Tecnológico de Costa Rica

Área de ingeniería en Computadores

Lenguajes compiladores e intérpretes

CE-3104

Documentación Tarea #2

PolePosition CR

Profesor:

Ing. Marco Rivera Meneses

Alumnos:

Abigail Abarca Brenes - 2018117463

José Agustín Venegas Vega - 2018250621

Fabián Ramírez Arrieta – 2018099536

I Semestre 2020

Objetivo General

Desarrollar una aplicación que permita reafirmar el conocimiento de los paradigmas de programación imperativo y orientado a objetos.

Objetivos Específicos

➔ Desarrollar una aplicación en el lenguaje de programación C y java.

➔ Aplicar los conceptos de programación imperativa y orientada a objetos.

➔ Crear y manipular listas como estructuras de datos.

1. Descripción de las estructuras de datos desarrolladas.

- Clase “Car” contiene todos los atributos necesarios para la manipulación grafica de los mismos. Está conformado por una imagen, y tres enteros que representan los impulsos que puede tomar el carro, la velocidad y la distancia de este.
- Clase “Hole” al igual que la clase “Car”, la clase “Hole” contiene atributos necesarios para manejar al mismo a nivel lógico y de interfaz. Este mismo está conformado por una imagen que representa el hoyo, y al mismo tiempo contiene dos números enteros, uno que representa la cantidad de hoyos en la pista, y otro es una “flag” que se encarga de definir cuándo será lanzado un hoyo (ambos son enviados por el servidor).

2. Descripción detallada de los algoritmos desarrollados.

- Método movement: Este método recibe dos parámetros, ambos de la clase “Car”, siendo “clientCar” y “enemyCar”, respectivamente, al mismo tiempo se encarga de identificar qué tecla fue presionada, y posteriormente llamar a una función auxiliar (movementAux), la cual se encarga de realizar el movimiento a nivel gráfico de la imagen respectiva a dicho movimiento.
- Método movementAux: Al igual que el método “movement”, recibe dos instancias de la clase “Car”, clientCar y enemyCar, respectivamente, pero, recibe un tercer parámetro llamado “dir” que es un integer y hace referencia a la dirección del movimiento del carro. Ya que cada carro puede moverse en “x” y en “y” en sentido positivo y negativo. Este método realiza más que todo las restricciones a nivel gráfico de las imágenes, como colisiones, velocidad mínima, y movimiento alrededor de la carretera, todo esto mencionado es realizado mediante 3 condicionales.
- movementEnemies: Mediante parámetros recibidos por el sevidor, se realiza el movimiento de los enemigos sobre la interfaz (carro enemigo y huecos). Entre los carros, se compara la velocidad de los mismos, y se definen la función de alejamiento a utilizar

(sizeChange), de este modo, se creará la ilusión de que el enemigo desaparece hacia atrás si mi velocidad es mayor, o por el contrario, se podría apreciar como el enemigo se adelante y se encoge a profundidad (conforme avanza en carretera), como antes fue mencionado, son comparaciones, que mediante condicionales definen los parámetros del método a utilizar. El enemigo “hole”, o hueco, utiliza un método similar al método “sizeChange”, pero ligeramente distinto, este método es llamado “holeSizeChange” (se utilizan distintos porque no funcionan igual, al menos no del todo), y los parámetros de este son modificados dependiendo del carro escogido por el jugador, y todo aquello recibido por el servidor, la funcionalidad de la verificación que llama a este método es únicamente basada en el número asignado a la variable “flag” (antes mencionada), la cual puede variar entre “1” y “0”.

- Método sizeChange: Este método recibe como parámetros, clientCar, enemyCar, dir (Integer asignado por el servidor), posDir (Integer asignado desde el inicio dependiendo del carro escogido). Este funciona mediante la modificación de los atributos de la imagen del carro enemigo (posición x, posición y, ancho, alto), esta modificación se hace en proporción a la diferencia de la velocidad del mismo con respecto al cliente.
- Método holeSizeChange: Recibe, clientCar, una instancia de la clase “Hole”, una dirección (Integer proporcionado por el servidor), y posDir (Integer definido a la hora de escoger un carro). Al igual que el método “sizeChange”, este método modifica el tamaño y posición del hueco de forma continua, pero con una razón, dirección y lógica distintas.

3. Problemas sin solución

Debido a la lógica, el servidor y la interfaz implementados, resultó muy complicado crear una pista dinámica, con distintas trayectorias, etc, esto debido también al poco tiempo del que se disponía, de modo que se abarcaron todas aquellas partes relevantes del trabajo, y a últimas instancias, la realización de una pista dinámica resultó casi imposible.

4. Plan de Actividades realizadas por estudiante

Fecha	Abigail	Agustín	Fabián
03/06/2020 - 05/06/2020	Mediante WhatsApp se realizará la división de la tarea en partes, para un mejor desarrollo en la elaboración del proyecto. Una vez identificadas las partes, se asignarán a cada integrante un conjunto de tareas a realizar.		
06/06/2020 - 07/06/2020	Se debe investigar acerca de cómo conectar un servidor realizado en C con un cliente realizado en Java mediante sockets.	Se debe investigar sobre qué biblioteca es la más efectiva para el desarrollo de la tarea, al mismo tiempo empezar a hacer pruebas con la misma, solucionar los posibles errores que se presenten, probar componentes a utilizar como la adición de botones, imágenes, gifs, procesos complementarios, switch de ventanas, etc. Una vez realizado, se deberán crear las ventanes tres ventanas de juego (inicial, selección y juego), con sus respectivos componentes gráficos y lógicos de la propia interfaz.	
08/06/2020 - 13/03/2020	Se deberá realizar los sockets tanto el cliente como el servidor, se deberán conectar y preparar para que entre ellos puedan enviar y recibir información durante todo el desarrollo del juego, el servidor debe estar en capacidad de atender a más de un cliente a la vez.	Se deberá trabajar en su totalidad sobre la ventana de juego, crear una pista móvil, sobre la cual se puedan manipular a voluntad los dos carros en juego mediante entradas de teclado. Se deberá implementar las funciones lógicas necesarias (provisionalmente), y una vez sea funcional, dichas funciones o procesos deben ser implementados en el servidor, para que este mismo haga uso de ellos.	
13/06/2020 – 14/06/2020	Adaptar la información que recibe y envía el servidor a la lógica del juego, terminar algunos detalles que hagan falta en la lógica del servidor, el cliente y la interfaz para que todo funcione unificado.		
15/06/2020 - 17/06/2020	En estos momentos el proyecto debe estar funcional, para poder dedicar el tiempo restante en agregar detalles faltantes.		

(Cabe destacar, que, aunque se ve una clara división entre las dos partes relevantes de la tarea, todo se realizó en conjunto, de modo que, en todo

momento, todos los integrantes conocían que sucedía, y cómo sucedía, en ambas partes.)

5. Problemas encontrados

- Los lenguajes de programación C y Java representan los String de maneras diferentes, en C es un arreglo de char que termina con “\0” en cambio en java solo se representa con el string correspondiente por lo que si se envía a un servidor en C un string de java no lo va a poder leer de forma adecuada, se trató de seguir la solución implementada en chuidiang.org, pero esta utilizaba una librería propia de Unix lo que complicaba la implementación de esta solución en Windows, se trató de replicar esta solución con la librería winsock pero fue muy complicado por lo que se optó de pasar el string de java concatenándole “\0” al final de este para que el servidor lo pueda interpretar de la mejor manera y el servidor le envía al cliente un string terminado con “\n” para que el cliente lea la entrada como una línea en java.
- Se tuvo que añadir un módulo al proyecto que ayudara o facilitara la utilización de la biblioteca “javaFX” en el proyecto, ya que de otra forma no se iba a poder disponer de la misma.
- Se tuvo que utilizar una pista completamente estática, ya que no fue posible realizarla de forma dinámica (por los componentes de la tarea hasta el momento desarrollados), este problema no pudo ser solucionado.

6. Conclusiones y Recomendaciones

Recomendaciones

- Al realizar un servidor basado en sockets con el cliente y el servidor desarrollados en diferentes lenguajes de programación se recomienda investigar la representación de los datos en ambos

lenguajes y como poder pasar esos datos de un lenguaje al otro para que ambos los puedan interpretarlos de forma correcta.
- Se recomienda trabajar el servidor a servicio de todo aquello que necesite la interfaz, desde el inicio del desarrollo de esta.
- Se recomienda investigar a fondo las bibliotecas disponibles en el lenguaje sobre el que se realizará la interfaz, esto con el fin de conocer de qué se dispone, qué nos brinda y qué nos facilita la misma a la hora de desarrollar la misma.

Conclusiones

- Una buena interfaz permite una buena implementación de lógica, de modo que la lógica, puede ser tan independiente de la interfaz, que puede ser implementada de manera eficiente en un servidor sin afectar el funcionamiento de la interfaz.
- En un proyecto realizado en grupo, la comunicación, el entendimiento, y el interés sobre lo desarrollado por los demás integrantes del grupo, es esencial para un desarrollo fluido y agradable.

Bibliografía

Stemkoski, L. (19 de 05 de 2015). *envatotuts+*. Obtenido de Introduccion a javaFX para el desarrollo de videojuegos: https://gamedevelopment.tutsplus.com/es/tutorials/introduction-to-javafx-for-game-development--cms-23835