CODIGO CERO
INFORME SPRINT 1

Iker Fernández, Eneko Rodríguez, María Fernández,
Urko Horas y June Castro

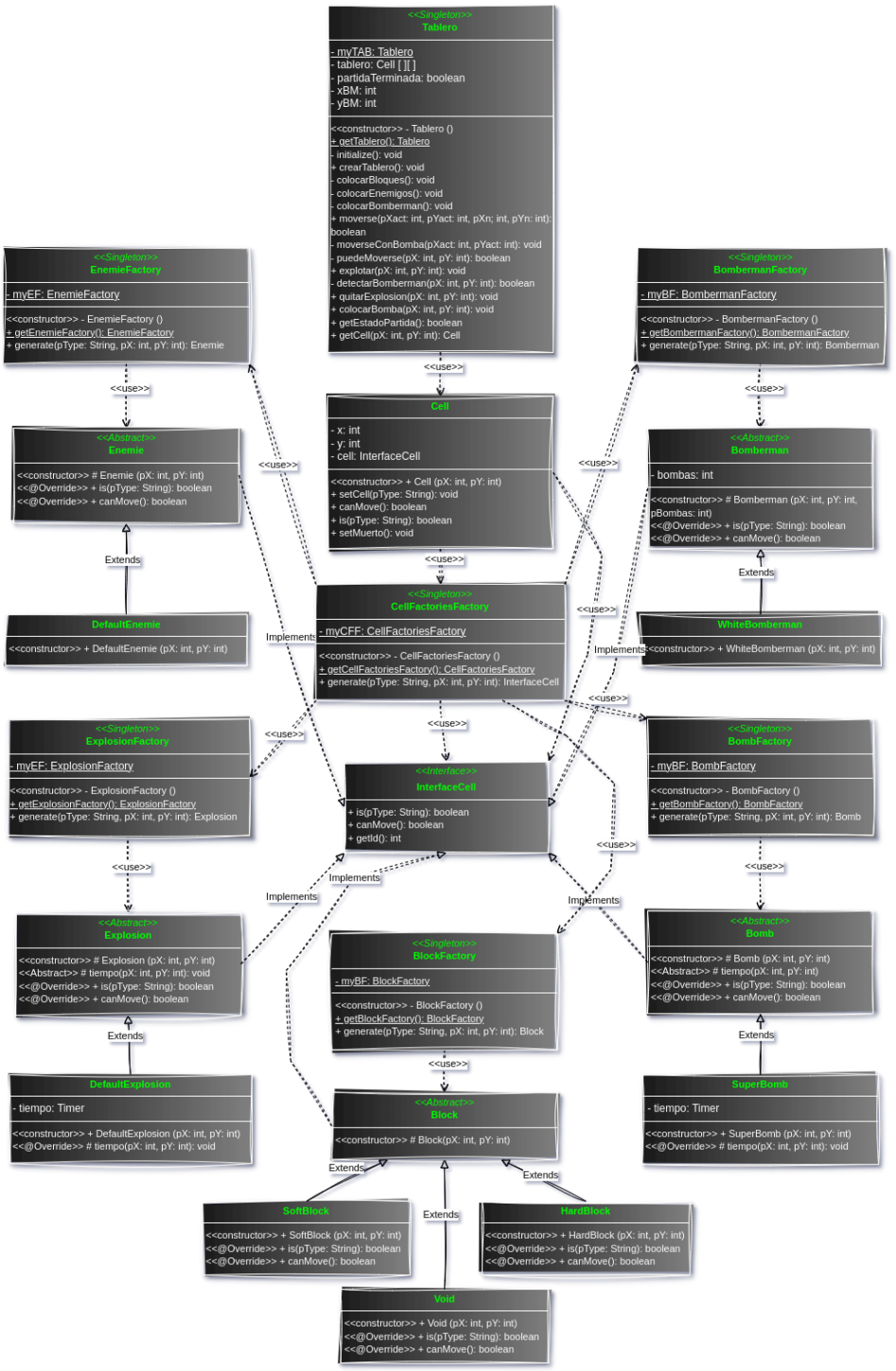
PROFESOR:
Ander Barrena
CURSO: 2

ÍNDICE

1. DIAGRAMA DE CLASES 3
1.1. MODELO 3
1.2. MVC 4
2. PLANIFICACIÓN Y REPARTO DE TAREAS 5
3. INFORME MVC 7
3.1. DESCRIPCIÓN GENERAL 7
3.2. QUÉ HACE EL CONTROLADOR 9
3.2.1. Captura de las entradas del usuario 9
3.2.2 Interpretacion de la tecla presionada 9
3.2.3 Actualización de la vista del juego 9
3.3. NOTIFYOBSERVERS/ UPDATE 10
4. GITHUB 11

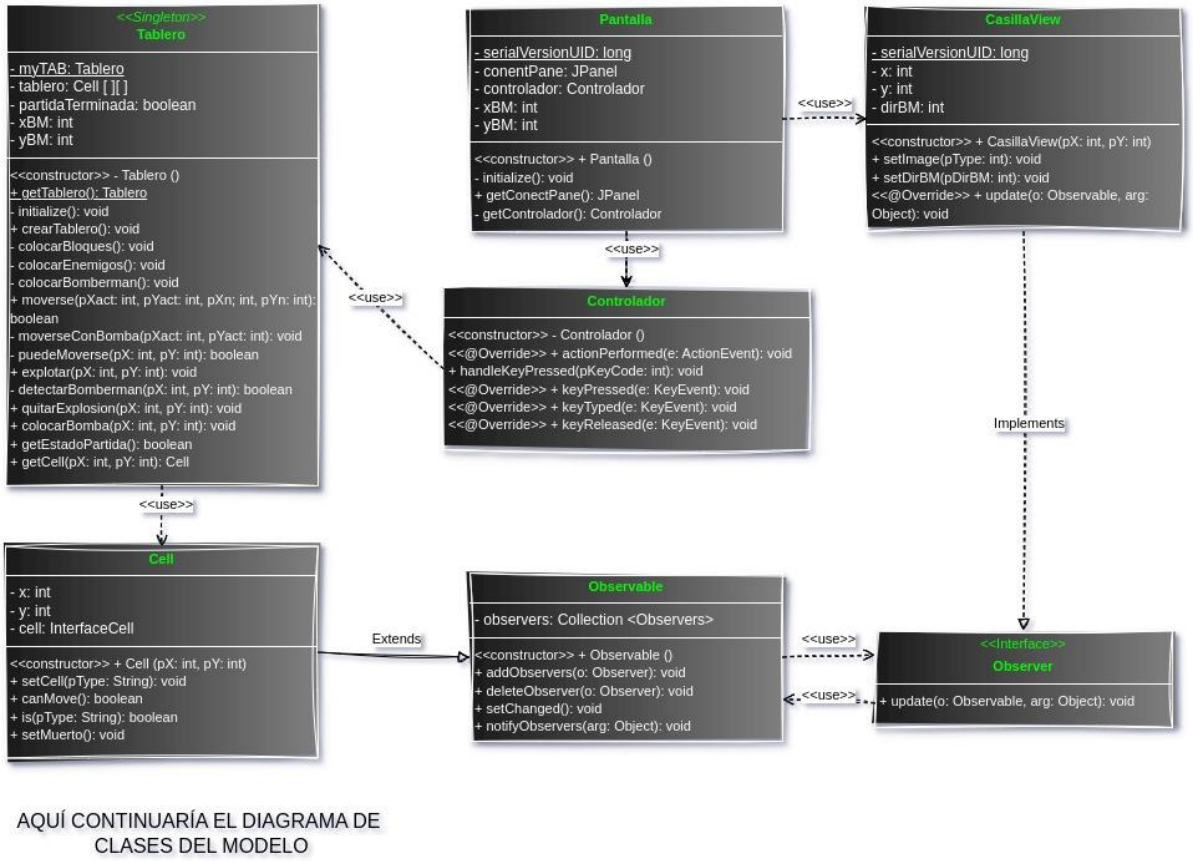
1. DIAGRAMA DE CLASES

1.1. MODELO



Pulsa **AQUÍ** para ver el diagrama en mayor tamaño

1.2. MVC



Pulsa **AQUÍ** para ver el diagrama en mayor tamaño

2. PLANIFICACIÓN Y REPARTO DE TAREAS

Este proyecto para la creación del juego BomberMan ha llevado una planificación previa y una distribución de todas las tareas que son necesarias para la entrega del Sprint 1. En primer lugar, es importante señalar que todos los miembros del grupo han participado en la realización de estas tareas dentro del plazo establecido.

Miembros del grupo:

- **Componente 1:** Iker Fernández
- **Componente 2:** Eneko Rodríguez
- **Componente 3:** María Fernández
- **Componente 4:** Urko Horas
- **Componente 5:** June Castro

Reparto de tareas y Estimación de horas:

- **Creación del código:**
  - *Bomba:* June Castro y María Fernández
  - *Bomberman:* Eneko Rodríguez y Urko Horas
  - *Explosión:* June Castro y María Fernández
  - *Bloques:* Iker Fernández
  - Tablero: Iker Fernández
  - *ViewPantalla:* Todos
  - ViewCasilla: Iker Fernández
  - *InterfaceCell:* Todos

  Para el desarrollo del código, se han utilizado todas las horas disponibles de laboratorio, que corresponden a unas 2 horas lectivas semanales. Además, todo el grupo decidió reunirse 2 veces a la semana para ir avanzar en el trabajo y poner todo el común para poder llegar a la entrega. En resumen, aproximadamente las horas estimadas semanalmente serían entre 4-5 horas.

- **Diseño Diagrama de Clases:** Iker Fernández

  Esta tarea ha conllevado entre 1 y 2 horas ya que se ha comprobado que todos los métodos estuviesen correctamente y se ha realizado de manera manual con la herramienta draw.io.

- **Documento Reparto de Tareas:** María Fernández

  La realización de esta parte del documento ha requerido entre 1-2 horas aproximadamente ya que se han explicado cada tarea con todo detalle.

- **Documento Informe MVC:** Eneko Rodríguez, Urko Horas y June Castro

  La redacción de esta sección se ha realizado entre 2-3 horas ya que se han explicado detalladamente cada apartado para que pueda haber una compresión clara y sencilla.

3. INFORME MVC

3.1. DESCRIPCIÓN GENERAL

El patrón MVC (Modelo-Vista-Controlador) es un patrón de diseño, enfocado en organizar la lógica interna de una aplicación. El MVC organiza el código en los tres componentes de los que se compone su nombre, lo que hace que se facilite el mantenimiento y escalabilidad del código, además de permitir cambiar cualquiera de las capas sin que esta modificación afecte al resto.

1. **Modelo:**
   El modelo representa los datos y la lógica de negocio. Además, se encarga de recuperar y manipular los datos, como gestionar la información y reglas del juego en este caso.

   En nuestro proyecto, el modelo está dentro del paquete "model", y este, a rasgos generales, se encarga de gestionar el estado del tablero (controlando cómo se genera el mapa, cómo se mueve el bomberman, cómo se dan las explosiones y cómo se finaliza la partida), las celdas (patrón observer), los bloques, los enemigos, las bombas y las explosiones, controlando la lógica de cada elemento del juego.

   Un ejemplo es cuando se coloca una bomba, esta espera 3 segundos y luego llama a Tablero.explotar(), que cambia el estado de las celdas afectadas.

2. **Vista:**
   La vista representa la interfaz gráfica que se encarga de mostrarle la información al usuario. No contiene lógica de negocio, solamente la representación gráfica, y su única función es actualizarse cuando el modelo cambia.

   En nuestro proyecto, la vista está dentro del paquete "viewController", que se encarga de la representación gráfica del juego. Hace su función en las clases "CasillaView" (representa las celdas del tablero, y se actualiza mediante el patrón observer) y "Pantalla" (un JFrame donde se visualiza el tablero).

   Un ejemplo es cuando una celda cambia su estado, y se cambia su imagen una vez el modelo ha notificado a CasillaView.

3. **Controlador:**
   El controlador se encarga de la gestión de la interacción entre el usuario, la vista y el modelo, es decir, recibe las entradas del usuario, llama a métodos del modelo para modificar los datos y hace que la vista se actualice.

   En nuestro proyecto, el controlador está en una clase privada llamada "Controlador" dentro de la clase "Pantalla". En este caso se encarga de detectar cuando el usuario pulsa una tecla, y según detecta la acción, llama al modelo (Tablero) que realizará los cambios correspondientes.

   Un ejemplo es cuando el usuario presiona la flecha hacia abajo (que se detecta como (VK_DOWN), y el controlador llama a Tablero.moverse() para actualizar la posición del Bomberman si la celda de debajo es adecuada.

3.2. QUÉ HACE EL CONTROLADOR

El controlador es el encargado de capturar las acciones del usuario, interpretarlo y actualizar la vista del juego dependiendo de esas acciones.

3.2.1. Captura de las entradas del usuario

Gracias a la interfaz *KeyListener* el controlador puede detectar cuando un usuario presiona una tecla. Cuando esto ocurre se activa automáticamente el método ***' keyPressed(KeyEvent e)'*** que recibe un objeto KeyEvent el cual nos da información sobre la tecla pulsada. Este método extrae el código de la tecla (identificador) mediante **e.getKeyCode()** y lo pasa al método **handleKeyPressed(int pKeyCode)**, encargado de procesar la acción asociada.

3.2.2 Interpretación de la tecla presionada

Una vez capturada la entrada, el controlador interpreta qué significa esa tecla en el contexto del juego y decide qué acción debe realizarse. Para esto contamos con el método ***'handleKeyPressed(int pKeyCode)'*** que al pasarle como parámetro el código de la tecla presionada, se encarga de identificar cual es la acción que le corresponde a esa tecla.

Por un lado, si se trata de una tecla de direccional (como VK_UP), el controlador llama al método ***Tablero.moverse()*** que comprueba si es posible mover al jugador en esa dirección, es decir, no hay **obstaculos** ni nada que lo impida, en ese caso, actualiza la posición del jugador en el tablero.

Por otro lado, si la tecla presionada es la barra espacial se llama al método ***Tablero.colocarBomba()*** que se encarga de colocar una bomba en el tablero, justamente donde se encuentra el bomberman.

3.2.3 Actualización de la vista del juego

Gracias al patrón observer se logra una sincronización entre el modelo y la vista. Cada celda de la vista observa su equivalente en el modelo (Cell). Cada vez que el modelo cambia (por ejemplo, un jugador se mueve a su izquierda) la celda notifica automáticamente a su vista asociada y esta se encarga de actualizar la imagen.

3.3. NOTIFYOBSERVERS/ UPDATE

- **Método notifyObservers():**

  Este método se encuentra en el modelo del proyecto y notifica a la vista, que un observador ha detectado un cambio. Antes de llamarlo, se debe llamar al método setChanged() para indicar que ha habido una modificación.

  En nuestro código, se utiliza por ejemplo a la hora de crear una nueva casilla o cambiar el estado de la misma. Cuando un enemigo por ejemplo pasa por x casilla, se actualiza el contenido de esta y para ello, una vez cambiada en el modelo, se ha de notificar a la vista para que se cambie la imagen.

- **Método update():**

  El método update() se encuentra en la vista del proyecto, y es llamado desde el modelo por el método notifyObservers() cuando este detecta un cambio.

  A este método se le pasan el Observador “o” que ha llamado al método, y una serie de argumentos que sirvan para poder actualizar correctamente esta vista.

  Siguiendo el ejemplo del método anterior, al método update() se le pasa el observador y una cadena de integers que contiene información sobre la nueva casilla que se quiere tener en la vista. En este caso se utiliza este número entero para seleccionar la nueva imagen que se colocará en la casilla.

4. GITHUB

El enlace del repositorio de GitHub, donde se encuentra el código, es el siguiente:

https://github.com/ikerfernandezmolano/CODIGO_CERO