Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN

Nombre del Centro

Trabajo Fin de Grado

PROTOTIPO PARA SIMULAR EL COMPORTAMIENTO DE LÍQUIDOS O GASES EN UN ENTORNO DINÁMICO DE MANERA PARAMETRIZABLE MEDIANTE AUTÓMATA CELULAR

Alumno: Javier Auñón del Barco

Tutor: Juan Roberto Jiménez Pérez

Guillermo Garrido Luque

Dpto: Departamento de Informática

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don Juan Roberto Jiménez Pérez y Guillermo Garrido Luque, tutores del Proyecto Fin de Carrera titulado: Prototipo para simular el comportamiento de líquidos o gases en un entorno dinámico de manera parametrizablemediante autómata celular, que presenta Javier Auñón del Barco, autorizan su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, Julio de 2021

El alumno: Los tutores:

Javier Auñón del Barco Juan Roberto Jiménez Pérez

Guillermo Garrido Luque

Índice

Índice de ilustraciones

Resumen

En el ámbito de la informática siempre se ha buscado mejorar los recursos tanto hardware como software, buscando una notable mejora de la potencia y una mayor capacidad para analizar grandes cantidades de datos. Estas mejoras a lo largo de los últimos años han ocasionado que se realicen simuladores más complejos con una mayor capacidad para mostrar resultados favorables en poco tiempo de cómputo. Gracias a la mejora de estos simuladores se han podido realizar estudios más profundos sobre distintos ámbitos de la ciencia, confirmando estudios que eran meramente teóricos o, incluso, haciendo nuevos descubrimientos. Específicamente, en el ámbito de la geología, hay muy pocos estudios sobre los procesos que ocurren en el interior de una cueva. El desarrollo de un simulador de fluidos en este ámbito puede dar lugar a distintas investigaciones sobre la evolución que puede sufrir una cueva a lo largo de los años.

En este trabajo se ha realizado un prototipo para simular el intercambio de energía que se produce en los distintos fluidos de un entorno dinámico. Este entorno está conformado por una cueva, dónde hay distintos materiales y se definen tres entidades principales: la atmósfera o el aire exterior, la roca en la que se encuentra las cavidades de la cueva, y el aire que hay en su interior. Esta clasificación nos va a permitir mejoras visuales como, por ejemplo, ver únicamente el flujo aire en puntos específicos de nuestro entorno.

Para la resolución del problema se ha utilizado un autómata celular que conforma el entorno dinámico y los distintos fluidos que se encuentran en él. Este autómata está formado de un conjunto de células que están almacenadas en una estructura de datos. Gracias a esta estructura se realizan algunas operaciones, como el intercambio de energía que se produce entre ellas a través de un entorno de vecindad, entre muchas otras. También se generarán o se destruirán células en zonas de interés, variando el tamaño del autómata y constituyendo así un sistema procedural. Cuanto más realista se quiera representar el problema, más difícil y más complejo va a ser su resolución.

El desarrollo de una aplicación de este tipo va a permitir en un futuro realizar investigaciones sobre este ámbito tan poco estudiado como es la geología. Pudiendo

predecir factores de riesgo y analizar datos sin necesidad de desplegar un equipo de monitorización.

Abstract

In the field of computing, there has always been a quest to improve both hardware and software resources, looking for a notable improvement in power and a greater capacity to analyse large amounts of data. These improvements over the last few years have led to the development of more complex simulators with a greater capacity to show favourable results in a short computational time. Thanks to the improvement of these simulators, it has been possible to carry out more in-depth studies in different fields of science, confirming studies that were merely theoretical or even making new discoveries. Specifically, in the field of geology, there are few studies on the processes occurring inside a cave. The development of a fluid simulator in this field can lead to different investigations on the evolution that a cave can undergo over the years.

In this work, a prototype has been created to simulate the energy exchange that takes place in the different fluids of a dynamic environment. This environment is made up of a cave, where there are different materials and three main entities are defined: the atmosphere or outside air, the rock in which the cavities of the cave are located, and the air inside the cave. This classification will allow us to make visual improvements such as, for example, seeing only the air flow at specific points in our environment.

In order to solve the problem, a cellular automaton has been used to form the dynamic environment and the different fluids found in it. This automaton is made up of a set of cells that are stored in a data structure. Thanks to this structure, some operations are carried out, such as the energy exchange that takes place between them through a neighbourhood environment, among many others. Cells will also be generated or destroyed in areas of interest, varying the size of the automaton and thus constituting a procedural system. The more realistic the problem is to be represented, the more difficult and complex it will be to solve.

The development of an application of this type will make it possible in the future to carry out research in the little-studied field of geology. It will be possible to predict risk factors and analyse data without the need to deploy monitoring equipment.

1. Introducción

Nuestro principal objetivo es simular el comportamiento de uno o varios fluidos, pero previamente hay que crear el entorno y realizar la identificación de las distintas entidades que lo componen. Después, definir y parametrizar las propiedades de los distintos materiales que se simulan. Posteriormente, realizar el intercambio de energía que se produce entre los distintos elementos y representarlo visualmente en la escena. Finalmente, al ser un sistema procedural, se deberán de definir las condiciones en las que el sistema crece o decrece.

Los principales obstáculos no son solo teóricos, sino también computacionales. Debido a la gran cantidad de cálculos que se tienen que realizar en cortos periodos de tiempo, la ejecución del programa se vuelve demasiado lenta. Y como el estado de un elemento es muy importante para realizar los cálculos adecuados, todos ellos tienen que actualizarse a la vez para evitar analizar datos erróneos. Si un elemento no se actualiza o modifica correctamente, se producirán fallos de simulación cada vez más grandes conforme avance la ejecución, provocando un efecto dominó de errores.

Como todo estudio, se requiere de pruebas y resultados bien definidos. En este caso, es necesario ver el flujo que se producen en los distintos fluidos del entorno. Por ende, se tendrá que obtener un vídeo por cada ejecución con datos extraídos de un entorno real y aplicándolos al sistema, viendo como pasan los días y los meses, apreciando los distintos procesos que ocurren en las distintas zonas que nos interese estudiar.

1.1. Descripción del problema

En primer lugar, no sólo hay que aplicar una fórmula para calcular el intercambio de energía que se producen entre elementos. También hay que tener en cuenta el cambio de densidad que se produce en un gas cuándo este tiene una u otra temperatura, la climatología de la zona en la que se encuentra el entorno, el vacío que se genera en el interior del entorno cuando hay ciertas condiciones (trampa de aire frío) y los distintos materiales con distintas propiedades que se pueden encontrar dentro de la cueva. Todos estos principios físicos han de tenerse en cuenta para que se asemeje lo máximo posible a la realidad.

Y por otro lado, al ser una aplicación que se puede utilizar en el ámbito de la geología, se tiene que parametrizar el mayor número de componentes posibles sin que la interfaz resulte demasiado abrumadora y poco intuitiva: la topología del entorno o la cueva, las propiedades de cada material y la temperatura de cada uno de ellos, aplicar un filtro para la visualización única de ciertas partes que nos puedan interesar estudiar y eliminando otras que no nos interese en un determinado instante, etcétera.

1.2. Objetivos

El objetivo principal de este proyecto es conseguir una simulación de fluidos realista en el que se muestre las condiciones físicas del entorno en distintos ambientes y condiciones:

- Definir y parametrizar propiedades para los distintos materiales del autómata.
- Parametrizar las condiciones del entorno.
- Calcular el flujo de energía que hay entre las distintas células del autómata.
- Visualizar el intercambio de energía entre células.
- Crear células a partir de una geometría para la estructura o geometría del entorno.
- Generar células de manera procedural en función de las condiciones del entorno.
- Evaluar y estudiar la EEDD más óptima para el problema.
- Obtener conclusiones sobre el comportamiento del entorno en las distintas condiciones que se simulan.

1.3. Metodología

Se presupone fundamental tener una metodología de desarrollo bien definida para que un proyecto cumpla con sus objetivos de calidad en el tiempo correspondiente. La metodología de desarrollo software que se ha utilizado en este

trabajo ha sido el modelo en espiral, que consiste en una combinación entre el modelo en cascada y el modelo basado en prototipos. El principal objetivo de este modelo en espiral es la resolución rápida de los fallos y errores que pueden surgir en el desarrollo.

Al implementar un simulador, hay que tener en cuenta muchos detalles, y todos y cada uno de ellos deben funcionar correctamente antes de pasar a la siguiente 'fase' del proyecto. Si una de las partes no funciona adecuadamente, hará que otras tampoco, provocando una sucesión de fallos en cascada y será difícil encontrar dónde se encuentra el error.

Como se puede apreciar en la *Ilustración 1* esta metodología consta de cuatro fases principales: Planificación, Análisis de riesgo, Implementación y Evaluación.

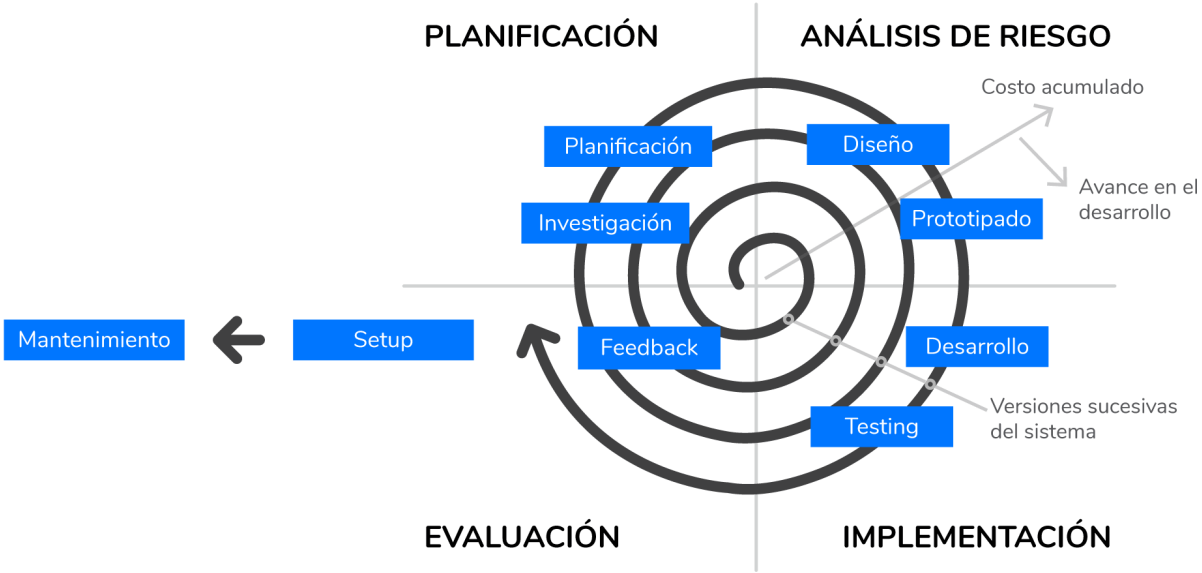

Ilustración 1. Esquema de la metodología en espiral. (https://aspgems.com/metodologia-de-desarrollo-de-software-iii-modelo-en-espiral/)

Planificación e investigación: es la primera fase y la que va a definir los conceptos que se van a implementar en esta iteración. Se investigará sobre los distintos conceptos físicos que hay que ir aplicando al simulador y sobre nuevas funcionalidades para mejorar la experiencia de usuario, como, por ejemplo, la elaboración de una interfaz y los elementos que la componen.

Análisis de riesgo: aquí se evalúa todo lo que la nueva implementación pueda afectar al proyecto. Si se añade un nuevo principio físico a nuestro programa, hay que evaluar el posible deterioro que se podría producir en otras implementaciones previas. Conforme más iteraciones y más funcionalidad haya, más difícil será superar el

análisis de riesgo. Por ejemplo, si se quiere añadir una funcionalidad nueva para mejorar el programa visualmente, ésta no puede producir errores en otros módulos del proyecto, como en la parte del cálculo de flujo de energía. Por ello se tiene que evaluar los riesgos que puede producir cada cambio para evitar problemas futuros.

Implementación: se desarrolla el software acordado en la planificación. Aquí es cuando toda la parte teórica de la investigación se adapta para ponerla en práctica. Esta interpretación de conceptos teóricos tiene que ser correcta para evitar confusiones y errores en el programa. Una vez producido el código que implemente los requisitos acordados, se realizarán pruebas de testing para verificar que el programa funciona como se espera.

Evaluación: antes de terminar el ciclo se debe de analizar al detalle lo que sucede en la aplicación para determinar riesgos y posibles errores que pueden surgir en un futuro y tenerlos en cuenta en la próxima iteración. En este proyecto, puede haber una falsa sensación de que el programa funciona adecuadamente, pero en realidad no realiza las operaciones en el orden o la forma correctas. Si la aplicación de un nuevo principio físico, evita que funcione otro, se detectará en esta fase y se arreglará en la siguiente iteración.

1.4. Estructura del documento

Resumen: breve explicación sobre lo realizado en este trabajo.

Abstract: breve explicación sobre lo realizado en este trabajo, pero en inglés.

Introducción: pone en contexto el proyecto que se va a desarrollar.

Descripción del problema: se presentan los problemas principales para implementar este simulador.

Objetivos: los objetivos finales del trabajo.

Metodología: presenta la metodología de desarrollo software utilizada.

Antecedentes: pone en situación las bases del desarrollo.

Antecedentes geológicos: base geológica de la que se sustenta el proyecto.

Estudios previos: pruebas con otra ley física distinta.

Conceptos previos: explicación de los conceptos principales que se han estudiado e implementado.

Principios físicos: base científica del trabajo.

Estructura y almacenamiento: presentación y explicación de la estructura de datos utilizada para la resolución del problema.

Software y recursos: recursos software evaluados.

Análisis previo: búsqueda del entorno de trabajo, probando distintos programas y lenguajes, evaluando sus pros y sus contras.

Software utilizado: evaluación de las ventajas y desventajas del software elegido, facilitando una breve explicación del motivo de su elección.

Presupuesto: coste que tendría el proyecto en un entorno de trabajo.

Planificación y desarrollo del proyecto: desglose de la división de trabajo realizada durante el desarrollo, aplicando la metodología software explicada previamente.

Semana 1 – Construcción de Autómata: inicialización del proyecto.

Semana 2 – Conceptos físicos: aplicación de los conceptos físicos a la aplicación.

Semana 3 – Intercambio de calor y de posición: primer principio físico implementado en el autómata.

Semana 4 – Construcción del entorno: parametrización y creación de la cueva.

Semana 5 – Construcción de la geometría: creación del autómata con sus elementos en función de la forma de la cueva.

Semana 6 – Interfaz: implementación de una interfaz para mejorar la usabilidad.

Semana 7 – Ampliación de principios físicos: mejora del funcionamiento del simulador implementando nuevas operaciones sobre el autómata celular.

Semana 8 – Lectura del archivo y construcción de un wireframe: almacenamiento de los datos y variables que se simularán y creación de un wireframe para mejorar visualmente la evolución del entorno.

Semana 9 – Aplicación de velocidad de intercambio: implementación de un nuevo principio físico para aumentar el realismo de la simulación.

Semana 10 – Obtención de resultados: generación de un vídeo mostrando el resultado de la ejecución para poder realizar estudios sobre lo simulado.

Explicación del desarrollo: descripción técnica de como se han resuelto e implementado los distintos problemas y funcionalidades.

Creación del autómata celular: almacenamiento y organización de los distintos elementos sobre los que se harán las futuras operaciones.

Sistema de intercambio de energía y parámetros: implementación del sistema de intercambio aplicando los conceptos físicos y parametrizando sus variables.

Geometría: construcción de la cueva mediante un modelo 3D

Problema de la geometría: explicación del por qué algunas geometrías son válidas y otras no.

Generación de capas del autómata: corresponde con la parte procedural del autómata.

Interfaz: presentación de la disposición y elementos de las interfaces.

Lectura del archivo: se obtiene los datos para la simulación.

Escala de colores: definición de los colores utilizados para la representación visual de la temperatura de un elemento.

Pruebas y resultados: elaboración de un vídeo a partir de la ejecución del simulador.

Conclusión y trabajo futuro: evaluación del resultado obtenido y el alcance que tendría el continuo desarrollo del trabajo.

Bibliografía: referencias bibliográficas del proyecto.

Anexo: información extra para facilitar información.

Anexo I: Instalación de Jupyter: pasos para instalar y ejecutar el código de dicha parte.

Anexo II: Elaboración de un vídeo a partir de las pruebas obtenidas: instrucciones para ejecutar un pequeño script que genera el vídeo de la ejecución.

2. Antecedentes

2.1 Antecedentes geológicos

En este proyecto necesitaremos información de cómo es la topología de una cueva para hacer un entorno lo más realista posible. Nuestra cueva en el prototipo tendrá una diagonal y mínimo dos entradas, una en la parte superior y otra en la inferior. Estas entradas/salidas de la cueva tienen que tener un tamaño específico. La boca superior no puede ser más grande que la boca inferior. En el caso de que haya más de una boca superior, la suma del tamaño de las bocas, no debe superar al tamaño de la boca inferior [*79*].

Como se puede ver en la *Ilustración 2*, la boca inferior debe de ser más grande que el resto. Y no solo esto, también el tamaño de las bocas viene determinado por el tamaño total de la roca. Normalmente, estas bocas inferiores tienen un total de un metro cuadrado de tamaño. Suelen ser formas ovaladas, de forma que de largo será un metro y medio (por ejemplo) y medio metro de ancho.

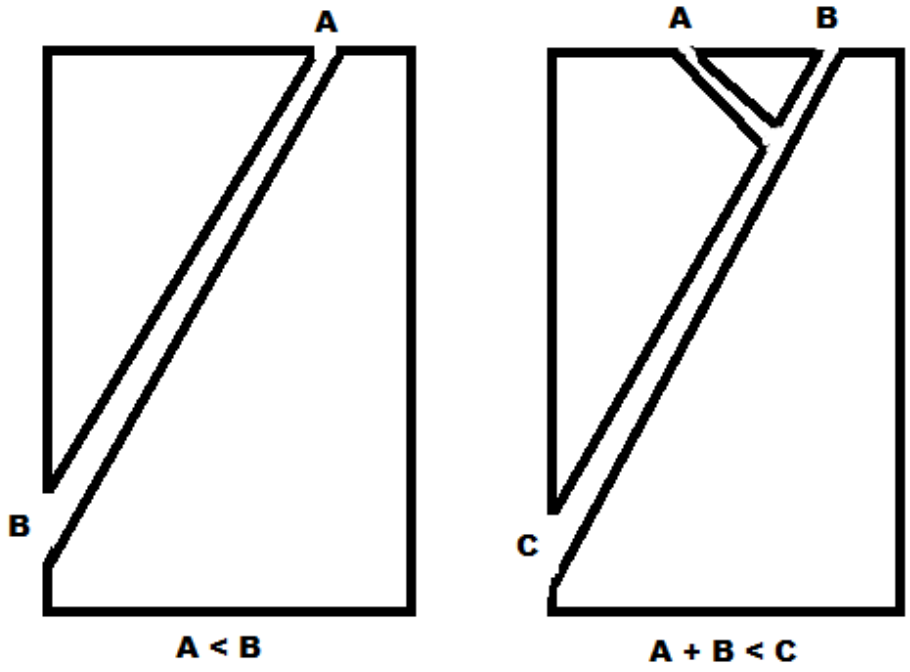

Ilustración 2. Representación esquemática de la cueva modelada. Elaboración propia.

Como se aprecia en la *Ilustración 3*, para una roca con una altura de treinta metros, la boca debería de tener un tamaño de un metro y medio, que es un cinco por ciento del tamaño de la roca. Lo ideal sería que esta boca tenga entre un cuatro y un ocho por ciento del tamaño total. Esto nos va a definir el tamaño máximo de las otras bocas superiores de la cueva.

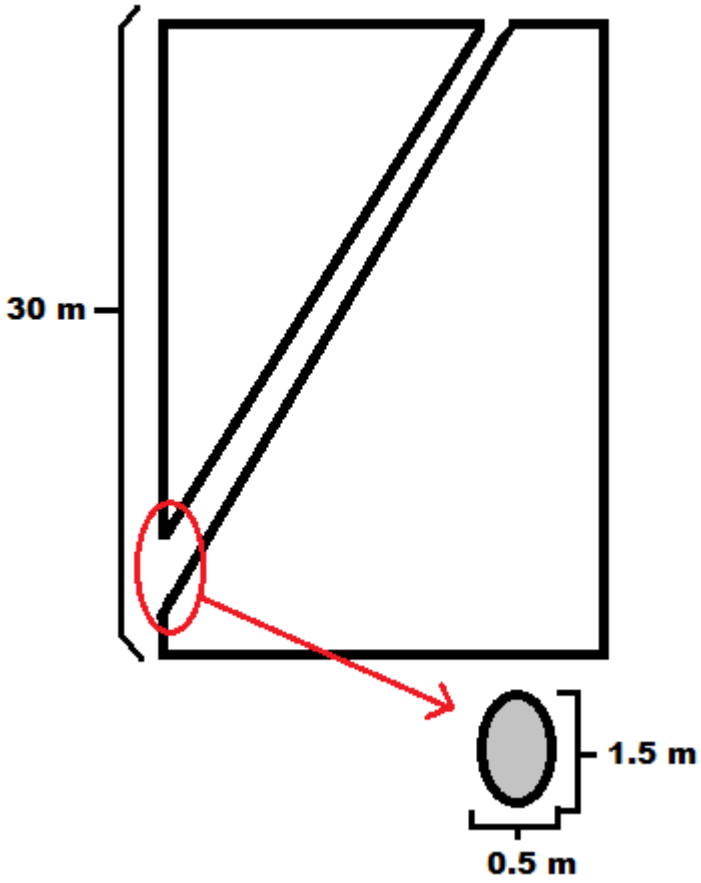

Ilustración 3. Representación esquemática del tamaño de las salidas de la cueva. Elaboración propia.

2.2 Estudios previos

Jupyter Lab aplicando Navier-Stokes: las librerías a utilizar son NumPy y Matplotlib. Para instalar las librerías, ver el [*Anexo I*].

Ejemplo 1: Ahora ya podremos usar el código de Navier Stokes [*Ejemplo Navier-Stokes*] y ver una representación con el siguiente ejemplo.

Una de las gráficas tiene una representación como la *Ilustración 4*:

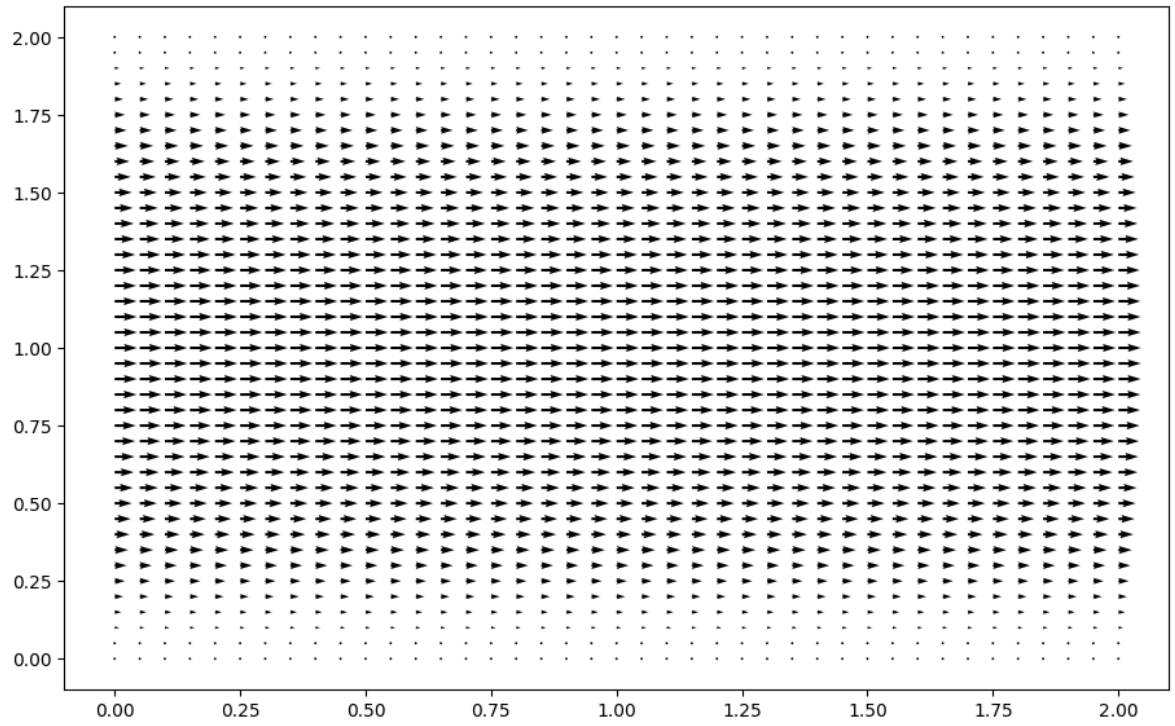

Ilustración 4. Solución Navier-Stokes con F=1. Elaboración propia.

Si invertimos la fuerza (variable F = -1):

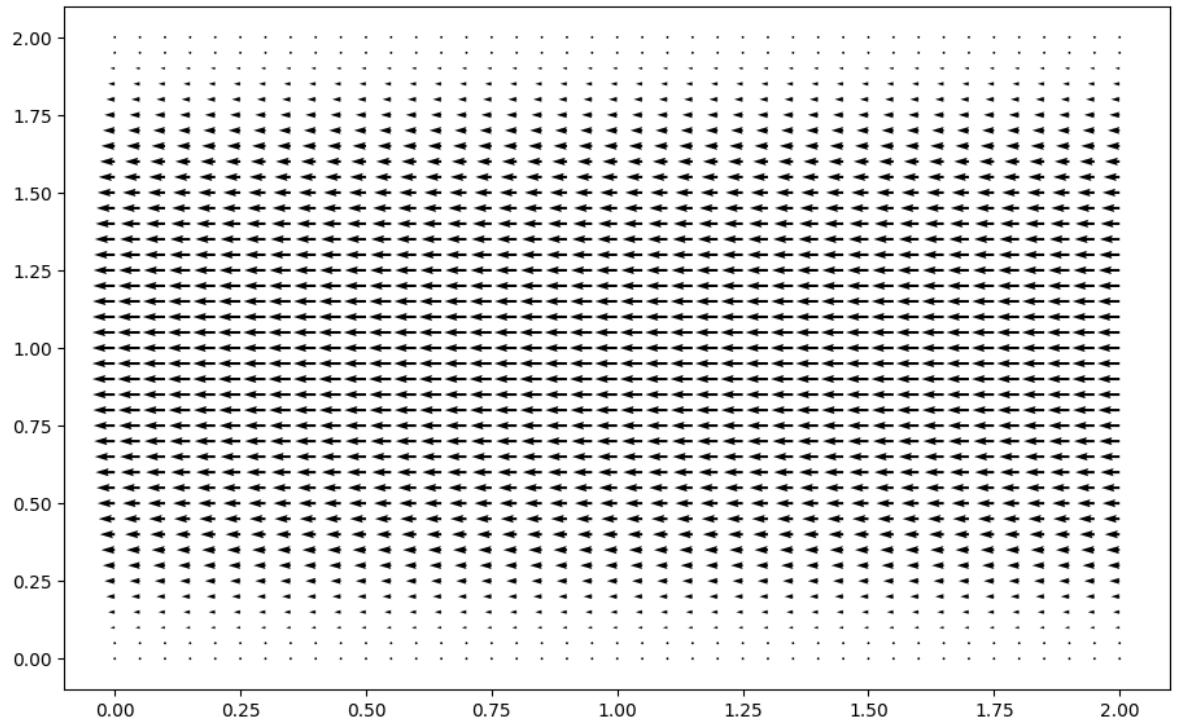

Ilustración 5. Solución Navier-Stokes con F=-1. Elaboración propia.

En la representación de la *Ilustración 5* se aprecia la fuerza o la cantidad de movimiento que tendría una partícula dependiendo de la posición de la gráfica en la que se encuentre. Se puede apreciar el flujo de movimiento en un entorno simple sin ningún obstáculo.

Representación en 3D con F = 1 y F = -1 respectivamente:

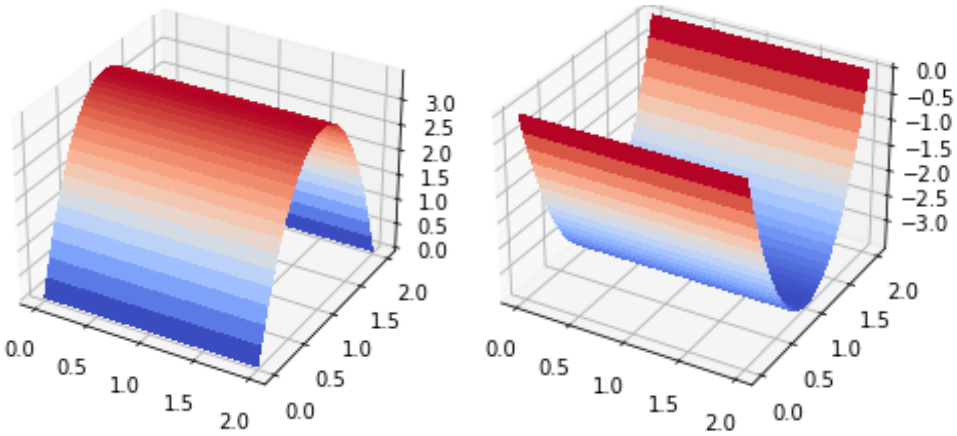

Ilustración 6. Representación de Navier-Stokes con una gráfica 3D. Elaboración propia.

Ejemplo 2: Una representación en 3D (`matplotlib.figure.gca.plot_surface`) con otro flujo distinto [*Ejemplo Navier-Stokes*]

```
u[0:15, 0:15]=2

for n in range(nt+1):
    un[:] = u[:]
    u[1:-1, 1:-1] = un[1:-1,1:-1]+\
        nu*dt/dx**2*(un[2:,1:-1]-2*un[1:-1,1:-1]+un[0:-2,1:-1])+\
        nu*dt/dy**2*(un[1:-1,2:]-2*un[1:-1,1:-1]+un[1:-1,0:-2])

    u[0,:]=1
    u[-1,:]=1
    u[:,0]=1
    u[:,-1]=1
```

Ilustración 7. Código Python utilizado para calcular Navier-Stokes. (https://2013.es.pycon.org/media/pycones-CAChemE-python-ingeneria-quimica.pdf)

Hay una variable *nt* que representa el intervalo de tiempo en el que se encuentra el flujo. Como se puede apreciar en la *Ilustración 8.* Representación Navier-Stokes del código de la Ilustración 8. Elaboración propia., si se modifica esta variable de forma

sucesiva, en cada iteración se vería como el fluido tiende a unificarse, alcanzando el mismo valor que el resto de elementos.

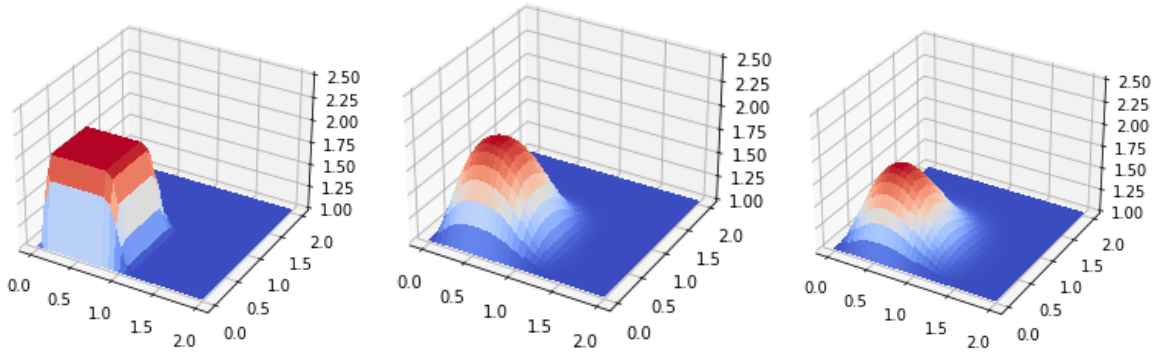

Ilustración 8. Representación Navier-Stokes del código de la Ilustración 8. Elaboración propia.

Por lo que en el intervalo 120 de esta variable, el fluido tiene que estar casi plano como se aprecia en la *Ilustración 910*:

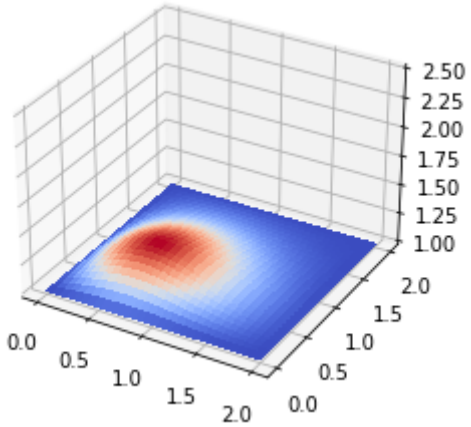

Ilustración 9. Representación Navier-Stokes del código de la Ilustración 7 con nt=120. Elaboración propia.

Todas estas gráficas representan el valor de la matriz *u*, que se puede extrapolar a unas partículas indicando su posición en el eje Y (en el entorno 3D) y moviéndolas en esa dimensión únicamente.

3. Conceptos previos

3.1. Principios físicos

En este prototipo es esencial el intercambio de información entre los distintos elementos de autómata. Esta información debe de modificar las condiciones en las que se encuentra la célula, y en función de esas condiciones, obtener una visualización u otra. Nuestro parámetro principal es la temperatura. Dependiendo de este valor, tendrá un color u otro, y se le aplicarán unas físicas u otras. El mecanismo principal es el intercambio de temperatura que hay entre dos células, o lo que viene a ser, calcular el flujo de energía.

La idea principal es hacer un sistema procedural dónde las células se crean o se destruyan en función de los parámetros de entrada. Por ejemplo, podemos tener una temperatura ambiental y una temperatura puntual (de una roca). Este cambio de energía tiene que aplicarse por cada célula en función de sus vecinos, por lo que cada célula tiene que cambiar su temperatura en ciertas condiciones. Para el cálculo de la nueva temperatura de la célula se utilizará la Ley de Fourier. Así calcularemos el flujo de energía entre los distintos elementos del sistema. Esta ley tiene en cuenta los siguientes parámetros [*Ley de Fourier*]:

$$\alpha\left(\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}+\frac{\partial^2 T}{\partial z^2}\right)+\frac{q_G}{\rho C_p}=\frac{\partial T}{\partial t}$$

$$q = -k\nabla T$$

$$\alpha = \frac{k}{\rho C_p}$$

$$\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}+\frac{\partial^2 T}{\partial z^2} \equiv \nabla^2 T$$

donde:

- $\boldsymbol{k}$, es la conductividad térmica del material
- $\boldsymbol{\rho}$, es la densidad del fluido
- $\boldsymbol{C_p}$, es el calor específico del material

- $\boldsymbol{\nabla^2 T}$, es el gradiente de la temperatura
- $\boldsymbol{t}$, el tiempo transcurrido

El gradiente de la temperatura se calculará en función del factor de conductividad $\boldsymbol{k}$ y la temperatura entre la célula actual y con la que intercambia energía [79]. De forma que:

$$\nabla^2 T = \frac{(k_1 T_1 - k_2 T_2)}{k_1 + k_2}$$

Este gradiente sólo es aplicable al aire porque tienen la misma conductividad térmica. Si lo hiciésemos entre una célula de aire y otra de roca, al tener distinta conductividad, no saldría un flujo correcto. Al ser un factor tan alto, tiene demasiado 'peso' en la fórmula y casi siempre perdería o ganaría energía, dependiendo de si el factor de la roca es el k1 (sumando) o el k2 (restando). Esto provoca que, en la iteración entre dos células, las dos ganen o pierdan energía, cuando entre ellas es imposible que ocurra, ya que una va a ganar y otra a perder, tendiendo al equilibrio térmico.

Por ende, en la iteración entre roca y aire, se utilizará una diferencia de temperaturas básica para aplicarlas como gradiente a la fórmula de la Ley de Fourier.

3.2. Estructura y almacenamiento

Una vez resuelta la física que se va a utilizar, se pueden dotar a las células de estas propiedades, aplicando distintos valores para los distintos materiales. Estas células se agrupan en un autómata celular. Un A.C. no es ni más ni menos que un modelo computacional para un sistema dinámico, como el nuestro, y que evoluciona a pasos discretos. Esta estructura es ideal para modelar sistemas naturales compuestos por una colección masiva de objetos simples (las células) que interactúan entre ellos. En este trabajo se han definido unas células con unas propiedades y cada una se comportará de una manera u otra, componiendo el autómata celular.

En cuanto a una célula, es el elemento más simple del autómata dotado de las características necesarias de nuestro sistema natural para poder resolver el problema.

Cada célula colinda con otras formando un entorno de vecindad que es fijo, puesto que las células no se mueven en el espacio.

Un entorno es uno de los conceptos básicos de la topología. Matemáticamente hablando, el entorno de un punto es el conjunto de puntos que se encuentran próximos a éste. Para nuestro problema se aplica de la misma forma, pero con células y lo llamaremos entorno de vecindad. Así indicamos que cada célula tiene sus vecinos, que en este trabajo corresponden con los elementos que está adyacentes a este.

Computacionalmente hablando, hay que almacenar estas células y darles una estructura de datos para poder acceder a ellas y realizar operaciones. Si nos imaginamos un autómata cúbico (en tres dimensiones), éste tiene tres ejes. Por lo que lo ideal sería almacenarlo en una matriz tridimensional, dónde cada posición corresponde con la posición real en la escena. Esto quiere decir que, si queremos acceder a la célula que está encima de otra, nos resulta bastante sencillo encontrar ese vecino puesto que corresponde justo con la que tiene encima en la EEDD. Este tipo de almacenamiento nos permite hacer operaciones complejas con la vecindad, aplicando los principios físicos con mayor precisión evitando búsquedas erróneas. Si se quisiese utilizar un árbol para una búsqueda eficiente del elemento en cuestión, habría que realizar un amplio estudio de qué se debería de almacenar en cada nodo para hacerla lo más eficiente posible y evitar duplicar memoria, almacenando los vecinos de cada célula por nodo, por ejemplo.

4. Software y recursos

4.1. Análisis previo

Javascript: Se estudió la eficiencia de partículas en Javascript. El principal problema es que no se pueden crear muchas partículas y se proyecta en dos dimensiones. Pocas librerías para la visualización y poco óptimas. La principal ventaja es que se puede hacer una aplicación muy interactiva con el ratón.

OpenGL en C++: Es el más eficiente. Se puede controlar el proceso de renderizado asignando únicamente los buffers necesarios para cada proceso, haciendo que se optimice mucho. El principal problema es la complejidad y la visualización para hacer pruebas. El entorno de visualización es en tres dimensiones y se pueden parametrizar muchas de las variables de dibujo.

VTK: Este es un sistema que permite hacer programas gráficos en tres dimensiones. Usa varias bibliotecas de clases de C++. El principal problema es la poca flexibilidad que ofrece, así como la posibilidad de parametrizar lo mayor posible. Y al generar tantas partículas/células no es eficiente.

LiquidFun: Aplicación desarrollada por Google que simula muy bien fluidos [*79*, Google]. El problema es que ya están las físicas implementadas y no hay mucha flexibilidad para los cambios.

Python: Hay librerías bastante buenas para hacer cálculos tanto matemáticos con estructuras sencillas como vectores. El principal problema es la representación visual. Se puede representar en un entorno en tres dimensiones, pero es algo complejo y se visualiza realmente mal. Pygame es una librería de dibujo en dos dimensiones para la representación de nuestro trabajo. Pero aun así es demasiado limitada.

La librería *pygame* te permite dibujar, pero solamente en un entorno 2D. Un ejemplo en dos dimensiones donde solo cambia la posición de la partícula en vertical, variando el instante de tiempo (*nt*).

En este ejemplo de la se generan unos puntos y se colocan en un vector unidimensional. Utilizaremos la librería NumPy para hacer los cálculos entre valores

de la estructura de datos. Una vez sepamos el valor resultante de los parámetros, se lo aplicamos a la posición de ese punto en el eje Y. De esta forma se representa visualmente el valor dado por la ecuación de Navier-Stokes. Solo tenemos que ir aumentando el valor de *nt* a través de alguna entrada de teclado y ver cómo se mueven los puntos.

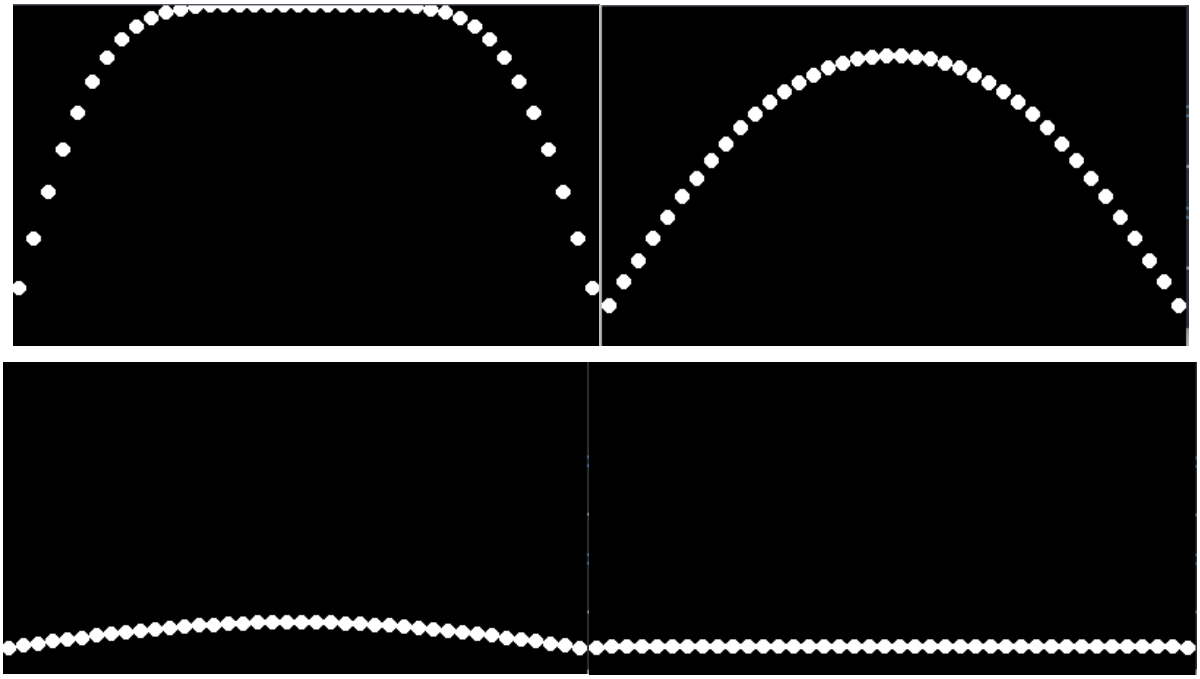

Ilustración 10. Representación de F de Navier-Stokes en pygame. Elaboración propia.

En vez de poner la fuerza en el centro podemos cambiarla hacia un lado como se aprecia en la *Ilustración 11*, esto va en función de cómo se inicializa la matriz *u*:

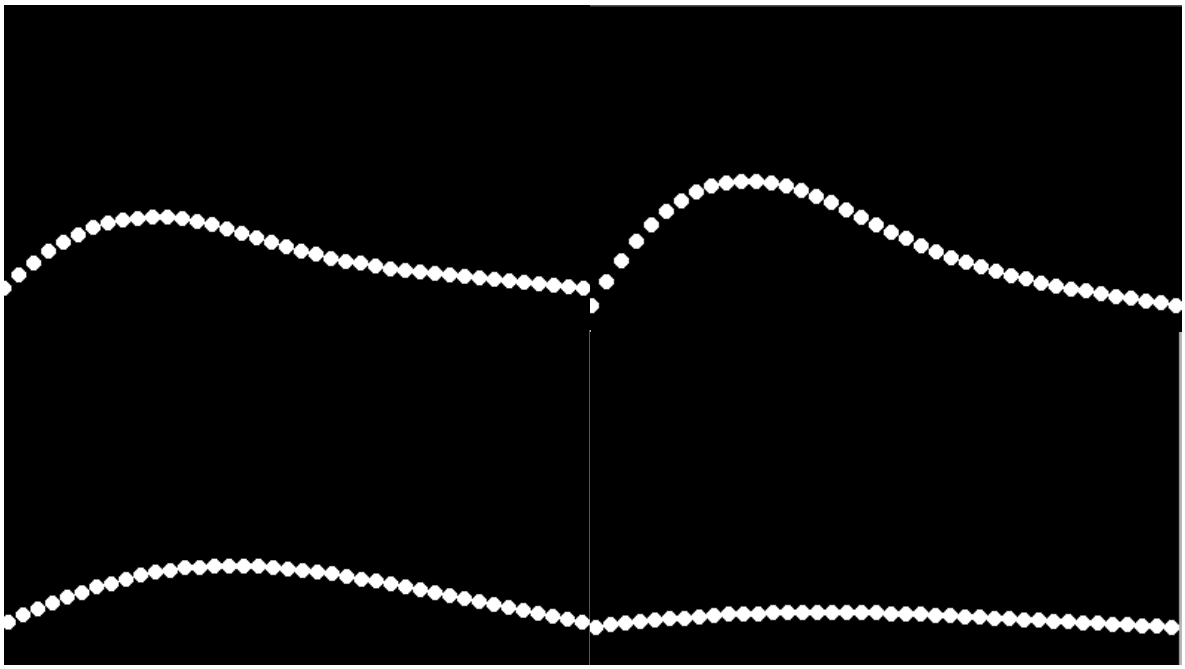

Ilustración 11. Representación del ejemplo 2 de Navier-Stokes en pygame. Elaboración propia.

4.2. Software utilizado

Unity: Se hicieron varios estudios en este motor de videojuegos con distintos sistemas y con distintos comportamientos. El lenguaje de programación es C# [*79*].

Para calcular la temperatura de una célula se tendrán en cuenta a todos sus vecinos, haciendo una **temperatura media**. Esto dará un valor más alto o más bajo según la temperatura de la célula actual. Si la temperatura media es mayor, significa que esta célula está rodeada de altas temperaturas, por lo que gana energía y viceversa. Se hará uso de un parámetro que simulará la temperatura del exterior. Esta temperatura cambiará automáticamente el valor de los elementos externos del sistema. En función de donde se aplique este valor, se corresponderá con la configuración de un entorno u otro.

Pero en la ejecución se debe de visualizar el cambio de información entre elementos, en este caso, la temperatura de cada uno. Dependiendo de la temperatura, el elemento tendrá un color u otro. Irá desde el más caliente de color rojo, hasta el más frío que es de color azul.

Esta solución es la misma que en un entorno 2D, solo que ahora hay un mayor número de partículas y de vecinos, aumentando el tiempo de cómputo exponencialmente.

Como se ve en la *Ilustración 12* se ha creado un autómata simple con células esféricas. Unity permite una buena visualización del entorno, con posibilidad de hacerlo lo más interactivo posible. Aquí, todas las células tienen el mismo valor de alta temperatura, por ello tiene tonalidad roja.

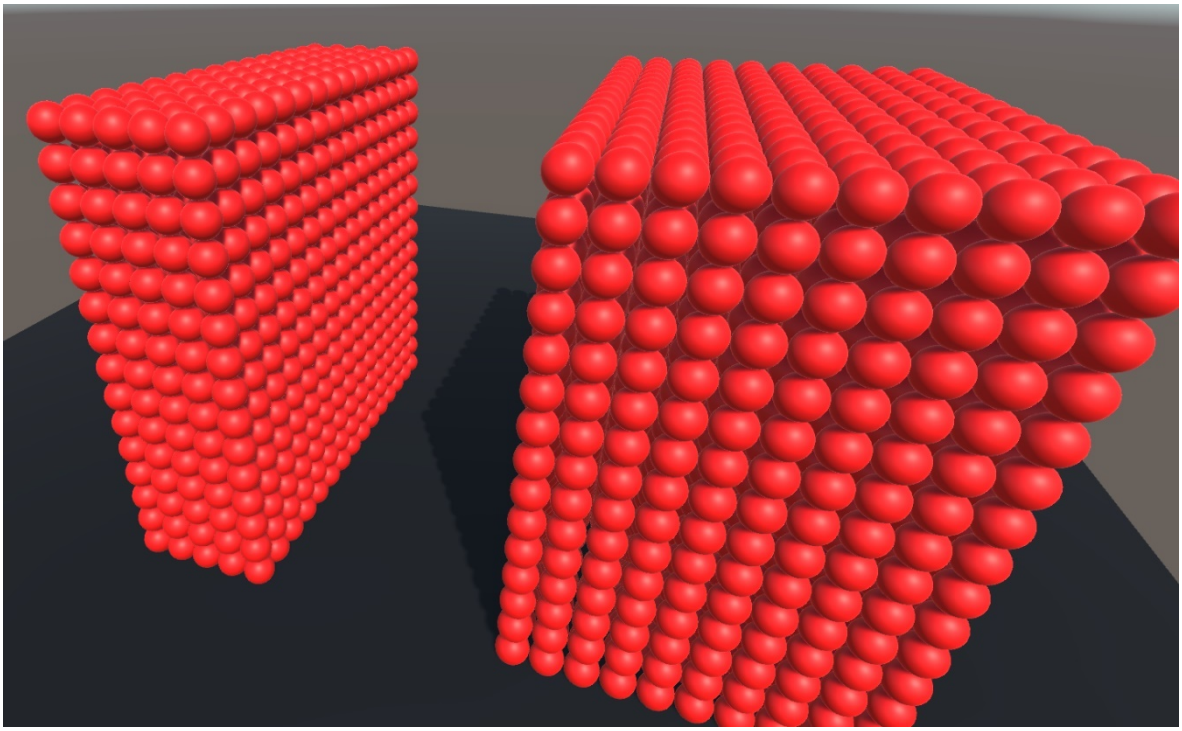

Ilustración 12. Representación de un sistema de células en Unity. Elaboración propia.

Como se aprecia en la *Ilustración 13* al cambiar la temperatura que rodea el exterior del autómata, el interior va a ir cambiando también. Esto sucede porque cada célula obtiene el valor de temperatura media de los vecinos que le rodean, y poco a poco, esta energía llegará al centro del autómata. Las únicas células que no varían de sus vecinos son las externas, es decir, los límites del autómata. Estás tomarán valor de un parámetro que define una temperatura externa de nuestro entorno.

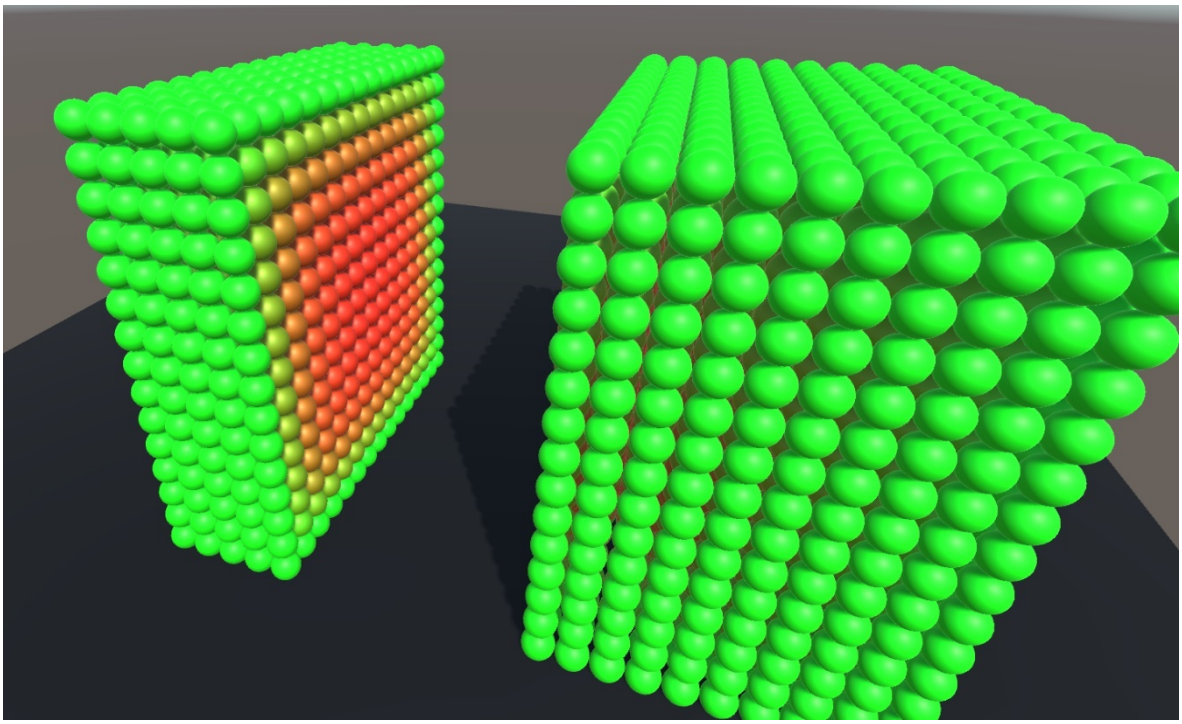

Ilustración 13. Representación del sistema en funcionamiento. Elaboración propia.

En la *Ilustración 14* la temperatura externa ha bajado al mínimo y se pueden apreciar los distintos colores que forman las distintas temperaturas.

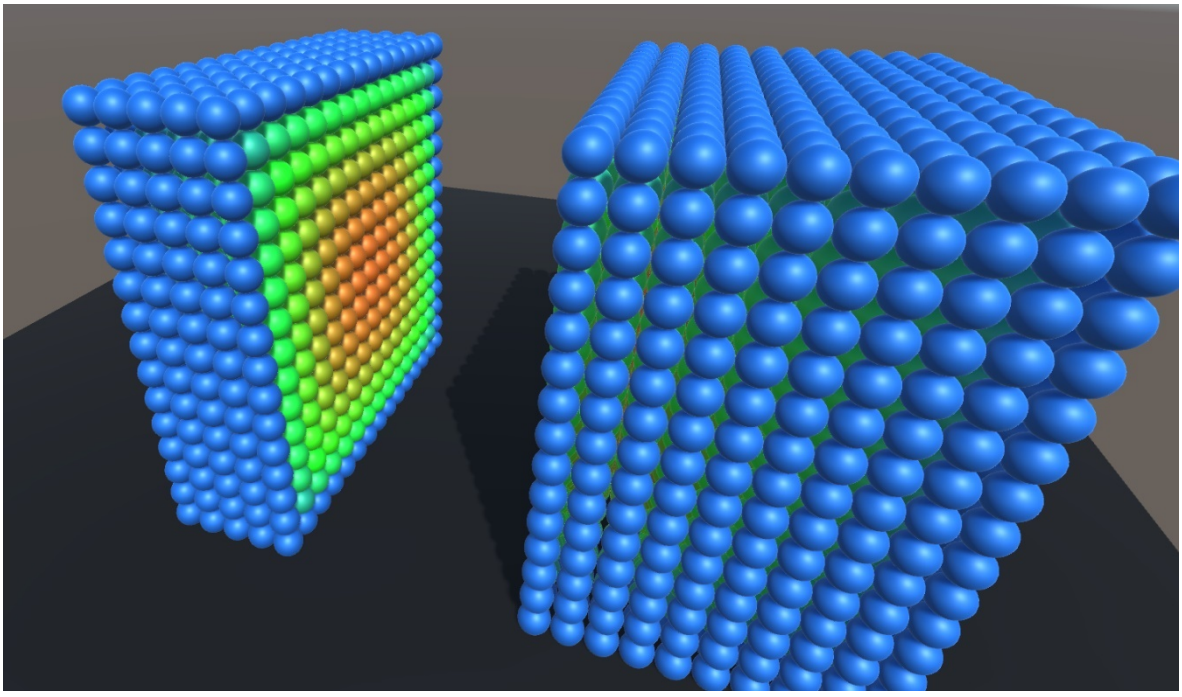

Ilustración 14. Representación del frío entrando en el sistema. Elaboración propia.

Una vez se vuelve el sistema estable, con todos los elementos a la misma temperatura, ésta empieza a subir hasta el máximo, apreciándose la escala de colores de la *Ilustración 15:*

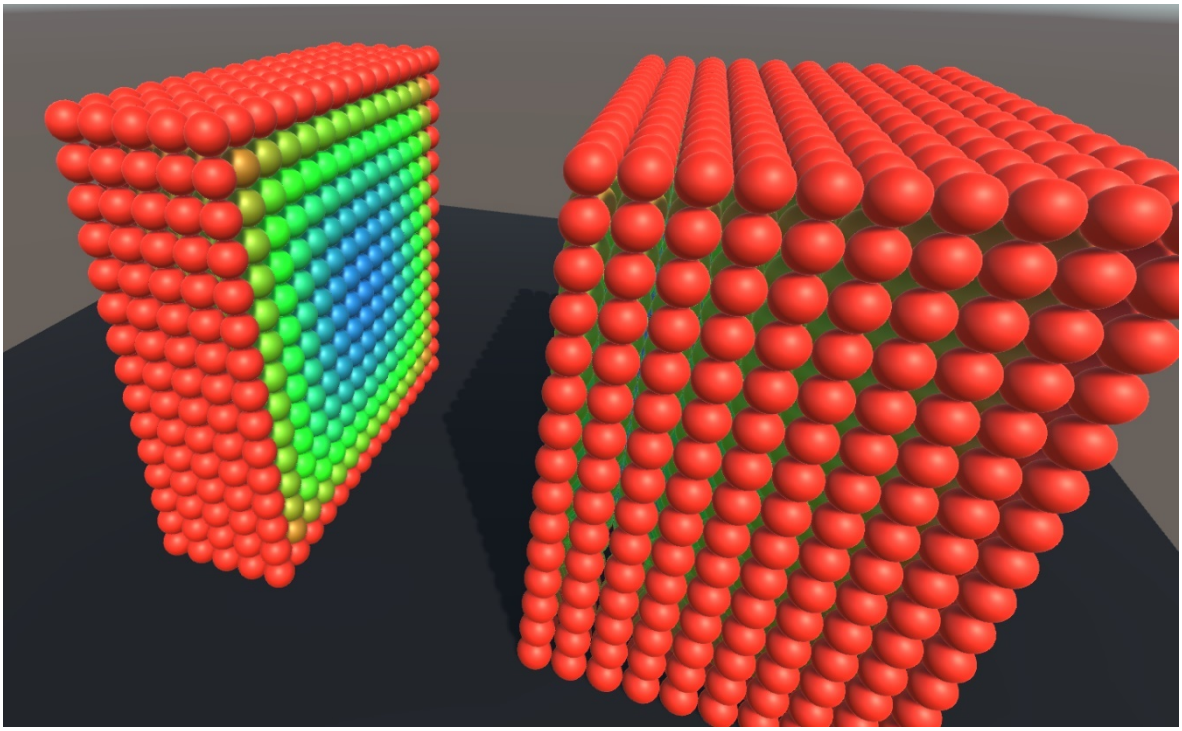

Ilustración 15. Representación del calor entrando en el sistema. Elaboración propia.

Ahora se va probar cambiar de posición el foco de energía, situándolo debajo del autómata. En la el autómata está estable, teniendo una temperatura templada en su totalidad.

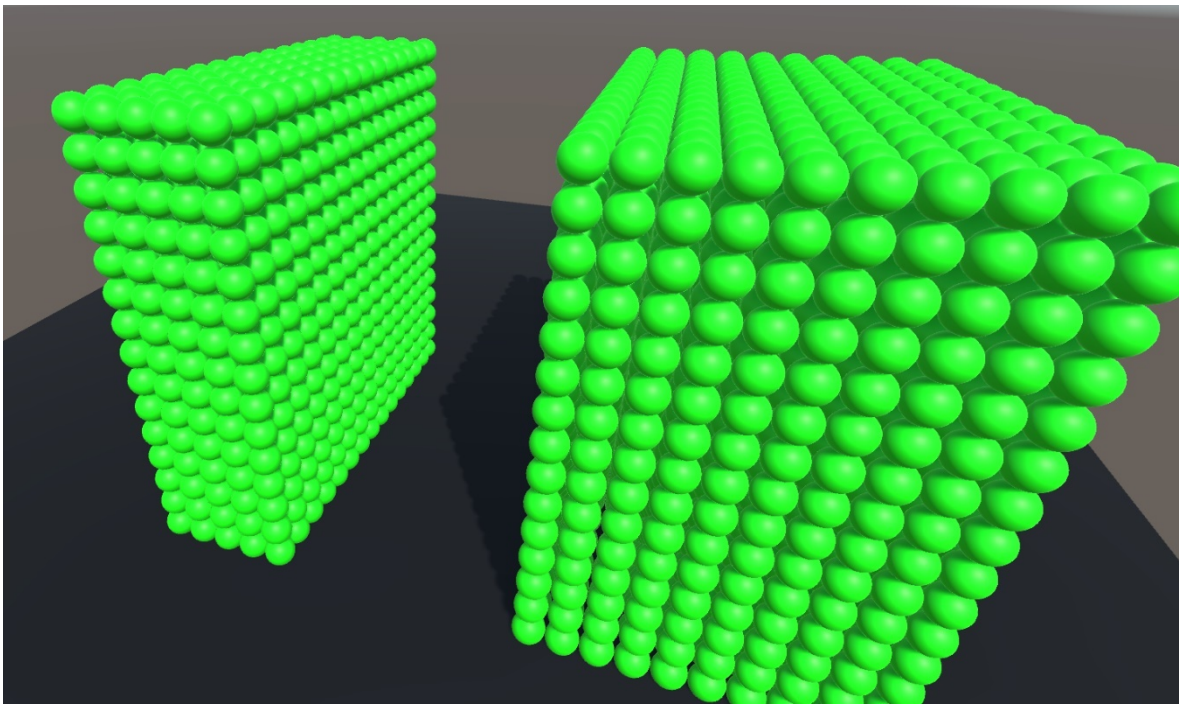

Ilustración 16. Representación del mismo sistema, pero con otro foco de calor. Elaboración propia.

A continuación, en la *Ilustración 17,* el foco comienza a calentar la parte inferior del autómata. La parte superior de este representa el exterior, que se encuentra más frío que el foco de calor, por lo que comienza a apreciarse con distinta tonalidad.

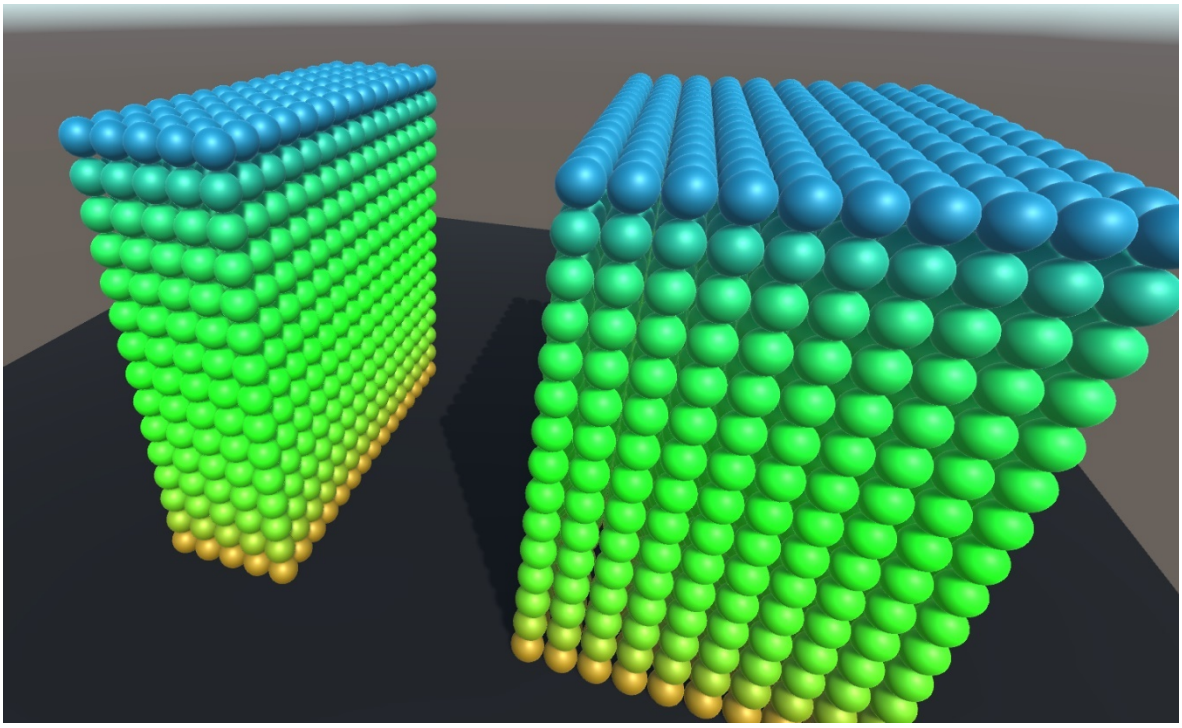

Ilustración 17. Representación del sistema calentándose por abajo y enfriándose por arriba. Elaboración propia.

Y cuando se alcanza el máximo de temperatura del foco, se puede apreciar el gradiente de la gama de colores que representa la temperatura como en la *Ilustración 18* .

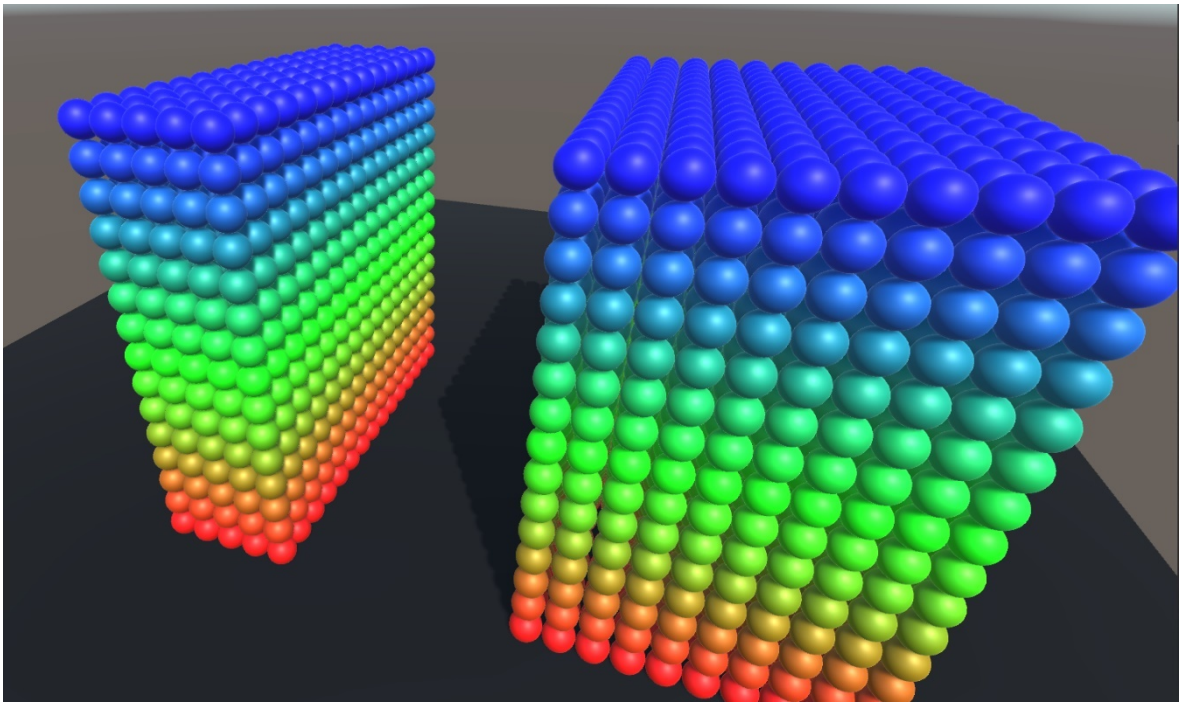

Ilustración 18. Representación del sistema calentándose por abajo y enfriándose por arriba. Elaboración propia.

Se concluyó que visualmente quedaba más bonito que el resto de aplicaciones y al estar orientada para la visualización e iteración con un entorno, se decidió usar Unity para este proyecto.

4.3. Presupuesto

El presupuesto necesario para la realización de este proyecto teniendo en cuenta tanto los recursos hardware como software es:

- **Recursos hardware:** para la ejecución de este trabajo se necesitará un ordenador con buenos componentes gráficos debido a la gran cantidad de operaciones que se van a realizar en el entorno gráfico utilizado. También es esencial que el disco duro sea SSD para una mayor velocidad de transferencia de datos. Por ende, el precio rondaría unos 1.250€ para poder ejecutar todas las operaciones en un tiempo razonablemente bueno.

- **Recursos software:** el principal recurso software a utilizar es *Unity*, junto con otro programa para editar los distintos scripts de la aplicación, que será *Visual Studio Code*. Ambos programas son de coste cero.
- **Trabajo autónomo:** las horas aproximadas para realizar este trabajo ronda unas 300 horas, dónde no solo hay que programar el código de la aplicación. También hay que realizar estudios e investigaciones sobre el tema en concreto para poder aplicarlas posteriormente al programa. La jornada media anual en España, según la referencia [*79*], es de 1766 horas. Si un Ingeniero Informático cobra de media 36.500€ anuales en España [*79*], el salario por hora saldría unos 20,66€. Por lo que el precio del trabajo autónomo rondaría los 6.198€.

Por lo que el trabajo completo, tendría un presupuesto aproximado de 7.448€

5. Planificación y desarrollo del proyecto

Como se puede apreciar en la *Ilustración 19*, se ha divido el trabajo con el objeto de que se necesiten unos requisitos mínimos para pasar a la siguiente fase del desarrollo. Por consiguiente, tendremos una mayor organización para modular mejor la aplicación y mejorar su escalabilidad.

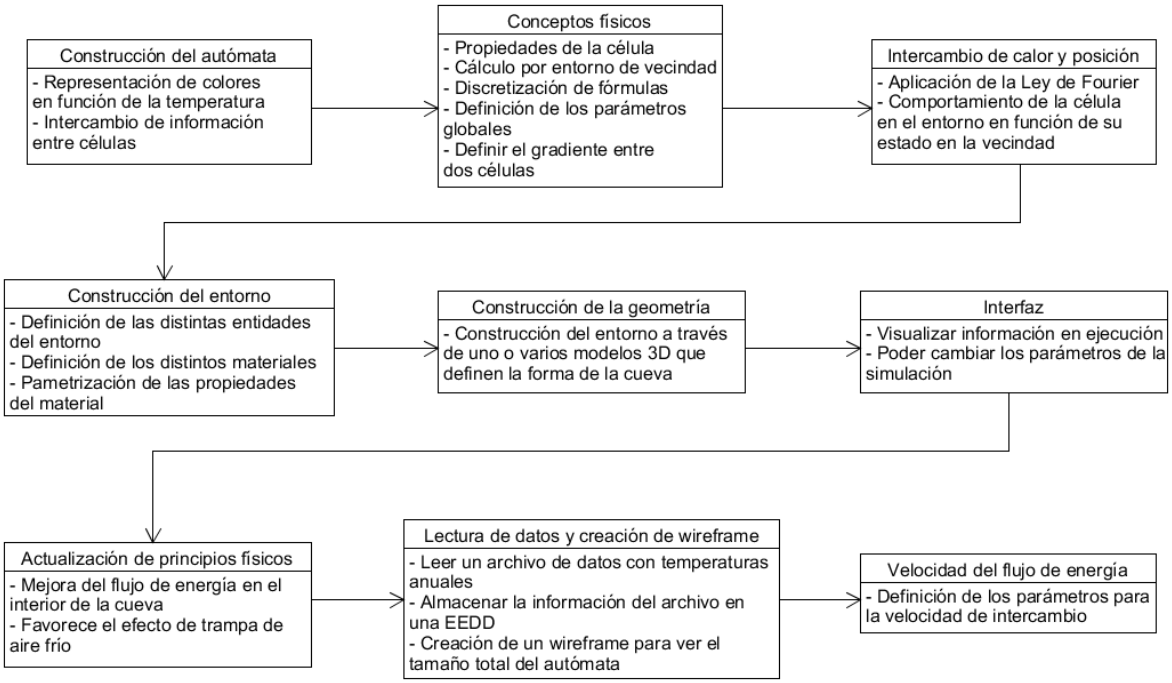

Ilustración 19. Esquema con la planificación del trabajo. Elaboración propia.

En la *Ilustración 20* se observa el diagrama de Gantt con las tareas repartidas en las semanas de desarrollo. Por ejemplo, el sistema de intercambio de energía se ha ido modificando a lo largo del proyecto para añadir nuevos principios físicos, por eso se retoma la actividad varias veces.

Mes	Marzo					Abril					Mayo
Nº Semana	1	2	3	4	5	5	6	7	8	9	10
Días	1 al 5	8 al 12	15 al 19	22 al 26	29 al 31	1 y 2	5 al 9	12 al 16	19 al 23	26 al 30	3 al 7
Creación del autómata											
Sistema de intercambio											
Geometría											
Generación procedural											
Interfaz											
Lectura archivo											
Escala de colores											
Pruebas y resultados											

Ilustración 20. Diagrama de Gantt con la planificación del proyecto. Elaboración propia.

5.1. Semana 1 - Construcción del Autómata

En los primeros pasos de este proyecto, se decide crear un prototipo de autómata celular, donde todas las células tenían el mismo comportamiento y no tenían propiedades físicas. El objetivo de esto era muy simple: Intercambiar información entre las distintas células en función de su entorno. Por lo que, para cada célula, hay que mirar las que le rodean o séase, sus vecinos. Para calcular los vecinos de cada célula se hizo de una manera sencilla. Todas las células se almacenan en una matriz 3D, puesto que estamos trabajando en un entorno en tres dimensiones. Y cada posición de la matriz, corresponde con una posición real del entorno, es decir, dos células que se almacenan juntas en la EEDD, van a estar juntas también en la escena. De esta forma, por cada célula, miramos en sus vecinos de la estructura de datos, que a su vez se corresponderán con sus vecinos reales.

El único funcionamiento es el de pasar información de temperatura entre ellas, y en función de esa temperatura, la célula adquiere un color u otro. Por lo que hay que hacer los cálculos pertinentes de color en función de la temperatura y un rango de temperaturas. Este rango de temperaturas también es variable, es decir, se va a definir una vez termine de leer todo el archivo de datos de las temperaturas para saber cuál es la máxima y cuál es la mínima. Esta parametrización es fundamental por si se quiere cambiar de archivo o lo que es lo mismo, cambiar el rango de temperatura en la que se simulará el prototipo. El color también es vital para la representación, ya que, si no se eligen bien los colores, no se apreciaría el cambio de información.

5.2. Semana 2 – Conceptos físicos

Como nuestro prototipo es una simulación, se ha investigado sobre los fundamentos físicos en los que basarnos para el intercambio de energía del fluido. Los cálculos se realizan en función de la ley de transferencia de calor de Fourier. Esto va a definir unas propiedades físicas a las células. De forma que cada una de ellas tendrá una densidad, un calor específico y un factor de conductividad térmica, entre otros. Dependiendo de estas propiedades, los distintos fluidos se comportan de manera distinta.

Pero estos nuevos objetivos pueden dar lugar a problemas futuros, las fórmulas hay que *discretizarlas* puesto que vienen en funciones con derivadas y segundas derivadas. A esta discretización se le pasan las propiedades del material y otras variables que influyen en la fórmula para obtener un valor final que será la cantidad de energía. Por ende, es esencial hacer una buena discretización para realizar las operaciones con los datos correctos. Posteriormente, esta energía se tiene que transformar a grados, puesto que se aplica a la temperatura de cada célula.

Todas estas fórmulas y variables se aplican a cada célula, y en principio, todas con las mismas propiedades.

Una vez implementados los nuevos requisitos, se evalúa el alcance de los nuevos objetivos. Al probar el programa con las fórmulas, se aprecia que intercambia medianamente bien la energía entre células, pero hay reglas físicas que no se cumplen. Como, por ejemplo, el peso del aire. Nosotros definimos una densidad para calcular el flujo, pero esta densidad varía en función de la temperatura actual de la célula. El aire frío pesa más que el aire caliente. Por lo que el aire frío tiende a bajar.

5.3. Semana 3 – Intercambio de calor y de posición

En la iteración anterior se concluyó que había que simular el peso del fluido de alguna manera. El aire frío debido a su mayor densidad ocupa menos espacio y tiende a bajar de posición, por lo que se tiene que añadir un intercambio de posición entre dos células, evaluando sus condiciones previamente.

El riesgo que supone hacer esto es elevado. Si una célula se intercambia de posición en pleno cálculo de flujo de energía, este valor no será adecuado puesto que los datos serán erróneos al cambiar de vecindad. Este intercambio de posición se tiene que realizar al final, evitando iterar otra vez por todo el autómata, con el objeto que sea lo más óptimo posible.

Por cada célula, se comprueba la temperatura de la célula inmediatamente superior. Si es mayor, se queda igual, y si es menor intercambian de posición. Se ha declarado una variable auxiliar que lleve este intercambio de posición y posteriormente se aplicará a la matriz tridimensional.

Como resultado, se ha obtenido un autómata dónde el aire frío desciende rápidamente y el aire caliente sube, logrando el flujo de energía deseado.

5.4. Semana 4 - Construcción del entorno

En este punto, el autómata no recibe energía externa de ningún sitio. Pero en nuestra simulación tenemos que aplicar una fuente de energía externa para visualizar el intercambio en un entorno dinámico. Por lo que se va a generar unas capas externas de aire que simulan una atmósfera fuera de la cueva. Aquí necesitaremos definir más entidades de las que hay, que por el momento es solo una. La nueva entidad, que es simulada junto con la entidad del aire de la cueva, es la atmósfera, que será enteramente la parte generativa del autómata. Una vez creada la atmósfera, se le aplica una temperatura a este tipo de células mediante una variable. En la generación, se necesitan unas condiciones para que se generen o se eliminen capas. Así logramos que nuestro autómata sea procedural. Además, por otro lado, hay que crear la otra parte que participa en el entorno: la entidad estática roca. Esta será la tercera entidad del autómata. Es estática poque no se generan o se destruyen células de este tipo. Pero sí que intercambian energía en función de sus propiedades de material sólido. La roca tampoco se intercambia de posición para hacer que baje el aire frío. El intercambio de posición solo sucede entre aire de la cueva y atmósfera, es decir, entre células de aire.

El hecho de añadir otra entidad va a producir que se añadan más parámetros y se tenga un mayor control sobre las células de nuestro autómata. Pero la parte procedural puede generar problemas en la ejecución. Una mala implementación

puede producir sobrecargas de memoria, mal posicionamiento de las células, eliminación no deseada de células en puntos de interés, y otros muchos efectos adversos.

En cuanto a la implementación, se define una nueva variable en las células que indique su rol en el entorno, aire externo, aire interno o roca. Una vez añadida esta variable, se hace uso de ella en el intercambio de energía para evitar que intercambien de posición dos células de roca, por ejemplo. Después, se crea una o varias capas de células en uno de los ejes coordenados para aumentar el tamaño del autómata. Una vez generada una cantidad de capas, se destruye capa por capa en el orden adecuado, manteniendo la consistencia del autómata.

Como resultado, se ha obtenido un autómata que puede crecer y decrecer en función de los nuevos parámetros dados por las nuevas entidades, haciendo que el entorno sea dinámico y produciendo cambios en los distintos fluidos.

5.5. Semana 5 - Construcción de la geometría

En esta iteración se crea la geometría del entorno. Para ello se necesita la topología del terreno, así como la forma interna de la cueva. Lo ideal es aportar como entrada de datos un modelo en tres dimensiones del terreno y que automáticamente se genere el autómata.

Los riesgos que puede producir esta parametrización de la geometría en el resto de funcionalidades ya implementadas no son elevados. Simplemente se hace una clasificación de células antes de iniciar la simulación. Eso sí, es fundamental que la geometría sea convexa para clasificar bien las células que se encuentran dentro o fuera de ella. Para mayor detalle, ver el punto [6.3.1].

Este autómata se crea de forma rectangular, y con un número de células que varía en función del tamaño de cada una. Si las células son más pequeñas, podrán situarse más células dentro del rectángulo, y la geometría se verá más definida. Dentro de este rectángulo se encuentra el modelo de la cueva. Todas las células que se encuentren dentro de este modelo, se clasifican como aire de la cueva, y el resto será roca. A partir de ahí, se generan capas externas al autómata inicial cuyas células corresponden con la atmósfera.

Como resultado, se ha obtenido un autómata con una forma específica de la cueva fácilmente parametrizable, donde se puede apreciar el aire interno de ésta haciendo una incisión en el autómata por el medio.

5.6. Semana 6 – Interfaz

En esta iteración se van a realizar dos interfaces. Una en la primera escena para introducir los parámetros del aire y del autómata. Y otra interfaz en la escena de la simulación tanto informativa como interactiva.

El riesgo que esto supone para el resto de la aplicación es mínimo, puesto que solo se enlazan los distintos parámetros de las dos interfaces. Al estar programado para un entorno dinámico, la variación de estos parámetros en tiempo de ejecución no supone ningún problema.

La primera interfaz contiene los parámetros iniciales del entorno, que son las propiedades de los materiales. La segunda interfaz de la simulación será la más informativa. Aquí sabemos la temperatura de la roca y de la atmósfera, pudiendo modificar ambas variables, activar o no las visualizaciones de las distintas entidades del autómata, definir un rango de temperaturas dónde solo se vean las células con esas propiedades, activar o desactivar la lectura del archivo de datos e interactuar con la distancia que divide el autómata celular en dos.

Como resultado, se ha obtenido una mejora en la usabilidad de la aplicación bastante notoria. Ahora, las interfaces facilitan una mejor información de lo que está ocurriendo en el entorno y una mayor flexibilidad al cambio sin necesidad de reiniciar la aplicación.

5.7. Semana 7 – Ampliación de principios físicos

Hasta ahora, las células intercambian energía y de posición en el caso que la célula de arriba sea de menor temperatura. Pero esto puede ser un problema en función de su geometría. Se generan columnas de aire más frío en la entrada superior de la cueva, y ese aire no produce a penas intercambio con el interior. Por lo que, se ha decidido hacer otro intercambio más de posición.

En este punto, el riesgo de hacer cualquier cambio en el núcleo del sistema de intercambio, puede provocar que no funcionen otros módulos ya implementados y empeorar los resultados de nuestro simulador.

La implementación de este principio se hizo de la siguiente manera: si la célula superior es roca y ésta es aire, selecciona la célula de aire con mayor gradiente (menor temperatura) de la capa superior y se intercambian de posición (siempre que sean dos células de aire). Hay ciertos escenarios en donde esto no cumple con su cometido, que es evitar una columna de aire tan abultada. Por ejemplo, en una cueva con dos bocas de aire en la parte superior, formando una especie de V en su interior, se genera esta columna en el pico de la V. Para evitar esto, se añade una probabilidad en la que algunas veces intercambiará con una célula fría de la fila superior, y otras veces con la célula más fría de sus vecinos.

Una vez implementado este concepto, se puede observar cómo este nuevo principio evita la generación de columnas de aire frío tan bruscas. También ha llegado a la conclusión de esto favorece el efecto de la trampa de aire frío (o de succión). En la boca inferior hace que entre el aire frío más bruscamente cuando es invierno, donde el aire del interior de la cueva es más caliente que el del exterior.

5.8. Semana 8 – Lectura del archivo y construcción de un wireframe

En este simulador, se ha usado un archivo con temperaturas anuales de Jaén, en que cada hora corresponde con una temperatura. Esta parte es esencial para la generación automática de las condiciones externas que se podrían producir en un entorno como Jaén.

El riesgo que esto supone es mínimo para el resto de la aplicación. Únicamente hay que definir bien las condiciones en las que se lee el siguiente parámetro del archivo para evitar cambios demasiado rápidos en el entorno y ver así los distintos efectos que se producen.

Antes de iniciar la simulación, se hace una lectura completa del archivo, y se almacena su información en una EEDD, la cual está formada por un vector donde se almacena un par de variables por posición. Este par está compuesto de una cadena

de caracteres, que conformará la fecha y hora, y un número en coma flotante, que es la temperatura. De esta forma, cuando leemos un dato, no se accede al archivo, sino que están guardados en memoria, pudiendo acceder a ellos mediante un índice que indicará en qué periodo del año se encuentra. Como cada posición del vector corresponde con una hora, tenemos la posibilidad de avanzar una o varias horas/días/semanas/meses sumándole la cantidad correspondiente a este índice. También se ha añadido un wireframe externo para saber cómo de grande es el autómata en cualquier momento. Por ejemplo, si no tenemos la visualización de la parte procedural (atmósfera) activada, no sabremos cuantas capas o células se han generado.

Como conclusión de esta iteración, se ha conseguido mejorar la optimización y ejecutar la simulación con los datos del archivo, pudiendo filtrar las células que eran de interés.

5.9. Semana 9 – Aplicación de velocidad de intercambio

En una previa investigación, se concluyó que otro principio físico importante es la velocidad a la que baja el aire frío en nuestro sistema. La naturaleza tiende al equilibrio, es decir, a que todo esté a la misma temperatura en nuestro simulador. Pero al ser un entorno dinámico, nunca se va a producir esto. Tiende y busca el equilibrio, aunque nunca se llegue. De esta forma, por la ley de Fourier, la roca cambia de temperatura más lentamente que el aire. Aun así, no se produce de la forma deseada. La cantidad de energía que la roca da al aire, es más lenta que la cantidad de energía que el aire se da a otra célula de aire. Es decir, hay una mayor velocidad de intercambio de energía entre células de aire, que entre células de roca y aire. Por lo que se debe de añadir una velocidad de intercambio de posición entre células de aire. De esta forma dejamos que la roca intercambie más energía sobre una célula fría de aire que no intercambie de posición. También, el aire frío tiende a bajar más rápido cuanto menos calor tenga, no baja a la misma velocidad que una célula más caliente.

Probablemente esta sea la iteración más crítica del trabajo. Con todas las tareas ya implementadas, puede ocurrir que dejen de funcionar otras puesto que es un gran cambio del núcleo de nuestro programa. Este principio físico no puede provocar que

dejen de funcionar otras funcionalidades más básicas como, por ejemplo, que el aire menos caliente baje.

Esta velocidad se simula de la siguiente manera. Solo las células de aire pueden intercambiar de posición. Se van a crear dos variables de control que empezarán a la temperatura mínima del autómata y poco a poco una de estas variables va a ir aumentando su valor. Las células cuyas temperaturas estén dentro del rango, se podrán intercambiar de posición si se dan las condiciones. De esta forma, las células más frías bajan primero que las menos frías. Añadiendo una velocidad de bajada del aire frío en el autómata.

Una vez implementada la nueva funcionalidad se procede a evaluar en su conjunto y se detectó un pequeño error en el cálculo del gradiente, previo a la aplicación de la fórmula de Fourier. Este gradiente dependía de la conductividad térmica del material, y cuando se comparaban dos células de distintas propiedades, perdía siempre energía, cuando una tiene que ganar y otra tiene que perder. Esto sucedía cuando una célula de aire y una célula de roca calculaban el gradiente entre ellas. La roca al tener una conductividad térmica de varias magnitudes mayor, el flujo siempre salía negativo. Se decidió dejar el gradiente entre roca y aire como una diferencia de temperatura básica. Una vez solucionado el problema, el simulador funciona correctamente, pudiendo ver que las temperaturas frías bajaban a distinta velocidad en función de su energía.

5.10. Semana 10 – Obtención de resultados

Una vez desarrollada la aplicación al completo, se procede a ejecutar las distintas pruebas y resultados. Como el entorno de trabajo (Unity), no permite grabación de vídeo con sus propias librerías, sino que solo permite sacar capturas de pantalla, se va a crear un pequeño programa en Python que genere un vídeo a través de un conjunto de imágenes. Para ejecutarlo, ver el [*Anexo II*] Estas capturas se almacenan en una carpeta y están nombradas por orden cronológico para que, posteriormente, en el vídeo se ejecuten los frames en orden y ver la evolución de nuestro autómata. Para más detalle, ver el punto [6.8]

6. Explicación del desarrollo

6.1. Creación del autómata celular

En todos los proyectos, y principalmente en este, es fundamental la optimización de la aplicación para que sea lo más eficaz posible. Por ello, ha de definirse una estructura de datos acorde con encontrar la solución más óptima para el problema. En este trabajo se han valorado varias estructuras en árbol puesto que son muy eficientes en las búsquedas, y luego en la parte procedural sería más sencillo añadir o eliminar células en ciertas partes focalizadas de la estructura. Pero hay un problema a la hora de buscar los vecinos de cada elemento. En cada nodo se debe almacenar información suficiente de los vecinos evitando repetir memoria lo máximo posible. Esta repetición de memoria se produce porque las células comparten vecinos, y almacenar por nodo todos sus vecinos acabaría ocupando mucho espacio. A parte de que el árbol no cumpliría su función puesto que pasaría por todos los elementos de EEDD igualmente, como en un vector o una matriz. Y esto, como hemos visto, no es una solución trivial.

Nuestro autómata en un principio va a ser rectangular en un entorno en tres dimensiones. Por lo que la EEDD que mejor representa al problema sería una matriz dinámica y tridimensional. Dinámica porque es procedural, creando y destruyendo filas, variando el tamaño de la matriz. Y tridimensional porque las células van a crearse en los tres ejes de la escena. Así, cada posición de la matriz, corresponde con una célula y el siguiente elemento de la matriz, corresponde con la célula que está al lado de la anterior. Los vecinos de la EEDD y de la escena real, son los mismos.

Tenemos una organización del espacio igual a la representación real del problema. Esto nos va a permitir también la separación entre las células de la escena. Por ejemplo, podemos dividir el bloque del autómata en dos para ver lo que ocurre en su interior. Las células de la frontera están separadas por una distancia definida por un parámetro. Pero en la EEDD van a seguir siendo vecinas puesto que se encuentran una al lado de otra y seguirán intercambiando información entre ellas, a pesar de la separación física que existe en la representación de la escena.

6.2. Sistema de intercambio de energía y parámetros

El siguiente paso es definir los siguientes parámetros que necesitaremos para definir unas condiciones climatológicas:

- Temperatura de la roca
- Temperatura externa

A su vez, también necesitaremos las condiciones para que nuestras células se comporten de una forma u otra. Estos parámetros vienen determinados por la fórmula de transmisión de energía de Fourier [*79*, 2009]:

- Factor de conductividad térmica del material (k)
- Calor específico del material (C_p)
- Densidad del material (ρ)

Con estos 3 parámetros podemos obtener el flujo de energía que hay entre partículas, siendo en la roca más lento que en el aire.

El aire al reducir su temperatura aumenta de peso (su densidad aumenta), por lo que tenderá a bajar. Esto hay que tenerlo en cuenta y es de vital importancia puesto que da una dinámica que solo con la fórmula no se consigue. Por ello hemos aplicado un intercambio de posición entre células de aire.

Los criterios a seguir para el intercambio de energía son:

- Mirará a todos sus vecinos. Al ser un entorno tridimensional habrá veintiséis vecinos. En cada una de ellas se comprobará su temperatura y realizará el cálculo del gradiente correspondiente. Luego se le aplicará la ley de Fourier para ambas células y continuará con la siguiente célula.
- Cuando el vecino se corresponda con la célula de arriba, se comprobará si su temperatura es menor que la célula central (con la que estamos iterando en sus vecinos) y si es aire. Si es menor, se intercambiarán de posición, haciendo que el aire frío baje.
- Cuando el vecino se corresponda con la célula de arriba también, se comprobará si es roca. Si esa célula es roca, aleatoriamente buscará en

los vecinos de arriba o en todos sus vecinos una célula con el menor gradiente que la célula central. Posteriormente se intercambiará de posición con esta célula de menor gradiente. Este efecto va a permitir un poco de succión y de dinamismo dentro de la cueva.

```
procedimiento FlujoEnergia (automata: array[numX][numY][numZ] de células)
      velocidad_intercambio++
      Por cada célula (i, j, k):
            Visitar vecinos (m, n, t):
                  Si automata[i][j+1][k] < automata[i][j][k], son aire y < velocidad_intercambio:
                        intercambiaPos(automata[i][j+1][k], automata[i][j][k])
                  Si automata[i][j+1][k] es roca:
                        Intercambia con aire más frío de alrededor
            Intercambio Fourier(automata[i][j][k], automata[i+m][j+n][k+t])
            finVisitaVecinos
      finIteración
      actualiza visualización(Tmax, Tmin)
fin procedimiento
```

6.3. Geometría

Ahora vamos a darle una forma geométrica, de forma que se pueda percibir la cueva y el aire de esta. Por lo que he definido 3 entidades distintas:

- Aire de la cueva
- Roca
- Aire de la atmósfera o aire saliente de la cueva

Esto nos va a permitir distintos comportamientos del autómata. De forma que, la roca tendrá sus propiedades y el aire otra. La división de dos entidades distintas del aire (cueva y atmósfera) se debe a que la atmósfera va a recibir el aire directamente del parámetro de la temperatura exterior. Esta atmósfera podría ser nula, es decir, puede que no tengamos ninguna célula de esta entidad puesto que corresponde con

la parte de la generación procedural del autómata. Sus propiedades físicas son las mismas puesto que las dos entidades siguen siendo aire.

En cuanto a la forma geométrica de la cueva, he definido una geometría como "**Roca**" que contiene a la geometría definida como "**Aire**". Así, las células que se crean estarán dentro de la geometría envolvente (la roca), y se distinguirá entre ellas si es roca o aire si está dentro o no de la geometría "**Aire**", tal y como se ve en la *Ilustración 21*.

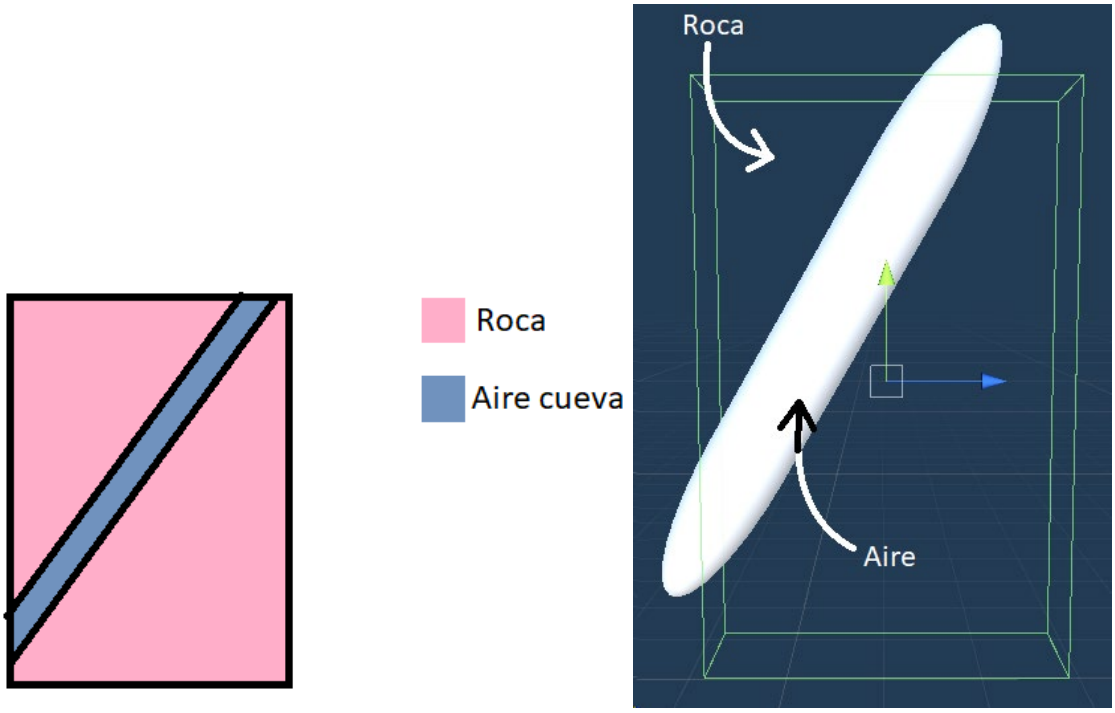

Ilustración 21. Representación de la cueva en el entorno de trabajo de Unity. Elaboración propia.

Y como al visualizarlas se ve un bloque sólido, se ha hecho una partición por la mitad para poder visualizar la cueva por dentro. Tal y como se ve en la *Ilustración 22*, esta separación viene determinada por un parámetro y puede cambiarse en tiempo de ejecución en la interfaz.

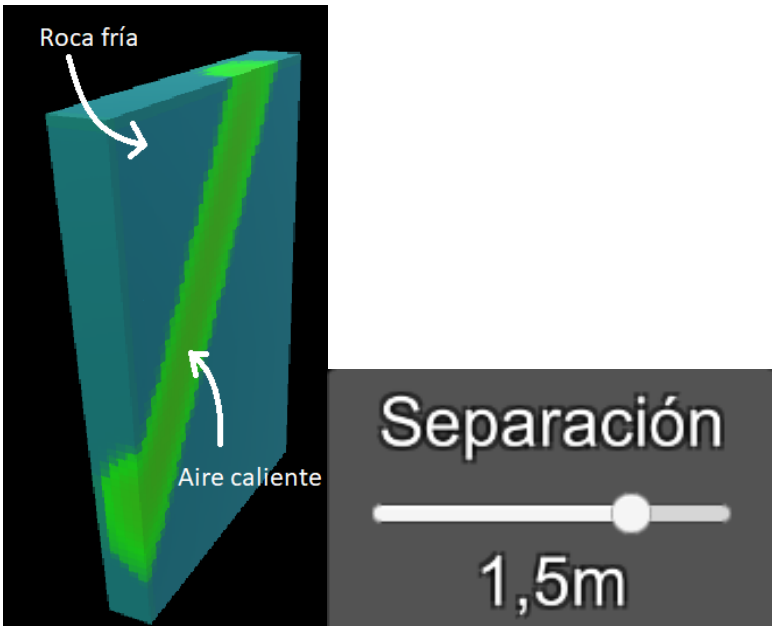

Ilustración 22. Representación de la separación del autómata. Elaboración propia.

Para generar las células dentro de la geometría, se va a necesitar el tamaño que va a abarcar la célula. De esta forma, si el tamaño es menor, habrá un mayor número de células y viceversa. Sabiendo el tamaño de la geometría de la roca y el de la célula, se puede obtener la cantidad de células que hay en los ejes XYZ, que serán las dimensiones iniciales del autómata.

Por ejemplo, en la *Ilustración 23*, para el tamaño de la célula de 1.5 y el de la roca de 25, en un eje habría 16 células.

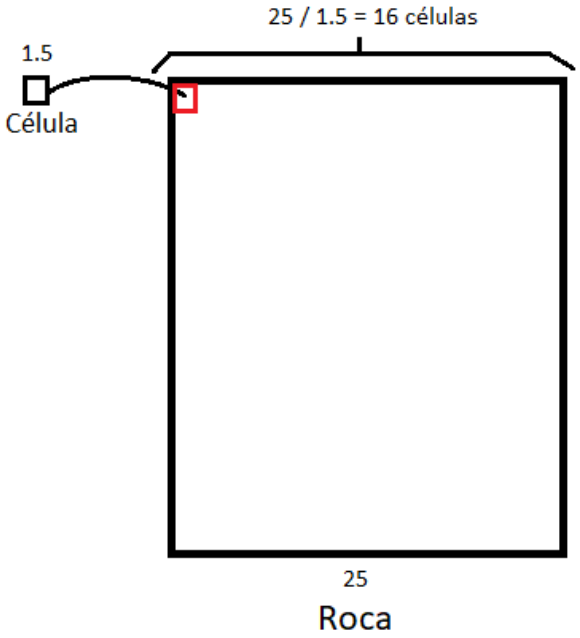

Ilustración 23. Representación esquemática del posicionamiento de las células. Elaboración propia.

Se hace la misma operación para los otros dos ejes. Una vez tengamos el número de células por eje, multiplicamos los 3 números y daría el número total de células del autómata inicialmente:

$$N_{cx} \cdot N_{cy} \cdot N_{cz} = Número\ total\ células$$

Una vez tengamos las dimensiones, hay que colocarlas en la escena una detrás de otra. Para ello es necesario saber el tamaño de las células y el de la roca. Haremos una iteración por el número de células de cada fila/columna para crear el objeto e iremos colocándolas según el punto del bucle en el que se encuentren.

```
Vector3 pos = new Vector3((parteRoca.transform.position.x - escaladoRoca.x) + (i * escaladoCelula.x) + escaladoCelula.x,
                          (parteRoca.transform.position.y - escaladoRoca.y) + (j * escaladoCelula.y) + escaladoCelula.y,
                          (parteRoca.transform.position.z - escaladoRoca.z) + (k * escaladoCelula.z) + escaladoCelula.z);
```

Ilustración 24. Cálculo del posicionamiento de células. Elaboración propia

Ilustración 25. Representación del posicionamiento de las células en el entorno. Elaboración propia.

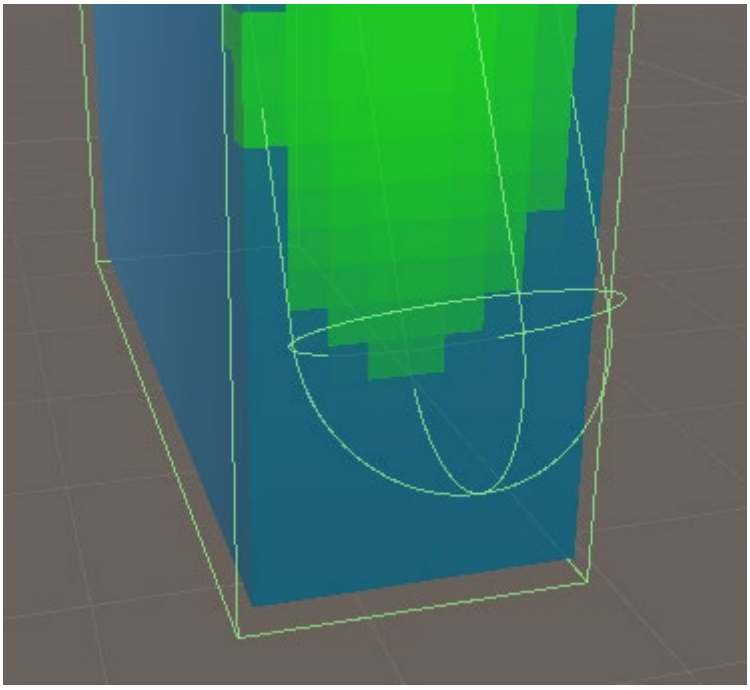

Ilustración 26. Representación del posicionamiento de las células en el entorno. Elaboración propia

Una vez colocadas las células, hay que saber cuáles de ellas son aire de la cueva. Esto vendrá determinado por la otra geometría, la del aire. Tendremos que saber sus dimensiones y su posición en la escena, para saber qué parte está dentro de la geometría de la roca. En cada punto, se comprobará si esa célula está dentro de la geometría del aire o no. Para ello, la posición se transformará en el centro de la geometría y se trazará un Raycasting. Si no colisiona con nada es que está dentro, y

si colisiona, es que está fuera (el Raycasting de dentro de una geometría hacia afuera no colisiona).

```
Collider malla = obj[i].GetComponent<Collider>();
Vector3 offset = malla.bounds.center - pos;
Ray input = new Ray(pos, offset.normalized);
RaycastHit rHit;

if (!malla.Raycast(input, out rHit, offset.magnitude * 1.01f))
{
    return true;
}
```

Ilustración 27. Cálculo de las células que están dentro y las que no. Elaboración propia.

Hay que obtener el Collider [80], puesto que necesitamos la variable *bounds* de esta clase. Esto nos dará los límites del objeto para saber si tiene rotación.

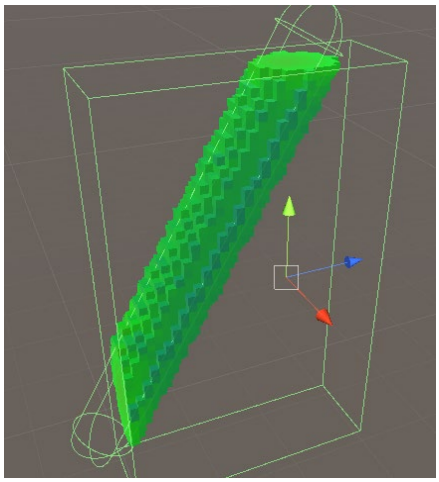

Ilustración 28. Representación de las células que corresponden con el aire de la cueva. Elaboración propia.

Así tendremos la opción de añadirle cualquier geometría de aire al objeto con distintos colliders. Se podría escanear una cueva y sacar un modelo en tres dimensiones de esta para pasarla como parámetro a la aplicación.

6.3.1. Problema de la geometría

Para que esto funcione, el collider de la geometría tiene que ser **convexo**, puesto que se cogen los límites de ese collider. Si fuese una geometría cóncava, habría

puntos externos que se clasificarían como internos puesto que se utiliza un raycast para obtener la ubicación de la célula dentro de la geometría. Si queremos añadir una geometría no convexa, tendremos que dividirla a su vez en partes convexas.

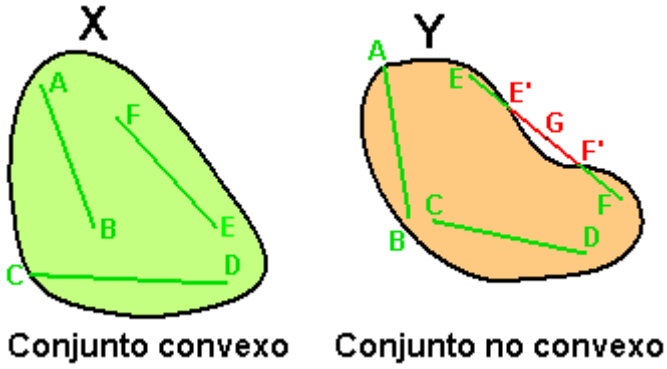

Ilustración 29. Ejemplo de geometría convexa y no convexa. Elaboración propia.

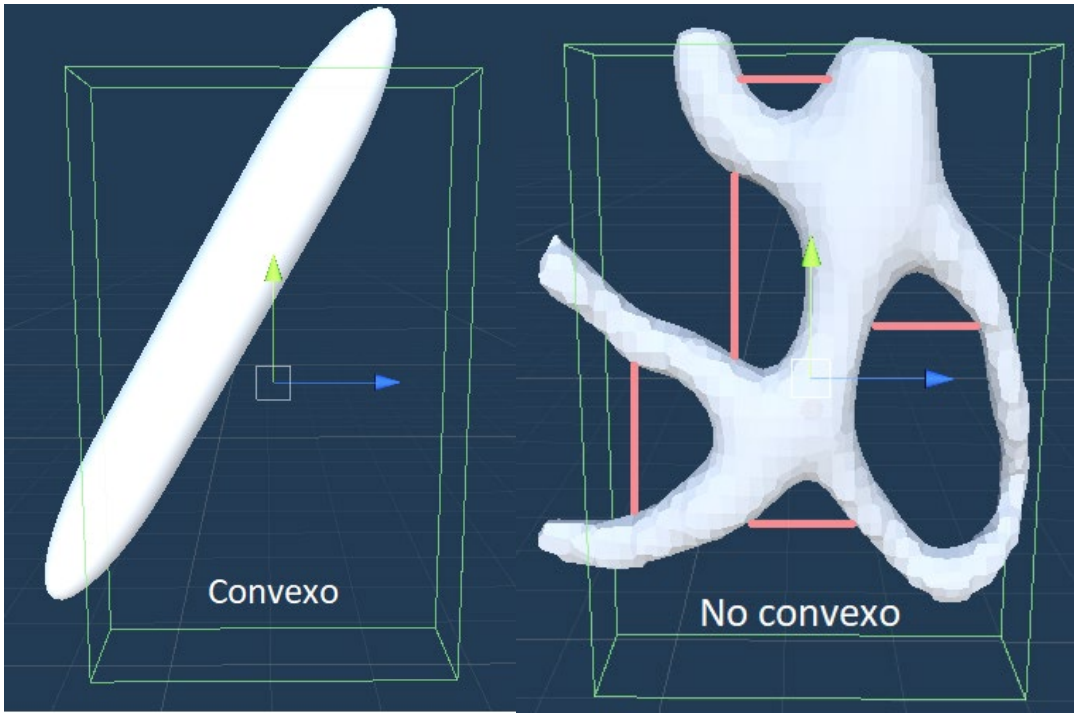

Ilustración 30. Representación de figuras aceptadas y rechazadas por el simulador. Elaboración propia.

En la geometría de la derecha de la *Ilustración* 30 no funciona el cálculo puesto que no es convexa, a esto se le denomina figura **cóncava**. Si queremos que funcione, hay que dividirla en distintos collider **convexos** e ir comprobando uno por uno si el punto se encuentra dentro, tal y como se muestra en la *Ilustración 31*.

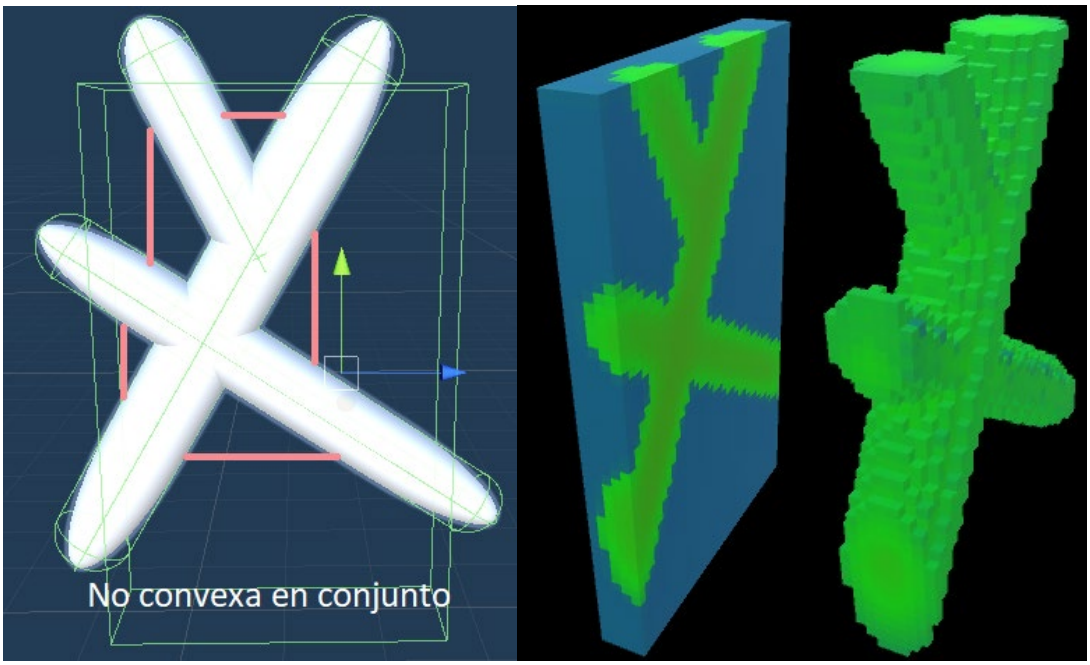

Ilustración 31. Representación de una forma de la cueva compleja. Elaboración propia.

Esta figura no es convexa en su conjunto, pero dividiéndolo en figuras convexas, los cálculos se realizan de forma correcta. Por lo que, si obtenemos un modelo más complejo y queremos dividirlo en figuras convexas, estaría bien aplicar algún algoritmo para que subdivida esta geometría compleja y no apta, en figuras o polígonos aptos para el problema.

Al tener las entidades bien definidas y diferenciadas, podemos controlar la visualización. Si no nos interesa ver la evolución de la roca se puede desactivar su visualización, como se aprecia en la *Ilustración 32*. Lo mismo para el aire de la cueva y el aire de la atmósfera.

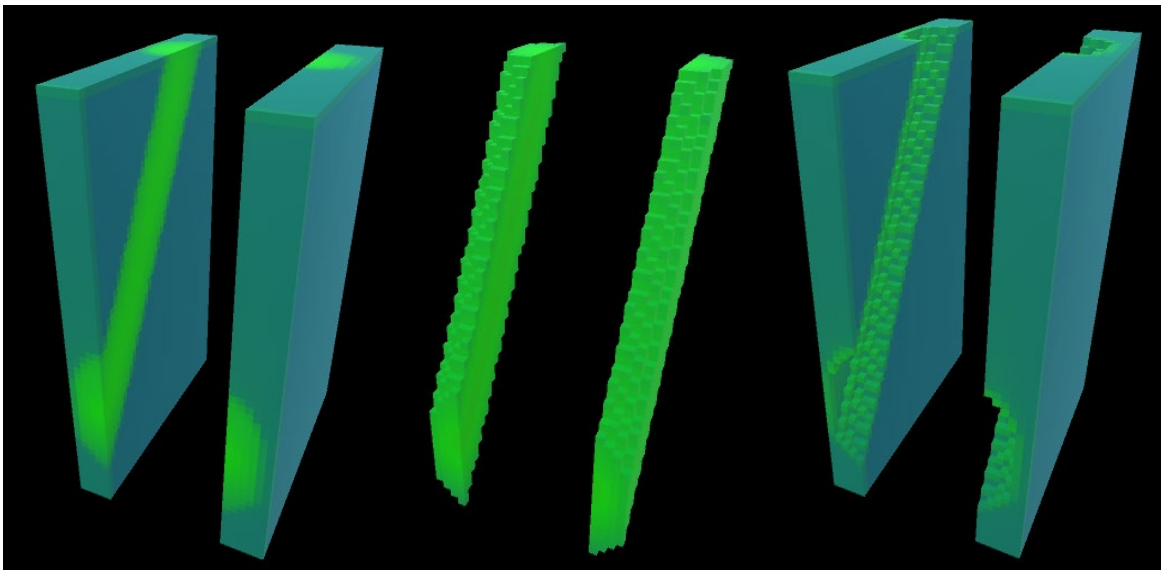

Ilustración 32. Representación del entorno visualizando distintas partes que lo componen. Elaboración propia.

También podemos variar la distancia de las dos partes, como se representa en la *Ilustración 33*, para ver mejor la forma de la cueva.

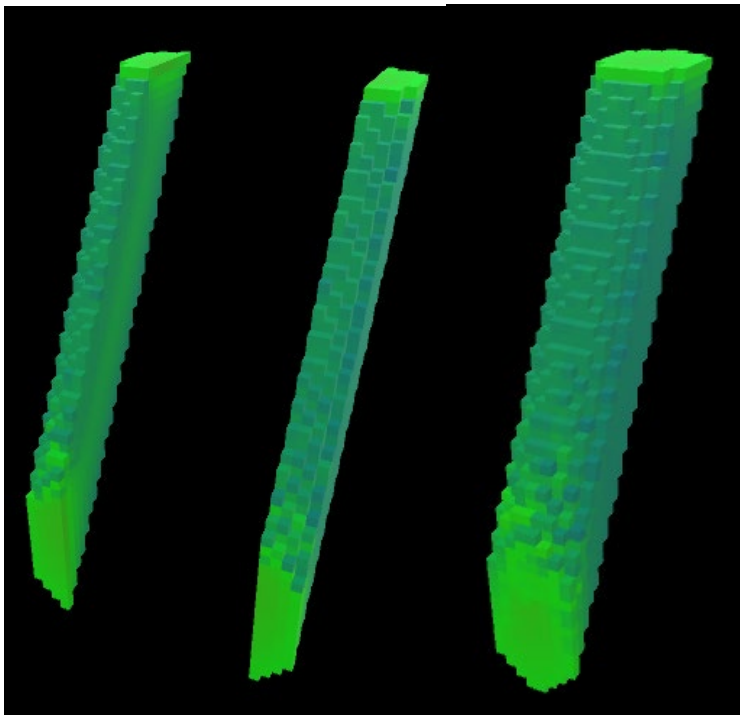

Ilustración 33. Representación del aire de la cueva con separación variable. Elaboración propia.

6.4. Generación de capas del autómata

Al ser un autómata procedural, se crearán o se destruirán capas en función de la temperatura de las últimas capas con la temperatura ambiental. Si la última capa de células tiene una temperatura media igual que la temperatura ambiente, se considera un autómata estable. Pero si no es igual a la temperatura ambiente, el autómata crece añadiendo una capa más. Y en el caso contrario, si la última y la penúltima capa, son estables, se eliminará ésta última para que decrezca el autómata.

Como se puede apreciar en la *Ilustración 34*, un autómata cuyas células tengan la misma temperatura decrece debido a que se considera estable y estas capas no aportan información alguna.

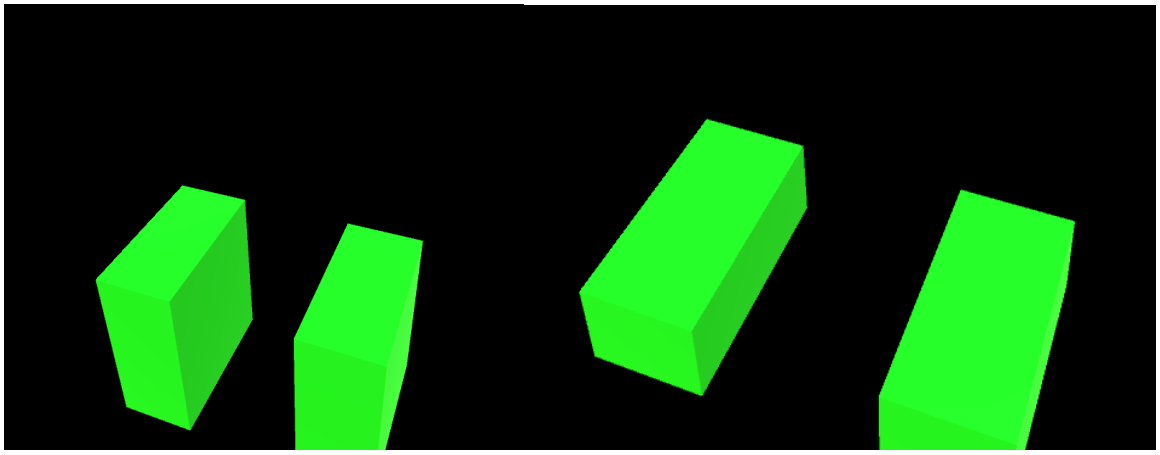

Ilustración 34. Representación de la generación por capas del autómata. Elaboración propia.

Por otro lado, un autómata dónde las capas no tienen la misma temperatura y no se asimilen a la temperatura ambiente, tenderá a crecer hasta que se considere estable, como se aprecia en la *Ilustración 35*.

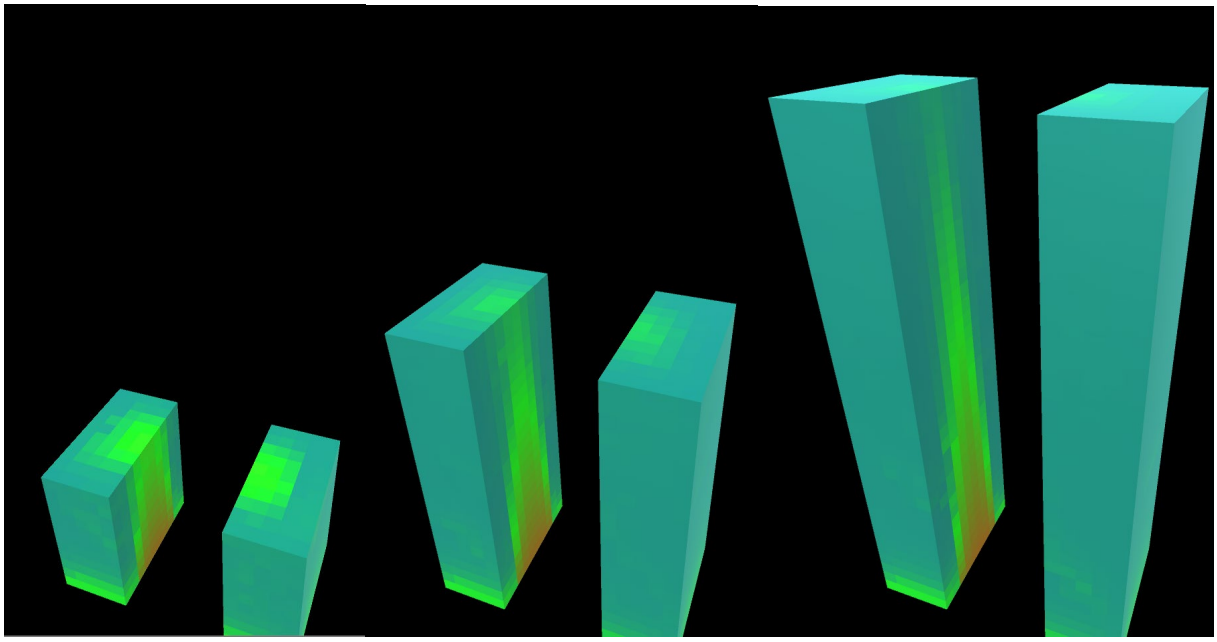

Ilustración 35. Representación de un sistema en crecimiento. Elaboración propia.

Esta estabilidad puede parametrizarse. De forma que la media te salga más o menos cercana a la temperatura ambiente y que no crezca un número de filas mayor, que implicaría más células y, por lo tanto, más tiempo de cómputo. También se pueden acotar los valores de células visibles en función de un rango de temperaturas. Para ello, se dibuja una especie de *Wireframe* en el límite del sistema para ver su crecimiento. Este *Wireframe* de la *Ilustración 36* posteriormente se sustituirá por uno que envuelva a la cueva y ver una representación más fiel del problema, como el de la *Ilustración 42*.

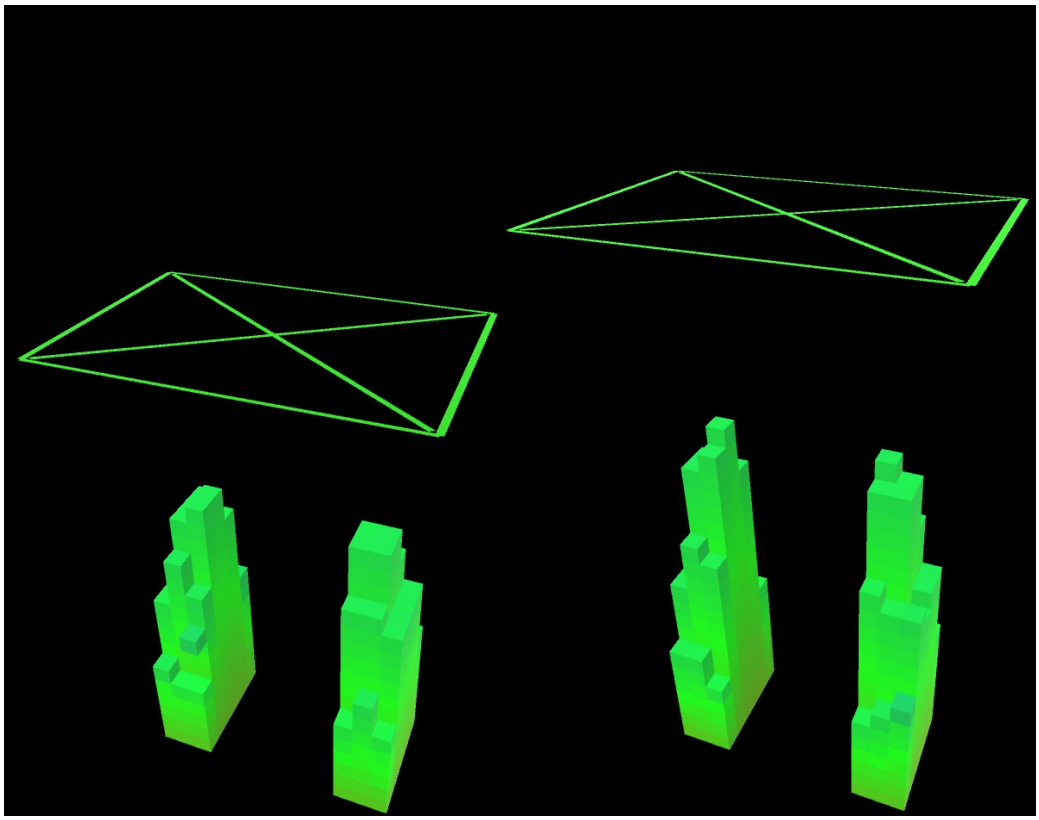

Ilustración 36. Representación de un sistema en crecimiento con un wireframe. Elaboración propia.

Esta vista acotada de la *Ilustración 37* puede darnos una mejor visibilidad de las células interiores y la forma en la que sube la energía por nuestro sistema.

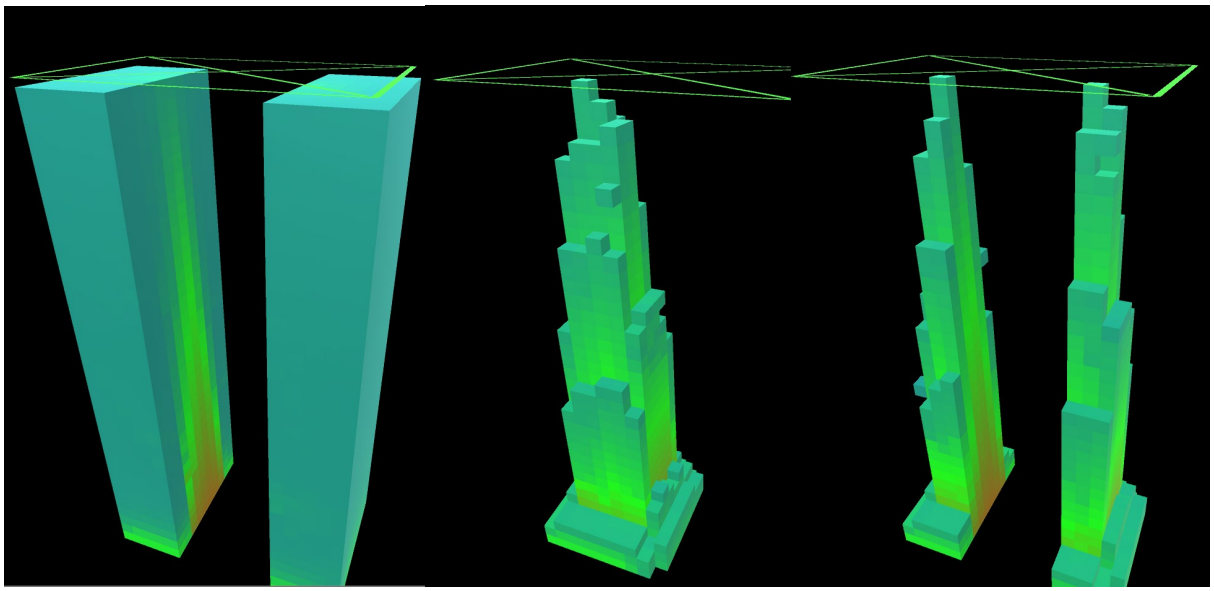

Ilustración 37. Representación de las distintas visualizaciones por rango de temperatura. Elaboración propia.

Pero para nuestro problema, necesitamos una generación de capas por ejes. Habrá que comprobar si las caras de nuestro rectángulo inicial contienen aire de la cueva. Esto servirá para saber si hay una salida de la cueva o no. Solo se generan capas en las caras donde haya salidas, puesto que es la zona que nos interesa visualizar en cuanto a la parte generativa. No es lo mismo generar una capa en una de las caras laterales que en la cara superior. Nuestra EEDD, la matriz dinámica, debe de crecer en un eje o en otro si se dan las condiciones. Se comprueban las últimas capas de las caras para saber su temperatura media y saber hasta dónde crecer o decrecer. Esta comprobación se realiza cada x tiempo debido al alto coste computacional que se necesita.

En un sistema procedural hay partes que se generan y se destruyen. En nuestro caso nos interesa las partes salientes de la cueva para ver el flujo del aire. La EEDD utilizada inicialmente será una matriz tridimensional. Por lo que vamos a tener varios casos de generación:

- Generación en el eje X de la matriz y del autómata
- Generación en el eje Y de la matriz y del autómata
- Generación en el eje Z de la matriz y del autómata

En función de dónde se genere la capa, hay que copiar los elementos de una forma u otra:

- Si la capa se genera al principio de este eje, hay que añadirle una fila al principio de la matriz:
    - Nuevos elementos en la posición 0
    - A partir de la posición 0, copiamos el resto de elementos
- Si la capa se genera al final de este eje, hay que añadirle la capa al final de la matriz:
    - Copiar todos los elementos desde la posición 0 hacia delante
    - Nuevos elementos en la última capa que estará vacía

Esta funcionalidad será la misma para todos los ejes.

¿Cómo se tienen que generar las células?

Pues bien, para generarlas en la posición correcta de la escena, ya no depende de la geometría de la roca, solo depende del tamaño de ella misma (el tamaño que vaya a ocupar). El número de células que se generan es el mismo que al principio, lo cual corresponde con el tamaño de la matriz.

- Si se generan al principio del autómata, se obtendrá el valor de la posición de la primera capa de ese eje que haya en el autómata y se le restará el tamaño de la célula.
- Si se generan al final del autómata, se obtendrá el valor de la posición de la última capa de ese eje que haya en el autómata y se le sumará el tamaño de la célula.

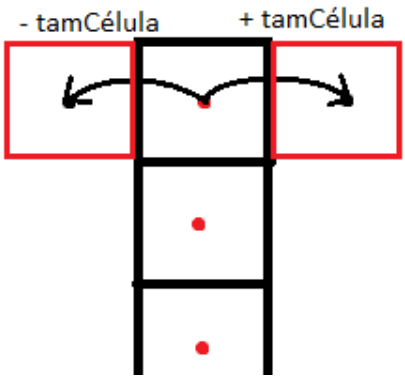

Ilustración 38. Representación esquemática de la generación de capas. Elaboración propia.

De esta forma mantendremos siempre el centro del autómata y se generarán hacia fuera.

Las capas que se generan forman parte de la entidad de **aire de la atmósfera**. Esto nos va a permitir distinguirlas y saber a qué elementos le aplicamos la energía del entorno.

¿Y qué condición se utiliza para generar la capa?

Si la primera o la última capa del eje contiene parte de roca y parte de aire, se genera una capa instantáneamente. Esto quiere decir que hay una salida de la cueva por la que saldrá el aire.

Si la capa es completamente roca, no se generan capas en ese eje. Esto quiere decir que no contiene ninguna salida de aire. Y no solo que sea aire, sino que también sea aire de cueva. Porque recordemos que tenemos tres entidades: la roca, el aire de la cueva y el aire de la atmósfera. Imaginemos que se genera una capa en la parte superior del autómata. Cuando se comprueben en las capas externas de las caras si hay aire (de cualquier tipo), saldrá que genere una capa externa puesto que la nueva capa generada en la parte superior es de aire. Por ende, hay que comprobar si en una capa hay células de aire de la cueva, y no de la atmósfera.

Pero esto da problemas posteriormente. Una vez se genere una capa, en la siguiente comprobación, toda esta última capa es de aire de la atmósfera, y por el criterio anterior no se generaría ninguna puesto que no existe aire de la cueva. Para resolver este problema, se utilizará una variable de control por si en una cara se deben o no generar capas. Al inicio se hará una comprobación de la geometría, y cómo ésta es estática, ya se saben las caras con salidas de aire. Esto también sirve para optimizar la aplicación, ahorrándonos comprobaciones y bucles innecesarios.

Si la capa contiene aire de atmósfera en su totalidad, quiere decir que no es la primera capa que se genera. Por lo que se comprueba la siguiente condición:

- Si la temperatura media de la capa es igual que la temperatura ambiente, no se generan más capas. Se considera que ha llegado a un equilibrio térmico.
- Si la temperatura media de la capa es distinta a la temperatura ambiente, se genera una capa en ese eje. Se considera que no ha llegado a un equilibrio térmico todavía.

```
Procedimiento GenerarCapas (automata: array[numX][numY][numZ] de células)
   comprueba si hay boca de aire en la capa +X:
         comprueba si no es una capa estable:
               generaCapa(+X)
   comprueba si hay boca de aire en la capa -X:
         comprueba si no es una capa estable:
               generaCapa(-X)
   comprueba si hay boca de aire en la capa +Y:
         comprueba si no es una capa estable:
               generaCapa(+Y)
   comprueba si hay boca de aire en la capa +Z:
         comprueba si no es una capa estable:
               generaCapa(+Z)
   comprueba si hay boca de aire en la capa -Z:
         comprueba si no es una capa estable:
               generaCapa(-Z)
   reset velocidad_intercambio
fin procedimiento
```

Ahora se va a crear una malla o wireframe que envuelva al autómata para ver hasta dónde crece. El principal problema es el crecimiento de este. Para ello

utilizaremos un GameObject [80] que cambiará su escalado en función de la generación del autómata. Este escalado tiene que hacerse por ejes y mover el objeto para mantenerse en el centro.

Por ejemplo, si se genera una capa más, hay que escalar y mover el wireframe en la dirección esperada. Sabemos el tamaño de la célula, por lo que sabemos el escalado que hay que aplicarle al wireframe. Como al generar la capa el autómata mantiene su centro estático para que se vean traslaciones no deseadas de éste, el wireframe también debe de mantener el mismo centro que el autómata. El escalado se hace en ambos sentidos de una misma dirección o eje. Por lo que, si se genera en uno de los lados, habrá que escalar sumándole el tamaño de la célula y trasladando el wireframe. En nuestro caso, no podemos trasladarlo al centro del autómata teniendo de referencia la roca puesto que hay una separación que el wireframe debe de cumplir.

En la *Ilustración 39* se aprecia que no es fácil de calcular las nuevas dimensiones del wireframe. El problema viene cuando separamos el autómata para poder verlo por dentro. Esa separación va a hacer que las células se salgan del wireframe y que éste no cumpla su función principal. El wireframe se deberá de ajustar a esta separación:

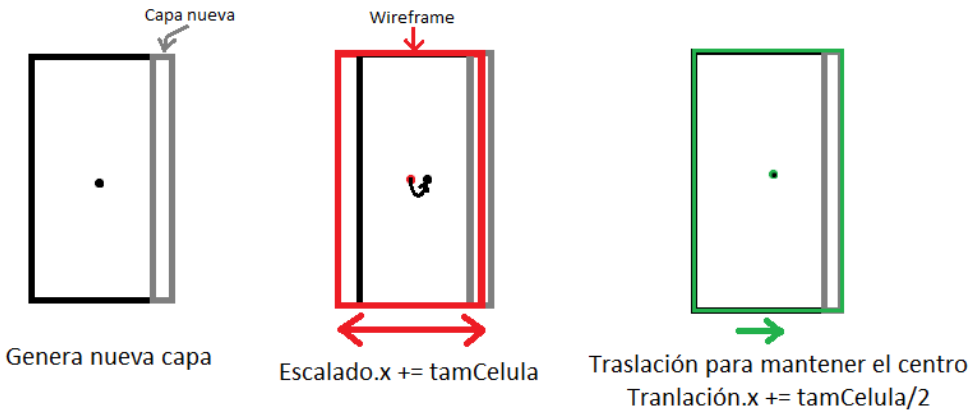

Ilustración 39. Representación esquemática del ajuste del wireframe envolvente. Elaboración propia.

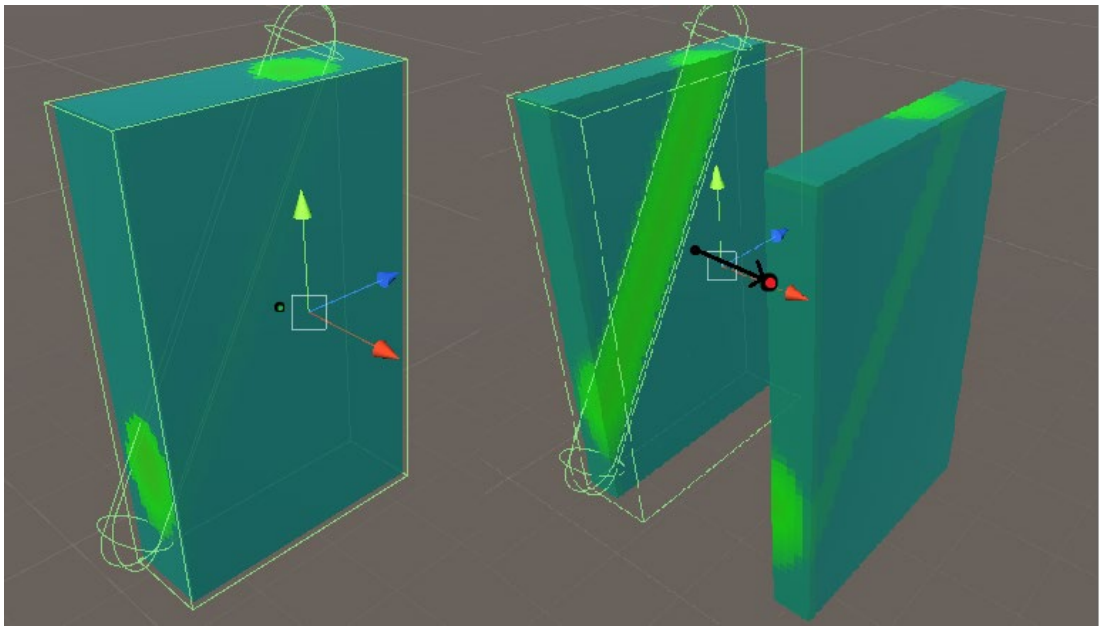

Ilustración 40. Representación del wireframe no ajustable a la separación. Elaboración propia.

Por lo que el cálculo del escalado y la traslación que sufre es distinta. Ahora hay que tener siempre en cuenta esto. Cada vez que genere una capa, el *wireframe* se verá afectado por los cálculos vistos anteriormente. Pero cuando varíe la separación hay que hacer algunos ajustes. En nuestro caso, solo se va a dar la separación en el **eje X**, por lo que estas transformaciones van a darse en este eje, como vemos en la *Ilustración 41*.

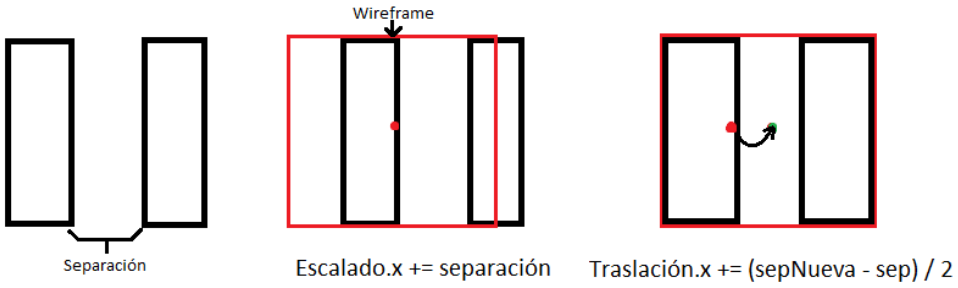

Ilustración 41. Representación esquemática del ajuste del wireframe a la separación. Elaboración propia.

En el escalado no hay ningún problema, pero en la traslación sí. Necesitamos saber la separación que había anteriormente por si crece o decrece el tamaño del

wireframe y saber el sentido de la separación (si se separa más o menos). De esta forma, conforme aumente o disminuya su separación, se irá ajustando el *wireframe*.

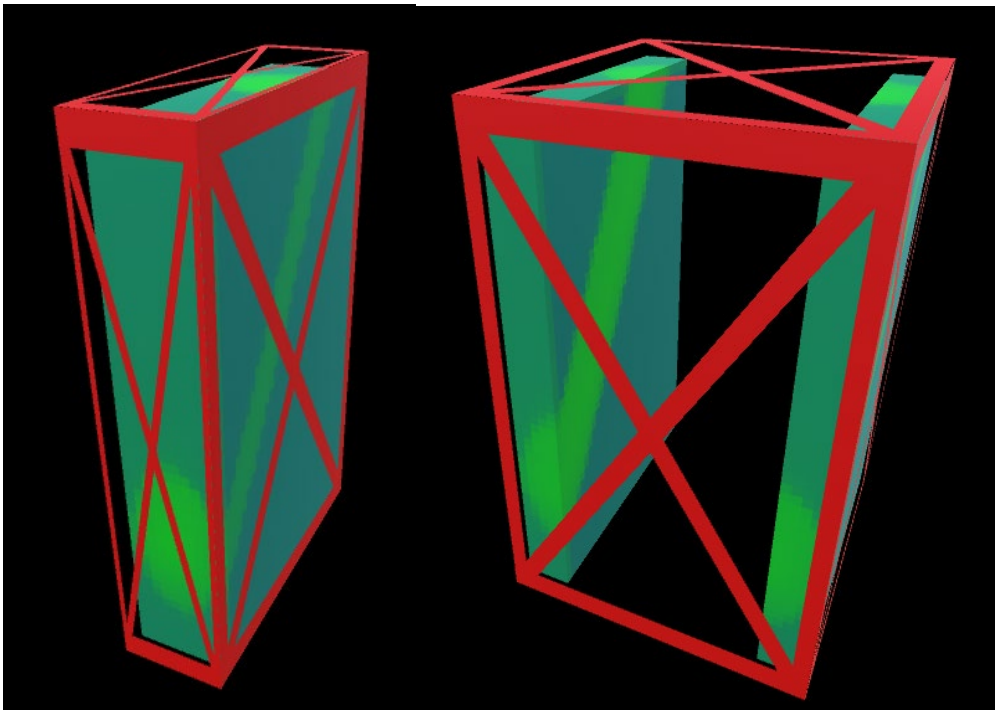

Ilustración 42. Representación del wireframe bien ajustado. Elaboración propia.

6.5. Interfaz

En esta simulación hay un total de 2 interfaces, una en el menú de inicio y otra en la visualización del simulador. De esta forma, vamos a poder darle valores a los parámetros iniciales e ir viendo la información del entorno conforme evolucione la simulación. También podremos elegir qué parte visualizar o activar/desactivar los distintos elementos de la aplicación.

En la *Ilustración 43* se puede ver el menú de inicio que se compone de varios elementos:

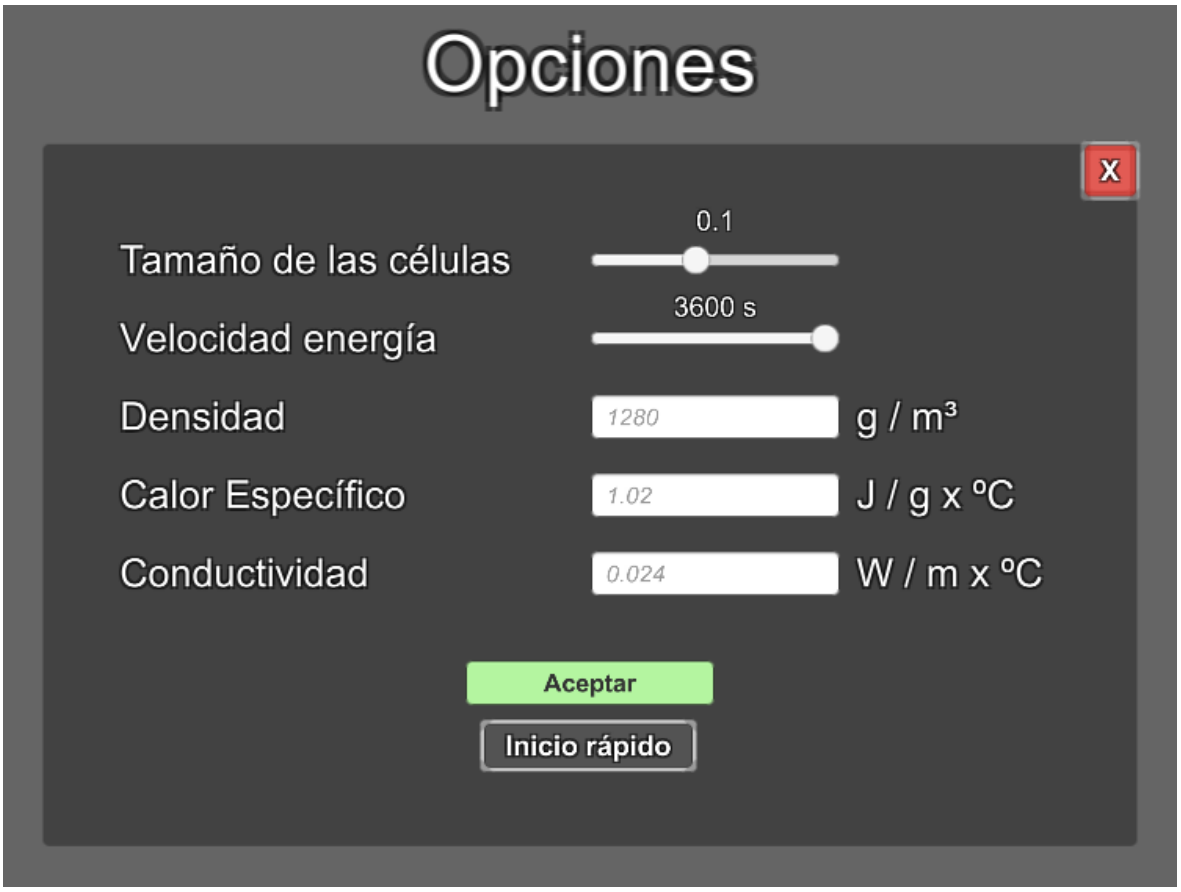

Ilustración 43. Interfaz del menú de inicio. Elaboración propia.

- Tamaño de las células: En función de este valor se van a generar más o menos células, en función del tamaño de la roca y de cuantas caben dentro. Gracias a este tamaño, podremos estimar un cálculo aproximado de cuántas células tendremos que procesar en el instante inicial. Para ellos también necesitaremos las dimensiones de la roca, que será donde se vayan a generar. Por eso este valor no sale hasta que no se inicie el programa.

Ilustración 44. Interfaz con cálculo de células a tiempo real. Elaboración propia.

- Velocidad de energía: Este es el parámetro de la velocidad a la que se produce el flujo de energía. Cuanta mayor velocidad, más flujo habrá y más rápido cambiarán de temperatura.
- Densidad: La densidad del aire de la cueva medido en g/m³

- Calor específico: El calor específico del aire de la cueva medido en J/(g·ºC)
- Conductividad: La conductividad térmica del aire de la cueva medido en W/(m·ºC)
- Botón de Aceptar: En principio está desactivado hasta que no se rellenen todos los parámetros de texto. Una vez se activa y se pulsa, comienza la escena con los parámetros dados. [80]
- Botón de Inicio rápido: Los parámetros tienen unos valores por defecto. Si no se quiere introducir los datos, se iniciará la escena con los valores que hay en cada casilla cuando no está escrita. [80]

Como se aprecia en la *Ilustración 45*, el menú del simulador se compone de los siguientes elementos:

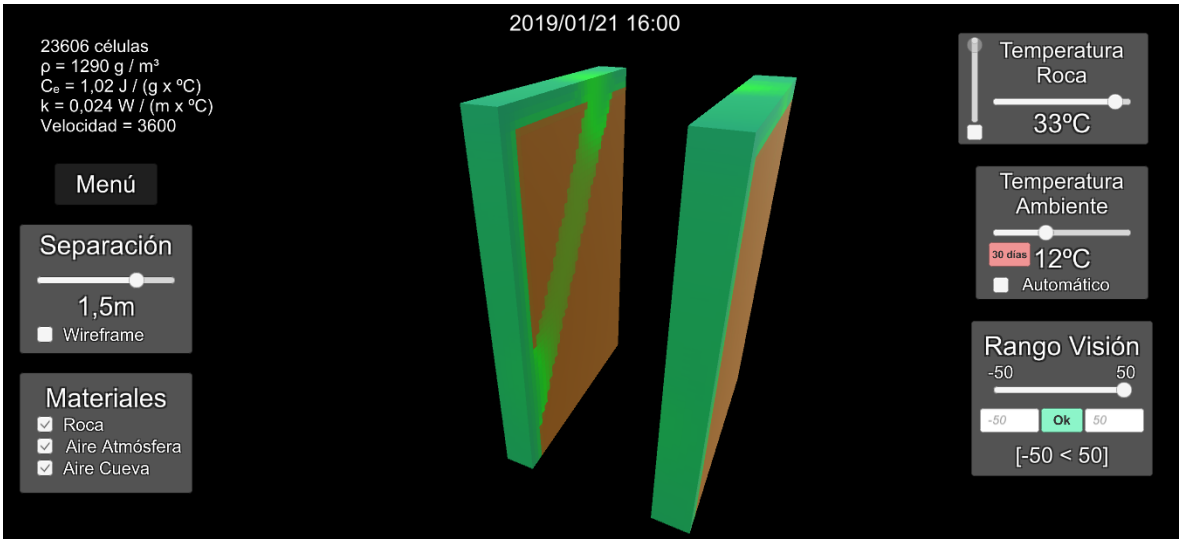

Ilustración 45. Representación de la escena con su interfaz. Elaboración propia.

De arriba a la izquierda hacia abajo a la derecha:

- Información sobre los datos actuales de la escena, indicando el número de células que hay procesándose, la densidad del aire, el calor específico, el factor de conductividad térmica y la velocidad del flujo.
- Botón Menú: Accede al menú inicial del simulador. [80]
- Panel de Separación: Está compuesto por un slider [79] que indica la separación que hay entre las dos partes. Debajo del slider hay un texto que se actualiza en función de la separación que haya entre las partes. El rango del

slider va entre 0 y 2. Y hay una casilla para activar o desactivar el wireframe de la escena.

- Panel de Materiales: Se compone de 3 casillas que indican las distintas partes/entidades de la escena. Podemos activarlas o desactivarlas en función de las partes que queramos ver del autómata.
- Texto Fecha: La fecha por la que va el archivo en formato AAAA/MM/DD hh:mm
- Panel de Temperatura Roca: Compuesto por un slider cuyo rango viene definido por la temperatura mínima y máxima del archivo de temperaturas. El texto informativo de la temperatura no solo nos dice el valor del slider, sino también el valor de este parámetro si cambia en tiempo de ejecución por algún motivo. Este slider podemos manejarlo nosotros mismos o hacer que varíe automáticamente en función a la temperatura del entorno con una velocidad indicada en los elementos de la izquierda. Estos elementos son un slider, que indica la velocidad, y una casilla que activa o desactiva este efecto.
- Panel de Temperatura Ambiente: Está compuesto por un slider cuyo rango viene definido por la temperatura mínima y máxima del archivo de temperaturas. También tiene un texto informativo de la temperatura ambiental, por ejemplo, para ver qué temperatura pilla en cada hora del día cuando se lea el archivo. Hay un botón rojo [80] para pasar 30 días por el archivo de temperaturas por si queremos ir rápidamente de verano a invierno o viceversa. También hay una casilla para activar o desactivar la lectura del archivo.
- Panel de Rango Visión: Está compuesto por un slider y dos casillas de texto. Estas dos casillas de texto son numéricas e indican el rango de temperaturas que se quiere visualizar del autómata. También hay un botón para aplicar estas temperaturas. Se puede meter un valor mayor o menor en cualquiera de las dos casillas, el programa calcula cuál de ellos es el mayor de los dos valores dados. Si uno de los dos valores no está completos, el botón no hace nada. [80]

6.6. Lectura del archivo

Si queremos leer los datos directamente de un fichero, tendremos que ajustar las temperaturas mínimas y máximas que se van a alcanzar para poder definir una gama de colores adecuada.

Nuestro archivo consta de una columna con la fecha y otra con la temperatura. Cada fila corresponde con la lectura de una temperatura en **una hora**. Por lo que la transmisión de energía que se hace entre células por cada lectura, es en el periodo de 1 hora. Esto hay que tenerlo en cuenta para calcular el flujo de energía en cada frame o pasada.

Una vez conocemos la temperatura mínima y máxima, podemos definir el rango de colores para la representación del flujo de temperatura del autómata:

- El valor que buscamos está entre 0 y 1 para los 3 canales de color RGB.
- Calculamos el valor absoluto de la temperatura mínima y lo sumamos a las dos temperaturas. Esto se debe a que cabe la posibilidad de que la temperatura mínima sea negativa, y al hacer los cálculos salga valores negativos para el color, por lo que no estaría entre 0 y 1. Por lo que, al calcular el color, hay que sumarle a la temperatura de la célula, este valor absoluto también. De esta forma nos aseguramos de que el rango de cálculo sea siempre positivo y no de valores negativos en los canales.
- Hay que calcular el valor intermedio de las temperaturas, es decir, si la temperatura mínima es 3ºC y la máxima 37ºC, la mitad serían 17ºC ((37 - 3) / 2).
- Una vez sepamos el pivote, comprobamos si la temperatura de la célula es mayor o menor que este pivote, por lo que tendremos dos mitades:
    - Si es menor, el canal rojo se mantiene a 0 (colores azules y verdes). El canal verde tiene que ir aumentando hasta 1, que será cuando la temperatura alcance el pivote. Y el canal azul, tiene que ir disminuyendo conforme aumente la temperatura, de tal manera que en la temperatura mínima tendrá todo el canal azul, nada de rojo y nada de verde.
    - Si es mayor, el canal rojo tiende a subir conforme aumenta la temperatura. El canal verde está al máximo en la temperatura pivote, por

lo que tenderá a bajar conforme aumenta la temperatura. Y el canal azul se mantiene a 0 todo el tiempo.

De esta forma obtenemos un gradiente de temperatura en los 3 canales dado un rango de temperaturas.

Para obtener el rango de temperaturas no se consume tiempo de ejecución, puesto que la lectura se hace antes de cargar la escena, teniendo que pasar por todos los elementos e ir insertándolos en una estructura de datos para leerlos posteriormente poco a poco.

En la interfaz se ha añadido un botón para pasar 30 días en la lectura del archivo, de forma que las filas que tienen que pasar son:

30 días x 24 horas = 720 filas

Cada vez que lea una temperatura, se modificará el parámetro de la **temperatura externa** y se **aplicará directamente** a la última capa de aire de la atmósfera.

Se simularán los datos de temperaturas recogidas del 1 de septiembre de 2018 hasta el 31 de junio de 2020. En cuanto a la temperatura de la roca, se hará una media de las primeras x temperaturas para hacernos una idea de la temperatura que tendrá. Esto se calculará al iniciar la escena.

6.7. Escala de colores

Para la representación de la temperatura, se utilizará una escala de color de temperaturas, donde el más caliente se representa como rojo, y el más frío como el azul, como se puede ver en la *Ilustración 47*.

Ilustración 46. Gama de colores para representar la temperatura. (https://www.blog.lamparas.es/temperatura-color-concepto/)

Para que los colores de las células cambien en función de la temperatura, hay que establecer la temperatura máxima y mínima que van a alcanzar. Una vez realizada la lectura del fichero, sabremos estos datos. Como se puede observar en la *Ilustración 47*, hay varios colores en la escala. Por lo que tendremos que tener varios rangos para alcanzar estos colores. Al haber 5 rangos, hay que dividir la diferencia de temperatura (temperatura máxima - temperatura mínima) entre 5. Esto es para saber en qué rango se encuentra la temperatura y si tiene que sumar o restar valores de un canal u otro (puesto que trabajamos con RGB).

Las temperaturas deben de ser de valor positivo. Si la temperatura mínima, por ejemplo, es negativa, habrá que sumarle lo que haga falta para que salga positiva. Esta suma también se aplicará a la temperatura máxima y a la temperatura actual de la célula. Así trabajamos siempre con valores positivos y evitamos tener valores negativos en el RGB, puesto que no funcionará bien.

En Unity, los valores que se almacenan en el RGB estarán siempre entre 0 y 1, no entre 0 y 255. Por lo que el valor que salga, habrá que normalizarlo entre 0 y 1.

6.8. Pruebas y resultados

Una vez realizada la puesta en marcha del prototipo, debemos visualizar los resultados y almacenarlos para poder realizar estudios sobre ello. Por ejemplo, generando un vídeo que permita ver esos cambios que buscamos. Como en este prototipo los datos se visualizan muy lentamente, lo ideal sería generar una foto por

cada vez que se procese todo el autómata. Y al final, juntar esos fotogramas generando un vídeo.

En Unity, para realizar pantallazos sobre la escena que se está visualizando por la cámara, se utiliza el método `ScreenCapture.CaptureScreenshot()`, pasándole como parámetro la ruta con el nombre del archivo. Tendremos una variable global que vaya contando el número de fotogramas que lleva para que las fotos tengan un orden, importante para generar el vídeo de forma correcta. Como se puede ver en la *Ilustración 48*, se ha añadido un botón [80] de grabación en la interfaz para pausar o reanudar la grabación.

Ilustración 47. Botón desactivado y activado de grabación. Elaboración propia.

Esta grabación únicamente genera fotos por cada *frame*. Luego hay que juntarlas todas ellas y convertirlas en un vídeo. Unity no permite generar un vídeo directamente, por lo que se utilizará un programa externo. Este programa externo está programado en Python, que es más sencillo manipular imágenes con la librería de OpenCV. El programa Python consta del siguiente procedimiento [*79*]:

```
procedimiento generarVideo (pathFotos: cadena, fps: entero)
      frames[ ]
      fotos[ ] = obtenerArchivos(pathFotos)
      Si fotos no está vacío:
            ordena(fotos)
            Por cada foto (i):
                  img = imread(pathFotos + foto[i]) //Método de cv2
                  dimensiones = img.shape
                  frames.add(img)
                  remove(pathFotos + foto[i])) //Elimina la foto
            fin iteración
            video = creaArchivoVideo(pathSalida, format, fps, dimesiones)
```

```
            Por cada frame (i):
                  video.write(frame[i])
            fin iteracion
fin procedimiento
```

Como se puede apreciar en el procedimiento, cada vez que se almacena una imagen devuelta por el método `imread()` de cv2 (OpenCV), se elimina. Esto hace que las fotos se borren del disco y evitamos que se sature la máquina en la que se ejecuta. Este método `imread()` devuelve una matriz con las dimensiones de la resolución de la imagen, almacenando en cada posición la información del pixel. Y como se almacenan en una estructura, la información de esta imagen eliminada no se pierde, pudiendo generar el vídeo posteriormente.

Para ejecutar el programa en Python ver [*Anexo II*].

7. Conclusión y trabajo futuro

Después de todos los estudios realizados en este proyecto, se podría decir que queda mucho trabajo por hacer todavía. Se han establecido las bases de un simulador que no se había visto antes. La optimización y el desarrollo del proyecto puede abrir muchas puertas a la investigación de este campo tan poco estudiado. No estamos hablando solo de cuevas que estén en la Tierra, sino también de futuras investigaciones con cuevas marcianas. La parametrización de los distintos elementos que componen el autómata es clave para lograr una aplicación escalable. Puede haber cambios desde la implementación de los distintos principios físicos en función de las leyes físicas del entorno, como las trampas de aire frío que se generan en ciertas circunstancias [*79*], hasta las propiedades que conforman los distintos materiales y fluidos del entorno.

Los resultados obtenidos han sido bastante prometedores, logrando confirmar algunos de los efectos que se buscaban y teniendo mayor información sobre ellos. Toda esta información nos va a permitir analizar las condiciones en las que sucede y poder interactuar con ellas para simular ciertos escenarios que nos interesen.

Pero todavía queda mucho por pulir. Aplicar más principios físicos como la incidencia del Sol sobre la superficie de la roca, y como varía con respecto a la noche. Incluir mecánica de fluidos para el flujo de aire externo que pueda haber en función de las condiciones climatológicas. La definición de fluidos líquidos como el agua. La generación procedural más focalizada en zonas de interés. Y por supuesto, mayor optimización del prototipo porque consume muchos recursos al intentar realizar muchos cálculos en un breve periodo de tiempo.

Bibliografía

H.Y. McSween, J.E. Moersch, D.M. Burr, W.M. Dunne, J.P. Emery, L.C. Kah, and M.C. McCanta. *Conductive Heat flow.* In Planetary Geoscience, pages 139 - 142. (2019)

Stephen J. Blundell and Katherine M. Blundell (2009). *Concepts in Thermal Physics*

J. Steven Kite, John Tudek. *Cold-Air trap temperature records support simple High-Density Air-Flow mechanisms at an appalachian limestone cave entrance sinkhole*

https://google.github.io/liquidfun/ [*Última vez visitado: 07 de septiembre de 2021*]

https://nbviewer.jupyter.org/github/barbagroup/CFDPython/blob/master/lessons/14_Step_11.ipynb [*Última vez visitado: 07 de septiembre de 2021*]

https://nbviewer.jupyter.org/github/barbagroup/CFDPython/blob/master/lessons/15_Step_12.ipynb [*Última vez visitado: 07 de septiembre de 2021*]

https://docs.unity3d.com/ScriptReference/index.html [*Última vez visitado: 07 de septiembre de 2021*]

https://www.thermal-engineering.org/es/que-es-la-ley-de-conduccion-termica-de-fourier-definicion/ [*Última vez visitado: 07 de septiembre de 2021*]

http://www.quimicafisica.com/ley-de-fourier-conduccion-termica.html [*Última vez visitado: 07 de septiembre de 2021*]

https://www.sciencedirect.com/topics/physics-and-astronomy/fourier-law [*Última vez visitado: 07 de septiembre de 2021*]

https://es.wikipedia.org/wiki/Conductividad_t%C3%A9rmica [*Última vez visitado: 07 de septiembre de 2021*]

https://www.researchgate.net/figure/Schematic-behaviour-of-air-circulation-in-an-ice-cave-A-Cold-air-trap-During-winter_fig3_228492766 [*Última vez visitado: 07 de septiembre de 2021*]

https://2013.es.pycon.org/media/pycones-CAChemE-python-ingeneria-quimica.pdf [*Última vez visitado: 07 de septiembre de 2021*]

https://aspgems.com/metodologia-de-desarrollo-de-software-iii-modelo-en-espiral/ [*Última vez visitado: 07 de septiembre de 2021*]

https://www.cuestioneslaborales.es/jornada-trabajo-descansos-segun-estatuto-los-trabajadores/ [*Última vez visitado: 07 de septiembre de 2021*]

https://www.jobted.es/salario/ingeniero-inform%C3%A1tico [*Última vez visitado: 07 de septiembre de 2021*]

https://programacionpython80889555.wordpress.com/2020/10/08/creando-video-a-partir-de-imagenes-en-python-con-opencv/ [*Última vez visitado: 07 de septiembre de 2021*]

https://docs.unity3d.com/es/530/ScriptReference/UI.Slider.html [*Última vez visitado: 07 de septiembre de 2021*]

https://docs.unity3d.com/ScriptReference/GameObject.html [*Última vez visitado: 07 de septiembre de 2021*]

https://docs.unity3d.com/ScriptReference/UIElements.Button.html [*Última vez visitado: 07 de septiembre de 2021*]

https://docs.unity3d.com/ScriptReference/Collider.html [*Última vez visitado: 07 de septiembre de 2021*]

Anexo

Anexo I: Instalación de Jupyter.

Para instalar pip:

```
sudo apt install python3-pip
```

Para instalar Jupyter:

```
pip3 install jupyterlab
```

Para instalar Jupyter Notebook:

```
pip3 install notebook
```

Para ejecutar JupyterLab:

```
jupyter lab
```

o

```
jupyter-lab
```

En el caso de que no ejecute, hacerlo de forma manual:

```
/home/username/.local/bin/jupyter-lab
```

Sustituyendo username por tu nombre de usuario.

Se abrirá una ventana en el navegador. Ahora tenemos que abrir un notebook y ejecutar las siguientes líneas para instalar las librerías, tal y como se aprecia en la *Ilustración 49*:

```
[3]: import sys
     !{sys.executable} -m pip install numpy

Collecting numpy
  Using cached https://files.pythonhosted.org/
Installing collected packages: numpy
Successfully installed numpy-1.19.5

[4]: import sys
     !{sys.executable} -m pip install matplotlib

Collecting matplotlib
```

Ilustración 48. Código ejecutable en Jupyter. Elaboración propia.

Anexo II: Elaboración de un vídeo a partir de las pruebas obtenidas.

Para ejecutar este programa, es recomendable que se encuentre en la misma ruta que la carpeta '*videos*' (dónde se van a guardar los pantallazos tomados durante la ejecución del programa). Así el *path* para leer las fotos es más sencillo y siempre sería el mismo en cualquier máquina (`pathFotos = './videos/'`). Una vez en la carpeta, podemos ejecutar el programa con la instrucción `python` si la tenemos instalada en nuestro equipo. Dicho ejecutable corresponde con el fichero '`Ejecutable/TFG_Data/generarVideo.py`' que se adjunta a esta memoria. La salida tiene que ser algo similar a la *Ilustración 50*.

```
(python-env) E:\Universidad\TFG\Unity\TFG_V32\TFG>python generarVideo.py
Introduzca el nombre del video: prueba_01
./videos/foto0.png
./videos/foto1.png
./videos/foto2.png
./videos/foto3.png
./videos/foto4.png
./videos/foto5.png
./videos/foto6.png
./videos/foto7.png
```

Ilustración 49. Salida del programa Python. Elaboración propia.