Angry Tao Project

1. Project Overview

This game's main mechanic focuses on completing stages using random items that the user receives. The player won’t know which item they will get and must use it creatively to complete the stage. The second core mechanic involves a non-moving stage where enemy and obstacle positions change dynamically, ensuring that no two stages play the same. This setup encourages different strategic approaches to overcoming challenges.

2. Project Review

This game will improve the intensity of the thinking process to make users have different ways to complete instead of using what is specific for the stage. The original of this game is the same type as the “Angry bird” game. In Angry bird the attack will have a angle in attack or can be horizontal lines but for this game project the attack will only be horizontal lines without angle come included in the shooting so that mean you no need to calculate the perfect angle to shoot, the attack can only be going left horizon line or right horizon line depend on current character location if the character at left side box the attack will go only right horizon line but the opposite side will do the opposite(The shooting side will always facing the stage box).

3. Programming Development

3.1 Game Concept

The objective of the game is to destroy all enemies in the stage that have random obstacle location and random enemy location and the user will have a different item or weapon from the random pool so they don’t know which they are gonna get for the creative experience for the user to find ways to complete the stage.

3.2 Object-Oriented Programming Implementation

The game’s architecture follows an object-oriented approach with the following primary classes:

3.2.1 Stage Class

- Purpose: Handles obstacle, enemy, and civilian placement in randomized positions for each stage.
- Key Attributes:
  - stage_number: Current stage number.
  - stage_obstacles: Group of Obstacle sprites.
  - enemies: Group of Enemy sprites.
  - civilians: Group of Civilian sprites.
  - border_rect: Defines the playable area boundary.
- Key Methods:
  - generate_level(): Randomly generates obstacles, enemies, and civilians for the stage.
  - check_stage_clear(): Returns True if all enemies are eliminated.
- Details: Obstacles have different materials (Glass, Wood, Iron, Diamond, Gold, Bomb, Vibranium) and are placed randomly in the middle stage box. The stage area is visually separated with a red border.

3.2.2 Character Class

- Purpose: Represents the player character, managing movement, shooting, weapon switching, and score.
- Key Attributes:
  - image: Character sprite image.
  - rect: Sprite rectangle for position.
  - items_list: List of available weapon/item objects.
  - bullets: Group of bullet sprites.
  - score: Player’s current score.
  - current_item: Currently selected weapon/item.
- Key Methods:
  - move(dy): Move character vertically (with wrap-around at edges).
  - shoot(): Fire the current weapon.

    - switch_item(direction): Change the selected weapon.
    - update_cooldown(): Update firing cooldown.
    - all_ammo_empty(): Check if all weapons are out of ammo.

3.2.3 Item Class

- Purpose: Defines the various weapons/items players can use.
- Key Attributes:
    - name: Name of the weapon (Normal, Spread, Explosive, Shatter, Heat, Ice).
    - skill: Skill type.
    - fixed_ammo: Ammo count.
    - max_ammo: Maximum ammo.
    - color_code: Color for UI display (Normal: Grey, Spread: Green, Explosive: Red, Shatter: White, Heat: Orange, Ice: Blue).
- Key Methods:
    - use_skill(): Performs the weapon's action.

3.2.4 Config Class

- Purpose: Stores game configuration settings (window size, stage dimensions, colors, etc.).
- Attributes: Static configuration values used throughout the game.

3.2.5 UI Class

- Purpose: Manages the user interface, including item selection, ammo tracking, game-over screens, and resets.
- Key Methods:
    - draw_item_ui(character): Draws weapon/item UI.
    - draw_score(character): Draws current score.

3.2.6 Game Class

- Purpose: Main controller for game loop, events, rendering, and state management.
- Key Attributes:
    - screen: Pygame display surface.
    - background_image: Background image surface.

    - ui: UI manager.
    - character: The player character object.
    - stage: Current stage object.
    - drawer: Handles drawing elements.
    - explosive_blocks: Group of explosive block sprites.
    - current_stage, running, game_over, game_over_due_to_civilian, etc.
- Key Methods:
    - run(): Main game loop.
    - update_game(): Updates game state.
    - handle_collisions(): Handles all collisions.
    - check_game_over(): Checks for game over conditions.
    - draw_game_over(): Draws game over UI.
    - show_upgrade_menu(): Shows upgrade menu.
    - handle_portals(): Handles portal logic.

3.2.7 Drawer Class

- Purpose: Handles drawing of all game elements (portals, stage, characters, bullets, etc.).
- Key Methods:
    - draw_portal_lines(), draw_stage(stage), draw_portals(character), draw_character(character), draw_bullets(character), draw_enemies(stage), draw_explosive_blocks(blocks).

3.2.8 Obstacle Class

- Purpose: Static object that blocks bullets and characters; can be destroyed.
- Key Attributes:
    - image: Obstacle sprite image (with black border).
    - rect: Sprite rectangle for position.
    - material: Material type.
    - health: Hit points before destruction.
- Key Methods:
    - take_damage(damage): Reduces health, returns if destroyed.

3.2.9 Enemy Class

- Purpose: Target to be eliminated by the player.
- Key Attributes:
    - image: Enemy sprite image.
    - rect: Sprite rectangle for position.
    - health: Hit points before destruction.
- Key Methods:
    - update(): (Optional) Handles movement or behavior.
    - take_damage(damage): Reduces health, returns if destroyed.

3.2.10 Civilian Class

- Purpose: Non-hostile character; shooting a civilian penalizes the player.
- Key Attributes:
    - image: Civilian sprite image.
    - rect: Sprite rectangle for position.
    - health: Hit points before removal.
    - move_timer: Timer for random movement.
    - direction: Current movement direction.
- Key Methods:
    - update(): Handles random movement.
    - take_damage(damage): Reduces health, returns if destroyed.

Relationships:

- Game owns Character, Stage, Drawer, UI, and manages sprite groups.
- Stage manages collections of Obstacle, Enemy, and Civilian.
- Character manages a list of Item objects (weapons/skills).
- Drawer and UI are utility classes for drawing.

<u>3.3 Algorithms Involved</u>

Several core algorithms will be used to power the game mechanics:

3.3.1.Randomization Algorithm: Used to determine the item pool and spawn locations of obstacles and enemies.

3.3.2.Sorting Algorithm: Applied to manage item selection and upgrades.

4. Statistical Data(Prob Stats)

4.1 Data Features

Feature	Why it is good to have this 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 data(via summarization statistics or via graph?)
Movement of character	Track movement used of player and used it for analyzes how much move can player used to clear the stage per stage	Make log of player movement in every second by track the button clicking for each stage, by making list separate for each stage to store the moved	Class:Character Subclass:position	Stack bar graph , by there gonna be two block per one bar the first is up and another is down, the x-axis is gonna be number of stage and y-axis is gonna the amount of move
Items usage	Show trends of player to see which item is most used, To track and balancing the item to be on par	Record each item selection with they own record number to see which is selected the most or less	Class:Items Subclass:item_type	Bar graph to show each items usage, for the x-axis is the item names and for the y-axis is gonna be the

	with other items			num of usage the player chose
Num of defeated enemy	This is used for balancing the difficult of each stage to make sure the game don't go to long and make sure the success hit rate from player	Record the count of enemies defeated per stage with the list of data for each stage and record time the enemy die by time will start at time the character first move	Class:Stage Subclass:enemy _count	Line graph for the x axis is gonna be time that used to clear enemy and the y axis is for the amount enemy
Num of destructed obstacle	This is used for analyzes average of destructed obstacle that need to complete the stage(Kill all enemy at that stage) and for balancing the number of obstacle can be in one stage	Record the count of obstacle defeated per stage with the list of data for each stage and collect type of the obstacle to see which type got destroyed	Class:Stage Subclass:obstacl e_count	Bar graph to comparing between the destructed obstacle (I use bar graph because the obstacle come with various type), In x-axis gonna be type of the obstacle and for y-axis is number of destructed
Ammo traking	Used for ensure that player don't overuse the ammo and make sure the game is over if all ammo is out and this feature can help analyze the average the amount of ammo used to clear per stage	Log of used ammo consumption for per stage by sum all ammo from all items used	Class:character Subclass:ammo_ count	Stacked bar graph, the one stack gonna be separated of the item ammo and the x-axis is gonna be number of stage and y-axis is gonna be amount of ammo used
Bullet	To used in analyzes the	Log missed short (Hit	Class:character	Summarization statistics for

hitted borderline	accurate of player	second border line) with data for per stage	Subclass:missed _shot	number of bullet hitted borderline per stage

Graph no:	Feature name	Graph objective	Graph type	X-axis	Y-axis
Graph 1	Movement of character	Analyze player movement per stage	Stacked bar graph	Number of stages	Amount of moves, by separate between up and down
Graph 2	Items usage	Track most used items for balancing	Bar graph	Item names	Number of times chosen
Graph 3	Num of defeated enemy	Balance difficulty and track player accuracy	Line graph	Time used to clear stage	Number of enemies defeated
Graph 4	obstacle	Track average obstacles destroyed per stage	Bar graph	Obstacle type	Number of obstacles destroyed
Graph 5	Ammo traking	Ensure fair ammo use and analyze consumptio n per stage	Stacked bar graph	Number of stages	Amount of ammo used, separate between difference item

4.2 Data Recording Method

The game project will store statistical data using a CSV file for structured logging. This format allows easy access for later analysis. Alternatively, SQL databases may be used for more efficient data management if real-time querying is required.

4.3 Data Analysis Report

The recorded data will be analyzed using:

4.3.1.Descriptive Statistics (mean, median, mode) to summarize key gameplay trends(Item used).

4.3.2.Graphs and Charts (bar charts, line graphs) to visualize player choices and performance.

5. Project Timeline

Week	Task
1 (10 March)	Proposal submission / Project initiation
2 (17 March)	Full proposal submission
3 (24 March)	Created stage and character shooting mechanic
4 (31 March)	Created items skill and score
5 (7 April)	Decorations
6 (14 April)	Submission week (Draft)

5.1 Milestone Breakdown

5.1.1.50% Completion by 16 April:

-Stage creation

-Shooting mechanics

-Initial item implementation (Skill)

5.1.2.75% Completion by 23 April:

-UI development

-Statistical data tracking

-Game balancing and early testing

5.1.3.100% Completion by 11 May:

-Final polishing

-Fixing bug and separate file

6. Document Version

Version 5
Date: 10 May 2025

Date	Name	Description of Revision, Feedback, Comments
15/3	Phiranath	Don’t forget to remove the italic format and fill out the timeline. Project overview and review are a little bit confusing, especially the horizontal and vertical part.
16/3	Rattapoom	I think the “stage” needs a brief explanation of what it is.