función crearArrayBucle

5.5 Servidor y Repositorio GITHUB

Para concluir con el apartado de Implementación, se va a comentar el mantenimiento que hay hasta el momento para el juego.

Por un lado, el juego se ha subido a un servidor disponible de la Universidad Politécnica de Valencia para poder ser visitado por cualquiera a través de una URL. La dirección para acceder al juego es https://robot.testar.org/ [18] (figura 5.44).

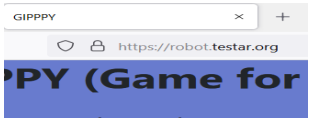


Figura 5.44: URL de GIPPPY

Se espera que el juego se pueda seguir manteniendo tras la finalización de este TFG. Por lo tanto, se ha subido el código fuente a un repositorio en GitHub [19]. El repositorio GIPPPY ha sido la manera de mostrar el progreso e ir subiendo las diferentes versiones del código, para que en un futuro se pueda retomar el desarrollo del juego e ir ampliando los niveles, añadir más funcionalidades, etc.

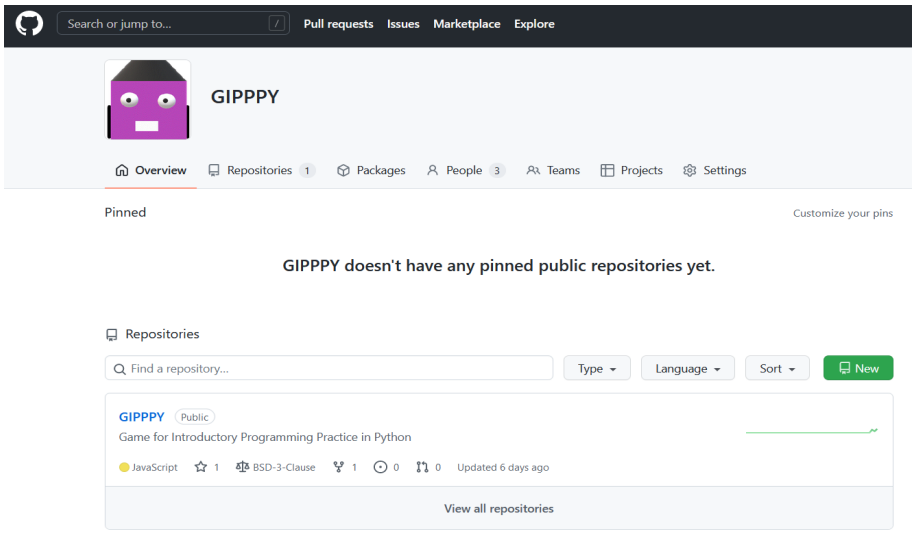


Figura 5.45: Repositorio GIPPPY GitHub

Los commits que se han hecho al repositorio explican con bastante claridad el progreso hecho en cada versión. En la figura 5.45 se muestra el repositorio en la web de GitHub, y en la figura 5.46 se muestra el historial de *commits* que se han hecho sobre el repositorio.

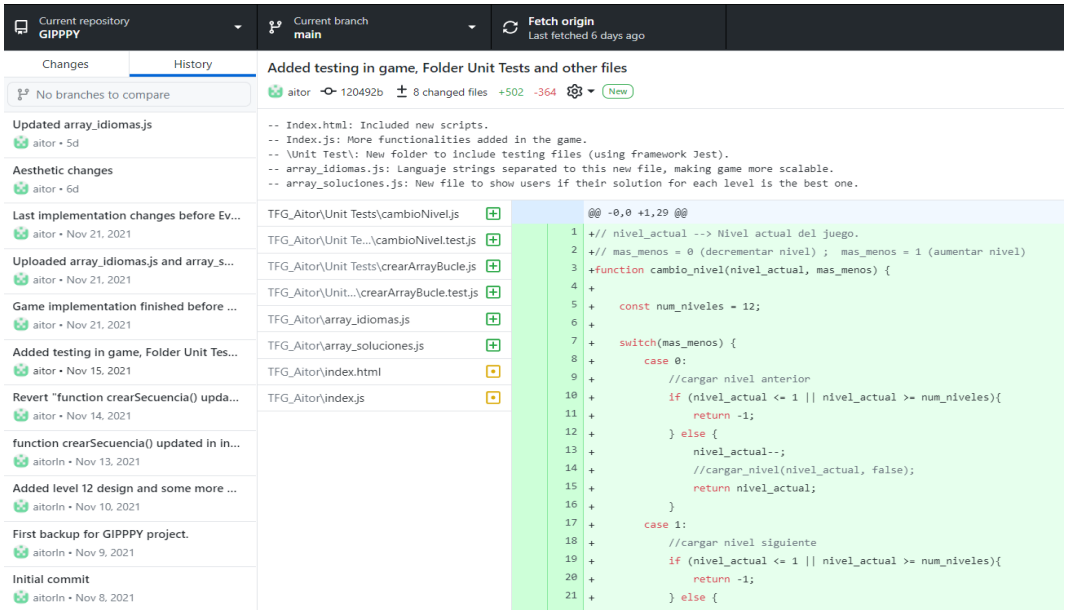


Figura 5.46: Historial de commits en el repositorio

En el repositorio se encuentran los archivos necesarios para poder cargar todo el proyecto GIPPPY, y a parte se han subido en la carpeta Unit Tests los archivos que se han creado para los test unitarios en el apartado 5.4 (ver figura 5.47).

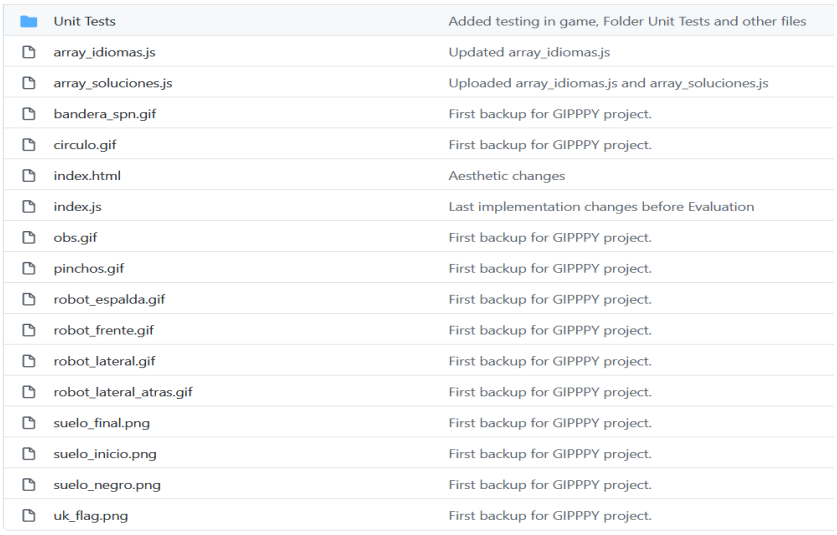


Figura 5.47: Archivos subidos al repositorio

También se ha generado en el repositorio un archivo que contiene la licencia que tiene el código del programa, aunque también se ha incluido el texto de la licencia como comentario dentro del mismo código.

29 lines (23 sloc) | 1.48 KB

```
BSD 3-Clause License

Copyright (c) 2021, GIPPPY
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this
   list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice,
   this list of conditions and the following disclaimer in the documentation
   and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
   contributors may be used to endorse or promote products derived from
   this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
```

Figura 5.48: Licencia BSD3-clause license del trabajo

La licencia elegida ha sido la licencia BSD3-clause license (ver figura 5.48).

CAPÍTULO 6
Evaluación y validación del juego

En este capítulo se va a presentar la manera en la que se ha validado el juego. Se ha seguido el modelo en V para la validación, como se explican a continuación:

- Test unitarios: Como nivel más bajo de la V, ya se ha explicado en el capítulo 5.4 que se han testeado unitariamente las funciones que se han podido adaptar para usar el framework Jest.

- Test de integración: En el siguiente nivel, se ha validado la integración comprobando las llamadas entre funciones, es decir, la arquitectura del juego. La manera de hacerlo ha sido mediante el uso de las herramientas de desarrollador del navegador, siguiendo la evolución del valor de ciertas variables.

- Test de sistema: El sistema completo se ha validado tras subir cada commit al repositorio de GIPPPY. Cuando se sube una nueva versión funcional al repositorio, se actualiza el servidor y se hacen pruebas del sistema completo jugando al juego.

- Test de aceptación: Este capítulo trata principalmente de estos tests. Los tests de aceptación consisten en enviar el producto final a los clientes, es decir, a las personas que van a jugar al juego para aceptarlo.

Para la aceptación y evaluación del juego, se ha creado un cuestionario [20] en castellano y otro en inglés [21], diseñado para las personas que se han seleccionado para jugar al juego.

El objetivo del formulario es obtener una evaluación de las personas que han sido seleccionadas para evaluar el juego. Mediante esta evaluación, se puede ver qué niveles han sido los más complicados por diseño, las estructuras de programación más complicadas, etc. El resultado de las preguntas del cuestionario se presenta a continuación en las siguientes secciones.

6.1 Perfil de los participantes

Han participado en la evaluación un total de 30 personas entre los 2 cuestionarios (en castellano y en inglés). Los perfiles de las personas que han respondido el cuestionario son variados, pero aún así se ha elegido como perfil predominante un perfil joven adolescente, de entre los 18-23 años, que es la edad más habitual con la que se suelen inscribir a los grados/ciclos/cursos para aprender a programar.

En la figura 6.1 se muestra la gráfica de resultados, donde predominan las personas de 22 años con un 35,7 %, y justo después las de 18 años con un 21,4 %.

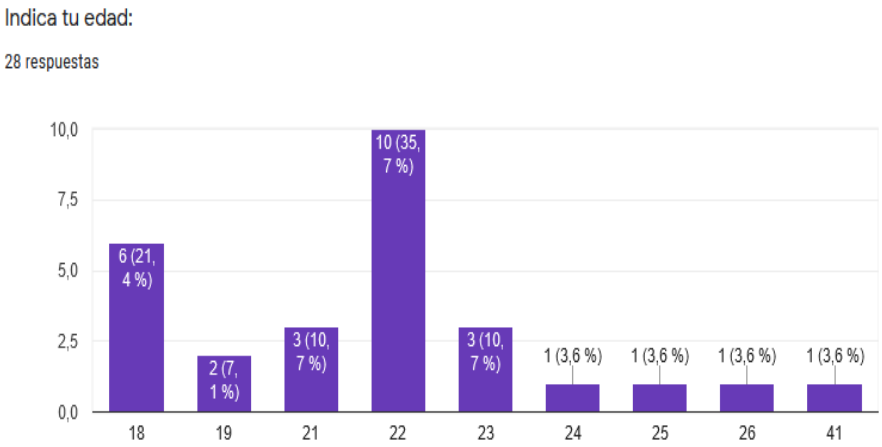


Figura 6.1: Gráfica Edad del cuestionario

Respecto al género de los participantes, ha habido diversidad, pero han resuelto el cuestionario un 71,4 % de hombres, un 25 % de mujeres y un 3,6 % otros géneros (ver figura 6.2).

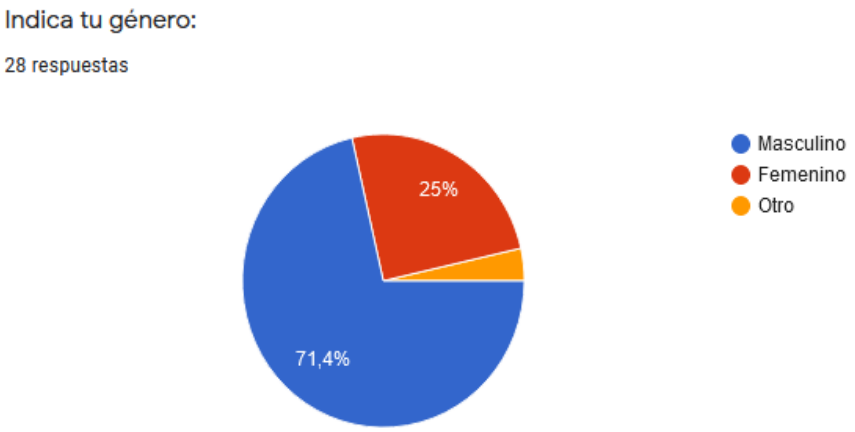


Figura 6.2: Gráfica Género del cuestionario

Dentro del perfil de los participantes, era importante conocer la experiencia de los mismos en programación y también sobre el lenguaje Python. En este aspecto, ha habido mucha diversidad (ver figuras 6.3 y 6.4).

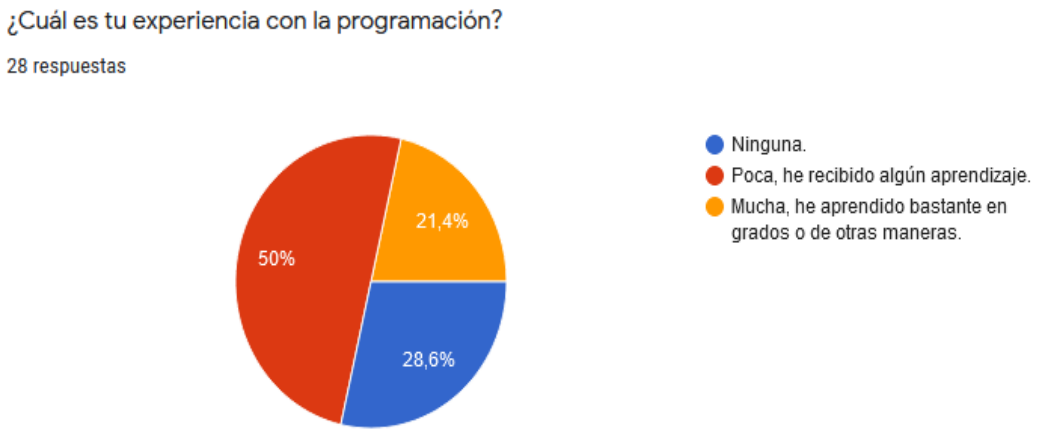


Figura 6.3: Gráfica Experiencia en programación del cuestionario

Respecto a la experiencia en programación, el 50 % de los participantes han respondido que tienen poca experiencia en programación, pero habiendo aprendido ya las bases. Luego, entre ninguna experiencia y mucha experiencia, ha habido prácticamente igualdad en el número de participantes (cerca del 25 %).

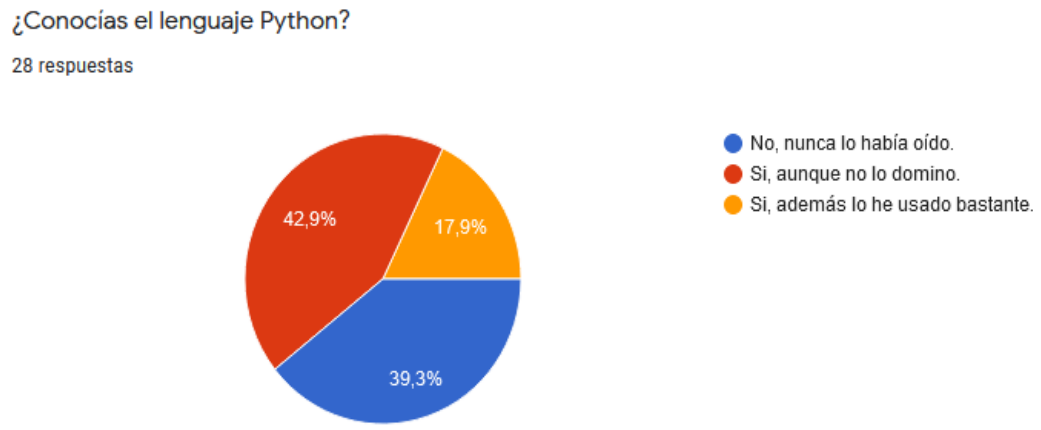


Figura 6.4: Gráfica Experiencia en Python del cuestionario

Y respecto a la experiencia en el lenguaje Python, la mayoría de participantes han oído hablar del lenguaje, pero no lo han usado. Justo después, un 39,3 % no conocía la existencia del lenguaje Python. Por último, un 17,9 % es experto en programar con Python.

6.2 Preguntas técnicas

Una vez ya se tiene la información personal de cada participante, se presentan las preguntas técnicas sobre el juego.

Para comenzar, se preguntó al participante si ha sido capaz de superar todos los niveles, y en caso de no haber superado todos ellos, que indicara hasta qué nivel había podido superar. En la figura 6.5 se adjunta el resultado de la primera pregunta, donde el 71,4 % de usuarios han sido capaces de superar todos los niveles. El 28,6 % restante no ha sido capaz, que es el mismo porcentaje de personas que no tienen ninguna experiencia en programación, por lo que parece que hay una relación entre estos 2 resultados.

Figura 6.5: Gráfica superar preguntas del cuestionario

A las personas que no han sido capaces de superar los 12 niveles, se les preguntó por los niveles que fueron capaces de superar o hasta qué nivel pudieron completar. En la figura 6.6 se muestran las 8 respuestas que ha habido en esta pregunta.

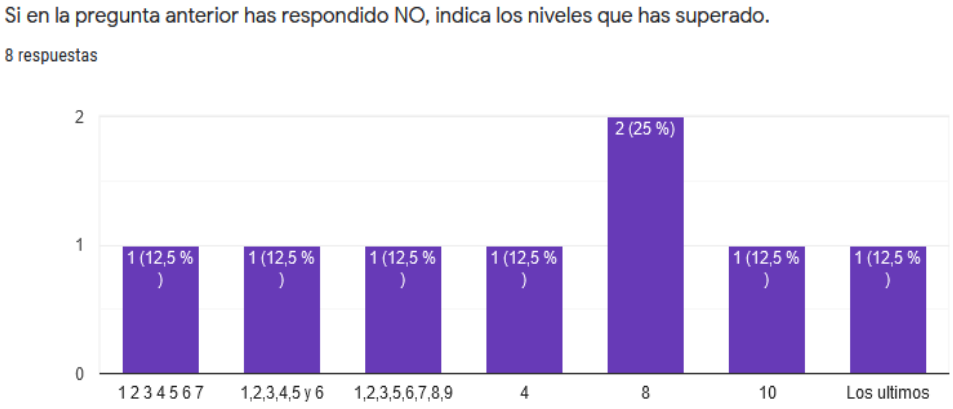


Figura 6.6: Gráfica niveles no superados del cuestionario

A excepción de una persona que se ha quedado en el nivel 4, el resto ha sido capaz de superar sin problemas hasta el nivel 6. A partir del nivel 6, que ya se introducen los bucles, una persona no pudo seguir superando niveles. Pero sobre todo, los últimos niveles han causado más problemas, a partir del nivel 9-10 ya hay más personas que no han podido seguir. Esto se debe probablemente al aumento de dificultad en estos niveles, y a la introducción de las estructuras condicionales y los casos de prueba.

Para saber más de la opinión de los participantes, se ha preguntado sobre cuál les han parecido el nivel más entretenido y el nivel más difícil del juego. Con esta pregunta, se pretende saber si la intención que se ha tenido al diseñar los niveles se ha plasmado bien en los jugadores. Los resultados están en la figura 6.7.

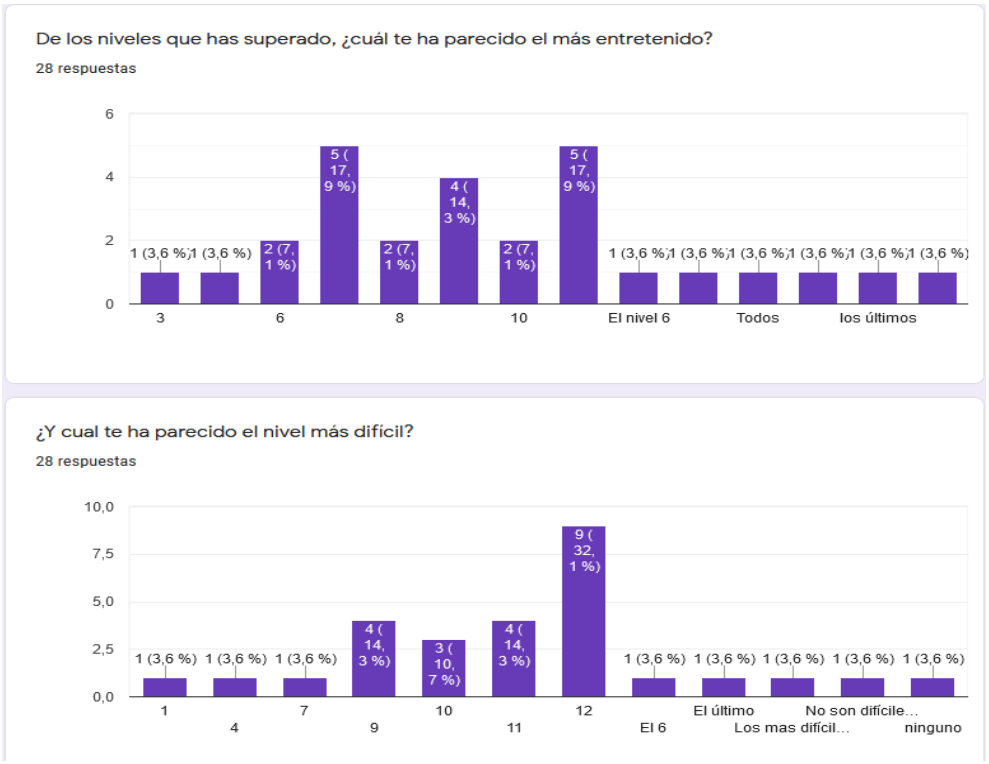


Figura 6.7: Gráfica valoración de niveles del cuestionario

Los niveles más divertido según los encuestados son el nivel 7, 9 y 11. Esta respuesta era la esperada, ya que la razón principal de esta respuesta es que los jugadores venían de superar 6 niveles usando las instrucciones base avanza() y gira(), y en el nivel 7 se introducen los bucles, lo que supone una dificultad extra para ellos. En el nivel 9 se introducen los condicionales, y en el 11 los casos de prueba.

El nivel más difícil por mayoría es el 12, que es lo que se esperaba. El nivel 12 tiene el diseño más difícil del juego, y a parte se pueden usar todas las estructuras diseñadas para el juego, lo que lo convierte en el nivel más complicado.

6.3 Preguntas UMUX

Las preguntas UMUX (Usability Metric for User Experience) [32] son unas preguntas muy útiles para evaluar la satisfacción y la usabilidad de cualquier proyecto orientado a usuarios. La manera de responder estas preguntas es mediante una escala Likert de 1-5 , donde el 1 significa estar totalmente en desacuerdo, y el 5 significa estar totalmente de acuerdo.

Se han añadido las siguientes preguntas UMUX, siguiendo el ejemplo del documento que se ha añadido como referencia [31].

Los resultados de las preguntas se encuentran en las figuras 6.8 y 6.9.

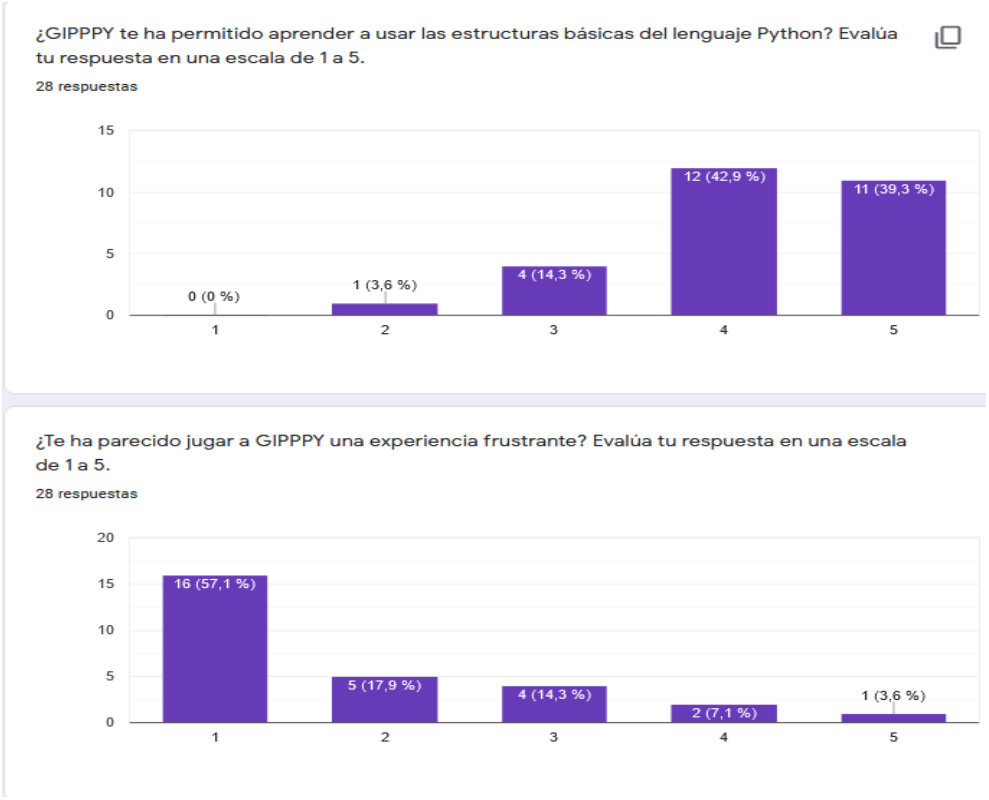


Figura 6.8: Gráfica primeras preguntas UMUX del cuestionario

La primera pregunta se centra en cómo se ha transmitido el uso de estructuras de Python con el juego. Más de un 80 % ha respondido con un 4 y un 5, por lo que parece que el juego consigue que la mayoría de personas entienda las características que tiene el lenguaje Python a la hora de programar.

La segunda pregunta es más general, y pretende conocer la satisfacción de los usuarios jugando a GIPPPY. La mayoría de encuestados han encontrado GIPPPY como una experiencia buena en la que divertirse jugando, pero algunos de ellos han sentido que jugar a GIPPPY ha sido algo frustrante. Esto puede deberse a la dificultad de algunos niveles, y a que el pensamiento computacional no es algo que se obtiene rápidamente y sin esfuerzo.

La tercera pregunta pretende conocer la dificultad que han tenido los usuarios entendiendo el funcionamiento de GIPPPY. Excepto un participante, el resto han encontrado GIPPPY un juego sencillo de entender. Esto es precisamente lo que se buscaba al diseñar GIPPPY, ya que va orientado a alumnos que quieren empezar a programar con Python.

Para terminar, la última pregunta trata sobre el tiempo que han necesitado los encuestados para superar los niveles del juego. En esta pregunta hay mayor diversidad de respuestas. Esto puede deberse a la diversidad de perfil de jugadores que han respondido el cuestionario. A los jugadores más experimentados, los niveles los han superado en poco tiempo, pero los jugadores sin experiencia ninguna en programación o en Python habrán ocupado más tiempo para superar los niveles.

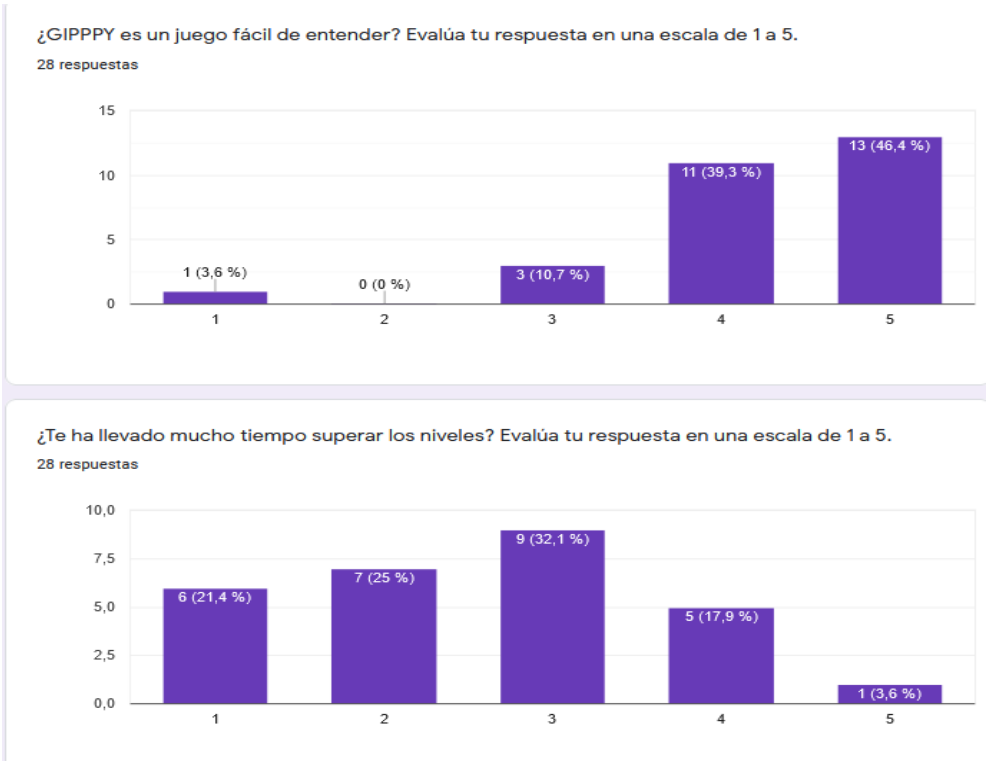


Figura 6.9: Gráfica últimas preguntas UMUX del cuestionario

6.4 Preguntas generales

Ya para terminar el cuestionario, se han añadido tres preguntas genéricas sobre el juego. Las preguntas están orientadas a una evaluación general de GIPPPY y de sus objetivos.

La primera pregunta es para ver si los usuarios, una vez ya han probado el juego, recomendarían jugar a GIPPPY a alguien que quiera aprender Python. Según los resultados obtenidos, un 92,9 % ha decidido que GIPPPY es una buena herramienta para aprender las bases de programación Python. La gráfica con los resultados están en la figura 6.10.

Figura 6.10: Gráfica recomendaciones GIPPPY del cuestionario

La segunda pregunta busca que los participantes opinen sobre GIPPPY como herramienta complementaria para las universidades, escuelas, etc. Principalmente, para demostrar que los elementos gamificados motivan a las personas a querer seguir aprendiendo mientras se divierten. Los resultados son idénticos a los de la pregunta anterior, ya que en parte ambas preguntas están relacionadas (ver figura 6.11).

Figura 6.11: Gráfica aprendizaje y GIPPPY en el cuestionario

Y la última pregunta del cuestionario está más orientada a conocer la sensación que les ha producido a los usuarios jugar a GIPPPY. Como el objetivo de GIPPPY es divertir a los jugadores a la vez que aprenden, es importante conocer si les ha parecido divertido el juego. Los resultados son similares a las preguntas anteriores, aunque esta vez el 89,3 % de los encuestados se han divertido con GIPPPY (ver figura 6.12).

Figura 6.12: Gráfica diversión con GIPPPY del cuestionario

Los resultados obtenidos en todo el cuestionario evidencian que GIPPPY es un juego que transmite bien la sintaxis de Python, que hace pensar bastante a los usuarios para superar los niveles, y que a la vez es divertido. Algunos participantes no han estado tan satisfechos, así que es un juego que debe perfeccionarse, viendo las razones que han provocado esos resultados.

CAPÍTULO 7
Conclusiones

Para finalizar este trabajo, se presentan las conclusiones obtenidas analizando los resultados del trabajo.

El objetivo principal de este trabajo ha sido aportar una herramienta divertida y efectiva de iniciar el proceso de aprendizaje para programar en Python, ya que el proceso de intentar aprender un lenguaje de programación es un proceso duro para cualquiera. Según los resultados del formulario, GIPPPY es un juego divertido y entretenido, que tiene un nivel de dificultad medio-alto y que ha conseguido enganchar a los jugadores que lo han probado. Además, la interfaz del juego es sencilla y con un diseño poco recargado, lo que ayuda a su jugabilidad. Por lo tanto, el objetivo de aportar un juego serio con elementos de gamificación se ha cumplido.

Por otro lado, se ha visto que el número de usuarios del lenguaje Python está en constante crecimiento. Muchas personas quieren aprender Python, y no existen demasiados juegos que enseñen a usar el lenguaje. Entonces, aportar una herramienta para enseñar las bases de programación en Python ha sido otro objetivo que se ha cumplido.

Otra conclusión que se obtiene tras leer el capítulo de Estado del arte, donde se explican los juego serios que existen para los principales lenguajes de programación, es que todos estos juegos están diseñados en inglés. Si aprender a programar ya es un proceso complicado, hacerlo en inglés para una persona que no domine el idioma puede complicarlo aún más. Por esta razón. GIPPPY tiene soporte a varios idiomas: castellano e inglés. Incluso se ha dejado el código abierto y con la documentación correspondiente para facilitar añadir más idiomas en un futuro.

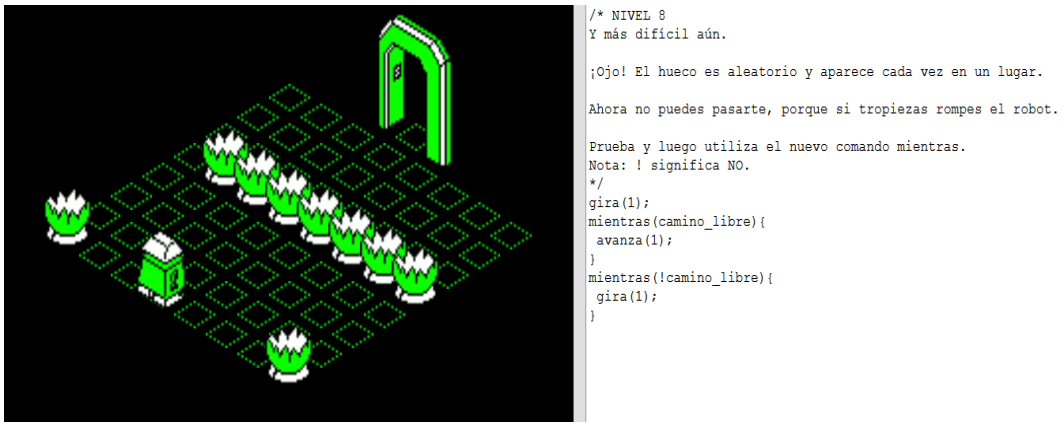
Respecto al trabajo realizado, y teniendo en cuenta que se dispone de un tiempo limitado para desarrollar el trabajo, existen algunas limitaciones que se deben tener en cuenta para futuras mejoras:

- **Combinación de estructuras**: Como Python es una lenguaje muy restrictivo en los bloques de códigos (la identación es la manera en la que el intérprete de Python reconoce las estructuras), no se ha podido implementar un compilador de Python completo. Para este trabajo, el compilador Python analiza las estructuras con una identación de 2 espacios estrictamente (mediante expresiones regulares). Por ejemplo, si queremos hacer un condicional con un bucle dentro, habría que identar la instrucción repite() con 2 espacios, y las instrucciones dentro del repite() irían con 4 espacios, lo que habría complicado mucho la realización del compilador. Por lo tanto, si se diera la opción a combinar estructuras, habría que modificar el sistema de espaciado que se ha implementado.

- **Estructura mientras()**: En el juego del robot para el aprendizaje de C, del que se ha hablado en el apartado 2.1, existe una estructura iterativa a parte del repite(). La estructura empieza con la palabra clave mientras(), y como parámetro solo puede recibir las

siguientes opciones: camino_libre o !camino_libre (ver figura 7.1). Esta estructura pretendía diseñarse en el juego, pero ha faltado tiempo para su implementación.

- **Evaluación con cuestionario**: La evaluación para este juego se ha llevado a cabo mediante un cuestionario, lo cuál no representa de manera 100 % real el contexto para el que se ha diseñado este juego. Lo ideal habría sido poder llevar este juego a una clase de un curso de programación y ver los resultados usando técnicas de grabado de pantallas o cuestionarios de cara.



```
/* NIVEL 8
Y más difícil aún.

¡Ojo! El hueco es aleatorio y aparece cada vez en un lugar.

Ahora no puedes pasarte, porque si tropiezas rompes el robot.

Prueba y luego utiliza el nuevo comando mientras.
Nota: ! significa NO.
*/
gira(1);
mientras(camino_libre){
 avanza(1);
}
mientras(!camino_libre){
 gira(1);
}
```

Figura 7.1: Estructura mientras() en el juego del robot para C

Respecto a los resultados del cuestionario, se puede concluir que GIPPPY ha sido útil como herramienta para introducir al pensamiento computacional a las personas sin experiencia que han probado el juego. Si vemos la figura 6.3, un 28,6 % de los jugadores no tenían ningún conocimiento en programación antes de jugar a GIPPPY. Ese porcentaje es idéntico al de la figura 6.6, es decir, al de jugadores que no han podido superar todos los niveles del juego. Si se miran ahora los resultados de la pregunta 6.7, prácticamente todos ellos han podido superar hasta el nivel 7-8, lo cuál está muy bien ya que han podido superar más de la mitad de los niveles. Entonces GIPPPY ha hecho a estos jugadores desarrollar un pensamiento computacional para encontrar solución a unos problemas, que es uno de los principales objetivos del juego.

Respecto a la relación que hay entre el trabajo desarrollado y los estudios cursados en la carrera, el diseño de una página web con HTML, CSS, Bootstrap y Javascript ha sido uno de los principales conceptos que se ha explicado a lo largo de la carrera de informática en la asignatura Desarrollo Web, principalmente, así que se han puesto a prueba los conceptos aprendidos en la asignatura. Luego, teniendo en cuenta la parte lógica, al ser una implementación difícil, se han usado muchas funciones específicas de Javascript, por lo que el conocimiento del lenguaje y de la programación en general que se ha adquirido en la carrera ha sido fundamental para el código. Las asignaturas fundamentales han sido en general las de programación y de nuevo Desarrollo Web. También mencionar el testing de código, que es un concepto que también se ha introducido en la carrera en la asignatura LTP (Lenguajes, tecnologías y Paradigmas de la programación), por lo que simplemente se ha profundizado un poco más para poder ser usados validando GIPPPY.

Por último, el código del juego se va a mantener, ya que va a seguir siendo utilizado tras la finalización de esta trabajo de fin de grado. Al haber subido el código de forma abierta y pública a GitHub, se espera que se sigan desarrollando más funcionalidades aparte de las ya explicadas en este trabajo. Como trabajos futuros sobre GIPPPY, se esperan los siguientes:

- **Más dificultad:** Pese a que GIPPPY ya tiene un nivel dificultad medio-alto, se espera que se pueda añadir más dificultad. Esto podría hacerse añadiendo más niveles con

diseños más complicados, creando nuevos obstáculos que dificulten la resolución de los niveles, etc. Se podría incluso añadir más instrucciones que entienda el robot para ampliar las funciones del juego. El código actual está bien documentado y preparado para mantenerse, para que otros programadores puedan seguir con el mantenimiento del juego.

- **Más lenguajes de programación:** Ampliar GIPPPY para soportar otros lenguajes de programación es algo factible, por ejemplo siguiendo la estructura que hay actualmente en el cambio de idioma mediante un click en unas banderas, solamente que ahora sería para modificar el lenguaje de programación. Java es un lenguaje que no tiene tantos juegos serios para aprender a programar, por lo que podría ser una buena opción para añadir a GIPPPY.

- **Mayor evaluación:** Actualmente, GIPPPY ha sido enviado como herramienta de aprendizaje a una universidad en Holanda, y también se ha difundido en la Universidad Politécnica de Valencia en varios cursos de programación. Estos alumnos no han podido responder el cuestionario por razones de tiempo, pero aún así van a evaluar GIPPPY y se espera que usen el juego como herramienta complementaria al aprendizaje.

Bibliografía

[1] Gráfica indicando crecimiento del lenguaje Python en los últimos años. ¿Cuántos lenguajes de programación existen? https://www.epitech-it.es/cuantos-lenguajes-existen/

[2] Action Research | School of Education | University of Bristol. http://www.bris.ac.uk/education/study/continuing-professional-development-cpd/actionresearch/

[3] Usos del lenguaje Python según los usuarios de StackOverflow en 2020. https://steelkiwi.com/blog/python-for-ai-and-machine-learning/

[4] Lenguajes de programación para Machine Learning. https://aprendeia.com/lenguajes-de-programacion-para-machine-learning/

[5] Programando videojuegos en Python. https://elbauldelprogramador.com/programando-videojuegos-en-python/

[6] Robot Turtles - The Board Game that Teaches Programming to Kids http://www.robotturtles.com/

[7] CoderBunnyz - Be Coder, Be Cool! http://www.coderbunnyz.com/

[8] Elevator Saga - the elevator programming game. https://play.elevatorsaga.com/

[9] Untrusted - a user javascript adventure game. https://alexnisnevich.github.io/untrusted/

[10] Robocode, juego para aprender a programar en Java. https://robocode.sourceforge.io/docs/ReadMe.html

[11] The SQL Murder Mystery. https://mystery.knightlab.com

[12] CSS Diner - Where we feast on CSS Selectors. https://flukeout.github.io/

[13] Juego del Robot en personales UPV para aprender las bases de la programación en C. http://personales.upv.es/daguelo/INF/robot/

[14] Identación en Python. GeeksforGeeks. https://www.geeksforgeeks.org/indentation-in-python/

[15] Bootstrap. The most popular HTML, CSS and JS library in the world. https://getbootstrap.com/

[16] Los 5 lenguajes de programación más usados durante 2021. https://www.imaginacolombia.com/articulos/5-lenguajes-de-programacion-mas-usados-durante-el-2021

[17] Delightful Javascript testing: facebook/jest. https://github.com/facebook/jest

[18] Game for Introductory Programming Practice in Python https://robot.testar.org/

[19] Repositorio con el código del juego GIPPPY https://github.com/GIPPPY/

[20] Cuestionario GIPPPY TFG Aitor Lapeña https://forms.gle/ACtcercGFsnzUm1XA

[21] GIPPPY FORM Aitor Lapeña Final Project https://forms.gle/G7KvSTSLCaRx5uUX8

[22] Beecher, K. *Computational Thinking: A beginner's guide to problem-solving and programming*. Swindon: BCS Learning & Development Limited, 2017.

[23] Beatriz Marín, Jonathan Frez, José A. Cruz-Lemus, Marcela Genero. *An Empirical Investigation on the Benefits of Gamification in Programming Courses*. ACM Trans. Comput. Educ. 19(1): 4:1-4:22 (2019).

[24] J. Vargas-Enríquez, L. García-Mundo, M. Genero, and M. Piattini. *Análisis de uso de la gamificación en la enseñanza de la informática.* Actas de las XXI Jornadas de la Enseñanza Universitaria de la Informática (JENUI'15). 105–112, 2015.

[25] I. Caponetto, J. Earp, and M. Ott. *Gamification and education: A literature review.* Proceedings of the 8th European Conference on Games Based Learning (ECGBL'14). 50–57, 2014.

[26] Y. Attali and M. Arieli-Attali. *Gamification in assessment: Do points affect test performance?* Comput. Educ. 83, 57–63, 2015.

[27] S. de Sousa, V. Durelli, H. Macedo Reis, and S. Isotani. *A systematic mapping on gamification applied to education.* Proceedings of the 29th Annual ACM Symposium on Applied Computing (SAC'14). 216–222, 2014.

[28] T. M. Connolly, E. A. Boyle, E. MacArthur, T. Hainey, and J. M. Boyle. *A systematic literature review of empirical evidence on computer games and serious games.* Comput. Educ. 59, 2 (2012), 661–686.

[29] A. Calderón and M. Ruiz. *A systematic literature review on serious games evaluation: An application to software project management.* Comput. Educ. 87 (2015), 396–422.

[30] Walter Cazzola, Diego Mathias Olivares. *Gradually Learning Programming Supported by a Growable Programming Language*. IEEE Transactions on Emerging Topics in Computing, 4(3), 404-415, 2015.

[31] Beatriz Marín. *Lessons Learned About Gamification in Software Engineering Education.* Latin American Women and Research Contributions to the IT Field (2021), DOI 10.4018/978-1-7998-7552-9.ch008.

[32] Finstad, K. *The usability metric for user experience.* Interacting with Computers, 22(5), 323-327, 2010.