Module: player
==============

Class: Player

    A class to represent the player character in the game.

    Attributes:

    GRAVITY : int
        The gravitational force applied to the player.
    SPRITES : dict
        A dictionary containing the player's sprite sheets.
    ANIMATION_DELAY : int
        The delay between animation frames.
    rect : pygame.Rect
        The rectangle representing the player's position and dimensions
on screen.
    x_vel : int
        The velocity of the player along the x-axis.
    y_vel : int
        The velocity of the player along the y-axis.
    direction : str
        The direction the player is facing, either "Left" or "Right".
Default is "Right".
    animation_count : int
        Counter for the current frame in the animation sequence.
    fall_count : int
        Counter for tracking how long the player has been falling.
    hit : bool
        Flag indicating whether the player has been hit or not.
    hit_count : int
        Counter for how long the player has been in the hit state.
    jump_count : int
        Counter for the number of jumps the player has performed.
    dead : int
        Counter for tracking the player's death state.
    sprite : pygame.Surface or None
        The current sprite image of the player. None if no sprite is
assigned.

    Methods:
    ----
    move(dx: int, dy: int) -> None:
        Moves the player by the specified amounts.
    make_hit() -> None:
        Sets the player to the hit state.
    move_left(vel: int) -> None:
        Moves the player to the left with the specified velocity.
    move_right(vel: int) -> None:
        Moves the player to the right with the specified velocity.
    jump() -> None:
        Makes the player jump.

```
    update() -> None:
        Updates the player's rectangle based on the current sprite.
    loop(fps: int) -> None:
        Updates the player's position and state for each frame.
    landed() -> None:
        Resets the player's fall count and vertical velocity when they
land.
    hit_head() -> None:
        Inverts the player's vertical velocity when they hit their head.
    update_sprite() -> None:
        Updates the player's sprite based on their current state and
direction.
    draw(win: pygame.Surface, offset_x: int) -> None:
        Draws the player on the given window with the specified
horizontal offset.


Module: object
==============
Class: Object


    A class to represent a generic object (physical entity) in the game.

    Attributes:
    -----------
    rect : pygame.Rect
        The rectangle representing the object's position and size.
    image : pygame.Surface
        The surface for the object's image.
    width : int
        The width of the object.
    height : int
        The height of the object.
    name : str
        The name of the object.
    path : str
        The path to the object's image file.
    topleft : tuple
        The top-left coordinate for cropping the image.
    scale : tuple
        The scale to which the object image should be resized.

    Methods:
    --------
    draw(win: pygame.Surface, offset_x: int) -> None:
        Draws the object on the given window with the specified
horizontal offset.
    get_block(size: int, topleft: tuple) -> pygame.Surface:
        Crops the object's image from the specified top-left corner and
scales it.

Class: Block
```

A class to represent a block object in the game (such as terrain, lives, start checkpoint, end checkpoint,...), inheriting from Object.

Methods:
---
draw_block() -> None:
    Draws the block image on the object's surface.

Class: Fire

A class to represent a fire object in the game, inheriting from Object.

Attributes:

ANIMATION_DELAY : int
    The delay between animation frames.
fire : dict
    A dictionary containing the fire's sprite sheets.
animation_count : int
    The current frame count for animation.
animation_name : str
    The current animation state ("on" or "off").

Methods:
---
on() -> None:
    Sets the fire animation to "on".
off() -> None:
    Sets the fire animation to "off".
loop() -> None:
    Updates the fire's animation based on the current state.

Module: map
==========

Class: Map

A class to represent a game map composed of different objects such as terrain, fire, and checkpoints.

Attributes:
---
block_size : int
    The size of each block in the map.
map_data : list
    A list of lists containing the map data for each level.

```
    levels : list
        A list to keep track of the loaded levels.
    up_level : int
        The current level index.
    objects : list
        A list to store Block objects present in the current level.
    fire : list
        A list to store Fire objects present in the current level.

    Methods:
    --------
    load_level(level: int) -> None:
        Loads the specified level, initializing the objects and fire
lists.

    next_level() -> None:
        Advances to the next level and loads it.

    draw(window, offset_x) -> None:
        Draws the objects and fire on the given window with the specified
offset.


Module: menu
============

Class: Menu


    A class to represent different game menus (start, win, lose).

    Attributes:
    -----------
    menu_type : str
        The type of menu ("start", "win", "lose").
    font : pygame.font.Font
        The font used for the main title text.
    small_font : pygame.font.Font
        The font used for the smaller option text.
    title_text : pygame.Surface
        The surface for the main title text.
    quit_text : pygame.Surface, optional
        The surface for the quit text (only for "start" menu).
    option_text : pygame.Surface, optional
        The surface for the option text (only for "win" and "lose"
menus).
    title_rect : pygame.Rect
        The rectangle for positioning the title text.
    start_rect : pygame.Rect, optional
        The rectangle for positioning the start text (only for "start"
menu).
    quit_rect : pygame.Rect, optional
        The rectangle for positioning the quit text (only for "start"
menu).
```

```
    option_rect : pygame.Rect, optional
        The rectangle for positioning the option text (only for "win" and
"lose" menus).

    Methods:
    --------
    draw(window: pygame.Surface) -> None:
        Draws the menu on the given window.
    handle_event(event: pygame.event.Event) -> str or None:
        Handles events (mouse clicks and key presses) and returns the
menu action.


Module: main_game
=================

Class: MAIN_GAME


    A class to represent the main game logic and control flow.

    Attributes:
    -----------
    player : Player
        The player character.
    block_size : int
        The size of each block in the game.
    map_data : list
        The map data for different levels.
    map : Map
        The map object containing the game levels.
    name_background : list
        List of background image filenames for each level.
    platform_count : int
        Counter for platforms reached by the player.
    reset : bool
        Flag to reset the game state.
    lives : list
        List of Block objects representing the player's lives.
    offset_x : int
        Offset for horizontal scrolling.
    scroll_area_width : int
        Width of the scroll area to trigger scrolling.
    menu : Menu
        The menu object for displaying menus.
    in_menu : bool
        Flag indicating if the game is currently displaying a menu.
    game_over : bool
        Flag indicating if the game is over.

    Methods:
    --------
    loop(FPS: int) -> None:
        Main game loop for updating player and fire animations.
```

```
    collide(dx: int) -> Block or None:
        Checks for horizontal collisions and returns the collided object.
    handle_vertical_collision(dy: int) -> list:
        Handles vertical collisions and returns a list of collided
objects.
    operate_game() -> None:
        Handles player movements, collisions, and game state transitions.
    reset_game() -> None:
        Resets the game state to the beginning of the level.
    draw(window: pygame.Surface) -> None:
        Draws the game elements on the given window.
    scroll() -> None:
        Handles horizontal scrolling based on player movement.
    end_game(win: bool = False, lose: bool = False) -> None:
        Ends the game and shows the appropriate menu.
    handle_events() -> None:
        Handles pygame events such as keyboard and mouse inputs.

Module: utils
=============
    A supporting module includes importing pygame, pygame window's and
other basic setting as well as methods allowing flip/load sprite
sheet/get background used by other class which has graphic features
```