Generador de mapas 2D con autómatas celulares

Pablo Rodríguez Saseta

Trabajo de Fin de Máster para Máster Universitario en Lógica Computacional e Inteligencia Artificial

Resumen

La generación procedimental, esto es, la generación de contenido mediante algoritmos pseudoaleatorios, es un campo de estudio relativamente nuevo que es de gran interés para varios campos creativos, especialmente el de los videojuegos debido a su capacidad para aumentar la variedad de cada título y ofrecer experiencias únicas en cada sesión de juego. Dentro de este campo, es de especial interés la generación procedimental de los niveles en los cuales se desarrolla el juego, debido a que esto ofrece cambios radicales a cada experiencia de juego.

En este trabajo se presenta un algoritmo para la generación de mapas en dos dimensiones enfocado al ámbito de los videojuegos usando autómatas celulares con el objetivo de presentar una alternativa válida a los métodos tradicionales basados en ruido continuo. Se han desarrollado tres modelos diferentes para el propósito, analizando sus prestaciones, particularidades y fallas, hasta finalmente llegar a uno capaz de generar mapas variados y con la suficiente flexibilidad, robustez y rendimiento como para usarse en un ámbito de desarrollo real.

Este documento está estructurado de la siguiente forma: en el primer capítulo presentamos los conceptos preliminares y exploramos el contexto y estado del arte de la generación procedimental hoy en día. Tras ello, hay tres capítulos presentando, explicando y analizando cada uno de los modelos desarrollados, y finalmente un último capítulo con las conclusiones del trabajo, tras el cual se incluye la bibliografía.

Índice

1. Introducción

1.1. Contexto

La generación procedimental, o *procedural generation* en inglés, es el método de crear datos mediante algoritmos en lugar de forma manual. Esta técnica se usa ampliamente en el cine tanto de animación como de imagen real, pero también se lleva usando en el ámbito de los videojuegos desde tan antiguamente como 1980, con el videojuego *Rogue*, juego que ha dado nombre a todo un género que tiene como una de sus características principales la generación procedimental, los *roguelikes*.

Hoy en día, la generación procedimental es usada ampliamente en el medio de los videojuegos para la creación en tiempo real de contenido, como niveles, único a cada sesión o experiencia de juego. Esta técnica alarga enormemente la longevidad de los juegos que la utilizan y pueden incluso ser partes fundamentales de la experiencia, como en *No Man's Sky* [3], que genera algorítmicamente un universo completo, incluyendo planetas, flora y fauna.

A nuestro parecer, la generación procedimental de terreno[1] es uno de los aspectos más interesantes y útiles de esta técnica. Mientras que su uso para la generación de texturas o vegetación es increíblemente útil para ahorrar trabajo, usarla para generar los niveles en sí tiene un impacto directo en la jugabilidad tanto mecánica como estéticamente, y permite alargar la vida de los videojuegos que la usan en gran medida. *Minecraft*, el videojuego más vendido de la historia, tiene precisamente como uno de sus aspectos y atractivos fundamentales el hecho de que su mundo es generado de forma completamente procedimental y con una granularidad lo suficientemente grande como para permitir a los jugadores moldear el mundo a su antojo.

1.2. Generación procedimental de terreno

Existe un problema con respecto a las técnicas de generación procedimental en la industria del videojuego: es difícil encontrar documentación formal, esto es, artículos académicos sobre ella. Estas técnicas se usan en una mucha mayor proporción en la escena *indie*, compuesta por individuos y equipos pequeños, que en la "triple A", el dominio de grandes empresas. Estos equipos pequeños no suelen publicar sus descubrimientos o sus técnicas, y cuando lo hacen, generalmente lo hacen en formatos más informales o sin elaborar

[1] En este documento, con *terreno* nos referimos a la superficie terrestre, esto es, tierra, montañas, valles, etcétera

en gran detalle sobre su desarrollo o implementación final. Algunos artículos científicos existen sobre el tema, pero son relativamente pocos, y muy raramente implementan o apoyan sus resultados en juegos reales. Todo esto resulta en grandes dificultades para estudiar lo que han hecho otros antes, requiriendo por tanto reinventar la rueda constantemente. Afortunadamente, existen muchas guías básicas que cubren las bases más sencillas de las técnicas más populares [12] [9], y los motores de desarrollo más conocidos suelen traer en el paquete básico el código para implementarlas. Sin embargo, estas guías raramente entran en profundidad sobre el por qué o el cómo del funcionamiento de la técnica, estando generalmente orientadas a la simplicidad para ser inteligible por público sin demasiado conocimiento del tema.

Volviendo a la generación procedimental de terreno, la técnica más ampliamente extendida es la de los paisajes fractales. Esta técnica consiste en algoritmos estocásticos e iterativos que generan superficies con ciertos comportamientos fractales, sin ser fractales puros. Dentro de esta técnica el algoritmo más popular es el basado en ruido Perlin o derivados.

El ruido Perlin es una función de ruido gradiente, también muy popular en la generación procedimental de texturas en el cine animado. Tiene la ventaja respecto a funciones de ruido aleatorio el hecho de ser continuo y derivable y cambiar de forma suave y uniforme, lo que en la práctica resulta en contenido con aspecto realista. La forma más simple de usarlo para generar terreno es generar un mapa de altura a partir de él y transformarlo en montañas y llanuras, aunque también se puede combinar con otros métodos o incluso consigo mismo para conseguir otros resultados. El previamente mencionado *Minecraft* es uno de los ejemplos más notables en cuanto al uso de ruido Perlin para la generación procedural.

El ruido Perlin fue desarrollado en 1983 por Ken Perlin y es libre de patentes, pero tiene un sucesor desarrollado por el susodicho Perlin en 2002 que arregla algunos de los problemas que presenta este, como la alta complejidad con mayores dimensiones, llamado ruido Simplex [5, 13]. Al contrario que el Perlin original, este tiene *copyright* asociado que limita su uso en ciertos contextos, por lo que el ruido Perlin sigue siendo el que ve uso mayoritario. En respuesta a las limitaciones de Simplex, ha aparecido también OpenSimplex [8], una alternativa a Simplex de teoría muy similar pero con licencia *open-source*.

1.3. Autómatas Celulares

Un autómata celular es un sistema dinámico que evoluciona en pasos discretos, compuesto por una rejilla o matriz n-dimensional de celdas ocupadas por números enteros, una función de transición f y un entorno o vecindad v.

En cada paso i de su evolución, aplica f a cada una de sus celdas, teniendo como entrada el número de la propia celda y los de las celdas vecinas a esta según la vecindad v. La salida de f es el valor que tendrá la celda en el paso siguiente, $i + 1$.

El autómata celular más famoso y sobre el cual la mayoría habrá oído hablar es Conway's Game of Life [4], un autómata celular de dos dimensiones de reglas extremadamente simples que sin embargo modela de forma abstracta la vida de las poblaciones, y con la capacidad de usarse como una máquina de Turing [14]. También existen muchos otros autómatas de una dimensión que siguen siendo objeto de investigación, como las reglas 184 o 30 de la clasificación de Wolfram [15].

Los autómatas celulares permiten modelar sistemas muy complejos con interacciones de múltiples agentes mediante reglas significativamente más sencillas, haciéndolos muy útiles en el desarrollo de videojuegos. Reducir la complejidad es esencial para hacer simulaciones complejas en tiempo real, especialmente cuando el programa debe también simular paralelamente el resto del mundo y reaccionar a los *inputs* del jugador. Algunos de los usos que se les puede dar a los autómatas en este ámbito es modelar la IA de una horda de enemigos, como se puede ver en el artículo de Stawonir Nikiel [11], o simular dinámica de fluidos, visto en la investigación de Heintz, Christian y Grunwald, Moritz y Edenhofer, Sarah y Hähner, Jörg y von Mammen, Sebastian [6].

1.4. Generación procedimental y Autómatas Celulares

Volviendo al ámbito de la generación de terreno, lamentablemente no hay apenas documentación sobre el uso de autómatas celulares en este. Un uso muy popular para los autómatas es la generación de cuevas, usando reglas parecidas al *Game of Life* para generar sistemas de cuevas aleatorios y con apariencia natural. En Lawrence Johnson, Georgios Yannakakis y Julian Togelius [7] se demuestra que es posible usar este modelo para generarlas en tiempo real y tal que sean infinitas y conexas. Más tarde, Yuri P. A. Macedo y Luiz Chaimowicz desarrollaron una mejora del algoritmo con el objetivo de que este creara un camino crítico[2] durante la generación y que adaptase las cuevas a este [10]. Chad Adams y Sushil Louis [1] demostraron que es posible usar los autómatas para generar laberintos, combinados adicionalmente con un algoritmo genético para determinar la mejor función de transición del autómata, aunque hasta donde llega nuestro conocimiento esta técnica no se

[2]En el ámbito del diseño de niveles, el camino crítico es el camino más corto que debe seguir el jugador para llegar hasta el objetivo sin desviarse

ha usado en ningún juego real, al contrario que el algoritmo de generación de cuevas.

1.5. Objetivos

Los modelos de autómatas celulares para la generación procedimental de terreno de la sección anterior obtienen muy buenos resultados, pero solamente están probadas para una aplicación muy específica, la de generar cuevas. No existe o no se ha popularizado ningún autómata que genere terreno de forma mucho más genérica y compleja, tal y como el ruido Perlin permite.

El objetivo de este trabajo es el de investigar el uso de autómatas celulares aplicados a la generación de terreno 2D de uso general, aunque generalmente enfocado al ámbito del desarrollo de videojuegos, y desarrollar finalmente un modelo de autómata celular que tenga una utilidad en este campo similar a la del ruido Perlin, esto es, una base sólida sobre la que poder fundamentar diversos algoritmos de generación procedimental. No se pretende entonces alcanzar un producto competitivo a nivel comercial por su cuenta, sino desarrollar un método que pueda usarse en aplicaciones reales.

De forma más concreta, se pretende que el autómata genere mapas de vista aérea a nivel de suelo, teniendo distintos tipos de terreno como agua, tierra o hierba, y que resulte en accidentes asociados, como montañas, lagos, ríos o penínsulas. Se quiere también que mediante manipulación de los parámetros el autómata genere mapas de naturaleza distinta, por ejemplo continentes, archipiélagos o desiertos[3].

1.6. Métricas y evaluación

A la hora de evaluar la adecuación del modelo desarrollado, hay dos aspectos principales que estudiar: el modelo en sí, y la calidad de los mapas generados. Este segundo aspecto es especialmente problemático debido a su subjetividad: la única manera de decidir si los mapas generados por el modelo son "buenos", "divertidos" o "interesantes" es preguntar a voluntarios o jugadores y recopilar los datos. El previamente mencionado artículo original sobre cuevas de Lawrence Johnson, Georgios Yannakakis y Julian Togelius [7] no es capaz de presentar ninguna métrica para decidir si los mapas son buenos más allá de su propia percepción, mientras que el artículo sobre cuevas con camino crítico de Yuri P. A. Macedo y Luiz Chaimowicz [10] basado en

[3]En el ámbito de la generación de terreno, los diferentes estilos o naturalezas que tienen ciertos mapas o ciertas partes de cada mapa se denominan "biomas", p.e. bioma de montaña, bioma nevado o bioma de océano

el anterior admite que la única manera de juzgar la calidad de los resultados requeriría construir un juego alrededor de ellos, lo cual requiere mucho tiempo y esfuerzo. El autómata celular para la generación de laberintos de Chad Adams y Sushil Louis [1] sí que implementó esta metodología con el objetivo de guiar a su algoritmo genético de optimización, lo que resultó en un generador cuya calidad a los ojos del público objetivo está asegurada. Por limitaciones tanto del trabajo como de las circunstancias alrededor de las cuales se desarrolla no se ha llevado a cabo ningún estudio de este tipo, quedando las conclusiones subjetivas sobre calidad enteramente desde nuestra perspectiva.

Para evaluar el modelo en sí, se han establecido criterios a cumplir de genericidad, flexibilidad y robustez que aseguren la viabilidad de su uso para los objetivos previamente definidos en la sección 1.5. Debido a que estas métricas son definidas objetivamente, es posible evaluar los resultados de la investigación de forma más rigurosa y sin necesitar un estudio con múltiples sujetos. El valor final para cada métrica para cada modelo estudiado será Sí o No, en función de si el modelo cumple el objetivo que establece la métrica. Las métricas a estudiar son las siguientes:

- **Flexibilidad**: Cómo de capaz es el modelo de generar distintos tipos de terreno variando sus parámetros: por ejemplo, que con una configuración cree archipiélagos y con otra continentes desérticos. El modelo será flexible si permite al menos tres tipos de terreno distintos cualesquiera, suponiendo que con esa cantidad se podrá configurar para una mayor variedad de terrenos. *Palette swaps*[4], en los que se cambia la interpretación de los diferentes estados de un autómata para convertir, por ejemplo, un archipiélago en un desierto, no cuentan de cara a definirlo como flexible.

- **Continuidad**: Un modelo es continuo si cambiar los parámetros de forma gradual resulta en cambios graduales en los resultados finales. Un modelo es continuo por tanto si no tiene umbrales en los parámetros que causen comportamiento radicalmente diferente al pasarse.

- **Usabilidad**: Cómo de fácil de configurar y modificar son los parámetros para obtener los resultados deseados. Dicho de otra forma, cómo de intuitiva es la configuración del modelo respecto a los mapas generados. Un modelo es usable si todos sus parámetros definen de forma simple y directa su influencia en el resultado final.

[4]En el ámbito de los videojuegos, *palette swaps* o "cambio de paleta de colores" es el método de generar contenido nuevo a base de recolorear o cambiar ligeramente contenido antiguo.

- **Estabilidad**: Un modelo es estable si los resultados para una misma configuración son siempre del mismo tipo pero no son tan parecidos como para ser idénticos. Por ejemplo, una configuración determinada siempre genera desiertos, pero no el mismo desierto. Se considera estable si para las configuraciones usadas para definirlo como flexible se cumple que son estables, con la suposición de que será posible hacerlo estable para las configuraciones que se desee. Esta métrica, al contrario que las anteriores, requiere estudiarse de forma estrictamente experimental; esto es, ejecutar el modelo un número lo suficientemente grande de veces como para asegurar que los resultados generados son los deseados. Se exigirá al menos 20 ejecuciones por cada configuración para marcar el modelo por estable.

Hay otra serie de factores a considerar en un generador o los mapas que produce que son deseables para un sistema real, pero que generalmente en dichos sistemas reales pueden ser rectificados mediante combinaciones con otras técnicas o modificaciones específicas al modelo para las necesidades del producto. Estos factores se estudiarán brevemente pero no tendrán demasiado peso sobre la valoración del modelo.

- **Variedad**: Un mapa tiene variedad si en un único mapa conviven diferentes tipos de paisajes: por ejemplo, un desierto a un lado y un archipiélago a otro. Normalmente estas cuestiones se resuelven en la práctica definiendo de forma exterior al generador "biomas" para cada sector del juego y usando diferentes parámetros del generador para cada uno.

- **Granularidad**: Cuán fácil es adaptar el generador a varias escalas: es decir, cuánto *zoom* puede hacerse en una región cualquiera del mapa. Los algoritmos de ruido continuo, como Perlin, tienen por naturaleza granularidad infinita. Un autómata celular tiene por el mismo motivo granularidad finita, por lo cual sería necesario un procedimiento extra en caso de querer aumentar la granularidad.

- **Rendimiento**: Cuál es la complejidad del algoritmo y su velocidad en pruebas reales. Es deseable que los algoritmos de generación procedimental tengan complejidad baja para poder generarse en tiempo real, aunque algunos, como el de *Terraria*, pueden permitirse ser más lentos al generar todo el terreno jugable de una vez y almacenarlo para uso posterior. Obviamente, sin embargo, si el rendimiento es tan bajo como para no ser utilizable en la práctica, el algoritmo es inútil.

- **Entropía**: Cuán ordenados son los resultados finales entre ser completamente caóticos y ser geométricos o fractales. Resultados demasiado caóticos son efectivamente ruido aleatorio, mientras que figuras geométricas perfectas no son naturales y por ende van en contra del espíritu de la generación procedimental. Por lo tanto, lo idóneo es que el modelo esté en un punto medio o incluso permita ajustar este parámetro.

1.7. Netlogo

Netlogo [2] es un entorno programable de modelado para simulaciones multiagente. Este programa permite realizar simulaciones de agentes individuales y autómatas celulares muy fácilmente, como la propagación de un virus o el *Game of Life* de Conway, gracias a un lenguaje de modelado muy sencillo y muchas herramientas para este propósito.

Para este proyecto, usaremos Netlogo 6.1.1 para el desarrollo y análisis de los modelos, ya que permite crear autómatas celulares muy fácilmente y está muy optimizado para este uso. Adicionalmente, la interfaz gráfica que provee es extremadamente útil para visualizar rápidamente los resultados y la evolución de los autómatas en tiempo real. Junto a este documento se adjuntan los diferentes modelos desarrollados durante el proyecto.

2. Modelo basado en *GoL*

2.1. Definición

Para el primer acercamiento al problema, se ha usado una variación de la misma técnica usada para la generación de sistemas de cuevas de Lawrence Johnson, Georgios Yannakakis y Julian Togelius [7]. Para la generación de cuevas, solamente existen dos estados: pared y suelo, lo que simplifica las reglas de transición a simplemente los rangos del número de vecinos que son suelo para los cuales la celda también será suelo. Este modelo para generar cuevas es, a su vez, una variante del famoso Juego de la Vida de Conway [4].

Para este modelo, sin embargo, el objetivo es tener una mayor variedad de posibles estados para cada celda que definan una más amplia variedad de terreno, por lo que la función de transición basada en el número de vecinos necesita replantearse. En este modelo, se tendrán los estados agua, arena, tierra y montaña, teniendo estos asignados los números enteros 0, 1, 2, y 3 respectivamente, lo que los pone en una jerarquía estricta basada aproximadamente en su "altura".

Se definen tres puntos de corte, c_{agua}, c_{arena} y c_{tierra}, como parámetros del modelo, teniendo que $0 < c_{agua} < c_{arena} < c_{tierra}$. La función de transición toma como entrada los valores de las celdas del vecindario de Moore[5], considerando la rejilla como un toroide de cara a los bordes[6], y el valor de la propia celda, y hace la media m de estos valores, tras lo cual devuelve el resultado de pasar a la función $g(m)$ la media.

$$g(m) = \begin{cases} 0 & \text{si } m \leq c_{\text{agua}} \\ 1 & \text{si } c_{\text{agua}} < m \leq c_{\text{arena}} \\ 2 & \text{si } c_{\text{arena}} < m \leq c_{\text{tierra}} \\ 3 & \text{si } m > c_{\text{tierra}} \end{cases}$$

2.2. Resultados

Para este primer experimento, se ha usado un tamaño de rejilla de 50x50 celdas, para iteraciones rápidas; además, para cada ejecución, se ha generado un estado inicial aleatorio con probabilidad uniforme para cada tipo de *tile*. Para probar el modelo y sus resultados, se han hecho del orden de cien ejecuciones, cambiando ligeramente los parámetros c_{agua}, c_{arena} y c_{tierra} hasta

[5]El vecindario de Moore está compuesto por las 8 celdas que rodean a la celda original en las cuatro direcciones cardinales y las cuatro intercardinales.

[6]Esto es, las celdas del borde superior se consideran adyacentes a las del borde inferior, y las del borde izquierdo a las del borde derecho

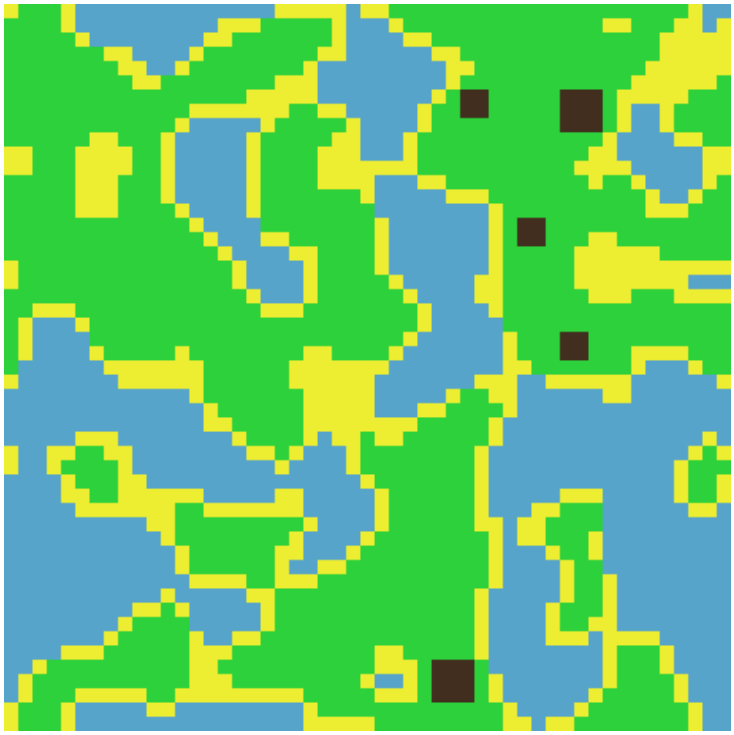

Figura 1: Ejemplo de mapa generado con el primer modelo

que se ha dado con unos valores que dan unos resultados aceptables. Se han probado distintos valores para el número de pasos de evolución y se han encontrado ciertos problemas con ello, detallados en la métrica de estabilidad.

Con este primer modelo se ha conseguido generar un único tipo de mapa, una especie de continente de tierra serpenteante con arena en la costa y alguna montaña dentro de él, como se puede observar en las figuras 1, 2 y 3. Plantea sin embargo varios problemas y no puntúa bien en las métricas:

- **Flexibilidad: No**. El modelo es bastante inflexible, siendo muy difícil generar resultados diferentes a los mostrados en las figuras. Además, únicamente tiene tres parámetros posibles para modificar, lo que dificulta más la tarea.

- **Continuidad: No**. El modelo no presenta continuidad en sus parámetros: hay estrechos en los que cambiar los valores no afecta al resultado, y puntos de corte que modifican enormemente la evolución del modelo. Esto es debido a que en esos puntos los resultados de ciertos vecindarios cambian: por ejemplo, una frontera agua-arena de 0.77 hará que un *tile* con un vecindario de 2 de agua y 7 de arena pase a ser agua, mientras que una frontera de 0.78 hará que siga siendo arena.

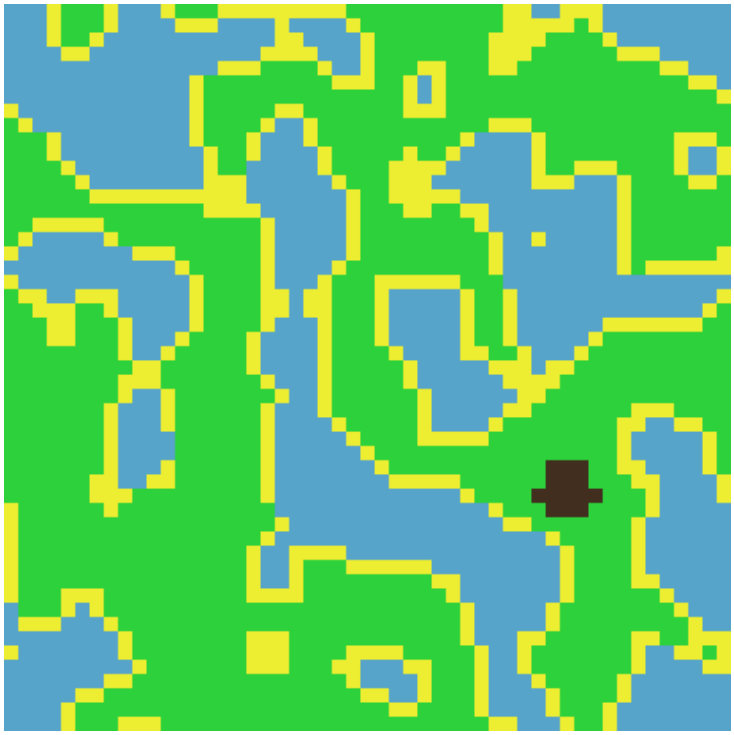

Figura 2: Ejemplo de mapa generado con el primer modelo

- **Usabilidad: No**. Los parámetros afectan cada uno a múltiples aspectos, lo que hace difícil configurarlos para encontrar los resultados deseados, necesitando mucha prueba y error. No solamente eso, añadir nuevos estados o parámetros complica incluso más los parámetros existentes, por lo que no es escalable ni modificable.
- **Estabilidad: No**. Casi todas las configuraciones de parámetros probadas tienden a converger al mismo estado tras apenas diez pasos de evolución, uno en el que todos los *tiles* tengan el mismo valor, usualmente el mayor o el menor en la jerarquía (agua o montaña). Esto significa que solamente las primeras evoluciones son usables: las figuras muestran resultados para 5 iteraciones. Esto es debido a un efecto de bola de nieve: si en un paso las celdas tienen facilidad para pasar a ser agua, las celdas del siguiente paso lo tendrán más fácil aún.

Hay también un análisis interesante que hacer en los otros aspectos menores:

- **Granularidad**: El modelo únicamente sirve para escala macro, debido a varios motivos: el modelo solamente mira su entorno más inmediato,

Figura 3: Ejemplo de mapa generado con el primer modelo

y tiene la tendencia a ordenar las agrupaciones de *tiles* tal que haya una gran cantidad de uno de los tipos, gran cantidad del otro dos puestos más arriba, y una fina línea del elemento medio separándolos (p.e., la arena siempre divide con una línea de grosor 1 o 2 el agua y la tierra, aunque en otros experimentos la tierra ha separado de la misma forma montaña y arena). Permitir una mayor escala pasaría por aumentar el vecindario, pero eso provocaría incluso mayor convergencia pues acabaría haciendo el valor medio de un área mayor del mapa. Esto significa que algo tan trivial como hacer una playa a escala micro sería imposible o extremadamente difícil.

- **Entropía**: Tiende bastante más a la ordenación completa que a tener caos, "erosionando" las formaciones que se salgan de lo establecido.
- **Variedad**: Ninguna; todo el mapa tendrá siempre un aspecto parecido en toda su extensión.

2.3. Complejidad y rendimiento

Para cada paso de la evolución, cada *tile* hace un cálculo que escala con el tamaño de su vecindario. Debido a que el tamaño del vecindario y el cálculo es constante, la complejidad del algoritmo para cada paso es *O(n)*, siendo n el número de *tiles* del modelo.

Se han hecho pruebas técnicas para comprobar el rendimiento real del algoritmo. Estas pruebas se han hecho en Netlogo, optimizado para este tipo de algoritmos, por lo que en otros lenguajes de uso general o de más alto nivel estos números son muy dados a aumentar. El equipo usado para las pruebas es el siguiente:

- **CPU**: AMD Ryzen 5 3600, sin *overclocking*
- **RAM**: 16 GB 3200 MHz DDR4
- **Almacenamiento**: INTEL SSD 660P

Se han hecho 20 ejecuciones para cada prueba, siendo los resultados presentados la media de todas las pruebas ejecutadas, redondeadas hacia arriba.

Prueba	Generaciones	Tamaño	Tiempo
1	50	50x50	43ms
2	50	150x150	408ms
3	500	50x50	412ms

Como era de esperar, el tiempo invertido aumenta de forma casi lineal con las generaciones y el número de *tiles*: multiplicar por diez las generaciones multiplica el tiempo por diez, y aumentar el número de *tiles* por nueve produce el mismo efecto. Como era de esperar, también es extremadamente rápido gracias al entorno Netlogo.

2.4. Conclusiones

Hemos desarrollado un modelo muy simple de autómata celular que es capaz de generar continentes sinuosos, pero con muchas limitaciones. Este primer acercamiento al proyecto tiene bastantes problemas, no llegando a ninguno de los objetivos establecidos, pero aún así, los resultados iniciales son prometedores. Este modelo tiene mucho espacio para experimentar y mejorar, pero también posee problemas fundamentales que ninguna cantidad de arreglos o mejoras puede arreglar, por lo que es necesaria otra alternativa.

3. Modelo basado en *GoL* mejorado

3.1. Definición

Este modelo es una mejora del modelo anterior, con cambios orientados a disminuir los principales problemas de este. Los cambios introducidos se resumen en dos grupos: cambios en la función de transición y cambios en la generación del estado inicial.

3.1.1. Cambios en función de transición

Desde el punto de vista del algoritmo en sí y menos desde el punto de vista de la facilidad para el usuario final, uno de los problemas principales es la convergencia del modelo, que tiende a "aplanar" todo lo posible el mapa, hasta acabar tras las suficientes iteraciones con el mismo estado para todas las celdas. Para este problema, se han implementado dos mejoras: limitar los saltos entre estados e introducir un valor de "inercia" para los *tiles*.

El primer cambio ha sido el establecer que un *tile* solamente pueda cambiar a uno de los estados adyacentes en la jerarquía en cada iteración, sin dar saltos a un extremo u otro. Esto significa que, por ejemplo, la arena (1) solamente puede pasar en el mismo paso a ser agua (0) o tierra (2). El razonamiento de esta nueva regla es que, de esta forma, se ralentiza el efecto bola de nieve que ocurre cuando un *tile* o agrupación de *tiles* queda rodeado por terreno del otro extremo de la jerarquía.

El segundo cambio ha sido el de introducir inercia para las celdas, un parámetro llamado $p_{inercia}$. Este valor modifica los puntos de corte c_{agua}, c_{arena} y c_{tierra}, tal que una celda solamente cambia de estado cuando supera el punto de corte entre estados por una cantidad mínima, en lugar de hacerlo en cuanto lo supere estrictamente. Este cambio por tanto suaviza los puntos de corte de los parámetros del modelo anterior al imponer que haya una diferencia significativa para que un *tile* cambie de estado.

3.1.2. Definición del estado inicial

Se ha querido explorar la influencia del estado inicial en los resultados finales del modelo. Se han añadido por tanto parámetros que permitan cambiar la probabilidad de cada tipo de *tile* en el estado inicial. Adicionalmente, se ha añadido una nueva funcionalidad al modelo: la posibilidad de crear "semillas".

En esencia, se especifica al modelo que cree un determinado número de agrupaciones de cierto *tile* de determinado tamaño en el estado inicial, creando así "semillas" para ciertas características del modelo final. Por ejemplo,

especificar tres semillas de agua de gran tamaño para que el resultado final tenga mayor probabilidad de tener tres océanos. Esto daría más control de forma más intuitiva sobre la forma que debería tener el resultado final a escala macro.

3.2. Resultados

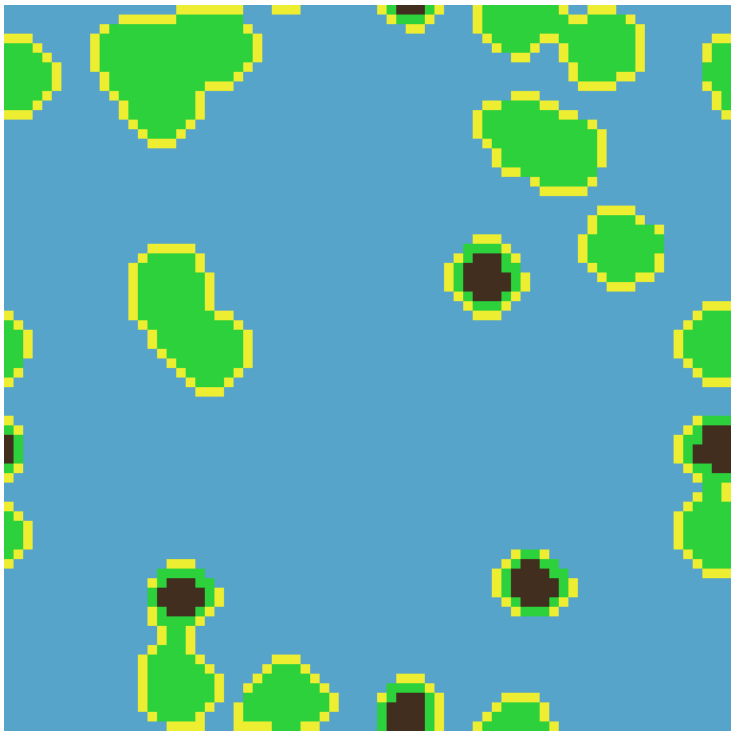

Figura 4: Ejemplo de mapa de archipiélagos generado con el modelo mejorado

Para este modelo se ha aumentado la rejilla estándar a 75x75, para acomodar la nueva funcionalidad de semillas, e igual que el anterior, se han hecho del orden de cientos de ejecuciones. A diferencia de este último, se han conseguido encontrar varias configuraciones de parámetros que resultan en mapas aceptables.

Los cambios introducidos en el modelo han resultado en una mejoría considerable en casi todos sus aspectos, solucionando algunos de los problemas más graves e introduciendo la capacidad de generar muchos más tipos de mapas distintos. Aún así, sigue puntuando bajo en las métricas:

- **Flexibilidad: Sí**. El modelo es ahora significativamente más flexible, permitiendo generar mapas de bastantes más tipos distintos más fácil-

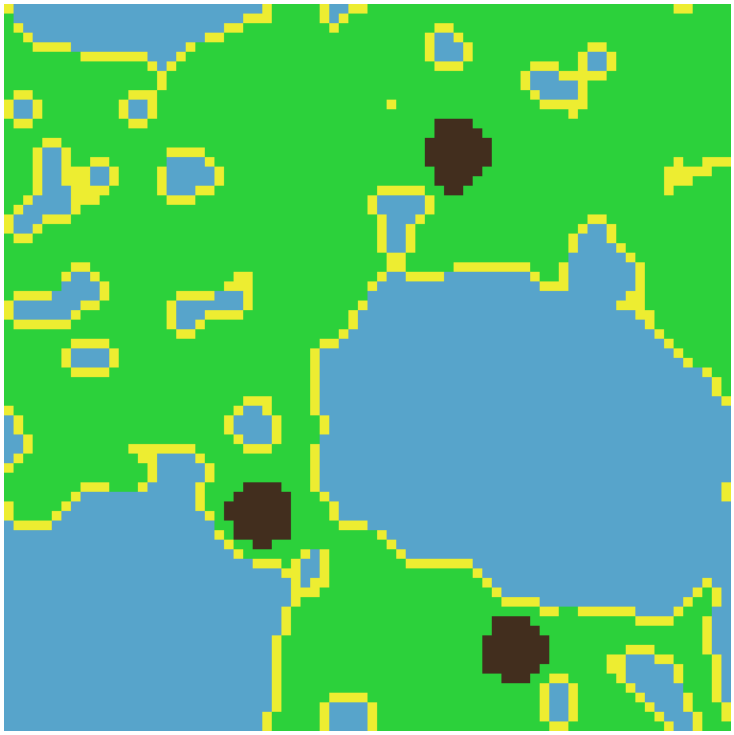

Figura 5: Ejemplo de mapa de lagos generado con el modelo mejorado

mente; se puede comprobar como los resultados de las figuras 4 y 6 son completamente diferentes a pesar de usar el mismo modelo, solamente habiendo cambiado los parámetros. Sin embargo, sigue teniendo el problema de que la jerarquía de estados fuerza el terreno a tener una estructura estrictamente ordenada en base a la altura.

- **Continuidad: No**. La regla de inercia añadida a la evolución ha suavizado el problema de los puntos de corte, haciendo que cambios pequeños en los parámetros de los rangos asignados a cada tipo de *tile* tengan los cambios menos marcados, pero la mejor señal de continuidad está en los parámetros del estado inicial: cambiar la frecuencia de ciertos estados o añadir más o menos semillas a la generación permite ajustar de forma gradual y predecible los resultados finales. Por ejemplo, los mapas de las figuras 4 y 5 tienen prácticamente los mismos parámetros excepto por la probabilidad de aparición de agua en el estado inicial. Sigue, sin embargo, teniendo algunos puntos en los que los parámetros crean cambios demasiado fuertes para modificaciones muy pequeñas en los mismos, por lo que sigue sin cumplir el requisito de continuidad deseado.

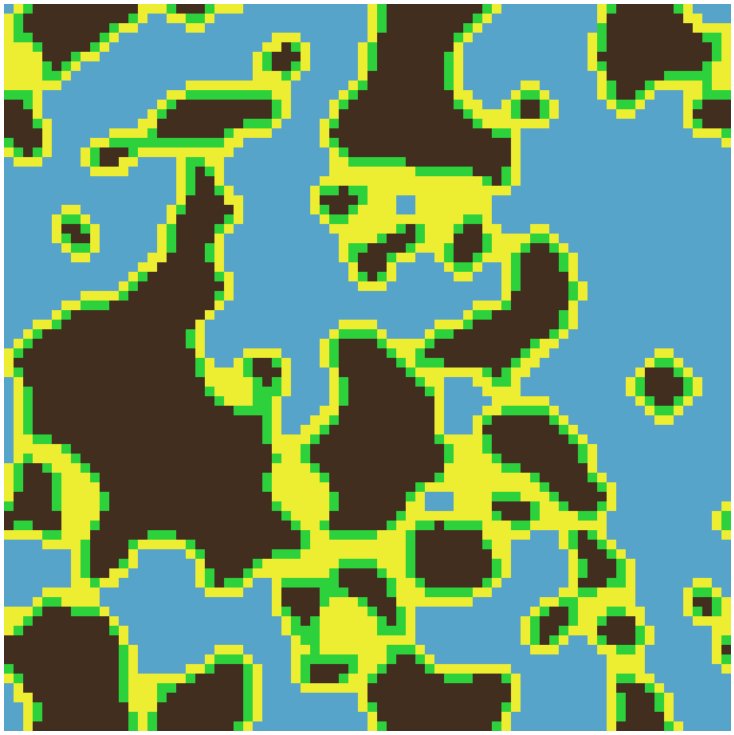

Figura 6: Ejemplo de mapa de archipiélago rocoso generado con el modelo mejorado

- **Usabilidad: No**. Los parámetros del estado inicial y de inercia son intuitivos de usar y tienen un efecto muy predecible, pero los valores de c_{agua}, c_{arena} y c_{tierra} siguen siendo difíciles de ajustar y requiriendo mucho ensayo y error.
- **Estabilidad: Sí**. Al contrario que el anterior, este modelo es mucho más estable debido al parámetro de inercia y el limitar los saltos en cada paso de la evolución.

En los aspectos secundarios hay también algunas cosas que remarcar:

- **Entropía**: El valor de inercia funciona efectivamente como un multiplicador de la entropía. A un valor de cero, no existe inercia y ocurre el problema de convergencia del modelo anterior. A valores bajos (experimentalmente entre 1 y 1.1) se consigue el efecto deseado de evitar la convergencia y crear estructuras más variadas. Por otro lado, aumentar la inercia demasiado crea mapas completamente caóticos o estáticos.
- **Variedad**: Al igual que en el modelo original, es difícil generar distintos tipos de paisaje en el mismo mapa, incluso con la ayuda de las semillas.

Una solución futura a esto pasará por usar diferentes generadores o parámetros sobre el mismo mapa y buscar una manera de generar las transiciones entre las distintas áreas.

3.3. Rendimiento

Prueba	Generaciones	Tamaño	Tiempo
1	50	50x50	49ms
2	50	150x150	485ms
3	500	50x50	461ms

Igual que en el modelo original, se han hecho 20 pruebas y presentado la media de estas. El modelo mejorado tiene el mismo orden de complejidad que el original y un rendimiento muy similar, algo más lento sin embargo por las nuevas comprobaciones añadidas a la función de transición.

3.4. Algoritmo estocástico

Una posible solución considerada para el problema de la usabilidad de este modelo es el de usar algún algoritmo de optimización estocástico para ajustar los parámetros del estado inicial y la función de transición para conseguir el resultado deseado; esto viene inspirado por el hecho de que el modelo requiere grandes cantidades de prueba y error a mano para conseguir resultados aceptables.

El principal problema de optimizar estocásticamente es que se requiere una función que optimizar; para este modelo, se definió una función que tenía en cuenta las proporciones de los *tiles* en el resultado final y cuántos vecinos de cada tipo tenía cada *tile* basado en su tipo, con el objetivo de que variando estos parámetros se podría especificar el resultado deseado: por ejemplo, exigiendo que cada celda de agua tenga al menos otras cinco de agua cerca se podría conseguir mapas con agrupaciones grandes de agua. Esto además tenía la ventaja de ser más intuitivo con respecto a especificar el resultado final deseado que con los parámetros originales del modelo. Para algoritmo estocástico, se eligió usar un algoritmo genético, que como se vio en el artículo [1], ya había funcionado previamente para optimizar las reglas de un autómata celular.

Desgraciadamente, no se consiguieron buenos resultados con las pruebas iniciales, y problemas técnicos dificultaban desarrollo posterior, lo cual combinado con que el modelo estaba alcanzando ya su límite resultó en la decisión de abandonar esta línea de investigación. Sin embargo, la idea de especificar el número de vecinos deseados para cada tipo de *tile* sobrevivió e inspiró el modelo del capítulo 4.

3.5. Conclusiones

Se ha mejorado el modelo de la sección anterior para solucionar algunos de sus problemas más graves. El nuevo modelo es ahora capaz de generar muchos más tipos distintos de mapas y no hacerlos converger tan fácilmente. Este modelo ahora cumple los objetivos de flexibilidad y estabilidad de forma mínima, pero todavía quedan muchos aspectos mejorables y dos objetivos sin alcanzar: continuidad y usabilidad. Sin embargo, ninguna cantidad de arreglos pueden cambiar los problemas de base de la función de transición, por lo que para el siguiente experimento desarrollaremos un modelo nuevo.

4. Modelo de matriz de aceptación

4.1. Definición

Este modelo, a diferencia del anterior, no está basado en otros modelos o técnicas ya existentes, y su principal objetivo con respecto al primero es eliminar el problema de la jerarquía entre los posibles estados, su uso para la evolución y las complicaciones que derivan de ello.

Siendo n el número de estados posibles, para cada posible estado e_i se define un vector v_i de longitud n. En la posición j de v_i se halla el valor de aceptación que tiene el estado e_i para el estado e_j[7], entendiendo "aceptación" como cuánto de permisible o deseable es que el estado e_j esté en el vecindario de e_i. La función de transición suma los vectores v_i asociados a los estados de las celdas del vecindario en el vector v_{suma} y devuelve el índice del máximo valor de v_{suma}, esto es, del estado cuya aceptación sea mayor en el vector suma.

Este modelo efectivamente funciona a base de definir qué *tiles* son aceptables de ubicarse al lado de cuáles, y darle preferencia a los más adecuados en base a la estructura local del área. Por ejemplo, permite especificar que el agua dulce debería estar cerca de la nieve o la hierba, raramente cerca de la tierra y nunca al lado de la arena. Esto permite configuraciones más flexibles y menos restrictivas que las del modelo anterior.

4.2. Resultados

Para este nuevo modelo, hemos cuadruplicado la rejilla estándar a 150x150, debido a que en los experimentos se ha encontrado con que el algoritmo es dado a crear muchas macroestructuras que no se aprecian en un tamaño pequeño. Este es el algoritmo con el cual más hemos experimentado por diferencia, en el orden de doscientas o más ejecuciones, debido a la facilidad para definir nuevas configuraciones con resultados radicalmente distintos. El modelo por tanto ha conseguido excelentes resultados en la experimentación, y altos valores en las métricas:

- **Flexibilidad: Sí**. Este modelo tiene una gran flexibilidad; como se puede observar en las figuras 7-10, se pueden generar paisajes muy distintos en apariencia y estructura solamente cambiando los parámetros.

[7] Generalmente se espera que v_{ij} sea igual que v_{ji}, es decir, que la relación de aceptación entre dos estados sea simétrica

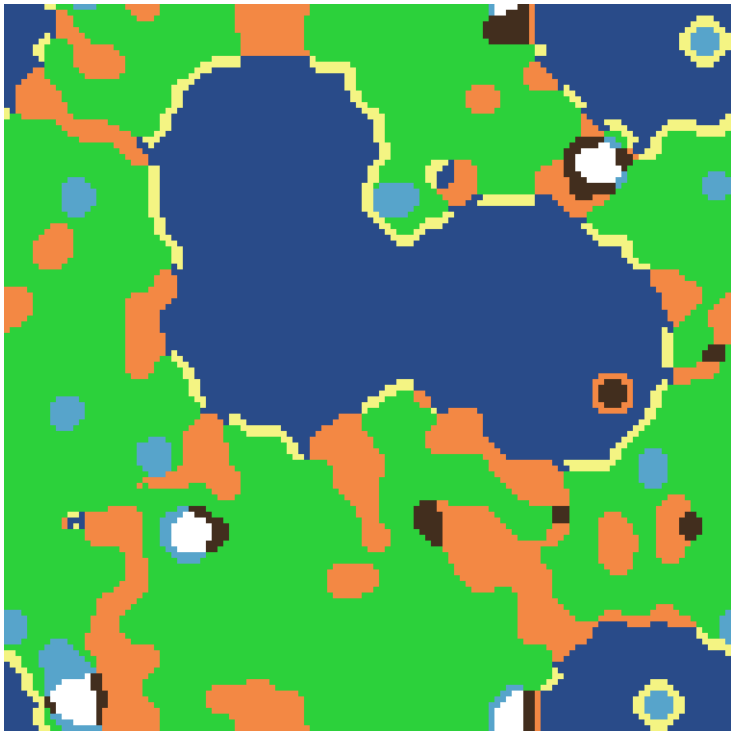

Figura 7: Ejemplo de mapa generado con el modelo de matriz de aceptación

- **Continuidad: Sí**. Modificar los parámetros crea consecuencias lógicas y escaladas con la magnitud de los cambios, no teniendo puntos de corte extremos ni irregularidad en los cambios.
- **Usabilidad: Sí**. El modelo tiene una lógica muy simple con parámetros que impactan de forma muy predecible y controlada en el resultado final, lo que lo hace potencialmente fácil de usar para un usuario no experto. Extender el modelo con nuevos *tiles* es además muy sencillo y no cambia los resultados anteriores; es decir, no tiene dependencia entre parámetros.
- **Estabilidad: Sí**. A menos que se pretenda, los resultados no convergen a estados únicos con la evolución, y genera mapas significativamente distintos con cada nueva ejecución.

Los objetivos principales establecidos quedan cumplidos, pero también hay resultados interesantes en los objetivos secundarios:

- **Granularidad**: Generalmente este modelo solamente es capaz de generar a escala macro por el limitado alcance del vecindario de cada

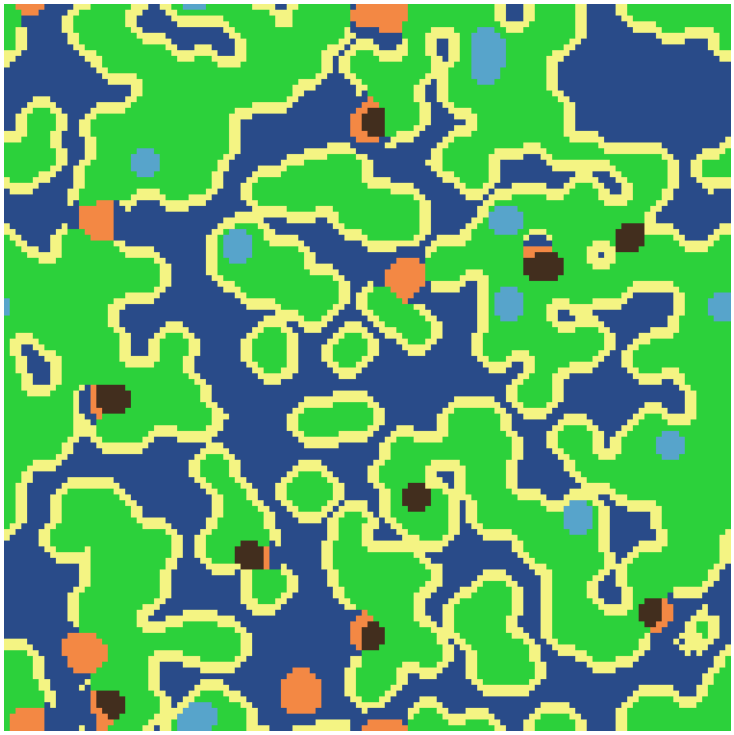

Figura 8: Ejemplo de mapa generado con el modelo de matriz de aceptación

celda. Aumentar el vecindario de las celdas normalmente acaba eliminando los detalles del paisaje dejando solamente las macroestructuras del mapa.

- **Variedad**: Al igual que el anterior, es difícil generar distintos paisajes en el mismo mapa con los mismos parámetros.
- **Entropía**: La entropía de los resultados finales depende principalmente de los valores de la matriz de aceptación, permitiendo ajustarla según las necesidades o preferencias del momento. Darle valores similares de aceptación a todos los *tiles* aumenta el caos, mientras que dar valores en extremos opuestos tiende a ordenar el mapa de forma más rígida.

4.3. Rendimiento

Prueba	Generaciones	Tamaño	Tiempo
1	50	50x50	3139ms
2	50	150x150	30582ms
3	500	50x50	35978ms

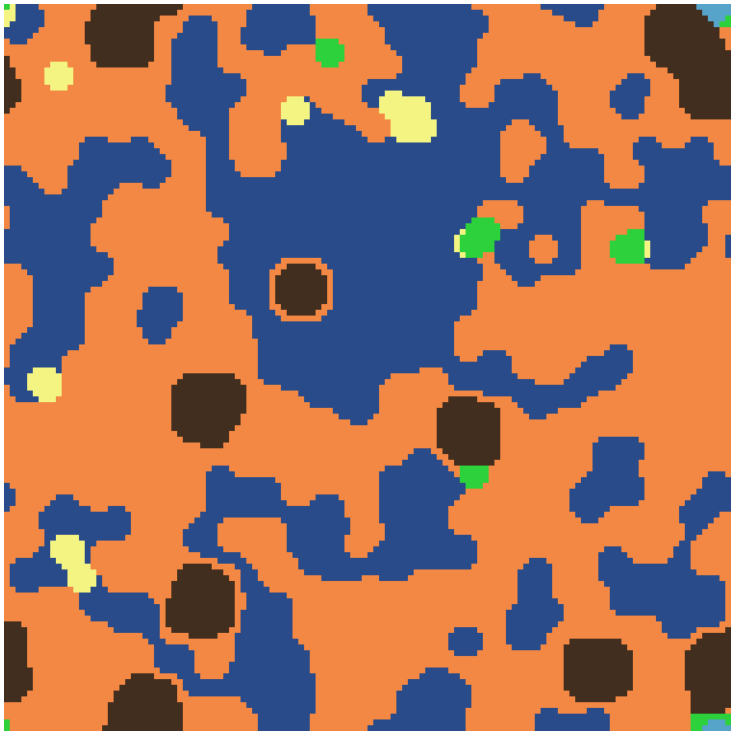

Figura 9: Ejemplo de mapa generado con el modelo de matriz de aceptación

Igual que en el modelo anterior, se han hecho 20 pruebas y presentado aquí la media de los resultados de estas. El orden de complejidad de este modelo es igual al del anterior; sin embargo, el rendimiento en la práctica ha bajado considerablemente debido a que el programa usado, Netlogo, no está especialmente optimizado para algoritmos de esta complejidad y forma. Debido a que los cálculos efectuados son simples operaciones matriciales, una implementación en otro lenguaje podría ser mucho más eficiente usando para ello la GPU u optimizando debidamente las operaciones.

4.4. Fusionado de biomas

Como se ha mencionado previamente, un problema común a los modelos estudiados es la dificultad de generar distintos tipos de paisaje o biomas en un único mapa, y hacer transiciones suaves entre ellos. Esta es una funcionalidad muy necesaria en las aplicaciones reales que ofrecen un mundo continuo en lugar de instanciado[8]. Como añadidura a este modelo, se ofrece un posible

[8]Una posible solución para tener diferentes tipos de biomas es dividirlos en varios niveles o "instancias" discretas con transiciones marcadas para ocultar el cambio súbito de

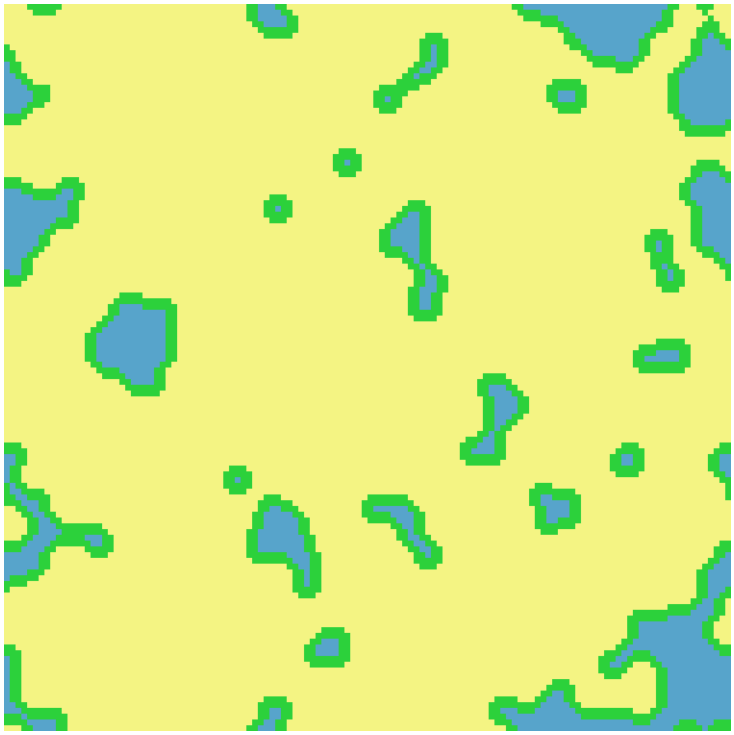

Figura 10: Ejemplo de mapa generado con el modelo de matriz de aceptación

método para solventar este problema.

El primer paso es dividir el mapa en el número de secciones deseadas, aplicando a cada sección los parámetros apropiados para generar el bioma deseado, y evolucionando de forma independiente cada sección, aunque a la hora de elegir el vecindario de las celdas ubicadas en los bordes estas deberían incluir las celdas de las secciones contiguas en lugar de ser un mundo toroide como en el modelo general.

Una vez generados los biomas o secciones, se define qué área entre dos secciones diferentes se considerará el área de transición. Este área incluye una porción cercana al borde de ambas secciones; para secciones cuadradas, este área se definiría como las celdas a una distancia r del borde de la sección, creando un área de transición rectangular. Una vez definida el área, se deberán evolucionar las celdas de la susodicha área usando como matriz de aceptación la suma de las matrices asignadas a las secciones cuya transición se está generando.

Esta técnica tiene la ventaja de ser una variación del modelo base, lo que permite aplicar todas las optimizaciones del primero en esta. Es también una

atmósfera y diseño

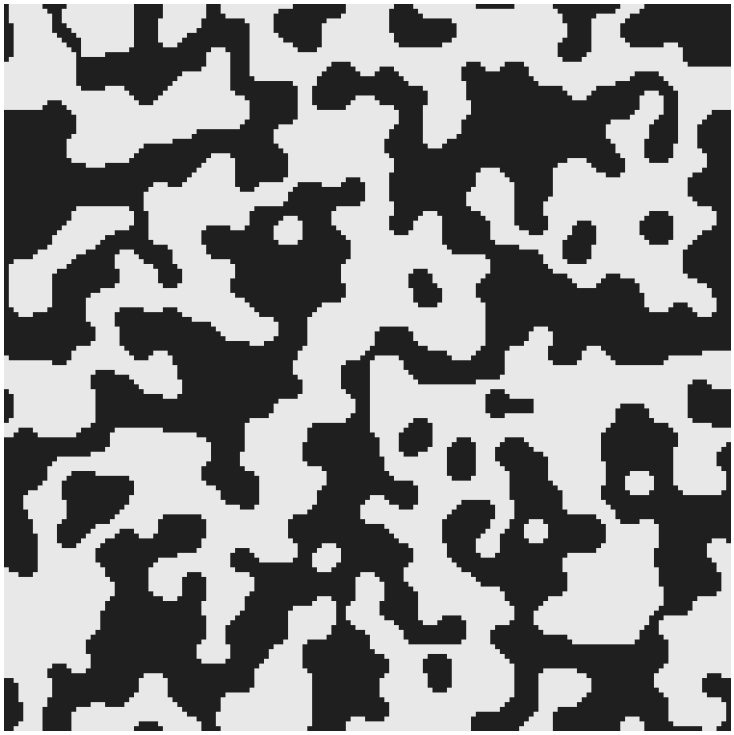

Figura 11: Ejemplo de sistema de cuevas generado con el modelo de matriz de aceptación

técnica que no requiere especial desarrollo para implementarse o diseñarse al basarse sus parámetros en otros previamente definidos. Sin embargo, tiene algunas consideraciones de uso:

- **Escala**: Las escalas de los valores de las matrices de aceptación deben ser similares, o si no una matriz subsumirá a la otra.
- ***Tiles***: Todas las secciones deben tener los mismos tipos de *tiles* o posibles estados de las celdas. Esta limitación no es tan restrictiva como parece pues es posible simplemente poner todos los estados en todas las matrices y no generarlos en los mapas en los que no se deseen.
- **Similitud**: Las secciones a interpolar deben tener cierta similitud en su composición de terreno, o si no las áreas de transición no quedarán naturales. Por ejemplo, puede observarse este fenómeno en las figuras 12 y 13, en las cuales la frontera entre el área nevada y las adyacentes son muy evidentes, mientras que en 14 se puede observar que las fronteras son menos evidentes debido a la similar composición de terreno entre las secciones.

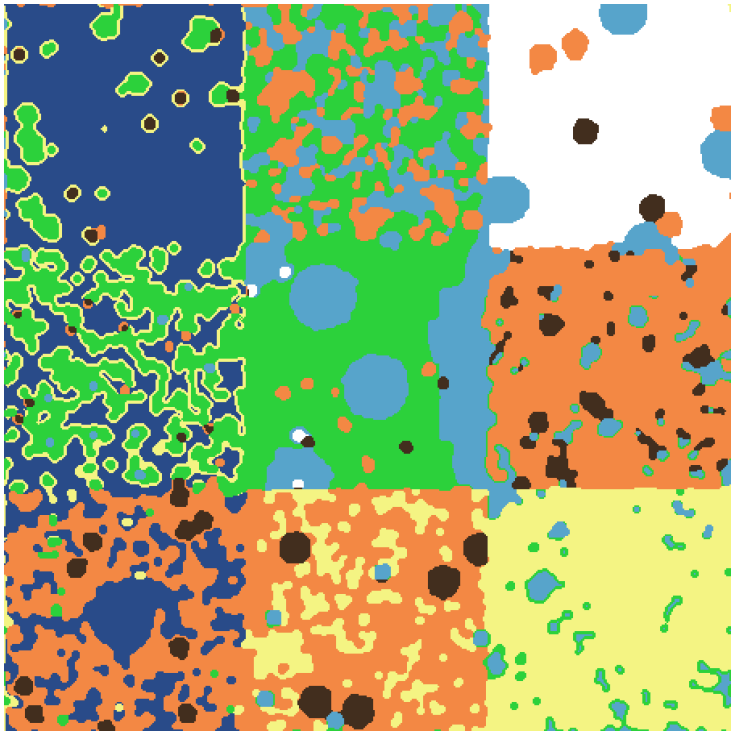

Figura 12: Ejemplo de mapa con fusión de biomas

En general, esta es una técnica muy simple con relativo éxito para hacer transiciones con el modelo basado en matriz de aceptación. Sin embargo, es seguro que mejores técnicas o mejoras sobre esta existen, pero quedan estas fuera del alcance del proyecto.

4.5. 3D

Como un breve experimento para finalizar, se ha implementado este modelo en 3D, cambiando nada excepto el vecindario para incluir la dimensión extra. Se ha hecho esta implementación en Godot 3.2, debido a que es un motor *open-source* muy fácil de usar y apto para prototipos rápidos.

Resultados preliminares muestran varios desafíos difíciles a resolver: debe de poder haber *tiles* vacíos o con aire, los fluidos no se pueden modelar, y es mucho más difícil conseguir terreno con diferencias de altura. Debido a la falta de tiempo no se ha desarrollado mucho este experimento, por lo que queda como posible línea de investigación futura.

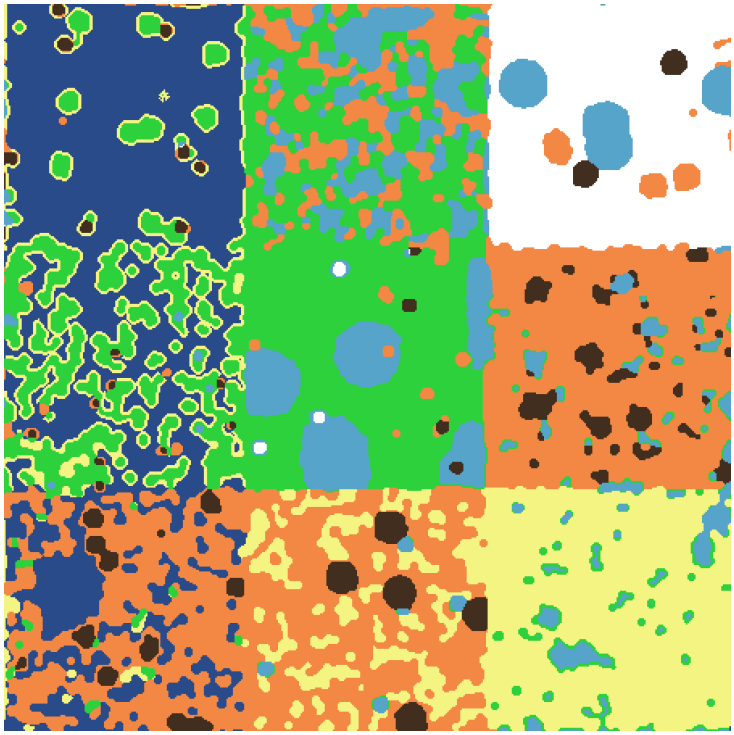

Figura 13: Ejemplo de mapa con fusión de biomas

4.6. Conclusiones

Este nuevo modelo cumple con todos los objetivos marcados por el proyecto. Puede generar mapas de muchas clases distintas, y los parámetros son fáciles de configurar y entender sin comportamientos inesperados. Es además una mejora estricta y significativa del anterior de forma global, especialmente en usabilidad y continuidad, problemas serios del anterior. También permite una solución fácil al problema de la variedad en un mapa a base de usar el mismo principio de la generación para crear transiciones suaves. Debido a que todos los objetivos quedan cumplidos, este modelo es la conclusión final y principal del proyecto.

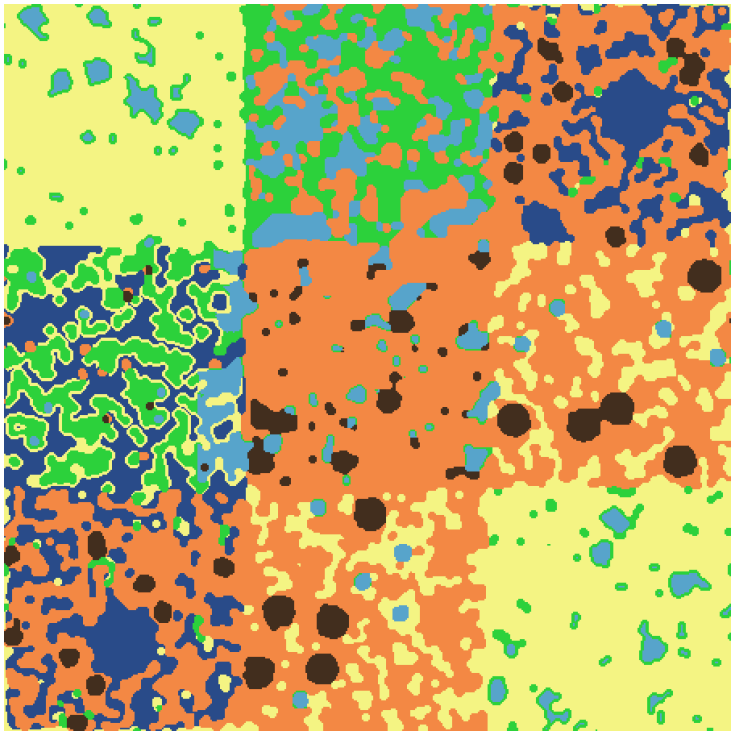

Figura 14: Ejemplo de mapa con fusión de biomas

5. Conclusiones

Hemos experimentado en este proyecto con varios modelos de autómatas celulares y analizado su potencial como técnica alternativa al ruido continuo en el ámbito de la generación procedimental de terreno en 2D. Tras analizar los resultados, hemos desarrollado un modelo generalizable y ampliable de autómata celular enfocado a la generación de mapas, específicamente de terreno terrestre desde vista aérea en 2D. El modelo ha cumplido todos los objetivos especificados y deseados al principio del proyecto: flexibilidad para generar distintos tipos de terreno, usabilidad para ser intuitivo para cualquier tipo de usuario, continuo para no tener cambios súbitos en los resultados y estable para generar resultados diferentes. Este modelo por tanto presenta suficiente flexibilidad y robustez como para ser una alternativa viable en el ámbito de los mapas 2D a los métodos tradicionales basados en ruido continuo, con la ventaja de permitir definir de forma directa los tipos de terreno deseados y las reglas generales para su ubicación y estructura, pero con el inconveniente de generar mapas compuestos por puntos discretos en lugar de un espectro continuo.

A nivel personal, este trabajo nos ha permitido indagar sobre y com-

prender en más profundidad tanto el ámbito de la generación procedimental general, especialmente el ruido Perlin, como los autómatas celulares, ambos campos de interés personal. La definición de las métricas ha sido un desafío especialmente notable, debido a que ha sido necesario abstraerse y valorar qué aspectos de una posible herramienta de esta categoría serían deseables para un usuario potencial, cómo medirlos objetivamente, y cómo medirlos en la práctica, así como el problema de la subjetividad de los resultados. Durante el desarrollo en sí ha sido especialmente estimulante el proceso de identificación de problemas de los modelos, trazando aspectos generales del resultado final de vuelta a idiosincrasias de las reglas del modelo, e ideando posibles modificaciones a estas para mejorarlas.

Un trabajo de esta naturaleza presenta ciertas dificultades a la hora de evaluar los resultados debido a la subjetividad inherente a la hora de juzgar la calidad de un mapa de juego, siendo solamente posible con un investigador y sin sujetos de prueba el probar que el algoritmo funciona adecuadamente, y no que los resultados finales son "interesantes" o "divertidos". Por lo tanto un siguiente paso obvio para esta línea de investigación sería implementar el algoritmo, junto con un algoritmo similar basado en los métodos tradicionales de ruido, en algún juego concreto y evaluar y comparar las impresiones de diversos jugadores. Aún así, esto solamente probaría su calidad o falta de ella para el juego o género de juego concretos, siendo posible que en otro ámbito destacase más.

Otra línea de investigación posible sería la de adaptar el algoritmo a 3D, lo cual presenta complicaciones interesantes, como por ejemplo el manejo de los fluidos, la gravedad, la distribución de aire y el tamaño y forma del vecindario a considerar. También queda pendiente investigar el adaptarlo a 2D en vista lateral, lo cual introduce similares complicaciones a 3D.

Referencias

[1] C. Adams and S. Louis. Procedural maze level generation with evolutionary cellular automata. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8, 2017.

[2] The Center for Connected Learning and Computer-Based Modeling. Netlogo. `https://ccl.northwestern.edu/netlogo/`.

[3] Hello Games. No Man's Sky. `https://www.nomanssky.com/`, 2016.

[4] Martin Gardner. MATHEMATICAL GAMES: The fantastic combinations of John Conway's new solitaire game "life". In *Scientific American 223*, pages 120–123, 10 1970.

[5] Stefan Gustavson. Simplex noise demystified. `http://webstaff.itn.liu.se/~stegu/simplexnoise/simplexnoise.pdf`, 2005.

[6] Christian Heintz, Moritz Grunwald, Sarah Edenhofer, Jörg Hähner, and Sebastian von Mammen. The game of flow - cellular automaton-based fluid simulation for realtime interaction. In *Proceedings of the 23rd ACM Symposium on Virtual Reality Software and Technology*, VRST '17, New York, NY, USA, 2017. Association for Computing Machinery.

[7] Lawrence Johnson, Georgios N. Yannakakis, and Julian Togelius. Cellular automata for real-time generation of infinite cave levels. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, PCGames '10, New York, NY, USA, 2010. Association for Computing Machinery.

[8] KdotJPG. Noise! `https://uniblock.tumblr.com/post/97868843242/noise`, 2015.

[9] Sebastian Lague. Procedural landmass generation. `https://www.youtube.com/playlist?list=PLFt_AvWsXl0eBW2EiBtl_sxmDtSgZBxB3`, 2016.

[10] Y. P. A. Macedo and L. Chaimowicz. Improving procedural 2D map generation based on multi-layered cellular automata and Hilbert curves. In *2017 16th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, pages 116–125, 2017.

[11] Slawomir Nikiel. Game-logic simulation based on cellular automata and flocking techniques. In *25th European Conference on Modelling and Simulation, ECMS 2011*, pages 299–303, 06 2011.

[12] Amit Patel. Making maps with noise functions. `https://www.redblobgames.com/maps/terrain-from-noise/`, 2015.

[13] Ken Perlin. Improving noise. *Proceedings of the 29th annual conference on Computer graphics and interactive techniques - SIGGRAPH '02*, 21:681, 07 2002.

[14] Paul Rendell. A Turing Machine in Conway's Game Life. `https://www.ics.uci.edu/~welling/teaching/271fall09/Turing-Machine-Life.pdf`.

[15] Stephen Wolfram. A new kind of science. pages 53–56, 2002.