Memoria del Proyecto de Programación: Tetris

Ignacio Pastor Padilla

1º DAM Semipresencial Grupo A

Planteamiento

En la primera fase del proyecto reflexionamos sobre cómo plantear el juego, la decisión sobre cómo solucionar el diseño de las piezas condicionará todo el diseño posterior del juego por lo que será muy importante plantearse distintos tipos de soluciones y sus posibles dificultades en el tratamiento posterior para el correcto funcionamiento del juego

Planteamiento equivocado

Tras darle bastantes vueltas, pienso que la solución más sencilla pasa por entender la pieza no como una unidad indivisible, sino como una composición de cuatro partes (cuatro cuadritos, de ahora en adelante, tetraedros). De esta forma podremos eliminar una parte de la pieza cunado completemos una línea. El siguiente problema es el tablero, fácil: un tablero que sea un array bidimensional, y en cada posición del tablero colocaré la parte de la pieza que corresponda. Y por último, la caracterización de cada pieza. Tenemos 7 piezas, las cuales pueden rotar, con lo que tenemos 7 piezas en 4 rotaciones distintas (7 x 4). La cuestión ahora, es cómo hacer la pieza; si el tablero es un array bidimensional, parece razonable que cada pieza sea también una matriz, y teniendo cuatro rotaciones de piezas, habrá que crear 28 matrices diferentes, la cosa se va complicando, pero parece asumible. Por último, y para termiar de enfangarme, matrices de enteros. En lugar de recurrir a múltiples objetos, cada pieza será una matriz de enteros de 5 x 5, las partes huecas serán 0, y las sólidas 1. Tal que la “Te” quedaría asíasí:
00000
01110
00100
00000

Desarrollo del Proyecto

Para almacenar todo esto de forma accesible, me decido por un array de 4 dimensiones, en la primera posición estará el número de la pieza, en la segunda el valor de rotación, y en las dos últimas tendremos la matriz de la pieza. Quedando así: int[,,,] piezas = new int[7, 4, 5, 5].
Asignar manualmente un 0 o 1 a cada parte de la pieza se llevará unas cuantas horas, y 900 líneas de código.

Ya voy viendo que el planteamiento no es adecuado, pero la falta de tiempo hace que decida seguir adelante. Asigno a la posición central de cada pieza un número distinto, un 4, de esta forma podré reconocer el centro de la pieza una vez esté dentro del tablero. Además, me doy cuenta de que en el tablero tendré que distinguir entre bordes, bloque de piezas, pieza en juego, y su centro. Soluciono esto asignando diferentes números. Los bordes serán un 1, los bloques de piezas serán un 3, la pieza en juego un 2 y el centro un 4. Cuando meta la pieza en el tablero tendré que cambiar sus 1 por 2, no es difícil, y cualquiera mete mano en el monstruoso array de piezas.

Por lo demás, iré recorriendo el tablero cada vez, muevo los doses según proceda, y examino que allí donde pretende moverse no hay ni unos ni treses. Cuando en un movimiento inferior se detecta un 3 o un 1, habrá hecho contacto, convertiré los 2 y 4 en 3 y meteré una nueva pieza. Añado una pantalla de bienvenida, y una pantalla final que muestra un ranking de las cinco mejores puntuaciones y doy el proyecto por finalizado. Tras una semana de trabajo, y cosa de más de 100 horas (no soy muy rápido, y cada paso que doy genera dos errores que cuesta un buen rato solucionar).

Cuando le enseño el proyecto a la profesora me hace ver el cúmulo de chapuzas en el planteamiento, después de todo, y con todo el ingenio invertido, no es programación orientada a objetos.

Versión 2.0

Toca replantear el proyecto desde su base. Me decido por mantener como pieza fundamental el tetraedro, pero esta vez no será la posición de un array. Será un objeto con propiedades, a saber, dos enteros que será las coordenadas x e y, y un color. La pieza será un array de 4 tetraedros, y de la clase pieza heredarán 7 clases que caracterizarán a cada una de las piezas. Cada rotación no será ya una matriz, serán diferentes valores para cada uno de los tetraedros. Lo que haremos será pasarle a la pieza unos valores x e y, estos serán su centro, y a partir de ese tetraedro central, mediante un algoritmo se recalcularán los valores x e y de cada uno de los otros tres tetraedros. Cada rotación de cada pieza tendrá su cálculo diferenciado.

El tablero será una lista dinámica de tetraedros, donde por defecto añadimos los bordes (conformados por tetraedros de color amarillo), y posteriormente iremos añadiendo las piezas. En un primer momento, cometo otro error: La pieza la meto en el tablero, y una vez más, me olvido de la pieza. Readaptaré los métodos del anterior tetris y recorro el tablero entero, añado a cada tetraedro un booleano “serPieza” y “serCentro”, para detectar qué tetraedos conforman la pieza en movimiento, y cuál de ellos es el centro. Cuando la pieza haga contacto y pase a formar parte del bloque de piezas, simplemente pondré su serPieza a false. Todo movimiento pues pasa por recorrer el tablero y mover todos aquellos tetraedros cuyo serPieza == true.

Esta labor no solo es farragosa, lenta, sino que también es muy delicada de implementar. En otra tutoría con la profesora le enseño el planteamiento nuevo, esta vez orientado a objetos y mucho más serio. Vemos también algunos problemas que aún no había solucionado y me hace ver que sería mucho más sencillo gestionar la pieza desde el objeto pieza, aunque esté dentro del tablero, son punteros al fin y al cabo, es el mismo objeto. Pero llevo cinco días con el nuevo planteamiento, y aunque he dedicado más de 60 horas, soy lento en extremo y queda aún mucho que hacer. Estamos a viernes antes de la semana de exámenes y ya no me da tiempo a entregar el proyecto en junio, además tengo que dedicarme a estudiar los exámenes. Dejaré el proyecto de lado hasta el viernes siguiente.

Cuando retomo el proyecto la cosa avanza rápido, elimino los booleanos de serCentro y serPieza, gestionaré la pieza desde el objeto pieza y tendré que replantear de cero la forma en la que muevo la pieza y compruebo si puede o no moverse (si hay un borde u otra pieza que se lo impide). Diez horas después, todo funciona bastante bien. Solo queda eliminar las piezas, implementar las puntuaciones, reutilizar las pantallas de bienvenida y finales, y añadir al ranking una fecha, apunte también de la profesora durante la tutorización. En un par de horas debería acabar.

20 horas después, y mil bugs mediante, acabo el tetris. He añadido la fecha de la partida al ranking, que será propiamente un objeto que tiene un string (nombre de usuario), int (puntuación) y DateTime(fecha partida). Haré un array de Rankings e iré guardando las mejores puntuaciones.
La forma de gestionar el Ranking la reutilizaré de la versión inicial. Un array de 6 posiciones, de las cuales solo mostraré las cinco mejores. Tras jugar una partida, creamos un objeto nuevo y metemos en la última posición del array, el valor previo. Ordenamos el array en función de la puntuación, y mostramos las 5 primeras posiciones. Guardamos el array ordenado, por lo que siempre la sexta posición será la sexta peor puntuación, podemos eliminarla sin problema.

La opción de Guardar y Cargar partida también ha planteado dificultades inesperadas. Si guardo el tablero, guardo también la pieza en movimiento, pero al cargarlo la pieza ya no es la pieza en movimiento, por lo que se queda congelada en mitad del tablero. Guardo por separado la puntuación, el nivel, el tablero y la pieza, pero ahora el problema es casar la pieza guardada con la pieza que debe estar en movimiento. Finalmente tengo que sacar la pieza en movimiento del tablero antes de guardarla, y aunque en un primer momento guardo la pieza en movimiento por separado; finalmente decido crear una nueva pieza cada vez que se cargue la partida. Decisión tomada ya en terminos de jugabilidad.

Comentar todo el código y limpiar todas las partes comentadas y dejadas de lado por cambios en el planteamiento, junto con el diagrama de clases y preparar la documentación se me llevará otras 15 horas. (Encuentro numerosos vestigios de planteamientos anteriores en el código actual que tengo que depurar mientras hago los comentarios del código).

Otra diferencia entre la primera y última versión será que se ha eliminado el molesto parpadeo de la pantalla. Mientras que antes se imprimía todo continuamente, ahora se imprimirá solo la pieza, o los elementos que sean modificados.

Conclusión

El resultado final es mucho más sólido y manejable. Mientras que en el primer caso era impensable aumentar el tamaño del tablero, hacer cualquier cambio en las piezas, y cualquier pequeño cambio suponía hacer encaje de bolillos. Ahora bastará con añadir una nueva clase, añadir un método o una propiedad. Conseguir por ejemplo, que el tablero aumente de tamaño resulta sencillo en extremo, ya que todo está configurado en torno a variables y objetos. Tan fácil como anchoTablero++.

He hecho pues dos proyectos perfectamente funcionales, pero cuyo funcionamiento interno es radicalmente distinto. El coste de mantenimiento, la versatilidad para introducir modificaciones, y el desarrollo del proyecto se reducen enormemente, incluso se posibilitan. Porque ciertos cambios bien sencillos, supondrían cambiar todo el proyecto en el caso de la primera versión.

En definitiva, creo que he cogido mucha soltura en el manejo de los objetos y control de detalles. Y es reseñable también la distinta mentalidad que he ido desarrollando a la hora de programar. Mientras que al principio el enfoque era más bien cortoplacista, soluciono un problema y ya veré cómo soluciono el siguiente. Al final pensaba continuamente en las implicaciones de cada solución de un problema o diseño de algún método. De esta forma he ido aprendiendo a buscar la forma más óptima de trabajar, consiguiendo que la solución de los distintos problemas sea de dificultad constante, evitando así generar problemas de crecimiento exponencial. Soluciones de circunstancias que en van dificultando las siguientes etapas del programa, hasta que la pelota es tan grande que no hay una forma buena de solucionar el problema.

Un proyecto más grande de lo que estamos acostumbrados ha propiciado ese cambio de mentalidad, y si bien el diagrama de clases es algo que he hecho al final “porque se pide”, sí que he trabajado con unos cuantos papeles siempre delante donde tenía las principales clases reflejadas, qué estaba en cada parte y anotaciones sobre las relaciones entre objetos. En resumen, he tenido la suerte de hacer las cosas mal, sufrirlas, y después hacerlas de forma más consistente, aprendiendo así la potencia de la programación orientada a objetos y de un planteamiento reflexionado, evaluando bien las consecuencias futuras de cada diseño. Y no es que en un primer momento me lanzara a la piscina sin pensar, sino que no era capaz de preveer el crecimiento exponencial de un planteamiento deficiente.