

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 2025

СОДЕРЖАНИЕ

1	Формулировка задачи	3
2	Краткие теоретические сведения.....	4
3	Описание функций программы.....	6
3.1	Функция get_color	6
3.2	Функция generate_new_number	6
3.3	Функция display_grid	6
3.4	Функция shift_and_merge.....	6
3.5	Функции move_up, move_down, move_left, move_right	7
3.6	Функция check_game_over	7
3.7	Функции update_leaderboard и display_leaderboard.....	7
3.8	Функция main.....	7
4	Пример выполнения программы	8
4.1	Запуск программы и процесс выполнения	8
	Вывод.....	10
	Список использованных источников	11
	Приложение А (обязательное)	12

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.

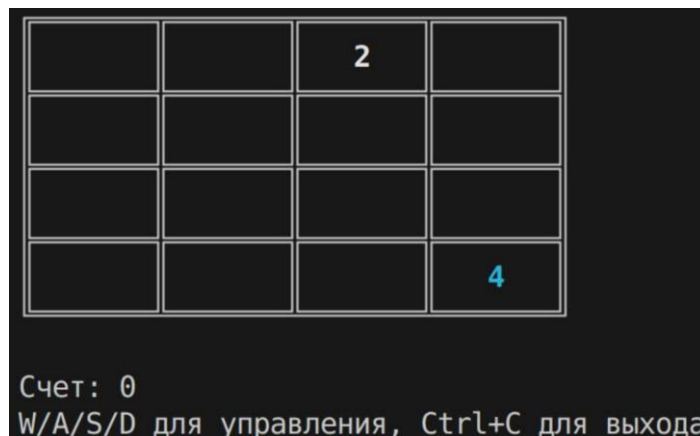


Рисунок 4.1 – Начальное состояние игрового поля

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.

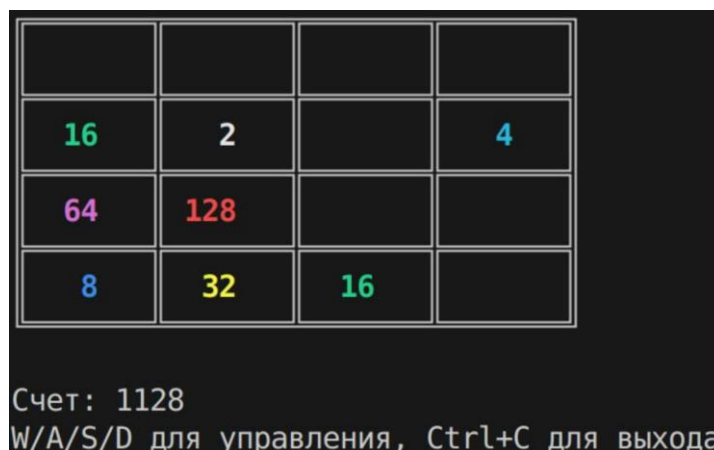
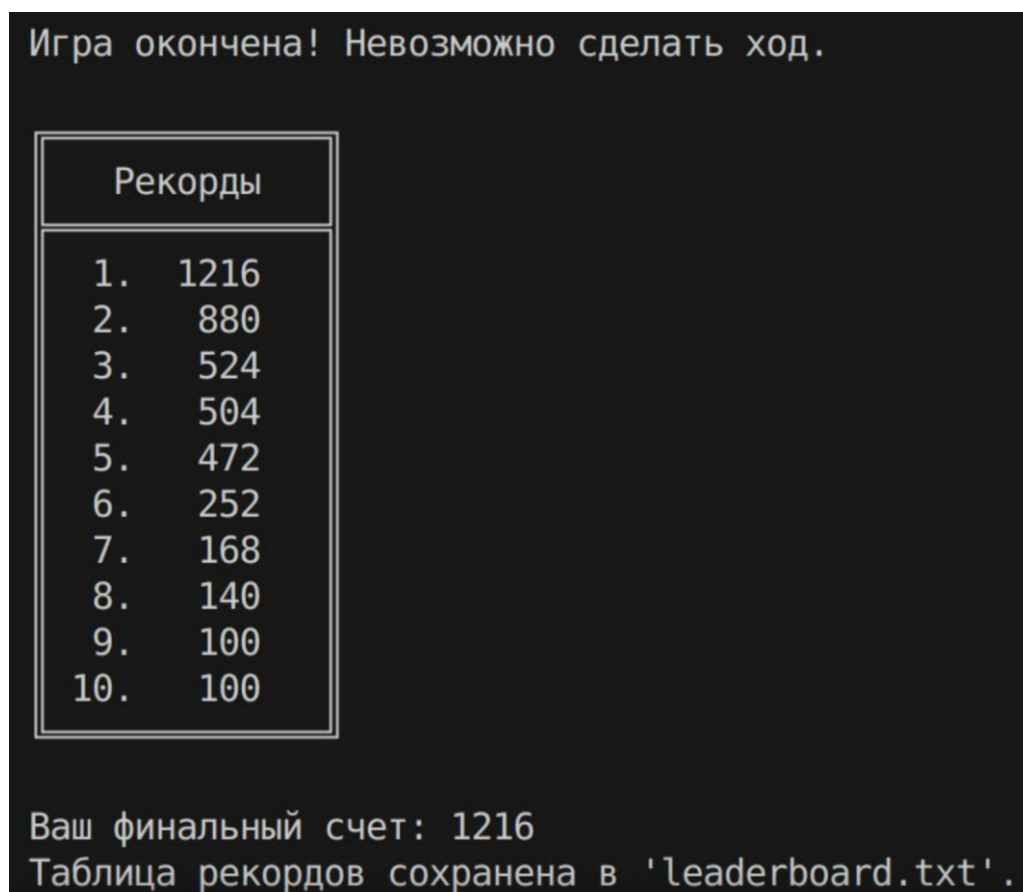


Рисунок 4.2 – Процесс игры после нескольких ходов

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.



Игра окончена! Невозможно сделать ход.

Рекорды	
1.	1216
2.	880
3.	524
4.	504
5.	472
6.	252
7.	168
8.	140
9.	100
10.	100

Ваш финальный счет: 1216
Таблица рекордов сохранена в 'leaderboard.txt'.

Рисунок 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 "
    echo "
    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
        fi
    done
}

```

```

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

```