**Tic-Tactical-Toe**

May 01, 2021

Corey Anderson Charles “Chip” Beach

G0xxxxxxx G00229087

# **Introduction**

Tic-Tactical-Toe is a simple 1v1 turn-based offense/defense balance game on grid-based board. The game design features extremely simple front-end graphicswith an appropriate amount of game input/control complexity. The players start with control of a single “home” cell on the opposite sides of the board (See Figure 1). The goal of the game is to capture the opponents home cell. The goal is achieved by taking over cells until a cell adjacent to the opponent’s home cell is captured. At this point a user can attempt to take over their opponent’s home cell and thus win the game. The users take turns acquiring “tacs”, moving “tacs” or attempting to take over a cell.

Each player has “tacs” in the cells they “own”. “tacs” represent power units; the more in a cell the stronger is defense and the strong its offense can be. An owned cell can hold between 1 and 9 tacs. If a cell is unoccupied a user can simply take it over, but must add “tacs” to it. If the opponent is occupying the cell to be taken over, there is a conflict. The winner of the conflict determined by the game. Influences to the outcome are the number of “tacs” applied by the user trying to takeover the cell, the number “tacs” available to defend the cell, and a randomly generated weight. The rules of the game are provided in Section 5 of this document.

<fig 1. image of game board>

The game is written in python using the Django web framework using a Postgres DB. Git was used for code configuration management.

# **Design/Architecture**

NOTE: Just threw a couple diagrams in to inspire our thinking about what should go in here.

Diagram

Description automatically generated

**Chart, line chart

Description automatically generated**

# **Installation**

Docker installation?

# **Operating Instructions**

Server:

Client :

# **Game Rules**

1. Player that takes over opponents home cell wins

2. One move per turn

3. Number of "tacs" (1 - 9) per cell is displayed in the celll

4. A player can only takeover from a cell they own (src) to a cell adjacent (dest) to src

5. Move is one of three choices

- Idle, src cell == dest cell — gain 3 tacs in src cell

- Move to build up strength

- Move tacs - x tacs from src cell to adjacent dest cell — gain 2 tacs in src cell

- Move to reposition strength

- Attempt to takeover a dest cell with x tacs from src cell — gain 1 tac in src cell

- Move to expand control

- If dest cell unoccupied, x tacs are moved to dest

- If dest cell is occupied by opponent, results are calculated and displayed on board (see “results” below)

6. A src cell will receive additional tacs per move as noted above, but will not have more than 9.

7. If destination cell is occupied by opponent, opponent must be “beat” in a “conflict” in order to take over the cell.

8. Results from a conflict are calculated by  (“dest tacs”\*1.2/“src tacs moved”) \* RNG(between 1-10) = R.  If R > 5, successful defense, otherwise successful takeover

- Loser loses all allocated tacs, victor loses 20% (result rounded up to next whole digit)

# **Why the Game is Secure**

*From project description:*

* + - *Assurance case should be a convincing argument for why it’s secure. Justify in the assurance case that it is enough, ensuring a fair minded decision maker would find the risk acceptable*
    - *Possibly walk through a list of common vulnerabilities and explain why your program isn’t vulnerable to any of them.*
    - *Describe peer review, static analysis tools, dynamic tools you used to check it,*
    - *Describe how the design counters attack (including privilege limiting mechanisms),*
    - *Describe how the input filters counters attack,*
    - *Describe how the configuration counters attacks*
    - *Describe problems that were found and fixed and if they provide evidence that the analysis was thorough enough to find and fix problems.*

***Important:****Include a justification in your assurance case that what you've done is enough. Do not just include a list of things you did, since that does not justify that you did enough. How can the reader know that you covered all important issues? I’m not looking for a formal mathematical proof that it’s secure; I’m looking for a set of arguments that would convince a fair-minded decision-maker that the risk of vulnerabilities is low enough that it’s ready to deploy because you've clearly addressed everything that is important. You should be able to show a "complete set" and that you "addressed everything in the set". See the material about assurance cases for more.*

Bottom of Form