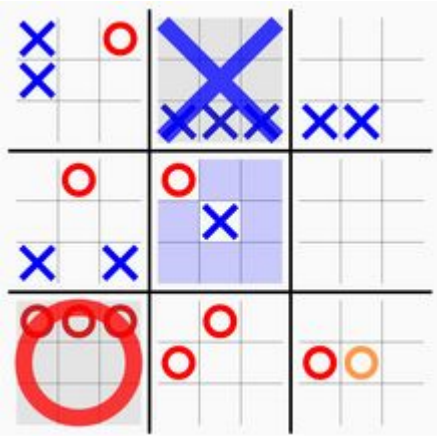
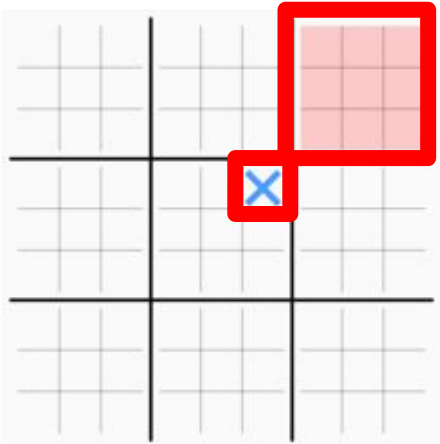
Ultimate Tic-Tac-Toe

https://github.com/jcbhmr/ultttt

Jacob Hummer

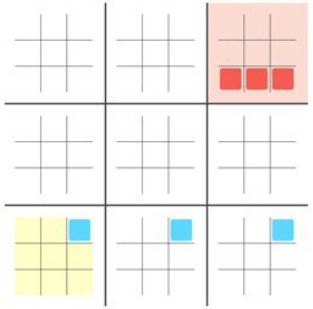




https://bejofo.net/ttt

Ultimate Tic Tac Toe

A stategic boardgame for 2 players.



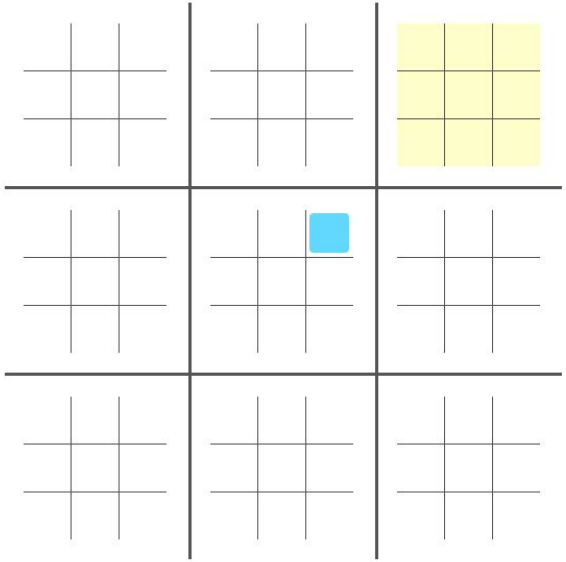
Win three games of Tic Tac Toe in a row. You may only play in the big field that corresponds to the last small field your opponent played. When your are sent to a field that is already decided, you can choose freely.

You can find a better write up of the rules here.

It should work on most devices.
Javascript and cookies have to be enabled.

Start a new game

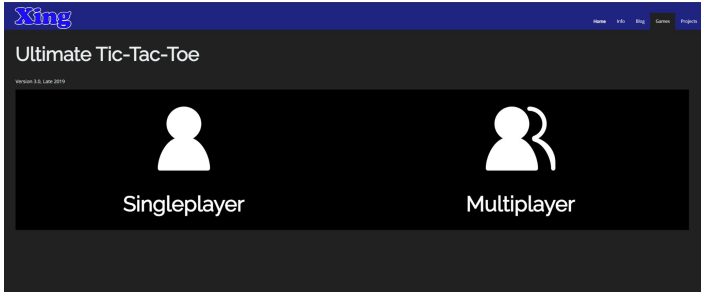
Ultimate Tic Tac Toe



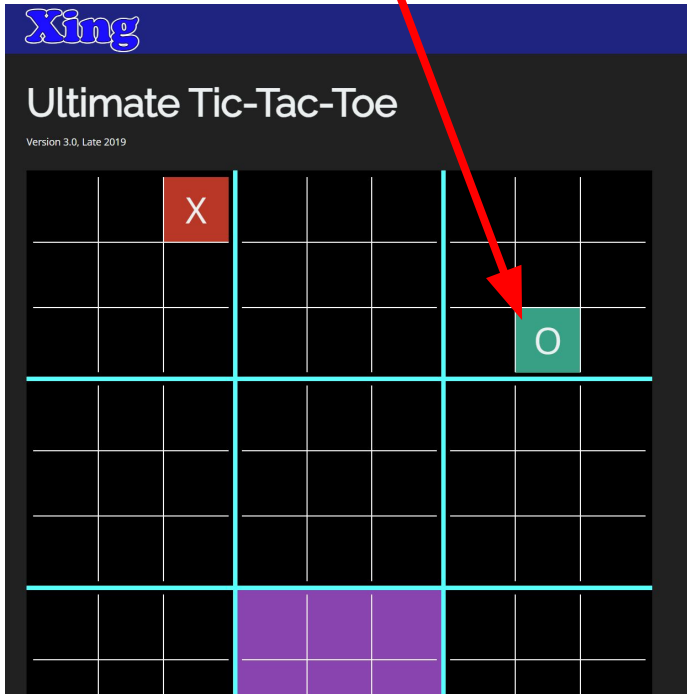
Move-Nr: 1 Waiting for opponent...

Your opponent has not yet opened the game.
Send him/her the URL to this page.

https://michaelxing.com/UltimateTTT/v3/



AI opponent

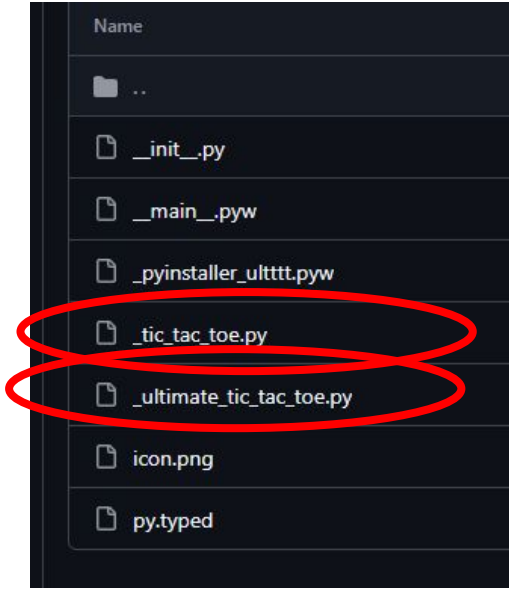


1. **The game is played on a 3 x 3 grid of smaller tic-tac-toe boards.** This creates a total of 81 squares to play in.
2. **Players take turns, starting with `X`.**
3. **On your turn, place your mark (`X` or `O`) in any empty square of the small board you're allowed to play in.** The starting player can choose any square on any small board.
4. **Your move determines where your opponent will play next.** The square you choose within a small board corresponds to the location of the next small board in the larger grid where your opponent must play. For example:
   - If you play in the top-left square of a small board, your opponent must play in the top-left small board of the larger grid.
5. **If your opponent is sent to a full or won small board, they can play in any other board.**
6. **To win a small board, get three of your marks in a row (up, down, across, or diagonally) on that board.** Once a small board is won, it is marked with a large `X` or `O`, and no further moves can be made in that board.
7. **The goal is to win the larger board by winning three small boards in a row.** This can be achieved vertically, horizontally, or diagonally.
8. **The game ends when one player wins the larger board or when no legal moves remain.** If no player wins the larger board and all possible moves are used up, the game is a draw.

DEMO

https://github.com/jcbhmr/ultttt?tab=readme-ov-file#installation

5. **If your opponent is sent to a full or won small board, they can play in any other board.**


Name
..
__init__.py
__main__.pyw
_pyinstaller_ultttt.pyw
_tic_tac_toe.py
_ultimate_tic_tac_toe.py
icon.png
py.typed

```python
import typing

class TicTacToe:
    """
    1.  The game is played on a grid that's 3 squares by 3 squares.
    2.  You are `X`, your friend (or the computer in this case) is `O`. Players take turns putting their marks in empty squares.
    3.  The first player to get 3 of their marks in a row (up, down, across, or diagonally) is the winner.
    4.  When all 9 squares are full, the game is over. If no player has 3 marks in a row, the game ends in a tie.
    """

    grid: list[list[typing.Literal["X"] | typing.Literal["O"] | None]]
    turn: typing.Literal["X"] | typing.Literal["O"]

    def __init__(self) -> None:
        self.grid = [[None for _ in range(3)] for _ in range(3)]
        self.turn = "X"

    def can_play(self, space: tuple[int, int]) -> bool:
        return not self.winner and not self.grid[space[0]][space[1]]

    def play(self, space: tuple[int, int]) -> None:
        if not self.can_play(space):
            raise ValueError("Invalid move")
        self.grid[space[0]][space[1]] = self.turn
        self.turn = "X" if self.turn == "O" else "O"

    @property
    def winner(self) -> typing.Literal["X"] | typing.Literal["O"] | None:
        for i in range(3):
            if self.grid[i][0] == self.grid[i][1] == self.grid[i][2] is not None:
                return self.grid[i][0]
            if self.grid[0][i] == self.grid[1][i] == self.grid[2][i] is not None:
                return self.grid[0][i]
        if self.grid[0][0] == self.grid[1][1] == self.grid[2][2] is not None:
            return self.grid[0][0]
        if self.grid[0][2] == self.grid[1][1] == self.grid[2][0] is not None:
            return self.grid[0][2]
        return None
```

```python
import typing
from . import _tic_tac_toe

class UltimateTicTacToe:
    """
    1. **The game is played on a 3 x 3 grid of smaller tic-tac-toe boards.** This creates a total of 81 squares to play in.
    2. **Players take turns, starting with `X`.**
    3. **On your turn, place your mark (`X` or `O`) in any empty square of the small board you're allowed to play in.** The starting player can choose any square on any small board.
    4. **Your move determines where your opponent will play next.** The square you choose within a small board corresponds to the location of the next small board in the larger grid where your opponent must play. For example:
    - If you play in the top-left square of a small board, your opponent must play in the top-left small board of the larger grid.
    5. **If your opponent is sent to a full or won small board, they can play in any other board.**
    6. **To win a small board, get three of your marks in a row (up, down, across, or diagonally) on that board.** Once a small board is won, it is marked with a large `X` or `O`, and no further moves can be made in that board.
    7. **The goal is to win the larger board by winning three small boards in a row.** This can be achieved vertically, horizontally, or diagonally.
    8. **The game ends when one player wins the larger board or when no legal moves remain.** If no player wins the larger board and all possible moves are used up, the game is a draw.
    """

    grid: list[list[_tic_tac_toe.TicTacToe]]
    turn: typing.Literal["X"] | typing.Literal["O"]
    next_small_board: _tic_tac_toe.TicTacToe | None

    def __init__(self) -> None:
        self.grid = [[_tic_tac_toe.TicTacToe() for _ in range(3)] for _ in range(3)]
        self.turn = "X"
        self.next_small_board = None

    def can_play(self, space1: tuple[int, int], space2: tuple[int, int]) -> bool:
        if self.winner:
            return False
        if self.next_small_board and self.grid[space1[0]][space1[1]] != self.next_small_board:
            return False
        return self.grid[space1[0]][space1[1]].can_play(space2)

    def play(self, space1: tuple[int, int], space2: tuple[int, int]) -> None:
        if not self.can_play(space1, space2):
            raise ValueError("Invalid move")
        self.grid[space1[0]][space1[1]].play(space2)
        for i in self.grid:
            for j in i:
                j.turn = "X" if self.turn == "O" else "O"
        self.turn = "X" if self.turn == "O" else "O"
        self.next_small_board = self.grid[space2[0]][space2[1]] if not self.grid[space2[0]][space2[1]].winner else None

    @property
    def winner(self) -> typing.Literal["X"] | typing.Literal["O"] | None:
        for i in range(3):
            if self.grid[i][0].winner == self.grid[i][1].winner == self.grid[i][2].winner is not None:
                return self.grid[i][0].winner
            if self.grid[0][i].winner == self.grid[1][i].winner == self.grid[2][i].winner is not None:
                return self.grid[0][i].winner
        if self.grid[0][0].winner == self.grid[1][1].winner == self.grid[2][2].winner is not None:
            return self.grid[0][0].winner
        if self.grid[0][2].winner == self.grid[1][1].winner == self.grid[2][0].winner is not None:
            return self.grid[0][2].winner
        return None
```

Draw loop

Setup first

```python
pygame.init()
screen = pygame.display.set_mode((600, 600))
icon = pygame.image.load(pathlib.Path(__file__).parent / "icon.png")
pygame.display.set_icon(icon)
pygame.display.set_caption("ultttt")
clock = pygame.time.Clock()
dt = 0
game = _ultimate_tic_tac_toe.UltimateTicTacToe()
```

```python
while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
    mouse_pos = pygame.mouse.get_pos()
    mouse_pressed = pygame.mouse.get_pressed()

    screen.fill("black")
```

```python
pygame.display.flip()
dt = clock.tick(60) / 1000
```

```python
screen.fill("black")
# The big board is 600x600.
# The big board is 3x3 200x200 squares.
for i in range(1, 3):
    pygame.draw.line(screen, "gray", (i * 200, 0), (i * 200, 600), 4)
for j in range(1, 3):
    pygame.draw.line(screen, "gray", (0, j * 200), (600, j * 200), 4)
```

```python
# The small boards are 150x150 inside a 200x200 square.
# The small boards are 3x3 50x50 squares.
for i in range(0, 3):
    for j in range(0, 3):
        for k in range(1, 3):
            pygame.draw.line(screen, "white", (i * 200 + 25 + k * 50, j * 200 + 25), (i * 200 + 25 + k * 50, j * 200 + 175), 2)
        for l in range(1, 3):
            pygame.draw.line(screen, "white", (i * 200 + 25, j * 200 + 25 + l * 50), (i * 200 + 175, j * 200 + 25 + l * 50), 2)
```

```python
for i, iv in enumerate(game.grid):
    for j, jv in enumerate(iv):
        for k, kv in enumerate(jv.grid):
            for l, lv in enumerate(kv):
                if lv:
                    x = i * 200 + 25 + k * 50
                    y = j * 200 + 25 + l * 50
                    if lv == "X":
                        pygame.draw.line(screen, "red", (x, y), (x + 50, y + 50), 4)
                        pygame.draw.line(screen, "red", (x, y + 50), (x + 50, y), 4)
                    else:
                        pygame.draw.circle(screen, "blue", (x + 25, y + 25), 25, 4)
```

```python
for i, iv in enumerate(game.grid):
    for j, jv in enumerate(iv):
        if jv.winner:
            x = i * 200
            y = j * 200
            if jv.winner == "X":
                pygame.draw.line(screen, "red", (x, y), (x + 200, y + 200), 8)
                pygame.draw.line(screen, "red", (x, y + 200), (x + 200, y), 8)
            else:
                pygame.draw.circle(screen, "blue", (x + 100, y + 100), 100, 8)
```

https://docs.astral.sh/uv/

uv

Search

uv 0.5.6 28.6k 809

uv

An extremely fast Python package and project manager, written in Rust.

Installing Trio's dependencies with a warm cache.

Highlights

- 🚀 A single tool to replace `pip`, `pip-tools`, `pipx`, `poetry`, `pyenv`, `twine`, `virtualenv`, and more.
- ⚡ 10-100x faster than `pip`.
- 🐍 Installs and manages Python versions.
- 🛠 Runs and installs Python applications.
- 📜 Runs scripts, with support for inline dependency metadata.
- 📁 Provides comprehensive project management, with a universal lockfile.
- 🔩 Includes a pip-compatible interface for a performance boost with a familiar CLI.
- 🗂 Supports Cargo-style workspaces for scalable projects.
- 💾 Disk-space efficient, with a global cache for dependency deduplication.
- ⬇ Installable without Rust or Python via `curl` or `pip`.
- 🖥 Supports macOS, Linux, and Windows.

uv is backed by Astral, the creators of Ruff.

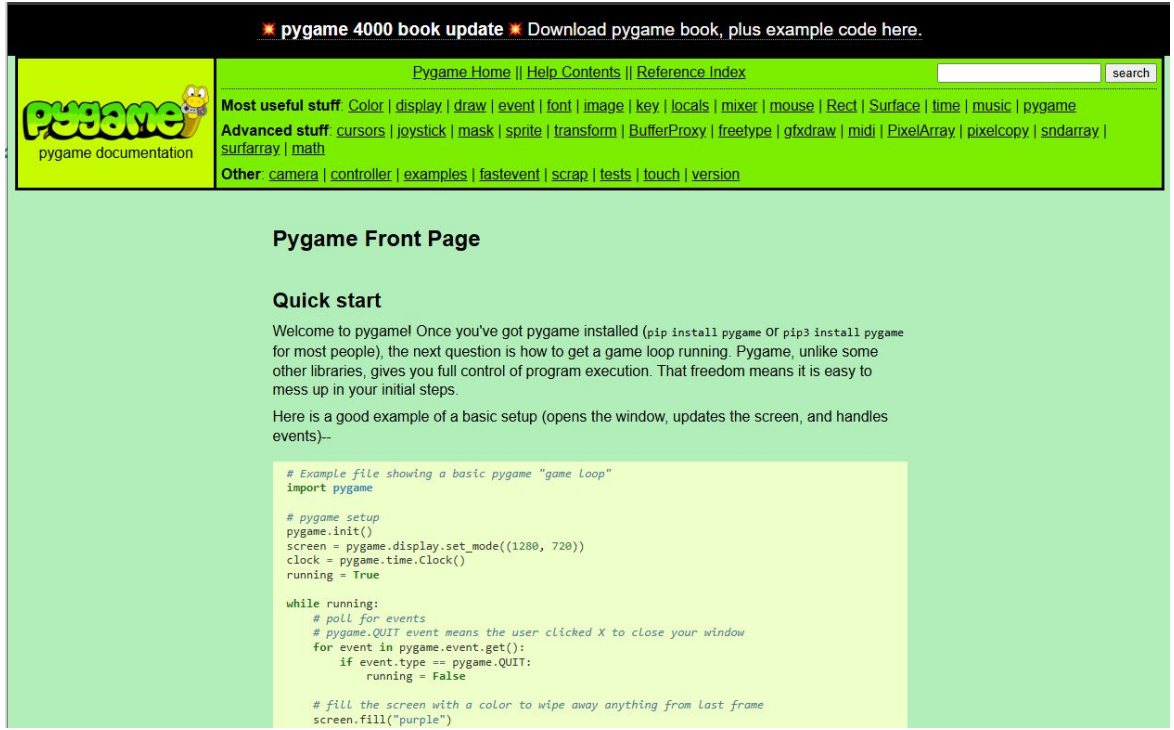
Getting started

Install uv with our official standalone installer:

macOS and Linux | Windows

```
$ powershell -c "irm https://astral.sh/uv/install.ps1 | iex"
```

https://www.pygame.org/docs/



https://pyinstaller.org/en/stable/

PyInstaller

6.11.1

Search docs

/ PyInstaller Manual

View page source

PyInstaller Manual

Version:	PyInstaller 6.11.1
Homepage:	https://pyinstaller.org/
Contact:	pyinstaller@googlegroups.com
Authors:	David Cortesi, based on structure by Giovanni Bajo & William Caban, based on Gordon McMillan's manual
Copyright:	This document has been placed in the public domain.

PyInstaller bundles a Python application and all its dependencies into a single package. The user can run the packaged app without installing a Python interpreter or any modules. PyInstaller supports Python 3.8 and newer, and correctly bundles many major Python packages such as numpy, matplotlib, PyQt, wxPython, and others.

PyInstaller is tested against Windows, MacOS X, and Linux. However, it is not a cross-compiler; to make a Windows app you run PyInstaller on Windows, and to make a Linux app you run it on Linux, etc. x PyInstaller has been used successfully with AIX, Solaris, FreeBSD and OpenBSD but testing against them is not part of our continuous integration tests, and the development team offers no guarantee (all code for these platforms comes from external contributions) that PyInstaller will work on these platforms or that they will continue to be supported.

Quickstart

Make sure you have the Requirements installed, and then install PyInstaller from PyPI:

```
pip install -U pyinstaller
```

Open a command prompt/shell window, and navigate to the directory where your *.py* file is located, then build your app with the following command:

```
pyinstaller your_program.py
```

Your bundled application should now be available in the *dist* folder.

```
build-exe = "pyinstaller -y --name=ultttt --collect-all=ultttt --icon=src/ultttt/icon.png --windowed src/ultttt/_pyinstaller_ultttt.pyw"
```

```yaml
pyinstaller:
  needs: version
  strategy:
    fail-fast: false
    matrix:
      include:
        # https://packaging.python.org/en/latest/specifications/platform-compatibility-tags/#platform-tag
        # target=$(python -c 'import sysconfig; print(sysconfig.get_platform().replace("-", "_").replace(".", "_"))')
        - { os: ubuntu-latest, target: linux_x86_64 }
        - { os: macos-latest, target: macosx_11_0_arm64 }
        - { os: macos-13, target: macosx_10_9_x86_64 }
        - { os: windows-latest, target: win_amd64 }
  defaults:
    run:
      shell: bash
  runs-on: ${{ matrix.os }}
  steps:
    - uses: actions/checkout@v4
      with:
        submodules: recursive
    - name: Install the latest version of uv
      uses: astral-sh/setup-uv@v3
      with:
        version: "latest"
    - run: uv run poe build-exe
    - env:
        version: ${{ needs.version.outputs.version }}
        target: ${{ matrix.target }}
      run: |
        mkdir stage
        mv dist/ultttt stage/ultttt-$target-$version
    - uses: actions/upload-artifact@v4
      with:
        name: ultttt-${{ matrix.target }}
        path: stage
```

v1.0.0 Latest

github-actions released this last week

Full Changelog: `v0.1.0...v1.0.0`

▾ Assets 6

- ultttt-linux_x86_64-1.0.0.tar.gz
- ultttt-macosx_10_9_x86_64-1.0.0.tar.gz
- ultttt-macosx_11_0_arm64-1.0.0.tar.gz
- ultttt-win_amd64-1.0.0.zip
- Source code (zip)
- Source code (tar.gz)

```python
if game.winner:
    font = pygame.font.Font(None, 100)
    text = font.render(f"{game.winner} wins!", True, "purple")
    text_rect = text.get_rect(center=(300, 300))
    screen.blit(text, text_rect)
else:
    # The next small board is outlined in green.
    if game.next_small_board:
        for i, iv in enumerate(game.grid):
            for j, jv in enumerate(iv):
                if jv == game.next_small_board:
                    pygame.draw.rect(screen, "green", (i * 200, j * 200, 200, 200), 4)
    # If one of the 89 small squares is hovered over it is outlined in yellow.
    for i, iv in enumerate(game.grid):
        for j, jv in enumerate(iv):
            for k, kv in enumerate(jv.grid):
                for l, lv in enumerate(kv):
                    if game.can_play((i, j), (k, l)):
                        r = pygame.Rect(i * 200 + 25 + k * 50, j * 200 + 25 + l * 50, 50, 50)
                        if r.collidepoint(mouse_pos):
                            pygame.draw.rect(screen, "yellow", (i * 200 + 25 + k * 50, j * 200 + 25 + l * 50, 50, 50), 4)
                            if mouse_pressed[0]:
                                    game.play((i, j), (k, l))
```

Installation

GITHUB PYPI

The best way to install this application is to download the platform-specific precompiled binary from the latest release.

Windows x86-64
https://github.com/jcbhmr/ultttt/releases/download/v1.0.0/ultttt-win_amd64-1.0.0.zip

macOS x86-64
https://github.com/jcbhmr/ultttt/releases/download/v1.0.0/ultttt-macosx_10_9_x86_64-1.0.0.tar.gz

macOS AArch64
https://github.com/jcbhmr/ultttt/releases/download/v1.0.0/ultttt-macosx_11_0_arm64-1.0.0.tar.gz

Linux x86-64
https://github.com/jcbhmr/ultttt/releases/download/v1.0.0/ultttt-linux_x86_64-1.0.0.tar.gz

This package is also published to PyPI if you prefer to install it from there:

```
uv tool install ultttt
```