CS 251 Intermediate Programming
Snake Game: Part 1 – Game Manager

Brooke Chenoweth

Spring 2022

In this lab, you will begin work on a snake game. You will be working on internal logic and bookkeeping independent of the game GUI which you will write in the next assignment.

Game Description

In a snake game[1] such as Nibbles[2] the objective is to navigate a snake through a walled space (or maze), consuming food along the way. The user must avoid colliding with walls or the snake's ever-growing body.[3] The length of the snake increases each time food is consumed, so the difficulty of avoiding a collision increases as the game progresses.

Problem Specification

You are going to create a `GameManager` class to handle your snake, wall, and food bookkeeping.[4] You will demonstrate its functionality in a `GameManagerTester` class. The `GameManager` and related classes should be independent of any future GUI code.

You will configure your GameManager object by reading in a level configuration file.

Level File Format

A level configuration file specifies the size of the game area and the locations of the walls. It is a text file of numbers separated by whitespace.

The first line in the file contains two integer values specifying the width and height of the game area. If there is additional data on this line, it may be ignored.

The following lines (if any) specify the walls. A wall line contains contains four integer values: the first two are the x and y position of the upper left corner of the wall, the second

[1] Try searching for "snake game" or "play snake" on Google.

[2] see `https://en.wikipedia.org/wiki/Nibbles_(video_game)`

[3] Or other snakes, in a multiplayer game.

[4] You will probably want some additional classes/interfaces to accomplish this, but I leave that design up to you.

two are the x and y position of the the lower right corner of the wall. If there is additional data on this line, it may be ignored.

If you want to add additional information to your level configuration files, you may do so, but you must still be able to handle the basic file format described above. I have provided you with some sample input files on the course website.

`GameManager`

Although you may chose to use one fixed level design for your final game, your class should be able to handle arbitrary positive values for width and height of the playing area.

Your class must provide the following functionality.

- Construct a new manager of a given size.
- Add walls in specified locations, as read from a level configuration file.
- Add at least one snake at some unoccupied location.
- Add at least one food item at some random unoccupied location.
  - The final game will use random placement for the food, but for this test program you should have reproducible results each time it is run.
  - One simple way to have "random" behaviour that can be repeated is to use a `Random` object that has been initialized with a fixed seed.
- Override `toString` to provide a string representation of your board. Make it clear what locations are walls, food, snake, and empty space.
- Update snake position one step and check for collisions.
  - If the snake collides with food, the snake eats the food (removes it from game) and grows in length. A new food item is placed in at some random unoccupied location.
  - If the snake collides with a wall or its own body, detect this somehow. In the full game, the game would be over[5], but for the testing code, we'll just report the collision.

I fully expect that much of the GameManager logic will be handled by additional game object classes, such as Snake, Wall, and Food. You may want to make some or all of these classes extend a common abstract parent or implement a common interface.

[5] or maybe lose a life, depending on your game design

`GameManagerTester`

The purpose of this testing class is to demonstrate to us that your game manager works correctly, so you need to put it through its paces.

- You should create at least two `GameManager` objects with separate sizes and wall configurations. At least one of them must be initialized from a level configuration file. Demonstrate that they operate independently of each other. (I want to make sure you aren't locked into just one configuration and that you aren't making things static inappropriately.)
- Add at least one food item and snake to each manager.
- Test moving the snake around in each manager.
  - Print out the string representation of the manager before you begin.
  - Update snake position and print the manager after the update.
  - Demonstrate changing snake direction.
  - Demonstrate eating food and show that snake grows.
  - After testing the each manager, continue testing with the first manager again to demonstrate that it is independent of the second.
  - Demonstrate collisions of the snake with a wall and the snake with itself.
- Don't just copy and paste the same test code for each manager. Structure your code sensibly with methods rather than writing one giant `main` method.
- Your testing output should come from `GameManagerTester`, not `GameManager`.[6] Don't put console output in the middle of your game logic.

I am providing you with example output from my testing code to give you an idea what I'm expecting this to look like.

Turning in your assignment

Submit your `GameManager.java`, `GameManagerTester.java`, and any other source files you used for the project to UNM Learn.

If you are submitting more than two files, please consolidate them all into a zip file and submit that to Learn so that the graders will not have to download a large number of files individually.

[6] or other game object classes