Help

1. Project Overview

This project aims to create a 2D platform game using Python and the `Pygame` library. The game will feature a player-controlled character that can run, jump, and navigate through a series of custom-designed levels while avoiding obstacles and defeating enemies through shooting mechanics.

Game Mechanics:

- The player can move left, right, jump, and shoot.
- Shooting will involve directional aiming, with bullets or projectiles being fired toward enemies.
- The game world will consist of unique and custom-designed maps, featuring platforms, hazards, and power-ups.
- Enemies will have AI-based movement and behavior.
- The player will have a health bar and limited lives.
- The game ends when the player loses all lives or completes all levels.

2. Project Review

The inspiration comes from this https://youtu.be/2gABYM5M0ww?si=MbEsIMn0YEOSjE27.
The key differences will be the map design and the attack mechanics, which will be changed to shooting instead.

3. Programming Development

3.1 Game Concept

The platform game will feature a player-controlled character that navigates levels by running, jumping, and shooting enemies. The player will encounter various platforms, obstacles, and enemies.

Key Features:
- Player movement (moving, jumping, shooting).
- Gravity-based physics and collision detection.
- AI-driven enemy behavior (chasing, patrolling, and attacking).
- Directional shooting (fixed or mouse-aimed).
- Level completion and game-over conditions.
- Score calculation based on collected items and defeated enemies.

3.2 Object-Oriented Programming Implementation

- `Player()` Represents the player character. Manages movement, jumping,and shooting
- `Platform()` Represents static or dynamic platforms in the game world.
- `Enemy()` Represents an enemy. Handles AI-based movement and health.
- `Projectile` () Represents bullets or projectiles fired by the player or enemies.
- `Game()` Manages the game state and game loop. Controls player turns, game rules, and win/loss conditions.

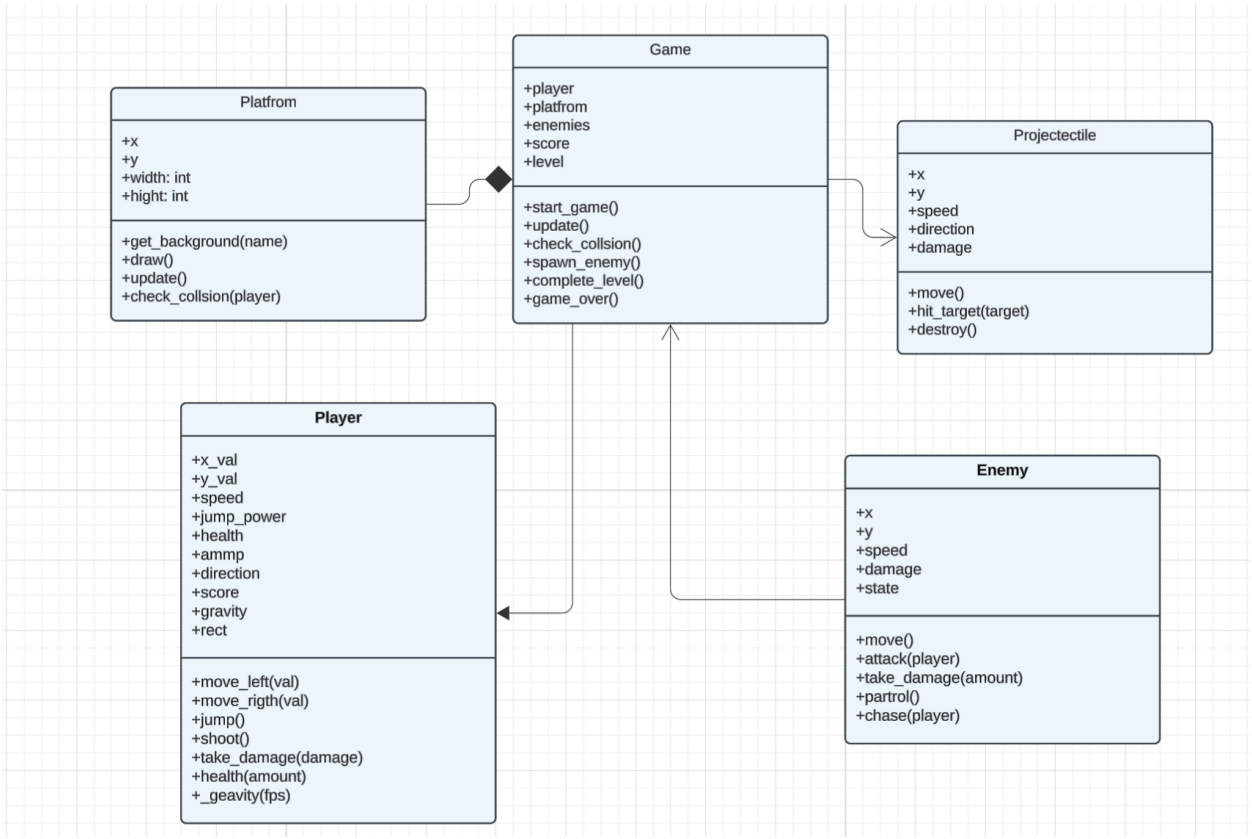
Class	Role	Attributes	Methods
Player	Represents the player character. Manages movement, jumping,	`x_val`, `y_val` – Player's position on the screen. `speed` – How fast the player	`move_left()` – Moves the player left. `move_right()` – Moves the player right.

	shooting, and health.	moves. `jump_power` – How high the player can jump. `health` – Current health of the player. `ammo` – Number of bullets available. `direction` – Direction the player is facing ("left", "right"). `score` – Total points collected. gravity - Gravity in game rect - defines the **position** and **size** of an object on the game screen. ani_count, fall_count - check that animation works or not, and check that the gravity works or not when the player falls.	`jump()` – Makes the player jump. `shoot()` – Fires a bullet. `take_damage(damage)` – Reduces player health. `heal(amount)` – Restores player health. _gravity() - make the gravity in game
Platform	Represents solid platforms where the	`x`, `y` – Platform position.	`get_background()` -

	player and enemies can stand.	`width`, `height` – Size of the platform.	designed to load a background image and tile it across the game screen. `draw()` – Displays the platform on the screen. `update()` – Moves platform (if moving). `check_collision(player)` – Checks if the player is standing on the platform.
Enemy	Represents an enemy with AI-based movement and health.	`x`, `y` – Enemy position. `speed` – How fast the enemy moves. `health` – Current health of the enemy. `damage` – Damage dealt to the player. `state` – Current state of the enemy ("patrolling",	`move()` – Moves the enemy. `attack(player)` – Damages the player. `take_damage(amount)` – Reduces health when hit. `patrol()` – Moves back and forth.

		"chasing", "attacking").	`chase(player)` – Follows the player when nearby.
Projectile	Represents bullets or projectiles fired by the player or enemies.	`x`, `y` – Position of the projectile. `speed` – How fast the projectile moves. `direction` – Direction of movement ("left", "right"). `damage` – Damage dealt on hit.	`move()` – Moves the projectile. `hit_target(target)` – Damages the target. `destroy()` – Removes the projectile.
Game	Manages the game state, rules, and progression.	`player` – The player object. `platforms` – List of platforms. `enemies` – List of enemies. `projectiles` – List of projectiles. `score` – Current score. `level` – Current game level.	`start_game()` – Sets up the game. `update()` – Updates the game state. `check_collision()` – Handles collisions. `spawn_enemy()` – Creates new enemies. `complete_level()` – Advances to the next level. `game_over()`

			– Ends the game and shows the score.game and displays score.



3.3 Algorithms Involved

- **Physics Engine:**

- Gravity-based physics for realistic jumping
- Friction and collision detection with platforms.

- **Shooting Mechanic:**

- Aim-based shooting using mouse or fixed directional input.
- Bullet trajectory calculation using trigonometry.

- **Collision Detection:**

- Player-to-platform, player-to-enemy, bullet-to-enemy collision detection.
- Rectangle-based hitboxes.

4. Statistical Data (Prop Stats)

4.1 Data Features

- Player Score, Total points scored by the player.
- Enemies Defeated, Number of enemies killed by the player.
- Game Duration, Total time taken to complete a level.
- Platform Interaction, Number of platforms landed on.
- Damage Taken, Amount of damage the player received during gameplay.

Feature	Why It's Good	How to Collect 50 Values	Variable/Class	How to Display
Player Score	Measures player performance and game balance	Track score every 10 seconds during each level	`score` in `Game` class	Line chart for score trend
Enemies Defeated	Reflects player engagement and challenge level	Count each enemy killed	`enemies_defeated` in `Player` class	Bar chart for enemy type and count
Game Duration	Helps understand	Record time every 10	`duration` in `Game` class	Line chart for game time

	player engagement and level design	seconds while the player is in a level. If the player survives at least 10 seconds, it confirms they reached and engaged with the level		
Platform Interaction	Shows level design effectiveness	Count number of platforms landed on	`platform_count` in `Game` class	Pie chart for platform type
Damage Taken	Measures player challenge and difficulty	Track health decrease after enemies hit player	`health_lost` in `Player` class	Bar chart for damage trends

Feature Name	Graph obj	Graph Type	x-axis	Y-axis
Player Score	Show score trend over time	Line Graph	Time (sec)	Score
Enemies Defeated	Compare enemy in each level defeated	Bar Graph	Enemy in each level that player kill	Count
Platform	Show pass	Pie Chart	Level of	Proportion /

Interaction	rate for each level		platform that player pass	Percentage

3.2 Data Recording Method

- Data will be stored in a **CSV file** for easy processing and analysis.
- Each game session will generate a new row in the CSV file.

3.3 Data Analysis Report

- **Score Trends** – Average score per game.
- **Enemy Behavior** – Average number of enemies defeated.
- **Game Duration** – Average time per round.
- **Graphs and Charts** – Presentation data.
    - Pie chart for game over or win game.
    - line chart for enemy that player kills.

4. Project Timeline

Week	**Task**
1 (10 March)	Proposal submission / Project initiation
2 (17 March)	Full proposal submission
3 (24 March)	Initial Development Phase
4 (31 March)	Development Phase
5 (7 April)	Testing and Debugging Phase Enhancement Phase
6 (14 April)	Polishing Phase Submission week (Draft)
Week	**Task**
26 Mar - 2 April	Finish structure and make platform level one

3 April - 9 April	Finish platform level one (can play)
10 April - 16 April	Collect data to make a graph to submit in the first draft. Start making platform level 2 and 3.
17 April - 23 April	Finish platform level 2 and 3.
24 April - 11 May	Testing and Debugging Phase Enhancement Phase

List 50% of the tasks that you expect to complete by 16 April.

- Map of all levels in the platform.
- Platform level 1 can play (kill enemy, move, jump, dying).
- Collect data that needs to be used to make graphs.

List 75% of the tasks that you expect to complete by 23 April.

- Finish platform level 2 and 3. (kill enemy, move, jump, dying)

List 25% of the tasks that you expect to complete by 11 May.

- Testing and Debugging Phase Enhancement Phase.
- Make a graph.

5. Document version

Version: *1.0*
Date: *4 March 2025*

Date	Name	Description of Revision, Feedback, Comments
14/3	Pattapon	Good job. :) You should add more details on the OOP implementations, Data Features, and Data Analysis part as mentioned in my comments.
16/3	Phiranath	Missing two data features and detail on how you would display the data. Don’t forget to remove the template, fill in the time table, and update the document version.

29/3	Phiranath	Most of the text in this data is in italic format, don’t forget to remove them. 👍 There is a minor mistake in the class diagram, and I suggest some method for collecting data that might lower your data collecting time.