Programación

Práctica final

MINICIV

Diciembre 2015

Índice

1.Breve descripción de los contenidos del documento.

En este documento se hará una explicación detallada de los pasos que se han seguido para la realización del videojuego MiniCiv. El punto de inicio del trabajo comienza con una interfaz gráfica programada con anterioridad y a la cual se debe dar una estética y lógica determinada para la generación del mapa inicial en el que se desarrollará el juego. Por lo tanto la intención de esta memoria es recopilar los puntos claves y/o más problemáticos que se hayan presentado a la hora de realizar la práctica. Para poder ver de forma práctica cómo se ha ido construyendo el juego paso a paso, los textos vendrán acompañados de capturas de pantalla de las diferentes fases de desarrollo del juego.

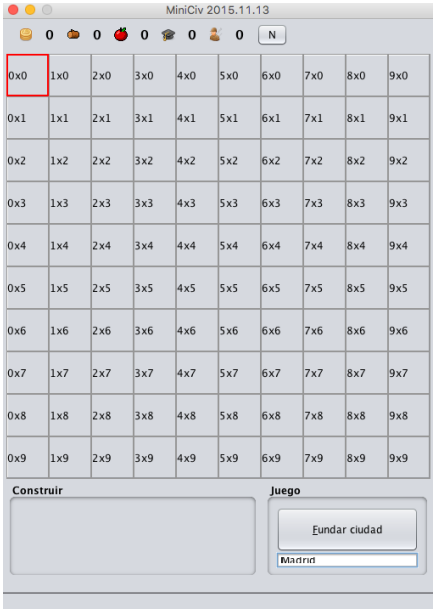
Además de las características obligatorias que se dan en el documento de especificaciones de la práctica final, también se recogen las distintas funcionalidades adicionales que se le han dado al videojuego una vez que se se terminaron de programar las partes básicas de MiniCiv.

Para finalizar se ha añadido a esta memoria un último punto en el que se recopilan las conclusiones acerca del proceso de programación, incluyendo de esta forma aspectos como los siguientes: problemas que se han tenido a la hora de realizar la práctica, comentarios personales y crítica personal del trabajo.

2.Descripción general de los algoritmos más relevantes utilizados

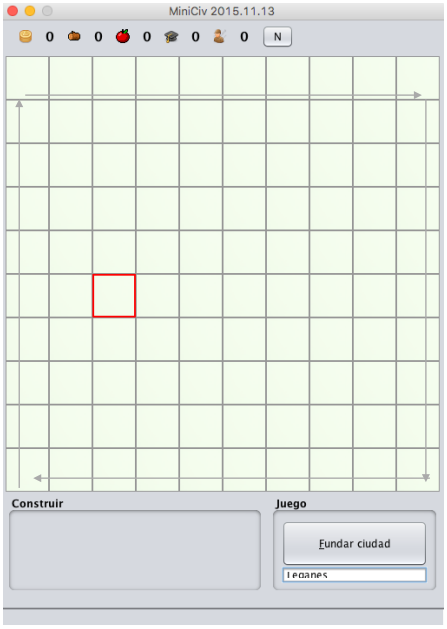
2.1 Generación del mapa:

El mapa se creará en diferentes fases ya que en un inicio, lo que nos ofrece la interfaz gráfica preprogramada es solo una matriz representada con "columna x fila". Este estado del programa es lo que he llamado *fase 0*. El objetivo de este apartado es rellenar todas las casillas del mapa con: Llanuras, bosques mares, montañas, lago y selvas.



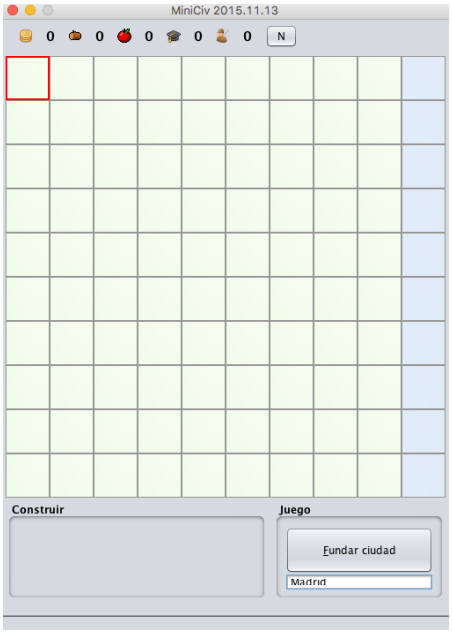
fase 0

En primer lugar, y considerando que la parte mayoritaria del mapa estará ocupada con llanuras se procede a crear un array que almacene datos de tipo Casilla y mediante un bucle for asignarle a su "type" el número ligado a las casillas tipo llanura. Es aquí cuando se ha llegado a la *fase 1* del desarrollo del mapa.



fase 1

Lo siguiente que se hace es empezar con el emplazamiento del mar siguiendo unas ciertas condiciones(todas las casillas de tipo mar no pueden estar separadas, deben estar en un borde del mapa y deben ocupar un 20% del total). Para ello primero se elige de forma aleatoria una de las 4 esquinas(utilizando un switch) y se rellenan todas las casillas de tipo mar en sentido horario por medio de una serie de bucles for(véanse flechas en ilustración de *fase 1*). Con este sencillo algoritmo se ha llegado a la *fase 2*.

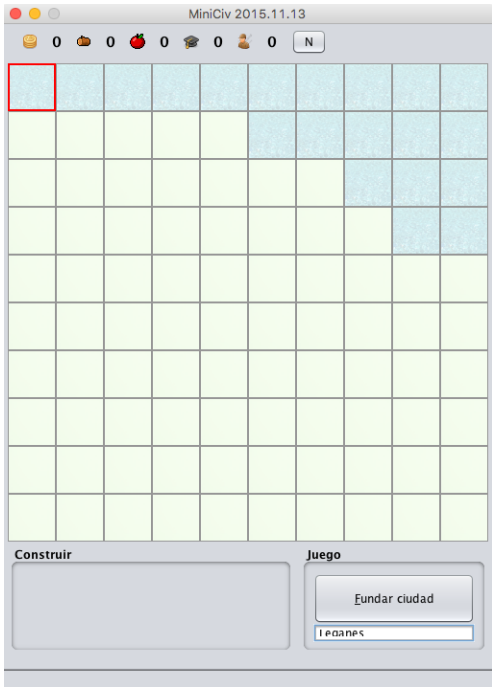


fase 2

Ahora ya se tiene un 10% del 20% del total al que se ha de llegar, se repartirá ese 10% entre una segunda, una tercera fila , y solo si fuese necesario ,una cuarta fila de mar de manera que quede repartido según se observa en la imagen de la *fase 3*, para ello, se toma de referencia la primera fila y se elige al azar uno de sus extremos(superiores, inferiores, derechos o izquierdos), entonces se comienza a rellenar la segunda fila(el número de casillas de tipo mar que habrá en la segunda fila es un número elegido al azar entre 4 y 6). Se procede de la misma forma para la tercera fila empezando también desde el extremo que se

ha elegido al principio(la única diferencia es que en este caso el número de casillas de tipo mar que habrá en la tercera fila será un número elegido al azar entre 2 y 4). Para saber si es necesaria una cuarta fila se hace la siguiente operación—>*20-(10+nºcasillasTipoMar2Fila +nºcasillasTipoMar3Fila)*

Si el resultado de esa suma es mayor a 0 ,se usa dicho resultado para saber el numero de casillas tipo mar que habrá en la cuarta fila. Se ha llegado a la *fase 3*.

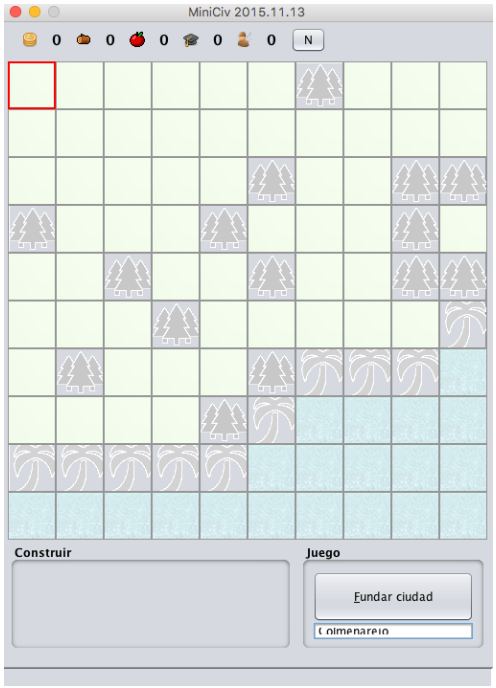


fase 3

A continuación, antes de crear montañas o bosques, se establecerán las casillas de tipo selva ya que sus condiciones de aparición están ligadas solo a la posición del mar(tienen que estar en todas las casillas adyacentes al mar, no se incluye la adyacencia diagonal).

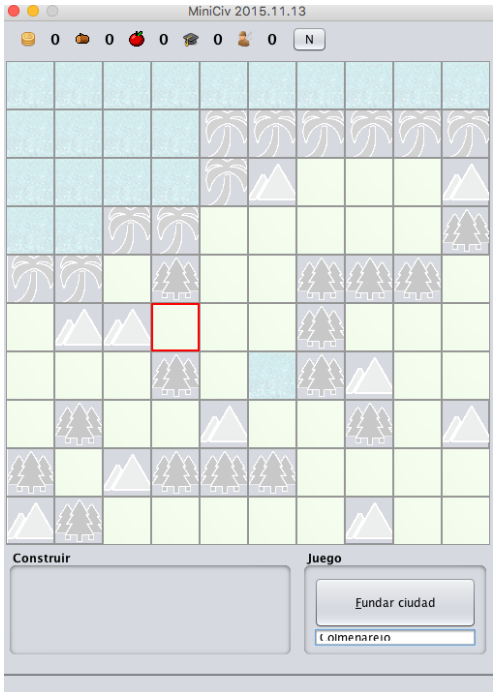
para ello se barre todo el mapa casilla a casilla comprobando primero si es mar o no, si no es mar, la comprobación continua hasta encontrar una casilla tipo mar. Cuando se encuentra una casilla tipo mar se comprueban todas sus casillas adyacentes (arriba, abajo, derecha e izquierda), también para librarnos del error "Out of bounds" que podría aparecer si se trata de una casilla de alguno de los

bordes se usa un try-catch que hará que continue el programa sin lanzar error. El proceso se repite hasta haber convertido finalmente todas las casillas adyacentes al mar. Ahora se procederá a rellenar el mapa de bosques en la una proporción del 15%, se hace de la siguiente forma: dentro de un bucle do-while que aumenta cada vez que se cambia una casilla a tipo bosque se eligen casillas al azar, si es cualquier otro tipo que no sea llanura se vuelve a elegir otra casilla, si no, la casilla elegida se convierte a tipo bosque. Se ha llegado a la *fase 4*.



fase 4

Ahora para poder llegar a la *fase 5,* se han de colocar en el mapa las montañas y el lago, este último aparecerá con una probabilidad del 50%. Para poder llevar a cabo el emplazamiento de las montañas se procede de la misma forma que se hizo con los bosques pero en una proporción del 10%. Por otro lado, que aparezca o no un lago en la generación del mapa dependerá de un número aleatorio que será 0 o 1 con un 50% de probabilidades. Si fuese 0 no habrá lago; en caso contrario, se buscará una única casilla de tipo llanura, cuando se encuentre se cambiará su tipo a mar. La generación del mapa ha concluido en 5 fases.



fase 5

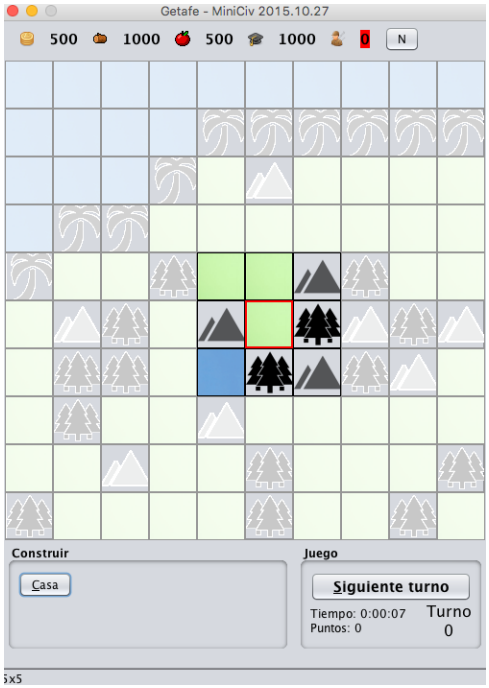
2.2 Desbloqueo del área de juego inicial

Antes de poder construir se deben poder desbloquear las 9 casillas del tablero iniciales a partir de una casilla genérica, (sin tener problemas a la hora de hacer el desbloqueo escogiendo como casilla inicial una esquina), se hace un uso, tengo que decir algo abusivo, de la funcionalidad de java try-catch, pero teniendo en cuenta que el programa no tiene que soportar gran cantidad de datos en memoria, me parecía una solución factible.

2.3 Construir

Ahora que ya hay casillas donde se puede construir, se comprueba que se tengan los recursos necesarios para construirlo y que la casilla seleccionada no esté bloqueada. Si las condiciones son óptimas para construir, el método *canBuild()* devolverá true haciendo que los botones para construir se activen(aunque todavía no funcionen).

Hecho esto solo falta cambiar el tipo de casilla dependiendo de "idConstruction" que determinará el tipo de casilla que se quiere construir. Así se ha llegado a la *fase 6*



fase 6

2.4 Demoler

Finalmente para poder demoler, se tiene que guardar en un array de arrays el tipo original de cada casilla tras crearse el mapa. Para ello primero se crea un método(*originalTypeAsigner()*) que asigna justo después de crearse el mapa todos los valores "*type*" a un array 10x10 para poder luego ser llamado según sus coordenadas.

Hecho esto solo hace falta cambiar el tipo a cada casilla que se vaya a demoler y hacer que se haga de su tipo original.

Ya se puede demoler se ha llegado a la *fase 7*.

3.Descripción de las partes optativas realizadas y de la funcionalidad que el alumno haya decidido incluir en ellas.

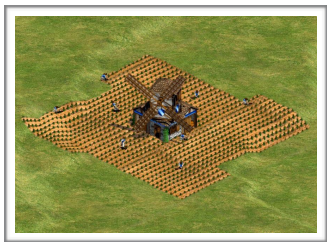
3.1 Cambio de la estética del juego para asemejarlo a la saga “Age of Empires".

A partir del método *getNameIcon()* se cambia la ruta de acceso y se asigna otras imágenes, de la saga Age of empires, tanto edificios como terrenos. Incluso se tiene en cuanta el terreno de cada edificio para dar sensación de continuidad en el mapa. Estas son las imágenes utilizadas para las CONSTRUCCIONES:

Casa

Barco

Aserradero



Granja

Mercado

Mina

Cuartel

Colegio

3.2. Se añade un nuevo edificio en “Personalizado 1”:

El coloso de Rodas, se añadirá a modo de maravilla y una vez construido el jugador habrá ganado la partida, pero costará 7000 de cada recurso material. Se podrá construir solo en terreno llano.

 Coloso de Rodas

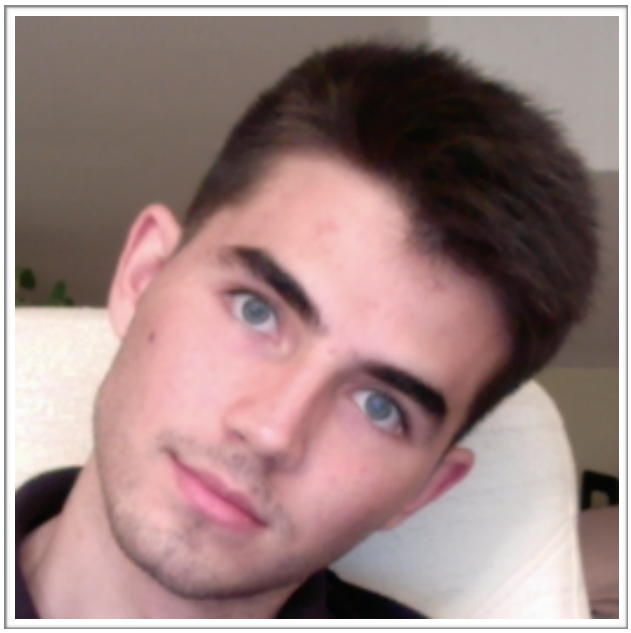
3.3 Se añade un nuevo edificio en “Personalizado 2”:

El Estadio del F.C Barcelona el Camp Nou, tendrá unos costes de construcción no muy altos 300 de madera y 100 oro , sin embargo en cada turno tendrá unos costes de mantenimiento bastante altos (soy del Real Madrid) que llevarán rápidamente a la ciudad a la bancarrota (800 madera, 200 oro).El estadio podrá construirse en los siguientes terrenos: Bosque y Llanura.

Camp Nou

3.4 Se añade un nuevo edificio en “Personalizado 3”(EASTER EGG):

Por último he querido añadir un Easter Egg que solo se activará para un número determinado de recursos, la madera tendrá que ser exactamente 150, el oro 300. Si los recursos son exactamente los indicados aparecerá la cara del programador del juego. Se podrá construir en todas las casillas excepto en selva.



4.Conclusiones finales sobre la práctica.

4.1 conclusiones finales sobre la práctica.

Creo que como primera práctica de programación que entregamos en la Universidad es de bastante buen nivel, sobre todo para personas que no tenían nociones previas de programación, y en especial POO. En mi caso, aunque había aprendido control de flujo en java por mi cuenta, he tenido dificultades para poder entender del todo la POO. No obstante, creo que la lógica del juego donde con lo que hay que jugar son las casillas ayuda bastante a darse cuenta de por qué se usan objetos y cuan útiles pueden llegar a ser.

4.2 descripción de los problemas encontrados a la hora de realizar la práctica

Una de las principales pegas a la hora de implementar los métodos del MiniCiv, sobre todo al principio cuando aún estaba entendiendo su funcionamiento, ha sido sin duda no tener la *main class* ya que desde ella se podría haber visto su funcionamiento más profundo. Sí que tengo que decir que en cuanto se seguían de forma rigurosa las instrucciones de parámetros y valores de salida de los métodos que debían ser implementados muy rara era la vez que en consola se mostraba algún tipo de error.

Y fue precisamente un error lo que me tuvo casi una semana sin poder avanzar. El problema era que yo estaba buscando mi error donde no debía. En consola se mostraba "null pointer exception..."".missing array...", en ese momento

acababa de implementar todos los métodos de la clase player incluido su constructor y por ello creía que tenía que deberse a un error de concepto en la definición del array que almacenaba los recursos. Finalmente pude darme cuenta que efectivamente había un error relacionado con los arrays, lo que no supe hasta ese momento era que el problema estaba en que había dos métodos que habían quedado sin definir su valor de retorno en la clase square(eran los de tipo getCosts...). Yo creo que ese sin duda fue uno de los mayores problemas que he tenido que solucionar, no por su complejidad sino por el tiempo que invertí en poder darme cuenta de qué debía hacer para subsanarlo.

4.3 Comentarios Personales. Opinión personal acerca de la práctica.

Si bien es cierto que ha habido varios momentos de profunda frustración, debo decir que la parte en la que he ido añadiendo la estética de The Age Of Empires y he visto como quedaba ha sido muy gratificante. Después, por otro lado si que tengo que decir que la práctica me ha servido para asentar algo más el material visto en clase, no tanto la parte de control de flujo sino la parte en la que realmente se requiere conocer conceptos de POO.

A modo de crítica me gustaría decir que me hubiese gustado mucho más hacer un juego más gráfico en el que predominase el movimiento del jugador o los elementos de juego sobre la arquitectura estratégica que requiere el MiniCiv.