Project Proposal: Tactical Hero Battle Game

1. Project Overview

This project aims to develop a 2D side-scrolling strategy game inspired by **Line Ranger** and **Battle Cats**. Players will deploy units with unique abilities onto the battlefield, strategizing to destroy the enemy's base while defending their own. The game will feature real-time unit management, resource allocation, and a variety of characters with different strengths and weaknesses.

2. Project Review

The game improves upon classic tower defense games by implementing:

Enhanced Unit System: 18 hero classes and ? enemy types with distinct abilities

Dynamic Resource System: Three resource types that accumulate at different rates based on time of day

Detailed Analytics: Tracks and visualizes battle performance

Progressive Unlock System: New heroes and levels unlock as players progress

3. Programming Development

3.1 Game Concept

- **Objective:** Destroy the opponent’s base while defending your own.
- **Mechanics:**
    - Deploy heroes using accumulated resources
    - Each hero has unique costs and combat abilities
    - Enemies spawn in waves with increasing difficulty
    - Complete levels to unlock new heroes and challenges

3.2 Object-Oriented Programming Implementation

The game will include at least the following five classes:

1. **AssetLoader (Singleton)**
    - Manages all game assets (hero sprites, enemy sprites, projectiles)

    - Ensures assets are only loaded once
2. **Tower (Base.py)**
    - Represents player and enemy bases
    - Tracks health and available units
    - Handles damage calculations
3. **Unit (Unit.py) - Base class for all combat units**
    - Attributes: health, attack_power, speed, cost
    - Methods: move(), attack(), update()
    - Subclasses for each hero/enemy type (LumberJack, Pantheon, etc.)
4. **EnemyLogic (Enemy.py)**
    - Attributes: health, attack_power, speed, cost
    - Methods: move(), attack(), update()
    - Subclasses for each hero/enemy type (LumberJack, Pantheon, etc.)
5. **Player (Player.py)**
    - Controller: Handles player input and unit deployment
    - Resources: Manages resource accumulation and spending
6. **GameStats (GameStats.py)**
    - Tracks battle statistics
    - Records to CSV for analysis
    - Generates post-battle reports
7. **LevelSelect/CharacterSelect (Level_select.py)**
    - Handles level progression
    - Manages hero selection interface
    - Controls unlock system

3.3 Algorithms Involved

These are the note worthy algorithms currently involved

- **Unit Deployment Algorithm:**
    - Checks resource requirements before spawning
    - Manages cooldowns between deployments
    - Prioritizes available units based on selection
- **Enemy Spawn Algorithm**
    - Spawns units based on level
      Example from the test level

```
def spawn_pattern(self, tower, player_resources):
    if self.enemy_spawn_timer > 180:
        mobs = [Centipede, BigBloated]
        EnemyLogic.spawn_unit(random.choice(mobs), tower)
        self.enemy_spawn_timer = random.randint(0, 30)
```

- **Combat Calculation Algorithm**
    - Calculates damage based on unit stats

    - Handles collision detection
    - Manages unit death and removal
    - Locates the closest unit first

- **Resource Management Algorithm**
    - Deducts based on user inputs
    - Adds resources based on user inputs and time

4. Statistical Data (Prop Stats)

4.1 Data Features

We will track at least five key metrics in the game, including:

1. **Units Deployed:** Number of units placed per battle.
2. **Damage Dealt:** Total damage each unit type inflicts.
3. **Battle Duration:** Time taken to complete a level.
4. **Player Resources Usage:** Resource spending efficiency.
5. **Win/Loss Ratio:** Number and percentage of victories per player.

4.2 Data Recording Method

- Data will be stored in a **CSV file** for easy analysis.

4.3 Data Analysis Report

- **Statistical Measures:** Mean, median, variance, and trend analysis.
- **Presentation:** Data will be visualized using graphs, tables, and charts

Visual Example:
Units Deployed

- Graph/Chart: Bar Chart
    - Purpose: Reveal player strategy preferences
    - Data: Number of units deployed for each battle or unit type.
    - Use Case: Compare frequency of different unit deployments

2. Damage Dealt

- Graph/Chart: Stacked Bar Chart or Line Chart
    - Purpose: Track the total damage dealt by players throughout a level.
    - Data: The damage dealt by each unit

    - Use Case: Identify most effective combat units

3. Battle Duration

- Graph/Chart: Text/Histogram
    - Purpose: Highlight difficulty spikes
    - Data: Time duration for each battle or level.
    - Use Case: Find levels needing balancing

4. Player Resources Usage

- Graph/Chart: Pie Chart
    - Purpose: Display resource allocation
    - Data: Resources used for deploying different units.
    - Use Case: Optimize spending strategies

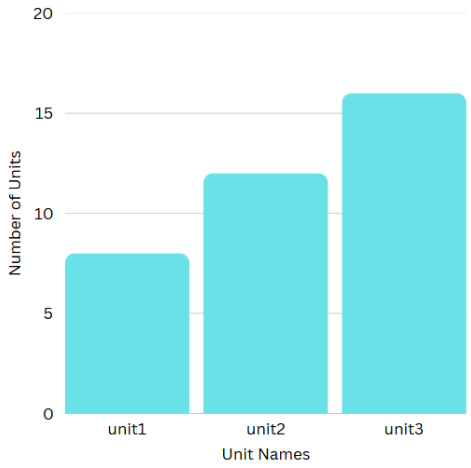
5. Win/Loss Ratio

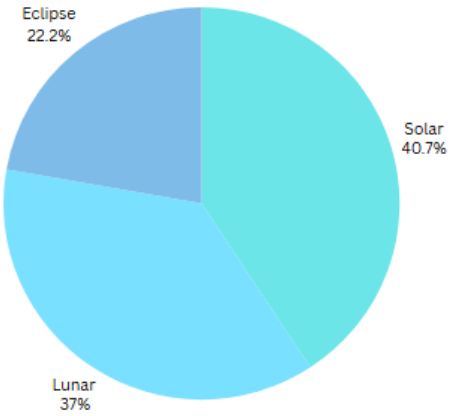
- Graph/Chart: Percentage Gauge
    - Purpose: Measure level difficulty
    - Data: Wins and Losses
    - Use Case: Adjust game balance

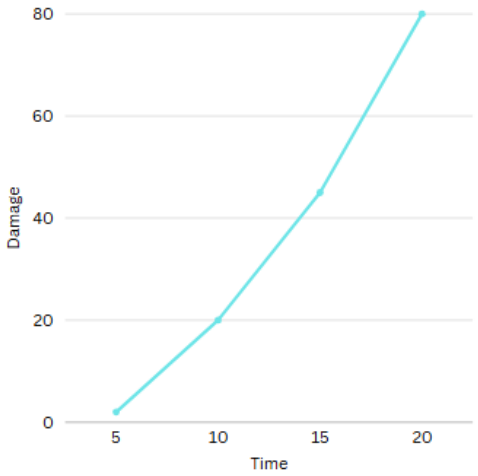
	Why it is a good to have this data what can i use it for	How will you obtain 50 values of this feature data	Which variable Will i collect it from	How will i Display The summarized statistics
Win/Loss Ratio	Tracks player success rate	Recorded when tower is destroyed	GameStats.battle_outcome	• Percentage Chart
Units Deployed	Shows deployment frequency	Counted per spawn command	GameStats.units_deployed	• Bar Chart
Damage Dealt	Measures unit effectiveness	Summed per attack instance Unit.attack_power	Unit.attack_power	• Stacked Bar Chart or Line Chart
Battle Duration	Reveals level difficulty	Timed from start to finish	GameStats.duration	• Text/Histogram

Player Resources Usage	Analyzes spending efficiency	Tracked per transaction	Resources.solar/lunar/eclipse_energy	• Pie Chart

Example of Graphs



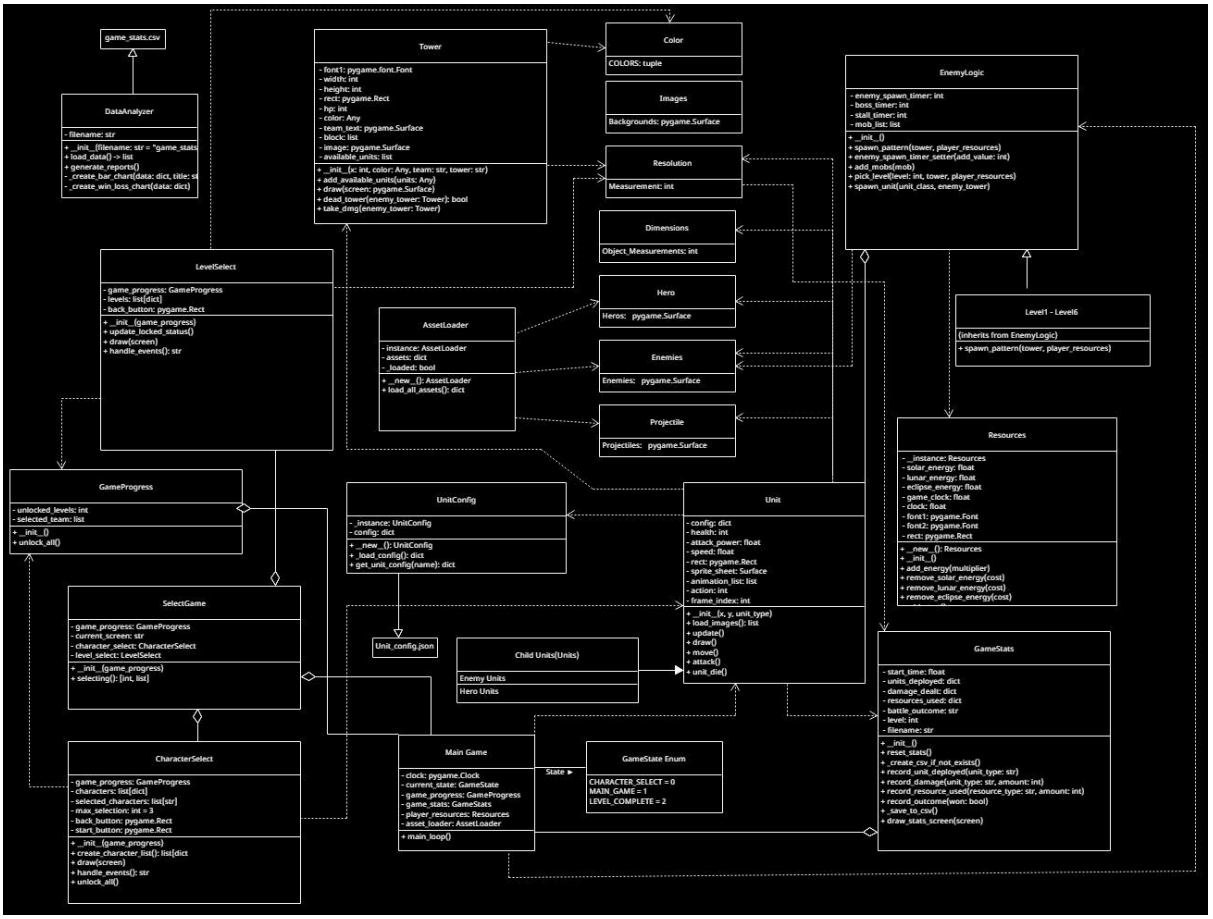




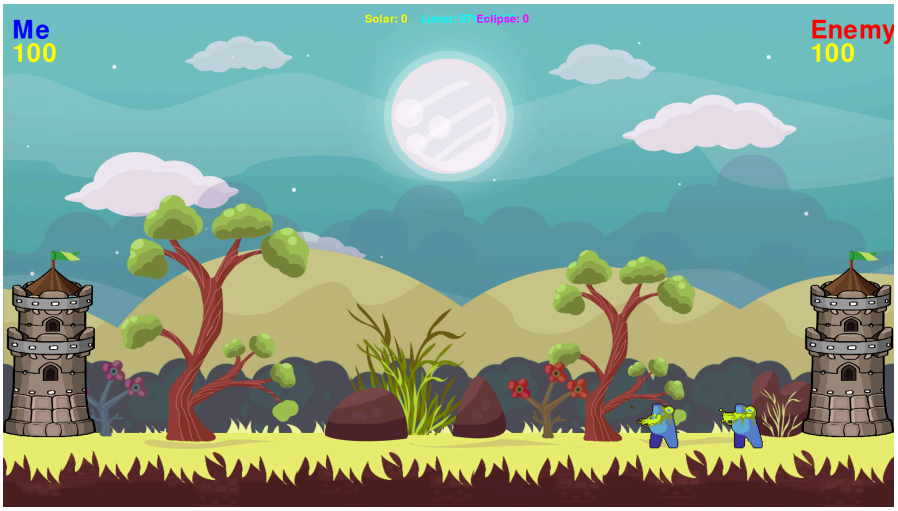
Feature	Mean	Median	Variance
Units Deployed	25	24	4.2
Damage Dealt	3,200	3,000	500
Battle Duration	120s	115s	15.8s

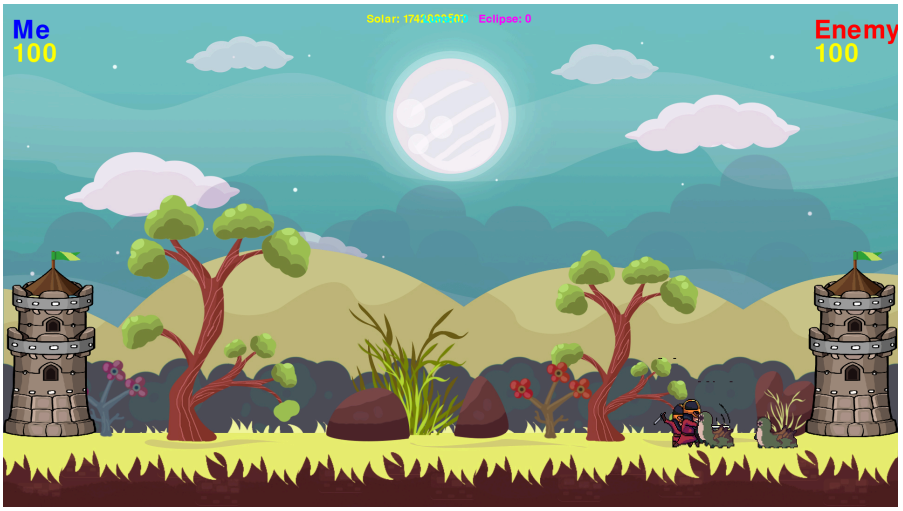
Resources Used	Solar:10 Lunar:10 Eclipse:2	Solar:7 Lunar:12 Eclipse:1.5	Solar:2 Lunar:1 Eclipse:0.1
Win/Loss Ratio	68%	70%	2.5

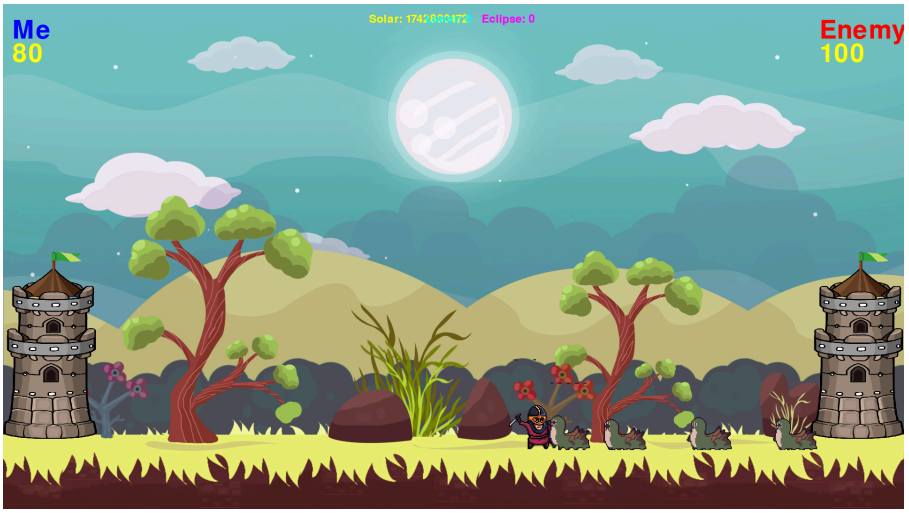
UML:



Demo:







5. Planning Submission

Weekly Goals

- Ensure statistical data collection is implemented by **Week 4**.
- Complete visualization prototypes by **Week 5**.

Milestone Goals

- **50% Completion:** 16 April
- **75% Completion:** 23 April
- **100% Completion:** 11 May

5. Project Timeline

Week	Task
1 (10 March)	Proposal submission / Project initiation
2 (17 March)	Full proposal submission
3 (24 March)	Specify more about the data collected from the game
4 (31 March)	SemiFinished UML and More complete Draft
5 (7 April)	
6 (14 April)	Submission week (Draft)

6. Document version

Version: *1.3*
Date: *31 March 2025*

Date	Name	Description of Revision, Feedback, Comments
15/3	Rattapoom	● Missing Project timeline and document version.
16/3	Parima	Data Analysis Report needs some more details.
29/3	Parima	- Game Concept needs some clarification. - Data Analysis is missing some features and should include more detailed information.
29/3	Rattapoom	● It's nice that you have a working demo already 👍 ● Since you have an autoplay feature, collecting 50 data features is easy. But be careful about the bot battles resulting in the same outcome.