Estudiantes:

2019042722 Fabricio Mena Mejía

2022279667 José Daniel González Chaves

2021050678 Braulio Retana Murillo

Instituto Tecnológico de Costa Rica

Curso: Paradigmas de programación

Tarea IV

Tema: iCE Climber

Grupo: 01

Profesor: Marco Rivera Meneses.

Fecha: 07/06/2025

1. 1 Descripción de la utilización de las estructuras de datos desarrolladas

Player

La clase Player es fundamental para representar a los jugadores en el juego. Cada jugador tiene atributos como su vida, nivel, puntuación, y una serie de métodos para interactuar con el juego (como moverse, saltar y recibir daño de los obstáculos).

Atributos:

- String name: El nombre del jugador (Popo o Nana).
- int life: La cantidad de vidas que le quedan al jugador.
- int score: La puntuación del jugador, que se actualiza cuando destruye un obstáculo o recoge frutas.

Métodos:

- move(): Este método se usa para mover al jugador en la pantalla. Según las teclas presionadas, el jugador se desplazará horizontalmente o saltará para subir de nivel.
- jump(): Permite al jugador saltar y subir un nivel si está en el nivel correcto.
- hitObstacle(): Este método es llamado cuando el jugador es alcanzado por un obstáculo. Resta una vida del jugador.

Obstacle

La clase Obstacle representa a los obstáculos que los jugadores deben evitar. Puede ser un Yeti, una ave, o un bloque de hielo, y cada tipo de obstáculo tiene diferentes comportamientos.

Atributos:

- String type: Tipo de obstáculo (Yeti, Ave, Hielo).

- int xPosition, yPosition: Posición en el plano 2D del juego donde aparece el obstáculo.

Métodos:

- move(): Este método controla el movimiento del obstáculo. Los obstáculos se mueven o caen dependiendo de su tipo.
    - **Yetis**: Se mueven horizontalmente, de izquierda a derecha.
    - **Aves**: Se mueven verticalmente, de arriba a abajo.
    - **Hielos**: Caen verticalmente desde un piso superior

GameSession

La clase GameSession se utiliza para gestionar las sesiones de juego activas. Cada sesión almacena la información de los jugadores y los obstáculos presentes.

Atributos:

- String sessionId: Identificador único de la sesión de juego.
- Player[] players: Array que almacena a los jugadores involucrados en la sesión.
- Obstacle[] obstacles: Array que almacena los obstáculos presentes en la sesión.

Métodos:

- addPlayer(): Añade un jugador a la sesión.
- removePlayer(): Elimina un jugador de la sesión cuando se desconecta o pierde el juego.
- startGame(): Inicia la sesión de juego y comienza la lógica del juego.

Fruit

La clase Fruit se utiliza para representar las frutas/verduras recolectables en la fase de bonus. Cada fruta tiene un tipo y un valor asociado.

Atributos:

- String type: Tipo de fruta (naranja, banano, berenjena, lechuga).
- int value: Valor de la fruta, que otorga puntos cuando se recoge.

Métodos:

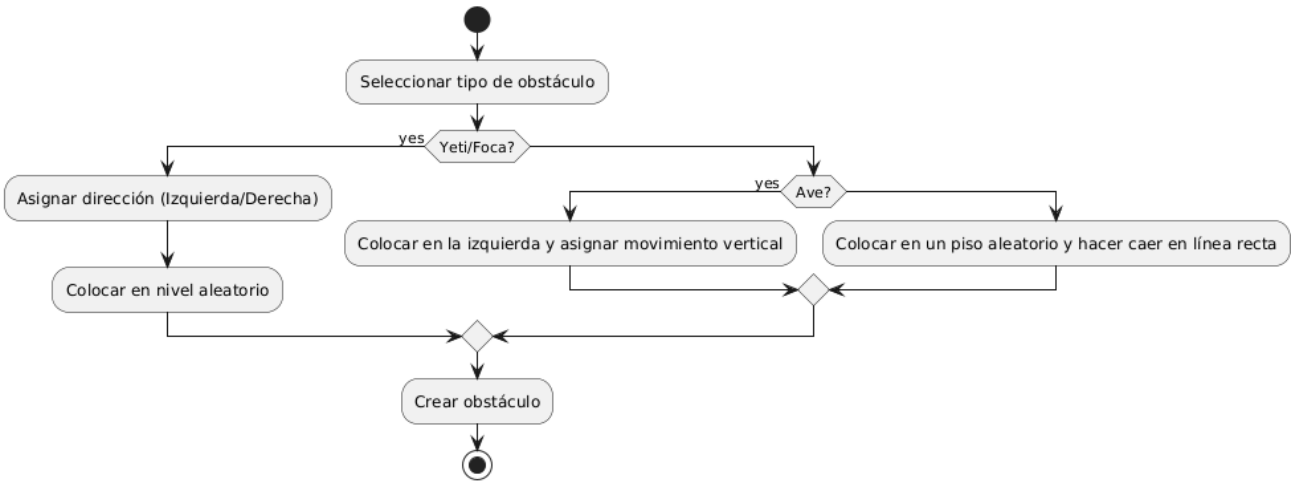
- collect(): Método que permite al jugador recoger la fruta y sumar puntos a su puntuación.

1. 2. Descripción detallada de los algoritmos desarrollados

Creación de Obstáculos

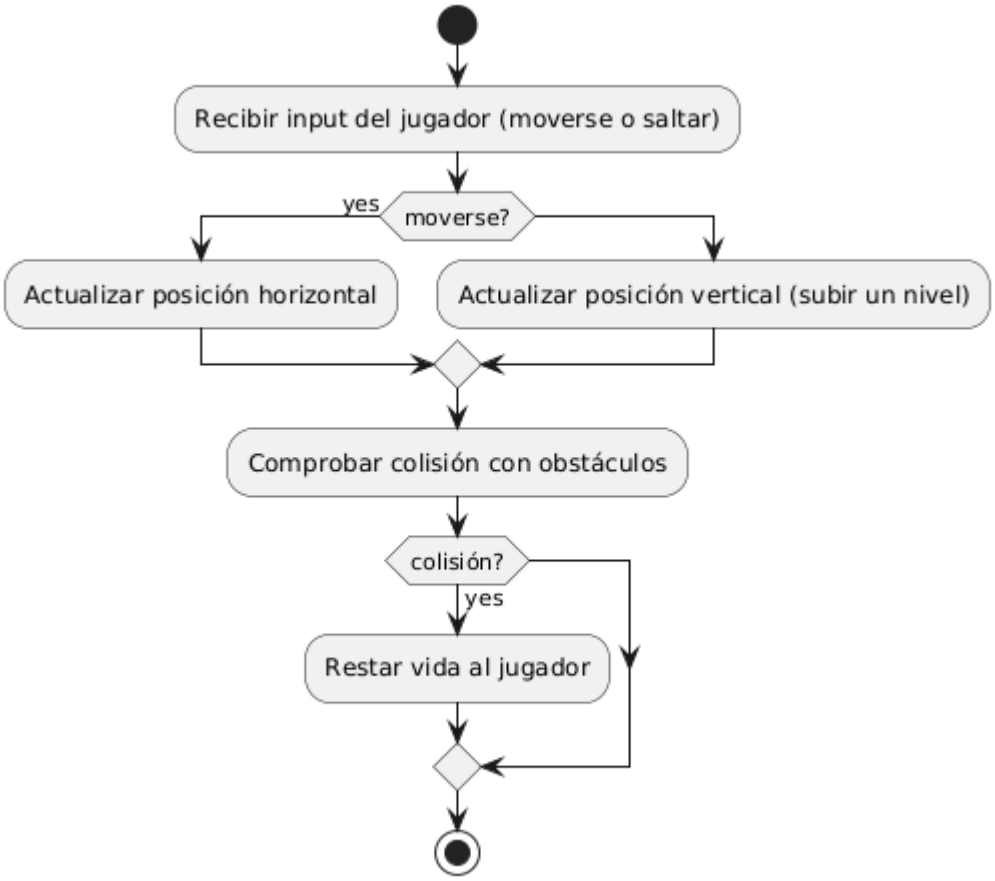
El algoritmo de creación de obstáculos se ejecuta en el servidor. Los usuarios pueden decidir qué tipo de obstáculo crear (Yeti, Ave, Hielo) y asignar su posición.

1. **Selección de obstáculo**: El usuario selecciona el tipo de obstáculo (Yeti, Ave, Hielo).
2. **Colocación**: Según el tipo, se asignan las propiedades adecuadas (dirección para Yeti/Foca, movimiento para las aves, caída para los hielos).
3. **Creación**: Se genera un obstáculo y se añade al juego.



Movimiento de Jugadores

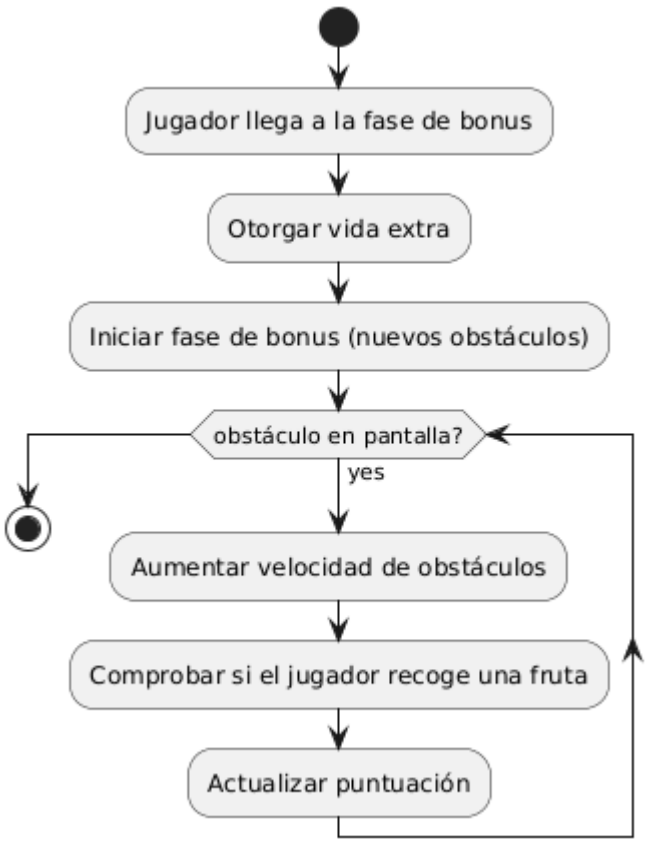
Este algoritmo se encarga de permitir que los jugadores se muevan por el mapa. Los jugadores pueden moverse horizontalmente o saltar para subir a un nivel superior.



- **Movimiento horizontal o salto**: Dependiendo de la acción del jugador, se mueve horizontalmente o se sube a un nivel superior.
- **Colisión**: Se comprueba si el jugador colisiona con un obstáculo. Si es así, se resta una vida.

Fase de Bonus

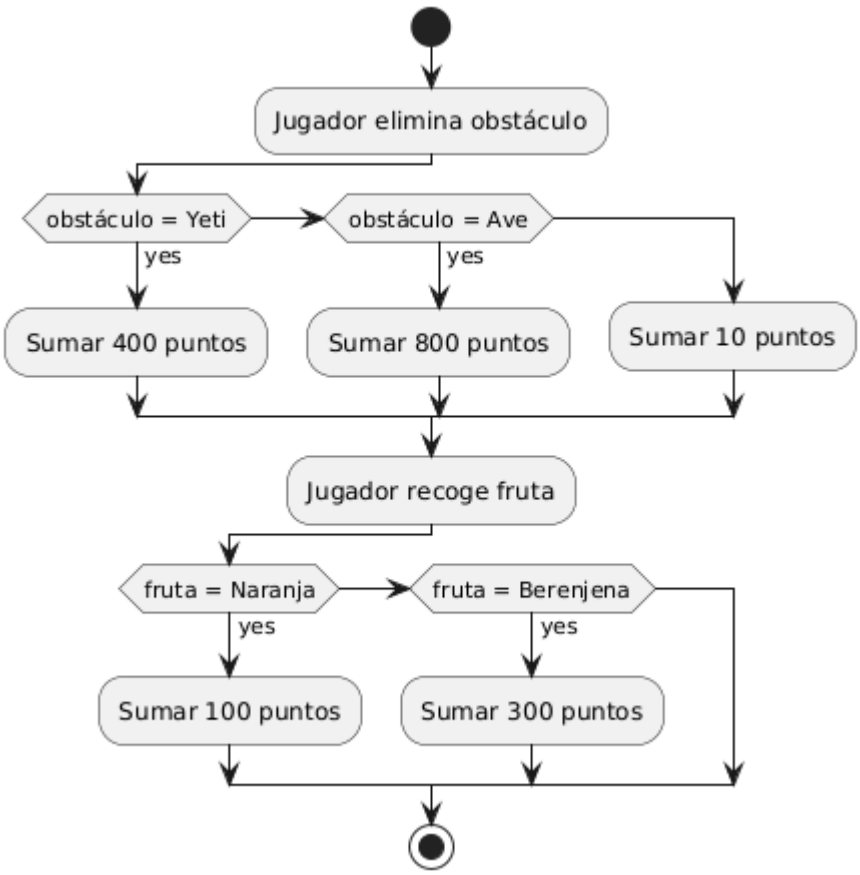
Cuando los jugadores llegan a la fase de bonus, se les otorga una vida extra y los obstáculos se mueven más rápido.



- **Fase de bonus**: Cuando el jugador llega, se le otorga una vida extra.
- **Movimiento de obstáculos**: Durante la fase de bonus, los obstáculos se mueven más rápido.
- **Recolección de frutas**: El jugador puede recolectar frutas que aumentan su puntuación.

Puntuación

La puntuación se calcula cada vez que un jugador elimina un obstáculo o recoge una fruta. Cada tipo de obstáculo tiene un valor asociado.



- **Eliminación de obstáculo**: Se suma una cantidad de puntos dependiendo del tipo de obstáculo.
- **Recolección de frutas**: Se suman puntos al recolectar frutas, dependiendo del tipo de fruta.

1.3. Problemas conocidos

Versión mínima de Java requerida para ejecutar el cliente

- **Descripción**: El cliente del juego está empaquetado en un archivo .jar que requiere como mínimo la versión **Java 21** para poder ejecutarse correctamente. Los usuarios que intenten ejecutar el archivo .jar con versiones de Java anteriores a la 21 recibirán un error indicando que no es posible ejecutar el archivo debido a incompatibilidades con la versión de Java.
- **Intentos de Solución**:
    - Se intentó realizar una compilación del cliente utilizando versiones más antiguas de Java, pero algunas funcionalidades no funcionaron correctamente debido a las nuevas características y mejoras de rendimiento introducidas en Java 21.
- **Solución Encontrada**:

- **Requiere Java 21 o superior**. Asegurarse de que los usuarios tengan Java 21 instalado en sus sistemas es necesario para la correcta ejecución del cliente. Se incluyó un aviso en el manual de usuario y en la documentación para que los jugadores instalen la versión correcta de Java.

1.4. Plan de Actividades realizadas por estudiante

Descripción del Plan de Actividades

Actividad	Responsable	Fecha de Entrega
Análisis de requisitos y diseño del servidor	Fabricio	28/05/2025
Implementación de la clase GameServer y TcpListener	Fabricio	30/05/2025
Implementación de obstáculos (yetis, aves, hielos)	Braulio y Daniel	02/06/2025
Desarrollo de la lógica de conexión cliente-servidor	Fabricio	04/06/2025
Desarrollo de la fase de bonus y frutas	Daniel y Fabricio	04/06/2025
Implementación del cliente en Java	Braulio y Fabricio	05/06/2025
Pruebas de conexión entre servidor y cliente	Fabricio	05/06/2025
Documentación final del proyecto	Todos	06/06/2025
Manual de usuario final	Braulio	06/06/2025

1.5. Problemas encontrados

-Manejo de obstáculos con diferentes velocidades y patrones de movimiento

- **Descripción**: La aparición de obstáculos y su movimiento no estaban sincronizados correctamente, lo que resultaba en un comportamiento errático de los obstáculos. El juego no era completamente predecible en cuanto a la velocidad y patrones de movimiento de los obstáculos.
- **Solución Implementada**:

- Para resolver este problema, se implementó un **temporizador** que controla la velocidad de los obstáculos y asegura que su movimiento sea uniforme durante el juego.

```
// Temporizador para ajustar la velocidad de los obstáculos
private Timer obstacleSpeedTimer;
 // Método para iniciar el temporizador
public void StartObstacleMovement()
{
   obstacleSpeedTimer = new Timer(OnObstacleSpeedUpdate, null, 0, 1000); // Actualiza cada segundo
}
 // Método que se llama para actualizar la velocidad de los obstáculos
private void OnObstacleSpeedUpdate(object state)
{
   // Aumentar la velocidad de los obstáculos según el nivel de dificultad
   foreach (var obstacle in obstacles)
   {obstacle.IncreaseSpeed();
   }
}
 // Método para incrementar la velocidad de un obstáculo
public void IncreaseSpeed()
{ this.speed += 1;  // Aumenta la velocidad de movimiento
}
```

- **Recomendaciones**:
- Mantener la limitación a dos jugadores para evitar problemas. En futuras implementaciones, explorar técnicas de balanceo de carga para gestionar múltiples conexiones de manera más eficiente.
- **Conclusión**:
- El uso de **`lockObject`** y la limitación de jugadores a 2 fue la solución más viable para este problema, pero no permite una escalabilidad total. A largo plazo, se debe pensar en un enfoque más robusto, como servidores dedicados o soluciones en la nube.

-Desincronización en la Fase de Bonus

- **Descripción**: Durante la fase de bonus, los obstáculos no siempre se movían con la velocidad deseada, lo que afectaba la jugabilidad y generaba una experiencia inconsistente para el jugador.
- **Intentos de Solución**:
    - Se intentaron ajustes manuales a la velocidad de los obstáculos sin resultados satisfactorios.
    - Se ajustó el algoritmo de temporización para manejar la velocidad en función de la fase del juego, pero el comportamiento seguía siendo inconsistente.
- **Solución Encontrada**:
    - Finalmente, se implementó un **temporizador sincronizado** para controlar el aumento de velocidad de los obstáculos durante la fase de bonus. Esto se logró ajustando el código en el siguiente fragmento:

```
// Método para ajustar la velocidad de los obstáculos en la fase de bonus
public void AdjustObstacleSpeedForBonusPhase()
{
    if (gamePhase == GamePhase.Bonus)
    {
        foreach (var obstacle in obstacles)
        {
            obstacle.IncreaseSpeed();  // Aumenta la velocidad de cada obstáculo
        }
    }
}
```

- **Recomendaciones**:
- El uso de temporizadores es una solución eficiente. Sin embargo, se recomienda revisar el **algoritmo de gestión de obstáculos** para optimizar el movimiento y la sincronización de todos los elementos en pantalla.
- **Conclusión**:
- La implementación del temporizador ha solucionado la desincronización de los obstáculos, pero el sistema aún podría mejorarse en términos de flexibilidad y adaptabilidad a diferentes tipos de juego.

- **Bibliografía Consultada**:
- "Design Patterns: Elements of Reusable Object-Oriented Software", Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.

1. 6. Conclusiones y Recomendaciones del proyecto

Conclusiones:

- El proyecto proporcionó una buena oportunidad para integrar el paradigma de programación orientada a objetos en un entorno de juego interactivo utilizando Java y C.
- Los patrones de diseño, como el Singleton, ayudaron a gestionar la instancia única del servidor y las sesiones de juego.

Recomendaciones:

- Se recomienda ampliar la capacidad de jugadores simultáneos y optimizar el uso de hilos para mejorar la experiencia multijugador.
- Incluir más patrones de diseño para optimizar la estructura del código y hacerlo más escalable.

1.7. Bibliografía consultada

- "Design Patterns: Elements of Reusable Object-Oriented Software", Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.
- "Java Programming: From Problem Analysis to Program Design", D. S. Malik, Cengage Learning, 2014.

1.8 Bitácora

28/05/2025

- Reunión inicial con el equipo para asignar responsabilidades.
- Fabricio se encarga de la implementación del servidor y la lógica de conexión.
- Braulio se encarga de la implementación de los obstáculos y frutas en el juego.
- Daniel se asigna la documentación y pruebas.

30/05/2025

- Implementación del servidor y el manejo de conexiones con TcpListener.
- Se realizó una prueba básica de conexión entre cliente y servidor.

02/06/2025

- Implementación de obstáculos como Yetis, Aves y Hielos.
- Se realizó la lógica para generar obstáculos aleatoriamente.

04/06/2025

- Fabricio completó la conexión entre el cliente y el servidor.
- Braulio implementó la fase de bonus con frutas.
- Daniel comenzó la documentación del servidor y los algoritmos.

05/06/2025

- Pruebas entre cliente y servidor realizadas con éxito.
- Documentación y pruebas de la interfaz de usuario.

06/06/2025

- Revisión final del proyecto y entrega de la documentación.
- Braulio completó el manual de usuario.