Documentación - Proyecto 1

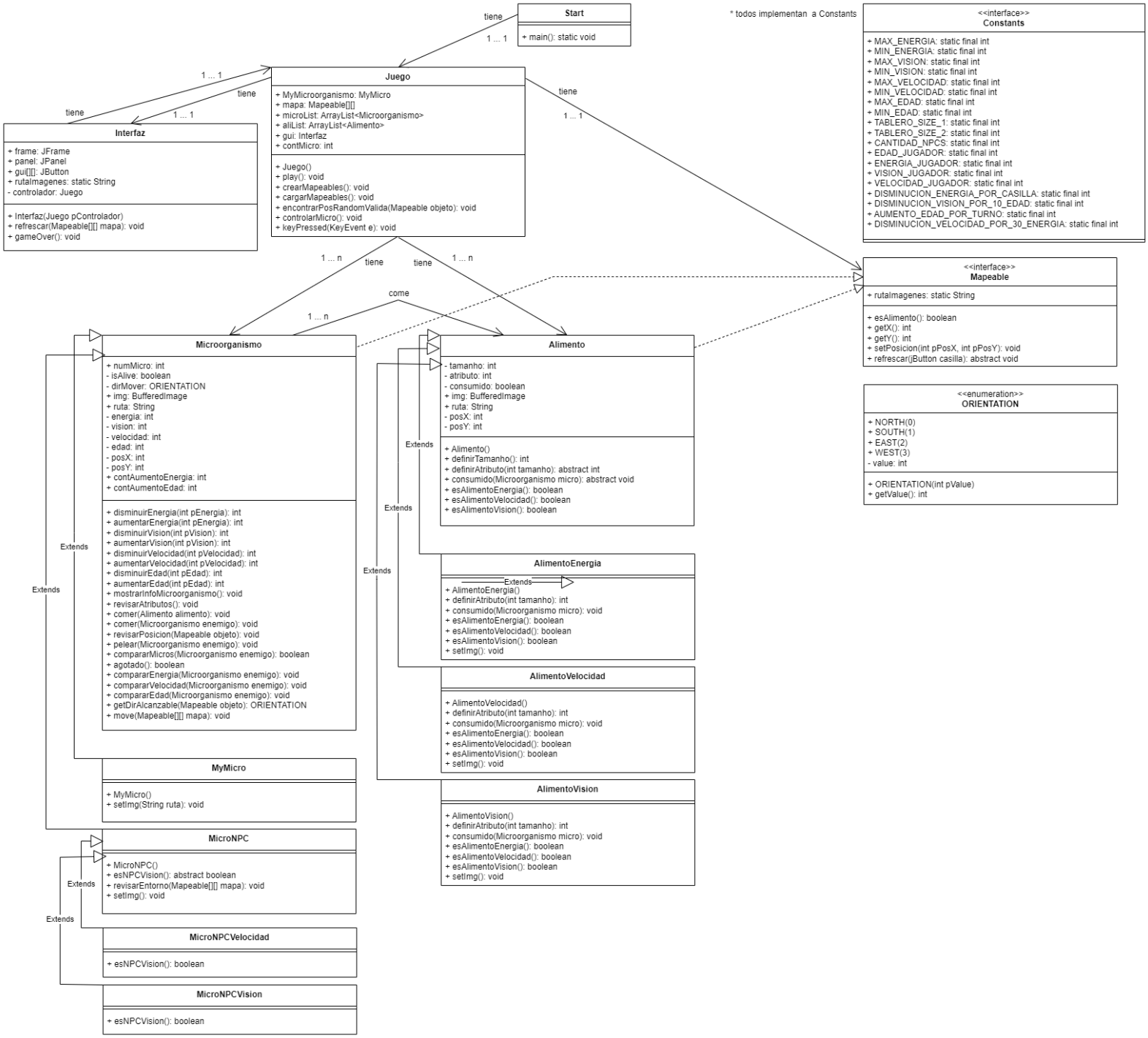
Instituto Tecnológico de Costa Rica - Sede Central Cartago
Programación Orientada a Objetos - Grupo 1
Erick Kauffmann Porcar
Samuel Valverde Arguedas
Prof. Yuen Law Wan
I Semestre 2023

Descripción General

Para la solución de este proyecto se implementaron características base de la programación orientada a objetos: la herencia, el polimorfismo. Además, se implementaron tipos de clases para facilitar y mejorar la calidad del código en general. Con esto se refiere a clases abstractas, interfaces y enums.

Gracias a la solución implementada se pudo crear un juego en el que se manejan dos matrices de tipo Mapeable y JButton de manera simultánea, un ciclo con un algoritmo de comportamiento para los objetos del juego y un objeto controlado por el usuario.

Diagrama de Clases



Explicación de Clases

1. *game*

Juego.java

Se llaman a todos los métodos principales del juego. Se instancian todos los Mapeables, se ubican todos los microorganismos y alimentos en la matriz de juego e inicia el juego.

Microorganismo.java

Clase padre de todos los microorganismos. Les hereda sus atributos básicos.

MyMicro.java

Clase del microorganismo controlado por el jugador. Cuenta con los métodos necesarios para que se pueda mover en base a las decisiones del jugador.

MicroNPC.java

Base de todos los NPCs del juego.

MicroNPCVelocidad.java

NPC de tipo Velocidad. Altera el comportamiento del microorganismo.

MicroNPCVision.java

NPC de tipo Visión. Altera el comportamiento del microorganismo.

Alimento.java

Clase padre de todos los tipos de alimentos del juego.

AlimentoEnergia.java

Tipo de alimento.

AlimentoVelocidad.java

Tipo de alimento.

AlimentoVision.java

Tipo de alimento.

ORIENTATION.java

Enum con las cuatro direcciones posibles y su valor asignado: NORTH(0), SOUTH(1), EAST(2), WEST(3).

Mapeable.java

Interface implementada por todos los microorganismos y alimentos. Contiene la ruta de imágenes para la interfaz, así como los métodos para asignar posiciones en la matriz y refrescar en las casillas.

Constants.java

Interface implementada por todos los microorganismos y alimentos. Se encarga de controlar los valores de comportamiento durante el juego y el tamaño de la matriz, así como los máximos y mínimos de los microorganismos.

2. *gui*

Interfaz.java

Se encarga de crear la interfaz gráfica, recorrer el mapa de juego y actualizar la matriz de botones en base al mapa.

Proceso

La implementación de la herencia y el polimorfismo fue clave para la solución de este proyecto siguiendo el modelo planteado, ya que casi todas las clases implementan al menos uno de estos dos. Todas las instancias del juego, los alimentos y los microorganismos, se heredan de clases padre. Además de heredar atributos base, estas clases implementan métodos abstractos heredados de sus padres y los sobreescriben para que se comporten con respecto a la clase y sus necesidades. Aquí es donde destaca el polimorfismo y se simplifica mucho el código.

Cabe destacar que de manera estratégica todas estas clases implementan interfaces en común que permiten juntar diferentes instancias en arreglos por ser del tipo de la interface, además de que permite que estos objetos también puedan compartir métodos abstractos como es el caso de refrescar() en la interface Mapeable. Gracias a que todos los alimentos y todos los microorganismos implementan esta interface, todos cuentan con este método que les permite refrescar la interfaz gráfica con la imagen asignada al objeto en específico.

Con respecto al polimorfismo se puede apreciar en casos en los que hay un “mismo” método pero pero que puede recibir dos parámetros de tipos diferentes, y según el parámetro se realizan diferentes funciones, como lo es el caso del método “comer(Microorganismo)” y “comer(Alimento)”. Otro ejemplo son los propios métodos abstractos en las clases padres, que se heredan en las clases hijas y según el tipo de clase hija este método se comportará diferente.

Una desventaja de todo esto es que conlleva un nivel de análisis más elevado y por consiguiente una mayor cantidad de tiempo analizando el problema y las diferentes soluciones así como la mejor forma de programarlo, y una vez completado esto se necesita analizar muy bien cómo se complementan y relacionan los diferentes objetos para poder hacer un uso más eficaz de la herencia y el polimorfismo. Sin embargo, la mayor ventaja de estos aspectos es que son muy útiles para evitar la repetición de código innecesario, un estilo de programación sin tanto “if(), else” y una mejor implementación del paradigma de Programación Orientada a Objetos.

Referencias Bibliográficas

Creating a clickable JButton matrix. (2013, 17 junio). Stack Overflow. https://stackoverflow.com/questions/17143338/creating-a-clickable-jbutton-matrix