Placement Calculator: In-Depth Documentation

This documentation provides a comprehensive overview of the `placementCalc.py` script. The script processes data regarding rooms and dwellers from the game **Fallout Shelter**, optimally assigning dwellers to rooms, analyzing their stats, and balancing room production times. It also provides visual analytics of the changes before and after balancing.

Purpose and High-Level Overview

The script aims to:

- **Extract and process data** from game files (`Vault2.json`, `vault_map.txt`, and `vault.db`)
- **Assign dwellers** to resource rooms (Power, Water, Food) based on their best, second-best, and worst stats
- **Optimize and balance** room assignments to achieve near-uniform resource production times
- **Visualize** the effects of balancing strategies

The script is highly specialized for **statistical optimization** and **resource management** in the context of Fallout Shelter or similar management games.

Key Dependencies

- **Standard Library**: `os`, `json`, `sqlite3`, `collections`, `queue`, `ast`
- **Third-Party Libraries**: `PIL` (for images, not used in core logic), `numpy`, `matplotlib`

Main Data Inputs

- **`Vault2.json`**: Contains information about all dwellers and their happiness
- **`vault_map.txt`**: Textual map of the vault's room layout
- **`vault.db`**: SQLite database with detailed stats per dweller

Core Data Structures

Variable	Description
`geothermal`	List of all Geothermal room instances
`waterPlant`	List of all Water Plant room instances
`cafeteria`	List of all Cafeteria room instances
`gym`, `armory`, `dojo`	Lists of training rooms (Gym = Strength, Armory = Perception, Dojo = Agility)
`dwellers_list`	All dwellers loaded from JSON
`dweller_stats`	Dict: dweller ID → stat values
`sortedL`	Dict: room key tuple → list of assigned dweller IDs
`Stats`	Dict: dweller ID → list of (statname, value, mode) from DB

File Loading and Parsing

JSON and Vault Map Parsing

The script starts by reading `Vault2.json` and `vault_map.txt` to load dwellers and rooms:

```
downloads_folder = os.path.expanduser(r"~\Downloads")
file_path = os.path.join(downloads_folder, "Vault2.json")
with open(file_path, "r", encoding="utf-8") as file:
    data = json.load(file)
vault_file = "vault_map.txt"
```

Rooms are parsed line by line, extracting their **type**, **level**, and **merge size** (how many rooms are merged into one).

- Each room is given a code (e.g., `Goe-lvl3-size9-1` for a level 3, size 9 Geothermal room).

Room and Training Room Extraction

Rooms are categorized and stored in lists:

- **Resource Rooms:** `geothermal`, `waterPlant`, `cafeteria`
- **Training Rooms:** `gym`, `armory`, `dojo`

Duplicate and merged rooms are filtered, ensuring only unique room instances remain.

Dwellers and Stat Extraction

Dwellers and Stat Assignment

For each dweller, stats (`Strength`, `Perception`, `Agility`) are fetched from the SQLite database, and the script determines:

- **Best stat** (highest value)
- **Second-best stat**
- **Worst stat**

Dwellers are then sorted into the following categories based on their stats:

- **bestGeo/bestWaP/bestCaf**: Dwellers best suited for Geothermal (Strength), Water Plant (Perception), Cafeteria (Agility)
- **secbestGeo/secbestWaP/secbestCaf**: For second-best stat
- **worstGeo/worstWaP/worstCaf**: For worst stat

This process enables optimal assignment by skill.

Room Assignment Logic

Assignment & Balancing Algorithm

Assignment Function

The core function for assignment is:

```python
def assign_rooms(rooms, dwellers):
    for room in rooms:
        room_parts = room.split("-")
        size = room_parts[2]
        size_map = {"size9": 6, "size6": 4, "size3": 2}
        LIMIT = size_map.get(size, 0)
        key = tuple(room_parts)
        current = len(sortedL.get(key, []))
        take = LIMIT - current
        if take > 0:
            assigned = dwellers[:take]
            del dwellers[:take]
            sortedL.setdefault(key, []).extend(assigned)
```

- **Rooms are filled in three rounds**:
    i. With dwellers best suited for the room
    ii. With dwellers whose second-best stat matches the room
    iii. With dwellers whose worst stat matches

Training Assignment

After resource room assignments, leftover dwellers are assigned to training rooms, prioritizing the weakest dwellers (for stat improvement).

Balancing and Auto-Balancing

Production Time Calculation

Each room's **production time** is calculated based on:

- Room base pool (e.g., 1320 for Power)
- Room size multiplier (1x, 2x, 3x)
- Sum of relevant stats for assigned dwellers
- Overall vault happiness

```
def get_room_production_time(room_key, dwellers, dweller_stats, happiness=1.0):
    ...
    total_stat = sum(dweller_stats[d][stat] for d in dwellers)
    ...
    return round(pool / (total_stat * happiness), 1)
```

Mean and Variance Analysis

- The script calculates the mean production time for each room type.
- It then **differentiates rooms above and below the mean**, identifying potential inefficiencies.

Balancing Passes

A multi-pass balancing loop attempts to minimize differences in production times across rooms of the same type:

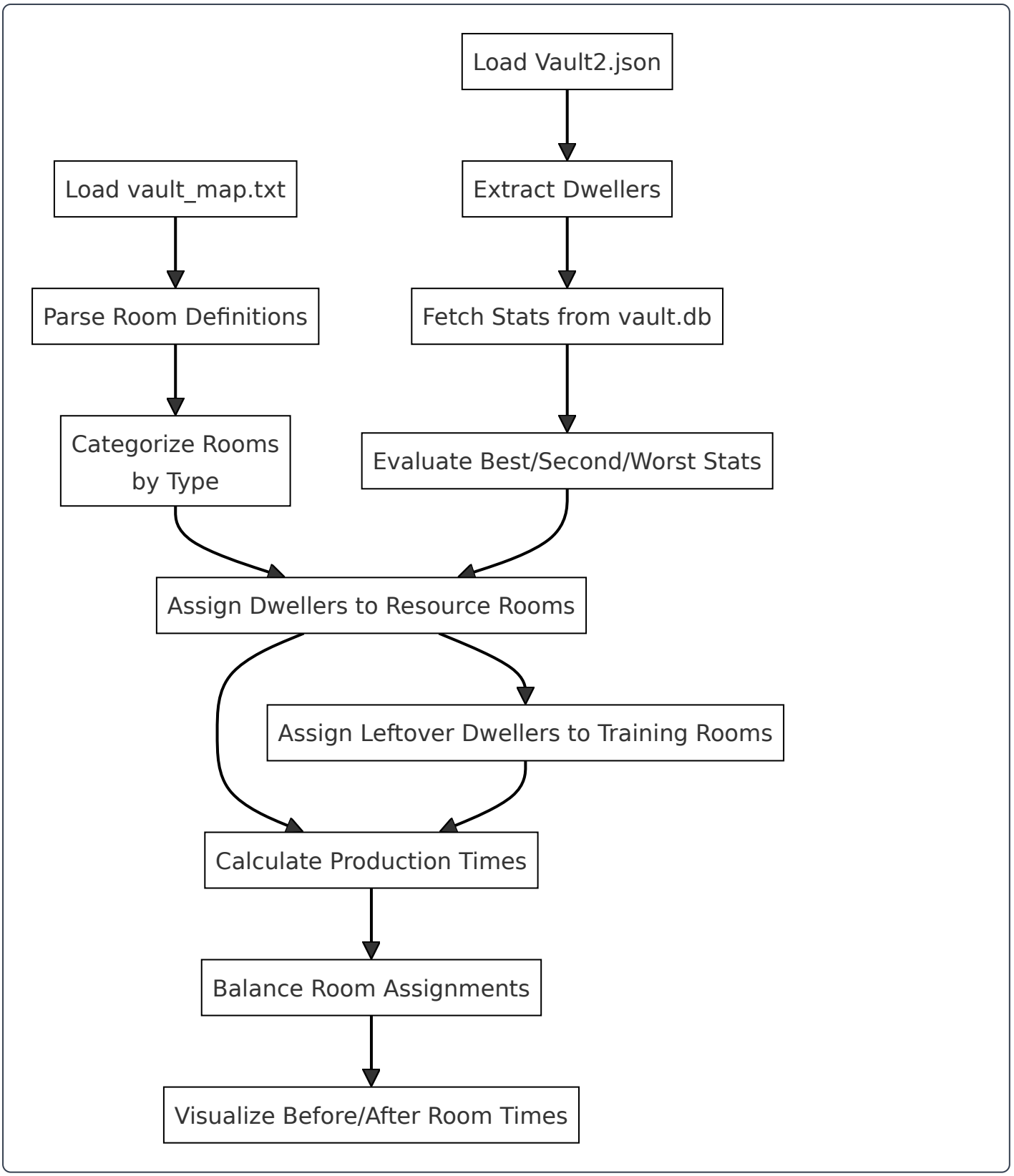
- **Best dwellers from fast rooms** are swapped with **worst dwellers from slow rooms**.
- This continues until no room differs by more than the `BALANCE_THRESHOLD` (5 seconds) from the mean.

Visualization

The script outputs a **bar plot** comparing room production times before and after balancing, using `matplotlib`.

Detailed Data Flow

Below is a visual flowchart of the key data and logic, from file loading to visualization:


Load Vault2.json
Load vault_map.txt
Extract Dwellers
Parse Room Definitions
Fetch Stats from vault.db
Categorize Rooms by Type
Evaluate Best/Second/Worst Stats
Assign Dwellers to Resource Rooms
Assign Leftover Dwellers to Training Rooms
Calculate Production Times
Balance Room Assignments
Visualize Before/After Room Times

Important Functions and Algorithms

Stat Extraction from Database

```
cursor.execute(
    "SELECT dweller_id, StatName, Value, Mod FROM Stats WHERE StatName IN (?, ?, ?) AND dweller_id = ?",
    (stats[0], stats[1], stats[2], serialize_id)
)
allStats = cursor.fetchall()
```

- Fetches `Strength`, `Perception`, and `Agility` for a specific dweller.

Room Assignment and ID Extraction

```
def extract_ids(data):
    return [x.split(" - ")[0].strip() for x in data]
```

- Extracts only the dweller IDs from stat strings.

Multi-Pass Balancing Loop

The balancing loop runs up to `MAX_PASSES` times, swapping or moving dwellers between rooms to even out production times:

```
while pass_num <= MAX_PASSES:
    recalc_times()
    ...
    if is_balanced():
        print(f"\nBalanced after {pass_num} passes")
        break
    ...
    pass_num += 1
```

Output and Visualization

At each major step, the script provides **console printouts**:

- Stats per dweller
- Assignment summaries (room to dwellers)
- Vault average happiness
- Before/After production times
- Balancing actions

A comparative **bar plot** is displayed at the end.

Error Handling

- If required files (`vault_map.txt`) are missing, an error message is printed.
- All database connections are closed at the end.

Design Choices and Considerations

- **Efficient Assignment**: The algorithm prioritizes best-fit dwellers but ensures no dweller is left unassigned unless all rooms are full.
- **Balancing for Uniformity**: Rather than maximizing output in some rooms, the script seeks an even distribution to avoid bottlenecks.
- **Training Utilization**: Leftover dwellers are slated for training to maximize future potential.
- **Visualization**: The bar plot offers a tangible view of the optimization impact.

Example Room Assignment Output

```
Room: ('Goe', 'lvl3', 'size6', '1') -> Dwellers: 101, 102, 103, 104
Room: ('WP', 'lvl2', 'size3', '1') -> Dwellers: 110, 111
...
```

Key Constants

Constant	Value/Description
`BASE_POOL`	Power: 1320, Food: 960, Water: 960
`SIZE_MULTIPLIER`	size3: 1, size6: 2, size9: 3
`ROOM_CAPACITY`	size3: 2, size6: 4, size9: 6
`BALANCE_THRESHOLD`	5 seconds

Best Practices & Takeaways

Balancing Resources
Assign dwellers based on their stats for optimal room efficiency. Periodic balancing ensures uniform resource production.

Data Integrity

Ensure all input files (JSON, TXT, DB) are up-to-date and formatted correctly for accurate calculations.

Limitations & Recommendations

- **Assumes stat relevance**: Only three stats are considered for assignment.
- **Batch balancing**: Moves are made in "passes" rather than continuous optimization.
- **No API/interactive endpoints**: This is a script, not a running service.

Summary

The `placementCalc.py` script is a powerful analytic and optimization tool for managing Fallout Shelter rooms and dwellers. By leveraging structured data extraction, stat-based assignment, iterative balancing, and clear visualization, it can significantly improve vault efficiency and resource production balance.

For further enhancements: Consider modularizing the code, adding error reporting/logging, or providing a simple CLI for parameter customization.