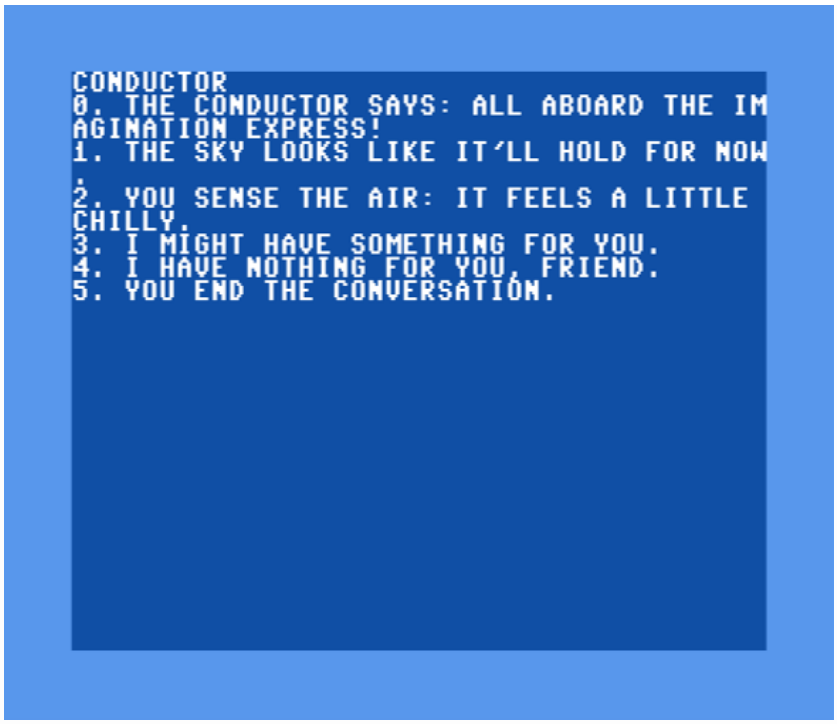
Everland — Manual

Overview

Everland is a single-file Commodore 64 text-adventure written in Kick Assembler. Build with KickAssembler and run in the VICE C64 emulator on Windows.

Screenshots

The following screenshots are included from the `images/` folder.



Fully Working Features

- Save/Load (Disk I/O): Persistent profile save/load using a custom binary format.
- Login/Profile: Username/display and class selection at first login.
- Class & Level System: Player class, level, and score; class base HP and HP-per-level.
- HP Tracking & Persistence: Player and NPC current HP tracked and saved.
- Inventory Screen: Full-screen Inventory accessible with `I`.
- Characters Menu & Sheets: `C` opens character menu; view player or NPC sheet.
- Talk/NPC Selection: `T` opens talk screen; select NPCs and begin conversations.
- Conversation Menu: Options implemented (speak, ask weather, comment temp, request quest, quest info, end).
- NPC Data & Stats: Arrays for NPC name, class, level, score, current HP.
- PETSCII Helpers & Rendering: Message buffer rendering and PETSCII helpers.
- KERNAL Routines: File handling and console I/O routines used.
- IRQ / Music Hook: Music IRQ handler and safe IRQ stub scaffold present.

Partially Working / Needs Attention

- Assembler Branch/Label Issues: Recent trampoline edits required fixes; re-verify with a full assemble.
- Conversation/Trampoline Robustness: Conversation menu logic works but needs final branch/label cleanup.
- SID Music Content: IRQ handler exists but SID tracks are not fully integrated.
- Quest Persistency Edge Cases: Basic quest assignment and save/load implemented; complex flows need expansion/testing.
- UI Polish: Dialog strings and message layout may need trimming.
- Multiplayer/Net: Not functional.

Beginner Player Manual

Startup / Compile / Run

- Assemble with KickAssembler on Windows (example):

```
java -cp C:\commodore\KickAssembler\KickAss.jar kickass.KickAssembler -odi
```

- Run in VICE: open `bin\everland.prg` or drag into x64.

First Run / Profiles

- On first run you'll be prompted for a username/display and to pick a player class. The chosen class affects HP and is saved.

Core Controls

- Movement: N, S, E, W (note: `S` is South only).
- `C`: Characters menu — view player or NPC sheets.
- `I`: Inventory (full screen).
- `T`: Talk — select NPC and start conversation.
- `ATTACK` / `FIGHT`: Melee attack an NPC at your current location. Use `ATTACK <npc>` or `FIGHT <npc>` to target someone.
- Conversation: Type menu number and press Enter; choose “End” to exit.

Gameplay Loop

- Explore map, talk to NPCs (`T`), check `C`haracter sheet and `I`nventory, accept quests, gain score/levels. Progress and HP changes are auto-saved on key events.
- Combat: Use `ATTACK <NPC>` to perform a simple melee attack. NPC and player HP are tracked and persisted to your profile; defeats remove NPCs from the location and update save state.
- Combat: Use `ATTACK <NPC>` to perform a simple melee attack. NPC and player HP are tracked and persisted to your profile; defeats remove NPCs from the location and update save state. Defeating foes now grants small XP rewards (every 10 XP increases your level) and increments your score.

Saving & Loading

- Saves use KERNAL file routines; the game auto-loads profile on startup and auto-saves on key changes.

Troubleshooting

- If KickAssembler reports branch/label errors, re-run and paste the build log; I can patch `everland.asm` to fix trampolines.
- If VICE hangs, try restarting the emulator or ensure the IRQ safe stub is enabled in code.

Upcoming / Recommended Features

- Full SID soundtrack and sound effects.
- Expanded quest system and quest log UI.
- Turn-based combat, skills, and status effects.
- Improved inventory with equip/unequip and item stacking.
- Enhanced PETSCII map, mini-map, and animations.
- Multiple save slots and in-game save manager.
- Joystick/controller support.
- Reorganize code to avoid long-branch patterns and improve assembler stability.

Build & Environment Notes

- Assembler: Kick Assembler (v5.25 in logs).
- Editor: Visual Studio Code.
- Emulator: VICE C64 Emulator on Windows.
- Source: `everland.asm` in the project root. Build output in `bin/`.

Credits

- Producer: Damon Hogan
- Inspired by: Perry Fraptic (RetroRecipies YouTube)
- Hardware inspiration: Commodore 64 Ultimate, commodore.net
- Tooling: Kick Assembler, Visual Studio Code, VICE

Contact / Next Steps

- To include screenshots: upload them into `images/` then re-run the PDF conversion instructions below.
- If you’d like, I can patch remaining assembler errors — paste the latest build log.

Appendix: Commands Used To Create The PDF

The PDF for this manual was produced using `pandoc` to convert Markdown to PDF.
Below are the exact commands you can run locally to reproduce the same searchable PDF.

1. Convert Markdown directly to PDF via LaTeX (recommended if you have a TeX engine installed):

```
pandoc MANUAL.md -o MANUAL.pdf --pdf-engine=xelatex --toc
```

2. Alternative: Generate HTML then convert to PDF with `wkhtmltopdf` (if you prefer HTML rendering):

```
# Convert Markdown -> HTML
pandoc MANUAL.md -o MANUAL.html --standalone --toc
# Convert HTML -> searchable PDF with wkhtmltopdf
wkhtmltopdf MANUAL.html MANUAL.pdf
```

3. If you need only HTML output (searchable in browsers):

```
pandoc MANUAL.md -o MANUAL.html --standalone --toc
```

Notes: - `--toc` generates a table of contents. - If images are large, consider resizing before embedding to reduce PDF size.

PDF Location

If I successfully generate `MANUAL.pdf` in the repository root, it will be placed at `c:/commodore/everland/MANUAL.pdf` and offered for download here.

Appendix B: Running the Automated VICE Playtest Script

This appendix describes how to run the provided playtest helpers (`tools/vice_playtest.ps1` and `tools/vice_playtest.bat`) to automatically launch VICE and run a short smoke-test sequence against `bin/everland.prg`.

1. Quick launcher (recommended)

- Use the batch file if you just want to start the emulator with the PRG. From a Command Prompt or PowerShell in the project root run:

```
tools\vice_playtest.bat "C:\Program Files (x86)\VICE\x64sc.exe" "c:\commod
```

If VICE is installed in the default path the arguments are optional:

```
tools\vice_playtest.bat
```

2. Full scripted playtest (PowerShell)

- The PowerShell script `tools/vice_playtest.ps1` launches x64sc, waits for the emulator window, brings it to the foreground and sends a small keystroke sequence to exercise a conversation flow (example: `TALK BARTENDER` → choose menu option `3`). To run it once without changing policy:

```
powershell -ExecutionPolicy Bypass -File .\tools\vice_playtest.ps1 -VicePa
```

- If the script is blocked by Windows Defender / SmartScreen or another AV, unblock it and re-run:

```
Unblock-File .\tools\vice_playtest.ps1
powershell -ExecutionPolicy Bypass -File .\tools\vice_playtest.ps1
```

Security and AV notes

- The PowerShell script sends keystrokes to the emulator window — some endpoint protection tools flag or block this behavior. If your AV blocks the script, prefer the batch launcher or add an exclusion for `tools\vice_playtest.ps1` in your AV settings.
- `-ExecutionPolicy Bypass` runs the script just for the process and does not permanently change system policy.

Script behavior and customization

- What the script does by default:
  - Launches x64sc with `-autostart <prg>` and `-warp` (fast boot).
  - Waits for the main emulator window and brings it to front.
  - Sends the keystrokes: `TALK BARTENDER{ENTER}` then `3{ENTER}` (selects the conversation menu's quest option).
  - Pauses and lets you inspect the emulator before exiting.
- You can modify the script to send other sequences (e.g., `TAKE COIN{ENTER}`, `GIVE COIN TO BARTENDER{ENTER}`) or longer test flows. Timings are configurable (delays

in milliseconds between SendKeys).

Automating screenshot capture (optional)

- The script does not capture screenshots by default. To capture screens you can:
  - Use a command-line screenshot tool (e.g., `nircmd.exe` or `magick` from ImageMagick) and call it after the SendKeys sequence.
  - Or use VICE's built-in screenshot feature (keyboard binding) and send that key sequence from the script (timing-sensitive).

Troubleshooting

- If VICE fails to start: verify the `-VicePath` you provided points at `x64sc.exe` (or `x64.exe`/`x64sc.exe` depending on your build).
- If the PRG doesn't autostart: ensure `bin\everland.prg` exists and is the output from the latest assembly.
- If SendKeys appear to be ignored: ensure the emulator window has keyboard focus and increase the script's delay values.

Extending the playtest harness

- I can add additional scripted flows (save/load, give item flows, combat sequence) and optional screenshot collection. Tell me which flows you'd like automated and I will add them to `tools/vice_playtest.ps1` as named scenarios.

Appendix C: Build environment & scripts (PDF automation)

This appendix documents the environment and scripts used to build the manual, and includes a text copy of the build wrapper for reproducibility.

1. Dependencies (for PDF creation)

- `pandoc` — the primary tool to convert Markdown to HTML/PDF. Install from https://pandoc.org/
- A TeX engine (one of):
  - `xelatex` (recommended) — part of TeX Live or MiKTeX; provides good Unicode and font handling.
  - `pdflatex` — alternative TeX engine (also part of TeX Live / MiKTeX).
- `wkhtmltopdf` (optional) — alternative path: convert Markdown -> HTML via `pandoc`, then HTML -> PDF via `wkhtmltopdf`.
- `wkhtmltopdf` (optional) — alternative path: convert Markdown -> HTML via `pandoc`, then HTML -> PDF via `wkhtmltopdf`.

wkhtmltopdf installation (Windows)

If you prefer converting via HTML -> PDF, install `wkhtmltopdf`. On Windows you can install it in several ways:

- Using Chocolatey (recommended if you have Chocolatey installed):

```
choco install wkhtmltopdf -y
```

- Using Scoop (if you use Scoop):

```
iwr -useb get.scoop.sh | iex
scoop install wkhtmltopdf
```

- Manual download (no package manager):

1. Download the Windows 64-bit `wkhtmltopdf` binary ZIP from the official releases page: https://github.com/wkhtmltopdf/wkhtmltopdf/releases
2. Extract `wkhtmltopdf.exe` from the ZIP and place it in a folder such as `C:\Program Files\wkhtmltopdf`.
3. Add that folder to your user PATH (or system PATH) and restart your shell.

Note about PowerShell execution policy

If `tools\build_manual.ps1` is blocked on your system, run it with `-ExecutionPolicy Bypass` so the wrapper can run without changing system policy permanently:

```
powershell -ExecutionPolicy Bypass -File .\tools\build_manual.ps1
```

After installing `wkhtmltopdf` and ensuring it's on your PATH, re-run the wrapper above — it will detect `wkhtmltopdf` and use it to convert the generated `MANUAL.html` to

`MANUAL.pdf`. - Docker (optional) — we provide a Dockerfile below to build the manual inside a container without installing TeX locally. See the `tools/Dockerfile` file. - Optional screenshot / helper tools (only for extended playtests): `nircmd.exe`, `ImageMagick (magick)`, or any command-line screenshot tool.

2. Build wrapper (text copy)

The repository includes `tools/build_manual.ps1`, a PowerShell wrapper that detects available engines and builds `MANUAL.pdf` if possible (falling back to `MANUAL.html` otherwise). The full script is below:

—- begin `build_manual.ps1` —-

—- end `build_manual.ps1` —-

Replace `<inserted-script>` with the following exact contents (copy-paste into a file):

```
<#
Build wrapper for MANUAL.md -> MANUAL.pdf
Detects available PDF engines (xelatex, pdflatex, wkhtmltopdf) and runs pa
Usage: powershell -ExecutionPolicy Bypass -File .\tools\build_manual.ps1
#>
param(
    [string]$ManualMd = "MANUAL.md",
    [string]$OutPdf = "MANUAL.pdf",
    [string]$OutHtml = "MANUAL.html"
)

function Which($name) {
    $c = Get-Command $name -ErrorAction SilentlyContinue
    if ($c) { return $c.Source }
    return $null
}

$pandoc = Which "pandoc"
if (-not $pandoc) {
    Write-Host "pandoc not found in PATH. Please install pandoc." -Foregro
    exit 2
}

$xelatex = Which "xelatex"
$pdflatex = Which "pdflatex"
$wkhtmltopdf = Which "wkhtmltopdf"

Write-Host "Found: pandoc=$pandoc" -ForegroundColor Cyan
if ($xelatex) { Write-Host "Found xelatex: $xelatex" -ForegroundColor Cyan
if ($pdflatex) { Write-Host "Found pdflatex: $pdflatex" -ForegroundColor C
if ($wkhtmltopdf) { Write-Host "Found wkhtmltopdf: $wkhtmltopdf" -Foregrou

if ($xelatex) {
    Write-Host "Building PDF via xelatex..." -ForegroundColor Green
    & $pandoc $ManualMd -o $OutPdf --pdf-engine=xelatex --toc
    if ($LASTEXITCODE -eq 0) { Write-Host "Created $OutPdf" -ForegroundCol
    else { Write-Host "pandoc/xelatex failed (exit $LASTEXITCODE)" -Foregr
}

if ($pdflatex) {
    Write-Host "Building PDF via pdflatex..." -ForegroundColor Green
    & $pandoc $ManualMd -o $OutPdf --pdf-engine=pdflatex --toc
    if ($LASTEXITCODE -eq 0) { Write-Host "Created $OutPdf" -ForegroundCol
    else { Write-Host "pandoc/pdflatex failed (exit $LASTEXITCODE)" -Foreg
}

if ($wkhtmltopdf) {
    Write-Host "Building HTML then converting via wkhtmltopdf..." -Foregro
    & $pandoc $ManualMd -o $OutHtml --standalone --toc
    if ($LASTEXITCODE -ne 0) { Write-Host "pandoc->HTML failed (exit $LAST
    & $wkhtmltopdf $OutHtml $OutPdf
    if ($LASTEXITCODE -eq 0) { Write-Host "Created $OutPdf via wkhtmltopdf
    else { Write-Host "wkhtmltopdf failed (exit $LASTEXITCODE)" -Foregroun
}

# Fallback: produce HTML and inform user
Write-Host "No PDF engine found (xelatex/pdflatex/wkhtmltopdf). Producing
& $pandoc $ManualMd -o $OutHtml --standalone --toc
if ($LASTEXITCODE -eq 0) { Write-Host "Created $OutHtml. Install a TeX eng
else { Write-Host "pandoc->HTML failed (exit $LASTEXITCODE)" -ForegroundCo
```

3. Docker build (one-step reproducible PDF)

Use the provided `tools/Dockerfile` to build the manual inside a container that already includes pandoc + TeX. Example:

```
docker build -t everland-manual -f tools/Dockerfile .
docker create --name tmp everland-manual
docker cp tmp:/work/MANUAL.pdf .
docker rm tmp
```

The Dockerfile uses the `pandoc/latex` base image and runs `pandoc MANUAL.md -o MANUAL.pdf --pdf-engine=xelatex --toc` during build; the generated PDF will be present at `/work/MANUAL.pdf` inside the image.

4. Rebuilding the manual using the wrapper

To rebuild locally (non-Docker):

```
powershell -ExecutionPolicy Bypass -File .\tools\build_manual.ps1
```

If the script reports missing `xelatex`/`pdflatex`/`wkhtmltopdf`, either install one of those or use the Docker method above.