UNIVERSIDAD NACIONAL
DE GENERAL SARMIENTO

PROGRAMACIÓN I

Trabajo Práctico Final

Integrantes

APELLIDO Y NOMBRE	LEGAJO	EMAIL
Pereira, Fabián	37.247.692/2015	fabianeze93@gmail.com
Sánchez, Matías Alejandro	38.391.082/2015	mattisanchez94@gmail.com
Tula, Ignacio Mariano	35.226.620/2014	itula@ungs.edu.ar

Versión 2

INTRODUCCIÓN:

Se desarrolló un videojuego inspirado en el antiguo juego “Donkey Kong”. Donde el jugador controla un personaje que debe desplazarse por el nivel hasta llegar a la posición del antagonista del juego, sin que ninguna de las dificultades y obstáculos móviles toquen al personaje.

Para ello se intentó que el apartado gráfico sea lo más fiel posible al juego original. Se han utilizado las herramientas propuestas por los métodos de la clase entorno. Principalmente las que permiten cargar una imagen (que puede ser estática o un gif animado).

Para la representación gráfica de objetos más sencillos se han utilizado las herramientas gráficas de dibujos (principalmente rectángulos y triángulos) que combinados entre ellos en posiciones y colores estratégicos permitieron simular vigas y escaleras.

De manera general, los métodos más importantes calculan las trayectorias y las posiciones de los objetos no controlados por el usuario.

Y en el caso particular del personaje, analizan diferentes escenarios posibles a la hora de permitirle al usuario o no ejecutar la acción requerida (no es posible saltar si se está cayendo por una cornisa, o posterior a un salto).

A su vez se detecta en cada instante de tiempo las posiciones extremas de cada objeto para conocer si hay colisiones que hagan perder al usuario.

DIFICULTADES:

Entre las dificultades que se hallaron fueron las acciones que realizaban acciones variadas a lo largo de un intervalo de tiempo.

Un ejemplo de ellas fué el salto, que en determinado momento se desplaza hacia arriba, hasta cierto punto donde cae. Pero la caída no debería estar limitada por una cantidad de tiempo sino que debería producirse hasta que en algún momento encuentre suelo.

Otra dificultad relacionada con el tiempo fué la producción del sonido que representan los pasos del personaje. El sonido no puede ejecutarse en cada tick, sino se produce un exceso de sonidos indistinguibles.

En todos los casos, fue necesaria la aplicación de una variable numérica que guardara el tick en el cual se produjo la última acción y prohibir a los métodos su ejecución hasta que no se dieran ciertas condiciones, entre ellas que hubiera una distancia temporal razonable.

Otra complejidad añadida fueron las colisiones del personaje con los barriles, donde los puntos de contacto pueden ser múltiples y provenir de una combinación de diferentes lados de los objetos estudiados.

Aquí la dificultad era matemática y eminentemente referida al resultados que arrojaban las diferentes inecuaciones de las distancias entre puntos “x” e “y” de cada extremo de cada objeto, pero a nivel informático realizable con condicionales encadenados o con varias condiciones.

Similar al problema anterior, lo fué la detección de cercanías del personaje con escaleras o los saltos de barriles, pero la complejidad nuevamente viene dada por la combinación de al menos dos puntos “x” por cada objeto y dos puntos “y” por cada objeto para conocer la verdadera cercanía y no obtener un falso positivo.

Un falso positivo de un salto de barril sería que uno de estos se encontrara en la misma posición “x” que el personaje pero en un piso inferior. Evidentemente, aquí no hay salto alguno.

MODIFICACIONES:

Por indicación de los docentes se realizaron las siguientes modificaciones al código fuente:

1. **Revisión completa de las variables de instancia de cada clase. No existe ninguna que no sea “`private`”.**

   En la anterior entrega existía algunas clases donde por error involuntario de este equipo, se omitió indicar que las variables sean privadas.

   De todas formas, ninguna clase accede a las variables de otra de forma directa, siempre que sea necesario existen métodos de obtener y cambiar (getter y setter escritos por el equipo).

2. **Se eliminó la variable `String estado` de la clase Personaje y se utilizó una sola en la clase Juego.**

   A su vez se eliminaron dos variables adicionales que existían en la clase juego: `boolean juegoPerdido y boolean juegoGanado = false;`

   `Ahora se utiliza una sola variable de tipo String para representar tres estados:`

   `String estadoDelJuego;`

   Los tres posibles estados pueden ser: "jugando", "ganado" o "perdido"

Este cambio facilitó que la función `tick()` de la clase juego, fuera fácilmente dividida (mediante condicionales) entre los tres estados, y el código sea más ordenado y explícito.

Es decir que la clase principal maneja el estado del juego. En la versión anterior, dependía de la clase jugador, y luego se informaba a la clase principal, siendo esto más confuso.

3. **Se cambió la variable `char ultima` de la clase Personaje por una variable `boolean miraDerecha`.**

   Este cambio, adicionado con la modificación de la lógica principal del estado y los movimientos del Personaje a la clase Juego, facilitaron que la animación del personaje mire hacia izquierda o derecha con menor código fuente.

   También facilitó la escritura de código, eliminando recurrentes condicionales que preguntaban por izquierda o derecha según el char. Ahora no se consulta con condicionales si mira a derecha o izquierda, la lógica del juego aplica el cambio directamente y se utiliza el valor guardado en la variable.

   Por otro lado, una nueva función `cambiarImagen(String s)` se encarga de cambiar la imagen y aplicar si debe mirar a izquierda o derecha con la ayuda de esta nueva variable.

4. **El Constructor de la clase Personaje no recibe un parámetro de tipo Viga.**

   Ahora el personaje se crea en coordenadas fijas.

```
 // Posición por defecto de spawn
this.posx = 50;
this.posy = 530;
```

5. **Se modificó la lógica principal del programa y ahora se ubica en la función `tick()` de la clase Juego.**

   Esto permitió descargar lógica de tres funciones distribuidas de Personaje que se encargaban de dibujar al mismo según diferentes condiciones. Si bien las funciones no se superponían, ahora el código es más legible y ordenado.

   Las funciones de personaje son para brindar cálculos auxiliares, para obtener datos de las variables de instancia o para setear datos nuevos, y una sola para realizar el dibujo en pantalla.

   Las posibles situaciones que puede realizar un personaje (desplazarse, saltar, caer, moverse por escalera) ahora son analizadas dentro de una sóla función y ya no distribuidas entre tres. Esa sola función es tick() de la clase Juego.

La función dibujar (de Personaje) se ejecuta una sola vez por cada tick y utiliza los parámetros guardados en sus variables de instancias. Las cuales fueron modificadas por los algoritmos nuevos implementados en tick().

6. **El personaje no puede ser controlado por el usuario una vez ganado el juego.**

   Se implementó con la función tick(), en los condicionales que analizan el estado del juego. Cuando el estado es "ganado" ya no se ejecutan los algoritmos que calculan los movimientos del personaje.

   Por otro lado, se mantuvo que el personaje cayera por completo cuando el estado es "perdido".

7. **Todos los randoms que utilizan las clases para tomar decisiones pseudoaleatorias, son variables locales de sus respectivos métodos.**

   La clase Donkey, utiliza randoms para decidir cuando va a lanzar su próximo barril. También se utiliza un random para decidir si el barril seguirá por la viga o caerá por la escalera al encontrarse alguna.

   La clase Escalera, utiliza randoms para decidir en qué posición con respecto al eje "x" va a construir la escalera en la ejecución actual. También para decidir si la escalera adicional que construye va a estar completa o no.

8. **Se eliminó un parámetro no utilizado en el método de instancia `estoyCercaDeEscalera(Escaleras[] escaleras, Viga[] suelos)` y en `pisando(Viga[] suelos)`**

   Se detectó que el método `pisando(Viga[] suelos)` requería un parámetro de tipo Entorno pero el mismo no era utilizado. A su vez el método `estoyCercaDeEscalera(Escaleras[] escaleras, Viga[] suelos)` utilizaba el primero, y por ende requería, el Entorno.

   Se solucionó este error.

9. **Explicación de porqué los objetos de la clase Barril tiene una variable de instancia llamada `saltado` de tipo `boolean`.**

   Se agregó dicha información a la documentación de la clase. También se explica en los comentarios del código fuente.

10. **Eliminación de la clase Mensaje e implementar lo que esta hacía en la clase Juego.**

    Las líneas de código fueron trasladadas a los apartados condicionales que ejecutan

acciones según el estado del juego.

IMPLEMENTACIÓN DE EXTRAS:

Se implementaron las siguientes funcionalidades que el equipo aspira a que sean considerados extras por los docentes:

1. **Sistema de puntuación.**
   Aplica 15 puntos cuando se detecta un correcto salto sobre un barril, pero no permite que un barril sea saltado varias veces. Aplica 100 puntos por ganar la simulación.

2. **Sonidos.**
   Existe un loop de audio en todo el juego. Tres variaciones de audio para los pasos del personaje. Sonido para el salto del personaje. Sonido para el correcto salto del personaje sobre un barril.

3. **Escaleras adicionales.**
   Por cada nivel de viga-suelo con su viga-techo existen 2 escaleras .

   La escalera obligatoria al final de la viga-suelo (las escaleras varían levemente su posición “x” en cada ejecución).

   Y una escalera adicional que también varía (poco más que levemente su posición “x”) y que además según un random puede estar completa o incompleta.

   Es decir que las escaleras varían sus posiciones y además se pueden obtener niveles del juego con escaleras todas completas (poco probable), todas incompletas (relativamente probable) o una combinación de ellas.

4. **Barriles pueden caer por las escaleras**
   Los barriles, cada vez que detectan que pasan por una escalera completa, deciden si continúan su rodamiento por la viga o bien pueden caer por la escalera. Se eligió un nivel relativamente bajo de probabilidades de que esto ocurra para que no sea imposible ascender por las escaleras.

5. **Combinación de apartado gráfico con estrategias gráficas nativas de la herramientas provistas.** (A consideración de los docentes)
   Las vigas son una combinación de un rectángulo sólido con triángulos negros (dando la sensación de estructura metálica) construidos mediante un bucle. No se colocaron manualmente los triángulos, los mismos se construyen mediante iteraciones calculando cantidad y posición de triángulos según el rectángulo (o la el objeto Viga a construir).

   El personaje está representado por animaciones gifs o imágenes estáticas png y la

lógica del programa es relativamente más compleja, no sólo cambia las posiciones “x” e “y”, sino que se encarga de mostrar la imagen adecuada (tipo y sentido) para que el gráfico sea representativo para el usuario.

Los barriles también son imágenes animadas para representar el rodamiento, y sus animaciones coinciden con el movimiento. La lógica implementada también genera cambios en la rotación, de la animación de la caída de un barril por una escalera para dar la sensación de que el mismo da tumbos en la caída.

Las escaleras fueron creadas emulando las vigas pero en vez de triángulos se utilizan rectángulos negros, y la construcción es vertical en vez de horizontal.
Al construir nuevas escaleras en esta segunda versión (las escaleras adicionales) no fue necesario cambiar la forma de la construcción, puesto que el bucle que aplica rectángulos negros funciona correctamente para las variables de instancia de dicha escalera (sea cuales sean, respetando el invariante de representación).