Creación de un videojuego educativo sobre células para la enseñanza de la programación

Álvaro Ramírez Míguez, Raquel Hijón Neira

Universidad Rey Juan Carlos

Escuela Técnica Superior en Ingeniería Informática

Grado en Diseño y Desarrollo de Videojuegos

a.ramirezm.2016@alumnos.urjc.es, raquel.hijon@urjc.es

Resumen: En este artículo, se describe el desarrollo de un videojuego educativo sobre programación para ordenador, creado en Unity. La aplicación se llama CoDN y consiste en un juego de puzles en el que se debe programar el código genético de una célula para que realice las siguientes funciones vitales: nutrición, interacción y reproducción. La célula se programa añadiendo tareas a su código genético a través de una interfaz gráfica, sin necesidad de escribir ninguna línea de código o conocer algún lenguaje de programación. Estas tareas dan la posibilidad de crear bucles y saltos condicionales, para escoger el comportamiento de la célula dependiendo de factores como su energía o el tipo de partícula que se haya introducido. El juego dispone de distintos niveles en los que se explica el funcionamiento de las tareas y se ponen a prueba a los usuarios y las usuarias con ciertos objetivos. En cada nivel se añaden nuevos comandos y los objetivos se vuelven más complejos de forma que, se ofrecen más posibilidades para modificar el comportamiento de la célula a la vez que se enseñan aspectos fundamentales de la programación. La aplicación se encuentra alojada en el siguiente enlace y se puede jugar tanto en inglés como en español: https://pisco.itch.io/codn

Palabras clave: CoDN, itch.io, videojuego educativo, programación visual, celular, Unity.

ABSTRACT: This article describes the development of an educational videogame about programming, made with Unity. The application is called CoDN and it consists of a puzzle game in which a cell's genetic code must be programmed to perform the following vital functions: nutrition, interaction and reproduction. The cell is programmed by adding commands to its genetic code through a graphical interface, without the need to write any line of code or to know any programming language. These tasks offer the possibility of creating loops and conditional jumps, to choose the behaviour of the cell depending on factors such as its energy value or the type of particle that has been introduced. The game has different levels where the different tasks are explained and the users are tested with certain objectives, specific to each level. At each level new commands are added, and the objectives become more complex. This way, more possibilities are offered to modify the behaviour of the cell while fundamentals aspects of programming are taught. The application hosted at the following link can be played in both English and Spanish: https://pisco.itch.io/codn

Keywords: CoDN, itch.io, educational videogame, visual programming, cellular, Unity.

1. Introducción

El objetivo de este proyecto es realizar un videojuego educativo para ordenador sobre fundamentos de la programación a través de organismos unicelulares.

La programación es un ámbito de creciente interés hoy en día y, en consecuencia, las ganas de aprender a programar. Aun así, se trata de una materia intimidante para muchas personas debido a la idea de

 © ADIE, Asociación para el Desarrollo de la Informática Educativa

que se deben aprender lenguajes complejos y muy diferenciados.

Si bien es cierto que cada lenguaje tiene sus particularidades, comprender los conceptos básicos facilita la transición de uno a otro. Por ello, la intención de este proyecto no es enseñar un lenguaje de programación determinado, sino servir como paso introductorio a la programación, enseñando fundamentos computacionales como la entrada y salida de datos o los saltos condicionales.

Por otra parte, las dinámicas de juego ofrecen un amplio rango de beneficios en la enseñanza, permitiendo seguir y recompensar los logros del alumnado fomentando la cooperación y competitividad. Además, su carácter lúdico facilita la interiorización de conocimientos de forma divertida, generando una experiencia positiva (Deterding, Dixon, Khaled, y Nacke, 2011).

Estas características hacen que combinar la enseñanza de la programación con los videojuegos sea una propuesta interesante.

Por último, las similitudes entre ciertos procesos celulares (Urry, Cain, Wasserman, Minorsky y Reece, 2017) e instrucciones computacionales hacen que un juego sobre programación del código genético sea una temática apropiada. Además, permite la familiarización con términos del ámbito celular a la vez que se aprende a programar.

A continuación, se muestran algunos ejemplos de relaciones entre procesos celulares e instrucciones:

- Endocitosis-Input: las células utilizan la endocitosis para obtener partículas del medio extracelular, al igual que un programa obtiene información del usuario a través de inputs.

- Exocitosis-output: las células pueden secretar sustancias al exterior a través de la exocitosis. De forma similar, un programa utiliza outputs para enviar información al usuario.

- Respiración-procesar información: se puede relacionar el proceso de transformar la materia obtenida del medio extracelular en energía, con el procesamiento de información recibida mediante inputs por un programa.

2. Documentación Previa

Previo al desarrollo del videojuego se han analizado distintas técnicas para enseñar a programar y aplicaciones similares a este proyecto.

2.1 Estrategias de Enseñanza de la Programación

La complejidad de los programas actuales incrementa la importancia de la enseñanza de la programación (Moroni y Señas, 2005). Actualmente las metodologías más relevantes son el aprendizaje mediante pseudocódigo y el uso de lenguajes de programación visual.

2.1.1 Pseudocódigo

Una estrategia habitual consiste en empezar diseñando algoritmos en papel utilizando pseudocódigo (Lisa, 1994) para después aplicarlos a un lenguaje de código popular. El pseudocódigo, o lenguaje de descripción algorítmico, es una descripción a alto nivel compacta e informal de un programa diseñada para la lectura humana (Kenneth y Julie, 2005).

Este método permite familiarizarse con la descripción de programas sencillos sin la barrera de aprendizaje inicial que conlleva aprender un lenguaje específico. Por otra parte, suelen ocupar menos espacio que el código de un programa, siendo de utilidad a la hora de documentar algoritmos o planificar el desarrollo de programas informáticos.

Si bien se ha demostrado la eficacia de esta estrategia, puede resultar poco atractiva,

especialmente para un público joven ávido de utilizar ordenadores. Además, al tratarse de una esquematización subjetiva de un programa, es complicado comprobar su validez o entender su semántica. Otra desventaja del pseudocódigo es la dificultad de representar programas extensos al estar diseñado para la descripción de algoritmos y secciones de código.

2.1.2 Lenguajes de Programación Visual

Con el objetivo de combinar las ventajas del aprendizaje mediante pseudocódigo con un sistema para visualizar y comprobar los resultados del código, surgieron los lenguajes de programación visual. Este tipo de lenguajes utilizan elementos gráficos para representar el código, en lugar de utilizar exclusivamente una especificación textual, permitiendo el desarrollo del pensamiento computacional sin conocimientos profundos sobre código (Remi, 2015).

En los últimos años, este tipo de programación ha logrado una gran difusión en el ámbito educativo con entornos como Scratch (Resnik, 2003) o Blockly (Google, 2019). Por lo general, utilizan bloques para representar acciones ejecutables que se pueden reordenar y conectar sin necesidad de escribir código (Sáez y Cózar, 2017).

Este acercamiento, cada vez más popular, se está implementando en motores de videojuegos como Unreal Engine (Epic Games, 1998), permitiendo a perfiles menos familiarizados con la programación una mayor participación en el desarrollo.

2.2 Videojuegos y Programación

La programación también está presente en los videojuegos actuales como mecánica. Poder programar permite a los jugadores y a las jugadoras superar obstáculos de forma creativa, personalizar su experiencia e incluso crear sus propios juegos.

La mayor referencia para este proyecto ha sido Human Resource Machine (Tomorrow Corporation, 2015), un juego de puzles en el que se deben programar las tareas de un oficinista a través de una interfaz visual. Para programar se dispone de una lista de comandos que se pueden añadir a una hoja numerada, que indica el orden en el que se ejecuta cada instrucción.

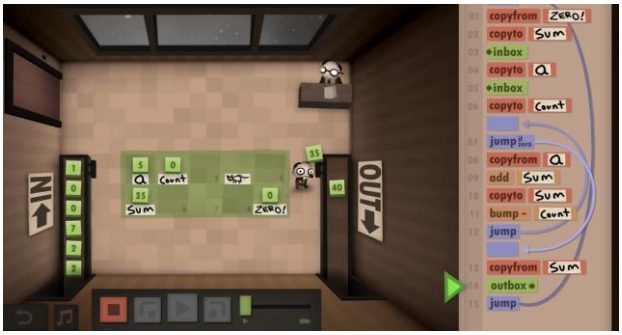

Figura 1: Human Resource Machine

Otra fuente de inspiración es CodeMonkey (CodeMonkey, 2014), un entorno de programación interactivo orientado a la enseñanza. Cuenta con varios niveles en los que se debe programar el comportamiento de un mono para que supere determinados obstáculos utilizando lenguajes de código reales. Además, se trata de un entorno preparado para aulas equipado con soluciones para estudiantes, calificación automática y gestión del plan de estudios.

También se pueden encontrar juegos en los que la programación es una mecánica secundaria y opcional como en Minecraft (Mojang Studios, 2011), que permite crear mecanismos a través de circuitos lógicos.

2.3 Videojuegos y Células

La mayoría de los videojuegos con una temática basada en microorganismos se tratan de juegos de estrategia en los que se debe gestionar los recursos disponibles para mantener con vida a una colonia.

En este tipo de juegos, es habitual tener que enfrentarse a otros microorganismos que necesitan los mismos recursos para sobrevivir. Otra característica común es la opción de modificar

genéticamente a las células, para mejorar sus capacidades o cambiar el estilo de juego. A continuación, se muestran dos ejemplos:

C.U.R.E. (Cryogenic Entertainment, 2017), es un juego de estrategia en tiempo real en el que se utiliza ingeniería genética para ayudar a microorganismos a defenderse contra enfermedades. Hay distintos escenarios basados en la anatomía humana con sus propios sistemas inmunitarios y depredadores.

Microcosmum (Satur Entertainment, 2018) también es un juego de estrategia, pero con un tono menos realista, ya que opta por una representación artística y relajante. El objetivo es infectar a otros organismos con esporas para hacerte con su control.

En ambos casos, las células no se programan para actuar por su cuenta, sino que dependen de la habilidad del jugador o de la jugadora para controlarlas. Tampoco se tratan de juegos adaptados para la enseñanza ya que están más orientados al entretenimiento.

2.4 Evaluación de la Idea del Proyecto

Entre los videojuegos analizados no se ha encontrado ninguno en el que se utilice programación visual para modificar el comportamiento de células. Lo más similar a este concepto sería "Game of Life", el juego matemático diseñado por John Horton Conway. Se trata de un entorno en el que se pueden colocar células que, a través de simples reglas matemáticas, pueden vivir, morir o multiplicarse (Gardner, 1970). Si bien es cierto que el comportamiento determinista de las células permite programar circuitos lógicos, se trata de un método complejo que requiere entender el funcionamiento de un ordenador a bajo nivel.

También se ha observado que la mayoría de los juegos de programación son juegos de puzles mientras que, los juegos que utilizan células o microorganismos como temática, suelen ser juegos de estrategia.

3. Objetivos

Una vez establecida la idea, fue necesario sentar una serie de objetivos que debe cumplir la aplicación. Al ser la enseñanza de la programación la meta principal del proyecto, disponer de una interfaz sencilla, comprensible y funcional era esencial. También era importante presentar los distintos comandos de forma progresiva, para facilitar su aprendizaje y asegurar su entendimiento antes de pasar al siguiente nivel.

Los conceptos de programación que se han decidido tratar a lo largo del juego son los siguientes:

- **Entrada y salida de información**: recibir datos de un medio externo, modificar su información y enviar los datos una vez modificados.
- **Funciones**: utilizar instrucciones que requieren parámetros.
- **Ciclos**: crear mecanismos de iteración.
- **Control de flujo**: tomar decisiones dependiendo de la información almacenada en las variables.

Otro objetivo del proyecto era familiarizar a los usuarios y a las usuarias con distintos procesos celulares por lo que se utilizarían para nombrar las distintas tareas que puede realizar la célula. Además, estas tareas deben ser sencillas y ofrecer la posibilidad de recrear las funciones vitales de la célula (nutrición, interacción y reproducción).

4. Diseño

Teniendo los objetivos en cuenta, se diseñaron las mecánicas del juego, las reglas que indican las acciones que puede realizar el jugador y la respuesta del juego a dichas acciones (Rubin, 2009). Dicho de otra forma, se tratan de los sistemas de interacciones entre el jugador y el juego que especifican su funcionamiento (Boller, 2013).

La mecánica principal del juego consiste en programar una célula con el fin de cumplir ciertas metas. Para ello se deben escoger y ordenar las instrucciones que constituirán su código genético. Este código se puede reordenar, borrar o compilar en una simulación para comprobar su validez. Durante

esta simulación, aparecerán partículas de forma aleatoria.

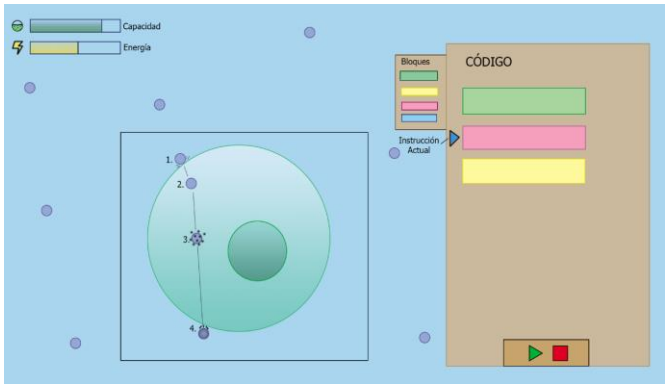

Figura 2: Boceto inicial del juego

En la figura 2 se puede ver el boceto inicial de un escenario de juego. En el centro se encuentra una célula procesando una partícula. A su derecha el código, compuesto por instrucciones diferenciadas. Arriba a la izquierda se sitúan la capacidad de almacenamiento y la energía de la célula.

La célula dispone de energía limitada que debe gastar para realizar ciertas instrucciones y podrá recuperarla procesando determinadas partículas. Estas instrucciones se encuentran en una ventana y su número irá aumentando a medida que se avance en los niveles del juego.

Las instrucciones disponibles son:

Endocitosis: introduce la primera partícula que entre en contacto con la célula si hay espacio suficiente.

Exocitosis: expulsa la primera partícula introducida en la célula.

Respiración: procesa la primera partícula que se haya introducido en la célula para obtener energía. Los resultados de la respiración dependerán del tipo de partícula procesada (Tabla 1).

Tabla 1: Procesamiento de partículas

Partícula Procesada	Energía Obtenida	Partícula Resultante
Positiva	+ 3	Neutra
Neutra	+ 0	Positiva
Negativa	- 9	Negativa

Saltar (línea): se salta a la línea de código que se haya introducido como parámetro en la instrucción.

Si partícula (tipo): comprueba de qué tipo es la primera partícula introducida en la célula. En caso de que no coincida con el tipo de partícula seleccionado, se salta la instrucción que realizaría a continuación.

Si energía > (valor): comprueba la cantidad de energía de la célula. En caso de que no supere el valor seleccionado, se salta la instrucción que realizaría a continuación.

Si reserva > (valor): comprueba la cantidad de partículas en la célula. En caso de que no supere el valor seleccionado, se salta la instrucción que realizaría a continuación.

Mitosis: Divide la célula en 2 células idénticas. Cada célula recibe la mitad de energía que poseía la célula original y comparten el mismo código genético

Destruir: La célula gasta 2 unidades de energía para destruir la primera partícula que se haya introducido.

Color: Cambia el color de la célula al color seleccionado.

Las instrucciones que se pueden utilizar en un nivel aparecen en la lista de tareas y se añaden al código genético de la célula con el botón izquierdo del ratón. Una vez el código está listo, se pulsa el botón de "Play" para iniciar la simulación.

Se pueden reordenar las tareas en el propio código arrastrándolas a la posición deseada. Para eliminar instrucciones, se pueden arrastrar fuera del código o pulsar el botón con el icono de la papelera para borrar todas las tareas.

Una vez comience la simulación, la célula realizará las tareas en el orden en el que se hayan colocado, de arriba abajo. Alterar el orden del código durante la simulación no cambiará el comportamiento de la célula. Se puede modificar la velocidad del juego utilizando la barra de deslizamiento situada a la derecha del botón "Play".

La célula debe cumplir los objetivos del nivel sin que se detenga la simulación una vez haya comenzado. La reproducción se puede detener por los siguientes motivos: La célula se ha quedado sin energía; Ha habido en error en el código; No se han cumplido los objetivos del nivel; El usuario o la usuaria la ha detenido, ya sea borrando el código o comenzando una nueva simulación.

Al terminar el nivel se comprobará si se han cumplido los objetivos, además del número de pasos realizados y la cantidad total de instrucciones en el código.

5. Implementación

La aplicación ha sido desarrollada en Unity, un motor de videojuegos que permite desarrollar aplicaciones para varias plataformas, utilizando C# como lenguaje de programación.

El juego se divide en dos escenas: el menú principal y la pantalla de juego, y cada escena cuenta con una clase administradora que se encarga de la navegación entre las distintas pantallas.

El menú principal se controla a través de la clase MenuManager, que se encarga del acceso a la selección de niveles y de iniciar el sonido del juego.

En el caso de la pantalla de juego, la clase principal es GameHandler, que controla los distintos estados de juego y ajusta los parámetros necesarios para iniciar la partida dependiendo del nivel seleccionado.

Para el funcionamiento y visualización del código genético hay tres clases fundamentales: CodeObject, CodeDisplay y TaskSystem.

CodeObject es la clase que contiene el código de la célula y la información necesaria para visualizarlo. Para cada instrucción se almacena su línea de código y el objeto que se va a instanciar a la hora de ser representada. En este objeto también se guarda el tipo de instrucción y sus parámetros de entrada.

CodeDisplay es la clase encargada de recorrer las instrucciones de CodeObject y representarlas visualmente. Además, añade eventos a cada representación de las instrucciones para poder modificar el código de forma interactiva. Cada vez que el código es modificado, es necesario actualizar su representación visual. De la misma forma, cada vez que se modifica la interfaz visual hay que actualizar el código.

TaskSystem es la clase que almacena las tareas que debe realizar la célula y se genera al iniciar la simulación a partir del código de CodeObject. Como no se puede modificar el código durante la simulación, no es necesario actualizarla.

6. Resultados

En la Figura 3 se puede observar el resultado final de la pantalla de juego.

En el centro se sitúa la célula, con una barra sobre ella que indica el porcentaje de energía que posee. A la derecha, se encuentran la Lista de Tareas, con las instrucciones que se pueden utilizar, y el Código Genético, donde se ubican las tareas añadidas al código. En la parte inferior del código se sitúa el botón para iniciar y detener la simulación, la papelera y la barra de velocidad.

Una vez que comienza la simulación, se resalta la tarea que está realizando la célula en el código. De esta forma se puede observar cual fue la última tarea en realizarse en caso de que se detenga su ejecución. Con el botón situado en la parte superior de la pantalla se accede al menú de pausa. Desde este menú se puede reanudar el juego o regresar al menú principal.

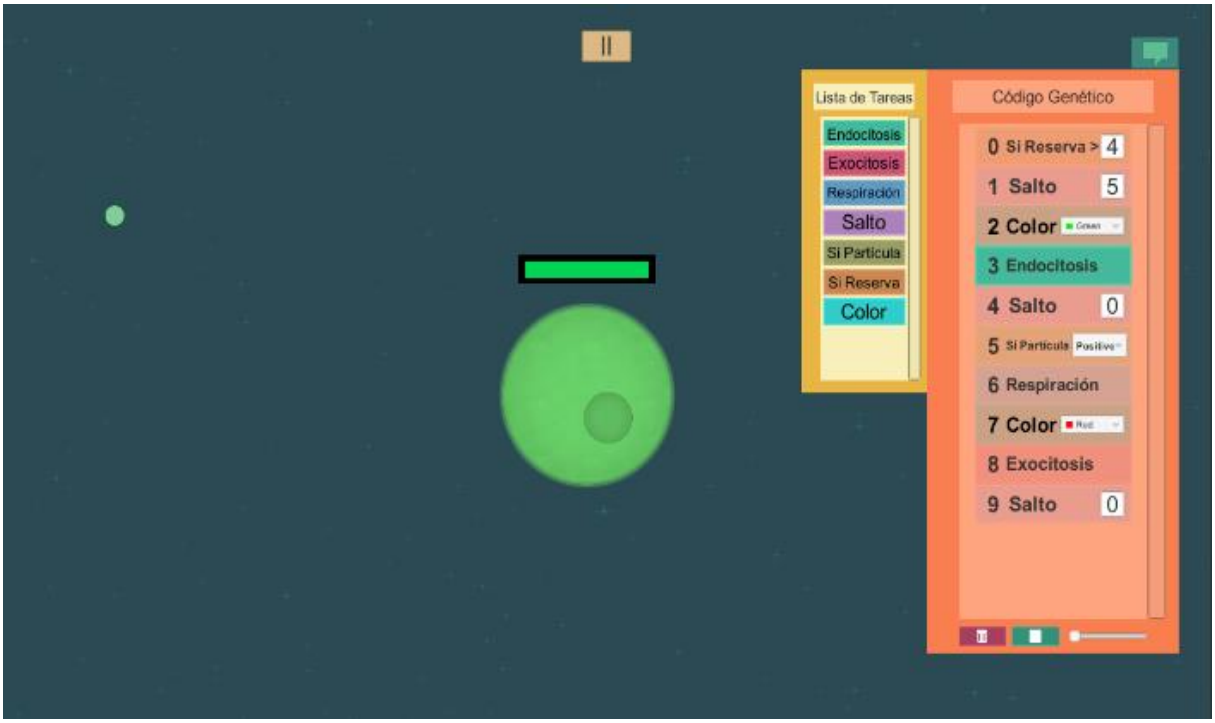

Figura 3: Captura del nivel 5 del juego

El siguiente paso fue alojar la aplicación en la página de itch.io, de forma que se pudiese probar desde distintos dispositivos sin instalación previa, simplemente accediendo a un enlace desde cualquier navegador.

A continuación, se probó el juego entre usuarios y usuarias con distintos niveles de experiencia con la programación. Debido a las restricciones de movilidad causadas por el Covid-19 a lo largo del 2020, el grupo de usuarios que probaron la aplicación se vio reducido a entornos familiares.

Por lo general, las personas testadas necesitaban unos minutos para interiorizar el control de la interfaz gráfica. Un aspecto a tener en cuenta para facilitar este proceso es ofrecer varias formas de interacción con la interfaz, ya que no todos los usuarios han utilizado aplicaciones similares.

Entre los usuarios que no tenían experiencia previa, se han detectado dificultades para comprender que el comportamiento de la célula debe estar totalmente definido en el código antes de comenzar la simulación. En varios casos, se ha intentado añadir tareas durante la propia simulación, lo cual no modificaba la conducta de la célula. Este problema era habitual en el primer nivel y solía resolverse sin necesidad de ayuda externa. En los niveles intermedios también había dificultades para comprender las tareas con saltos condicionales, concretamente, en qué parte del código se deben colocar.

Para los usuarios ya iniciados en la programación, se trató de una experiencia sencilla con cierto desafío en los últimos niveles. Aun así, encontraron el juego entretenido por la posibilidad de superar los niveles a su manera. Al no haber una solución única, se pueden adoptar varias estrategias siempre y cuando se cumplan los objetivos.

7. Conclusiones

7.1 Objetivos Alcanzados

El objetivo principal del proyecto era desarrollar un videojuego educativo sobre programación en el que se diseña el código genético de una célula, a la vez que se explican los procesos celulares más básicos.

La aplicación conseguida permite programar células de forma visual sin necesidad de conocimientos previos sobre lenguajes de programación. Desde el código genético se pueden añadir tareas,

reordenarlas, modificarlas y borrarlas con el uso del ratón.

Cada nivel del juego presenta nuevos conceptos de programación y desafíos en los que se ponen a prueba las capacidades de los jugadores y las jugadoras. Poder cambiar el idioma de la aplicación de inglés a español también era esencial para hacer más accesible el juego, debido a los diálogos explicativos previos a cada nivel.

El funcionamiento de los menús del juego permite navegar a través de las pantallas en pocos pasos, sin necesidad de realizar secuencias de acciones complejas. Además, se ofrece la posibilidad de jugar desde el navegador sin necesidad de descargar archivos.

Todos los elementos visuales se han creado desde cero, al igual que los efectos de sonido. Por falta de tiempo, no se ha logrado componer música original para la aplicación, por lo que se han buscado alternativas gratuitas para el tema principal del juego.

Al completar cada nivel se indica el tamaño del código y el número de tareas realizadas.

7.2 Dificultades en el Desarrollo

Uno de los mayores desafíos del proyecto ha sido el diseño e implementación de la interfaz gráfica del código genético. La capacidad de editar las tareas de la célula de forma sencilla e intuitiva era una prioridad ya que de ello dependería gran parte del atractivo del juego.

Representar visualmente el código no era el mayor problema, sino mantener la correlación entre la interfaz y el propio código después de editarlo. Los errores de visualización eran bastante comunes al borrar instrucciones o alterar su orden.

Otra de las dificultades en el desarrollo de la aplicación ha sido diseñar las tareas de forma que permitiesen ofrecer suficiente control en el comportamiento de la célula sin complicar de forma excesiva su programación.

Para solventarlo se han diseñado instrucciones con suficiente sinergia entre ellas como para poder generar profundidad en sus combinaciones y no tanto en su uso por separado. Por ejemplo, con las tareas condicionales tan sólo se puede evitar realizar la tarea inmediatamente después, pero con el uso de la instrucción de salto, permiten controlar el flujo del código.

En cuanto al sistema de puntuaciones, es necesario probar la aplicación con más usuarios para poder valorar la calidad de distintos códigos. Además, al no tratarse de un sistema determinista, es posible obtener distintos resultados utilizando códigos idénticos, lo que hace más complicado ofrecer un sistema de puntuaciones robusto. Por estas razones, no está disponible el sistema de puntuaciones en la versión actual del juego.

7.3 Trabajos Futuros

A continuación, se presentan las distintas áreas u oportunidades de mejora detectadas durante el desarrollo del proyecto y que podrán ser abarcadas en futuras versiones del Software.

7.3.1 Sistema de Puntuaciones

Es importante ofrecer un sistema con el que evaluar la calidad del código. En la versión actual se aporta información sobre el tamaño del código y la cantidad de instrucciones realizadas hasta lograr los objetivos, pero no se indica de ninguna forma si los resultados se pueden mejorar.

Un sistema de puntuaciones no sólo serviría para evaluar al usuario, también para invitarlo a mejorar.

7.3.2 Mensajes de Error

Aunque al detenerse la simulación se resalta la última tarea realizada, mostrar mensajes de error ayudaría a identificar mejor los fallos en el código.

7.3.3 Nuevos Niveles

En las pruebas realizadas, se ha detectado que la dificultad aumenta considerablemente al añadirse los saltos condicionales. Añadir más niveles ayudaría a suavizar la curva de aprendizaje.

7.3.4 Estudio Sobre el Impacto del Juego

A la hora de realizar un juego educativo es esencial comprobar la eficacia de este. Si bien es necesario aplicar principios de programación para superar los niveles, habría que comprobar el efecto de haber completado el juego a la hora de aprender a programar

7.3.5 Distribución de la Aplicación en Colegios e Institutos

El último paso sería la distribución de la aplicación en entornos educativos reales para servir de apoyo para estudiantes y profesores en la enseñanza de la programación.

8. Referencias

Google, MIT. (2019). Blockly [Consulta: 24 de julio de 2020]. Disponible en: https://developers.google.com/blockly

Boller, Sharon. (2013). Learning Game Design: Game Mechanics [Consulta 24 de octubre de 2020]. Diponible en: http://www.theknowledgeguru.com/learning-game-design-mechanics/

CodeMonkey. (2014). CodeMonkey. [Consulta 24 de octubre de 2020]. Disponible en: https://www.codemonkey.com/

Cryogenic Entertainment. (2017). C.U.R.E. [Consulta: 24 de julio de 2020]. Disponible en: http://www.curethegame.com/game/

Deterding, Sebastian; Dixon, Dan; Khaled, Rilla y Nacke, Lennart. (2011). From Game Design Elements to Gamefulness: Defining Gamification. [Consulta: 24 de julio de 2020]. Disponible en: https://dl.acm.org/doi/abs/10.1145/2181037.2181040

Gardner, Martin. (1970) Mathematical Games – The Fantastic Combinations of John Conway's New Solitaire Game 'Life'". [Consulta: 21 de octubre de 2020]. Disponible en: https://web.stanford.edu/class/sts145/Library/life.pdf

Kenneth E. Kendall, y Julie E. Kendall. (2005) Análisis y diseño de sistemas (6ª edición). Pearson Education.

Mojang Studios. (2011). Minecraft. [Consulta: 21 de octubre de 2020]. https://www.minecraft.net/es-es

Moroni, Norma, y Señas, Perla. (2005) Estrategias para la enseñanza de la programación [Consulta: 24 de julio de 2020]. Disponible en: http://sedici.unlp.edu.ar/handle/10915/18901

Lisa Levy Kortright. 1994. From Specific Problem Instances to Algorithms in the Introductory Course. SIGCSE BULLETIN ACM.

Rémi D. (2015) The Maturity of Visual Programming [Consulta: 24 de julio de 2020]. Disponible en: https://www.craft.ai/blog/the-maturity-of-visual-programming

Rubin, Steve. (2009) Introduction to Game Development 2 ed. 2010.

Sáez López, José Manuel; Cózar Gutiérrez, Ramón. (2017) Programación visual por bloques en Educación Primaria: Aprendiendo y creando contenidos en Ciencias Sociales [Consulta: 24 de julio de 2020]. Disponible en: https://revistas.ucm.es/index.php/rced/article/view/49381

Satur Entertainment. (2018). Microcosmum: survival of cells [Consulta: 21 de octubre de 2020]. Disponible en: https://store.steampowered.com/app/386260/Microcosmum_survival_of_cells/

Resnik, Mitch. (2003). Scratch [Consulta: 24 de julio de 2020]. Disponible en: https://scratch.mit.edu/

Tomorrow Corporation. (2015). Human Resource Machine [Consulta: 24 de julio de 2020]. Disponible en:

https://tomorrowcorporation.com/humanresourcemachine

Unity Technologies. (2005). Unity [Consulta: 24 de julio de 2020]. Disponible en: https://unity.com/es

Epic Games. (1998). Unreal Engine [Consulta: 24 de julio de 2020]. Disponible en: https://docs.unrealengine.com/en-US/Engine/Blueprints/index.html

Urry, L. A., Cain, M. L., Wasserman, S. A., Minorsky, P. V., y Reece, J. B. (2017). Campbell Biology (11th ed.). Pearson.