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

ОТЧЁТ

к лабораторной работе №1

на тему

**СКРИПТЫ SHELL**

Выполнил: студент гр.253504

Фроленко К.Ю.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

СОДЕРЖАНИЕ

[1  Формулировка задачи 3](#_Toc189133056)

[2  Краткие теоритеческие сведения 4](#_Toc189133057)

[3  Описание функицй программы 6](#_Toc189133058)

[3.1 Функция get\_color 6](#_Toc189133059)

[3.2 Функция generate\_new\_number 6](#_Toc189133060)

[3.3 Функция display\_grid 6](#_Toc189133061)

[3.4 Функция shift\_and\_merge 6](#_Toc189133062)

[3.5 Функции move\_up, move\_down, move\_left, move\_right 7](#_Toc189133063)

[3.6 Функция check\_game\_over 7](#_Toc189133064)

[3.7 Функции update\_leaderboard и display\_leaderboard 7](#_Toc189133065)

[3.8 Функция main 7](#_Toc189133066)

[4  Пример выполнения программы 8](#_Toc189133067)

[4.1 Запуск программы и процесс выполнения 8](#_Toc189133068)

[Вывод 10](#_Toc189133069)

[Список использованных источников 11](#_Toc189133070)

[Приложение А (обязательное) 12](#_Toc189133071)

1 ФОРМУЛИРОВКА ЗАДАЧИ

Целью лабораторной работы является изучение и практическое освоение элементов и конструкций скриптов *shell*, таких как переменные, параметры, ветвления, циклы, арифметические вычисления, а также использование встроенных команд *shell* и вызовов внешних программ (*sed*, *awk*, *wget*, различные фильтры и другие инструменты *Unix*).

В рамках лабораторной работы требуется разработать консольную версию игры 2048, реализованную с использованием *shell*-скриптов. Программа должна обеспечивать создание и отображение игрового поля размером 4x4 в текстовом формате, используя символы и числа в консоли. Обработка ввода от пользователя осуществляется с помощью клавиш *wasd*, что позволяет управлять игровым процессом. Реализация механики объединения чисел должна соответствовать правилам игры 2048, включая корректное обновление состояния игрового поля после каждого хода и появление новых значений.

Также необходимо реализовать проверку условий окончания игры, включая достижение плитки 2048, что означает победу, и ситуацию, когда отсутствуют возможные ходы, что приводит к завершению игры. Помимо этого, в рамках задания требуется создать систему хранения и отображения лучших результатов. По завершении игры пользователю должен демонстрироваться список сохраненных рекордов, которые записываются в файл.

Разработка может быть выполнена с использованием различных интерпретаторов командной оболочки, таких как *bash*, *zsh*, *csh*. При этом следует учитывать, что размер окна консоли является фиксированным и не изменяется в процессе выполнения игры.

Результатом выполнения лабораторной работы станет углубленное понимание возможностей и механизмов *shell*-программирования, а также практическое освоение методов обработки данных, автоматизации процессов и взаимодействия с системными утилитами *Unix*.

2 КРАТКИЕ ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ

*Shell*-скрипты в *Bash* являются мощным инструментом для автоматизации задач, взаимодействия с операционной системой и управления системными процессами. Они позволяют выполнять последовательности команд, использовать переменные, управлять потоками выполнения программ и обрабатывать данные. В отличие от компилируемых языков программирования, *Bash*-скрипты интерпретируются, что упрощает их использование и отладку. Их применение особенно актуально при создании утилит, автоматизации работы серверов, обработки текстовых файлов и выполнения фоновых задач. Встроенные средства командной оболочки позволяют выполнять операции без необходимости подключения сторонних программ, что делает *Bash*-скрипты удобным инструментом для администрирования и разработки системных приложений [1].

Одним из ключевых аспектов работы в *Bash* является использование переменных и параметров. Переменные могут быть локальными и глобальными, содержать строки или числа, а также передаваться в качестве аргументов при запуске скрипта. Существует несколько способов присваивания значений переменным, включая считывание данных от пользователя, обработку выходных данных других команд и автоматическое присвоение системных параметров. Работа с условными конструкциями (*if*, *case*) и циклами (*for*, *while*, *until*) позволяет реализовывать сложную логику, реагируя на изменение данных и управляющих условий. Например, условные операторы помогают анализировать пользовательский ввод, определять корректность данных и выполнять соответствующие действия в зависимости от результата проверки. Использование циклов в скриптах дает возможность обрабатывать массивы данных, выполнять повторяющиеся операции и управлять процессами без необходимости вручную прописывать каждую команду [2].

Для обработки данных и текстовых файлов широко применяются встроенные утилиты, такие как *sed* и *awk*. *sed* используется для потокового редактирования текстов, позволяя изменять, заменять или удалять строки, тогда как *awk* предназначен для более сложных манипуляций с табличными данными. Важную роль в обработке информации играют стандартные фильтры, такие как *grep*, *cut*, *sort*, *uniq*, которые позволяют выполнять поиск, разбиение строк, сортировку и устранение дубликатов. В сочетании с регулярными выражениями эти инструменты обеспечивают гибкость при обработке больших объемов текстовой информации, что особенно полезно при анализе лог-файлов, создании отчетов и обработке входных данных [3].

*Bash*-скрипты обеспечивают эффективное управление процессами и файловой системой. Для работы с файлами используются команды *touch*, *cp*, *mv*, *rm*, *chmod*, *chown*, которые позволяют создавать, копировать, перемещать и удалять файлы, а также изменять их права доступа. Управление процессами осуществляется через команды *ps*, *kill*, *jobs*, *bg*, *fg*, что дает возможность отслеживать фоновые задачи и управлять их выполнением. Важной особенностью *Bash* является возможность управления параллельным выполнением команд, что позволяет запускать ресурсоемкие процессы в фоновом режиме и контролировать их выполнение, используя команды *nohup* и *disown*. Это полезно при работе с серверами и длительно выполняемыми задачами, где важно минимизировать задержки и поддерживать стабильность работы системы.

Создание консольных приложений, таких как игра 2048, требует динамического взаимодействия с пользователем и управления вводом-выводом в терминале. Для этого в *Bash* используется команда *read*, позволяющая считывать пользовательский ввод, а также *echo*, *printf* и управляющие последовательности *ANSI*, обеспечивающие визуальное оформление вывода. Важным инструментом является *tput*, позволяющий управлять позиционированием курсора, цветом текста и другими параметрами отображения. Применение специальных управляющих последовательностей дает возможность реализовать динамическое обновление экрана без перерисовки всей консоли, что делает игру более удобной для восприятия пользователем. Помимо этого, в *Bash* можно использовать псевдографику, например, символы *Unicode* для создания границ игрового поля, что улучшает внешний вид приложения без привлечения сложных графических библиотек.

Хранение и обработка данных в *Bash*-скриптах могут выполняться с использованием файлов. Запись данных осуществляется через операторы > и >>, а чтение – с помощью *cat*, *less* или *awk*. Для обеспечения безопасности и предотвращения одновременной записи нескольких процессов могут использоваться механизмы блокировки, например, команда *flock*. Это особенно важно для сохранения лучших результатов игры, так как данные не должны повреждаться при множественных запусках программы.

Автоматизация *Bash*-скриптов позволяет запускать их по расписанию с помощью *cron*, а взаимодействие с сетью обеспечивается командами *wget* и *curl*. Эти инструменты позволяют загружать данные из сети, что может быть полезно, например, для обновления игрового рейтинга или получения информации из внешних источников.

Использование *Bash*-скриптов в разработке приложений дает возможность гибко управлять системными процессами, эффективно обрабатывать данные и автоматизировать выполнение рутинных задач. В рамках лабораторной работы изучение этих механизмов позволит реализовать консольную версию игры 2048 с динамическим обновлением игрового поля, управлением вводом и сохранением лучших результатов. Это обеспечит практический опыт работы с *Bash* и углубленное понимание возможностей автоматизации в операционной системе *Linux*.

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

В рамках разработки консольной версии игры 2048 на *Bash* были реализованы следующие функции:

– функция *get\_color*;

– функция *generate\_new\_number*;

– функция *display\_grid*;

– функция *shift\_and\_merge*;

– функции *move\_up*, *move\_down*, *move\_left*, *move\_right*;

– функция *check\_game\_over*;

– функция *update\_leaderboard* и *display\_leaderboard*;

– функция *main*.

3.1 Функция get\_color

Функция *get\_color* предназначена для выбора цветового оформления чисел на игровом поле. В зависимости от переданного числа она возвращает соответствующую *ANSI*-последовательность для изменения цвета текста в терминале. Это позволяет создать более наглядное отображение игры, где разные числа имеют различное цветовое оформление, помогая игроку быстрее ориентироваться в текущем состоянии игрового поля.

3.2 Функция generate\_new\_number

Функция generate\_new\_number отвечает за генерацию нового числа на игровом поле. Она находит все свободные ячейки и случайным образом выбирает одну из них для размещения нового числа (2 или 4). Генерация происходит после каждого успешного хода, что обеспечивает непрерывность игрового процесса.

3.3 Функция display\_grid

Функция display\_grid отрисовывает текущее состояние игрового поля в консоли. Она использует цветовое оформление чисел и рамки, чтобы обеспечить визуальную структуру сетки 4x4. Также выводится текущий счет игрока.

3.4 Функция shift\_and\_merge

Функция *shift\_and\_merge* выполняет сдвиг и объединение чисел в строке или столбце. Сначала она сдвигает все ненулевые элементы к началу массива, а затем объединяет соседние одинаковые элементы, удваивая их значение и добавляя очки к счету. После этого оставшееся пространство заполняется нулями.

3.5 Функции move\_up, move\_down, move\_left, move\_right

Эти функции обрабатывают ввод игрока и вызывают функцию *shift\_and\_merge* для выполнения сдвига и объединения чисел в соответствующем направлении. Если хотя бы один элемент изменился в результате перемещения, вызывается функция *generate\_new\_number* для добавления нового числа на поле.

3.6 Функция check\_game\_over

Функция *check\_game\_over* проверяет, завершена ли игра. Если на поле есть хотя бы одна пустая ячейка или два соседних элемента с одинаковыми значениями, игра продолжается. В противном случае игра завершается, и вызывается функция *update\_leaderboard*.

3.7 Функции update\_leaderboard и display\_leaderboard

Функция *update\_leaderboard* сохраняет лучший результат игрока в файле *leaderboard.txt*. Она добавляет текущий счет в список рекордов, сортирует его и оставляет только 10 лучших значений. Функция *display\_leaderboard* отображает таблицу рекордов после завершения игры.

3.8 Функция main

Функция *main* управляет игровым циклом. Она выполняет инициализацию игрового поля, отображает текущее состояние и обрабатывает ввод игрока (*W*/*A*/*S*/*D*). После каждого успешного хода обновляется игровая сетка, проверяется возможность дальнейшей игры, а в случае завершения вызываются функции сохранения и отображения рекордов. Если игра окончена, программа выводит итоговый счет и завершает выполнение.

4 ПРИМЕР ВЫПОЛНЕНИЯ ПРОГРАММЫ

4.1 Запуск программы и процесс выполнения

В результате выполнения программы была успешно разработана консольная версия игры 2048, реализованная с использованием *Bash*-скрипта. Программа запускается в терминале и предоставляет удобный интерфейс управления с помощью клавиш *W*, *A*, *S*, *D*.

На рисунке 4.1 представлено начальное состояние игрового поля сразу после запуска программы. В этот момент на поле появляются два случайных числа (обычно 2 или 4), с которых начинается игра.

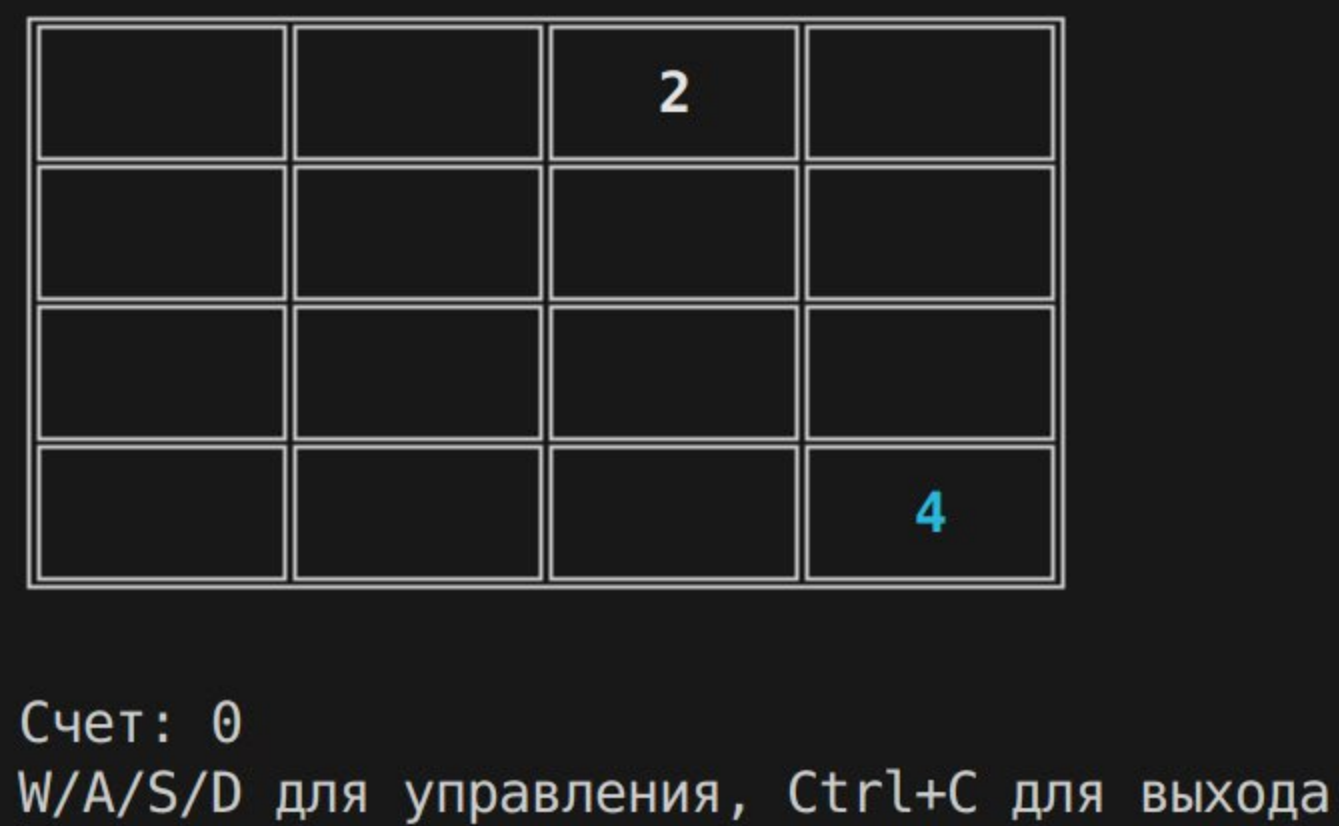
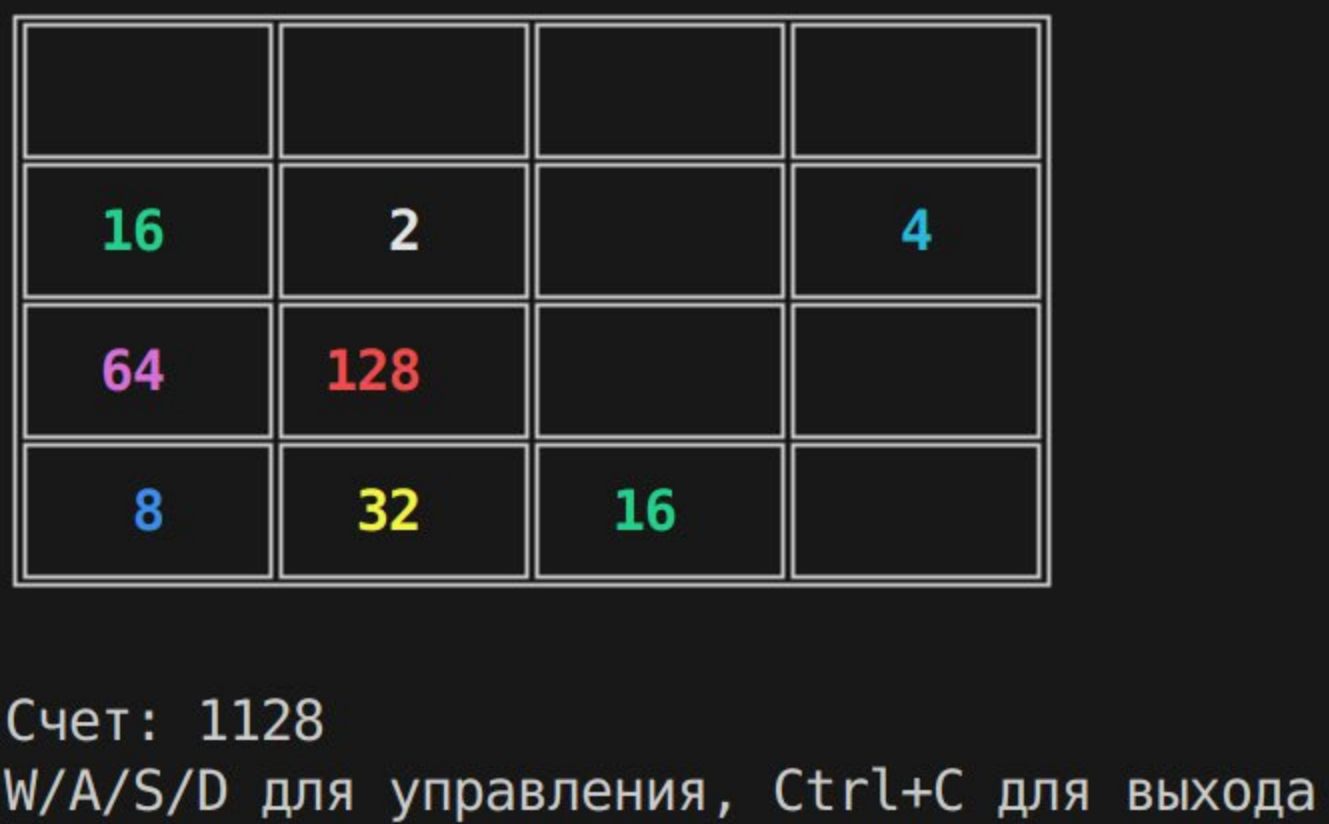


Рисунок 4.1 – Начальное состояние игрового поля

Каждый ход приводит к обновлению игрового поля, а текущий счет отображается в нижней части экрана. Пользователь может перемещать плитки в четырех направлениях, объединяя одинаковые числа. Визуальное оформление обеспечивается с помощью *ANSI*-кодов, что помогает игроку быстро ориентироваться в происходящем на поле.

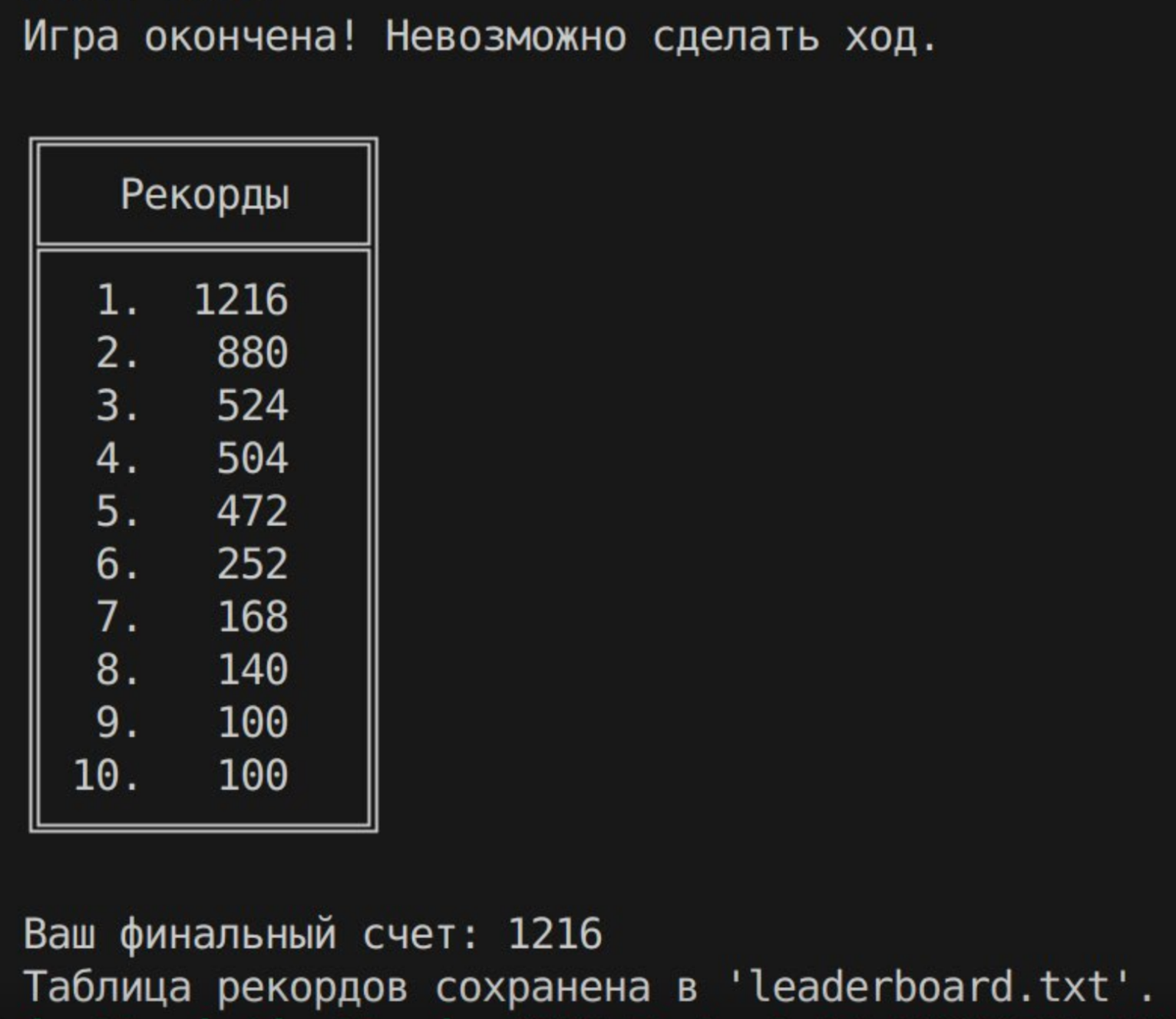
На рисунке 4.2 показан процесс игры после нескольких ходов.



**Р**исунок 4.2 – Процесс игры после нескольких ходов

После завершения игры программа проверяет возможность внесения результата в таблицу рекордов. Если текущий счет входит в десятку лучших, он сохраняется в файле ***leaderboard.txt***. Этот файл содержит список наилучших достижений, упорядоченный по убыванию, и может быть просмотрен в любой момент.

На рисунке 4.3 представлена таблица рекордов, в которой сохраняются наивысшие достигнутые результаты.



**Р**исунок 4.2 – Таблица рекордов после завершения игры

Программа обрабатывает возможные ошибки, такие как некорректный ввод или невозможность выполнить очередной ход. В случае завершения игры из-за отсутствия доступных действий выводится соответствующее уведомление. Все возможные исключения сопровождаются информативными сообщениями, обеспечивающими удобство использования.

Таким образом, разработанная консольная версия игры 2048 является удобным и функциональным инструментом для развлечения, позволяющим пользователям проверить свои стратегические навыки в классической игре 2048.

# ВЫВОД

В ходе выполнения лабораторной работы была успешно разработана консольная версия игры 2048, реализованная с использованием *Bash*-скрипта. Программа позволяет пользователю управлять игровым процессом с помощью клавиш *W*, *A*, *S*, *D*, динамически обновлять состояние игрового поля и отслеживать текущий счет. Реализация механики объединения чисел соответствует правилам игры 2048, что позволило достичь корректной работы алгоритмов сдвига, объединения плиток и генерации новых чисел.

Разработка программы позволила закрепить знания о структуре и возможностях *Bash*-скриптов, включая использование переменных, массивов, циклов, условий и встроенных команд. В процессе работы также были изучены и применены инструменты для обработки пользовательского ввода, управления цветовым оформлением вывода и работы с файловой системой. Важным аспектом реализации стало создание системы хранения и отображения лучших результатов, что позволило продемонстрировать навыки работы с файлами в *Bash*.

Особое внимание в разработке было уделено обработке ошибок, таких как некорректный ввод или невозможность выполнения очередного хода. Программа выводит понятные пользователю сообщения, обеспечивая удобство взаимодействия. Реализация механизма сохранения рекордов в файле ***leaderboard.txt*** позволяет сохранять и просматривать результаты предыдущих игр, что делает программу более функциональной и удобной для использования.

Таким образом, поставленные перед лабораторной работой задачи были успешно решены. Разработанная программа является эффективной реализацией игры 2048 в терминале и демонстрирует возможности *Bash* для создания интерактивных приложений. Полученные знания и навыки могут быть применены в дальнейшем для разработки скриптов автоматизации, работы с текстовыми данными и создания других консольных приложений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] LinuxConfig.org: Bash Scripting Tutorial: How to Write a Bash Script [Электронный ресурс]. – Режим доступа: <https://linuxconfig.org/bash-scripting-tutorial>. – Дата доступа: 30.01.2025.

[2] freeCodeCamp.org: Bash Scripting Tutorial – Linux Shell Script and Command Line for Beginners [Электронный ресурс]. – Режим доступа: <https://www.freecodecamp.org/news/bash-scripting-tutorial-linux-shell-script-and-command-line-for-beginners/>. – Дата доступа: 30.01.2025.

[3] CHARMM-GUI: Unix Tutorial [Электронный ресурс]. – Режим доступа: https://charmm-gui.org/?doc=lecture&lesson=10&module=unix. – Дата доступа: 30.01.2025.

ПРИЛОЖЕНИЕ А

(обязательное)

Исходный код программы

#!/bin/bash

shopt -s extglob

RESET=$'\033[0m'

COLOR\_2=$'\033[1;37m' # Белый

COLOR\_4=$'\033[1;36m' # Голубой

COLOR\_8=$'\033[1;34m' # Синий

COLOR\_16=$'\033[1;32m' # Зеленый

COLOR\_32=$'\033[1;33m' # Желтый

COLOR\_64=$'\033[1;35m' # Фиолетовый

COLOR\_128=$'\033[1;31m' # Красный

COLOR\_256=$'\033[0;32m' # Темно-зеленый

COLOR\_512=$'\033[0;33m' # Темно-желтый

COLOR\_1024=$'\033[0;36m' # Темно-голубой

COLOR\_2048=$'\033[0;34m' # Темно-синий

COLOR\_4096=$'\033[0;35m' # Темно-фиолетовый

COLOR\_8192=$'\033[0;37m' # Серый

COLOR\_16384=$'\033[1;90m' # Светло-серый

COLOR\_32768=$'\033[1;91m' # Светло-красный

COLOR\_65536=$'\033[1;92m' # Светло-зеленый

COLOR\_131072=$'\033[1;93m' # Светло-желтый

COLOR\_DEFAULT=$'\033[1;94m' # Светло-синий

get\_color() {

case "$1" in

2) echo -n "$COLOR\_2" ;;

4) echo -n "$COLOR\_4" ;;

8) echo -n "$COLOR\_8" ;;

16) echo -n "$COLOR\_16" ;;

32) echo -n "$COLOR\_32" ;;

64) echo -n "$COLOR\_64" ;;

128) echo -n "$COLOR\_128" ;;

256) echo -n "$COLOR\_256" ;;

512) echo -n "$COLOR\_512" ;;

1024) echo -n "$COLOR\_1024" ;;

2048) echo -n "$COLOR\_2048" ;;

4096) echo -n "$COLOR\_4096" ;;

8192) echo -n "$COLOR\_8192" ;;

16384) echo -n "$COLOR\_16384" ;;

32768) echo -n "$COLOR\_32768" ;;

65536) echo -n "$COLOR\_65536" ;;

131072) echo -n "$COLOR\_131072" ;;

\*) echo -n "$COLOR\_DEFAULT" ;;

esac

}

declare -A grid

for ((i=0; i<4; i++)); do

for ((j=0; j<4; j++)); do

grid[$i,$j]=0

done

done

score=0

leaderboard\_file="leaderboard.txt"

leaderboard=()

if [ -f "$leaderboard\_file" ]; then

mapfile -t leaderboard < <(sort -nr "$leaderboard\_file" | head -n 10)

else

touch "$leaderboard\_file"

fi

update\_leaderboard() {

leaderboard+=("$score")

leaderboard=($(printf "%s\n" "${leaderboard[@]}" | sort -nr))

leaderboard=("${leaderboard[@]:0:10}")

printf "%s\n" "${leaderboard[@]}" > "$leaderboard\_file"

}

display\_leaderboard() {

echo ""

echo "╔═════════════╗"

echo "║ Рекорды ║"

echo "╠═════════════╣"

local max\_leaderboard\_lines=10

for ((k=0; k<max\_leaderboard\_lines; k++)); do

if [ $k -lt ${#leaderboard[@]} ]; then

printf "║ %2d. %5d ║\n" "$((k +1))" "${leaderboard[$k]}"

else

printf "║ %2d. ------ ║\n" "$((k +1))"

fi

done

echo "╚═════════════╝"

echo ""

}

generate\_new\_number() {

local empty=()

for ((i=0; i<4; i++)); do

for ((j=0; j<4; j++)); do

if [ "${grid[$i,$j]}" -eq 0 ]; then

empty+=("$i,$j")

fi

done

done

if [ ${#empty[@]} -ne 0 ]; then

local rand\_index=$((RANDOM % ${#empty[@]}))

local pos=(${empty[$rand\_index]//,/ })

local value=$(( (RANDOM % 2 + 1) \* 2 ))

grid[${pos[0]},${pos[1]}]=$value

fi

}

display\_grid() {

clear

echo "╔═══════╦═══════╦═══════╦═══════╗"

for ((i=0; i<4; i++)); do

echo -n "║"

for ((j=0; j<4; j++)); do

if [ "${grid[$i,$j]}" -ne 0 ]; then

value="${grid[$i,$j]}"

color=$(get\_color "$value")

printf " %s%3d%s ║" "$color" "$value" "$RESET"

else

printf " ║"

fi

done

echo

if [ "$i" -lt 3 ]; then

echo "╠═══════╬═══════╬═══════╬═══════╣"

else

echo "╚═══════╩═══════╩═══════╩═══════╝"

fi

done

echo ""

echo "Счет: $score"

}

shift\_and\_merge() {

local -n merged\_line\_ref=$1

shift

local line=("$@")

merged\_line\_ref=()

local new\_line=()

for num in "${line[@]}"; do

if [ "$num" -ne 0 ]; then

new\_line+=("$num")

fi

done

local i=0

while [ $i -lt ${#new\_line[@]} ]; do

if [ $((i + 1)) -lt ${#new\_line[@]} ] && [ "${new\_line[$i]}" -eq "${new\_line[$i+1]}" ]; then

local merged\_value=$((new\_line[$i] \* 2))

merged\_line\_ref+=("$merged\_value")

score=$((score + merged\_value))

i=$((i + 2))

else

merged\_line\_ref+=("${new\_line[$i]}")

i=$((i + 1))

fi

done

while [ ${#merged\_line\_ref[@]} -lt 4 ]; do

merged\_line\_ref+=(0)

done

}

moved=0

move\_up() {

moved=0

for ((j=0; j<4; j++)); do

local line=()

for ((i=0; i<4; i++)); do

line+=("${grid[$i,$j]}")

done

shift\_and\_merge merged\_line "${line[@]}"

for ((i=0; i<4; i++)); do

if [ "${grid[$i,$j]}" -ne "${merged\_line[$i]}" ]; then

moved=1

fi

grid[$i,$j]=${merged\_line[$i]}

done

done

}

move\_down() {

moved=0

for ((j=0; j<4; j++)); do

local line=()

for ((i=3; i>=0; i--)); do

line+=("${grid[$i,$j]}")

done

shift\_and\_merge merged\_line "${line[@]}"

for ((i=3, k=0; i>=0; i--, k++)); do

if [ "${grid[$i,$j]}" -ne "${merged\_line[$k]}" ]; then

moved=1

fi

grid[$i,$j]=${merged\_line[$k]}

done

done

}

move\_left() {

moved=0

for ((i=0; i<4; i++)); do

local line=()

for ((j=0; j<4; j++)); do

line+=("${grid[$i,$j]}")

done

shift\_and\_merge merged\_line "${line[@]}"

for ((j=0; j<4; j++)); do

if [ "${grid[$i,$j]}" -ne "${merged\_line[$j]}" ]; then

moved=1

fi

grid[$i,$j]=${merged\_line[$j]}

done

done

}

move\_right() {

moved=0

for ((i=0; i<4; i++)); do

local line=()

for ((j=3; j>=0; j--)); do

line+=("${grid[$i,$j]}")

done

shift\_and\_merge merged\_line "${line[@]}"

for ((j=3, k=0; j>=0; j--, k++)); do

if [ "${grid[$i,$j]}" -ne "${merged\_line[$k]}" ]; then

moved=1

fi

grid[$i,$j]=${merged\_line[$k]}

done

done

}

check\_game\_over() {

for ((i=0; i<4; i++)); do

for ((j=0; j<4; j++)); do

if [ "${grid[$i,$j]}" -eq 0 ]; then

return 1

fi

if [ $i -lt 3 ] && [ "${grid[$i,$j]}" -eq "${grid[$i+1,$j]}" ]; then

return 1

fi

if [ $j -lt 3 ] && [ "${grid[$i,$j]}" -eq "${grid[$i,$j+1]}" ]; then

return 1

fi

done

done

return 0

}

generate\_new\_number

generate\_new\_number

while true; do

display\_grid

echo "W/A/S/D для управления, Ctrl+C для выхода"

read -n 1 -s key

key=${key,,}

case "$key" in

w)

move\_up

;;

s)

move\_down

;;

a)

move\_left

;;

d)

move\_right

;;

\*)

echo "Нераспознанная клавиша: '$key'. Используйте W/A/S/D."

continue

;;

esac

if [ "$moved" -eq 1 ]; then

generate\_new\_number

fi

check\_game\_over

if [ $? -eq 0 ]; then

display\_grid

echo "Игра окончена! Невозможно сделать ход."

update\_leaderboard

display\_leaderboard

echo "Ваш финальный счет: $score"

echo "Таблица рекордов сохранена в '$leaderboard\_file'."

exit

fi

moved=0

done