Proyecto 1
Sweet Swarm

Valerie Michell Hernández Fernández
valeriehernandez@estudiantec.cr
Mariana Navarro Jiménez
mariana12@estudiantec.cr

Instituto Tecnológico de Costa Rica
Escuela de Ingeniería en Computación

I Semestre

Viernes 22 de abril de 2022

Resumen

Mediante el presente documento se pretende ofrecer una síntesis de la funcionalidad del proyecto Sweet Swarm. Este proyecto consiste en desarrollar una simulación de un enjambre de abejas utilizando polimorfismo y herencia para modelar los diferentes comportamientos de las abejas en el enjambre.

Palabras clave: polimorfismo, herencia.

Abstract

This document is intended to provide a summary of the functionality of the Sweet Swarm project. This project consists of developing a simulation of a bee swarm, for which polymorphism and inheritance are used to model the different behaviors of the agents in the swarm.

Keywords: polymorphism, inheritance.

Índice

1. Descripción general

El propósito del proyecto es el de desarrollar una simulación de un enjambre de abejas utilizando polimorfismo y herencia para modelar los diferentes comportamientos de las abejas del enjambre y las cualidades de cada objeto en el entorno. Para la implementación del proyecto se crearon cuatro clases principales ***Honeycomb***, ***SweetSwarm***, ***Bee*** y ***Object***. ***SweetSwarm*** es una clase pública que hereda de ***JFrame***. Esta clase fue diseñada como la principal encargada de implementar los componentes gráficos de la aplicación, así como de inicializar el loop principal del juego. Mediante la clase abstracta ***Bee*** y la clase ***Object*** se generan los diversos elementos de la simulación, como son: las abejas, los enemigos, los obstáculos y los recursos a recolectar. Así mismo, estas dos clases poseen métodos que modelan el comportamiento de dichas entidades.

2. Arquitectura

2.1. Requerimientos

- Java versión 8
- Java SE Development Kit versión 15

2.2. Clases

2.2.1. Diagrama

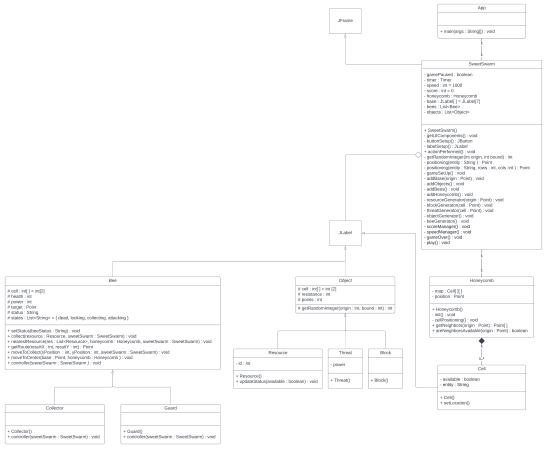

Figura 1. Diagrama de clases UML del proyecto Sweet Swarm.

2.2.2. Descripción

Según [1] el uso del polimorfismo y herencia simplifica los algoritmos desarrollados por los programadores. Esto porque la programación orientada a objetos es un paradigma adoptado por todos los lenguajes modernos y publicados en las últimas décadas. La programación orientada a objetos proporciona una sintaxis en el lenguaje para definir abstracciones que hacen sencillo utilizar conceptos cuya implementación es compleja.

En la necesidad de reutilizar algoritmos de otra clase existe la herencia pues una clase puede heredar los atributos y métodos de la superclase, de esta forma hereda su comportamiento y genera sus propios métodos y atributos. Por otro lado, el polimorfismo es una propiedad por la cual el método invocado varía en función de la clase de la instancia de un objeto [2]. En Sweet Swarm se aplicaron ambos conceptos, en las siguientes secciones se dará detalle de los casos implementados.

App

Clase pública ubicada en el paquete raíz del proyecto encargada de inicializar la clase ***SweetSwarm*** en su método público *main*.

SweetSwarm

Clase pública ubicada en el paquete *game*, la cual hereda de la clase ***JFrame*** de ***Java Swing***. Desarrollada como la clase principal, esta clase es la encargada de la implementación lógica y gráfica de los componentes de la aplicación. Por lo anterior, esta posee desde métodos genéricos para la creación de etiquetas y botones hasta métodos más complejos como posicionamiento de los componentes de tipo ***Bee*** y ***Object***.

La relación de herencia entre la clase ***SweetSwarm*** y la clase ***JFrame*** surge por la necesidad de implementar una interfaz gráfica de la simulación del enjambre. De esta forma, la disposición de todos los métodos de la clase facilita la implementación de los gráficos de la aplicación. Es relevante mencionar que el método *play* es el encargado del ciclo principal de la simulación, recorriendo la lista de las componentes tipo ***Bee***, donde toman decisiones según su entorno. De esta forma el usuario puede apreciar la simulación a la velocidad que desee (por default es 1 segundo por recorrido general). Además, puede pausar y seleccionar haciendo click sobre cada celda para conocer la locación en coordenadas y el tipo de componente que ocupa esta celda.

Figura 2. Resultado gráfico de la ejecución del método *main* de la clase ***App***.

Honeycomb

Clase pública ubicada en el paquete *game honeycomb*, la clase ***Honeycomb*** fue desarrollada con el objetivo de cubrir los aspectos lógicos de la matriz del tablero de la simulación del enjambre. Implementando un arreglo 2D compuesto por entidades de tipo ***Cell***, ***Honeycomb*** cubre el posicionamiento y cálculo de posibles posiciones cercanas a una ***Cell*** especificada. De esta forma, Honeycomb podría verse como un simple tablero y las entidades ***Object*** y ***Bee*** como las fichas que se ubican sobre el tablero. Además, en su constructor hace uso de su método *init*, donde instancia cada ***Cell*** según su posición. En este caso por ser una matriz hexagonal el cálculo de la posición de cada ***Cell*** dependerá de su fila y columna en el arreglo 2D.

Cell

Clase pública ubicada en el paquete *game honeycomb*, la cual hereda de la clase ***JLabel*** de ***Java Swing***. El objetivo principal de esta clase es tener una entidad visible en cada celda del ***Honeycomb***.

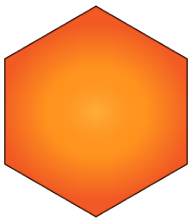

Figura 3. Sprite de la clase ***Cell***.

No obstante, en la parte lógica es primordial pues el arreglo 2D de la clase anterior está compuesto por esta entidad y durante la simulación del enjambre es necesario saber cuales casillas del tablero están vacías, de esta forma se posicionan las entidades de tipo ***Object*** y ***Bee*** durante la ejecución de la simulación y esta clase posee el identificador que determina esa disponibilidad.

Bee

Clase abstracta que hereda de ***JLabel***, ubicada en el paquete *game bee*, encargada de generar los objetos de tipo ***Bee***. Se hereda de ***JLabel*** para utilizar todos los métodos que esta clase facilita para el uso de la interfaz gráfica. Mediante esta clase se definen los comportamientos que comparten las abejas ***Guard*** y ***Collector*** así como el método abstracto *controller* que es único para cada tipo de ***Bee***. Cabe recalcar que mediante el método *nearestResource* se crea un algoritmo similar a pathfinding que genera el movimiento de la abeja.

Guard

Clase pública ubicada en el paquete *game bee*, la cual hereda de la clase ***Bee***. Esta clase le otorga todas las características propias de las ***Bee*** de tipo ***Guard***. De esta misma manera, mediante el método abstracto controller se define el comportamiento único de la ***Bee*** el cual consiste en atacar los objetos de tipo ***Threat***.

Figura 4. Sprite de la clase ***Guard*** en estado de recolección.

Para la implementación de ***Collector*** fue esencial que esta heredara de la clase ***Bee***. Pues al poder compartir métodos con ***Guard*** mediante ***Bee*** la cantidad de métodos que se tuvieron que programar se disminuyo.

Collector

Clase pública ubicada en el paquete *game bee*, la cual hereda de la clase ***Bee***. Esta clase asigna los atributos propios de los ***Collector***. Mediante el método abstracto controller se define el comportamiento de las ***Bee Collector*** el cual consiste en evitar de los objetos de tipo ***Threat***.

Figura 5. Sprite de la clase ***Collector*** en estado de recolección.

Object

Clase pública ubicada en el paquete *game object*, la cual hereda de la clase ***JLabel*** de ***Java Swing***. Durante la simulación las entidades de tipo ***Bee*** toman decisiones en base en su entorno. La entidad del tipo Object (***Resource***, ***Block***, ***Threat***) obliga a las abejas a variar su comportamiento según su tipo. Tal y como se explica en las clases ***Guard*** y ***Collector***. Por otro lado, la herencia de la clase ***JLabel*** surge de la necesidad de que las entidades deban ser mostradas en la simulación. Por lo tanto, para agregar los ***Object*** fue necesario heredar de esta clase así en la clase ***SweetSwarm*** únicamente debían crearse y mostrarse. Es importante mencionar que para los casos de ***Resource*** y ***Threat*** existen diversos sprites, con el propósito de mejorar la experiencia del usuario con la interfaz gráfica de la simulación.

Resource

Clase pública ubicada en el paquete *game object*, subclase de la clase ***Object***. ***Resource*** ofrece una entidad que posee una distribución de 7 ***Cell*** en un Honeycomb con variedad de sprites.

Figura 6. Sprite de la clase ***Resource***.

Cada uno de estos, posee una resistencia de 2 y debe ser recolectado por las ***Bee***, en caso de que estas tomen parte de la flor la entidad mostrara la cantidad restante de partes que le restan mediante un cambio de sprite en la interfaz gráfica. Además, suma puntos a la puntuación total de la simulación.

Block

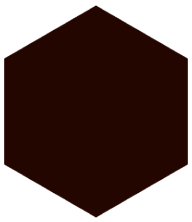

Figura 7. Sprite de la clase ***Block***.

Clase pública ubicada en el paquete *game object*, subclase de la clase ***Object***. ***Block*** implementa una entidad que no puede ser modificada durante toda la simulación. El propósito de esta es bloquear el paso de las ***Bee***.

Threat

Clase pública ubicada en el paquete *game object*, subclase de la clase ***Object*** con variedad de sprites. ***Threat*** proporciona una entidad capaz de infligir daño a las ***Bee***, aparecen durante la simulación y sólo pueden destruidas por las ***Guard***.

Figura 8. Sprite de la clase ***Threat***.

Por lo tanto, en caso de que todas las ***Guard*** hayan sido destruidas por las ***Threat*** la simulación terminará.

3. Recursos

Durante el desarrollo del proyecto se utilizaron recursos externos tanto para el diseño de la interfaz gráfica como para la implementación de los algoritmos desarrollados en la solución final.

3.1. Interfaz Gráfica

- `Javax Swing - Oracle [manual]`
- `Swing Tutorial - Youtube [video]`

3.2. Algoritmos

- `Hexagonal Grids - Red Blob Games [article]`
- `Multidimensional Arrays in Java - Geeks For Geeks [article]`

Referencias

[1] Cecilio Álvarez Caules. *Java Polimorfismo, Herencia y simplicidad*. 2020. URL: `https://www.arquitecturajava.com/java-polimorfismo-herencia-y-simplicidad/` (visitado 20-04-2022).

[2] Picodotdev. *Los conceptos de encapsulación, herencia, polimorfismo y composición de la programación orientada a objetos*. 2021. URL: `https://picodotdev.github.io/blog-bitix/2021/03/los-conceptos-de-encapsulacion-herencia-polimorfismo-y-composicion-de-la-programacion-orientada-a-objetos/` (visitado 20-04-2022).