DOCUMENTACIÓN TÉCNICA.
VERSIÓN 1.0

CARTAGO, COSTA RICA.
SEPTIEMBRE, 2019.

Tabla de Contenidos

Documentación Técnica DonCE y Kong Jr

Participantes

Nombre	Rol
Angelo Ortiz Vega	Desarrollador.
Jonathan Esquivel Sánchez	Desarrollador.
Iván Solís Ávila	Desarrollador.

Introducción

Donkey Kong Jr. es un juego de plataformas de estilo arcade de 1982 de Nintendo. Apareció por primera vez en salas recreativas y luego se lanzó para una variedad de plataformas, especialmente el Sistema de Entretenimiento Nintendo. En el transcurso de la década de 1980, también se lanzó para varios sistemas de consola, con la forma del título abreviado como Donkey Kong Jr. en la mayoría de las versiones. Donkey Kong Junior (DK Jr.), está tratando de rescatar a su padre Donkey Kong, quien ha sido encarcelado. La jaula de Donkey Kong está custodiada por Mario, en su única aparición como antagonista en un videojuego. Este juego es la secuela del videojuego Donkey Kong, que presentaba a Mario como protagonista y al padre de Junior como antagonista.

DonCE y Kong Jr. corresponde al Proyecto III para el curso de Lenguajes, Compiladores e Intérpretes. (CE3104), Módulo Lenguajes. El mismo consiste en la implementación de una aplicación que permita reafirmar el conocimiento de los paradigmas de programación imperativo y orientado a objetos.

Descripción de estructuras de datos desarrolladas

Listas:

1. fruitList: lista de frutas del juego.
2. cocrodileList: lista de cocodrilos del juego.
3. terrainList: lista de terrenos del juego.

Structs:

1. DKJr: datos del jugador
2. Fruit: datos de cada fruta
3. Crocodile: datos de cada cocodrilo
4. Game: datos del juego
5. Lists: struct que se inicializa al inicio del programa, de aquí se obtienen los datos básicos de listas.

HashMap

1. vinesCoords: la llave es el número de liana y su valor son las coordenadas

Descripción detallada de los algoritmos desarrollados

Algoritmo de comunicación:

Las funciones que se utilizan son:

- ServerSockets: esta clase implementa Runnable, lo cual la convierte en una clase que puede correr en un hilo diferente a la ejecución principal. Esta se encarga de controlar el flujo de información del lado del servidor.
- send/listen: estas dos funciones se encargan de controlar el flujo de información del lado del cliente: Luego de que el servidor esté preparado para comenzar el juego, al comenzar, este no envía nada hasta que haya algún cambio inicial (ej: que obtenga una fruta). Luego del cambio, el flujo de información comienza a moverse, cada cierto tiempo se envía un json para actualizar el servidor y este responde con el json conteniendo todas las entidades nuevas que ha creado el servidor y sus datos para manejar en cliente. Toda la información en ambos se maneja con el algoritmo de parseo de json.

Algoritmo de parseo de json:

Las funciones que se utilizan son:

- updateEntities: Tiene como entrada un char* del jason y su función es igualar los valores recibidos por el servidor a los valores locales, ya sean cocodrilos o frutas.
- updateJsonFromStruct: Actualiza el json que se maneja en el cliente para enviar al servidor utilizando la struct lists.
- get IntValueInJsonArray: Obtiene un valor int en el array que recibe del json
- getIntValueInJson: Obtiene in valor int del json

- getCrocodileByNumber: Obtiene el valor i del array de cocodrilos que recibe del json
- getFruitByNumber: Obtiene el valor i del array de frutas que recibe del json Este algoritmo utiliza la biblioteca cJSON de DaveGamble para el parseo y creación de json en el cliente y la biblioteca Jackson de FasterXML. En forma general, el json que se envía del servidor al cliente contiene todos los atributos de la clase JsonEntities en el servidor, mientras que el json que envía el cliente al servidor contiene todos los datos de la clase JsonClient en el servidor. Por dentro del programa del cliente se utilizan los datos que provienen del json del servidor para actualizar los datos internos de las diferentes structs y funciones que se utilizan a lo largo del programa. Por otro lado, en el servidor se utiliza el json que proviene del cliente para actualizar el json del servidor y hacer una especie de unión (parecido al merge en github) de ambos jsons, tomando el estado de los cocodrilos y las frutas del cliente y las nuevas entidades creadas en el servidor.

Algoritmo de la colisión del jugador:

- collisionDetect: La presente función se utiliza para detectar las colisiones entre DKJr y las plataformas del juego, se tienen dos tipos de plataformas, las vigas por las cuales camina y los terrenos que se encuentran en la parte inferior de la pantalla. La función toma las posiciones x,y del jugador y verifica que sus posiciones sean iguales a las de las plataformas, posee cuatro grandes validaciones para las cuatro caras de las plataformas: arriba, abajo, derecha, izquierda, si entra en la validación para la parte superior de la plataforma, hace que DKJr deje de caer.
- ObjectCollision: Esta función se utiliza para detectar la colisión entre el jugador y los objetos o entre objetos y otros objetos. Funciona por medio de rectángulos, ya que con el x, y, ancho y alto del cuadro, se realiza un rectángulo y se detecta si un lado de los objetos está adentro del otro. Después dependiendo del objeto, detecta si hay colisión con un objeto con el que interactúan como cocodrilos con lianas o si el jugador está tocando un enemigo o una fruta. Para realizar lo anterior se itera por la lista que el cliente recibe del servidor.

Problemas Conocidos

- Algunas veces la colisión con ciertos objetos pueden causar movimientos impredecibles en el jugador.
- Para que el server pueda iniciar su correcto funcionamiento, debe de tener como mínimo un cocodrilo y una fruta creada.

Problemas Encontrados

- Hubo conflicto con el json que se obtuvo del cliente y el json que se manejaba en el servidor, se arregló agregando un paso más para asegurarse que ambos json estuvieran sincronizados.
- Hubo un problema con las colisiones del jugador ya que el primer intento se realizó haciendo un rectángulo que funcionara como la colisión del jugador y otro rectángulo para cada plataforma en el juego, cuando hubiera colisión en los rectángulos se le restaba o agregaba distancia al jugador pero por varios bugs no funcionó esta forma. Después tomamos la idea de los rectángulos y intentamos colocar al jugador en el borde del rectángulo, pero el jugador no se estaba colocando bien por lo que se descartó esta forma. Por último se usó cada borde de los rectángulos y velocidades para evitar que el jugador se moviera cuando hubiera una colisión.

Planificación y Administración del Proyecto

LISTA DE HISTORIAS DE USUARIO

1. **Servidor**
    a. Como usuario, deseo que el servidor esté implementado en Java utilizando orientación a objetos como paradigma de programación fundamental.
    b. Como usuario, deseo que el servidor tenga interfaz para controlar elementos de aplicación cliente.
    c. Como usuario, deseo crear cocodrilos rojos o azules.
        i. El Cocodrilo Rojo no se cae de la liana, sube y baja por ella.
        ii. El Cocodrilo Azul elige una liana y desciende en forma vertical y cae,
    d. Como usuario, deseo crear frutas que otorguen puntos a Donkey Jr.
        i. Se deben guardar las coordenadas de las frutas para el siguiente nivel.
        ii. Las frutas deben de tener un valor definido y se asigna al crear la fruta.
    e. Como usuario, deseo que se puedan eliminar las frutas, de igual forma que se crean.
    f. Como usuario, deseo que se reinicie el juego después de llegar al final nivel.

g. Como usuario, deseo poder agregar vidas al jugador desde aplicación de servidor.
h. Como usuario, se desea correr dos usuarios al mismo tiempo (Implementar threads)
i. Como usuario, se desea correr 4 espectadores al mismo tiempo y permitirles elegir cual partida desean ver (máximo 2 por juego).

2. **Cliente Jugador**
   a. Como usuario, deseo que el cliente esté implementado en C utilizando programación imperativa como paradigma de programación fundamental.
   b. Como usuario, deseo controlar al personaje principal por medio de las direcciones del teclado y la tecla de espacio para saltar.
   c. Como usuario, deseo que el jugador muera si colisiona con un cocodrilo
   d.
3. **Cliente Observador**
   a. Como usuario, deseo unirme a una partida existente, y actuar como observador

Conclusiones

- Se utilizar rectángulos para manejar la colisión de los objetos y el jugador ya que es más eficiente que utilizar rangos de coordenadas
- Una ventaja del paradigma de programación orientada a objetos es que es mucho más fácil de abstraer y por lo tanto de entender
- Una ventaja de utilizar el lenguaje C y otros con bajo nivel de abstracción es el poder manejar la memoria para agregar a la eficiencia del programa.

Recomendaciones

- Sincronizar el trabajo del socket del cliente y del servidor
- Abrir y cerrar sockets para optimizar la comunicación
- Utilizar hashmaps para relacionar datos sencillos con datos complejos
- Ya que C es un lenguaje con muchísima menos complejidad, actividades como el parseo de json se vuelven más largas y complejas; crear un json más sencillo para comunicarse con el servidor.

Bibliografía

- cJSON, Ultralightweight JSON parser in ANSI C. https://github.com/DaveGamble/cJSON