Architectural Document

Project: Garbage Collector

Authors:
Hayyan Serwer
Farees Farooq Ismail
Sahil Hardasani
Hussain Ahmed
Mohsin Siddiqui

Creation Date: 25.05.2025

---

1. Introduction

This document describes the architecture of the Trash Sorter Game, a simple multiplayer game built in Godot Engine. The game teaches players about waste sorting by having two players compete to correctly sort trash items (paper, plastic, bio) into the right bins.

The game is designed for exactly two players who connect over a network. Players use keyboard controls to move around and pick up trash items. The first player to reach 20 points by correctly sorting trash wins the game.

---

2. System Architecture Overview

2.1 Description of System Architecture

The Trash Sorter Game uses a simple client-server architecture where:

- One player hosts the game (becomes the server)
- The second player joins using the host's IP address
- The host player controls game logic and spawning
- Both players see synchronized game state

2.2 Key Components

The game consists of three main parts:

1. Main Scene (Main.tscn) - Handles networking and overall game management
2. Player Controller - Manages player movement, item pickup, and scoring
3. Trash Spawning System - Creates new trash items every 4 seconds

---

3. Functional and Non-Functional Requirements

3.1 Functional Requirements (FR)

FR-01: The game shall support exactly 2 players
FR-02: Players shall connect using IP address

FR-03: The game shall spawn 4 types of trash: paper, plastic, bio and restmull
FR-04: Players shall move using WASD
FR-05: Players shall pick up items using 'E' key
FR-06: Players shall drop items using 'F' key
FR-07: The game shall spawn new trash every 4 seconds
FR-08: Correct sorting shall give +1 point
FR-09: Wrong sorting shall give -1 point
FR-10: First player to reach 20 points wins
FR-11: Player scores shall be visible on screen
FR-12: The game shall show dustbins for sorting

3.2 Non-Functional Requirements (NFR)

NFR-01 (Performance): The game should run smoothly without lag
NFR-02 (Network): Players should see each other's actions immediately
NFR-03 (Usability): Controls should be easy to learn
NFR-04 (Reliability): Game should handle player disconnection
NFR-05 (Simplicity): Game should be simple to set up and play

---

4. Prioritization of Non-Functional Requirements

Priority 1: Network Synchronization (NFR-02)

Why Important: Both players need to see the same game state
How to Achieve: Use Godot's RPC system to sync player actions
How to Measure: Test if both players see items being picked up at the same time

Priority 2: Performance (NFR-01)

Why Important: Game should be playable and responsive
How to Achieve: Keep code simple, don't spawn too many objects
How to Measure: Check if game runs at stable framerate

Priority 3: Usability (NFR-03)

Why Important: Players should understand controls quickly
How to Achieve: Use standard WASD movement, simple E/F for pickup/drop
How to Measure: New players can play within 1 minute of starting

---

5. Architectural Principles

5.1 Keep It Simple

- Use basic Godot multiplayer features
- Minimal number of scripts and scenes
- Clear separation between networking and game logic

5.2 Host-Client Model

- First player who joins becomes the host
- Host manages trash spawning

5.3 Immediate Feedback

- Player actions (movement, pickup) show immediately
- Score updates happen right away

---

6. System Interfaces

6.1 Network Interface

- Connection Type: Direct IP connection
- Protocol: Godot's built-in multiplayer system
- Data Sync: RPC calls for player actions and scores

6.2 Input Interface

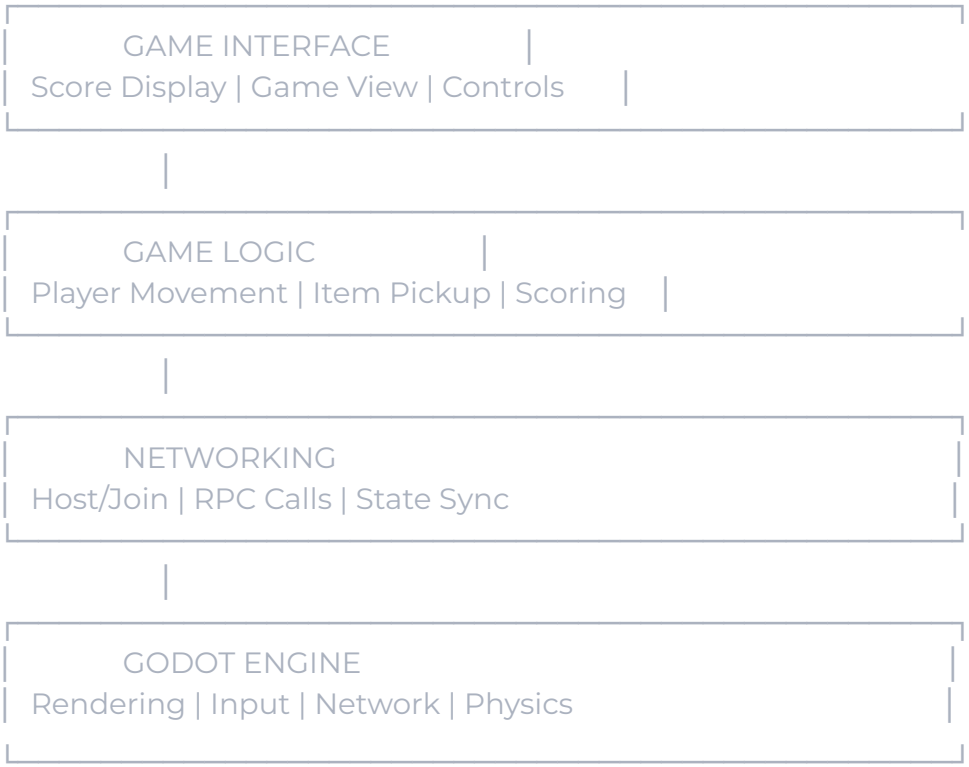
- Movement: WASD keys (Player 1) and (Player 2)
- Pickup: E key
- Drop: F key
- No mouse input required

6.3 Visual Interface

- Game View: Top-down 2D view
- UI Elements: Score display, dustbins, trash items
- Menu

---

7. Big Picture System Architecture



---

8. System Design and Decomposition

8.1 Main Scene (Main.tscn / main.gd)

Purpose: Controls the overall game Responsibilities:

- Handle network connections (host/join)
- Manage game state
- Spawn trash items every 4 seconds
- Receive scores from players
- Decide winner when someone reaches 20 points

Key Functions:

- _on_host_pressed() - Start hosting a game
- _on_join_pressed() - Join existing game
- update_player_score(player_id, new_score) - Update player scores [Separate RPC functions for syncing the score inside of main.gd]

8.2 Player Controller (player.gd)

Purpose: Controls individual player behavior Responsibilities:

- Handle player movement (WASD/Arrow keys)
- Pick up trash items (E key)
- Drop trash items (F key)
- Calculate score based on correct/wrong sorting
- Send score updates to main game

Key Functions:

- _process() - movement is handled inside of the _process() function, built into godot.
- pickup_trash() - Pick up nearby trash
- drop_trash() - Drop held item

Note: pickup_trash() is backed up by functions such as sync_pickedup_trash() for syncing it with the client and the host and _on_pickedup_area_entered(area) which detects area2D nodes that are inside of the PickupArea node.

8.3 Trash Items

Purpose: Objects that players sort Properties:

- Type (paper, plastic, bio and restmull)
- Position on game field
- Visual appearance (different colors/shapes)
- Can be picked up by players

8.4 Dustbins

Purpose: Sorting targets for trash items Properties:

- Type (paper bin, plastic bin, bio bin)
- Fixed position on game field
- Detect when items are dropped nearby
- Visual indicators for bin type

---

9. Design Alternatives and Decisions

9.1 Network Architecture Choice

Client-server architecture Reason: Uses godot's ENetMultiplayerPeer which is simpler to implement, host manages game state, fewer sync conflicts.

9.2 Control Scheme Choice

Option 1: Mouse-based drag and drop Option 2: Keyboard-based pickup/drop Decision: Keyboard controls (E to pickup, F to drop) Reason: Easier to implement, works better with multiplayer, simpler for players

9.3 Trash Spawning

Option 1: Random spawn locations Option 2: Fixed spawn points Decision: Random spawn locations Reason: More interesting gameplay, prevents camping strategies

---

10. Cross-Cutting Concerns

10.1 Error Handling

- Network Disconnection: Player score disappears but game continues
- Invalid Input: Ignore keypresses when player can't perform action

10.2 Performance

- Trash Cleanup: More than 10 trash items cannot be spawned at the same time
- Limited Spawning: Only spawn new trash every 4 seconds
- Simple Graphics: Use basic sprites and colors

10.3 Game State Management

- Score Tracking: Each player tracks own score, reports to host
- Win Condition: Host checks if any player reaches 20 points

---

11. Human-Machine Interface

11.1 Requirements for Human-Machine Interface

Interface Requirements:

- Show both player scores clearly
- Display different trash types with distinct visuals

- Show dustbins with clear type indicators
- Provide immediate feedback when items are picked up/dropped by updating the score

11.2 Design Principles and Style Guide

Visual Design:

- Colors: Different colors for each trash type
    - Paper: Blue
    - Plastic: Yellow
    - Bio: Green
    - Restmull:Black

- Simple Graphics: Basic shapes and sprites
- Clear Text: Large, readable score display

Layout:

- Top of Screen: Player scores
- Game Area: Main play field with dustbins and trash
- No Complex Menus: Start game directly with the start button

11.3 Interaction Modeling

Player Movement Flow:

Press WASD/Arrows → Player moves → Position syncs to other player

Item Pickup Flow:

Move near trash → Press E → Item attaches to player → Other player sees pickup

Sorting Flow:

Carry item to bin → Press F near bin → Check if correct type → Update score → Sync score

Game Flow:

Host starts game → Second player joins by IP → Game begins → Players sort trash → First to 20 points wins

Control Mapping:

- Player 1: WASD for movement
- Player 2: WASD for movement
- Both Players: E to pickup, F to drop
- Connection: Enter IP address to join

---

12. Summary

The Trash Sorter Game uses a simple architecture suitable for a beginner multiplayer project. The design focuses on:

Strengths:

- Simple host-client networking model
- Easy keyboard controls
- Clear game objectives
- Minimal but functional UI

Key Technical Features:

- Godot's built-in multiplayer system with RPC
- Real-time score synchronization
- Automatic trash spawning every 4 seconds
- Immediate visual feedback for player actions

Areas for Future Improvement:

- Handle player disconnection more gracefully
- Add reconnection capability
- Include single-player mode
- Add sound effects and background music

The architecture successfully delivers a working multiplayer game with simple, understandable code structure appropriate for learning software engineering concepts.