TicTacTwo Terminal User Guide

Project: `tic-tac-two`

February 7, 2026

Contents

1 Overview

TicTacTwo is a terminal-based strategy game inspired by Tic-Tac-Toe, but played on a 5x5 board with a movable active 3x3 grid. You can play human versus AI, AI versus AI, and switch control between human and AI during a match.

1.1 Core Features

- Three startup modes: play as X, play as O, or watch self-play (computer vs computer).
- Legal move types: `place`, `move`, and `shift`.
- AI powered by minimax with alpha-beta pruning.
- Real-time engine analysis panel with principal variations.
- Scrollable move history with keyboard navigation.
- Pluggable evaluation functions (default and positional included).
- Optional strict repetition rule for no-repeat states.
- Interactive command system during gameplay (`ai`, `restart`, `quit`, and more).

2 Install and Run

2.1 Requirements

- Node.js 20+
- npm

2.2 Install and start

1. Install dependencies:

   ```
   npm install
   ```

2. Start the TUI:

   ```
   npm start
   ```

2.3 Useful launch options

`-engine-depth=<n>`	Search depth for AI move selection (default: 6).
`-multi-pv=<n>`	Number of PV lines shown in analysis panel (default: 3).
`-eval=<name>`	Select evaluation plugin (for example: `default` or `positional`).
`-list-evals`	Print available evaluation plugins and exit.
`-self-play`	Start directly in AI vs AI mode.

`-repetition-rule=strict`	Disallow repetition by a matching board plus active-grid state.

3 Rules and Game Model

3.1 Board and objective

- The full board is 5x5.
- Only the active 3x3 window is playable for placement and movement.
- Win by forming a three-in-a-row for your symbol in the current position.

3.2 Move types

- **Place**: put your marker on an empty cell inside the active grid.
- **Move**: reposition one of your existing markers to another empty cell in the active grid.
- **Shift**: move the active 3x3 grid one step in any of the eight directions (up, down, left, right, or diagonally) when legal.

3.3 Piece limits and movement requirements

- Each player has exactly four markers total.
- After placing all four pieces, you must move an existing marker rather than placing new ones.
- Shifting or moving pieces only becomes legal after placing at least two markers.

3.4 Repetition handling

- Default mode avoids immediate move-level backtracking loops.
- `-repetition-rule=strict` enforces no repeated full game state, including grid location.

4 Walkthrough 1: Start a Match and Make a Move

When the app starts, choose your mode:

- `X`: you move first.
- `O`: AI moves first, then you respond.
- `C`: self-play mode.

After mode selection, the screen shows:

- Current board and highlighted active grid.
- Engine panel with depth, node count, and top PV lines.
- Move selector with legal moves and a filter prompt.

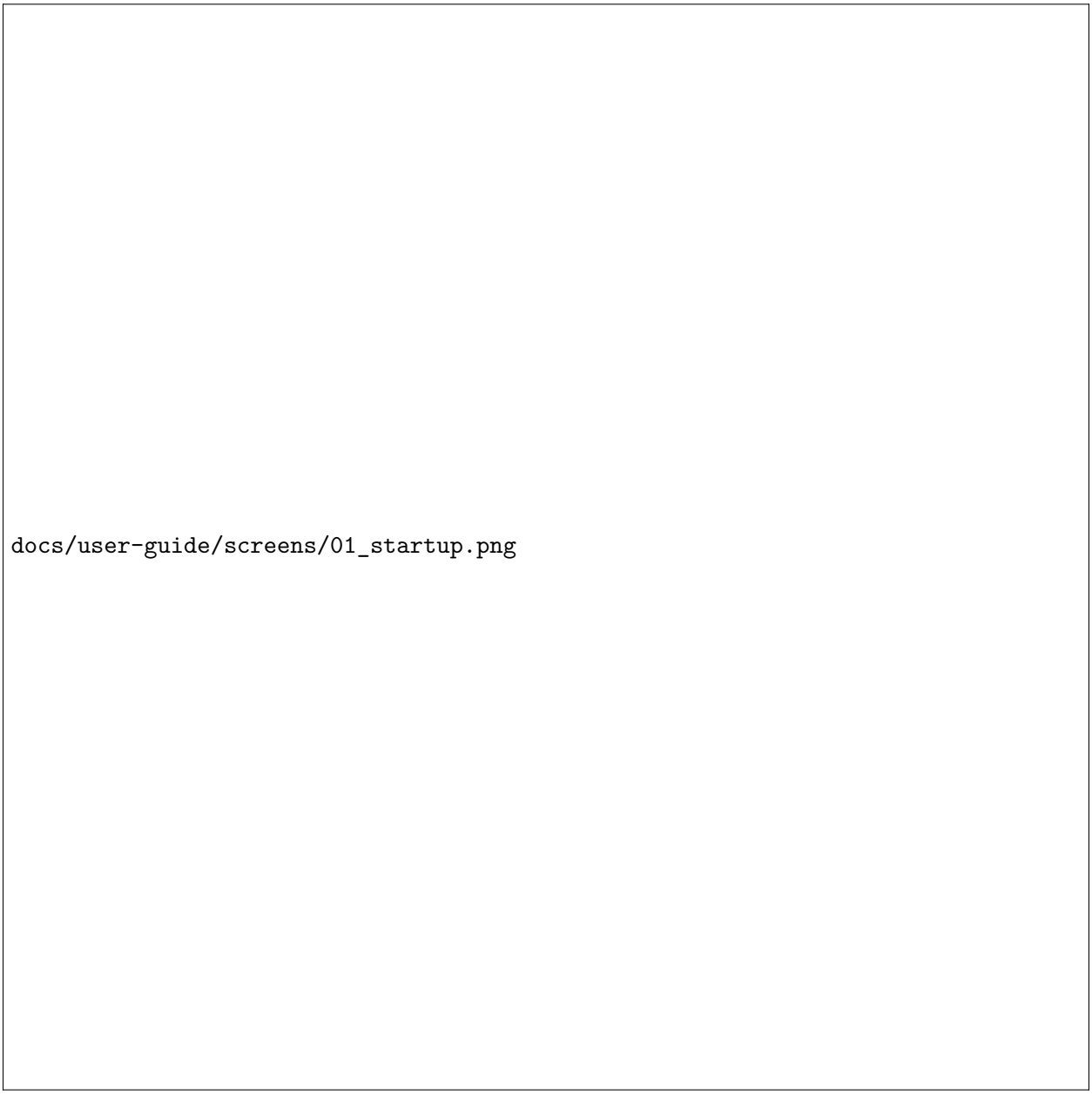

Figure 1: Startup prompt, board rendering, engine panel, and move selector.

5 Walkthrough 2: Use Keyboard Controls During Play

5.1 Move selector controls

- `Tab`, `Up`, `Down`: cycle through legal moves.
- Type any text: filter candidate move list.
- `Enter`: confirm the currently selected move.

5.2 History controls

- `PgUp` and `PgDn`: scroll move history.
- The history panel updates after each move and shows the latest sequence.

5.3 Command shortcuts

- `ai` or `auto`: let AI take the current turn.
- `restart` or `r`: restart the match.
- `exit`, `quit`, or `q`: leave the app.

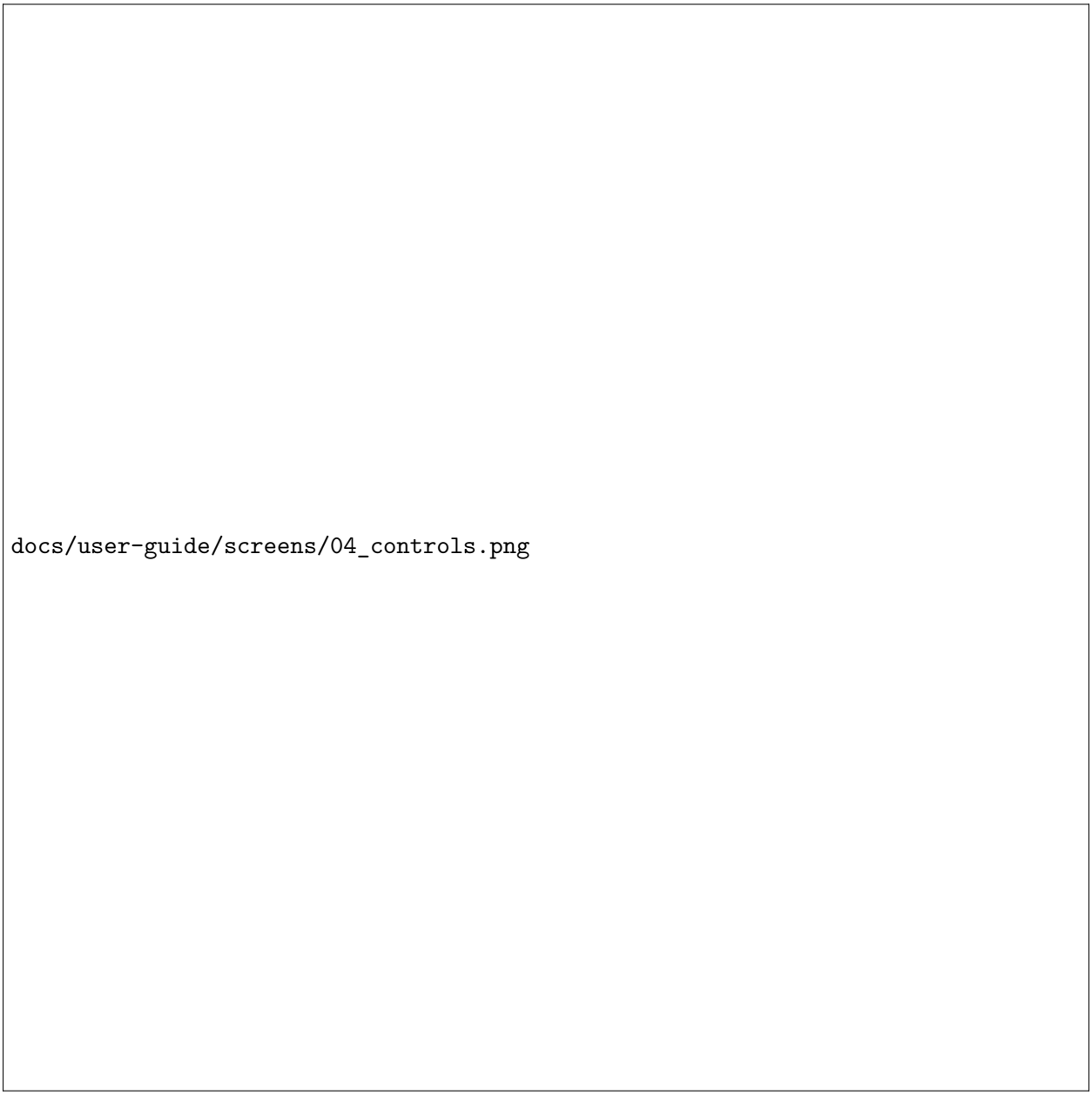

Figure 2: Interactive controls and command shortcuts available in the TUI.

6 Walkthrough 3: Observe Self-Play and Grid Shift

Launch self-play directly:

```
npm run play -- --self-play --engine-depth=2 --multi-pv=2
```

In this mode, the AI alternates moves automatically. This is useful for:

- Understanding strategy and shift timing.
- Watching how the active grid changes move legality.
- Comparing engine lines in real time.

The capture below shows a sequence where O executes `shift left`, which relocates the active grid and changes future legal moves.

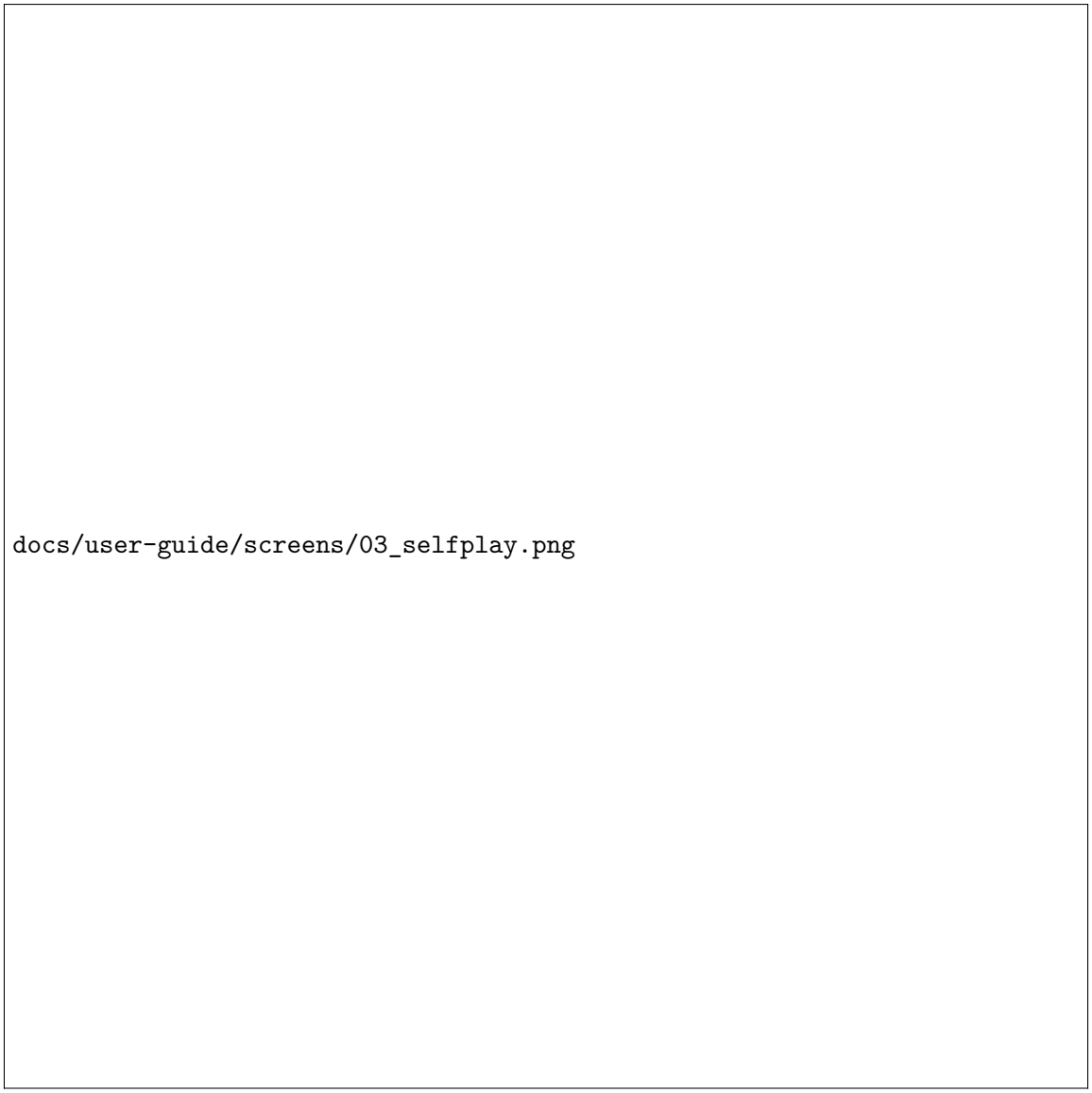

Figure 3: Self-play with engine analysis, history, and an active-grid shift event.

7 Walkthrough 4: Discover and Select Evaluation Plugins

List available evaluation plugins:

```
npm run play -- --list-evals
```

Current built-in plugins:

- `default`: terminal-only scoring (wins, losses, draws).
- `positional`: adds heuristics for threats, center influence, and active-grid presence.

Start with a specific plugin:

```
npm run play -- --eval=positional --engine-depth=7 --multi-pv=4
```

Figure 4: Evaluation plugin listing output.

8 Feature-by-Feature Reference

8.1 Gameplay and interaction

- Human vs AI and AI vs AI modes.
- Type-ahead filtering for move commands.
- Immediate command entry without leaving the move prompt.
- Restart flow that preserves runtime settings.

8.2 Engine and analysis

- Minimax plus alpha-beta pruning.
- Configurable search depth and Multi-PV count.
- Per-turn analysis details: nodes, TT hits, cutoffs.
- Principal variation preview for top candidate lines.

8.3 Extensibility

- Plugin registry for custom evaluation strategies.
- CLI selection by plugin id (`-eval=<id>`).
- Discovery endpoint via `-list-evals`.

9 Development and Testing

9.1 Build and test commands

```
npm run build
npm test
npm run coverage
```

9.2 Quality notes

- The project includes unit tests for core game logic, AI behavior, CLI parsing, and TUI integration paths.
- Coverage reports are generated under `coverage/`.

10 Troubleshooting

- If terminal rendering looks off, resize the terminal and relaunch.
- If keys are not recognized, confirm focus is in the terminal window.
- For strict anti-loop behavior, add `-repetition-rule=strict`.
- Use lower depth values if AI turns feel slow on your machine.

11 Quick Command Cheat Sheet

```
npm run play
npm run play -- --self-play
npm run play -- --engine-depth=8 --multi-pv=4
npm run play -- --eval=positional
npm run play -- --list-evals
npm run play -- --repetition-rule=strict
```