CENTRO DE ENSEÑANZA TÉCNICA Y SUPERIOR

CETYS UNIVERSIDAD®

Escuela de ingeniería

Análisis y Diseño de Sistemas de Información

Reporte de investigación:

Proyecto: Kirby

Presenta:

Miguel A. Vélez Del Callejo	17031
Rodolfo A. Contreras Ramírez	17079

Tijuana, B. C., 22 de Febrero de 2012

Introducción

El proyecto tiene como propósito imitar de manera simple un juego de Kirby. Se busca generar un juego que limite la física de gravedad y solidez de suelo. Se espera también tener un nivel con enemigos, los cuales actuarán como obstáculo para el jugador. Finalmente se debe añadir un medio para que la pantalla “siga” al jugador para que el usuario pueda apreciar el movimiento por todo el nivel.

Requerimientos

- El personaje principal (Kirby) tiene que ser controlado por medio de las felchas del teclado, para que se pueda mover a la izquierda y derecha, y brincar.

- Tener una barra de vida, y al perder todos los puntos de vida, se acabará el juego.

- Tiene que contener enemigos en ese nivel, y cuando Kirby toque a uno de los enemigos, tanto Kirby como los enemigos recibirán daño y bajarán sus puntos de vida.

Diseño

El programa consistirá de diversas clases con trabajos específicos.

1. La clase `GameObject` representa todo objeto visualizable en el juego. Esta debe poseer un campo de Textura reservado para el “sprite”, un rectángulo que represente su posición y dimensiones, y un diccionario de estados [véase punto 5], junto con métodos para verificar la existencia de un estado, agregar y remover.

2. De `GameObject` hereda la clase Character, la cual agrega propiedades específicas a objetos “vivos” (por ejemplo: vida restante, ataque).

3. De `Character` heredará la clase Enemy, que agregará características como la capacidad de ejecutar una inteligencia artificial.

4. Debe haber una clase llena de constantes llamada States que contendrá números enteros que representarán un “estado” en que se puede encontrar un objeto. Por ejemplo: PARADO, BRINCANDO, CAYENDO.

5. Habrá una clase `Stage` para contener una lista de `GameObject` que representarán el terreno, una lista de `Enemy` para representar a los enemigos que habitan el terreno y una posición inicial para que la clase `GameLoader` [ver punto 7] pueda colocar al personaje en el “punto de partida”.

6. Se utilizarán archivos de texto para definir las propiedades del nivel y de los enemigos. La clase `PropertyReader` deberá leer dichos archivos y regresar un diccionario de propiedades.

7. Debe haber una clase llamada `GameLoader` que hará uso de los diccionarios de propiedades generados por `PropertyReader` y así crear los objetos `Stage` y todas sus propiedades requeridas.

8. Deberá haber una clase Llamada `StateManager` que cambie los estados de todo `GameObject` que posea relevancia. Deberá cambiar los estados del jugador dependiendo de los botones presionados, así como determinar si una operación es válida en ciertos momentos (por ejemplo: no se permite brincar si estás cayendo o brincando). También debe manejar la solidez de los `GameObject` que representen el terreno y los estados de los enemigos en pantalla. Finalmente, determinará los estados del “viewport” (lo que se ve en pantalla) para determinar si este debe desplazarse siguiendo al jugador o quedarse quieto.

Convenciones de código:

Class Names: Uppercase (eg: GameObject)

Nombre de variables:

- Propiedades de una Clase: Uppercase (ejemplo: GameObject.Position)
- Constantes: Caps (ejemplo: State.STANDING)
- Parámetros de funciones: Lowercase (ejemplo: public void Move(int dx, dy))
- Nombre de métodos: Uppercase (public void Move)

Uso de llaves:

```
method()
{
    . . .
}
```

Progreso esperado para la primera iteración:

- Tener la estructura de `GameObject` y su descendencia tomando en cuenta el diccionario de estados y el resto de las propiedades a excepción de la inteligencia artificial de la clase `Enemy`.
- Tener un `PropertyReader` funcional y un `GameLoader` que haga uso de dicho `PropertyReader` para cargar el terreno de juego.
- Presentar visualmente el terreno generado.

Progreso logrado para la primera iteración:

- Se generó toda la estructura de `GameObject` y sus descendientes.
- `PropertyReader` y `GameLoader` funcional.
- Se presenta visualmente parte del terreno, lo que cabe en la pantalla.

Progreso esperado para la segunda y última iteración:

- Agregar al personaje al juego y hacer uso de `StateManager` para manejar los estados del personaje y del terreno.
- Agregar a los enemigos al nivel, darles solidez y manejo de estados.
- Permitir el movimiento del viewport conforme al movimiento del personaje.
- Hacer uso de los atributos de CurrentHealth de los personajes para permitir interacción con los enemigos.
- Agregar una representación visual de la salud restante del jugador.
- Detener el juego cuando el jugador pierde toda su salud o sale de la pantalla.

Progreso logrado para la segunda y última iteración:

- Se agregaron archivos de propiedades para los enemigos y se manejan sus estados.
- La pantalla se mueve conforme al movimiento del personaje.
- La salud de personajes y enemigos decrementa con el contacto con otros objetos.
- El juego se detiene cuando el jugador “muere”.
- Se agregó una interfaz `IStateble` para que tanto `GameObject` como Stage puedan tener estados.