Instruction for project 2

Podstawy Programowania & Basics of Computer Programming 2024/25

author: Dariusz Dereniowski[1]

date: 24/10/2024

Snake Project

The goal

The goal of the project is to implement the "Snake" video game. An exemplary gameplay can be found e.g. here:

https://www.youtube.com/watch?v=THbKk0hMGDo

A a somewhat detailed description of the game and its history can be found e.g. here:

https://en.wikipedia.org/wiki/Snake_(1998_video_game)

The Programming Environment

The instruction comes with a helper program that implements: the time increment calculation, which allows you to track its passage, displaying graphic files in the BMP format on the screen, drawing pixels, lines and rectangles, and text display. These should cover enough of the SDL2 (see below) functionality to implement the project.

The program utilizes the SDL2 library (2.0.3) – http://www.libsdl.org/. It is included in the helper program and there is no need to download its sources. The helper program illustrates all tools of the library needed to implement this project. When uploading the program do not upload the SDL2 library.

Compilation under Linux is performed using the following command (in a 32-bit system):

```
g++ -O2 -I./sdl/include -L. -o main main.cpp -lm -lSDL2 -lpthread -ldl -lrt
```

or (in a 64-bit system):

```
g++ -O2 -I./sdl/include -L. -o main main.cpp -lm -lSDL2-64 -lpthread -ldl -lrt
```

1 Note: In case of any ambiguity in the instruction, please contact the author, either by email or directly. The deadline for instruction verification is November 10th.

In order to successfully compile the helper program, the main.cpp file should be in the directory with
- bitmaps with required drawings (cs8x8.bmp, eti.bmp);
- libSDL2.a file (libSDL2-64.a for 64-bit compilation);
- sdl directory attached to the program.

The helper program includes scripts that can be used for compilation (comp in a 32-bit environment and comp64 in a 64-bit environment).

The presentation of the program (required if you want to get any points) will take place in an environment chosen by you from the following two options:
- Linux: you should check, before coming to the examination, whether the program compiles and runs correctly under the distribution available in the laboratory;
- Windows: in the MS Visual C++ environment in a version compatible with the one available in the laboratory.

Showing that your program runs (during the presentation) is a necessary condition for obtaining any points.

The C++ **stl** library should not be used in your program (do not use *string*, *vector* etc.) You can use other C++ features like classes, operator overloading. You can use a data structure of vector and such if you implement it yourself.

Mandatory Requirements (5 points)

All items listed here must be implemented. The absence of any of the following elements results in no points.

1. Preparing the graphic design of the game: outline of the board, preparing a place to display additional information (e.g. time that has passed since the beginning of the game), points etc.. Control key implementation:
   a. ***Esc***: exit from the program - the program ends immediately;
   b. ***'n'***: new game.

2. Implementation of the basic movement of the snake by using arrow keys. Here it is enough when the snake has fixed (initial) length and you can control its movements up, down, left and right. Do not implement lengthening of the snake here. In this project do not implement the wrapping of the board. By wrapping we mean the effect of crossing from the right/left (respectively up/down) side of the board to the opposite side. Hence the user navigates the snake so that the edges of the board are not crossed. If the snake hits an edge of the board and the player does not make a turn, then the snake should turn in a valid direction (to the right if possible, otherwise to the left).

3. Implement detection of the snake hitting itself. When this happens the game should end giving an option of quitting (*Ecs*) or starting a new one (‘*n*’).

4. The elapsed time should be displayed along with the information about implemented requirements (mandatory and optional). For the latter, the program should give a list of points (1-4 listed here and A,B,C,… listed below). Please list only fully implemented requirements.

Optional Requirements (11 points)

Before you start implementing the following, please implement first the required 1-4 points above.

A. **(1 pt) Lengthening of the snake**. Implement a random blue dot that is placed on the board. Whenever the snake `eats' the dot, it gets one unit longer and a new random dot appears on the board. Note that the detection of the game ending of the snake hitting itself is already implemented in the mandatory requirements so it should continue working when adding any optional requirements including this one.

   [Dependency: none]

B. **(1 pt) Speedup.** After a fixed time interval (as usually, to be determined by an appropriate constant in the program) the game speeds up by a certain factor (for example 20% but again, make it easy to be changed).

   [Dependency: none]

C. **(2 pt) Red dot bonus.** At random times, a red dot appears and is displayed on the board for a fixed time interval. This time interval should be indicated by a `progress bar' somewhere next to the board. The bonus gives randomly one of two things: either shortens the snake by a given number of units or slows down its movement by a given factor (not necessarily the same as the speedup factor set in B).

   [Dependency: implement only if A and B have been implemented]

D. **(1 pt) Points**. Implement counting the points that the player can get in the game. Points are obtained each time the snake eats a blue or a red dot.

   [Dependency: implement only if A, B and C have been implemented]

E. **(2 pts) Saving and loading the game**. Hitting the key `s' saves the current state of the game into a file, without interrupting the actual game. Hitting `l' at any point loads the lastly saved state from the file. If no game has been saved before, then hitting `l' has no effect. Saving the state of the game should include all its components (the current state of the board, number of points, the elapsed time, current speed, and the progress bar regarding the red dot if it is present at the moment of saving the game, and teleportation elements introduced in H if they have been implemented).

   [Dependency: implement only if A, B, C and D have been implemented]

F. **(1 pt) Best scores**. The program should keep (in a file) three best scores ever achieved. They should be displayed at the end of each game. If the game ends with a score that qualifies for the best crores' list, then the program should ask the player for a name and update the 3-element list of scores accordingly.

[Dependency: implement only if A,B,C and D have been implemented]

G. **(1 pt) Fancy graphics effects.** Give the snake differently looking head and tail (differently from its body) and an `animated' body. For the body do the following: each „cell" of the body should have one of two sizes (smaller and bigger) and they alternate. So, say the cell right after the head is bigger, the cell after that one is small and so on. The blue and red dots should be `pulsating' by shrinking and extending in size.

[Dependency: implement only if A,B and C have been implemented]

H. (1 pt) Teleportation. Put a few randomly placed pairs on the board (the number of pairs should be naturally easily adjustable by a constant). For each pair: it has two elements, each of which occupies one cell of the board. Both elements of the i-th pair should be marked with number i, so that the player can see which cells belong to which pair. Whenever the snake hits an element of any pair, it starts reappearing at the other element of the cell. Note that his is not a standard part of the snake game.

[Dependency: implement all previous requirements A-G]

I. **(1 pt) Game configurator.** Prepare your own format of a text file that keeps the following game parameters: initial game speed, initial snake length, frequency of the bonus, the number of points for hitting each type of dot and board dimensions, speed up factor, slow down bonus factor. After editing and saving the file, the subsequent execution of the program should load these parameters and run the game according to the parameters' values. The configurator does not need to include the teleportation elements from H, i.e., we keep them random.

[Dependency: implement all previous requirements A-H]

Extra Requirements (2 points)

(2 pts) Auto player. Implement a simple auto player that should follow the following strategy. Whenever the snake eats the blue ball, the snake should consider one of two[2] possible paths to the next blue ball: first go left or right until reaching the column with the blue ball, then go up or down to the ball, or first go up or down until reaching the row with the blue ball and then go either left or right. Among those two options pick any that gets the snake successfully to the next blue ball. If none of them gets the snake to the blue ball then stop the game displaying the information that the auto player fails. When the red ball appears, go for it if one of the two paths (measuring from the current location can successfully get the snake to the red ball --- if not then simply ignore the red ball and continue to the blue one).

[Dependency: implement all mandatory and all optional requirements]

2 There are only two paths, not eight, because there is no „wrapping" of the board.

Final Remarks

- Graphic design requirements: it is sufficient to use any bitmaps (appropriately selected in terms of their sizes).

- The program configuration should enable easy change of all parameters, not only those clearly indicated in the above description. By easy change we mean modifying a constant in the program. For example, by changing appropriate constants defined in the program it should be possible to change the dimensions of the board, speed of the game, initial snake length, initial position of the snake, frequency of the appearance of the red dot, positions of different items on the screen (like the board, points), the number of points for getting each ball and others.

- The program can be written in an object-oriented way, but it is completely forbidden to use the C++ standard library (including the types string, cin, cout, vector, etc.) (Note: the string type from the C++ library should not be confused with the string.h library from C – you can use functions located in string.h)

- File handling should be implemented using the C standard library (f???? family of functions - e.g. fopen, fread, fclose, etc.)

- Each fragment of the code submitted for assessment should be written independently by you.

- The speed of the program should be independent of the computer on which the program is running. Constant units in the program should be described with appropriate comments, for example:

```
const int BOARD_WIDTH = 320;    // pixels
const double TEXT_X_COORDINATE = 60.0; // percentage of the width of the
screen
const double TEXT_Y_COORDINATE = 5.0;  // percentage of the height of the
screen
```