Embedded Programming – Play the Classic Snake Game in a Terminal in Linux

Chia-Man Hung
CDT AIMS, University of Oxford
chia-man.hung@eng.ox.ac.uk

December 14, 2017

Abstract

Snake is a very classic game in which the player controls a zigzag line resembling a snake on a bordered plane. The snake grows in length as it eats, with its entire body being a primary obstacle. The concept dates back to the arcade game Blockade in 1976 and has been popularized on Nokia mobile phones since 1998. In this report, we implement a simple version in C++ that can be played in a terminal in Linux.

1 Introduction

This is the assignment from an one-week long Embedded Systems Programming course[1] in which we learned to program in C++. To illustrate what is learned in the course, we implement a simple classic Snake game in C++. The code can be found on my github[2].

In the following, we first describe the main features of the game. Then, we explain how it is implemented and we show a screenshot.

2 Controls & Design

A snake of initial length 5 moves in a 36 x 15 plane along a direction. A fruit is placed randomly on one of the empty cells. Whenever it is eaten, the snake's length is increased by 1 and another fruit is placed randomly again. The game is over when the snake hits an obstacle (the border or itself). The player controls the direction of the snake – up (w), down (d), left (a), right (d). A score, the total number of fruits eaten, is displayed.

Some additional details are listed below.

- Acceleration: The snake moves one step every 300 milliseconds. If the player presses a direction button before the step time is reached, the snake will also move along the changed direction. This is checked every 10 milliseconds.

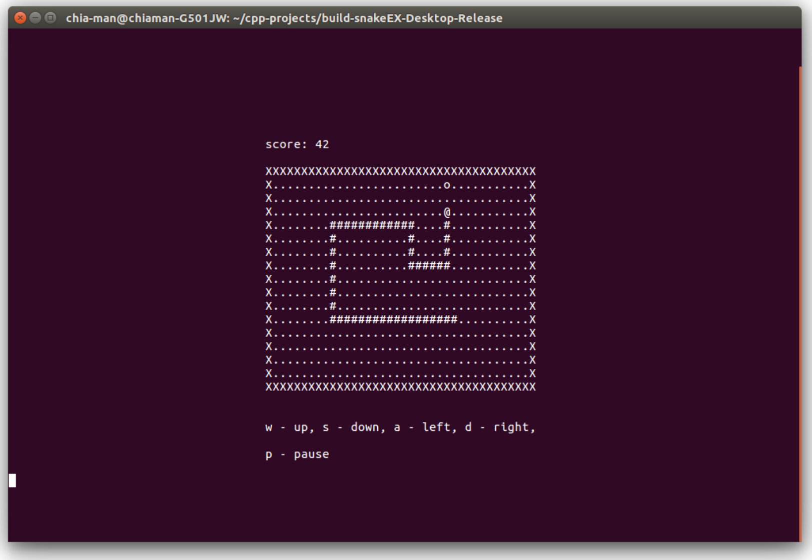
[1] http://aims.robots.ox.ac.uk/embedded-systems-programming-mt2017/

[2] https://github.com/ascane/snake

- Pause: The p button pauses and unpauses the game.
- Opposite direction disabled: When the snake is moving in a direction and a button of the opposite direction is pressed, it has no effect. For example, if the snake is moving upward, the down button has no effect.
- Game board centered: The width and the height of the terminal is taken into account and the game board is centered. This is done every time before we draw the game board.
- Different symbol for head: We use the @ symbol to show the position of the snake's head to avoid any confusion of head and tail.

3 Implementation

main.cpp creates an instance of the *Game* class, defined in *game.h*. *Game* encapsulates an instance of the *Map* class, defined in *map.h*, and an instance of the *Snake*, defined in *snake.h*. The input control functions are defined in *input.h*. When a game is running, it checks if an effective button is pressed and acts accordingly. The cell type (empty, wall, snake head, snake body, fruit) is stored in the associated map. The snake also stores a pointer to the map. When the snake moves, the map is updated.



chia-man@chiaman-G501JW: ~/cpp-projects/build-snakeEX-Desktop-Release

```
score: 42

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
X........................o...........X
X....................................X
X........................@...........X
X........############....#...........X
X........#..........#....#...........X
X........#..........#....#...........X
X........#..........######...........X
X........#...........................X
X........#...........................X
X........#...........................X
X........##################..........X
X....................................X
X....................................X
X....................................X
X....................................X
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX


w - up, s - down, a - left, d - right,

p - pause
```

Figure 1: Screenshot of a mid-game of Snake.

4 Conclusion

We described the design of a classic Snake game and showed an working implementation in C++.