Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

ОТЧЕТ

к лабораторной работе №1

на тему

**СКРИПТЫ SHELL**

Студент И. В. Бобко

Преподаватель Н. Ю. Гриценко

Минск 2024

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146752068)

[2 Теоретические сведения 4](#_Toc146752069)

[3 Результат выполнения 6](#_Toc146752070)

[Заключение 6](#_Toc146752071)

[Список использованных источников 7](#_Toc146752072)

[Приложение А (обязательное) Листинг кода 8](#_Toc146752073)

1. **ЦЕЛЬ РАБОТЫ**

Целью данной работы является изучение элементов и конструкций скриптов shell: переменных, параметров, ветвлений, циклов, вычислений, команд shell и вызовов внешних программ (shell, sed, awk, wget, различные фильтры и т.д.) для решения достаточно сложной задачи, имеющей практическое значение, а также принципов интеграции Unix-программ скриптами shell. В ходе работы будет написан bash-скрипт, который представляет из себя игру про лабиринт.

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Shell или командная оболочка – это программа, которая организовывает среду для выполнения других программ и команд Командная оболочка имеет свои встроенные команды, арифметические операторы и другие синтаксические выражения, но основная её задача упрощать запуск других программ. Именно командная оболочка занимается поиском программ в текущем каталоге и в путях, указанных в переменной среды PATH, управляет сменой текущего каталога и переменными окружения. Таким образом, основная задача оболочки - интерпретировать команды пользователя и выполнять их не зависимо от того внешние ли это программы или внутренние команды [1].

Bash – усовершенствованная и модернизированная вариация командной оболочки Bourne shell. Одна из наиболее популярных современных разновидностей командной оболочки UNIX. Особенно популярна в среде Linux, где она часто используется в качестве предустановленной командной оболочки.

Представляет собой командный процессор, работающий, как правило, в интерактивном режиме в текстовом окне. Bash также может читать команды из файла, который называется скриптом (или сценарием). Как и все Unix-оболочки, он поддерживает автодополнение имён файлов и каталогов, подстановку вывода результата команд, переменные, контроль над порядком выполнения, операторы ветвления и цикла. Ключевые слова, синтаксис и другие основные особенности языка были заимствованы из sh. Другие функции, например, история, были скопированы из csh и ksh. Bash в основном соответствует стандарту POSIX, но с рядом расширений [2].

В скриптах shell переменные используютсядля хранения значений, которые могут быть использованы в различных частях скрипта. Объявление переменных происходит с помощью оператора =, например: variable=«value». Значение переменной можно получить с помощью префикса $, например: echo $variable. Переменные могут быть глобальными или локальными в зависимости от их области видимости.

В скриптах shell можно выполнять простые арифметические операции с помощью встроенных математических операций, таких как +, -, \*, /. Для выполнения более сложных вычислений можно использовать внешние программы, такие как expr или bc [3].

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В результате лабораторной работы был написан скрипт, который представляет собой игру про нахождение выхода из лабиринта (рисунок 1).

Карта с лабиринтом считывается из файла. На карте кроме игрока и выхода имеется замок, не позволяющий игроку пройти дальше, а также сам ключ от этого замка.

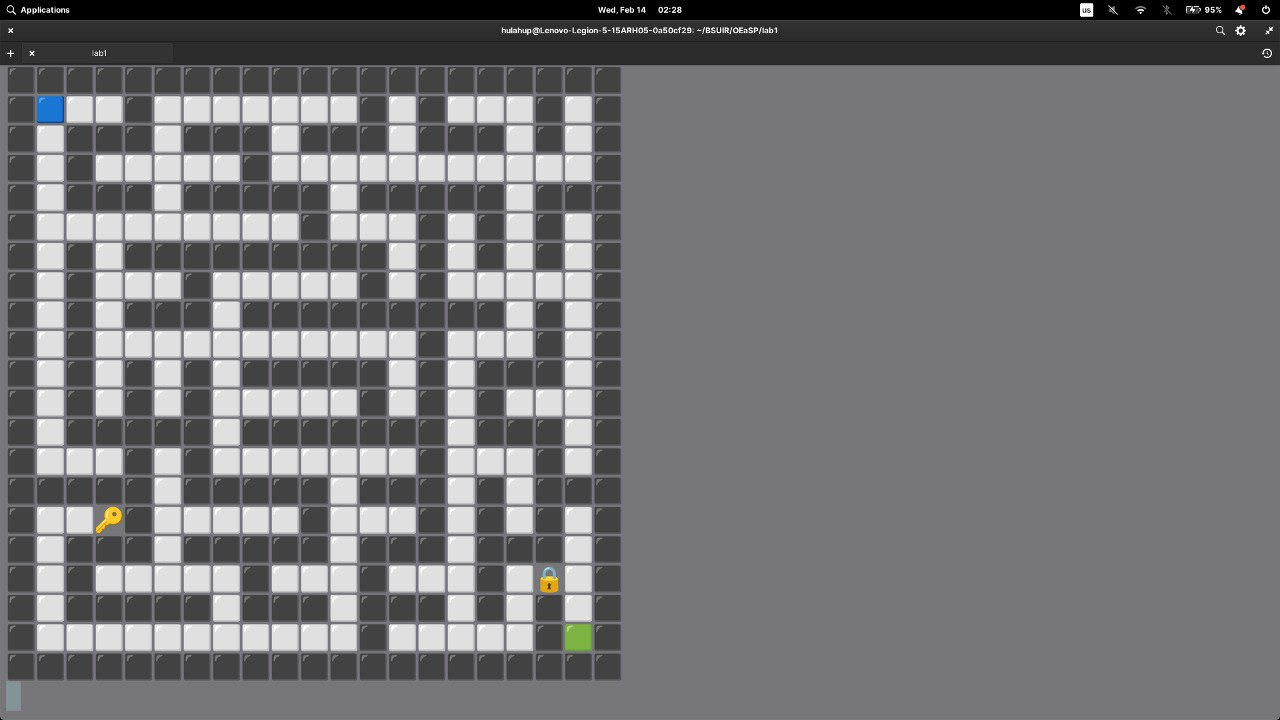


Рисунок 1 – Интерфейс игры

# ЗАКЛЮЧЕНИЕ

В ходе данной лабораторной работы были изучены элементы и конструкции скриптов shell: переменные, параметры, ветвления, циклы, вычисления, команды shell и вызовы внешних программ (shell, sed, awk, wget, различные фильтры и т.д.) для решения достаточно сложной задачи, имеющей практическое значение, а также принципов интеграции Unix-программ скриптами shell. Был написан скрипт для оболочки shell, который представляет собой реализацию игры по нахождению выхода из лабиринта.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Что такое Shell [Электронный ресурс]. – Режим доступа: https://losst.pro/chto-takoe-shell – Дата доступа: 11.02.2024.
2. *Bash* скрипты [Электронный ресурс]. – Режим доступа: https://habr.com/ru/companies/ruvds/articles/325522/. – Дата доступа: 11.02.2024.
3. *Introduction to Bash Scripting* [Электронный ресурс]. – Режим доступа: https://bjpcjp.github.io/pdfs/bash/bash-scripting.pdf. – Дата доступа: 11.02.2024.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – Файл game

#!/bin/bash

stty intr ^?

maze\_file="maze.txt"

screamer\_file="prize.txt"

sound\_file="gg.wav"

player\_char="🟦"

wall\_char="⬛️"

exit\_char="🟩"

empty\_char="⬜️"

key\_char="🔑"

door\_char="🔒"

rows=$(wc -l < "$maze\_file")

cols=$(($(head -n 1 "$maze\_file" | wc -c) - 1))

player\_row=1

player\_col=1

door\_key=0

maze\_text=$(<"$maze\_file")

IFS=$'\n' read -r -d '' -a maze <<< "$maze\_text"

maze[$player\_row]=${maze[$player\_row]:0:$player\_col}@${maze[$player\_row]:$((player\_col + 1))}

display\_maze() {

graphic\_maze=${1//"#"/"$wall\_char"}

graphic\_maze=${graphic\_maze//" "/"$empty\_char"}

graphic\_maze=${graphic\_maze//"E"/"$exit\_char"}

graphic\_maze=${graphic\_maze//"K"/"$key\_char"}

graphic\_maze=${graphic\_maze//"D"/"$door\_char"}

graphic\_maze=${graphic\_maze//"@"/"$player\_char"}

clear

echo "$graphic\_maze"

}

finish() {

clear

cat "$screamer\_file"

aplay "$sound\_file"

clear

exit 0

}

read -rsn1 key

case "$key" in

"A")

if [[ "${maze[$((player\_row-1))]:$player\_col:1}" != "#" ]]; then

if [[ "${maze[$((player\_row-1))]:$player\_col:1}" == "D" && $door\_key -eq 0 ]]; then

return

fi

if [[ "${maze[$((player\_row-1))]:$player\_col:1}" == "K" ]]; then

door\_key=1

fi

if [[ "${maze[$((player\_row-1))]:$player\_col:1}" == "E" ]]; then

finish

fi

maze[$player\_row]=${maze[$player\_row]:0:$player\_col}" "${maze[$player\_row]:$((player\_col + 1))}

((player\_row--))

maze[$player\_row]=${maze[$player\_row]:0:$player\_col}@${maze[$player\_row]:$((player\_col + 1))}

fi

;;

"B")

if [[ "${maze[$((player\_row+1))]:$player\_col:1}" != "#" ]]; then

if [[ "${maze[$((player\_row+1))]:$player\_col:1}" == "D" && $door\_key -eq 0 ]]; then

return

fi

if [[ "${maze[$((player\_row+1))]:$player\_col:1}" == "K" ]]; then

door\_key=1

fi

if [[ "${maze[$((player\_row+1))]:$player\_col:1}" == "E" ]]; then

finish

fi

maze[$player\_row]=${maze[$player\_row]:0:$player\_col}" "${maze[$player\_row]:$((player\_col + 1))}

((player\_row++))

maze[$player\_row]=${maze[$player\_row]:0:$player\_col}@${maze[$player\_row]:$((player\_col + 1))}

fi

;;

"C")

if [[ "${maze[$player\_row]:$((player\_col+1)):1}" != "#" ]]; then

if [[ "${maze[$player\_row]:$((player\_col+1)):1}" == "D" && $door\_key -eq 0 ]]; then

return

fi

if [[ "${maze[$player\_row]:$((player\_col+1)):1}" == "K" ]]; then

door\_key=1

fi

if [[ "${maze[$player\_row]:$((player\_col+1)):1}" == "E" ]]; then

finish

fi

maze[$player\_row]=${maze[$player\_row]:0:$player\_col}" "${maze[$player\_row]:$((player\_col + 1))}

((player\_col++))

maze[$player\_row]=${maze[$player\_row]:0:$player\_col}@${maze[$player\_row]:$((player\_col + 1))}

fi

;;

"D")

if [[ "${maze[$player\_row]:$((player\_col-1)):1}" != "#" ]]; then

if [[ "${maze[$player\_row]:$((player\_col-1)):1}" == "D" && $door\_key -eq 0 ]]; then

return

fi

if [[ "${maze[$player\_row]:$((player\_col-1)):1}" == "K" ]]; then

door\_key=1

fi

if [[ "${maze[$player\_row]:$((player\_col-1)):1}" == "E" ]]; then

finish

fi

maze[$player\_row]=${maze[$player\_row]:0:$player\_col}" "${maze[$player\_row]:$((player\_col + 1))}

((player\_col--))

maze[$player\_row]=${maze[$player\_row]:0:$player\_col}@${maze[$player\_row]:$((player\_col + 1))}

fi

;;

\*)

;;

esac

maze\_str=$(IFS=$'\n'; echo "${maze[\*]}")

display\_maze "$maze\_str"

}

maze\_str=$(IFS=$'\n'; echo "${maze[\*]}")

display\_maze "$maze\_str"

while true; do

handle\_input

done