# hungman

*Release 1.0*

**Kacper Piszczek**

**May 20, 2025**

# CONTENTS:

Add your content using `reStructuredText` syntax. See the reStructuredText documentation for details.

CHAPTER

# ONE

# GAME LOGIC MODULE

`Game_Logic.are_there_players()` → bool

Checks if both players are registered or logged in.

**Returns:**
bool: True if both players are set, False otherwise.

`Game_Logic.check_time_over(`*gui:* HangmanGUI`)` → [<class 'bool'>, <class 'str'>]

Checks whether the game timer has run out.

**Args:**
gui (GUI.HangmanGUI): The GUI instance to call game-end behavior.

**Returns:**
list[bool, str]: [True, remaining_time_str] if time is up, else [False, remaining_time_str].

`Game_Logic.clear(`*gui:* HangmanGUI`)`

Resets the game state and clears player information.

**Args:**
gui (GUI.HangmanGUI): The GUI instance to reset interface state.

`Game_Logic.export_player1()`

Exports player 1's game result and statistics to the database.

`Game_Logic.export_player2()`

Exports player 2's game result and statistics to the database.

`Game_Logic.get_categories()` → list[str]

Fetches the list of word categories including 'random'.

**Returns:**
list[str]: A list of category names.

`Game_Logic.get_guessed_words()` → str

Returns the string of guessed letters or words so far.

**Returns:**
str: A comma-separated string of guessed entries.

`Game_Logic.get_statistics(`*player: int*`)` → str

Retrieves and formats the statistics for the specified player.

**Args:**
player (int): Player number (1 or 2).

**Returns:**
str: A formatted string of player statistics.

`Game_Logic.`**`get_word`**`()` → str

Formats and returns the current masked word with line breaks.

**Returns:**
str: The formatted masked word with line breaks every 10 characters.

`Game_Logic.`**`is_player_defined`**`(`*player: int*`)` → bool

Checks if a player is defined (registered or logged in).

**Args:**
player (int): Player number (1 or 2).

**Returns:**
bool: True if the specified player is set, False otherwise.

`Game_Logic.`**`login_player1`**`(`*name: str*, *password: str*, *gui:* HangmanGUI`)`

Logs in player 1 using the provided credentials.

**Args:**
name (str): Username. password (str): Password. gui (GUI.HangmanGUI): The GUI instance to update the interface.

`Game_Logic.`**`login_player2`**`(`*name: str*, *password: str*, *gui:* HangmanGUI`)`

Logs in player 2 using the provided credentials.

**Args:**
name (str): Username. password (str): Password. gui (GUI.HangmanGUI): The GUI instance to update the interface.

`Game_Logic.`**`on_submit`**`(`*entry: str*, *gui:* HangmanGUI`)`

Handles user input during standard mode and updates the game state.

**Args:**
entry (str): The guessed letter or word. gui (GUI.HangmanGUI): The GUI instance to update the interface.

`Game_Logic.`**`register_player1`**`(`*name: str*, *password: str*, *gui:* HangmanGUI`)`

Registers player 1 using the provided credentials.

**Args:**
name (str): Username. password (str): Password. gui (GUI.HangmanGUI): The GUI instance to update the interface.

`Game_Logic.`**`register_player2`**`(`*name: str*, *password: str*, *gui:* HangmanGUI`)`

Registers player 2 using the provided credentials.

**Args:**
name (str): Username. password (str): Password. gui (GUI.HangmanGUI): The GUI instance to update the interface.

`Game_Logic.`**`set_timer`**`(`*minutes: int = 2*, *seconds: int = 0*`)`

Sets the countdown timer for the game.

**Args:**
minutes (int): Minutes to set. Defaults to 2. seconds (int): Seconds to set. Defaults to 0.

`Game_Logic.`**`set_word_number`**`(`*number: str*`)`

Sets how many words will be used in special mode.

**Args:**
number (str): The number of words as a string.

`Game_Logic.`**`setup_special_mode`**`(`*gui:* HangmanGUI, *category: str*`)`

Initializes the game in special mode with multiple words.

**Args:**
gui (GUI.HangmanGUI): The GUI instance to update the interface. category (str): The category to select words from. Can be 'random'.

`Game_Logic.`**`setup_standard_mode`**`(`*gui:* HangmanGUI, *category: str*`)`

Initializes the game in standard mode with a word from a given category.

**Args:**
gui (GUI.HangmanGUI): The GUI instance to update the interface. category (str): The category to select the word from. Can be 'random'.

`Game_Logic.`**`special_on_submit`**`(`*entry: str*, *gui:* HangmanGUI`)`

Handles user input during special mode and updates the game state.

**Args:**
entry (str): The guessed letter or word. gui (GUI.HangmanGUI): The GUI instance to update the interface.

`Game_Logic.`**`update_statistics`**`(`*name: str*, *hits: int = 0*, *misses: int = 0*, *wins: int = 0*, *losses: int = 0*`)`

Updates the player statistics in the database.

**Args:**
name (str): Player name. hits (int): Number of hits to add. Defaults to 0. misses (int): Number of misses to add. Defaults to 0. wins (int): Number of wins to add. Defaults to 0. losses (int): Number of losses to add. Defaults to 0.

`Game_Logic.`**`update_word_state`**`(`*w*, *c_w_s*, *e*`)`

Updates the masked word state based on a correct guess.

**Args:**
w (str): The full word or phrase. c_w_s (str): Current masked word state. e (str): The guessed letter or word.

**Returns:**
str: Updated masked word state.

CHAPTER

TWO

# GUI MODULE

class GUI.**HangmanGUI**

Bases: `object`

**`clear_statistics`**()

Clears the displayed statistics from the statistics frame.

**`end`**(*result: bool*)

Ends the game, resets the interface, and shows the result screen.

**Args:**
result (bool): True if the game was won, False if lost.

**`next_image`**()

Updates the hangman image for standard mode to the next stage.

**Returns:**
int: The current image counter value.

**`on_select`**(*event*)

Handles category selection from the dropdown.

**Args:**
event: The event object from the combobox selection.

**`player1_logged_in`**()

Updates the UI to indicate that player 1 is logged in.

**`player1_not_logged_in`**()

Updates the UI to indicate that player 1 is not logged in.

**`player2_logged_in`**()

Updates the UI to indicate that player 2 is logged in.

**`player2_not_logged_in`**()

Updates the UI to indicate that player 2 is not logged in.

**`repeated_over_time_code`**()

Updates the special mode timer display and schedules itself to repeat every second.

**`set_statistics_frame`**(*player: int*)

Displays statistics for the specified player.

**Args:**
player (int): Player number (1 or 2).

`set_time`(*minutes: str*, *seconds: str*)

Sets the game timer using the provided minutes and seconds.

**Args:**
minutes (int): Minutes to set. seconds (int): Seconds to set.

`show_frame`(*frame_to_show*)

Displays the specified frame in the UI, ensuring proper setup for game modes.

**Args:**
frame_to_show: The tkinter Frame object to be shown.

`special_next_image`()

Updates the hangman image for special mode to the next stage.

**Returns:**
int: The current special image counter value.

`special_update_word`()

Updates the displayed word and guessed letters in special mode.

`start`()

Starts the tkinter main event loop and shows the main menu.

`update_word`()

Updates the displayed word and guessed letters in standard mode.

CHAPTER

THREE

# DATABASE LOGIC MODULE

`Database_Logic.decrypt`(*text: str*) → str

Decrypts a string that was encrypted with the *encrypt* function, shifting letters and digits backward by 1.

**Parameters**
`text` – The encrypted string to decrypt.

**Returns**
The original decrypted string.

`Database_Logic.encrypt`(*text: str*) → str

Encrypts a string by shifting letters and digits forward by 1. Wraps around alphabetically and numerically.

**Parameters**
`text` – The input string to encrypt.

**Returns**
The encrypted string.

`Database_Logic.export`(*name: str*, *text: str*) → bool

Exports a given text to a file in the 'exports/' directory.

**Parameters**

- `name` – The name of the export file (without extension).
- `text` – The content to write into the file.

**Returns**
True if export was successful.

`Database_Logic.get_categories`() → list[str]

Returns a list of all available word categories from 'Categories.txt'.

**Returns**
A list of category names.

`Database_Logic.get_random_word`() → str

Returns a random word from a randomly selected category file listed in 'Categories.txt'.

**Returns**
A random word, or "Error" if something fails.

`Database_Logic.get_word_from_category`(*category: str*) → str

Returns a random word from a given category file.

**Parameters**
`category` – The name of the category file (without extension).

**Returns**
A random word from the category, or "Error" if it fails.

`Database_Logic.`**`login`**`(`*name: str*, *password: str*`)` → bool

Validates a player's login credentials.

**Parameters**

- **`name`** – The username of the player.
- **`password`** – The password to check.

**Returns**
True if login is successful, False otherwise.

`Database_Logic.`**`read_statistics`**`(`*name: str*`)` → str

Reads and decrypts the game statistics of a player.

**Parameters**
**`name`** – The username of the player.

**Returns**
A decrypted statistics string, or default stats if none exist.

`Database_Logic.`**`register`**`(`*name: str*, *password: str*`)` → bool

Registers a new player by encrypting and storing their password.

**Parameters**

- **`name`** – The username of the player.
- **`password`** – The password of the player.

**Returns**
True if registration succeeds, False if the user already exists.

`Database_Logic.`**`write_statistics`**`(`*name: str*, *statistics: str*`)` → bool

Writes encrypted game statistics for a player.

**Parameters**

- **`name`** – The username of the player.
- **`statistics`** – A string containing player statistics to store.

**Returns**
True if write was successful.

# PYTHON MODULE INDEX