**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 a grid-based board. The game design features extremely simple front-end graphicswith an appropriate amount of game input/control complexity. The players start owning a single “home” cell on opposite sides of the board (See Figure 1). The goal of the game is to capture the opponents home cell. Players take turns advancing their position by taking over cells and/or gaining strength. The goal of capturing the opponent’s home cell is achieved by taking over cells until a cell adjacent to the opponent’s home cell is captured. At this point the user is in a position to attempt a take over their opponent’s home cell and thus win the game. A more complete description of the game play is provided with the rules in Section 5.

A picture containing text

Description automatically generated

Fig 1. Image of Game Board>

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

Our team is made up of two diverse IT professionals. Cory ….

Chip is a management consultant in the IT field. Originally a programmer (C, UNIX, pre-web) he moved into System Engineering, management and finally consulting. Chip has returned to GMU to pursue a Masters in ISA and outside of some course assigned programming exercises has not been an active programmer for some time. So for him, the challenge on this project has been to learn the tools and technology used (i.e. python, Django, javascript, json, web, etc). xxxxxx

# **Design/Architecture**

Used self certification

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**

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 its defense and the stronger its offense can be. A cell is owned when a player has moved “tacs” in it. An owned cell can hold between 1 and 9 tacs. Only “tacs” of one player can occupy a given cell. 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.

Specific Rules

1. Player that takes over opponent’s home cell wins

2. One move per turn

3. Game start:

Player 1 home cell has 1 tac

Player 2 home cell has 2 tacs

4. Player 1 has the first move.

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

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

7. Move is one of three choices

- **Idle**, src cell == dest cell — gain 3 tacs in src cell, plus all owned cells gain 1 tac

Move to build up strength

- **Move tacs** - x tacs from src cell to adjacent dest cell — plus all other owned cells gain 1 tac

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 — plus all other owned cells gain 1 tac

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

After a conflict if the cell is not won by the attacker, the attacker retains the turn and can try again or make a different move

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

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

10. Results from a conflict are calculated using virtual dice rolls. Following the tried and true method used in Risk, the attacker uses 3 dice or fewer and the defender 2 dice or fewer depending on the number of “tacs” each has, respectively. The dice results are compared from highest to lowest, where the defender always wins the tie.

- Each side loses “tacs” based on which dice comparisons they’ve lost.

- If the attacker succeeds in winning the cell, the conquered cell gets the attack tacs — plus all other owned cells gain 1 tac

# **Game Security**

Tic-Tactical-Toe is a game that can be played either in league or tournament style competition, where winners can accumulate cash and losers, well, can lose their money. Because of that, it is important that the game is secure against players or outside adversaries from intentionally or unintentionally influencing games or otherwise skirting the rules. For example, a user shouldn’t be able to assign themselves cells not properly captured, make moves out of turn, change win/loss records or block other players from competing. Non-authorized users should not have access to account information, audit files, game logs, win/loss records, etc. Therefore, we secured the game against moderate threats, where a possible breach could have a serious adverse impact such as a loss of money or inappropriate awarding of money. For instance, in a tournament if a user could independently update their win/loss record, they could potentially set themselves up to win the money.

We focused on maintaining a high level of fidelity for the games in play. Although we limited what the users could select for input by providing pull down menus

We reviewed OWASP top ten vulnerabilities ( https://owasp.org/www-project-top-ten/ ) to see if any could be potential vulnerabilities for our game. Below is our evaluation.

Injection Flaw (e.g. SQL injection)

Broken Authentication

Sensitive Data Exposure

XML External Entities (XXE)

Broken Access Control

Security Misconfiguration

Cross Site Scripting (XSS)

Insecure Deserialization - use REGEX to validate move data from deserialization of JSON from client

Using Components with Known Vulnerabilities

Insufficient Logging & Monitoring

We used the “safety” static evaluation tool to verify that the libraries we used did not have known vulnerabilities.

Diagram

Description automatically generated

*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