

Computer Games Development CW208

Technical Design Document

Year IV

EcoSynth

Haotai Xiong

C00265675

28/04/2024

Contents

[Technical Design 4](#_Toc165256011)

[Architecture 4](#_Toc165256012)

[Overview Diagram 4](#_Toc165256013)

[Resource Management 5](#_Toc165256014)

[Resource Management System 5](#_Toc165256015)

[Usage 5](#_Toc165256016)

[AI 6](#_Toc165256017)

[Enemy AI Update Cycle 10](#_Toc165256019)

[Movement and Obstacle Avoidance 11](#_Toc165256020)

[Gravity Field Mechanics 11](#_Toc165256021)

[Tower Detection and Behaviour 11](#_Toc165256022)

[Weapon System 12](#_Toc165256023)

[Base Gun 18](#_Toc165256024)

[Gravity Gun 19](#_Toc165256025)

[Laser Gun 19](#_Toc165256026)

[Heavy Laser Gun 19](#_Toc165256027)

[Building System 20](#_Toc165256028)

[Building Class Architecture 20](#_Toc165256029)

[Specialized Building Types 20](#_Toc165256030)

[Resource Processing and Effects 20](#_Toc165256031)

[Defensive Structures 20](#_Toc165256032)

[Events 21](#_Toc165256033)

[UI 22](#_Toc165256034)

[Resource Panel 22](#_Toc165256035)

[Building Panel 22](#_Toc165256036)

[Tech Tree Panel 23](#_Toc165256037)

[Tutorial Scene 24](#_Toc165256038)

[Save and Load System 25](#_Toc165256039)

# Technical Design

# Architecture

## Overview Diagram

A black screen shot of a computer

Description automatically generated

# Resource Management

## Resource Management System



These variables are global so I could easily deal with these resource values.

## Usage

The resource management system in EcoSynth is at the heart of the gameplay experience, demanding careful and strategic allocation of vital materials and energy to build and sustain your futuristic city. With in-game resources like wood, metal, and energy quantified by easily trackable integers, players can readily gauge their available materials briefly.

The wood, harvested from renewable Wood Resource Nodes, serves as the primary construction material, fostering urban expansion while underscoring the game's commitment to renewable resources.

Metal, a more durable and advanced building component, becomes crucial as players progress to develop more sophisticated structures and technologies.

Energy, the lifeblood of the city's various infrastructures and technologies, must be judiciously managed to keep the metropolis humming and thriving.

Waste management introduces a nuanced layer to the simulation, with a floating-point number reflecting the accumulation of waste over time, challenging players to implement efficient recycling systems and waste-to-energy solutions to mitigate environmental impact and contribute to the city’s health.

This holistic approach to resource management not only encapsulates the complexities of running a modern, eco-conscious city but also highlights the delicate balance between growth and sustainability that players must master.

# 

# AI





## 

[ 9](#_Toc165253599)



## Enemy AI Update Cycle

The Enemy class is central to the adversarial component of EcoSynth, representing dynamic challenges to player-driven city development. The update method encapsulates the enemy's behaviour cycle, where each enemy instance assesses its state during each game tick. If an Enemy object is marked alive, it proceeds to check for the absence of a target.

Without a target, it invokes findClosestBuilding, utilizing a lambda expression within std::min\_element to iterate over the t\_buildings vector. This selection process prioritizes buildings based on their weakLevel, demonstrating a preference for more vulnerable structures, which is a strategic factor in gameplay challenge.

## Movement and Obstacle Avoidance

Once a target is acquired, the Enemy mobilizes towards it via the moveToTarget method. This method computes a vector direction towards the target and normalizes it, guiding the movement decision. Should this direct path be obstructed, as determined by rayIntersectsObstacle, the enemy employs an obstacle avoidance algorithm.

This method involves a rotational scan in incremental angular steps, checking for viable alternative paths. This avoidance mechanism utilizes the rotateVector utility function, which mathematically rotates a vector by a given angle, signifying a potential direction change. The inclusion of different speed modifiers based on weather effects, like those during Event::Rainy, influences the enemy's pace, adding a layer of complexity to the movement dynamics and providing variability in gameplay.

## Gravity Field Mechanics

The Enemy class is further augmented by gravity field interactions, as demonstrated by the moveToGravityCenter method. This method enables a gravitational pull towards a specified center point when the enemy falls under a gravity effect, indicated by the m\_underGravity flag.

During such an effect, which is temporarily applied, enemies modify their movement behavior, demonstrating a form of dynamic environmental response within the game world.

## Tower Detection and Behaviour

Detection routines, like detectTower, play a pivotal role in the Enemy AI's response to the player's defensive structures. This routine iterates through buildings within the TileMap, dynamically casting to the Tower type to assess proximity and apply corresponding behavior changes. This interaction exemplifies the game’s strategy element, where enemy units respond intelligently to player-placed structures, necessitating thoughtful placement and use of defensive towers.

# Weapon System

A diagram of an object

Description automatically generated













## Base Gun

The BaseGun class serves as the primary weapon archetype within EcoSynth. Constructed to handle the fundamental mechanics of projectile creation and management, this class initializes bullets as sf::CircleShape instances with predefined radius and color attributes.

The update method iterates through the bullets vector, propelling each bullet by its velocity vector scaled by the elapsed time (deltaTime). Bullet lifespan is controlled through bounds checking, ensuring they exist only within the confines of the screen, thereby preserving performance and game balance.

Firing mechanics are regulated by fireInterval and bulletLimit constraints to prevent excessive shooting rates and on-screen clutter, while the fire method instantiates new bullets based on the player's position and direction of fire. Collision detection is an essential aspect, with checkCollision assessing bullet intersections with enemy sprites, ceasing bullet momentum upon impact to simulate a hit, and potentially signaling enemy damage.

## Gravity Gun

The GravityGun represents an advanced weapon system, encapsulating a GravitationalVortex object that is activated upon firing. This class exemplifies a departure from conventional projectile weaponry, shifting focus to area-of-effect (AoE) utility.

The update method delegates the frame-by-frame behavior of the vortex, which includes the intricacies of visual effects and gameplay impact within its area.

The rendering function, render, is responsible for drawing the vortex to the RenderWindow, while fire sets the vortex's activation state and location. The class provides accessors such as getShape and getPosition for collision checking and gameplay logic, emphasizing the weapon’s utility in controlling enemy positioning and influencing battlefield dynamics.

## Laser Gun

The LaserGun class introduces a direct and continuous beam weapon into EcoSynth's arsenal. The beam, an sf::RectangleShape, is initially defined with size and color properties to visually represent a laser.

The update function incorporates a fade-out mechanic using a clock to linearly decrease the beam's opacity over time (fadeDuration), creating a visual effect of the laser dissipating. Activation and orientation of the beam are handled within the fire method, aligning the laser according to the firing direction—a design that reflects different player input scenarios. The class supports an active state toggle, allowing the beam to be drawn to the screen only during active phases, thus aligning with efficient rendering practices and gameplay flow.

## Heavy Laser Gun

The HeavyLaserGun class builds upon the LaserGun foundations by introducing a charge mechanic through the isCharging state. The update method manages both the charging phase, tracked by chargeClock, and the releasing phase, where the beam's visual attributes such as size and opacity are dynamically adjusted according to the charge duration.

The release method transitions the weapon from charging to firing state, involving a visual and functional expansion of the beam to simulate a powerful energy discharge. This class illustrates a more complex weapon behavior, where player interaction dictates the potency and visual representation of the weapon's output, demonstrating a marriage of gameplay mechanics with user engagement.

# Building System

## Building Class Architecture

At the core of the building system is the building base class, which establishes common attributes such as position (m\_pos), texture name (m\_textureName), sprite (m\_sprite), and a weakness level (m\_weakLevel) indicative of the building's resilience. The render method ensures that each building's visual representation is properly drawn onto the game window, allowing for seamless integration into the game's graphical interface.

## Specialized Building Types

Derived classes such as House, Factory, Landfill, Recycling Centre, Plant, Connection, and Tower extend the building class, each overriding and enriching the base functionality with specialized behaviour. For example, the House class subtracts wood resources upon construction, representing resource expenditure. The Factory class includes mechanisms to establish or sever wood connections, reflecting its role in resource processing within the game.

## Resource Processing and Effects

Resource management is a focal point in the building system. The Factory and Landfill classes contain methods like updateWoodCollection and woodToEnergy, which simulate the conversion of resources into usable materials or energy, playing into the larger game economy.

These methods interact with global resource variables such as woodAmount, metalAmount, energyAmount, and wasteAmount, affecting the overall resource pool and requiring players to carefully plan their resource allocation and production strategies.

## Defensive Structures

The Tower class introduces a defensive element to the city-building aspect of EcoSynth. The presence of an energy-based m\_emitter attribute, with its drawEmit method, allows towers to consume energy resources to produce a defensive radius that can affect enemy units.

The dynamic radius expansion simulates the charging and discharging of defensive capabilities, providing a tactical layer to the gameplay where energy management becomes crucial to maintaining city defences.

## Events



In addition to their primary functionalities, buildings in EcoSynth are designed to respond to environmental factors, as indicated by the integration with the Event system. This interaction exemplifies how structures may have varied efficiencies and outputs depending on weather conditions, further intertwining resource management with the strategic adaptation to a changing game environment.

# UI

## Resource Panel

A grey background with black text

Description automatically generated

The Resource Panel acts as the information hub, presenting real-time updates on the player's available resources, including wood, metal, energy, and waste. This UI element is strategically designed for at-a-glance monitoring, with clear icons and counters that reflect the current resource levels.

Players rely on the Resource Panel to make informed decisions regarding resource allocation, construction, and technological upgrades. It is critical that the Resource Panel updates dynamically, reflecting changes instantaneously to provide accurate information as players gather, consume, or trade resources throughout the game.

## A screenshot of a video game Description automatically generatedBuilding Panel

The Resource Panel acts as the information hub, presenting real-time updates on the player's available resources, including wood, metal, energy, and waste. This UI element is strategically designed for at-a-glance monitoring, with clear icons and counters that reflect the current resource levels. Players rely on the Resource Panel to make informed decisions regarding resource allocation, construction, and technological upgrades. It is critical that the Resource Panel updates dynamically, reflecting changes instantaneously to provide accurate information as players gather, consume, or trade resources throughout the game.

## A screenshot of a video game Description automatically generatedTech Tree Panel

The Tech Tree Menu is the gateway to progression and innovation in EcoSynth. It outlines the paths players can take to unlock new technologies, buildings, and advanced game mechanics. This menu is visually structured as a branching path, with nodes representing individual technologies that can be unlocked sequentially or through specialized development routes. The Tech Tree Menu is not only a tool for advancement but also serves as a planning guide, helping players strategize their growth and adapt their city's evolution to their playstyle. Highlighting prerequisites, costs, and benefits of each technology, the menu offers depth and complexity to the gameplay, encouraging players to engage deeply with the game's strategic elements.

# Tutorial Scene

In EcoSynth, there is a tutorial scene where players will get lists of instructions before they actually play the game, this will help new players to get more familiar with the main game. A screenshot of a video game

Description automatically generated

# Save and Load System





The save and load system within EcoSynth employs an efficient and straightforward approach to persisting and retrieving game state, ensuring that players can seamlessly continue their city-building ventures across sessions.

The saveGameData function collects current state information—such as the player's position and the specifics of each building, including position, type, and texture—and populates the SaveData structure. This data is then serialized to a file through the saveGameDataToFile method, which meticulously formats each piece of information for clarity and ease of access, following a simple key-value pair convention.

Conversely, the loadSavedData function performs deserialization, parsing the stored data back into the game's data structures with ample error checking to handle file accessibility issues. It leverages std::istringstream to separate keys and values and uses standard library functions such as std::stoi and std::stof to convert string data back to numerical values. This bi-directional process is fundamental to the game's user experience, allowing for both the preservation of gameplay progress and the ability to restore the game world to its last known state with precision.