Punnapa Pramotekul ID : 6610545863

Phantom Diner

1. Project Overview

In Phantom Diner, players act as a ghost waiter in a haunted diner. Ghostly customers arrive at tables, place orders, and wait for their food. The player must take orders and deliver dishes before the customers lose patience. The game rewards speed with points, and players advance to the next level by meeting score targets.

Mechanics:

- Ghost Customers:
  - Arrive at random tables and place orders for spooky dishes.
  - Have a "patience meter" that decreases over time. The patience meter decreases by 5% per level until it reaches the minimum patience decrease. This will ensure that the game remains challenging but fair at higher levels.
- Player Actions:
  - Take orders and deliver dishes.
  - Earn points for serving customers quickly (10 points per serving).
- Score and Level Progression:
  - Players must achieve a minimum score (e.g., 100 points) to advance to the next level.
  - Each level lasts 1 minute, with the timer increasing by 10 seconds every fourth level.
- Kitchen Mechanics:
  - Players must go to the kitchen to prepare dishes.
  - Dishes take a fixed time to prepare (e.g., 3 seconds).
- Haunt Events:
  - If a customer’s patience meter reaches zero, they leave, and the player loses points.

Functionality:

- Real-Time Gameplay:
  - Multiple ghost customers arrive and place orders.
- Visual Feedback:
  - Patience meters, orders, and waiting customers are displayed.
- Score Tracking and Level Progression:
  - Players earn points for serving customers quickly.
  - Players must meet score targets to advance to the next level.

Punnapa Pramotekul ID : 6610545863

2. Project Review

Existing Project:

Diner Dash is a popular time-management game where players act as a waitress in a busy diner.

Improvements :

- **Simplified Mechanics:**
  - Focus on core gameplay: serving dishes and managing customer patience.
- **Statistical Tracking:**
  - Track key metrics like score, waiting time, dishes served, and haunt events.
  - Analyze player performance using visualizations (e.g., bar charts, line graphs).
- **Level Progression with Score Targets:**
  - Players must achieve a minimum score (e.g., 100 points) within a time limit to advance.
- **Haunt Events:**
  - Customers leave if their patience runs out, and the player loses points.
- **Supernatural Theme:**
  - Replace human customers and dishes with ghostly customers and spooky dishes.

3. Programming Development

3.1 Game Concept

Detail :

Phantom Diner is a fast-paced, time-management game where players take on the role of a ghost waiter in a haunted diner. Ghostly customers arrive at tables, place orders for spooky dishes, and wait for their food. The player must take orders, deliver dishes before the customers lose patience. The game rewards speed with points, and players advance to the next level by meeting score targets within a time limit.

Objectives

1. **Serve Customers Quickly:**
   - Take orders and deliver dishes before customers lose patience.
2. **Earn Points:**
   - Score points by serving customers quickly.
3. **Advance Through Levels:**
   - Achieve the minimum score within the time limit to unlock the next level.
4. **Avoid Haunt Events:**
   - Prevent customers from leaving by serving them before their patience runs out.

Key Features:

1. **Ghost Customers:**
    a. Arrive at random tables and place orders.
    b. If no tables are available, customers wait in line.
2. **Player Actions:**
    a. Move to the kitchen to prepare dishes.
    b. Pick up and serve dishes to customers.
3. **Patience Meter:**
    a. Customers have a patience meter that decreases over time.
    b. If it reaches zero, they leave, and the player loses points.
4. **Score System:**
    a. Players earn points for serving customers quickly.
    b. A minimum score (e.g., 100 points) is required to advance to the next level.
5. **Time-Limited Levels:**
    a. Each level lasts 1 minute, with the timer increasing by 10 seconds every fourth level.
6. **Queue System:**
    a. Customers wait in line if all tables are full.
    b. A random customer arrives every 5 seconds.
7. **Visual Feedback:**
    a. Display customer patience meters and orders.
    b. Show waiting customers at the bottom left of the screen.

3.2 Object-Oriented Programming Implementation

1. **Player**
    - **Role:** Represents the ghost waiter.
    - **Attributes:**
        - name (str): Player's name
        - speed (int): Movement speed.
        - positions (list): Position of player
        - is_busy(bool): Indicates whether the player is preparing or carrying a dish.
        - current_dish(Dish): Dish currently being prepared or carried
        - normal_animation(AnimatedSprite) : For the player's animation.
        - carry_animation(AnimatedSprite): For the player's animation when carrying a dish.
        - current_animation(AnimatedSprite)
    - **Methods:**
        - move(keys): Moves the player based on key presses.
        - draw(screen) : Draw the player on screen.
        - get_position(), set_position(x,y)
        - get_current_dish(), set_current_dish(current_dish)
        - get_is_busy(), set_is_busy(busy)

2. **Customer**
    - **Role:** Represents a ghost customer at a table.
    - **Attributes:**
        - patience_meter (int): Current patience level.
        - order (str): Dish ordered by the customer.
        - table (Table): Table where the customer is seated.
        - arrival_time (Time) : Time when the customer arrived
        - leave_time (Time): The time when the customer will leave after being served.
        - position(tuple): The position where the customer is drawn on the screen.
        - is_served (bool): Indicating whether the customer has been served.
        - animation(AnimatedSprite): For the customer's animation.
        - served_time(Time) : Track when customer was served.
    - **Methods:**
        - update_patience_meter(paused): Decreases the patience meter over time.
        - leave(): Customer leaves if patience runs out.
        - draw(screen) : Draws the customers and patience meter
        - serve(): Marks the customer as served.
        - should_leave(paused) : Check if the customer should leave after being served.
3. **Table**
    - **Role:** Represents a table in the haunted diner.
    - **Attributes:**
        - position (tuple): (x, y) coordinates of the table.
        - customer (Customer): Customer seated at the table.
        - order_status (str): Status of the order (e.g., "waiting", "served").
        - dish (Dish): The dish served to the customer.
        - chairs (list[Chair]): A list of chairs associated with the table.
    - **Methods:**
        - seat_customer(customer): Seats a customer at the table.
        - clear_table(): Clears the table after the customer leaves.
        - draw(screen) : Draw the table on screen.

4. **Chair**
    - **Role** : represents a chair at a table.
    - **Attributes**:
        - position (tuple): The position of the chair on the screen.
        - customer (Customer): The customer seated on the chair.
    - **Methods:**
        - seat_customer(customer): Seats a customer on the chair.
        - clear_chair(): Clears the chair (removes the customer).
        - draw(screen): Draws the chair and the customer (if seated) on the screen.

5**. Dish**

- **Role**: Represents a dish that can be served.
- **Attributes**:
  - name (str): The name of the dish ("Spider Soup").
  - prep_time (int): The time required to prepare the dish.
  - position (tuple): The position where the dish is drawn on the screen.
- **Methods**:
  - prepare(): Simulates dish preparation (e.g., delays for prep_time seconds).
  - draw(self, screen): Draws the dish on the screen.

6**. Game**

- **Role:** Manages the overall game state.
- **Attributes:**
  - player (Player): The player object.
  - tables (list[Table]): A list of tables in the game.
  - customers (list[Customer]): A list of customers currently seated at tables.
  - waiting_customers (list[Customer]): A list of customers waiting in line.
  - kitchen (Kitchen): The kitchen object.
  - time_left (int): The remaining time in the current level.
  - run (bool): Indicating whether the game is running.
  - last_customer_time (Time): The last time a customer arrived.
  - paused (bool) : Indicating whether the game is paused
  - score (int): The player's current score.
  - level (int): The current level.
  - haunt_events (int): The number of haunt events triggered by customers leaving.
  - dishes_served (int): The number of dishes served.
  - waiting_times (List[Time]): A list of waiting times for customers.
  - player_name (str): The name of the player.
  - screen (Pygame): The Pygame screen object.
  - ui (UI): For displaying game information.
- **Methods:**
  - reset_game_state(): Resets the game state for the next level.
  - place_customer(): Adds a new customer to the waiting list or seats them at an available table.
  - update_customers(): Updates the patience meters of all customers and handles haunt events.
  - draw_waiting_customers(): Draws waiting customers at the bottom of the screen.
  - near_kitchen(player): Checks if the player is near the kitchen.
  - near_table(player, table): Checks if the player is near a table.

- handle_spacebar(): Handles context-sensitive actions when the Spacebar is pressed.
- check_level_progress(): Checks if the player has achieved the minimum score to advance to the next level.
- draw():Draws all game objects on the screen.
- wait_for_enter(): Waits for the player to press Enter.
- save_statistics(): Saves the game statistics to a CSV file.
- running(): Runs the main game loop.
- toggle_pause(): Toggle the pause state of the game.

7**. Kitchen**

- **Role**: Represents the kitchen area where dishes are prepared.
- **Attributes**:
  - position (tuple): The (x, y) coordinates of the kitchen.
  - is_preparing (bool): Indicates whether a dish is being prepared.
  - preparation_time (int): The time required to prepare the dish.
  - preparation_start_time (Time): Time when dish preparation started.
  - current_dish (Dish): The dish currently being prepared.
- **Methods**:
  - start_preparation(): Starts preparing the dish.
  - is_dish_ready(paused): Checks if the dish is ready to be picked up.
  - draw(screen): Draws the kitchen and its current state on the screen.

8. **Config**

- **Role**: Keep the constants used in the game.
- **Class attributes** :
  - __ALL_CONFIGS (dict) : stores all constants (eg., screen width, height ,color, player size)
- **Class method :**
  - get() : retrieve a __ALL_CONFIGS dictionary value.

9**. AnimatedSprite**

- **Role**: handles the animation of sprites by loading and displaying a sequence of frames.
- **Attributes** :
  - Frames (list): Store the loaded frames of the animation.
  - frame_duration (int): The time (in milliseconds) to display each frame.
  - current_frame (int) : The index of the current frame being displayed.
  - last_update (Pygame): The time when the frame was last updated.
  - scale (tuple): Representing the scaling factor for the frames.
- **Method :**
  - update() : Updates the current frame based on the elapsed time.

    - draw(self, screen, position) :Draws the current frame on the screen at the specified position.

10. **UI**
    - **Role** : handles the user interface, including displaying the score, time, and level.
    - **Attributes**:
        - screen (Pygame): The Pygame screen object.
    - **Methods**:
        - draw_time(time_left): Draws the remaining time on the screen.
        - draw_score(score): Draws the player's score on the screen.
        - draw_level(level): Draws the current level on the screen.
        - draw_game_over(score): Displays the Game Over screen and waits for the player to press Enter.
        - draw_pause_button(paused) : Draws the pause button on the screen.
        - draw_level_complete_screen(level, score) : Draw the level completion screen with the level, score, and continue message.
11. **SoundEffect**
    - **Role** : handles the sound of the game
    - **Attributes**:
        - instance (SoundEffect): Stores the single instance of SoundEffect
        - sounds (dict) : Stores sound effects.
    - **Methods**:
        - get_instance(): Provides global access to the singleton instance.
        - generate_tone(duration, f, sample_rate): Returns a pygame.mixer.Sound object.
        - play_sound(effect): Called by other classes to play sounds.
12. **PhantomDinerApp**
    - **Role** : Main application window (root Tkinter container) that manages frame transitions.
    - **Attributes**:
        - container: A frame to hold other frames (like MainMenuFrame).
    - **Methods:**
        - show_frame(frame_class): Destroys current frame and loads the specified frame class (e.g., MainMenuFrame).

13. **MainMenuFrame**
    - **Role** : Main menu screen, Provides buttons to start game, view stats, manual, or quit.
    - **Attributes**:
        - original_image, bg_image: Background image handling.
        - bg_label: Label for background image
        - button_frame: Container for menu buttons.
        - controller: Reference to the root app (PhantomDinerApp).
    - **Methods**:
        - create_styles(): Configures button styles (colors, fonts).
        - create_widgets(): Adds buttons (Play, Statistics, Manual, Quit).

    - resize_image(event): Dynamically resizes the background image.
    - Navigation methods (start_game(), show_statistics(), show_manual()): Launch respective dialogs/frames.

14. **NameInputDialog**
    - **Role** : Modal dialog for player name input, Launches the Pygame game window, Bridges Tkinter and Pygame interfaces
    - **Attributes**:
        - name_entry: Input field for the player's name.
        - game_running: Flag to track if the game is active.
        - controller_alive: Checks if the main window exists.
    - **Methods**:
        - submit(): Validates the name and launches the game via launch_game().
        - launch_game(player_name): Initializes Pygame, runs the game loop, and handles cleanup.
        - safe_destroy(): Safely closes the dialog.
        - Handle_controller_destroy : Callback when main window is destroyed
15. **StatisticsFrame**
    - **Role** : Displays game statistics in tabbed view, shows player stats, overall stats, and graphs
    - **Attributes**:
        - stats_manager: Instance of StatisticsManager.
        - graph_generator: Instance of GraphGenerator.
        - notebook: Tabbed interface for stats/graphs.
    - **Methods**:
        - create_player_stats_tab(): Shows highest score/level and latest player.
        - create_numerical_stats_tab(): Displays statistical summaries (min, max, etc.).
        - create_graphs_tab(): Hosts graphs via GraphGenerator.

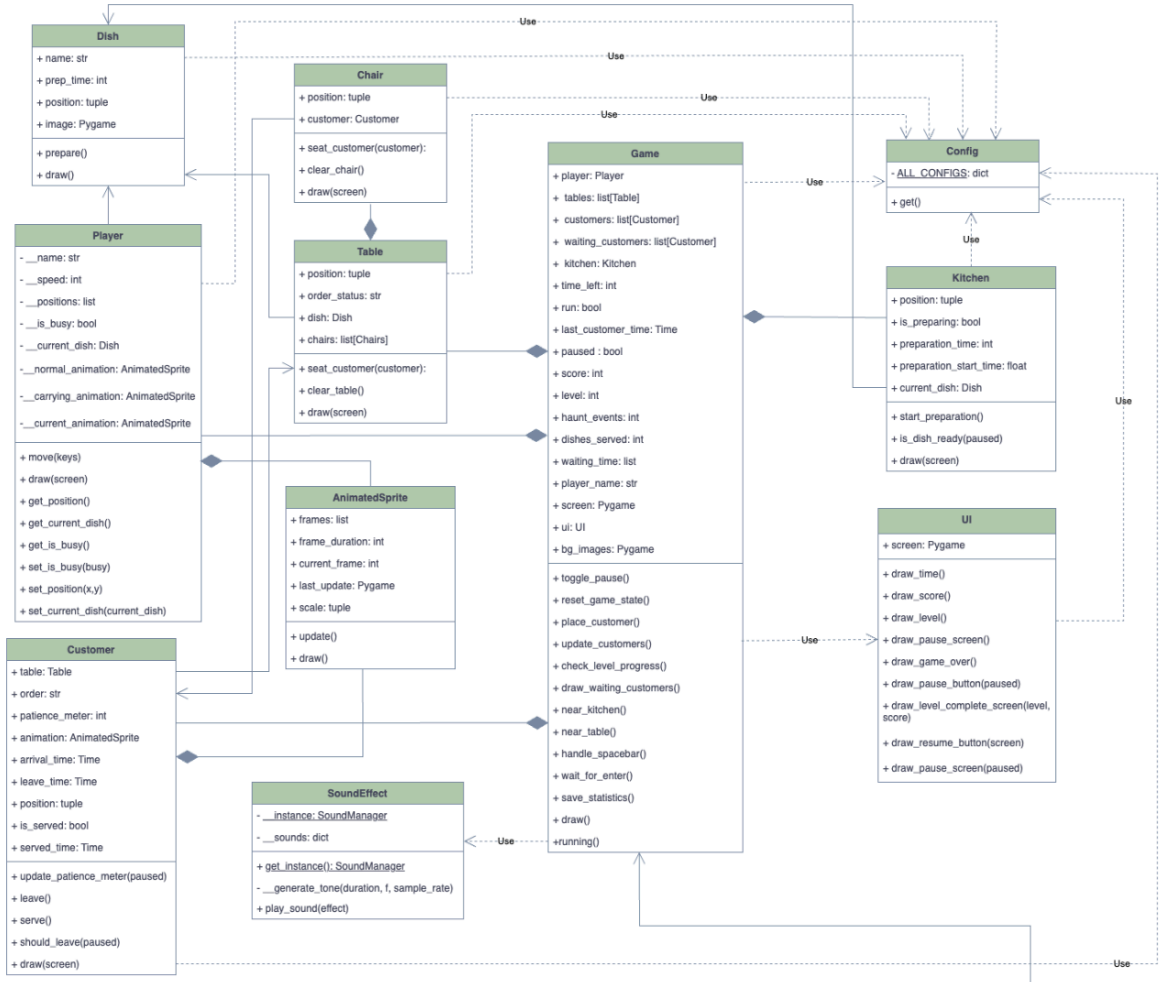
16. **StatisticsManager**
    - **Role** : Manages game statistics data, Loads CSV data, Calculates aggregates (max scores, averages, etc.)
    - **Attributes**:
        - __statistics_file: Path to the CSV file.
        - __statistics_df: Pandas DataFrame holding all stats.
    - **Methods**:
        - load_statistics(): Loads data from CSV into a DataFrame.
        - get_latest_player(): Returns the most recent player's stats.
        - get_highest_score(), get_highest_level(): Returns top performers.
        - get_numerical_stats(): Computes min/max/median/etc. for numerical features.

17. **GraphGenerator**
    - **Role** : Creates visualization plots, Handles boxplots, bar charts, scatter plots, and pie charts
    - **Attributes**:
        - stats_manager: Accesses stats data.
        - figures: Tracks active matplotlib figures for cleanup.
    - **Methods**:
        - remove_outlier(): Filters outliers using IQR.
        - Graph creation methods (create_score_boxplot(), create_dishes_graph(), etc.): Generate specific plots.
        - close_figures(): Prevents memory leaks by closing figures.

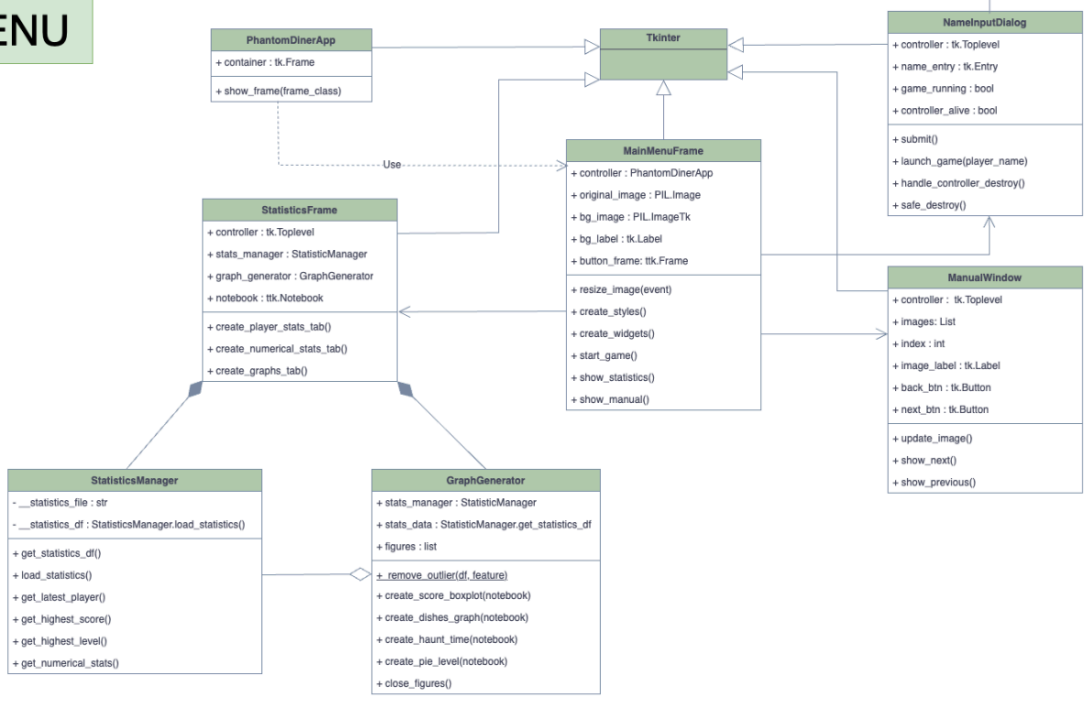
18. **ManualWindow**
    - **Role** : Displays a manual/slideshow with game instructions.
    - **Attributes**:
        - images: List of manual image paths.
        - index: Current image index.
        - controller: Reference to the main app.
    - **Methods**:
        - update_image(): Resizes and displays the current manual image.
        - show_next(), show_previous(): Navigate between manual pages.

Punnapa Pramotekul ID : 6610545863

GAME



MENU



3.3 Algorithms Involved

1. **Timer-Based Mechanics:**
    - Each level lasts 1 minute, with the timer increasing by 10 seconds every fourth level.
    - Customers have a patience meter that decreases over time.
2. **Randomization:**
    - Ghost customers arrive at random tables.
3. **Event-Driven Logic:**
    - Player actions (e.g., taking orders, serving dishes) trigger updates to the game state.
4. **Level Progression Mechanic:**
    - Players must score at least 100 points within the time limit to proceed to the next level.

4. Statistical Data (Prop Stats)

4.1 Data Features

1. **Score**:
    1.1. Tracks the player's score over time (per level).
2. **Waiting Time**:
    2.1. Records the waiting time of each customer (average per level).
3. **Haunt Events**:
    3.1. Number of times customers leave due to low patience (per level).
4. **Dishes Served**:
    4.1. Records the total number of dishes served (per level).
5. **Level**:
    5.1. Tracks the current level of the player.
6. **Player's Name**:
    6.1. The name of the player.

4.2 Data Recording Method

- Data will be stored in a CSV file.
- Example CSV structure:

Player,Score,Waiting Time,Haunt Events,Level,Dish Served

Player1,300,10.50,5,16,50

Player2,100,10,10,3,30

4.3 Data Analysis Report

1. Statistical Data Revision

Punnapa Pramotekul ID : 6610545863

Feature	**Why is it good to have this data?**	**How to obtain 50 values?**	**Variable (and Class)**	**How to display**
Score	The score reflects the player's performance and progression. It helps analyze how well the player is doing.	Collect the score when completing one level or quitting early.	score (from the Game class).	**Statistics Table :** mean, median, min, max and standard deviation **Graph :** Boxplot distribution per level
Waiting Time	Measures customer satisfaction and the player's efficiency. Helps identify bottlenecks in gameplay (e.g., long waiting times).	Record the waiting time for each customer and calculate the average per level.	waiting_times (from the Game class).	**Statistics Table**: mean, median, min, max and standard deviation. **Graph :** Scatter Plot VS. haunt events
Haunt Events	Tracks how often customers leave due to low patience. Indicates the player's ability to manage time and serve customers efficiently.	Count the number of haunt events per level.	haunt_events (from the Game class).	**Statistics Table**: mean, median, min, max and standard deviation. **Graph:** Scatter Plot VS. waiting time
Dishes Served	Measures the player's productivity and efficiency. Helps analyze how many dishes are served.	Record the cumulative number of dishes served per level.	dishes_served (from the Game class).	**Statistics:** mean, median, min, max and standard deviation. **Graph:** Bar graph of average dishes served per level
Level	Tracks the player's progression through the game. Helps analyze difficulty scaling and player skill.	Record the level reached by the player when completing one level or when quitting.	level (from the Game class).	**Statistics Table**: Calculate the highest level reached. **Graph**:

				Box-plot (score - level) Bar graph (Dish served - level) Pie - chart (Player-level)
Player's name	Personalizes the game experience and allows tracking of individual player performance.	Collect the player's name at the start of the game.	player_name (from the Game class)	**Statistics Table:** Display the player with the highest score and highest level.

Table

1. Display the player with highest score and score of current player

	Name	Score
Current player	Per	60
Highest player	Fah	190

2. Display the player with highest level and level of current player

	Name	Level
Current player	Per	60
Highest player	lml;	17

3. Display Statistical data of game

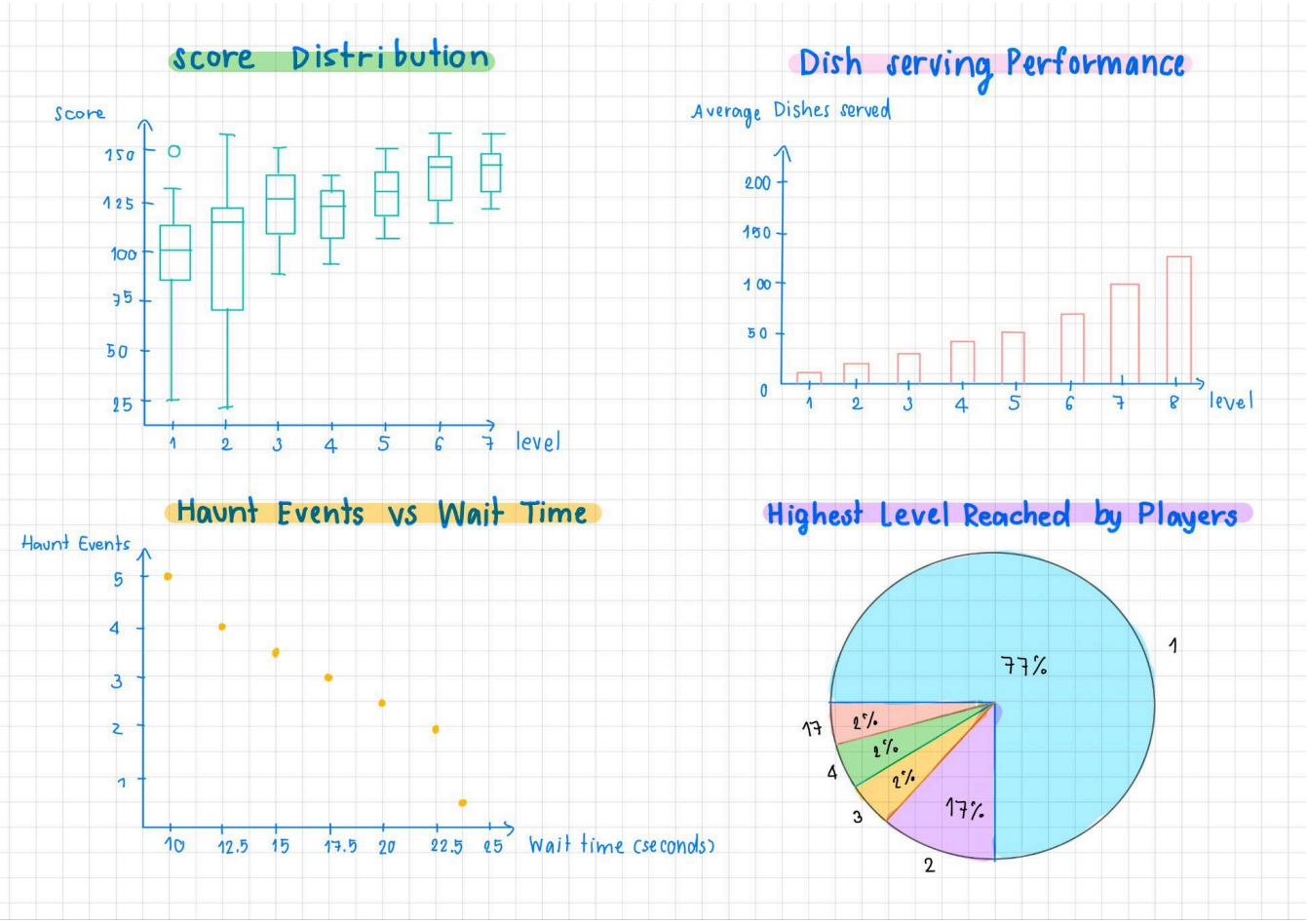
Feature	Statistical value
Score	Mean, Median, SD, Min, Max
Waiting time	Mean, Median, SD, Min, Max
Haunt Events	Mean, Median, SD, Min, Max
Dish served*	Mean, Median, SD, Min, Max

*Dish served is a cumulative total of dishes by level, so I add a temporal feature called "New dishes" to reveal actual number of dishes per-level for calculating mean, median, standard deviation, min, max and graph visualization.

Punnapa Pramotekul ID : 6610545863

Graph

	Feature Name	Graph Objective	Graph Type	X-axis	Y-axis
Boxplot of score	Score, level	Show score distribution across levels	Boxplot	Level	Score
Bar graph of dishes served per level	Dish served, level	Show dish volume across levels	Bar graph	Level	Average dishes per level
Scatter plot between Haunt events and waiting time	Haunt Events , waiting time	Analyze haunt events-waiting time relationship	Scatter Plot	Waiting time	Haunt events count
Highest Level Reached by Players pie chart	Level, Player	Compare the highest level reached across all players.	Pie Chart	(sections) Highest Level reached	compare to all players (%)



Punnapa Pramotekul ID : 6610545863

2. Planning submission

WEEK	GOALS
26 March - 2 April	● Complete all game logics ● Collect statistical data ● Create statistical table
3 April - 9 April	● Decorate game and develop visualization of project ● Implement 2 graphs and put in tkinter
10 April-16 April	● Implement all graphs and put in tkinter ● Develop projects ● Submit draft
17 April-23 April	● Revise project ● Complete game project ● Complete documentation
24 April-11 May	● Submit project

Punnapa Pramotekul ID : 6610545863

5. Project Timeline

Week	Task
1 (10 March)	Proposal submission / Project initiation
2 (17 March)	Full proposal submission
3 (24 March)	Add data collected table
4 (31 March)	Statistical Revision and planning submission
5 (7 April)	Create graph and statistical table
6 (14 April)	Submission week (Draft)

6. Document version

Version: 6.0
Date: 11 May 2025

7. Sources

- Images : I drew by myself and got inspiration from:
  - Player and Customer : Get inspired by Dead Pixels Ghost Club
  - Background : Get inspired by Diner Dash background
  - Resume button : From @sketchify in Canva.com
- Code: Get inspired by Diner Dash

8. The different from Proposal 4

- Features : Completing all key features and functionality.
- OOP : Adding more classes. The final version has 18 classes.
- Statistical data : Adjust the table by only showing Mean, Median, SD, Min, Max of score, Waiting time, Haunt Events and Dish served.