Escuela de Computación

Área Académica Ingeniería en Computadores

Curso: CE-3104 Lenguajes, Compiladores e Intérpretes

Grupo #2

Tarea #3: Space Invaders

Prof. Marco Rivera

Grupo de Trabajo #2

Estudiantes:

Dylan Garbanzo Fallas

Irene Garzona Moya

Alejandra Rodríguez Castro

I Semestre 2023

- Algoritmos de solución desarrollados:
  Patrón Observer: Este patron de diseño se utiliza en la clase Alien, cada objeto de alien tiene asignado un observador. En el método move() de Alien, en caso de que el alien colisione con alguna de las dos esquinas de la pantalla (derecha u izquierda), este va a aumentar su posición en 'y' y cambiar el signo del atributo de velocidad, de manera que se empieza a mover en dirección contraria hasta alcanzar el otro lado de la pantalla. Cuando estas colisiones con el final de la pantalla ocurren, el alien que colisionó notifica a todos sus observadores (todos los otros alien del juego), que hizo un cambio de posición en 'y' y de velocidad, para que estos aliens observadores hagan lo mismo, y de esta manera crear el conocido efecto de una matriz de aliens moviéndose en conjunto del juego spaceInvaders.
- Diagrama (s):

Diagrama del servidor:

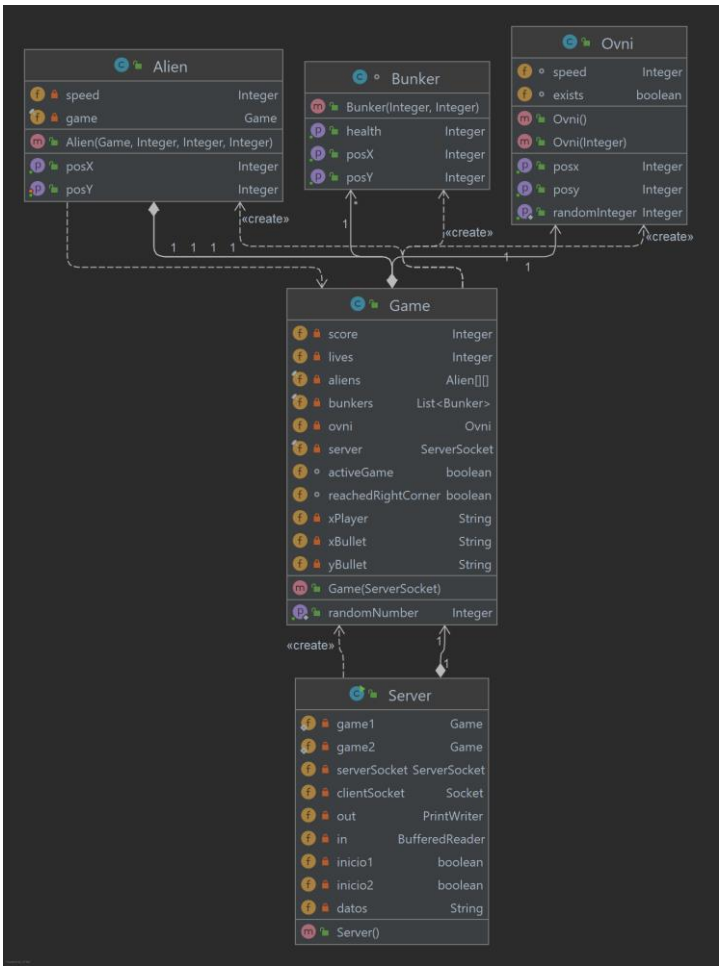
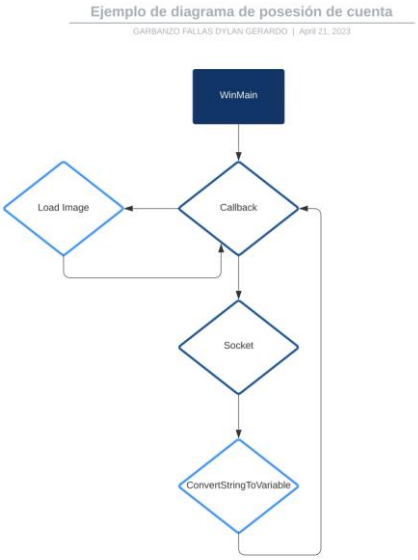


Diagrama del Cliente, Cliente2 y Espectador:



- Funciones implementadas:
    - Servidor:
      -Public void start(int port): Es la función que inicializa el servidor, e intenta conectar con un cliente en el puerto dado.
      -public void stop(): Cierra la conexión con el cliente.
      -public void sendToClient(String message): Envia el String dado al Cliente.
      -public Boolean isClientConnected(): Retorna true si el cliente sigue conectado y false si no.
      -public String receiveMessage(): Retorna el String recibido por el cliente.
      -public void updateGame(): Mantiene el juego activo, mueve todos los aliens y envia la matriz nueva de aliens al cliente.
      -void setReachedRightCorner(): cambia el valor del booleano por su opuesto, en caso de ser true los aliens alcanzaron el borde derecho de la pantalla, en caso de false es el borde izquierdo de la pantalla.
      -void getAlienCoords(String message): Obtiene el tipo y coordenadas de un alien a partir del String recibido por el Cliente.
      -public void createAliens(Integer numRows, Integer numCols, Integer x, Integer y, Integer speed): Esta function crea la matriz de aliens del tamaño dado en las posiciones dadas, y les el valor de speed definido a todos los aliens.
      -public List<Integer> findPosAlien(Integer xPos, Integer yPos): Encuentra en que posición de la matriz se encuentra un alien a partir de sus coordenadas x y y.
      -public void deleteAlien(Integer Row, Integer Column): Elimina un alien de la matriz de aliens a partir de su posición en la matriz.
      -public void createBunkers(): Crea la lista de Bunkers, siempre inicia con la misma vida y en las mismas posiciones.
      -public void createOvni(): Crea un nuevo Ovni.
      -public void update(Observable o, Object arg): Metodo del patron Observable, se actualiza constantemente para ver si hubo alguna colisión con los extremos de la pantalla de algún alien.

-private void updateSpeed(): En caso de haber un cambio en la velocidad y posición de algún alien, hace el mismo cambio para todos los aliens.
-public static Integer getRandomNumber(): Genera un numero random entre 0 y 20 para ver si sale un Ovni.
-public void verifyWin(): Verifica si todos los aliens de la matriz están muertos, de ser así le da mas vidas y puntaje al jugador.
-String generateAliensMatrix(): Convierte la matriz de aliens a un String para pasársela al cliente.
-String generateBunkersMatrix(): Convierte la lista de bunkers en un String para pasársela al cliente.
-String generateOvniString(): Convierte las coordenas del Ovni en un String para pasársela al cliente.
-String generateFinalString(): Une todos los datos necesarios en un solo String para pasarle únicamente todo este String al cliente.

- Cliente:
  -int Cliente(): tiene toda la lógica necesaria para la conexión de sockets TCP, y también el envío y recibimiento de la información con el servidor.
  -int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow): función principal de la API de Windows en C, que permite la creación de la ventana base de la interfaz gráfica.
  -LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam): función en respuesta al WInMain(...). Se encarga de toda la lógica de la ventana.
  -void convertStringToVariables(char str[]): función que traduce el string con comandos reciido por el servidor.
  -void LoadMyImage(void): función que se encarga de cargar las imágenes necesarias para la interfaz antes de ser necesitadas.

- Ejemplificación de estructuras de datos desarrolladas:

Las estructuras de datos utilizadas en el presente código son principalmente matrices, ya sea con coordenadas, o información extra que requiere el juego para funcionar.

Además de esto de parte del Cliente, se tienen 2 estructuras de datos: Coord y disparo, las cuales guardan lo mismo, dos integer x y, que corresponden a coordenadas, y una imagen de tipo HBITMAP, al ser lo mismo, se tiene por separado para facilitar su uso y evitar confusiones al utilizar estas estructuras, ya que muchas veces se trabajan juntas para verificar colisiones.

También se utiliza una

- Problemas sin solución:

A veces, debido a la mala optimización de la api de interfaz utilizada, las imágenes se cargan por error encima de las otras, generando un bug visual que se arregla después de actualizar la interfaz

Debido a la naturaleza de la api de interfaz utilizada, se consumen demasiados datos y recursos, los cuales afectan el rendimiento de la computadora, y se refleja en un movimiento lento de las naves.

- Problemas encontrados:

Muchas veces al ejecutar el programa, después de realizar un push del repositorio de GitHub, la librería que utilizamos “ws2_32.lib” no carga correctamente, lo que implica estar refrescándola (eliminarla del CMake.txt, compilar el proyecto, y luego volver a cargarla).

- Plan de actividades:

Actividad		Encargados(as)	1/4/2023	3/4/2023	10/4/2023	15/4/2023	16/4/2023	17/4/2023	18/4/2023	19/4/2023	20/4/2023	21/4/2023
Leer documento instructivo de la tarea		Todos										
Trabajo Escrito		Dylan y Alejandra										
Cliente	Confección de la Interfaz	Dylan										
		Alejandra										
	Interpretación Reglas	Dylan										
Servidor	Análisis estado de juego	Alejandra										
	Refrescamiento condiciones del juego	Alejandra										
Sockets	Base Sockets	Dylan										
		Irene										
	Implementación con interfaz	Alejandra										
		Dylan										
	Implementación con lógica	Alejandra										
		Dylan										
Unificar trabajo final		Dylan y Alejandra										
Revisión de bugs y detalles		Dylan y Alejandra										

- Bitácora:
- Fecha: 01-04-2023
    - Miembros:
    - Dylan Garbanzo Fallas
    - Alejandra Rodríguez Castro
    - Irene Garzona Moya

Se hace una lista de actividades y cronograma, además de la creación del repositorio de Github.

- Fecha: 03-04-2023
    - Miembros:

    - Dylan Garbanzo Fallas

El estudiante Dylan Garbanzo termina con la base de los sockets

- Fecha: 15-04-2023
    - Miembros:
    - Dylan Garbanzo Fallas
    - Alejandra Rodríguez Castro
    - Irene Garzona Moya

El estudiante Dylan Garbanzo inicia con la interfaz, probando diferentes librerías. La estudiante Irene Garzona realiza pruebas de las conexiones de sockets. Y la estudiante Alejandra Rodrígez empieza con la base de la lógica del servidor y cliente.

- Fecha: 16-04-2023
    - Miembros:
    - Dylan Garbanzo Fallas
    - Alejandra Rodríguez Castro

Tanto el estudiante Dylan Garbanzo, como la estudiante Alejandra Rodríguez, se encargan de ir conectando e implementando el sistema de mensajería de datos entre el cliente y servidor, así como la interpretación del mismo en ambas partes.

- Fecha: 17-04-2023
    - Miembros:
    - Dylan Garbanzo Fallas
    - Alejandra Rodríguez Castro

La estudiante Alejandra Rodríguez termina con la lógica del servidor para la estructura del momento, además el estudiante Dylan Garbanzo logra la interpretación exitosa de esta información, y que sea reflejada en la interfaz.

- Fecha: 18-04-2023
    - Miembros:
    - Dylan Garbanzo Fallas
    - Alejandra Rodríguez Castro

La estudiante Alejandra Rodríguez logra arreglar conexiones entre cliente/servidor, así como la adición de detalles como los bunkers. El estudiante Dylan Garbanzo logra la implementación del disparo del jugador, así como la detección de colisión con los aliens.

- Fecha: 20-04-2023
    - Miembros:
    - Dylan Garbanzo Fallas
    - Alejandra Rodríguez Castro

Se inicia la elaboración de la documentación externa, así como la implementación del sistema de colisiones de disparos con los bunkers. Además se intenta la implementación de conexión de un segundo jugador y un espectador. Se logra la lógica de separación de información, pero no la implementación completa de estos.

- Fecha: 21-04-2023
    - Miembros:

    - Dylan Garbanzo Fallas
    - Alejandra Rodríguez Castro

Se termina la documentación externa. Y se revisa posibles errores que se puedan presentar en la defensa del proyecto.

- Conclusiones:
  -A la hora de crear una conexión cliente-servidor, ambos lados deben de estar altamente coordinados para que el funcionamiento sea adecuado.
  -Las bibliotecas graficas de C no están hechas para estarse actualizando, en especial si son muchos elementos, ademas de que hay varias librerias en Linux que ayudan un poco con el rendimiento.
  -
- Recomendaciones:
  -Cuando se programen el orden de acontecimientos tanto de la interfaz grafica como de el juego, colocar el código de manera que ambos sean compatibles, es decir, intentar que las cosas sucedan primero en el servidor, pasarlo al cliente, sucede en el cliente y lo retorna al servidor, de otra forma, el juego no estará sincronizado con la interfaz.
  -Evitar hacer la interfaz grafica en C, ya que no estan optimizadas para soportar mucha carga, y si es necesario hacerla en C, es mejor realizarla en Linux y no en WIndows.

- Referencias bibliográficas:

The GDI in Windows API. (n.d.). Retrieved April 20, 2023, from https://zetcode.com/gui/winapi/gdi/

DiscoDurodeRoer (Director). (2018). *Ejercicios Java—Sockets #1—Conexión TCP cliente/servidor*. https://www.youtube.com/watch?v=3wJTI9LMOsk

Profe Maty (Director). (2020). *Creando un servidor y un cliente con sockets*. https://www.youtube.com/watch?v=4JIuR5REknE

Patrones de diseño / Design patterns. (n.d.). Retrieved April 20, 2023, from https://refactoring.guru/es/design-patterns