Universidad Nacional del Sur
Departamento de Ciencias e Ingeniería de la Computación

Tecnología de Programación (TdP)

Segundo Cuatrimestre 2023

Planilla de Corrección Proyecto 1 ***Comisión 12***

Nota final: A	**Aprobado**

Análisis preliminar del proyecto: *descarga y compilación del código fuente; uso de la aplicación en ejecución.*	
¿Todos los alumnos mostraron actividad en el uso del repositorio?	**SI.** - Se observan commits de consideración de todos los alumnos.
¿El código se compila?	**NO.** - El código se compila, pero no funciona. Múltiples excepciones ante el uso de direcciones de recursos tal como imágenes o archivos que se encuentran incorrectamente resueltas.
¿El juego muestra tiempo y movimientos disponibles, así como los caramelos faltantes para cumplir con el objetivo propuesto?	**SI.** - Se distingue como objetivo, únicamente un tipo de caramelo, cuando sería esperable que el modelado de esta opción sea más general.
¿El juego se comporta como se espera (estado inicial, condición de finalización, transición de niveles)?	**SI.** - En algunas situaciones, la GUI manifiesta que el jugador perdió y ganó al mismo tiempo. Esto, se entiende, resulta en consecuencia de no contar, por ejemplo, con más movimientos pero al mismo tiempo cumplir con los objetivos a partir de ese último movimiento.
¿El juego se comporta como se espera respecto del intercambio de caramelos (comunes, gelatina, glaseados)?	**SI.**
¿El juego se comporta como se espera respecto de la detonación y caída de caramelos?	**NO.** - No fue posible observar las animaciones asociadas a la detonación de caramelos, lo que hace muy compleja la jugabilidad. Es dificultoso entender qué entidades son alcanzadas por una detonación. Las caídas, en apariencia se comportan adecuadamente.
¿El juego se comporta como se espera respecto de la detonación y generación de potenciadores (envueltos y rayados)?	**SI.** - Para los casos observados, en apariencia, se comporta tal como corresponde.
Virtudes de la GUI y usabilidad	**SI.** - El dominio de aplicación es diferente al original. - El juego se presenta con un nivel de detalle adecuado y efectivo. - Los objetivos se observan de forma gráfica.

Debilidades de la GUI y usabilidad	**SI.** - Las detonaciones deberían ser más evidentes.
Errores y/o características que deberían modificarse	**SI.** - Las detonaciones deberían ser evidentes y visibles. - Las imágenes deberían cargarse efectivamente sin importar la ubicación del proyecto en particular.

(sigue debajo)

Análisis preliminar del proyecto: *diagramas reducidos, extendidos y de secuencia.*	
¿Se encuentran accesibles los *diagramas reducidos de clases* de las defensas del 12-14 de septiembre y 26-28 de septiembre?	**SI** Los diagramas que fueron presentados y defendidos se encuentran disponibles, y guardan un nivel adecuado de presentación para el estadío del proyecto.
¿Se encuentran accesibles los *diagramas completos de clases* de las defensas del 12-14 de septiembre y 26-28 de septiembre?	**SI**. Los diagramas que fueron presentados y defendidos se encuentran disponibles, y guardan un nivel adecuado de presentación para el estadío del proyecto.
¿Se encuentran accesibles los *diagramas reducido y completo de clases* de la entrega final?	**SI.**
Errores y/o características que deberían modificarse en los diagramas desde el punto de vista técnico.	**SI.** - No se distinguen las clases abstractas como corresponde. Tampoco se observa en las entidades descendientes la indicación de las operaciones que fueron efectivamente redefinidas. - La cardinalidad y el nombre de los atributos en las relaciones se encuentran en orden inverso. - La relación entre *Entidad* y las interfaces que implementa no se encuentra graficada de forma consistente (implements no extends). - Se define una relación de *agregación* entre *Timer* y *Nivel*, cuando en realidad, lo que se busca modelar es una relación de *composición*.
Errores y/o características que deberían modificarse desde el punto de vista del modelado del problema.	**SI.** - Como se modela una relación de composición entre *Timer* y *Nivel*, en apariencia, se producirá un acoplamiento entre el comportamiento del *Timer*, el control del tiempo que debería ejecutar la lógica del *Juego*, y la notificación sobre la *GUI* para observar el avance del mismo. - En la clase *Juego* se observa declarado el servicio público *getMiGUI*(): teniendo en cuenta que tanto la clase *GUI* como la clase *Juego* encapsulan las cuestiones gráficos y lógicas, respectivamente, resulta cuestionable la necesidad de que *Juego* ofrezca acceso a la *GUI*: ¿por qué ocurre esto? ¿Esto permite evidenciar un error de modelado? ¿Qué clase, a través de *Juego*, debería tener acceso a la *GUI*? ¿Es esto correcto?

(sigue debajo)

Análisis del código fuente: *inspección del código fuente entregado.*	
¿El código fuente se encuentra adecuadamente organizado en paquetes?	**SI.**
¿Existe una adecuada división de responsabilidades entre las clases que implementan cuestiones de lógica/gráfica?	**SI/NO.** - Existe un **acoplamiento inaceptable respecto del manejo de mensajes de transición de niveles** (pasa nivel, pierde o gana el jugador). Bajo ningún aspecto se pueden operar mensajes sobre *JOptionPane* desde la clase *Juego*: ¿para qué está definida la clase *GUI*? ¿Por qué desde el *Juego* se asume que mostrar un mensaje de esta forma se encuentra en línea con las decisiones de visualización que se hayan tomado desde la *GUI*? ¿No es acaso la *GUI* quien debe ofrecer un servicio *mostrarMensaje*()? - Existe acoplamiento entre las responsabilidades de *Timer*/*Juego*/*GUI*. ¿Por qué el *Timer* notifica a la GUI? ¿No es responsabilidad de Juego contabilizar el tiempo y, eventualmente, solicitar se grafique esta información? - Desde la clase *Tablero* y desde la clase *Nivel* se accede en varias oportunidades a la *GUI* para notificar eventos, decisión de diseño cuestionable. Sería adecuado que toda la notificación entre la lógica y la gráfica se canalice mediante las clases *Juego* y *GUI*.
¿Existe un modelado adecuado de las entidades? ¿Se resuelve de forma consistente el modelado de la gelatina?	**SI.** - Sin observaciones.
¿Existe una adecuada resolución de las explosiones? ¿Se resuelven de forma consistente las responsabilidades en cada una de las clases intervinientes?	**SI.** - En ocasiones, se observa que ciertas responsabilidades caen por sobre las *Entidades*, corrompiendo, de cierta forma, el encapsulamiento de la clase *Tablero*. Por ejemplo, sería adecuado que la reubicación de las *Entidades* cuando estas cambian de posición se realicen a partir de un servicio *reubicar*() de la clase *Tablero*, en lugar de tener acceso directo al mismo mediante el servicio *getGrilla*().
¿Existe una adecuada resolución de las caídas? ¿Se resuelven de forma consistente las responsabilidades en cada una de las clases intervinientes?	**SI.** - La solución propuesta es poco eficiente. Al detonar un conjunto de *Entidades* desde el *Tablero*, estas *Entidades* detonadas, pueden ampliar su ámbito de aplicación y detonar *Entidades* adyacentes. Esta decisión de diseño hace que el *Tablero* pierda total control de las *Entidades* detonadas en definitiva. En consecuencia, se programa un régimen de caída de *Entidades* que recorre todo el *tablero* (*ordenarColumnas*()) cuando esto podría haberse evitado. Si el control de las detonaciones las lleva el *Tablero*, se hubiese esperado que las Entidades a detonar puedan indicar si su aplicación, amplia el margen de detonación a otras, y retornar estas otras *Entidades* al *Tablero*. Luego, con este conjunto final de *Entidades* a detonar, es el *Tablero* quien

	ordena esta situación, y aplica las caídas considerando únicamente las ubicaciones que quedaron sin *Entidades*.
Virtudes	- Código compacto, claro, y efectivo en la resolución simplista de los problemas.
Debilidades	- Aunque la resolución de animaciones visualmente es correcta, lo dispuesto no funciona de forma concurrente a nivel de animaciones de igual grado de prioridad. La *CentralDeAnimaciones* secuencializa todas las animaciones recepcionadas luego de un cambio en la *Lógica*, por lo que la concurrencia solo se ejecuta a nivel Hilo Principal (creado a partir de *main*()) y el único hilo que se ejecuta en determinado momento para graficar un intercambio, caída, o destrucción. De hecho, en ciertos hilos el tiempo de *sleep*() es excesivamente pequeño (por ejemplo, en *AnimadorCaida* el sleep es de 1 microsegundo. - Se esperan mejoras relacionadas con las buenas prácticas de *Clean Code:* por ejemplo, trabajar sobre los identificadores de clases, variables, servicios, etc; además, sería adecuado un refactoring respecto de el formateo vertical y horizontal del código fuente (tabs, enters, ubicación de llaves).
Errores y/o características que deberían modificarse	- Resolver problemáticas asociadas con la carga de imágenes. - Resolver problemáticas asociadas a la visualización de las detonaciones. - Resolver el acoplamiento *Lógica*/*Gráfica* para transición de niveles. - Mejoramiento del código fuente aplicando técnicas de *Clean Code*. - Se espera que el código entregado no tenga absolutamente ningún println() incorporado.