Zombie Shooter

Complete Project Documentation

Project Name: Zombie Shooter
Language: Python 3.x
Framework: Pygame
Total Files: 14 Python modules
Total Lines of Code: ~11,500+ lines

Project Overview

Zombie Shooter is a comprehensive 2D top-down survival game developed using Python and Pygame. The game features both single-player and multiplayer modes, where players must survive waves of zombies while progressing through increasingly difficult levels.

The project demonstrates advanced game programming concepts including: - Network programming for multiplayer - Particle systems for visual effects - AI pathfinding for enemies - Modular architecture design

---

Key Features

Game Modes

- **Single-Player Campaign:** 6 progressive levels with increasing difficulty
- **Multiplayer Co-op:** 2-player cooperative mode via local network or Hamachi VPN

Character System

- **Classic Agent:** Standard character with balanced abilities
- **Commando (Soldier):** Special character with unique abilities
  - Dash: Quick movement ability with cooldown
  - Shield: Temporary damage protection

Weapon System

- **Pistol:** Unlimited ammo, fast fire rate, single bullet
- **Shotgun:** 5 pellets spread pattern, limited ammo
- **Grenade:** Explosive area damage with particle effects

Skin System

- 5 unique character skins: Soldier, Crimson, Ranger, Champion, Mystic
- Network-synchronized skin selection for multiplayer

Game Elements

- Speed Crates: Temporary speed boost pickups
- Health Packs: Medical kit pickups for healing
- Ammo Boxes: Ammunition pickups for weapons
- Level Doors: Navigation system between levels

Visual Effects

- Blood Particles: Dynamic blood splatter effects
- Explosion Effects: Multi-layered explosion animations
- Camera Shake: Screen shake on explosions
- Horror-themed UI: Dark, atmospheric menu design

Multiplayer Features

- Real-time Chat System
- Minimap display
- Full game state synchronization
- Hamachi IP auto-detection

Persistence

- Settings System: Saved volume, resolution, key bindings
- Leaderboard: High score tracking

Project Structure

```
python_learnfullversion/

  main.py             - Entry point and game flow
  game.py             - Single-player game logic
  multiplayer_game.py - Multiplayer game logic
  characters.py       - Player and Enemy classes
  weapons.py          - Weapon system
  network.py          - Network communication
  skins.py            - Character appearances
  settings.py         - Game settings
  chat.py             - Multiplayer chat
  minimap.py          - Mini-map display
  leaderboard.py      - High scores
```

```
walls.py             - Level wall generation
util.py              - Utilities and UI
multiplayer_setup.py - Multiplayer setup (legacy)
images/              - Game sprites
sounds/              - Sound effects
```

File Documentation

---

1. main.py

Lines: 434
Purpose: Application entry point and game flow controller

Functions

get_hamachi_ip() - Parameters: None - Returns: String (IP address) - Description: Automatically detects Hamachi VPN IP address using ipconfig command parsing for easy multiplayer setup.

get_local_ip() - Parameters: None - Returns: String (IP address) - Description: Gets the local network IP address using socket connection.

multiplayer_menu(screen, clock, version) - Parameters: screen (pygame.Surface), clock (pygame.time.Clock), version (str) - Returns: Tuple or String - Description: Displays the multiplayer setup screen with Host/Join options, IP input, and skin selection.

main() - Parameters: None - Returns: None - Description: Main game loop managing game states (menu, single-player, multiplayer, leaderboard).

show_character_select(screen, clock) - Parameters: screen (pygame.Surface), clock (pygame.time.Clock) - Returns: String ("player" or "commando") - Description: Displays character selection screen before single-player mode.

---

2. game.py

Lines: 2,624
Purpose: Complete single-player game implementation

Classes

Camera Class Handles viewport management and screen shake effects.

init(world_w, world_h, view_w, view_h) - Initializes camera with world and viewport dimensions

follow(target_rect, lerp=0.15) - Smoothly follows a target using linear interpolation

trigger_shake(intensity=12.0) - Initiates camera shake effect for explosions

update(dt) - Updates shake effect decay over time

apply_rect(rect) - Converts world coordinates to screen coordinates

apply_xy(x, y) - Converts world position to screen position

MenuBackground Class Creates animated horror-themed menu background.

init(screen) - Creates layered background with fog, ghosts, and lightning

_build_skyline() - Generates tombstone/ruins silhouettes

_make_fog(alpha) - Creates blood-red fog effect layer

_spawn_ghosts(n) - Spawns floating ghost entities

_create_blood_spots() - Generates random blood splatter positions

draw(screen, dt) - Renders complete animated background with all effects

---

SpeedCrate Class Speed boost pickup with open/close animation.

init(x, y) - Creates crate at specified position with closed/opened sprites

rect (property) - Returns collision rectangle

trigger_open(show_time=0.35) - Initiates opening animation

update(dt) - Updates opening timer and removes when done

draw(screen, cam) - Renders crate with current state

LevelDoor Class Level transition door with navigation system.

init(x, y, level) - Creates door with glow effects and beacon system

activate() - Activates door when kill requirement is met

update(dt, player_x, player_y) - Updates glow animation and beacon particles

draw_navigation(screen, player_x, player_y) - Draws directional arrow pointing to door when off-screen

draw(screen, cam) - Renders door with glow effects and level indicator

---

Zombie Class Enemy AI with pathfinding and level scaling.

init(x, y, level) - Creates zombie with level-scaled speed, HP, and damage

_pick_waypoint(W, H, walls) - Selects random patrol waypoint avoiding walls

_slide_move(dx, dy, walls) - Moves with wall collision sliding

update(player_pos, walls, dt) - Updates AI behavior with raycast vision and pathfinding

draw(screen, cam) - Renders zombie with bobbing animation and health bar

---

BloodParticle Class Blood splatter particle effect.

init(x, y) - Creates particle with random velocity and lifetime

update(dt) - Updates position with velocity decay

draw(screen, cam) - Renders fading blood particle

Pickup Class Health and ammo pickup items.

init(x, y, kind) - Creates pickup of type: "medkit", "shotgun_ammo", "grenade_ammo"

draw(screen, cam) - Renders 3D-styled pickup with floating animation

---

VictoryScene Class Victory screen with statistics display.

init(screen, score, kills, levels) - Creates victory display with stats

draw_background() - Renders animated victory background

draw_stats() - Displays score, kills, and levels completed

handle_event(event) - Handles restart/menu/quit input

GameOverScene Class Game over screen with retry options.

init(screen, score, kills, level) - Creates game over display

draw() - Renders game over screen with options

handle_event(event) - Handles restart/menu/quit input

---

game.py Standalone Functions

main_menu(screen, clock, version) - Displays main menu with all navigation options

_draw_howto_panel(screen) - Renders "How to Play" instructions panel

_draw_skins_panel(screen, cx, cy) - Renders skin selection panel in menu

_draw_settings_panel(screen, cx, cy, music_slider, sfx_slider, back_btn) - Renders settings panel with volume sliders

normalized(x, y) - Returns normalized direction vector

calculate_door_direction(player_x, player_y, door_x, door_y) - Calculates angle and distance to door

create_navigation_arrow(angle, size) - Creates rotated arrow surface for navigation

create_distance_indicator(distance) - Returns distance text and color based on proximity

generate_background_effects(level, effects_dict, W, H) - Pre-renders level background with effects

draw_level_background(screen, level, cam, effects_dict) - Draws pre-rendered level background

raycast_clear(a, b, walls, step) - Checks line-of-sight between two points

get_muzzle_xy(player, target_x, target_y) - Calculates weapon muzzle position based on aim direction

find_free_spawn(walls, W, H, w, h) - Finds valid spawn position avoiding walls

far_from_player(px, py, x, y, min_dist) - Checks if position is far enough from player

find_door_location(walls, player_x, player_y, W, H) - Finds valid door placement location

run_game(screen, clock, version, character) - Main single-player game loop

run_demo_level(screen, clock, version) - Compatibility wrapper for run_game

---

3. multiplayer_game.py

Lines: 3,035
Purpose: Network-synchronized multiplayer game

Additional Classes

MultiplayerVictoryScene Class Multiplayer-specific victory screen showing both players' stats.

init(screen, kills_by_player, score_by_player, player_id) - Creates victory display for multiplayer

draw_player_stats() - Renders statistics for both players

MultiplayerGameOverScene Class Multiplayer game over with spectator mode.

init(screen, player_id, is_spectator) - Creates game over for multiplayer

draw_spectator_mode() - Renders spectator overlay when watching partner

---

Key Functions

get_current_skin() - Gets selected skin from game.py module

run_multiplayer_game(screen, clock, network, player_id, version, skin_id, character_type) - Main multiplayer game loop with network synchronization

---

4. characters.py

Lines: 400
Purpose: Character definitions and behaviors

Player Class (dataclass)

Main player character with movement and abilities.

post_init() - Loads sprites and initializes character based on type

_prepare_sprite(surf, target_h) - Crops transparent edges and scales sprite

_load_image_candidates(names) - Attempts to load first available image from list

_load_commando_direction(direction) - Loads commando-specific directional sprites

_apply_skin_tint() - Applies skin color overlay to all sprites

set_skin(color) - Changes character skin color

rect (property) - Returns collision rectangle

move_try(dx, dy, walls, W, H) - Attempts movement with collision detection

draw(screen, use_skin) - Renders character with current facing direction

_draw_commando_effects(screen) - Draws special effects for commando (dash trail, shield)

update_abilities(dt) - Updates ability cooldowns and active effects

activate_dash(dx, dy) - Activates dash ability in specified direction

activate_shield() - Activates temporary shield protection

is_shielded() - Returns True if shield is currently active

get_ability_status() - Returns ability cooldowns for HUD display

---

Enemy Class (dataclass)

Basic enemy for single-player mode.

post_init() - Loads enemy sprite

rect (property) - Returns collision rectangle

_move_axis(dx, dy, walls) - Moves on single axis with collision

update_seek(target, walls, dt) - Updates movement toward target position

draw(screen) - Renders enemy sprite

---

5. weapons.py

Lines: 814
Purpose: Complete weapon management

WeaponType Enum

```
PISTOL = 1   (Default weapon, unlimited ammo)
SHOTGUN = 2  (Spread weapon, limited ammo)
GRENADE = 3  (Explosive, area damage)
```

WeaponBullet Class (dataclass)

Individual bullet/projectile.

update(dt) - Updates position, handles grenade fuse timer

rect (property) - Returns collision rectangle

draw(screen, cam_offset) - Renders bullet with type-specific visuals

---

ExplosionEffect Class

Multi-layered explosion animation.

init(x, y, radius) - Creates explosion with particles and shockwave

_create_all_particles() - Generates fire, smoke, spark, and debris particles

update(dt) - Updates all particle positions and lifetimes

draw(screen, cam_offset) - Renders complete explosion with all effects

---

WeaponManager Class

Central weapon control system.

init(player_id) - Initializes with pistol, sets ammo counts

switch_weapon(weapon_type) - Switches to specified weapon if ammo available

switch_by_key(key) - Switches weapon by keyboard key (1, 2, 3)

can_fire() - Checks if weapon can fire (cooldown and ammo)

fire(start_x, start_y, target_x, target_y) - Creates bullets, handles shotgun spread

add_ammo(weapon_type, amount) - Adds ammunition to weapon

update(dt) - Updates all bullets and explosions

draw_bullets(screen, cam_offset) - Renders all active bullets

draw_explosions(screen, cam_offset) - Renders all active explosions

draw_hud(screen, x, y) - Draws weapon/ammo HUD element

to_dict() - Serializes for network transmission

from_dict(data, player_id) - Deserializes from network data

---

6. network.py

Lines: 412
Purpose: Multiplayer network communication

NetworkManager Class

init() - Initializes socket, queues, and connection state

start_server(player_name) - Starts TCP server on port 5555, waits for client

connect_to_server(server_ip, player_name) - Connects to existing server as client

_server_listen() - Thread: Accepts incoming connection, receives data

_client_receive() - Thread: Continuously receives data from server

_receive_data(sock) - Receives data with proper message framing

_send_data(data) - Sends data with length prefix framing

send_game_state(game_state) - Sends complete game state (zombies, crates, pickups)

send_player_data(player_data) - Sends player position and state

send_player_action(action) - Sends player action (shoot, pickup, etc.)

send_start_game() - Host sends game start signal

send_chat_message(message, player_name) - Sends chat message to other player

send_skin_data(skin_id) - Sends selected skin for synchronization

send_character_type(character_type) - Sends character type (player/commando)

send_weapon_data(weapon_type, ammo) - Sends current weapon state

get_received_data() - Thread-safe retrieval of received messages

disconnect() - Closes connection and cleans up resources

---

7. skins.py

Lines: 228
Purpose: Character appearance system

Skin Definitions

```
soldier:  RGB(100, 150, 200) - Blue, Default military
crimson:  RGB(200, 60, 60)   - Red combat suit
ranger:   RGB(60, 180, 60)   - Green forest outfit
champion: RGB(255, 200, 50)  - Golden armor
mystic:   RGB(150, 60, 200)  - Purple cloak
```

Functions

get_skin_names() - Returns list of all skin IDs

get_skin_data(skin_id) - Returns skin data dictionary

get_skin_color(skin_id) - Returns RGB color tuple for skin

get_next_skin(current_skin) - Returns next skin in cycle

get_prev_skin(current_skin) - Returns previous skin in cycle

apply_skin_tint(surface, skin_id, intensity) - Applies color tint to sprite surface

draw_skin_preview(screen, x, y, skin_id, size, selected) - Renders skin preview circle

draw_skin_selector(screen, current_skin, center_x, y) - Renders complete skin selection UI

get_clicked_skin(click_pos, center_x, y) - Returns clicked skin ID or None

draw_player_indicator(screen, x, y, skin_id, player_name, is_local) - Draws colored indicator above player

---

8. settings.py

Lines: 154
Purpose: Persistent settings management

GameSettings Class (Singleton)

new(cls) - Singleton pattern - returns single instance

init() - Initializes with defaults, loads saved settings

save() - Saves settings to settings.json

load() - Loads settings from settings.json

set_music_volume(volume) - Sets music volume (0.0-1.0), applies immediately

set_sfx_volume(volume) - Sets sound effects volume (0.0-1.0)

set_resolution(width, height) - Sets game window resolution

get_resolution_index() - Returns current resolution index in preset list

apply_resolution(screen) - Applies resolution and returns new screen surface

is_key_for_action(key, action) - Checks if key is bound to action

get_key_name(action) - Returns display name for key binding

---

9. chat.py

Lines: 301
Purpose: Text chat for multiplayer

ChatMessage Class (dataclass)

get_age() - Returns message age in seconds

is_expired() - Returns True if message has expired

get_alpha() - Returns fade alpha based on age

to_dict() - Serializes for network

from_dict(data) - Deserializes from network data

ChatSystem Class

init(screen_width, screen_height, player_id, player_name) - Initializes chat with input box and message list

set_player_info(player_id, player_name) - Updates player information

get_player_color(player_id) - Returns color for player messages

add_message(message) - Adds message to display list

add_system_message(content) - Adds system notification

send_message(content) - Creates and returns message for network

receive_message(data) - Processes received network message

get_pending_messages() - Returns messages waiting to be sent

toggle_input() - Opens/closes chat input box

handle_event(event) - Handles keyboard input for typing

update(dt) - Updates message expiration and cursor

is_typing() - Returns True if input is active

draw(screen) - Renders chat UI with messages

draw_indicator(screen, x, y) - Draws typing indicator above player

---

10. minimap.py

Lines: 224
Purpose: Game world overview display

Minimap Class

init(world_w, world_h, screen_w, screen_h, size) - Initializes minimap with scale calculations

toggle() - Toggles minimap visibility

handle_event(event) - Handles M key to toggle visibility

world_to_map(world_x, world_y) - Converts world coordinates to minimap position

set_walls(walls) - Caches wall rectangles for rendering

_render_walls(walls) - Pre-renders walls to surface

draw(screen, player_pos, other_players, zombies, door, walls) - Renders complete minimap with all entities

draw_simple(screen, player_x, player_y, level_no) - Renders simplified minimap for single-player

---

11. leaderboard.py

Lines: 430
Purpose: Score tracking and display

ScoreEntry Class (dataclass)

```
Fields:
- player_name: str (Player's name)
- score: int (Total score)
- kills: int (Total zombie kills)
- level: int (Level reached)
- date: str (Date achieved)
```

LeaderboardManager Class

init(file__path) - Initializes and loads saved scores

load__scores() - Loads scores from JSON file

save__scores() - Saves scores to JSON file

add__score(player__name, score, kills, level) - Adds new score, returns rank or -1

get__top__scores(count) - Returns top N scores

is__high__score(score) - Checks if score qualifies for leaderboard

get__rank(score) - Returns expected rank for score

LeaderboardScreen Class

init(screen, manager) - Initializes display with particle effects

_create_particles() - Creates background particle animation

update(dt) - Updates particle positions

draw(highlight__rank) - Renders leaderboard with optional highlight

---

NameInputDialog Class

init(screen, score, kills, level) - Creates input dialog

update(dt) - Updates cursor blink

handle__event(event) - Handles text input, returns name on Enter

draw() - Renders input dialog

Functions

show_leaderboard(screen, clock, highlight_rank) - Shows leaderboard screen, returns True to exit

---

12. walls.py

Lines: 134
Purpose: Wall generation for each level

Functions

collide_rect_list(rect, rects) - Returns first colliding rect or None

draw_walls(screen, walls) - Renders all walls with 3D effect

_rp(W, H, x, y, w, h) - Helper: Creates rect from proportional values

_level1(W, H) - Creates Level 1 walls: simple corners and center

_level2(W, H) - Creates Level 2 walls: vertical corridors

_level3(W, H) - Creates Level 3 walls: maze-like pattern

_level4(W, H) - Creates Level 4 walls: cross pattern with corners

_level5(W, H) - Creates Level 5 walls: complex maze

_level6(W, H) - Creates Level 6 walls: arena with inner box

create_walls_for_level(level, width, height, tile) - Returns wall list for specified level

---

13. util.py

Lines: 395
Purpose: Common utility functions and UI components

Functions

draw_text(surface, text, pos, size, color, bold, center) - Renders text with options

draw_shadow_text(surface, text, pos, size, color, shadow, offset, bold) - Renders text with drop shadow

clamp(v, lo, hi) - Clamps value between min and max

load_image(name, scale) - Loads image from images/ directory

load_image_to_height(name, height) - Loads and scales image to specified height

load_sound(name) - Loads sound from sounds/ directory

Button Class

init(rect, label, disabled) - Creates button with horror theme styling

draw(surface) - Renders button with flicker and glow effects

hit(pos) - Returns True if position is within button

Slider Class

init(rect, value, label) - Creates horizontal slider (0.0-1.0)

_get_handle_rect() - Returns handle rectangle position

draw(surface) - Renders slider with label and value

handle_event(event) - Handles mouse drag, returns True if changed

_update_value_from_pos(x) - Updates value from mouse position

---

Dropdown Class

init(rect, options, selected_index, label) - Creates dropdown with option list

get_selected() - Returns currently selected option

draw(surface) - Renders dropdown with expansion state

handle_event(event) - Handles click to open/select, returns True if changed

get_expanded_rect() - Returns full rect when expanded

---

14. multiplayer_setup.py

Lines: 187
Purpose: Alternative multiplayer setup screen (Legacy)

TextInput Class

init(rect, placeholder) - Creates text input with placeholder

handle_event(event) - Handles keyboard input, returns True on Enter

update(dt) - Updates cursor blink animation

draw(screen) - Renders input field with text and cursor

Functions

get_local_ip() - Gets local network IP address

get_hamachi_ip() - Attempts to detect Hamachi VPN IP

multiplayer_setup_screen(screen, clock) - Alternative multiplayer setup UI

Technical Details

Network Architecture

- **Protocol:** TCP with custom message framing (4-byte length prefix)
- **Serialization:** Python pickle for game state objects
- **Threading:** Separate receive threads for non-blocking I/O
- **Message Types:** game_state, player_data, player_action, chat, skin, weapon

AI System

- **Vision:** Raycast-based line-of-sight checking
- **Pathfinding:** Waypoint-based patrol with obstacle avoidance
- **Difficulty Scaling:** Speed, HP, and damage increase per level

Rendering Optimization

- **Pre-rendered Backgrounds:** Level backgrounds cached to surfaces
- **Particle Pooling:** Blood and explosion particles with age-based removal
- **Camera Culling:** Only visible objects rendered

Design Patterns Used

- **Singleton:** GameSettings for global configuration
- **Dataclass:** Player, Enemy, ScoreEntry for clean data structures
- **State Machine:** Game states (menu, playing, game_over, victory)

- **Observer:** Event-based input handling

Controls

```
Movement:
  W / Z / Up Arrow      - Move Up
  S / Down Arrow        - Move Down
  A / Q / Left Arrow    - Move Left
  D / Right Arrow       - Move Right

Combat:
  SPACE / Left Click    - Fire Weapon
  Right Click           - Fire Shotgun
  1                     - Select Pistol
  2                     - Select Shotgun
  3                     - Select Grenade

Interface:
  M                     - Toggle Minimap
  T                     - Open Chat (Multiplayer)
  H                     - Toggle HUD
  ESC                   - Pause / Back

Commando Abilities:
  SHIFT                 - Dash Ability
  CTRL                  - Shield Ability
```

Dependencies

Required: - Python 3.10+ - Pygame 2.0+

Standard Library Modules Used: - socket, threading, pickle, json - dataclasses, enum, typing - math, random, os, sys - time, datetime, queue - subprocess, re

---

Document generated for academic review purposes.

Total project scope: ~11,500 lines of Python code across 14 modules.