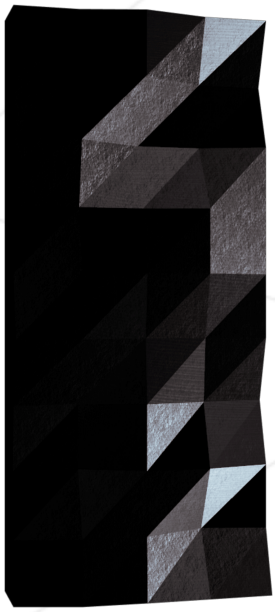

Rush Algorithms

Power 4

42 sta ff staff@42.fr

Summary: This rush is there for you to re fl ect on algorithms and techniques e ffi cient programming. Or at worst, to make you re fl ect at all.

Contents

Chapter I

Preamble

Joshua: Greetings, Professor Falken.

Stephen Falken: Hello, Joshua.

Joshua: A strange game. The only winning move is not to play. How about a nice game of chess?

Chapter II

Subject

Part II.1 mandatory

If you had a happy childhood, you probably played the famous game Power 4. If it does not, well then it's time to make up the shortfall.

As you have understood by reading the title of this document, the purpose of this rush is to write a play Power 4. It's already a great subject in the state, but know that you'll also have to write an AI against which to play. It's wonderful is not it?

- You must implement the rules available right here .
- At the di ff erence of the rules of version o ffi cial of the game, your game grid does not automatically make 6 rows and 7 columns. The number of rows and columns will be sent to your program in the form of two numeric parameters. However, the minimum acceptable size of the grid is the grid size o ffi cial. If the two parameters do not meet these conditions or have no sense, your program must report and exit cleanly.

- The human and AI players play in turn. The first to play is determined randomly at the beginning of the game.
- At the beginning of the game, at the end of the game and between each shot, your program must a ffi expensive in the terminal state of the grid in a clear format for reasonable grid sizes. We tolerate a chage a ffi less clear for large grids. For a ffi clear drying means that the pieces of both players and their relative and absolute references are easy to identify.

- Each turn the human player, your program must invite him to enter the column on which the player wishes to play. If the data entered is not valid, your program should call the human player to enter a new value until the one-Ci be valid.

- Contrary to Joshua in the film Wargames, your AI has no right not to play. In addition, your AI must try to win, it is irrelevant to write an AI that plays randomly. Finally, an AI that takes several seconds to play is not a good AI, the answer to your needs as soon as possible.

- At the end of the game, your program should clearly indicate which player has won or if the game is drawn and exit cleanly.

II.2 Party Bonus

To allow you to go further, we offer a bonus game to rush. For the points of the bonus round are counted, your party must be mandatory and infallible PERFECT!

We recall that an unstable program (segfault, error bus, etc.) in the bonus game ... is 0. Do not add that stable features you have tested!

Here are some potentially interesting bonus ideas:

- **A graphical interface. For this you can use the graphics library that you want (including termcaps and ncurses).** You must add a para- meter in your program to enable or disable your GUI. When the GUI is disabled, your program should behave exactly like the mandatory part described. When the GUI is EBV Acti-, you must meet the mandatory part on the a ffi chage in the termi- nal, but entries can be directly from your GUI. It is forbidden to make the graphics library or sources on your deposit

  GIT.

- Advanced AI using a specific algorithm and implemented correctly, which will be possible to adjust the di ffi culty when the program or the beginning of the game. Of course, the instructions of the compulsory section on the AI must be followed. You will need to prove to your checker that your AI respects both the directions of the mandatory part and the bonus round and tell the algorithm that you used, why it is suitable Ci and prove that your implementation of this algorithm is correct to earn your points.

Chapter III

Rendering

- You must get to the root of your rendering deposit a file author 2 containing your logins, each on a line, and followed by a '\ n':

```
$> Cat -e author xlogin
ylogin $ $ $>
```

- Your program must be called Power 4.
- You must have a Makefile with the usual rules. When in doubt, put all the known rules.

- Only the content present on your deposit will be evaluated in defense.
- Under the GUI bonus, it is prohibited to make the source or a binary version of your video library on your rendering repository. Be smart and elegant in your solution to this problem.

chapter IV

licensed features

- read (2)
- write (2)
- malloc (3)
- free (3)
- errno
- strerror (3)
- srand (3)
- rand (3)
- time (3) (to the seed of srand)

Chapter V

instructions

The following guidelines will all party defense scheme. Be very careful at- tentifs and in their application because they are awarded a 0 final.

- In case anyone would doubt, that rush of course write in C.
- This project will be corrected only by humans.
- Your project should be the standard. An error standard is eliminatory, including your libft.
- You must handle errors significantly. In any case your program should exit unexpectedly (Segmentation fault, bus error, double free, a ffi nonsense chage, etc.) or from Loop in fi nite, whether in the mandatory part or in the bonus round. Such an error is eliminated. No, you do not need Valgrind for that.
- Any memory allocated on the heap must be freed properly. Forgetting to deallocate memory is eliminated.
- The return value of all your system calls must be veri fi ed. An oversight is eliminatory. We tolerate not veri fi ed the return value write (2). It defines a "system call" any function which man is in Section 2 on Macs school.
- You can use your library libft. For this, you must make the source to the root of your repository in a folder named libft. Your bibliography library will be compiled along with your made and linked with that Ci. Your libft should be the standard. use your libft to bypass the standard is eliminatory.

Good luck to all for the rush!