NCU Operating System (CE3002A) - Program 2

tags: 作業系統

> HackMD Report URL: https://hackmd.io/@Mibudin/H1QIsvrud (https://hackmd.io/@Mibudin/H1QIsvrud)

Contents

What is this

- This is a simple version on a terminal of the ***Game of Life***, a famous zero-player game.
    - Conway's Game of Life Wiki (https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life)
- This project uses ***C / C++*** as the major (or the only one) programming language.
- This project implementing multi-threading features by ***`std::thread`*** *(depending on* ***Pthread****)* and lots of associated things of the newer C++ (above C++11).

How to use

- There are two major mode in order when this program executed.
- The following keys in either lower or upper case are equivalent.

1. Map Set Mode

- It is possible to set the initial map in this mode.
- Press `W` , `A` , `S` , `D` to move the cursor, to choose which cell to be set.
- Press `[` to set a cell to be dead.
- Press `]` to set a cell to be live.
- Press `;` to decrease the turn period.
- Press `'` to increase the turn period.
- Press `\` to go to the ***Map Run Mode***.
- Press `P` to exit the game.

2. Map Run Mode

- Just see the cells live and dead, in a zero-player game.
- In default, there are at most 10000 turns to run.
- Press `;` to decrease the turn period.
- Press `'` to increase the turn period.
- Press `\` to go to the ***Map Set mode***.
- Press `P` to exit the game.

- WARNING : Exiting this program not in regular may let your current terminal have some I/O problems.

Project File Structure

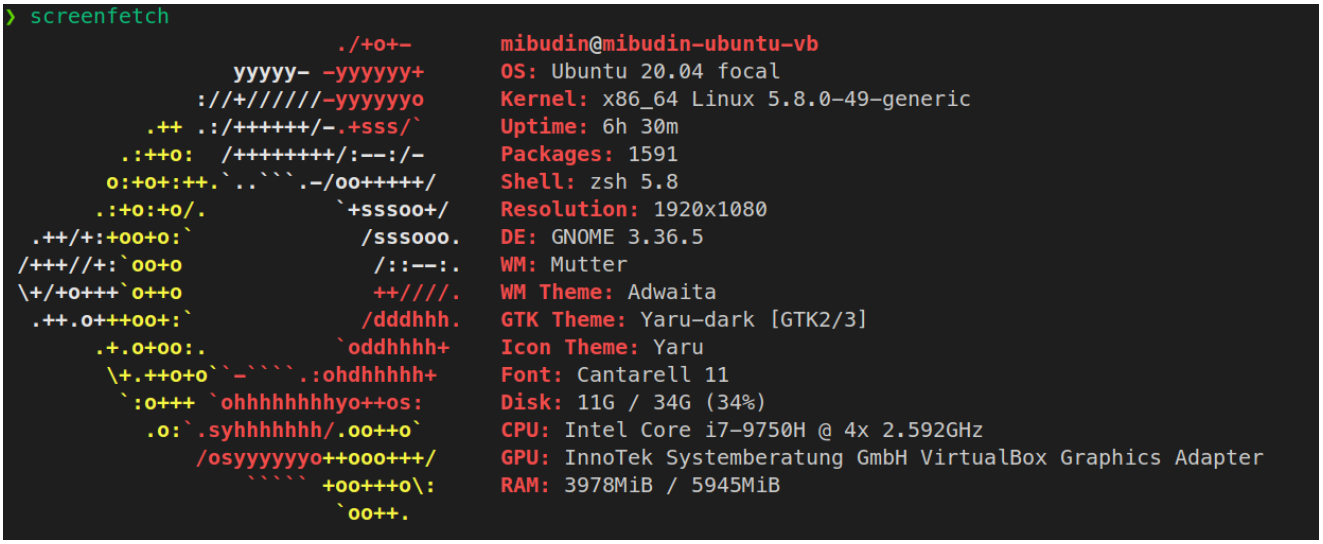
- `/` : The folder of this project.
- `/bin/` : The folder of binary files *(including executable files)*.
- `/inc/` : The folder of C / C++ header files.
- `/obj/` : The folder of C / C++ object files.
- `/src/` : The folder of C / C++ source files.
- `/Makefile` : The file to compile the source files of this project.

Development Environment

Operating System

- Ubuntu Version

```
> uname -a
Linux <device-name> 5.8.0-48-generic #54~20.04.1-Ubuntu SMP Sat Mar 20
13:40:25 UTC 2021 x86_64 x86_64 x86_64 GNU/Linux
```



Terminal

- GNOME Terminal Version

```
3.36.2
```

Program Language

- C / C++ *(C++ 17)*

C++ Compiler

- G++ Version

```
> g++ --version
g++ (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is
NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE.
```

- Compiling command
  - See `/Makefile` for more details

Function List

> There are only ***functions*** declared, not including variables, macros, classes, structures, types, outer headers, and so on.
> See the source codes for more information.

GoL

`gol.cpp`

Global

- `int main()`
  - The main function, including the main logic.
- `void init()`
  - Initialize many things.
- `void setMap(const int turn)`
  - Control the mode of setting the map.
- `void loop()`

    - Control the mode of running the map.
- `void deinit()`
    - Some essential processes before exiting this program, then exiting.

Config

```
config.hpp , config.cpp
```

Cell

```
cell.hpp , cell.cpp
```

Public

- `Cell()`
    - The constructor.
- `CellStatus* getStatus()`
    - Get the current and future status.
- `void setStatus(CellStatus _status, const int t)`
    - Set the specific status.
- `CellStatus interact(const WorldMap* map, const int size[2], const int x, const int y, const int turn)`
    - Let this cell interact with its neighbors.
- `void render(const int x, const int y, const int turn)`
    - Render this cell on the screen.

Private

- `int checkEnv(const WorldMap* map, const int size[2], const int x, const int y, const int t)`
    - Check that there are how many living cells.
- `CellStatus liveOrDie(const int liveCellCount, const int t)`
    - Do the determinations about whether this cell being going to die or not.

World

```
world.hpp , world.cpp
```

Public

- `World(const int width, const int height)`

    - The constructor.
- `void deinit()`
    - Some essential processes before exiting this program.
- `int goTurn()`
    - Calculate the world of the next turn.
- `int nextTurn()`
    - Go to the next turn formerly by adding a counter.
- `int backTurn()`
    - Go to the previous turn formerly by subtracting a counter.
- `int getTurn()`
    - Get the current turn number.
- `int* getSize()`
    - Get the size of the world.
- `Cell* getCell(const int x, const int y)`
    - Get the cell at (x, y) in the map.
- `void setSampleMap()`
    - Set the map to the sample map.
- `void countAllCells(const int t)`
    - Count the amount of alive cells in specific status buffers.

Private

- `void updateAllMap()`
    - Update all the cells in the map ***in this object***.
- `void updateAllCells()`
    - Update all the cells in the map ***in cells***.
- `void updateACell(const int i)`
    - Update a cell.
- `void updateACell(const int x, const int y)`
    - Update a cell.
- `void setMap()`
    - Initialize the map.
- `void fillMap()`
    - Fill the map with dead cells.
- `bool needRender()`
    - Whether this world needs to be rendered now.
- `void render()`

    - Render the world.
- `void renderInit()`
    - Render the world in the beginning.

Textarea

```
textarea.hpp , textarea.cpp
```

Public

- `Textarea(World* wld)`
    - The constructor.
- `void setMode(const ModeType mode)`
    - Set the current mode.

Private

- `void renderSpace(const int y)`
    - Render the blank space row section.
- `void renderBar(const int y, const BarType type)`
    - Render the divider bar row sections.
- `bool needRender()`
    - Whether this text area needs to be rendered now.
- `void render()`
    - Render the text area.
- `void renderInit()`
    - Render the text area in the beginning.

Screenio

```
screenio.hpp , screenio.cpp
```

Public

- `void initTty()`
    - Initialize the terminal.
- `void deinitTty()`
    - Some essential processes before exiting this program.
- `void getTty()`
    - Get the attributes of the current terminal.

- `void setTty()`
  - Set the attributes of the current terminal.

Private

- `void backupTty()`
  - Backup the original attributes of the current terminal.
- `void restoreTty()`
  - Restore the original attributes of the current terminal.

Keyio

```
keyio.hpp , keyio.cpp
```

Public

- `void startWait()`
  - Start to wait a key to be input.
- `bool waitKeyAsync(const std::chrono::steady_clock::time_point time)`
  - Wait a key to be input asynchronously ***(by running it on a new thread)***.
- `int getLastKey()`
  - Get the last key code to be input.
- `int blockWaitKey()`
  - Wait a key to be input synchronously.

Private

- `int kbhit()`
  - Whether a key input.
- `int getch()`
  - Get the key input.

Renderer

```
renderer.hpp , renderer.cpp
```

Public

- `void renderAll()`
  - Render all registered renderees on the screen.
- `void renderInit()`

    - Render all registered renderees on the screen in the beginning.
- `void addRenderee(IRenderee* renderee)`
    - Register a renderee to be rendered.

IRenderee

> `renderee.hpp`

Private

- `virtual bool needRender()`
    - Whether this renderee needs to be rendered or not.
- `virtual void render()`
    - Render this renderee on the screen.
- `virtual void renderInit()`
    - Render this renderee on the screen in the beginning.

Program Logic

> The major program login in `gol.cpp` .

1. Initialization about this app.
    1. Creating essential handlers.
    2. Set important configurations.
    3. Register renderees.
    4. Render the beginning screen.
2. Go into the Map Set Mode first.
    - While in the **Map Set Mode** loop.
        1. Wait a key input synchronously.
        2. Handle the input key.
        3. If needed, do the essential processes.
        4. If needed, go through this loop again.
        5. If needed, exit the game.
        6. If needed, go to another mode.
    - While in the **Map Run Mode** loop.
        1. Totally four thread synchronously running.
            - **Main thread**
                - Handle the ***main logic*** and ***other threads***.

        - **World thread**
            - Handle the ***calculating of the next turn***.
        - **Render thread**
            - Handle the ***rendering of the current turn***.
        - **Key thread**
            - Handle the ***inputting of the key codes***.
    2. If having, get the key input from the ***Key thread*** and do the corresponding handling.
    3. Wait the ***World thread*** and the ***Render thread*** both completed a turn task.
    4. If the processing time have not achieved the turn period time duration, waiting to achieve.
    5. If needed, go to another mode.
    6. If needed, go through this loop again.
    7. If needed, exit the game.
3. Some essential processes before exiting this program.
    1. Restore the original attributes of the terminal.
    2. Delete threads.
    3. Free resources.
    4. Exit this game.

References

Much many, and so on…