Option 1: Memory Game against the computer

Martin Horauer – horauer@technikum-wien.at

Table of Contents

GUI Design
Implement Step #1: Initialization
Implementation Step #2: The Game Play
Implementation Step #3: Development of the Graphical User Interface
Implementation Step #4: Integrate the Game Play and the GUI

The following structure stores the actual game state.

C

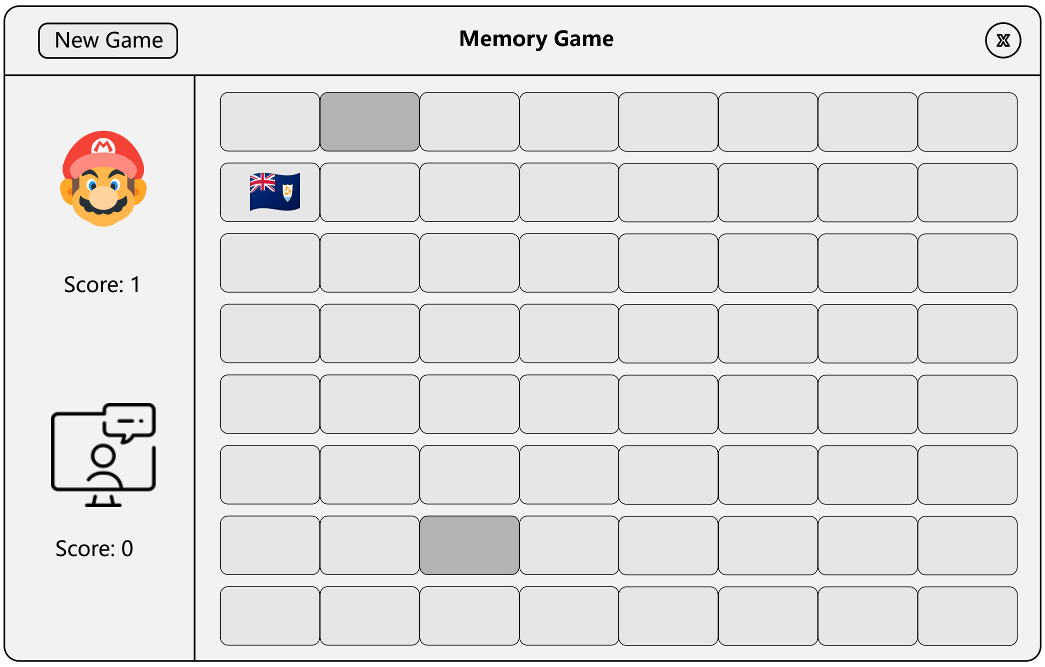
```c
#define N 64

struct state {
  unsigned int activeplayer;       // 0 ... gameplay is not active
                                   // 1 ... player is on the move
                                   // 2 ... computer is on the move
  unsigned int buttonstate[N];     // 0 ... button is "active" [default init]
                                   // 1 ... button is "inactive"
  unsigned int buttonpair[N];      // at every index store the pair index
  unsigned int attempt[2];         // [-1, -1] ... no button is clicked [default init]
                                   // [42, -1] ... button #42 is clicked
                                   // [42, 24] ... buttons #42 and #24 are clicked
  unsigned int scores[2];          // score[0] ... score of player
                                   // score[1] ... score of computer
};
```

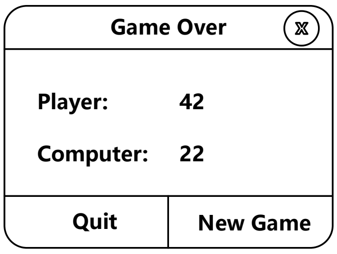
- The variable `activeplayer` indicates which player is *on the move*.
- The variable array `buttonstate[]` indicates whether a button is still active (clickable) or not; buttons become inactive when one player identified a valid pair.
- The variable array `buttonpair[]` stores at every index the index of the respective pair index.
- The variable array `attempt[]` indicates whether no, one or two buttons are clicked. When one button is licked the respective symbol on the button is *shown*. When two buttons are clicked, the symbols on both buttons are *shown* and a timeout timer is started. Once, this timeout timer elapses both symbols are *hidden* again.
- The variable `scores[]` manages the actual scores of the player and the computer.

GUI Design

The following images sketch a possible design of the GUI using the Gtk4 library. The sketch below shows:

- a headerbar with a `New Game` to the left and a program title.
- the main content area consists of a sidebar and a game play area; the sidebar shows the two players (e.g. using avatar images) and their score (e.g. using a label).
- the game play area consists of `N` buttons in a squared arrangement that have different states.





New Game

When the user presses the `New Game` button, the game play shall be initialized (see below) and started.

Game Play Area

The game play area consists of `N` buttons organized in a squared matrix layout. Buttons can be *active* (default after initialization) or *inactive* (whenever a player finds a matching pair).

When a player is on the move he can press a button that will reveal the image beneath it.

When the player presses another button, the image beneath the second button will be revealed too and a timer fires.

When the timer expires:

1. The buttons will be set to inactive (when a pair is found) and the score of the player will be increased.
   a. As long as there are active buttons the player can try another pair.
   b. The game is over when no more buttons are in the state *active*. In this case a `Game Over` dialog shall show up. The latter shall show the final scores and allow to quit the game or start over with a new one.
2. In case the buttons do not match up, they will simply be hidden and the other player gets the turn (by setting the variable `activeplayer`).

Implement Step #1: Initialization

Write a function `init()` that clears and initializes the state structure (see above). This initialization is done once the `New Game` button is pressed or when the program starts.

Program Start

The `buttonstate[]`, `attempt[]` and `scores[]` variables shall be initialized.

New Game Button

Pressing the `New Game` button will start a new game.

The `buttonpair[]` array needs to be initialized first. To do so, for example, pick two random numbers in the range `[0,N-1]` - let's say 42 and 24 (`buttonpair[24] := 42` , `buttonpair[42] := 24`). Now, again pick two random numbers in the same range excluding all values picked before. Similarly, initialize the respective `buttonpair[]` indices - repeat this until the entire array is filled.

Once, all elements are initialized the active player shall be set to the player using the `activeplayer` variable.

Implementation Step #2: The Game Play

Now implement the game play by writing a `player()` , a `computer()` and a `check()` function. Possible code sequences are sketched below:

player()

```
PSTART: read num1 <- in the range [0,N-1]
check button[num1] == active
if NO
  goto PSTART
else
  update the state.attempt
  read num2 <- in the range [0,N-1]
  if num2 == num1
    goto PSTART
  else
    check button[num2] == active
    if NO
      goto PSTART
    else
      update the state.attempt
      invoke: query = check()
      if (query == match)
        increase player score
        if there is still an active button
          goto PSTART
        else
          output state.scores
          END program
      else
        clear state.attempt
```

computer()

```
CSTART: get two random numbers in range [0,N-1]
update the state.attempt
invoke: query = check()
if (query == match)
  increase computer score
  if there is still an active button
    goto CSTART
  else
    output state.scores
    END program
else
  clear state.attempt
```

check()

```
check if state.attempt matches buttonpair
if YES
  return match
else
  return no match
```

The above `computer()` program logic is trivial and and doesn't remember old tries. Furthermore, the approach doesn't implement any sort of strategy. You are free to add these …

Implementation Step #3: Development of the Graphical User Interface

As a next step implement the graphical user interface (GUI) code for the game play using Gtk4. To do so start off with small demo programs.

Checkout the example code in the program `gtk4-demo` available on your Linux computer - if not you'll need to install it.

- A program to implement the code for a headerbar.
- A program to implement the sidebar that loads images and/or labels.
- A program to play with some button states.
- A program that implements a custom dialog.

Once, these smaller programs are working integrate them to a GUI program as outlined above.

You may download some free icons from the internet or, for example, make use of some flag icons found here.

Implementation Step #4: Integrate the Game Play and the GUI

Now add and modify the functions `init()`, `player()`, `computer()` and `check()` to be used from within the callback functions of the buttons and integrate them with the GUI code.

Note, you will need to add some functionalities, e.g. an indication whether the player or the computer is active. This can be done, e.g., by toggling between colored and grayed avatar images depending on the `state.activeplayer` variable

Finally, test and optimize your code.