Worksheet 4

COMP110: Principles of Computing

Ed Powley

January 2016

Introduction

In this assignment, you will create three small C++ programs:

A. A console application implementing the "terminal hacking" word-guessing game from the Fallout games;

B. A console application implementing the 2-player board game Connect 4;

C. A graphical application which generates and displays the Mandelbrot fractal.

This worksheet tests your ability to translate various program notations (pseudocode, flowcharts, mathematics, narrative descriptions) into C++ code.

Submission instructions

The GitHub repository at the following URL contains skeleton projects for the three parts of this worksheet.

`https://github.com/Falmouth-Games-Academy/comp110-worksheet-4`

Fork this repository into your own GitHub account. To submit, create a GitHub pull request.

For the most part, this worksheet requires you to edit C++ code in the skeleton projects provided. **Please do not rename or delete the projects or files provided, and please do not create new projects.** For those questions which require you to produce artefacts other than C++ code, e.g. pseudocode or flowcharts, this material should be formatted in Markdown (possibly with embedded images) and added to your GitHub repository.

Information on using GitHub is available on LearningSpace, and elsewhere online.

Marking

Timely submission: 40%

To obtain the marks for timely submission, you must submit Part A by **6pm, Monday 18th January**, **and** Parts B and C by **6pm, Monday 8th February**. As with other worksheets, you may resubmit after these deadlines in order to address any correctness issues or satisfy further quality criteria. This 40% is awarded as long as you submit *something* by the deadline.

Correctness: 30%

To obtain the marks for correctness, you must submit working solutions for the following:

- Part A, Sections 1–2;
- Part B, Sections 1–3;
- Part C, Sections 1–2.

Note that this is a threshold: the full 30% is awarded for work which is *complete* and contains *no clear errors*. In particular you will not be penalised for trivial errors which do not affect the overall functioning of your programs, nor will you receive extra credit for highly polished solutions.

Quality: 30%

The extra quality criteria for this worksheet are as follows:

1. **Part A stretch goal.** You have submitted a working solution for Part A, Section 3.
2. **Part B stretch goal.** You have submitted a working solution for Part B, Section 4.
3. **Part C stretch goal.** You have submitted a working solution for Part C, Section 3.
4. **Presentation.** Your solutions are appropriately presented in GitHub, with descriptive commit messages and appropriate documentation in `readme.md` files. You have edited the provided skeleton projects, and refrained from renaming the provided files or creating new projects.
5. **Sophistication.** Your solution for **any one** of the stretch goals is particularly sophisticated, demonstrating mastery of C++ programming concepts appropriate to the task at hand.

Part A. Terminal Hacking

In this part, you will implement a version of the "terminal hacking" minigame from *Fallout 4* (Bethesda, 2015). In this minigame you must guess a secret n-letter word, one of several options presented to you. On choosing an option, you are told the *likeness*: the number of letters which match the secret word (i.e. the same letter in the same position). For example if the secret word is `HOUSE` and your guess is `MOUSE`, the likeness is 4 out of 5. If your guess is `HOPES`, the likeness is 2 out of 5 (the letters `S` and `E` do not count as they are in the wrong positions).

1.

The following algorithm takes a guessed word and the secret word, and returns the likeness score as described above.

```
procedure GetLikeness(guessedWord, secretWord)
    result ← 0
    for $i = 0, 1, \ldots, \text{secretWord.length} - 1$ do
        if secretWord[$i$] = guessedWord[$i$] then
            increment result
        end if
    end for
    return result
end procedure
```

Implement the algorithm as a C++ function, choosing appropriate data types for the parameters, return value, and any variables.

2.

Implement the main loop of the game, structured according to the flowchart shown in Figure 1.

3. Stretch goal

In the skeleton project, the words are chosen at random. This may lead to instances of the game which are unsatisfying, for example where all words have a low likeness score with respect to the secret word.

Design (as pseudocode) and **implement** (within your C++ project) an improved word choosing algorithm.

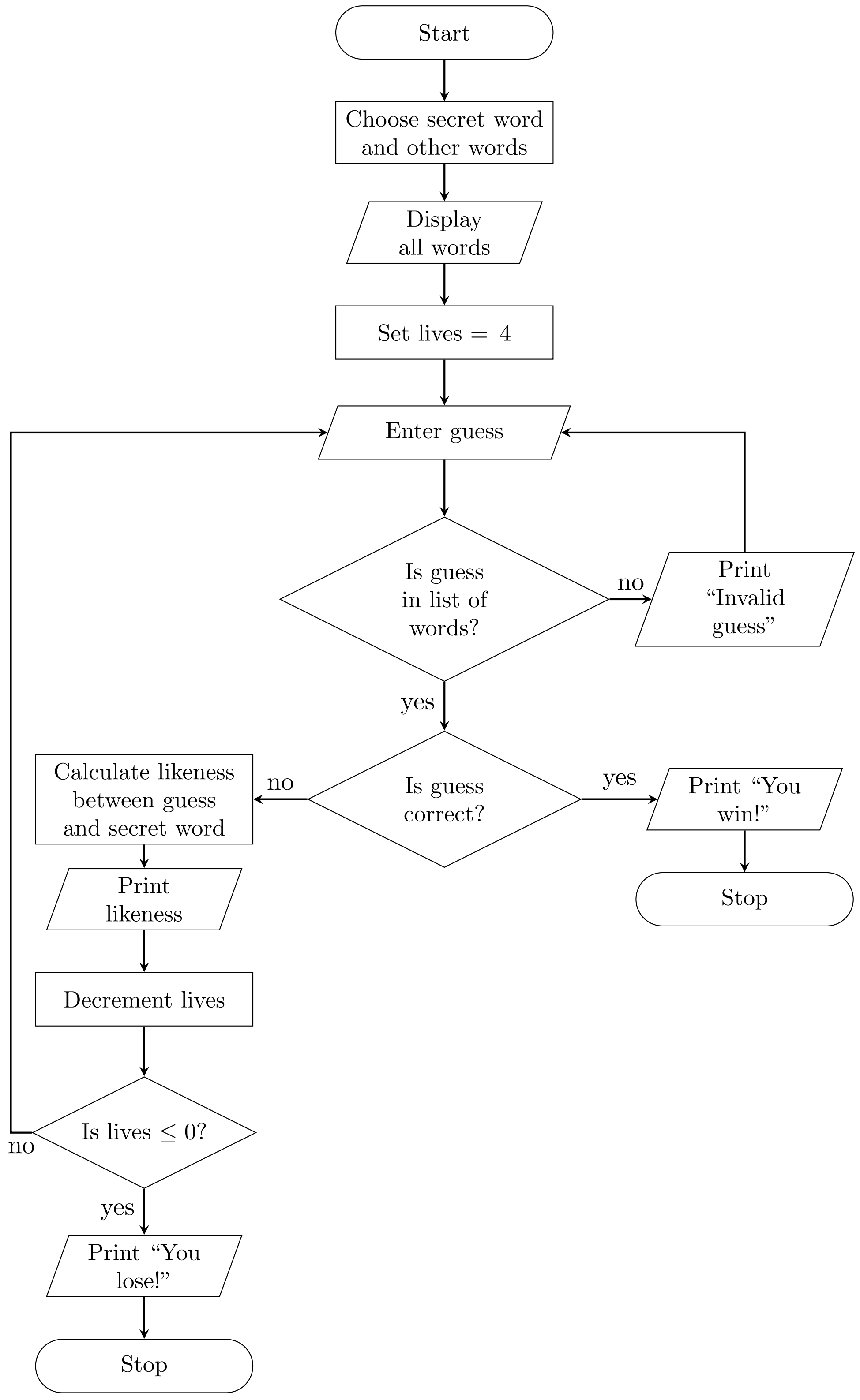

Figure 1: Flowchart for the Terminal Hacking game

Part B.
Connect 4

In this part, ...

1.

The following algorithm checks for a single row of 4 counters on the board. The coordinates s_x, s_y give the starting point of the line, and d_x, d_y give the direction.

```
procedure CheckLine(board, $s_x$, $s_y$, $d_x$, $d_y$)
    if board[$s_x, s_y$] is empty then
        return false
    end if
    for $i = 1, 2, 3$ do
        $t_x \leftarrow s_x + d_x \times i$
        $t_y \leftarrow s_y + d_y \times i$
        if $t_x, t_y$ is outside the bounds of the board then
            return false
        else if board[$t_x, t_y$] $\neq$ board[$s_x, s_y$] then
            return false
        end if
    end for
    return true
end procedure
```

Implement this algorithm as a C++ function.

2.

Given the above function, it is possible to check the entire board for a winning line as follows. Loop through each square on the board. For each square, use CheckLine() to check for the following four lines:

- A horizontal line ($d_x = 1, d_y = 0$)
- A vertical line ($d_x = 1, d_y = 0$)
- A diagonal line ($d_x = 1, d_y = 1$)
- A diagonal line in the other direction ($d_x = 1, d_y = -1$)

If CheckLine() returns true for any of these, then the winner is the player who owns the square currently being considered. If CheckLine() never returns true for any square on the board, then there is no winner.

Implement a C++ function `checkWin`, based on the above description. It should take the game board as a parameter, and return an integer: 1 or 2 if the corresponding player is the winner, or 0 if there is no winner.

```
for each space on the board do
    check if there is a horizontal ($d_x = 1, d_y = 0$) line starting at that space
    check if there is a vertical ($d_x = 0, d_y = 1$) line starting at that space
    check if there is a diagonal ($d_x = 1, d_y = 1$) line starting at that space
    check if there is a diagonal ($d_x = 1, d_y = -1$) line starting at that space
    if any of the above lines exist then
        the winner is the player who owns the current space
    end if
end for
if no lines were found then
    there is no winner
end if
```

3.

Implement the main loop of the game, structured according to the flowchart shown in Figure 2. Note that “switch current player” means to set the current player to 2 if it is currently 1, or vice versa.

4. Stretch goal

TODO

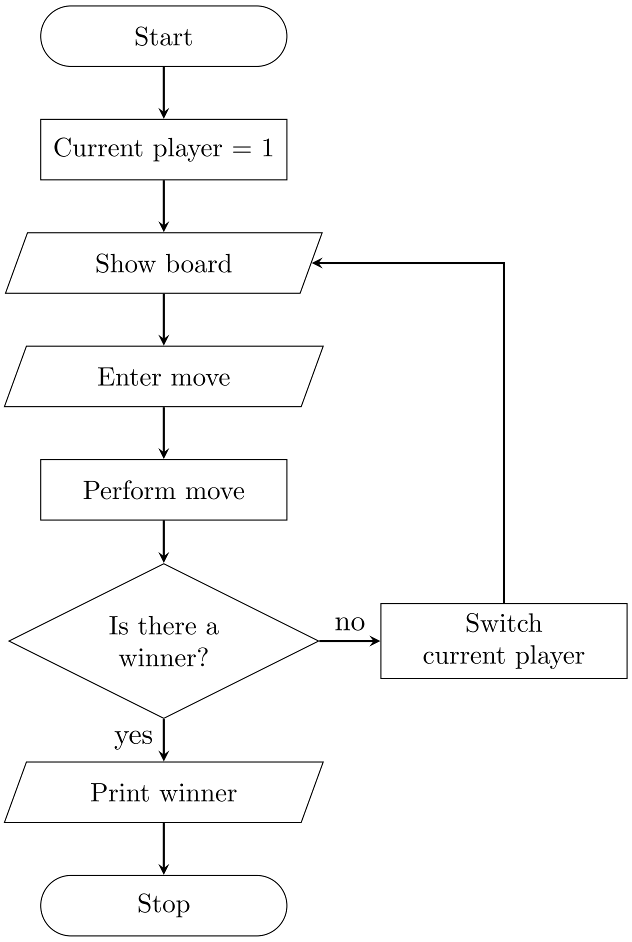

Figure 2: Flowchart for the Connect 4 game

Part C. Mandelbrot

In this part, ...

1.

Implement the following algorithm in C++. Use a `switch` statement, taking advantage of the fact that casting a `double` to an `int` causes it to be rounded down. The `abs` and `fmod` functions are defined in `<math.h>`.

```
procedure HSVtoRGB($h, s, v$)
    $c \leftarrow s \times v$
    $h' = 6 \times h$
    $x \leftarrow c \times (1 - \text{abs}(\text{fmod}(h', 2) - 1))$
    if $0 \leq h' < 1$ then
        $r = c$; $g = x$; $b = 0$
    else if $1 \leq h' < 2$ then
        $r = x$; $g = c$; $b = 0$
    else if $2 \leq h' < 3$ then
        $r = 0$; $g = c$; $b = x$
    else if $3 \leq h' < 4$ then
        $r = 0$; $g = x$; $b = c$
    else if $4 \leq h' < 5$ then
        $r = x$; $g = 0$; $b = c$
    else if $5 \leq h' < 6$ then
        $r = c$; $g = 0$; $b = x$
    else
        $r = 0$; $g = 0$; $b = 0$
    end if
    return $(r + v - c,\ g + v - c,\ b + v - c)$
end procedure
```

This algorithm converts a colour in HSV space to the equivalent colour in RGB space; all of r, g, b, h, s, v are assumed to be between 0 and 1.

2.

TODO

3. Stretch goal

TODO

4.