Snake-AI
Project Documentation

Ibrahim Enes Hayber, Rana MD Jewel, Maximilian Lüttich
Frankfurt University of Applied Sciences
Faculty 2, Computer Science (B. Sc.)
Object-oriented Programming in Java by Prof. Dr. Doina Logofatu

January 14, 2023

Contents

1 How we started **4**

2 Introduction **5**
2.1 General Topic 5
2.2 Similar problems in practice (References every time, look for actual ones) 5

3 Team Work **5**
3.1 Work Participation 5
3.2 work structure (communication, decisions, bug tracking, repository, engineering, ...) 5
3.3 ideas (brainstorming) 5

4 Problem Description **5**
4.1 Formal description: definitions, examples, ... 5

5 Related Work **6**
5.1 Related Algorithms, Applications 6

6 Proposed Approaches **6**
6.1 Input/Output Format, Benchmarks (Generation, Examples) . 6
6.2 Algorithms in Pseudocode 6

7 Implementation Details **6**
7.1 Application Structure 6
7.2 GUI Details 7
7.2.1 SnakeUIMain 7
7.2.2 BotLoader 7
7.2.3 SnakesWindow 8
7.2.4 SnakesRunner 8
7.2.5 SnakesGame 8
7.2.6 SnakeCanvas 9
7.2.7 Snake 9
7.2.8 Coordinate 9
7.2.9 Bot 10
7.3 UML Diagram 10
7.4 Used Libraries 10

8 Experimental Results and Statistical Tests **11**
8.1 Simulations (Play with the Parameters!) 11
8.2 Use Benchmarks 11
8.3 Tables: input data details, results of different algorithms . . . 11
8.4 Charts 11
8.5 Evaluations 11

9 Conclusions and Future Work **11**
9.1 How the team work went 11
9.2 What we have learned 11
9.3 Ideas for the future development of your application, new algorithms 11

1 How we started

This chapter describes how we started with the project.

As we all know the hardest part of a journey is the start. But once started it is a easy on going. It is like the starting impact on the first domino stone, which brings the whole project in rolling.
As usual in any task of life we need to understand what the problem is and which requirements are necessary to solve the problem. To do so, we read the description from the organizers carefully. In addition to that we watched a few videos about our topic to get a better understanding.
After gaining knowledge about the problem we felt ready to start working with the Snake-AI project provided by the organizers on a GitHub repository. The Project was written in the Java programming language. To get familiar with the project we looked into the UML class diagram. Besides that we followed a YouTube Tutorial which was also provided by the organizers, where we implemented our first working version. A detailed introduction and our results will now be presented.

2 Introduction

to-do

2.1 General Topic

to-do

2.2 Similar problems in practice (References every time, look for actual ones)

to-do

3 Team Work

to-do

3.1 Work Participation

to-do

3.2 work structure (communication, decisions, bug tracking, repository, engineering, ...)

to-do

3.3 ideas (brainstorming)

to-do

4 Problem Description

to-do

4.1 Formal description: definitions, examples, ...

to-do jflafjlajflajfla

5 Related Work

to-do

5.1 Related Algorithms, Applications

to-do

!for example: Fractals, Data Generation, Games, Evolutionary Algorithms (Genetic Algortihms; Collective Intelligence like Particle Swarm Optimization - PSO, Ant Colony Optimizaton - ACO; Evolutionary Multiobjective Optimization, ...)

6 Proposed Approaches

to-do

6.1 Input/Output Format, Benchmarks (Generation, Examples)

to-do

6.2 Algorithms in Pseudocode

to-do

7 Implementation Details

to-do

7.1 Application Structure

The Snake AI project consists of one Interface, one Enum and eight classes. They are following,

- Bot(Interface)
- Direction(Enum)
- BotLoader
- Coordinate

- Snake
- SnakeGame
- SnakeCanvas
- SnakesRunner
- SnakeUIMain
- SnakesWindow

7.2 GUI Details

In this section all the classes will be discussed in details.

7.2.1 SnakeUIMain

This class is the Entry point of the UI and implements tournament of the Snake game with several rounds. The main method of the class gets two classes(snake & opponent) implementing the Bot interface as arguments. The class is also responsible to launch several rounds of snake game between bots. In this class the basic I/O actions are performed to record scores of opponent, apple eaten by both the snakes and time taken in each rounds which will be later used to make further statistical analysis.

7.2.2 BotLoader

The class BoatLoader fetches the class given the name of the class and package. The class is taken from the classpath and could be dynamically added after the game of the Bot class. This class inharites the abstrack Java class ClassLoader.
A class loader is an object that is responsible for loading classes. The class ClassLoader is an abstract class. Given the binary name of a class, a class loader should attempt to locate or generate data that constitutes a definition for the class. A typical strategy is to transform the name into a file name and then read a "class file" of that name from a file system.(to-do code snippet)
The only method of BotLoader class gets classBinName as argument and returns an instance of the Bot class.

7.2.3 SnakesWindow

This class is responsible for the game's GUI window. The class creates and set ups the window and runs the UI. At the end it closes the frame. This class implements Runnable.
The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. The class must define a method of no arguments called run.
This interface is designed to provide a common protocol for objects that wish to execute code while they are active. For example, Runnable is implemented by class Thread. Being active simply means that a thread has been started and has not yet been stopped.
In addition, Runnable provides the means for a class to be active while not subclassing Thread. A class that implements Runnable can run without subclassing Thread by instantiating a Thread instance and passing itself in as the target. In most cases, the Runnable interface should be used if you are only planning to override the run() method and no other Thread methods. This is important because classes should not be subclassed unless the programmer intends on modifying or enhancing the fundamental behavior of the class.(to-do ref)

7.2.4 SnakesRunner

This class also implements Runnable interface and is used for running bots in a separate threads. The Constructor of the class SnakesRunner recieves running bot, snake that is controlled by the current bot, opponent's snake, size of the board and coordinate of the current apple as arguments. In addition the chooseDirection method of the current bot is executed and saved(current direction) by this class.

7.2.5 SnakesGame

The class SnakesGame implements the main games flow and runs game for two bots. The Override toString() method converts the game to string representation and return game state as a string. randomNonOccupiedCell() method selects random non-occupied cell of maze which is used to create new location of a apple. If a Snake takes more than one seconds to choose its next direction the runOneStep() methods stopps that rounds.

7.2.6 SnakeCanvas

The purpose of this class is to design the UI and give the UI a better look. This class helps to fill the snake Body wicht color and make it visible on the UI. After filling the body of snake and opponent and apple the render() method renders the game. After every movement of the snake, opponent and apple the previous occupied positions are recolord and marked the free for next movement. This class inharites the Java swing class JPanel which is used to create different generic lightweight container with one or more single component.

7.2.7 Snake

This class implements snake body(not brain) that determines the place of head, body, and length of body on the game board. For implementing the body of the snake two logical and efficient data structure from Java Collections framework are used. They are,

- Dequeue
- HashSet

Deque(double ended queue) data structure helps to add or delete elements from both side(head and tail) within $\mathcal{O}(1)$ constant time coplexity. On the other hand HashSet data structure ensures basics search operation in $\mathcal{O}(1)$ time complexity.
This class has a important feature of cloneability which allows to copy one object to another object without using new operator.
A class implements the Cloneable interface to indicate to the Object.clone() method that it is legal for that method to make a field-for-field copy of instances of that class. Invoking Object's clone method on an instance that does not implement the Cloneable interface results in the exception CloneNotSupportedException being thrown.(to-do ref)

7.2.8 Coordinate

Coordinates class implements position of a cell on the game board. It helps to grow up the snake by adding new coordinate at the beginning of the snake body. inBounds() methods helps to keep the snake in side the game board. This methods implements Comparable interface which is useful while comparing two Coordinate.

7.2.9 Bot

This Interface provides functions that should be implemented to create smart and intelligent snake bot for the game. chooseDirection() methds returns an appropriate next moves which plays a vital role to continue the game.

7.3 UML Diagram

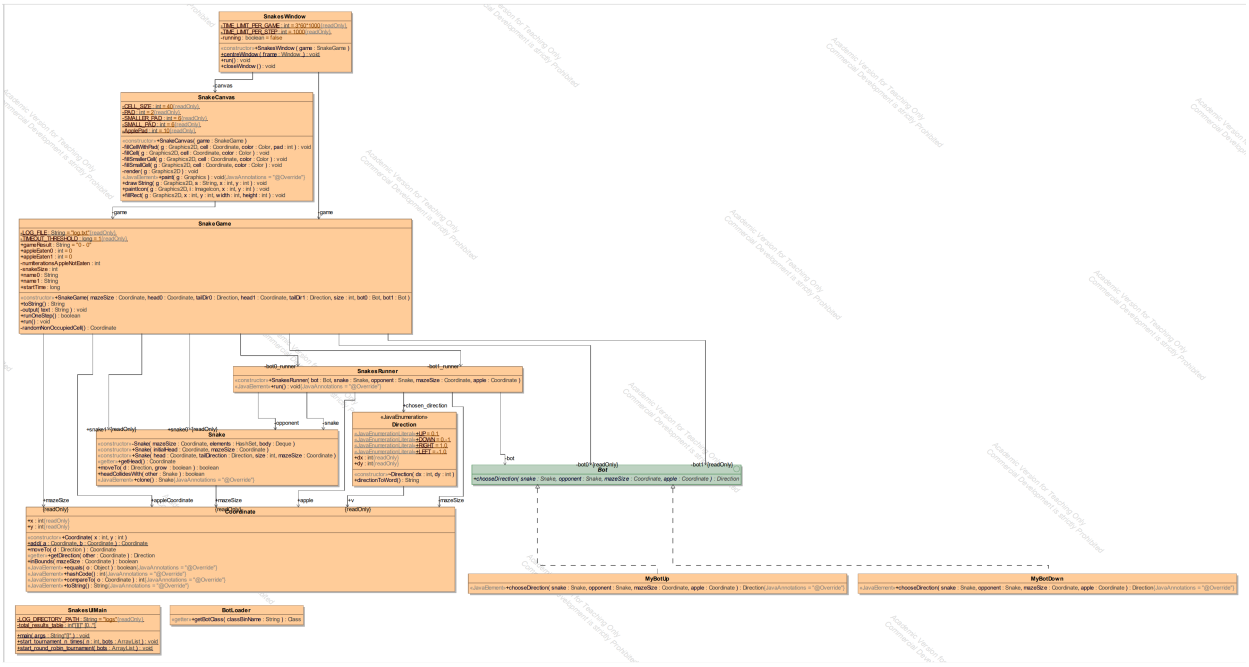

Figure 1: UML diagram of the GUI

7.4 Used Libraries

For the Developement of the Snake AI Game Java Swing Framework has been used. Some Components from AWT framework have also been used. Swing is a Java Foundation Classes [JFC] library and an extension of the Abstract Window Toolkit [AWT]. Swing offers much-improved functionality over AWT, new components, expanded components features, and excellent event handling with drag-and-drop support.(to-do ref)

8 Experimental Results and Statistical Tests

to-do

8.1 Simulations (Play with the Parameters!)

to-do

8.2 Use Benchmarks

to-do

8.3 Tables: input data details, results of different algorithms

to-do

8.4 Charts

to-do

8.5 Evaluations

to-do

9 Conclusions and Future Work

to-do

9.1 How the team work went

to-do

9.2 What we have learned

to-do

9.3 Ideas for the future development of your application, new algorithms