“Elemental dungeon”

1. Project Overview

This project is a ***2D survival wave-based*** game combining strategic ***deck-building mechanics with elemental-themed skills***. Players choose and combine different elemental skills to create unique strategies to survive endless waves of increasingly challenging enemies. Skills are stored in a CSV file, allowing easy customization and expansion. A cooldown system adds strategic depth, encouraging thoughtful usage of each skill. **The player needs to go through Trial and error to find the right deck, right strategy to survive as many waves as possible.**

2. Project Review

The game concept takes inspiration from classic wave-survival games such as "Vampire Survivors" and "Risk of Rain," but integrating a customizable elemental skill deck system, dynamic cooldown management, stamina system, and targeting the closest entities for enemies and summons. Unlike these existing projects, my game emphasizes creative combinations of elemental skills, deeper strategic gameplay via cooldown management, various skill types and intriguing visual effects.

3. Programming Development

3.1 Game Concept

- Elemental Deck-Building: Players build a deck of 4 skills from different elements.
- Strategic Cooldowns: Each skill has a cooldown period, necessitating strategic usage.
- Limited stamina: the player needs to manage their stamina wisely, else being killed :)
- Summoning Mechanics: Skills include summoning allies like the Shadow summons, enhancing tactical depth.
- Endless Waves: Gameplay scales in difficulty, testing players' strategic adaptability. As the game progresses, enemies become tankier.

- ***(If time allows)*** Various enemy types: Adding diversity to the game by adding Boss type (when reaching a certain level), Ranged type (player must manage their stamina to reach them, they may be able to teleport away or run away), Tank type (move slower but deal more damage and very tanky, this type will punish the player who don't manage their skills), Assassin type (move the quickest but the squishiest among all enemy types, this type will punish the player when doesn't manage their).

These are questions that need to be calculated wisely by the player.

How to develop a strategy to advance to higher surviving waves?

Which deck/skills should be picked for surviving?

What strategy should the player use?

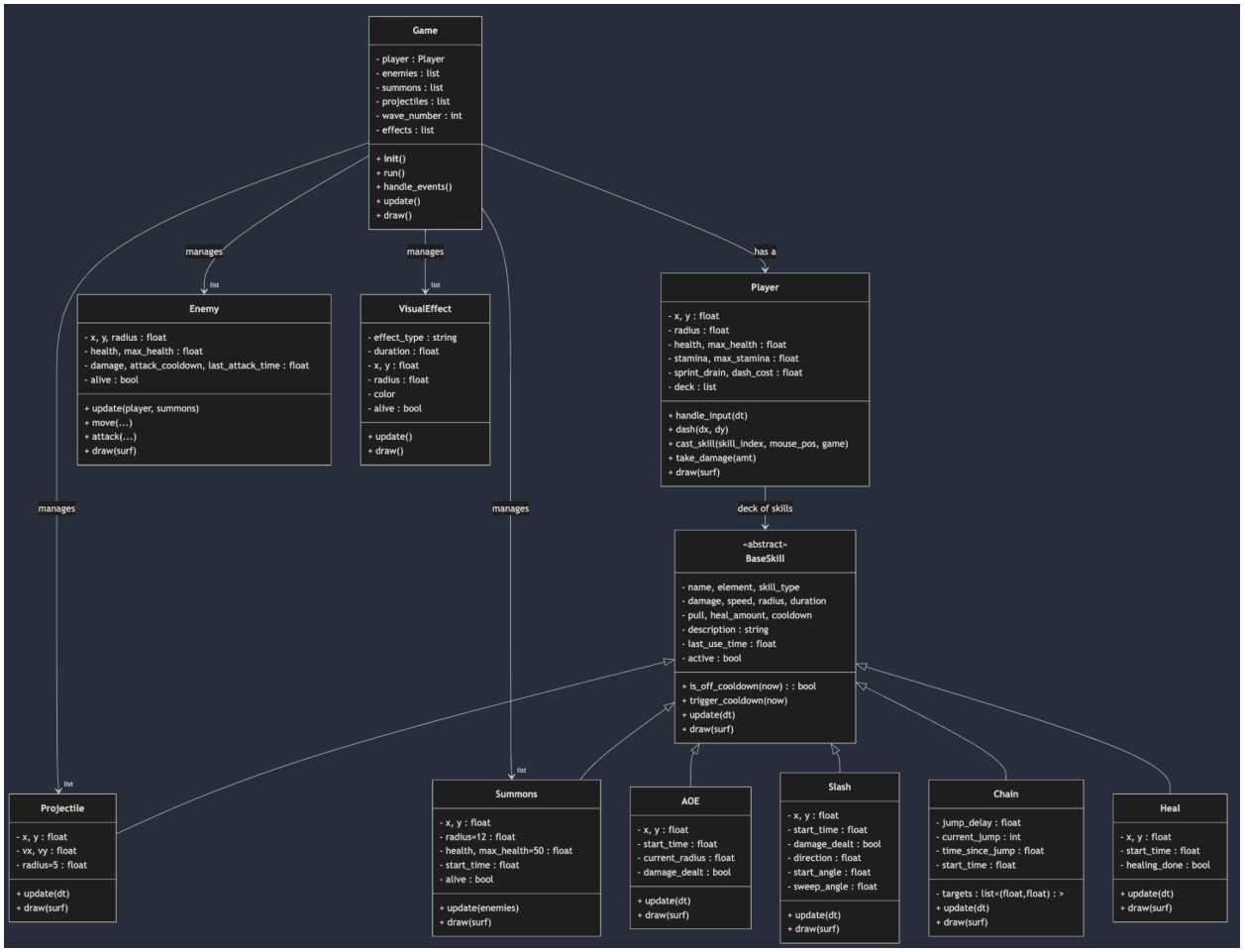
Keep grouping enemies and freeze them and kill them one by one?

Keep summoning and heal the summons?

Going all in with only offensive skills?

Keep poking the enemy while running around?

3.2 Object-Oriented Programming Implementation



(1) Game Class

Role:

The primary orchestrator that manages **global** game state and loops. It holds references to **Player**, **Enemy** objects, **Summons**, and **Projectiles**, updates them, and handles wave logic.

Key Attributes:

- self.player: The single Player instance.
- self.enemies: A list of Enemy objects in the current wave.

- self.summons: A list of **all** Summons objects (these no longer live inside Player, so they can be globally updated).

- self.projectiles: A list of Projectile objects.

- self.wave_number: Tracks progression of difficulty.

- self.effects: A list of short-lived VisualEffect objects (slashes, AOE indicators, etc.).

Key Methods:

- __init__(): Initializes the screen, loads skills, sets up the player, wave, etc.

- run(): Main loop, calls handle_events(), update(), draw() each frame.

- handle_events(): Reads input from Pygame, triggers skill usage or dashes, etc.

- update():

1. player.handle_input(dt) (move player, manage stamina).

2. Update **projectiles** globally with collision checks against enemies.

3. Update **summons** AI (chasing enemies).

4. Update **enemies** (chasing player or summons).

5. Remove any dead or inactive objects from their lists.

6. Check wave completion logic.

- draw(): Renders the entire game—player, enemies, projectiles, UI bars, etc.

(2) Player Class

Role:

A user-controlled entity with HP, stamina, and a **deck** of skills. But it **no longer** forcibly updates all Summons/Projectiles. Instead, it only **spawns** them; the Game class does the global updates.

Key Attributes:

- x, y, radius: Position & hitbox.
- Name
- health, max_health: Tracks HP.
- stamina, max_stamina, sprint_drain, dash_cost: Movement resource.
- deck: A list of Skill objects the player can cast.
- Optionally: walk_speed, sprint_speed.

Key Methods:

- handle_input(dt): Check WASD, SHIFT for sprint, SPACE for dash, clamp positions, drain/regain stamina.
- dash(dx, dy): Submethod for applying dash if enough stamina.
- cast_skill(skill_index, mouse_pos, game): Tells the chosen skill to cast itself, passing in the global game so that skill can spawn new Projectiles or Summons in game.projectiles / game.summons.
- take_damage(amt): Adjust HP.
- draw(surf): Renders the player's sprite or circle.

(3) Enemy Class

Role:

Represents wave-based enemies. They handle their own HP, cooldown, and movement logic (like chasing the player or the nearest Summons).

Key Attributes:

- x, y, radius: Position & hitbox
- health, max_health, alive: HP system.
- damage, attack_cooldown, last_attack_time: Attack logic.

Key Methods:

- update(player, summons): Or update(player, game) to find nearest target, move, handle attack checks.
- move(...): Basic chase logic.
- attack(...): If in range, deal damage if off cooldown.
- draw(surf): Renders the enemy's sprite or bounding circle.

(4) BaseSkill Class

Role

Provides a **common parent** for all specific skill classes. Holds shared data (damage, speed, radius, cooldown) and defines placeholders for update(...) and draw(...).

Key Attributes

- name, element, skill_type: Basic identification of the skill.
- damage, speed, radius, duration, pull, heal_amount, cooldown, description: Various stats.
- last_use_time: Tracks when it was last used (for cooldown checks).
- active: Boolean to indicate if the skill effect is still ongoing.

Key Methods

- **is_off_cooldown(now)**: True if (now - last_use_time) >= cooldown.
- **trigger_cooldown(now)**: Sets last_use_time = now.
- **update(dt)**: A default method (does nothing unless overridden).
- **draw(surf)**: Also a default placeholder.

Subclasses of BaseSkill : Each subclass represents a specific type of skill with unique behavior:

(4.1) Projectile:

Role

Represents a **flying projectile** (e.g., bullet or fireball) that travels in a straight line and deals damage on contact.

Key Attributes

- x, y: Current position of the projectile on screen.
- vx, vy: Calculated velocity based on speed and the target direction.
- radius=5: A small visual circle.
- Inherits damage, speed, color, etc. from BaseSkill.

Key Methods

- **update(dt)**:

1. Moves (x, y) by (vx * dt, vy * dt).
2. Deactivates (active=False) if out of screen bounds.

- **draw(surf)**:

(4.2) Summons:

Role

Spawns a **friendly minion** (like a wraith) that moves around, chases enemies, and deals continuous or small damage on contact.

Key Attributes

- x, y, radius=12: Position and collision circle.
- health, max_health=50: Summon's HP.
- start_time: So it can vanish after duration seconds.
- Inherits damage, speed, etc. from BaseSkill.

Key Methods

- **update(enemies)**:

1. If time.time() - start_time exceeds duration, sets alive=False.
2. Finds the nearest enemy and moves towards it.
3. Deals small damage if close enough.

- **draw(surf)**:

Draws a circle for the minion plus a small HP bar.

(4.3) AOE (Area of Effect):

Role

Creates an **Area of Effect** circle that expands (or animates) and can damage or otherwise affect things within it.

Key Attributes

- x, y: Center location of the AOE effect.
- start_time: For timing how long the effect lasts.
- current_radius: The expanding radius over time.

Key Methods

- **update(dt)**:

1. Compute elapsed time since start_time.
2. If beyond duration, active=False.
3. Otherwise, increase current_radius proportionally.

- **draw(surf)**:

Draws a circle outline or ring with the current radius around (x, y).

(4.4) Slash:

Role

Represents a **short-range slash** or arc in front of the player. Damages enemies that fall within the arc's sweep.

Key Attributes

- x, y: Slash's origin (player's position).

- direction: Angle from the player to the cursor.
- start_angle, sweep_angle: Define the arc range (e.g., 60° slash).
- start_time: For the slash's brief animation.

Key Methods

- **update(dt)**:

1. If (time.time() - start_time) > duration, set active=False.

- **draw(surf)**:

Draws an **arc** from start_angle to start_angle + partial_sweep, depending on elapsed time.

(4.5) Chain:

Role

A **chain lightning** or "jumping" effect that leaps from one target to the next in short intervals.

Key Attributes

- targets: A list of positions (or references) that the chain jumps between.
- jump_delay=0.1: Time between each jump.
- current_jump: Tracks which step of the chain we're on.
- time_since_jump: Accumulator to see if it's time to jump again.
- start_time: So we can end the chain after duration.

Key Methods

- **update(dt)**:

1. If total time > duration, active=False.

2. Accumulate time_since_jump += dt; once it surpasses jump_delay, increment current_jump.

3. If current_jump reaches the end of targets, set active=False.

- **draw(surf)**:

Draws a line from targets[current_jump] to targets[current_jump+1].

(4.6) Heal:

Role

Heals a location or caster over a brief animation or instantly, depending on your design.

Key Attributes

- x, y: Center of the heal effect (caster's location).
- start_time: For animating.
- healing_done: A boolean if you want a single-time heal or continuous.

Key Methods

- **update(dt)**:

1. If (time.time() - start_time) >= duration, active=False.

- **draw(surf)**:

Possibly draws a shrinking or expanding circle to show a heal effect.

5. VisualEffect Class

- Role: Handles visual effects for skills and abilities
- Key Attributes:
  - `effect_type`, `duration`
  - `position (x, y)`, `radius`
  - `color`, `alive`
- Key Methods:
  - `update()`: Updates effect state
  - `draw()`: Renders the effect

3.3 Algorithms Involved

1. Nearest-Target "Pathfinding"

• Each enemy (and friendly summon) picks the closest target—player or another entity.

• Linear search for the minimum distance: we iterate over possible targets, compute Euclidean distance with math.hypot(...), and keep track of the closest.

• Though not a classic pathfinding like A* or BFS, it's a greedy approach: move directly toward the nearest target.

2. Event-Driven Mechanics (Pygame)

• The game loop processes player input (keyboard & mouse) through Pygame's event queue.

• Algorithm:

• Continually poll pygame.event.get() for new events, handle them (key pressed, mouse clicked, etc.).

• Effect: robust event-driven input system, used for dashing, skill casting, etc.

3. Collision & Damage Checks

• Overlap detection for circle collisions:

- Algorithm: compute distance with math.hypot(dx, dy) and compare to (radiusA + radiusB).
- If distance < sum of radii, we say “overlap => damage” or “resolve overlap.”

4. Cooldown & Rule-Based Logic

- Skills and enemy attacks rely on a cooldown timer.
- If (current_time - last_attack_time) >= attack_cooldown, the next attack is allowed.
- The same approach is used for skill usage: “If current_time - skill.last_use_time >= skill.cooldown => skill is off cooldown.”

5. Wave Progression

- Each wave spawns more enemies or stronger enemies (by wave_number).
- After all enemies are cleared, wave_number++
- Then spawn e.g. (5 + wave_number) new enemies or scale HP by wave_number * WAVE_MULTIPLIER.

4. Statistical Data (Prop Stats)

4.1 Data Features

	Why is it good to have this feature data? What can it be used for?	How will you obtain 50 values of this feature data?	Which variable (and which class) will you collect this from?	How will you display this feature (via summarization statistics or via graph)?
Player Name	To determine who has played this game. This can be used for ranking players according to their highest wave count and game duration.	50 played games	Through Player class. (name instance)	Displayed through a bar-chart ranking system. X-axis: waves Y-axis: name (only top ten)
Wave Reached	Tracks the number of waves each player reached. Will be used for analyzing the best deck	50 played games	Through Game class. (wave_number instance)	Summarized with Deck composition, Best player, Skill selection
Individual Skill Selection Frequency	Evaluate how often individual skills appear in decks and correlate these with both wave reached and total game durations, highlighting consistently effective skills.	50 played games	We get this data from deck composition.	Displayed through comparative pie charts. (only top 10, +1 for others)

Skill used count per waves	To analyse user behavior: why did they die	At the end of the wave from each game (50 games)	Through Game class. (handle key)	Via correlation heatmap (how player usually die)
Hp at the end of the wave	To analyse user behavior: why did they die	At the end of the wave from each game (50 games)	Through Game class. (self.player.hp)	Via correlation heatmap (how player usually die)
Stamina at the end of the wave	To analyse user behavior: why did they die	At the end of the wave from each game (50 games)	Through Game class. (self.player.stamina)	Via correlation heatmap (how player usually die)
Game Duration	Records the total playtime of each session, indicating how long each deck/player combination lasted. Will be used for analyzing the best deck	50 played games	Through Game class. (time.time() - game_start_time instance)	Summarized with Deck composition, Best player, Skill selection

4.2 Data Recording Method

The game will store data as CSV files. These files will store in this format

log.csv

Play_ID	name	waves reached	skill1	skill2	skill3	skill4	Time in survived waves (second)
0001	Keen	2	a	b	c	d	25
…	…	…	…	…	…	…	…

waves.csv

Play_ID	name	waves	hp	stamina	skill1 frequency	skill2 frequency	skill3 frequency	skill4 frequency	time/ waves (sec)	Spawned enemies	Enemies left
0001	Keen	1	95	23	1	2	2	0	10	5	0
0001	Keen	2	20	2	3	2	1	3	15	7	0
0001	Keen	3	0	50	4	3	4	6	35	9	2
0002	uzimp	1	85	2	15	1	2	3	60		
…	…	…	…		…	…	…	…	…		

4.3 Data Analysis Report

Red one means it might took more time to complete it. Or if time allows

	Features	Objective	Type	X-axis	Y-axis
1	• The total time each game lasted • The wave reached	Analyse the range of wave the most people usually reached. and time consumed	table(display min, max, mean, mode, median, sd)	-	-
2	Player Name, Wave reached, and (as a tie-breaker) Game Duration.	Find the top 5 players with the highest reached wave with minimal time.	Bar graph	The number of survived waves	Player (If 2 players has reached the same waves, game duration will be used for justifying this)
3	4 selected skills	Find the top 5 decks	Bar graph	Deck	Frequency
4	Each skill's pick count across all recorded decks.	Find the top 5 skills from all games	Pie chart (showing percentage of the most selected skills)	-	-
5	• Stamina at the last wave • Hp at the last wave • Skills used at the last wave • Time at the last wave • Enemies spawn/left The last wave is the wave where hp=0	Analyse the cause of death. Players may struggle from running out of stamina. Cooldown management. This analysis can be used for balancing the game.	Correlation heatmap between those features	-	-
6	• The total time each game lasted • The wave reached	Analyse the range of wave the most people usually reached. and time consumed	Scatter plott	Duration	Wave reached
7	Wave reached	Analyse the range of wave the most people usually reached	Histogram	Wave numbers	Frequency

5. Project Timeline

Week	Task
1 (10 March)	Proposal submission / Project initiation
2 (17 March)	Full proposal submission
3 (24 March)	
4 (31 March)	Core features are 60% completed as of now
5 (7 April)	• Restructure classes (adding entities class for handling player and enemies) • Adding direction of player and enemies (for adding sprite)
6 (14 April)	Submission week (Draft)
26 March-2 April	Core features are all completed
3 April-9 April	• Adding more elem/skill types • UX/UI
10 April-16 April	Visual effects, Sound effects.
17 April-23 April	Balance stats between skills, attack interval, cooldown, stamina, etc.
24 April-11 May	, Data visualization, Finish

6. Document version

Version: *4.0*
Date: *30 March 2025*

Date	Name	Description of Revision, Feedback, Comments
12/3	Phiranath	Don't forget to change the project name and format some text. The overall idea is interesting. The project review seems to be very hard to do in the given time, some of the statements such as "improve significantly" or "advanced AI behavior" seems to be hard. For the data feature, you can also save the number of enemies killed, skill used, and total summons.
16/3	Pattapon	Great Job. I agree with Phiranath. :)