PACEman

Documentación

Ingeniería en Computadores

CE1106-Paradigmas de programación

Grupo 09

Profesor:

Marco Rivera Meneses

Estudiante:

Andrés Molina Redondo | 2020129522

Ignacio Lorenzo Martínez | 2019068171

Luis Alfredo Gonzáles Sánchez | 2021024482

Semestre 2, 2023

Índice

Descripción del uso de las estructuras de datos desarrolladas (Struct-nodos de listas-nodos)

Cliente Paceman

Todas las variables utilizadas en el cliente Paceman son variables de tipo objeto, por no cuál no hay ninguna que utilice variables primitivas, exceptuando ciertos casos donde por la naturaleza de java no permite realizar ciertas operaciones con estas variables de tipo objeto, las cuales son muy pocos casos alrededor de 2 ocasiones. Es importante tener en cuenta esto de que las variables son de tipo objeto para entender cómo funciona el paradigma POO en este proyecto y en caso de tener ciertos problemas al manejar,leer, o sobrescribir datos en estas variables. Algo importante sobre estas variables también es que su funcionalidad no suele alejarse mucho de las variables primitivas en conceptos básicos por lo cual no debería ocasionar muchos problemas al usarlas en vez de las variables primitivas

Se utilizan varias listas para guardar los objetos que son Enemy ya que se crean varios con diferentes atributos, de la misma manera se utiliza una lista para los objetos. La lista enemies es donde se guardan los objetos que son heredados de Enemy, esta lista se utiliza para durante el juego esté corriendo poder mover los enemigos con sus velocidades respectivas y que estén todos en constante movimiento, además de esto también se utiliza esta lista para saber la cantidad de enemigos sabiendo el largo de la lista esto para poder dibujar y redibujar los enemigos durante esten el movimiento, también para verificar que cada uno de los enemigos no se salga del mapa o cuando colisionan con el Jugador.

Servidor PACEMAN: Para el caso del servidor PACEMAN,se utilizan los structs, como estructuras de datos, estos structs son utilizados para 2 diversos casos:

- Almacenaje de la data de los usuarios(guarda los usuarios conectados , el socket del servidor, los sockets de clientes y el número de clientes)
- Almacenaje de los valores de juego, este struct se encarga de almacenar la información importante del juego, cantidad de vidas actuales, puntaje actual,entre otros.

Bajo las condiciones de los casos anteriores , se crearon los structs para un manejo dinámico a dichas variables , ya que durante el tiempo de ejecución necesitan ser accedidas para las diversas consultas del cliente.

Descripción detallada de los algoritmos desarrollados

Creación de enemigos.

El algoritmo que se encarga de crear los enemigos es una función llamada "factory" se encarga de crear un objeto Enemy o Object dependiendo del mensaje recibido. Si el mensaje comienza con "CreateEnemy", se crea un objeto Enemy utilizando la clase EnemyFactory y se agrega a la lista de enemies. Si el mensaje comienza con "CreateObject", se crea un objeto Object utilizando la clase ObjectFactory y se agrega a la lista de objects. En ambos casos, se imprime un mensaje indicando el tipo de objeto que se está creando. Finalmente, se llama al método "repaint" para actualizar el juego.

Posición inicial de los enemigos y restablecimiento de la posición del Jugador.

El algoritmo que se encarga de esto es una función llamada "continueLevel2" que se encarga de inicializar las posiciones y direcciones de los enemigos y el personaje principal en el nivel 2 del juego. En el algoritmo, se establece la posición inicial de cada enemigo en la parte superior de la pantalla y se les asigna una dirección horizontal alternativa. Además, se comprueba si hay enemigos en la lista "enemies" y, si no hay ninguno, se crea un nuevo enemigo utilizando la función "factory" y se agrega a la lista. También se establecen las posiciones iniciales del personaje principal y se reinician las direcciones de movimiento. Finalmente, se establece la variable "dying" en falso. También se establecen las posiciones

iniciales del personaje principal y se reinician las direcciones de movimiento esto debido a que el jugador pierde una vida y murió.

Servidor:

Creación de enemigos y manejo de input de usuario:

Inicialmente el manejo de input se realiza con la funciòn enviarestadojuego() , esta funcion está contenida dentro de un hilo de ejecucion , con el uso de mutex en la misma, esto debido a que esta funcion necesita acceder al struct que contiene los usuarios y sus respectivos sockets para enviar la informacion, mismo struct que la funcion escuchar cliente accede para actualizar datos de juego, el mutex es utilizado para evitar accesos multiples a la estructura de datos y errores al ejecutar.
El algoritmo se encarga de manejar la entrada del usuario y crea enemigos basados en la entrada del usuario. Luego envía los datos a través de un socket utilizando una cadena JSON. El algoritmo está escrito en C y utiliza pthreads para multihilo. Continuamente lee la entrada del usuario, envía la entrada a varios clientes a través de un socket y luego libera el mutex. También incluye manejo de errores para el desbordamiento del búfer de entrada y la limpieza del búfer de entrada. El algoritmo se puede desglosar en los siguientes pasos:

1. Leer la entrada del usuario utilizando scanf.
2. Bloquear el mutex para garantizar la seguridad del hilo.
3. Obtener la longitud del mensaje de entrada y convertirla al orden de bytes de red utilizando htonl.
4. Enviar la longitud del mensaje(crearfantasma o crear objetos) y el mensaje en sí a varios clientes a través de un socket utilizando Escribe_Socket.
5. Liberar el mutex y limpiar el búfer de entrada.

El algoritmo está diseñado para manejar continuamente la entrada del usuario y enviarla a varios clientes a través de un socket, lo que permite la interacción en tiempo real con el entorno del juego.El usuario envia los comandos necesarios como CrearPinky() ,crearFruta(), entre otros , al servidor.
En el cliente los fantasmas son creados bajo los estándares del abstract factory.

Recibimiento de notificación de colisiones por parte del cliente y actualización de juego:

Este algoritmo es bastante complejo , por lo que se explicara paso por paso:
se requirio de las siguientes estructuras y variables globales:

```
pthread_mutex_t mutex;
struct datatouse {
   int socketServidor;
   int socketCliente[6];
   int numeroClientes;
};
```

las cuales representa la data donde se almacenarán los clientes , también es necesario el siguiente struct:

```
struct gameData{
int puntaje;
int puntajeCounter;
int vidas}
```

el struct anterior almacena las variables necesarias para la funcionalidad del juego

Funciones auxiliares:

Las funciones auxiliares implementadas son las siguientes:

- obtiene máximo, que es una función que obtiene el máximo de un arreglo de enteros(utilizada para verificar si el servidor está en su máxima capacidad)
- nuevocliente() , función que acepta a un nuevo cliente y actualiza la lista de clientes total
- compacta claves, elimina en un arreglo las posiciones con un -1 (usada para quitar clientes desconectados del servidor)
- enviarclientes() , funcion que devuelve el estado actualizado del juego a los clientes mediante un json string

Funciones principal: escucharcliente():

De manera general la función escuchar cliente realiza las siguientes acciones:

- Aceptar nuevos clientes.
- Eliminar clientes que han cerrado la conexión.
- Esperar mensajes de los clientes utilizando select.

- Leer mensajes JSON de los clientes, actualizar el estado del juego, y enviar el estado actualizado a los clientes.

Inicialmente la función chequea si existen nuevos usuarios esperando a ser admitidos, si existen y la cantidad de usuarios no supera el máximo, son aceptados, si no , se les niega, luego de que se haya manejado la aceptación de nuevos usuarios, el juego busca eliminar la conexión entre clientes que han cerrado la conexión, por último , la función espera 1 segundo con select, si algún cliente le envía al servidor , ejecuta los procedimientos necesarios para leer el mensaje enviado por el cliente, cabe destacar que el protocolo de comunicación es mediante el uso de jsonString, por lo que la información debe de leerse con un jsonString , obtener los datos, modificar el struct del juego , entre otros.

Posteriormente en el flujo de acciones del servidor, el servidor envía de vuelta el estado actualizado del juego , nuevamente mediante un jsonString.

La estructura del Json String es la siguiente:

[“nombretag1:” valor, “nombretag2:” valor2}

Cuando el servidor genera el jsonString con la información necesaria que requiere el cliente, lo envía con el uso de la funcion auxiliar enviarClientes() , que envía el jsonString por medio del formato RED.

Análisis del Jsonstring:

EL algoritmo analizarInfo() (algoritmo para procesar el jsonString) es una función en lenguaje C llamada analizarInfo, que toma tres parámetros: una cadena de caracteres (el mensaje enviado por el cliente), un entero (valor), y un puntero a la estructura de almacenaje de variables del juego (vida, puntaje) llamada gameData. La función analiza la información recibida a través del jsonString , contiene un tag llamado Clave, y su valor , llamado “valor” , el cual cambia realiza lo contenido en gameData en función de estos valores.

Parámetros de Entrada:

- const char *clave: Una cadena de caracteres que representa el mensaje enviado por el cliente , estas son las “tags” del formato json.
- int valor: Un entero que representa el valor asociado a la clave o tag.

- struct gameData* juego: Un puntero a la estructura gameData que contiene la información sobre el estado del juego.

Acciones Detectadas por el algoritmo:

1. **Colisión con Fantasmas:**
    - Si la clave es "colision1" o "colision2", lo cual indica una colisión con fantasmas.
    - En el caso de "colision1", se decrementa el número de vidas (juego->vidas).
    - En el caso de "colision2", se incrementa el puntaje (juego->puntaje) y se actualiza el contador de puntaje (juego->puntajeCounter). esto debido a que se considera la colision con un fantasma cuando pacman esta empastillado.
    - Se imprime un mensaje en la consola con la clave y el valor recibido.
    - Se verifica si el contador de puntaje supera los 10000, en cuyo caso se reinicia el contador(variable auxiliar para saber cuándo se debe de aumentar el número de vidas) y se incrementa el número de vidas.
2. **Colisión con Puntos:**
    - Si la clave es "colision3" o "fruta", lo cual indica una colisión con puntos o una fruta.En ambos casos, se incrementa el puntaje y se actualiza el contador de puntaje.
    - Se imprime un mensaje en la consola con la clave y el valor recibido.
    - Se verifica si el contador de puntaje supera los 10000, en cuyo caso se reinicia el contador y se incrementa el número de vidas.
3. **Reinicio del Juego:**
    - Si la clave es "reinicio", se reinician todas las estadísticas del juego (puntaje, contador de puntaje, vidas).

Observaciones importantes:

- El algoritmo requirió de la función strcmp para comparar las claves y determinar qué acción tomar en función de la clave recibida.
- La función imprime mensajes en la consola para informar sobre las acciones realizadas, considéralo un indicador de que el mensaje se proceso apropiadamente
- El puntero a la estructura gameData ,puntero que se pasa como argumento permite que la función modifique directamente el estado del juego.

- El algoritmo es parte del funcionamiento del recibimiento de informacion del cliente,ya que con esta funcion se analiza el mensaje y se cambia el estado del juego en base a dicha accion.

Plan de trabajo (Actividades realizadas por estudiante)

Armado del control en arduino, código del arduino y conexión a Java.
Se necesita hacer un control con arduino para poder mover el personaje en la interfaz de Java, además de esto se necesita hacer un código básico para poder los datos del arduino en este caso del joystick y que sean enviados a la aplicación Java a través de la librería 'JSerialcomm', se le hizo una modificación para que funcione en otro hilo para obtener los datos del control de arduino, esto para evitar problemas con otros bucles en el juego
Encargado:Ignacio Lorenzo Martínez
Tiempo estimado: 2 horas

Implementación del Abstract Factory
Se implementará un Abstract Factory para manejar las creaciones de los enemigos y objetos en el juego, esté abstract Factory contará con 2 factory dedicadas, 1 dedicada a los enemigos que tendrá el nombre de EnemyFactory y la otra que está dedicada a los objetos que tendrá el nombre AbstracFactory, El tipo de objeto que crean es Enemy y Object respectivamente, existen varios tipos de enemigos y objetos, todos estos tienen en común que heredan cosas de sus clases iniciales que serían Enemy y Object, y también aplican poliformismo en este caso para cambiar el método de cargar la imagen ya que cada uno de ellos tendrá una imagen para cada uno
tiempo estimado: 5-6 horas
Encargado: Ignacio Lorenzo Martínez

Movimiento del Pacman/Enemigos y colisión con Enemigos/Objetos/Puntos
Se tiene que implementar el movimiento del Pacman a través de ciertas teclas del keyboard esto se manejara a través de una función que lea las teclas presionadas en el teclado y según la tecla que se presione se tiene que mover el pacman de cierta manera y en este caso con diferente animación, en el caso de los enemigos tendrán un movimiento aleatorio pero hay

que agregarles una velocidad a cada tipo de los sub enemigos que son creados por el Abstract Factory.
Adicional a esto se tiene que implementar las colisiones del Pacman con los enemigos, objetos y puntos que estén por el mapa/laberinto, esto para obtener puntos en caso de ser los objetos o puntos del mapa, o en caso de ser una colisión con un enemigo se tiene que restar una vida al jugador y enviarlo a la posición inicial e igual enviar a los enemigos a la posición inicial.
Tiempo estimado: 5-6 horas
Encargado: Ignacio Lorenzo Martínez

Elaboración de la conexión cliente servidor: Se necesita enlazar correctamente el cliente de java con el servidor de C, manejando los protocolos necesarios de comunicación , tanto servidor como cliente deben de tener la capacidad de enviar y recibir mensajes:
tiempo estimado : 1 a 2 semanas , encargado : Luis Alfredo González Sánchez

Manejo de hilos y multicliente: Se necesita establecer la conexión cliente servidor de tal manera de que se puedan recibir mensajes de múltiples clientes, permitir conexión de múltiples clientes y el envio a dichos clientes, además se necesita hilos extra para recibir input de usuario en consola
Tiempo estimado de 1 a 2 días. Encargado : Luis Alfredo González Sánchez

Comunicación y parseo de Json entre los sockets: La comunicación cliente servidor será por medio de Json en su formato Json String , por lo que tanto los clientes como el servidor deben de ser capaces de elaborar un Json string , enviarlo , recibir un Json string y leerlo para realizar las operaciones correspondientes.
Tiempo estimado : 2 días , Encargado : Luis Alfredo González Sánchez.

Problemas encontrados

Problemas con el control del Arduino, al leer el Joystick en X a veces el dato se queda como si el joystick estuviera pegado hacia un lado y se tiene que volver a mover con cierta fuerza para que se arregle este problema, no se encontró una solución clara para este problema debido a que no se sabe con certeza el problema si es desde arduino o en la librería de Jserialcomm a la hora de leer los datos

Problemas al crear un cliente observador : Existen problemas en la creación del cliente observador dada la imposibilidad del manejo de estructuras dinámicas para almacenar la cantidad n de fantasmas actuales en consola, el código fue implementado pero tiene fallas que crashean el pc, por lo que no se añade al proyecto, se estima mal manejo del malloc y free para las estructuras dinámicas.

Conclusiones del proyecto

Los sockets son una manera muy buena de comunicar datos entre Java y C pero hay que tener cuidado al momento de enviar estos datos/mensajes ya que Java los envía/recibe/lee de una manera y C de otra forma por lo cual puede complicarse esto pero sigue siendo una opción muy buena

Los sockets ayudan a mantener una comunicación continua entre el cliente y el servidor para una transferencia de datos continua y sin fallos

-Los hilos sincronizados por la librería mutex permitieron el acceso mùltiple a la estructura de datos denominada datatouse,sincronizando el acceso a los datos al congelar hilos cuando otro hilo estaba en pausa, evitando acceso multiple a una misma instancia, por lo que se considera la libreria mutex como una libreria eficiente en el manejo multiple de hilos en C.

Recomendaciones del proyecto

Manejar los sockets con muchísimo cuidado y en especial los datos enviados en estos, se tiene que aprender cómo funcionan los datos a la hora de ser enviados y de ser recibidos para que cada programa lo pueda leer de manera eficiente y correcta.

Utilizar Linux para el servidor en C ya que hay más ejemplos en la web para este caso ya que C estaba más pensado para sistemas operativos Unix que para windows, además de que algunas librerías utilizadas en videos explicativos o en ejemplos solo funcionan en sistemas operativos en Unix y es difícil encontrar librerías funcionales para todo en Windows utilizando C.

Bibliografía consultada

chuidiang.org. (2021, 10 21). Socket entre C y Java. Retrieved from Socket entre C y java: http://www.chuidiang.org/java/sockets/cpp_java/cpp_java.php

Pac-Man. (2023, 10 18). Pac Man. Retrieved from webpacman: https://www.webpacman.com
Wikipedia. (2023, 10 18). Pac-Man. Retrieved from Wikipedia: https://es.wikipedia.org/wiki/Pac-Man

Java socket multi clients Java Socket Programming - Multiple Clients Chat - YouTube
Pacman in Java Pacman in Java, Programming Tutorial 1/2 - YouTube
Pacman Multiplayer roxxid/Pacman-Multiplayer-Java-Sockets (github.com)

Bitácora

Ignacio Lorenzo Martínez

Jueves 19 de octubre
Se utilizó un código de arduino y un control utilizado en un semestre pasado, se probó que funcionara correctamente el joystick para recibir las coordenadas de X y Y del joystick en el arduino, además de estos se busco librerias de Java para realizar una comunicación serial con la aplicación cliente Java del juego.

Sábado 21 de octubre
Se empezó un código en Java para poder recibir los datos del control del arduino, se recibe los datos del joystick en X y Y actualmente, esto en una clase totalmente apartada de todo el proyecto PACEman esto para probarlo.
Me reuní con el compañero Luis para hablar sobre el proyecto, de lo que necesitamos e ir separando tareas para ir avanzando, además de aclarar detalles del proyecto como los sockets, entre otras cosas.
En la noche se modificó el código que recibe los datos del control del arduino para que sea un hilo y se pueda iniciar desde otro main, esto pensándolo ya que el juego es un bucle y recibir los datos también es un bucle podría haber un problema y quedarse en un bucle indefinido donde no siga el juego, por esto se creo un hilo para recibir los datos del joystick, y un ejemplo de como iniciarlo en otro bucle, funciona correctamente por lo cuál el control estaría conectado con JSerialcomm a Java.

Lunes 23 de octubre

Se empezó a modificar un pacman para las necesidades del proyecto, se inició cambiando todos los tipos de variables ya que se manejaba con variables primitivas y según las instrucciones del proyecto no se pueden utilizar por lo cuál las pase a un tipo de dato más complicado que es tomado también como Objeto para cumplicar con lo pedido en el proyecto de manejar todo como en el paradigma POO.

Jueves 26 de octubre

Se empezó un Abstract Factory que esté encargado de crear los enemigos y los objetos en el juego provisional pacman que tenemos, esto para ir implementando el patrón de diseño y el poliformismo ya que cada enemigo tendrá una imagen diferente y una velocidad diferente y todo lo otro lo heredan de la clase Enemigo principal, por lo cual el poliformismo afectará a la velocidad y la imagen de los sub Enemigos, lo mismo sucede con los objetos pero en este caso al no tener movimiento no se afecta su velocidad si no que simplemente su bonificación de puntos y su imagen.

Sábado 28 de octubre

Se consiguieron las imágenes para cada uno de los enemigos y los objetos, ahora se añadirán al código y se harán pruebas para ver que estén funcionando correctamente la carga de las imágenes y acomodarlas correctamente en el cliente.

Lunes 30 de octubre

Se empezó a añadir el movimiento a los enemigos creados por el abstract factory, ya que inicialmente solo se creaban y no tenían movimiento, se encuentran algunos errores al estar implementando esto pero son solucionados durante el día sin mucho más problema, además de esto se añade a la vez las colisiones de los enemigos con los bordes del mapa y con el jugador, esto para evitar que se salgan de la pantalla visible por el jugador y que cuando colisionan con el jugador el jugador pierda una vida y inicie de 0 el juego (esto sin afectar los puntos actuales en el mapa).

Andrés Molina Redondo

22 de octubre

Se lleva a cabo una reunión con los demás integrantes del grupo para coordinar las tareas que cada uno va a llevar a cabo. En dicha reunión se establece que Ignacio se encarga de el factory, Luis de la conexión cliente-servidor y yo me encargo de la interfaz.

24 de octubre

Se investigan sobre bibliotecas de gui en java. Tentativamente se ve la opción de usar JavaFX o swing. Queda pendiente investigar más sobre las ventajas de cada uno

27 de octubre

Según lo conversado con el grupo de trabajo se tomó la decisión de optar por seguir desarrollando el proyecto en linux ubuntu, debido a que la documentación encontrada como referencia indicaba que los sockets en c son menos complejos de implementar en ubuntu.

30 de octubre

Se hacen pruebas con javaFX, sin embargo, no se logran desarrollar con mucha claridad las interfaces de prueba. Tentativamente se comienza a investigar sobre swing

31 de octubre

Se realizan avances con los mapas utilizando swing con el entorno de desarrollo netbeans ya que tiene una tiene herramientas que facilitan la creación de interfaces de una manera interactiva.

02 de noviembre

Debido a una falta de comunicación se toma la decisión de utilizar una gui desarrollada por el compañero Ignacio debido a que ya tiene más componentes integrados como la creación de los fantasmas y movimientos, creación de puntos entre otros aspectos.

04 de noviembre

Se intenta ayudar a los compañeros con problemas en cuanto al traspaso de información con los sockets.

06 de noviembre

Se ayuda al compañero Luis con la creación de algunos métodos los cuales son los encargados de hacer los llamados respectivos al abstract factory implementado.

Luis Alfredo González Sánchez

22 de octubre

Se inicia con la construcción de la aplicación, se realizó una reunión por lo que nacho realizará el factory y yo la conexión cliente servidor y Andrés todo lo relacionado con interfaz. Inicialmente se decidió hacer los sockets en windows, se investigará más a fondo

24 de octubre

Se encontró la documentación respecto a los sockets de c en windows el problema es que la documentación no es muy clara por lo que provoca confusiones, no existen referencias para crear una aplicación cliente servidor, por lo que se seguirá trabajando pero se estima que se va a cancelar el desarrollo la creación en windows.

26 de octubre

se descarta por completo la implementación en windows y se procede con el desarrollo del servidor en linux, esto debido a la poca ejemplificaciones de sockets en windows y documentación confusa, se creó una pequeña implementación básica del cliente con hilos en java para enviar y recibir mensajes, queda pendiente crear sockets en c

30 de octubre

dada la ardua investigación se logró encontrar una aplicación de referencia, se procede a crear los sockets en base a esa referencia, se debe considerar que para pasar informacion de java y viceversa primero se debe enviar el largo de lo que se envia, lo que se envía y la cadena de cierre (/0). Se procederá a investigar más al respecto.

01 de noviembre

Se determinó que se debe mandar el largo de los mensajes en formato red, en base al proyecto de referencia que utiliza sockets, se desconoce la utilidad del formato pero es necesario para que c y java puedan leer apropiadamente

03 de noviembre

se logró implementar el recibimiento y envío de datos, sin embargo, bajo lo discutido en el grupo se cometió un error, al parecer el usuario administrador escribe desde el servidor no desde el cliente, por lo que, el servidor debe tener la capacidad de enviar datos cuando el cliente le envíe datos del estado del juego (actualizar el cliente) y poder crear enemigos con el factory en cualquier momento que el usuario administrador lo deseé. Se espera no tener que utilizar hilos, por lo que se tratará de hacer una implementación sin hilos.

05 de noviembre

La implementación sin hilos no fue posible dado que el recibir inputs de usuario desde consola bloquea todo el hilo de ejecución de la aplicación, se debe usar hilos definitivamente. Se ha de considerar entonces un hilo para escuchar del usuario, otro para escuchar el input del usuario administrador y poder enviarlo al usuario, ambos hilos van acceder a la lista de

usuarios, por lo que los hilos deben estar sincronizados para no interferir con el acceso de datos, es decir, que no accedan a la estructura al mismo tiempo 06 de noviembre :bajo malas horas de sueño se logró implementar la aplicación, se considera necesario el uso de json para enviar datos entre el cliente y servidor para no estar pasando de uno en uno. El cliente ha de tener un jsonhandler al igual que el servidor, sin embargo, no es necesario que cuando el usuario intenté crear un personaje envíe un json, se va a enviar un mensaje del estilo "crearpinky" o "crearfruta", se va a trabajar con eso

07 de noviembre

Se logró implementar la escritura de json y la totalidad de comunicación cliente servidor