Technical University of Košice
Department of Computers and Informatics

Problem Set 6
Death Factory

Stanislav Voloshyn
2020/2021

Assignment

Using the ncurses library, create a program (game, presentation or other artistic work), with the following minimal requirements:

- Project contains 2D world.
- Project meets at least 3 challenges:
  - Work with colors
  - Keyboard control (no Enter needed)
  - Multiple levels
  - Work in time (in the time the program is changed)
  - Work with command-line arguments
  - Work with files
- Project must be more complicated then the sample programs, with an adequate level of difficulty.

Game logic The code consists of

5 functions:

- **bool Kill_p_Collision(short numLvl) -** checks for collisions with enemies, returns **True** if the player touched an enemy, otherwise returns **false.**

- **void CheckAndPut(int rx, int ry, short numLvl) -** the function, each time it is called, puts the player on the map and makes the player material (ordinary), thereby preventing him from passing through walls and other objects. The function takes 3 parameters, the first and two parameters **rx**, **ry**, in which the previous position of the player on the map is saved, and the 3 parameter **numLvl** - tells us what LEVEL the player is on.

- **Level 1 (int ballMuve(void) - int ballMuve13(void)), Level 2 (int ballMuveW1(void) - int ballMuveW24(void)),**
  Level 3 (int ballMuveQ1(void) - int ballMuveQ18(void)) – each of these functions calculates the exact coordinates of the player at the next iteration of the loop, thereby creating the illusion of the player's movement in the following functions **(***void PutEnemies_lvl_1()*, *void PutEnemies_lvl_2(), void PutEnemies_lvl_3()***).**

- **void PutEnemies_lvl_1(void), void PutEnemies_lvl_1(void), void, PutEnemies_lvl_1(void) -** the functions takes already prepared coordinates for each enemy by placing it on the map and in parallel determining its speed of movement across the map.

- **int pick_coin(short numLvl) -** the function checks the collision of the player with the coins on the map and returns the number of coins the player has become.

- **int is_pressed_button(int numLvl) -** the function checks if the player is at the point at which the **“XXX” button** is located and returns the number of this button for the first button it is number 1, for the second 2, and so on ... .
  The function takes the value of the level number (**int numLvl**) at which the check will be.

- **bool winCol(int lvl) -** the function checks for a collision of the player with the “**WIN” button**, which means the transition to the next level or victory.
  If the player is in the same position as the Win button then the function returns True otherwise False, a takes the value of the level at which the check will be carried out.

- **void *timer(void *data) -** the function is part of the **&thread timer** thread that was created at the start of the **main()** function;

  With an interval of one second, the function changes the value of the variable **timex.sec**, decreasing it by one.
  Using this function, the interval for the timer is calculated, which is already displayed on the screen.

- **void about_the_game() -** the function clears the previous menu of the "**menuWin**" and creates a new one (**aboutGame**) on which it introduces the basic rules of the game and a short plot also the function reads the Escape key to exit to the previous window "**menuWin**".

- **void GAME_WON_MENU() -** the function clears all previous windows and creates a new window **GameWon** where it displays the content from the file "**gamewon.txt**", the function also reads the keystrokes, and if the player presses **ESC**, the program will return him to the main menu (**menuWin**). The function will be called in the code only when the player passing the third level collides with the **“Button WIN”.**

- **void GAME_OVER_MENU() -** the function clears all previous windows and creates a new window **GameOver** where it displays the content from the file "**gameover.txt**", then the function creates an additional menu for choosing further actions, the first is to play again, the second is to return to the menu and the third is to exit the game immediately. **GAME_OVER_MENU()** work only when the player collides with the enemy (“*”) or the timer expires.

The game data are stored in a 2-D array, where in two dimensions represent game world data. The array contains:

- **‘ ‘ -** empty position.
- **‘#’ -** painted white.
- **‘*’ -** denotes an enemy and also uses the term murder weapon. ‘*’ is colored white on a red background.
- **‘[ ]’ and ‘+’ -** indicates the place where the enemy appears.
- **‘$’ -** represents one coin on the card.
- **‘<->’ -** these symbols represent the player himself on the map painted black on a blue background.
- **‘X]’ -** these symbols represent the player himself on the map painted black on a blue background, the button itself glows red until the player press it, after that it will turn green and some doors will open.
- **‘%%%’ -** these symbols are a door or barrier for the player through which he cannot pass until he presses a certain button. Throughout the game This symbol is red.
- **‘W]’, ‘I]’, ‘N]’ -** the button performs the function of going to the next level or calls **GAME_WON_MENU()** in case the player is on the last level.

Game structure:

Initially, the main() function takes 2 arguments with the help of which two files of the txt format will be opened in the future. Then we set up work with the terminal using commands from **ncurses**.

Then, using a loop, we go through the already created two-dimensional array and, depending on which arrow the player presses up or down, a two-dimensional array will be displayed where a certain part will be highlighted in red, the loop will also check for key presses if the player selects the first item and press Enter, it will go to the next menu to select

the levels in which you can play, the level selection menu works in the same way as the previous ones ...

If the player, while scrolling through the level menu, selects the first level, then the program goes into a loop for the first level, if the player clicks on the second level, then the program goes into a loop for the second level, which does not differ much from the first loop.

With each subsequent iteration of the loop in the game, the player gets the opportunity to press the up-down and left and right keys, thus, with each next iteration, the player will change his position on the map, along with him, the position of enemies will change in parallel, s ince before **wgetch(* Win)** set **timeout(100)** to 100 milliseconds.

In the player's game loop, the structure of variables contains the (**struct player**) player in which the coordinates of the player before pressing the key (**kx; ky**) and the coordinates of the player after pressing the key (**x; y**) are written.
Then, at each iteration of the loop where the player changes his position on the map, the new position of the player and enemies is written to the array of the structure type map3, before that, overwriting **level1-2.map13[player.y][player.x]** from the main array of the map level **(level12.map-_3[][]).**

No matter what level the player is on, in the level loop the first action will be to write the result of the function **pick_coin(short numLvl)** to a variable of structured type - **dashpanel.coins.** Further in the loop, this variable will be displayed on the dashboard window.

The timer will also be displayed on the dashboard window.

Further in the loop, the map is updated by rewriting from the main array (**memcpy()**).

Next functions are:

CheckAndPut(int rx, int ry, short numLvl), Kill_p_Collision(short numLvl), PutEnemies_lvl_1-3().

After displaying the contents of the array and coloring the program waits for key input using the Get Chart function, and waits for exactly 50 milliseconds, if the key has not been pressed, the loop does not stop there, after that there is a check for pressing certain keys, such as:

- **ESC** - the command goes to the main menu.
- **R** - command performs level reload.
- **KEY_UP** – decreasing player.y as well as moving up.
- **KEY_DOWN** - increasing player.y as well as the player's downward movement.
- **KEY_LEFT** - decreasing player.x as well as moving left
- **KEY_RIGHT** - increasing player.x and also moving the player right.

All subsequent cycles of the levels will execute the same.

to compile the program use:

```
gcc -std=c11 -Wall -Werror -g program.c -lm -lpthread -lcurses -o program
```

to run the program use:

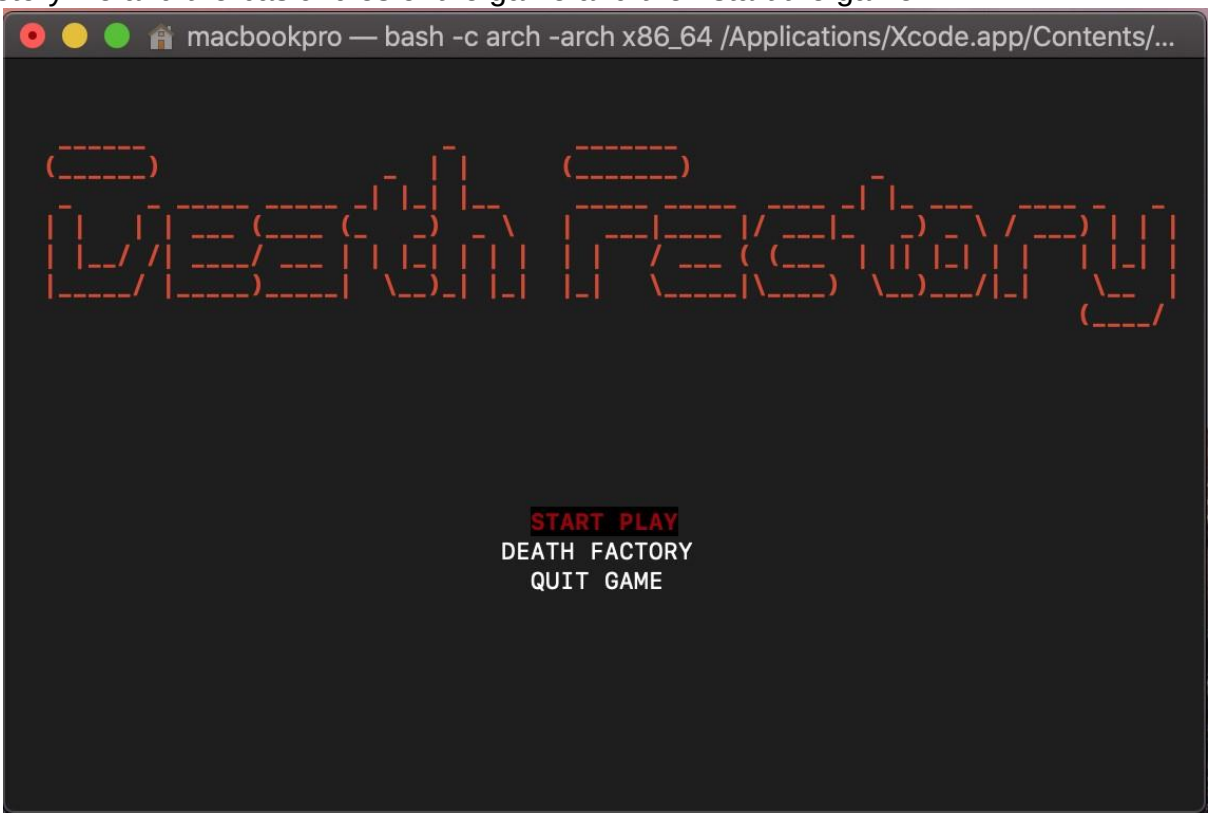
```
./program gamewon.txt gameover.txt
```

Game play

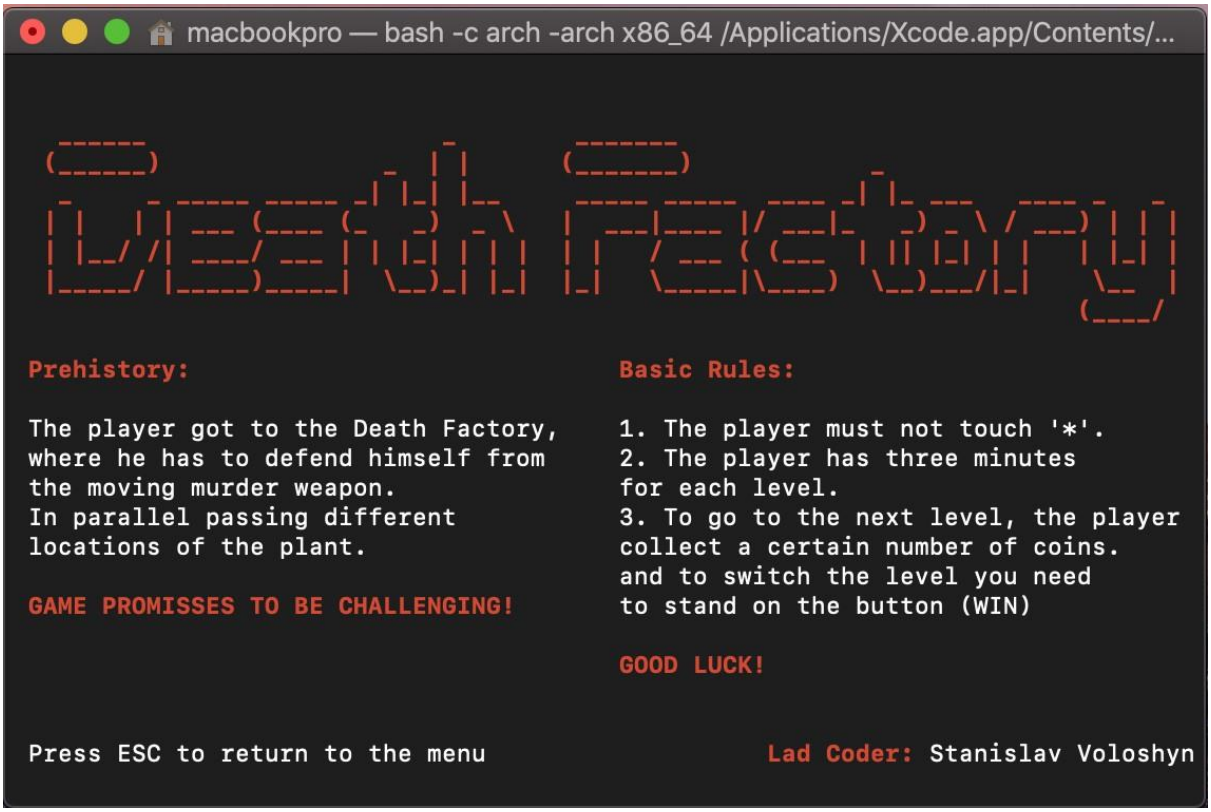
The game is called **Death Factory**.

The plot is very simple the player got to the death factory where he has to go through 3 locations where he will go through dangerous places, avoiding the murder weapon (enemies), while collecting coins in order to open new locations and continue the game...

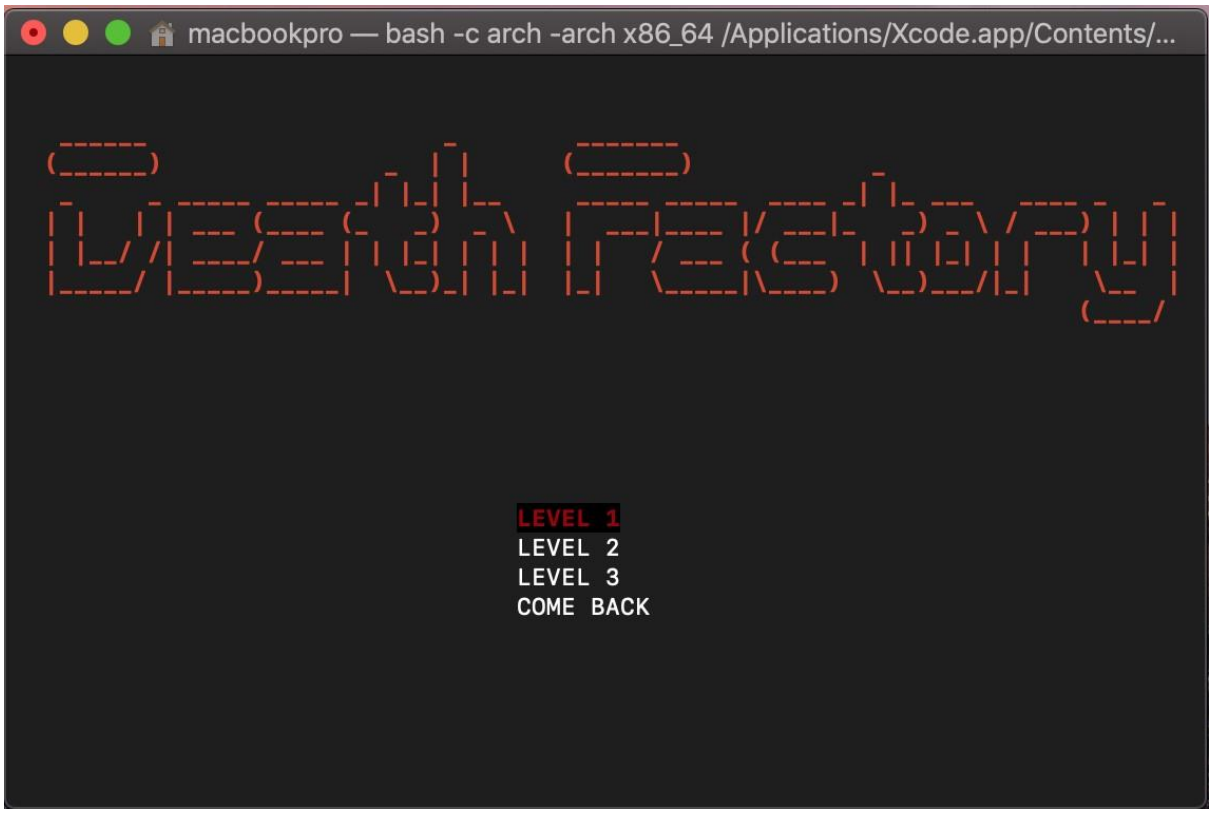
First, the player has the choice to start the game immediately or read a short storyline and the basic rules of the game and then start the game



After clicking on DEATH FACTORY we can read a short plot and basic rules of the game..



After clicking on ESC we will return to the main menu and then click on Start Play..

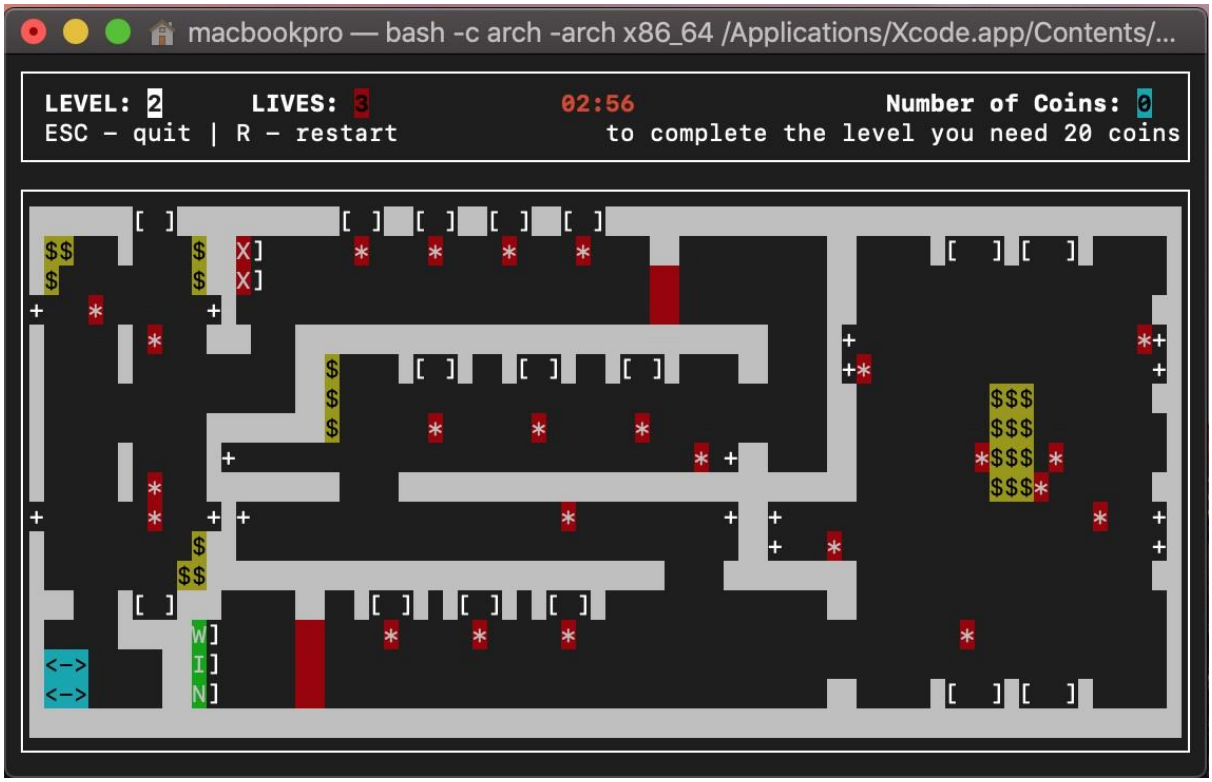


The comeback button returns to the main menu. Others start the level.

Here we see a player who is highlighted in blue and the tasks of the player are to get to the WIN button on time, while collecting a sufficient number of coins without dying from enemies.

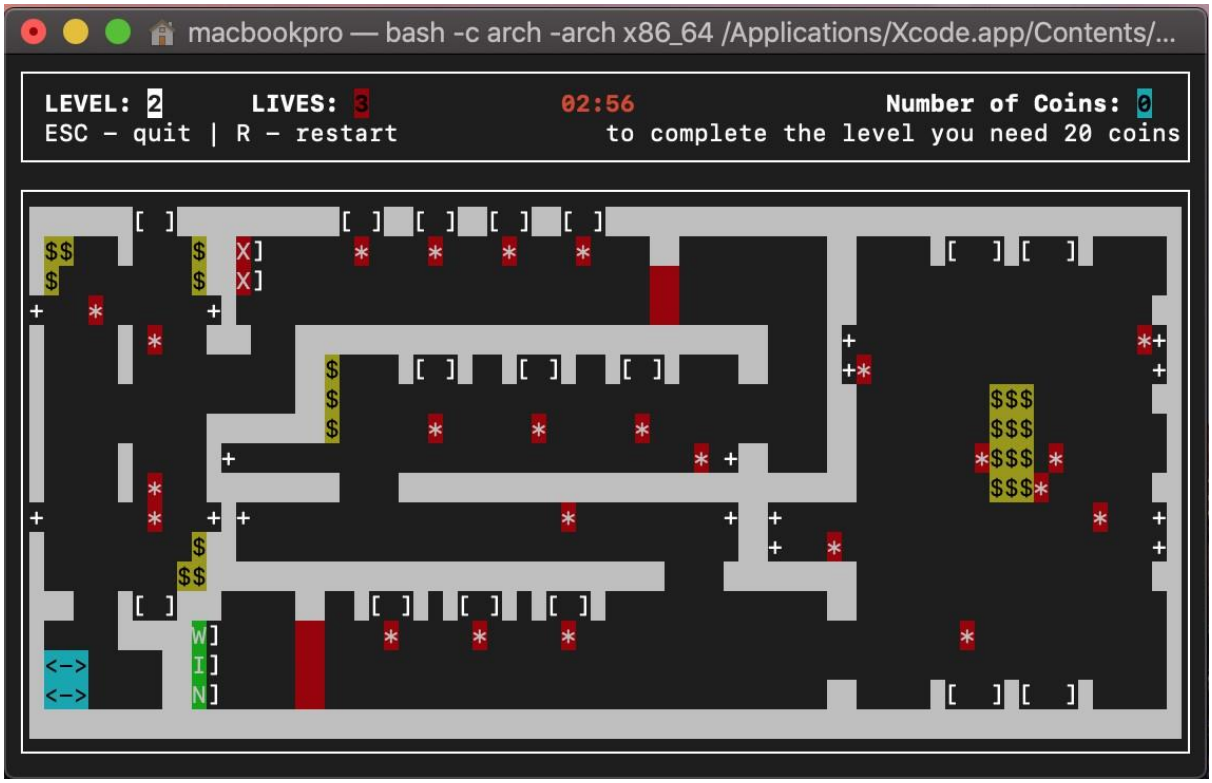
The enemies are the ‘*’ tinted in red.

Before the player steps on the button, it will be red after green, which will mean that the button is activated..

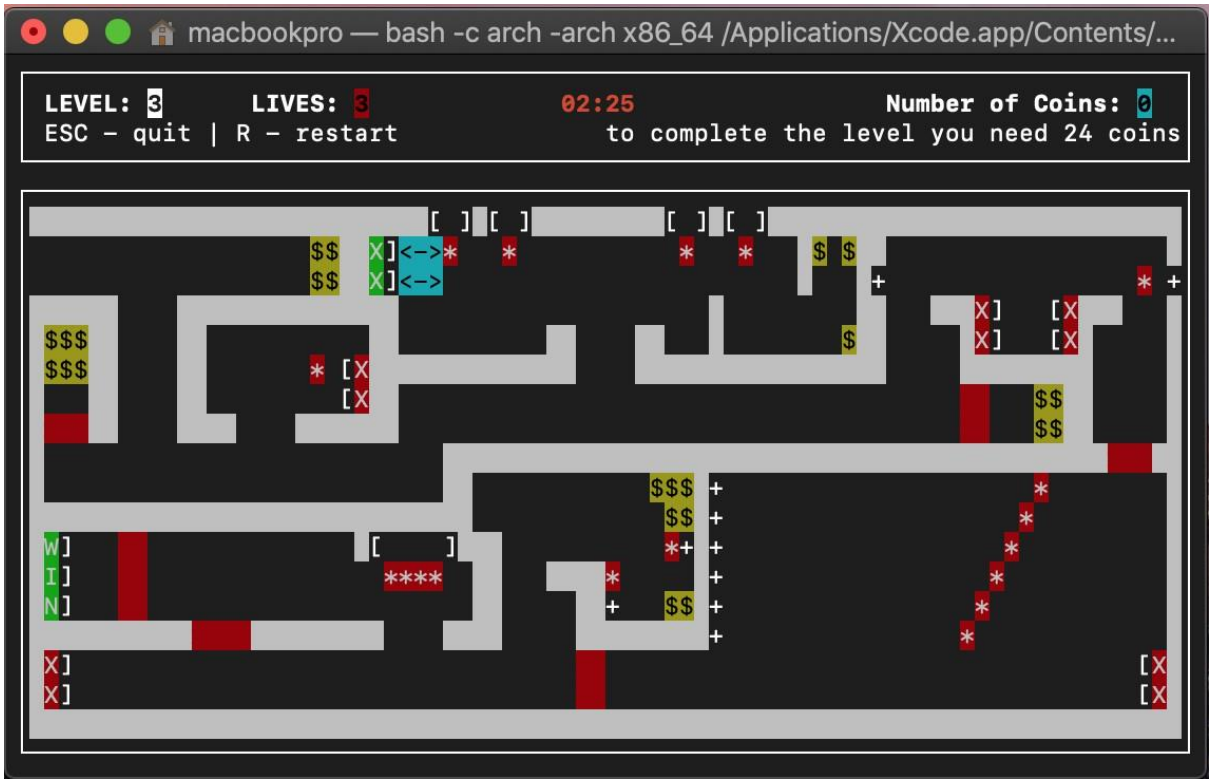


Also at the top is a panel where where where you can find out how many coins the player has collected how much life he has left and how much time he has left..

Also at the top you can find out how many coins you need to complete the level.



After the player has collected a sufficient number of coins, the passage to the WIN button will automatically open for him, which moves the game to the next level or, as in the case of level 3, symbolizes the player's victory



After the player on the third level pressed the Win button and passed the level, a new window is displayed for him.

Which means that the player has successfully passed the game

Conclusion

Programming this game I learned a lot of new material.

I think the main drawback of my code is that it is quite large,

Another drawback is how the movement of enemies is implemented

I would like to add some more levels to game. I would also like to add additional obstacles for the player, some kind of weapon or a method of protection for the player.