Costa Rican Institute of Technology

Computer Engineering

PaCeman

Documentación Técnica

Profesor Ing. Marco Rivera Meneses

Por:

Acuña Durán Oscar Arturo - 2022049304

Núñez Pérez Henry Daniel - 2022089224

Rojas Rojas Mariana Saray - 2020076936

Computer Engineering

10 de noviembre del 2023

Índice

1. Estructuras de datos desarolladas

Para el manejo y organización de partidas en ejecución se declararon las estructuras jugador y observador.

```
struct Jugador {
    int ID;
    SOCKET clientSocket;
    int puntos;
};

struct Observador {
    int ID;
    SOCKET clientSocket;
};
```

La estructura jugador se encuentra compuesta por su identificador (ID), su socket de conexión (clientSocket) y su cantidad de puntos acumulados (puntos). Por otra parte, la estructura observador de encuentra compuesta por su identificador de suscripción asociado (ID) y su socket de conexión (clientSocket). La disposición de estas estructuras permite un fácil acceso a las características de cada cliente. Además, la particularidad de tener como atributo un SOCKET permite al servidor comunicarse con varios sockets de manera paralela.

Para la comunicación en tiempo real con los clientes durante la ejecución de partidas se declararon dos arreglos de estructuras, una de cada

tipo de estructura.

```
11 struct Jugador jugadores[MAX_JUGADORES];
12 struct Observador observadores[MAX_OBSERVADORES];
```

Ambos arreglos permiten establecer un orden y fácil acceso a los jugadores y observadores en tiempo real de ejecución. Además, permiten establecer límites de aceptación de clientes. Para este programa se establece un valor máximo de jugadores de 2 y de 4 observadores.

El arreglo es utilizado para acceder fácilmente a los sockets de todos los clientes asociados a un ID, esto permite enviar instrucciones a los jugadores y actualizar el estado de sus observadores.

```
13
14     for (int i = 0; i < numJugadores; i++) {
15         if (jugadores[i].ID == targetID) {
16             send(jugadores[i].clientSocket, message, strlen(message)
                   , 0);
17         }
18     }
19
20     for (int i = 0; i < numObservadores; i++) {
21         if (observadores[i].ID == targetID) {
22             send(observadores[i].clientSocket, message, strlen(
                   message), 0);
23         }
24     }
```

2. Algoritmos Desarollados

Para el movimiento de los fantasmas en el tablero se utilizó la técnica random de pathfinding. Esto con el fin de simplificar la estructura del código. Este algortimo se implementa en la función moveGhosts(Graphics2D g2d).

```
26 if (ghost_x[i] % BLOCK_SIZE == 0 && ghost_y[i] % BLOCK_SIZE == 0) {
27
28 pos = ghost_x[i] / BLOCK_SIZE + N_BLOCKS * (int) (ghost_y[i] /
      BLOCK_SIZE);
29 count = 0;
```

Dentro de la función, en primer instancia se verifica si el fantasma a llegado a una nueva celda en el tablero, se calcula la posición y se inicia un contador de direcciones de movimiento posibles.

```
31 if ((screenData[pos] & 1) == 0 && ghost_dx[i] != 1) {
32     dx[count] = -1;
33     dy[count] = 0;
34     count++;
35 }
36 if ((screenData[pos] & 2) == 0 && ghost_dy[i] != 1) {
37     dx[count] = 0;
38     dy[count] = -1;
39     count++;
40 }
41 if ((screenData[pos] & 4) == 0 && ghost_dx[i] != -1) {
42     dx[count] = 1;
```

```
43      dy[count] = 0;
44      count++;
45  }
46  if ((screenData[pos] & 8) == 0 && ghost_dy[i] != -1) {
47      dx[count] = 0;
48      dy[count] = 1;
49      count++;
50  }
```

Seguidamente, se verifica, dirección por dirección si es posible moverse a a esa celda y se registra en el contador.

```
52
53  count = (int) (Math.random() * count);
54  if (count > 3) {
55      count = 3;
56  }
57  ghost_dx[i] = dx[count];
58  ghost_dy[i] = dy[count];
59
60  ghost_x[i] = ghost_x[i] + (ghost_dx[i] * ghostSpeed[i]);
61  ghost_y[i] = ghost_y[i] + (ghost_dy[i] * ghostSpeed[i]);
```

Si count es 0 no tiene direcciones disponibles pero si es diferente de cero, como se puede observar en el código de arriba, se elige una dirección aleatoria y se actualiza la posición.

```
63
64  if (pacman_x > (ghost_x[i] - 12) && pacman_x < (ghost_x[i] + 12)
```

```
65          && pacman_y > (ghost_y[i] - 12) && pacman_y < (ghost_y[i] +
                12)
66          && inGame) {
67      dying = true;
68  }
```

Por último, verifica si existe alguna colisión con el personaje Pacman, en caso de que sí, se procede con el protoco de eliminar una vida.

3. Problemas sin solución

3.1. Comportamiento de fantasmas

Uno de los principales problemas encontrados sin solución se encuentra en la clase observador. Como se mencionó en la sección anterior, los fantasmas funcionan mediante un algoritmo random de pathfinding. Y, como la única diferencia entre la clase observador y cliente es el movimiento del pacman, el comportamiento random de los fantasmas en la clase jugador y sus observadores varía aunque inicien en el mismo lugar al mismo tiempo. Por esto, los fantasmas en la clase observador se comportan diferente a los fantasmas del jugador que se está visualizando.

3.2. Comportamiento del juego en instancias avanzadas

Conforme avanza la partida y se añaden más objetos al tablero, la comunicación y replicación del observador se vuelve más lenta. En ocasiones se ve reflejada con perder un comando de movimiento del pacman del jugador lo cual causa un desfase entre el movimiento del observador y jugador. También en ocasiones, en instancias avanzadas de la partida ocurre que cuando se llega a los 10 000 puntos se asignan dos vidas en lugar de una.

4. Plan de Trabajo

Este es el plan de trabajo diseñado inicialmente con las tareas que se esperó cumplir durante el desarrollo del proyecto.

Tarea	Fecha de entrega	Duración estimada	Responsable
Reunión inicial del grupo	19/10/13	1 hora	Arturo Acuña, Henry Núñez y Mariana Rojas
- Comprensión del problema - Diseñar una solución a grandes rasgos - Dividir la tarea entre los miembros del equipo			
Investigación preliminar	21/10/23-22/10/23	2 horas	Mariana Rojas
- Investigar mejores ambientes de desarrollo para C y Java - Investigar la configuración de esos ambientes - Reorganizar el diagrama de clases hecho en el taller de POO y adaptar la parte del servidor en C			
Protocolo de conexión de sockets	23/10/23	4 horas	Henry Núñez
- Diseñar un socket cliente en Java - Diseñar un socket servidor en C - Adaptar el código para crear una conversación continua entre ambos			
Interpretación de estructuras	26/10/23	6 horas	Henry Núñez
- Crear una función que permita interpretar los mensajes ingresados en consola por el admin y estandarizarla para enviarla al servidor - Crear una función en cliente que permita interpretar la estructura y la muestre en pantalla			

Organización del servidor	2/11/23	4 horas	Henry Núñez
- Crear structs para jugador y observador, además de arreglos de estos - Adaptar el código para recibir múltiples clientes y elegir con cual comunicarse - Agregar función para interpretar los mensajes enviados por el cliente (puntos, fantasma, fruta, pastilla, nivel) y actúe en consecuencia ya sea asignado vidas, puntos o cambiando nivel - Crear función que reciba mensajes del cliente y actualice			

Diseño del cliente	31/10/23	8 horas	Mariana Rojas
- Buscar una guía para la creación base de un juego pacman - Implementar el diseño en el cliente - Implementar Abstract Factory para la creación de fantasmas y pastillas - Implementar funcionalidades faltantes en caso de que existan - Diseñar clase cliente y observador, además de un menú de escogencia			
Organización del cliente	5/11/23	10 horas	Arturo Acuña
- Adaptar el sistema de sockets previamente diseñado para que actúe en consecuencia y realice cambios respectivos según lo solicitado - Crear función se envío de mensajes que informe al servidor para que actualice a los observadores - Adaptar las funcionalidades de movimiento respectivas para cada cliente ya sea jugador o cliente			
Periodo de pruebas	5/11/23	10 mins – tiempo necesario en arreglo de bugs	
- Creación de archivo .exe y .jar - Probar casos y depulir errores			
Documentación interna	7/11/23	3 horas	Henry Núñez
- Documentar con formato javadoc servidor y cliente			
Documentación externa	8/11/23	4 horas	Arturo Acuña, Henry Núñez y Mariana Rojas

5. Problemas Encontrados

5.1. Manejo de hilos

Uno de los principales problemas encontrados durante el desarrollo del proyecto fue la interrupción entre procesos de ejecución. El desarrollo de la solución de este proyecto implica la utilización de múltiples hilos para permitir una correcta ejecución. La implementación de hilos puede ser contraproducente si no se tiene suficiente cuidado con factores como cuándo se inician y dónde se inician, que fue lo que ocurrió en el proceso de desarrollo.

Un ejemplo de esto es que el servidor iniciaba el hilo de escucha a la consola cada vez que un cliente se unía al programa, lo que provocaba que las peticiones del administrados se perdieran entre tantas preguntas que se hacían en la consola. Esto se solucionó iniciando el hilo una sola vez desde el comienzo de la ejecución.

Otro ejemplo de esto ocurrió cuando se iniciaba el juego en java. Cuando se abría el menú principal iniciaba el hilo que instanciaba el juego completo y solo se escondía la ventana, luego se volvía a iniciar un mismo hilo cuando se elegía el tipo de cliente. Esto provocaba superposición de procesos y pérdida de instrucciones del servidor. Se solucionó iniciando el hilo a partir de los botones de selección de cliente.

Para evitar esta problemática desde un inicio se recomienda dedicar tiempo a pensar la distribución de los hilos y cómo acomodarlos en el código para que se activen cuando sea necesario antes de solo colocarlos en el código y ya. Se debe tener en cuenta cómo los afecta el funcionamiento general del programa y no solo el particular.

5.2. Organización de código cliente

Un problema importante desarrollado durante el diseño del cliente fue la organización y distribución de clases. Esto fue principalmente provocado por la necesidad de varias clases de utilizar variables dadas por la clase Jugador u Observador. Esto provocó que varios procesos como la conexión de sockets o arduino se tuviera que implementar dentro de estos archivos, generando lineas de código bastante grandes. Si bien este problema de organización no fue solucionado se logró que el código funcionara de manera correcta. Sin embargo, sí se recomienda establecer una mejor arquitectura de diseño para el juego que permita establecer divisiones claras del contenido del programa.

6. Conclusiones del Proyecto

1. A lo largo del proyecto se logró experimentar la manera en que los paradigmas de programación imperativa y orientado a objetos pue-

den complementarse bastante bien. Si bien es cierto, cada paradigma ofrece sus ventajas y desventajas, pero una buen diseño de arquitectura puede permitir la creación de proyectos más robustos con su combinación. En este proyecto se vio demostrado con el manejo de instrucciones entre lenguajes. Realmente no fue un desafío comunicar y enviar estructuras entre ambos lenguajes e incluso la programación del servidor en C ofreció un diseño igual o más simplificado en comparación de haber sido en Java.

2. En este proyecto se logró ver claramente la diferencia entre la eficiencia y la abstracción que los paradigmas ofrecen y la importancia de escoger bien cuál paradigma utilizar para el diseño de una solución. Mientras que C se destaca por su eficiencia y control de memoria, Java ofrecía una abstracción más sólida y facilidades para la reutilización de código y adaptación de diseños.

3. La curva de aprendizaje del paradigma imperativo fue mucho menor a la curva de aprendizaje del paradigma orientado a objetos. Es más fácil familiarizarse con el lenguaje C y entender sus funcionalidades antes que la implementación de las características de objetos en Java. Por ejemplo, implementar el Abstract Factory fue complicado al inicio y entender cómo funcionaba aún más, sin embargo, fue útil para simplificar la solución del problema.

4. Cabe mencionar que, aunque C y Java describen paradigmas diferentes, C es un lenguaje muy adaptable y flexible en su diseño. Es

posible, mediante la utilización de structs, escribir un código en C de manera más orientada a objetos. Ya que los structs permiten asociar variables relacionadas en un solo lugar, como lo hace una clase.

7. Recomendaciones del proyecto

1. Antes de empezar un proyecto que implique la utilización de dos lenguajes de programación y dos paradigmas distintos, se recomienda realizar un diseño concreto y adecuado de las funcionalidades de cada parte del proyecto. Esto con el fin de elegir de manera adecuada cuál paradigma es más apropiado utilizar según la parte del proyecto y así un desarrollo más continuo.

2. Para el lenguaje de programación C, se recomienda la buena práctica de crear un archivo a parte de constantes para tener un control adecuado y fácil acceso de estas para su futuro modificación. Para Java, se recomienda la utilización del patrón Abstract Factory para la simplificación de diseño de objetos y reutilización de código.

3. Se recomienda la utilización de la libreria jSerialComm para la conexión de Arduino y Java ya que, por su gran estandarización y constante mantenimiento ofrece un protocolo sencillo de implementar y muy eficiente en el manejo de información en tiempo real.

4. Para proyectos cómo el realizado en el cual se implementen muchas funcionalidades en un solo programa se recomienda la buena práctica

de modularizar al máximo el programa. La importancia de la modularización radica en la ventaja que ofrece sobre el mantenimiento a largo plazo. Un código correctamente distribuido permite identificar errores fácilmente y que el código continúe funcionando aún con fallas en ciertas secciones.

Referencias

gaspar coding. (4 de septiembre de 2020). Pacman in Java, Programming Tutorial 1/2 [Archivo de Vídeo]. YouTube. https://www.youtube.com/watch?v=ATz7bIqOjiA

gaspar coding. (10 de septiembre de 2020). Pacman in Java, Programming Tutorial 2/2 [Archivo de Vídeo]. YouTube. https://www.youtube.com/watch?v=r7i25lbmBd4

8. Bitácora

8.1. Bitácora Henry Núñez

Actividad	Fecha
Reunión inicial del grupo	19/10/13
- Se comentó la disponibilidad de cada uno para trabajar en el proyecto y cómo se iba a organizar. - Como yo tenía la primera semana del proyecto libre propuse iniciar son el servidor en C	
Investigación preliminar	20/10/23
- Empecé a ver videos sobre el mejor ambiente para programar en C y me decidí por Clion por su estandarización y facilidad de manejo. - Como ya había trabajado previamente con sockets sabía que la comunicación no iba a ser difícil. Entonces empecé buscando un video que explicara cómo establecer un socket servidor en C. Utilicé la biblioteca WinSock ya que es propia de la biblioteca estándar de C99 y no necesita previa instalación.	
Protocolo de conexión de sockets	20/10/23
- Creé un repositorio en Github con un proyecto en C y otro en Java (Por el momento utilicé IntelliJ para trabajar porque ya había trabajado antes en este entorno). - Implementé el socket en C a partir de un protocolo encontrado en investigación. Además, implementé el cliente en Java usando java.net.Socket. Se enviaba un mensaje y se enviaba en consola con ambos sentidos. - Implementé ambos procesos de escucha y escritura en ambos códigos en dos hilos diferentes para que se pudiera crear una conversación fluida. - Establecí un bosquejo de comandos relacionada a la solución y empecé a adaptar el código para que enviara ciertos mensajes si recibía ciertas instrucciones. - Esto para que fuera más fácil la conexión con la interfaz.	
Arduino	22/10/23
- Ya tenía un código de botones previamente diseñado por lo que solo lo adapté al proyecto y lo probé. - Investigué sobre las mejores librerías para la conexión de java y Arduino y me encontré un video de jSerialComm el cual implementé en mi solución. - Añadí la clase de Arduino al proyecto de Java.	

Organización del servidor	26/11/23
- Añadí la opción de verificación en los sockets para que se cumplan los requisitos de que el cliente esté constantemente avisando al servidor lo que sucede dentro del juego. También que cuando se acaben los puntos del tablero solicite cambiar de nivel y el servidor lo autorice. - Implementé los structs de jugador y observador para ya empezar a implementar admitir múltiples jugadores a la vez. - Establecí un protocolo de comunicación que ordena en un array los jugadores u observadores que van ingresando y les asigna un id. Entonces cuando el servidor quiere enviar un mensaje a algún jugador, se verifica primero a cuál id quiere mandarlo y así busca entre sus opciones.	
Arduino	31/10/23
- Empecé a ordenar los circuitos del Arduino en la placa perforada y utilicé una caja negra para crear el control. - Lo soldé e hice pruebas para ver que todo funcionara correctamente.	

Organización del cliente	5/11/23
- Luego de recibir la base de la interfaz gráfica de pacman diseñada inicialmente por Mariana me reuní con Arturo para implementar todas las funcionalidades faltantes para lograr una conexión adecuada. - Inicié incorporando una clase Abstract Factory para la creación de fantasmas y frutas. Arturo me ayudó buscando las imágenes. - Luego se incorporó el cambio de niveles, el aumento de vidas, recuento de puntos correcto y la opción de comerse fantasmas a partir de una pastilla. - Luego se adaptó esto para que en la consola de java las cosas fueran ocurriendo según los comandos ingresados. - Se realizaron la clase cliente y observador para separar la forma de movimientos - Hubo problemas con separar ciertas clases del juego principal ya que muchos procesos del pacman original ocupaban variables privadas por lo que la clase cliente y observador quedaron muy grandes en términos de código. - Se implementó el socket de manera correcta y funcionaba con las instrucciones del servidor. - Se probó e implementó el movimiento con Arduino exitosamente.	
Diseño de clase Observador	7/11/23
- Para esto se creó la extensión .jar porque se necesitaba un cliente jugador funcionando para poder observarlo. Se adaptaron las cosas necesarias relacionadas con movimiento para que el cliente observador siguiera los mismos pasos del jugador y fue exitoso.	
Documentación interna y pruebas	7/11/23
- Documenté el código en C y Java en formato Javadoc. Además, probé los archivos en .exe y .jar.	
Realización de la documentación externa	8/11/23 y 9/11/23

8.2. Bitácora Arturo Acuña

Bitácora Arturo		
Fecha	Duración	Tarea Realizada
2023-10-28	2 horas	Investigación sobre control de PacMan Dediqué este tiempo a explorar diferentes estrategias de control para PaCEman en PacMan. Busqué algoritmos y enfoques que pudieran garantizar un movimiento eficiente y lógico del personaje.
2023-10-30	3 horas	Implementación del control de PaCEman Me enfoqué en traducir la investigación en acción, implementando el control de PaCEman en la aplicación cliente (Java). Esto incluyó la integración de la lógica de movimiento
2023-11-01	6 horas	Crear estructuras en Java para el control de PacMan Desarrollando las estructuras en Java necesarias para gestionar eficientemente el control de PaCEman. Esto implicó la creación de clases y paquetes siguiendo los principios de la programación orientada a objetos
2023-11-03	3 horas	Documentar la implementación del control de PacMan Me dediqué a documentar exhaustivamente el código desarrollado para el control de PaCEman. Esto incluyó comentarios en el código
2023-11-05	2 horas	Revisión general y ajustes en el control de PacMan Revisión completa del código del control de PaCEman, identificando posibles mejoras y haciendo ajustes para garantizar un rendimiento óptimo
2023-11-07	2 horas	Participar en reuniones del grupo de trabajo Dediqué tiempo a participar activamente en reuniones con otros miembros del equipo
2023-11-09	4 horas	Implementar conexiones entre C y Java (Sockets) Trabajé en la implementación de la conexión entre la aplicación servidor (C) y la aplicación cliente (Java) utilizando Sockets, facilitando la comunicación efectiva entre ambas partes del sistema.
2023-11-11	3 horas	Solucionar problemas de implementación en el control
2023-11-11	3 horas	Preparar sección de Problemas sin solución y problemas encontrados de la documentación

8.3. Bitácora Mariana Rojas

Bitácora Mariana		
Fecha	Duración	Tarea Realizada
2023-10-28	2 horas	Investigación sobre programación en C: Realicé una investigación exhaustiva sobre programación en C, centrándome en las mejores prácticas y estructuras de datos eficientes para la implementación del servidor del juego
2023-10-30	8 horas	Implementación de la lógica del juego en Java: Desarrollé la lógica principal del servidor en C, incluyendo la creación de fantasmas, pastillas, frutas, asignación de vidas, control de niveles y puntuación. Se implementaron las funciones necesarias para gestionar estas acciones.
2023-11-01	2 horas	Crear estructuras en C (structs): Diseñé e implementé las estructuras en C, como structs, para manejar eficientemente la información del juego, como los datos de los fantasmas, frutas y otros elementos del juego.
2023-11-03	2 horas	Documentar la implementación en C Documenté detalladamente el código implementado, explicando la funcionalidad de cada parte y proporcionando comentarios para facilitar la comprensión y mantenimiento del código.
2023-11-05	1 hora	Revisión general y ajustes en C Realicé una revisión exhaustiva del código, corrigiendo posibles errores y realizando ajustes para mejorar la eficiencia y claridad del código.
2023-11-07	2 horas	Reunión del grupo de trabajo para unir avances Participé en una reunión del grupo de trabajo para compartir avances, discutir posibles problemas y coordinar esfuerzos para una implementación más eficiente.
2023-11-08	2 horas	Revisión de la documentación general del proyecto Revisé la documentación general del proyecto, asegurándome de que esté completa y coherente con los avances realizados en la implementación.
2023-11-08	3 horas	Implementar conexiones entre C y Java (Sockets) Desarrollé el código necesario para establecer conexiones entre la aplicación servidor en C y la aplicación cliente en Java utilizando sockets.
2023-11-09	3 horas	Solucionar problemas de implementación en C Identifiqué y solucioné problemas en la implementación en C, asegurándome de que la comunicación entre el servidor y el cliente fuera fluida y sin errores.
2023-11-10	3 horas	Describir estructuras de datos y algoritmos desarrollados