The Snake Game Java Case Study

John Latham

January 27, 2014

Contents

1 Introduction

This case study presents much of the development of a program to play a snake game, similar to that found on certain old mobile phones. The core game playing functionality is actually left as a staged laboratory exercise.

2 Learning outcomes

The intentions of this case study are as follows.

- To reinforce many of the Java and programming concepts you have already met.
- To provide valuable experience of the design and implementation of a large program.
- To provide a framework for a more challenging, and thus rewarding, laboratory exercise.

Whilst there is no substitute for writing your own programs, watching the development of another's is still an extremely effective way of collecting design and programming experience, particularly as the programs can be a little more challenging than those you would be able to write yourself at this stage. (Caveat: this is only true for case studies that you are *not* supposed to be doing yourself – watching someone else develop code that you are supposed to be doing on your own is disastrous for your learning!)

How much you get from this case study depends on how much you put in. Ideally you should carefully follow every part of it and check your depth of understanding at every opportunity. All confusions or misunderstandings which it may reveal must be dealt with immediately.

3 Description of the game

The game is similar to snake on mobile phones (which is similar to a game played on Unix over 30 years ago), but with some 'improvements'.

The game is played by one player, who has the objective of obtaining the highest score possible. The player is in control of a snake which is constantly moving around a square field of cells. The length of the snake is a whole number of cells. At any time, the snake moves in one of the 4 directions, parallel to a side of the square, and the player can change the direction using the 4 arrow keys of the keyboard. If the snake crashes into a side, or into itself, then it is dead, and the game stops.

Also in the field is a single piece of food. When the head of the snake hits this, the food is eaten, and the snake becomes one cell longer. This event increases the score of the player. At the same time, a new piece of food is placed in a randomly chosen cell somewhere in the field, which was previously clear (empty).

The game has a score message bar, informing the player what is the current score, and also a single message which changes from time to time. For example, when food is eaten the player is informed how much score was just obtained, and when the snake crashes a suitable message is shown.

The player can make the game go slower or faster, and can alter the speed during the game, by pressing ‘s’ to slow down and ‘f’ to speed up. The score obtained when a piece of food is eaten is proportional to the speed setting. The game can also be paused and resumed by pressing ‘p’ and ‘r’ respectively. The game will be in a paused state when it starts. The speed controller can be placed in interactive mode by the player pressing ‘i’ – this enables the player to see the current speed and also alter it via buttons.

At any time, the player can end the game and start a new one, simply by pressing ‘a’ on the keyboard. The most common use of this will be after the snake has crashed, so as to continue playing with a new game.

When the snake is dead, the player has the option to ‘cheat’ by making it come back to life so play can continue. However, this costs the player half of his or her score. The cheat feature is invoked when the player presses ‘c’ on the keyboard.

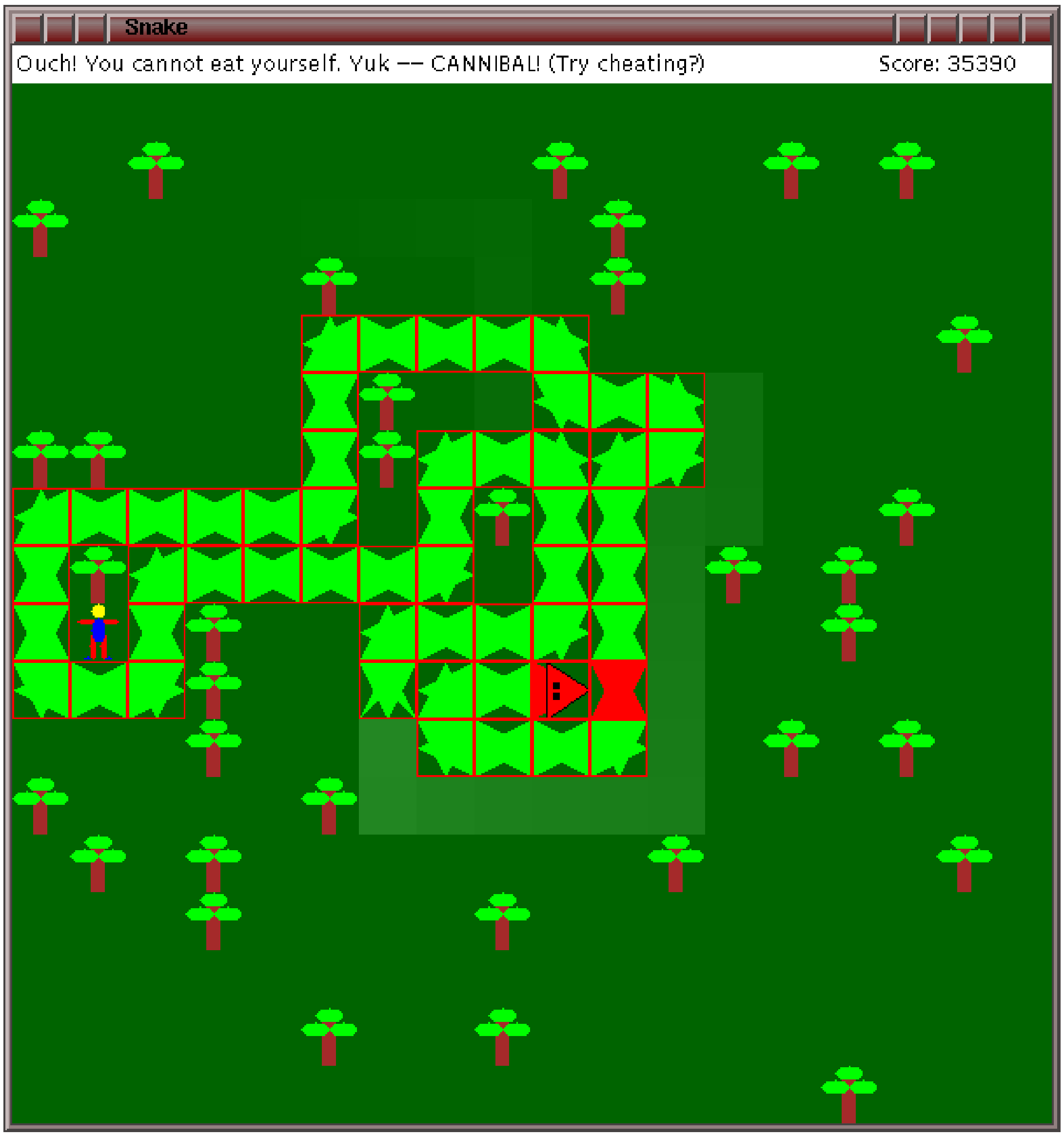
The player has the option of enabling trees to appear in the field. This feature is toggled on and off by pressing ‘t’ on the keyboard. When trees become enabled, a single tree appears in a randomly chosen clear cell. Then, each time the food is eaten, another tree appears somewhere, and so on. This makes the game more interesting, because the snake will die if it crashes into a tree. The game is thus more difficult, and so if trees are enabled when the food is eaten, the score obtained is multiplied by the number of trees in the field at the time. When the trees feature is toggled off, all the trees vanish.

Each time the snake crashes into something, it will not die immediately. Instead, a ‘crash countdown’ begins, and reduces by one each move time of the game. The player sees this count down via the score message bar. If the snake is moved away before the count down reaches zero, it has escaped death.

The game offers a single ‘about’ and ‘help’ box, popped up and down by pressing ‘h’.

There is no requirement for keeping track of high scores.

Figure 1 shows a sneak preview of the finished game program.



Ouch! You cannot eat yourself. Yuk -- CANNIBAL! (Try cheating?)

Score: 35390

Figure 1: A screen dump of a game. The optional extra feature of a gutter trail is visible behind the snake.

4 The classes of the program

As is usual in the object oriented approach, the program will be developed as a number of separate classes. Identifying the appropriate classes of a program is really an art which can slowly be learned from experience. And like the result of any art, the quality of any choice of classes is subjective (although most people will agree there are many more bad choices then good ones!).

By thinking about our understanding of the game, we can form a list of classes which will appear in the program.

Class list for Snake Game: main model classes

Class	Description
`Cell`	The game grid is made up of cells. Each cell has a type and a number of attributes. For example, a cell could be clear, or it could be a snake head facing in a particular direction, etc..
`Direction`	The snake in the game moves in one of four directions. This non-instantiable class provides those four directions, and various methods to manipulate them (such as obtaining the opposite of a direction).
`Game`	This is the main behaviour of the game play. Its primary purpose is to provide a method for making one move of the game.

We will also require some other classes to support the operating framework, including the following.

Class list for Snake Game: operating support classes

Class	Description
`GameGUI`	This will provide a user interface so that the player can see the snake and the food, etc., and be able to operate the game via the keyboard. It will also be the driving force for the game, that is, it shall contain a method to play the game by repeatedly invoking the move method from the `Game` class.
`SpeedController`	A speed controller is something which controls the speed of the game, to make it easier or harder for the player as they choose. This will be achieved rather simply by the program calling a method to cause a delay between every move. A controller will have various speed settings, the ability to increase or decrease speed, and to pause and resume the game.
`AboutBox`	This will provide a pop-up box which can display information about the program and help on how to use it.

Finally, we shall need a class to contain the main method.

Class list for Snake Game: top level class

Class	Description
`Snake`	This is the top level class, containing only the main method. This shall simply create an instance of `GameGUI`, invoke its game playing method and print out the score once the game has finished.