Tribz

Final Year Project Defense Report

A Prehistoric Tribe Management Game

Team "The Tribe"
Thomas Chen
Martin Doillon
Marie-Jasmine Andres-Yosrungruang
Emile Lassalle
Adrien Le Berre

May 2025

Abstract

Tribz is a real-time strategy and management game set in a prehistoric fantasy world. This report summarizes the development of *Tribz* by a team of five students, covering the project's concept, design, implementation, and outcomes. In *Tribz*, the player leads a tribe, manages resources, constructs a village, and survives threats like wild dinosaur attacks and raids, all while potentially cooperating with other players in a multiplayer mode. Over the course of the project, the team designed a cartoon-styled user interface, implemented complex AI behaviors (including A* pathfinding and dynamic role assignment for villagers), integrated original art and sound assets, and laid the groundwork for networked multiplayer gameplay. The document details the project context and motivation, the functional requirements derived from initial specifications, the chosen software architecture (using the Unity game engine and C), and the iterative design and development process. We present testing and validation results demonstrating the key features, discuss project management and team contributions, and reflect on challenges encountered and solutions applied. Finally, we outline potential future improvements for *Tribz* beyond this project and conclude with lessons learned and the project's overall achievements.

Acknowledgements

First and foremost, we thank our project supervisors and instructors for their invaluable guidance and feedback throughout the development of *Tribz*. Their expertise and encouragement helped us navigate challenges and refine our work. We are also grateful to our peers and play-testers who provided constructive feedback; their suggestions greatly improved the game's design and stability. Finally, we extend our heartfelt thanks to our families and friends for their constant support and understanding, which motivated us to persevere and deliver this project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Tribz is a real-time management and strategy game that transports the player several millennia into the past. It combines elements of history and fantasy, featuring prehistoric humans coexisting with dinosaurs in a unique tribal setting. The player is tasked with managing a prehistoric tribe: gathering and producing resources, building and upgrading a village, and protecting the community against various threats from the wild. The game is developed in C# using the Unity engine, which provides a robust framework for 2D simulation and user interface.

Inspired by games such as *Norland* [1], *Manor Lords* [2], *Anno 1800* [3], and *Jurassic World Evolution 2* [4], *Tribz* offers an immersive experience focused on resource management and the evolution of a community in a hostile environment. However, unlike these titles, *Tribz* explores a less common theme (prehistoric tribal life with fantastical elements) and targets a broad audience with its accessible cartoon art style. The project represents a unique opportunity for our team to combine our passions for strategy games, world-building, and creative design into a cohesive product. This report provides a comprehensive overview of the project, from its initial conception and requirements to the architecture, implementation, and final results. We begin by discussing the context and motivation behind *Tribz*, including the inspirations and goals that shaped its development.

Chapter 2

Project Context and Motivation

The idea for *Tribz* arose from our team's desire to create an original strategy game that combined elements each member was passionate about. We noticed that while there are many city-building and survival games, few are set in a prehistoric era with a light-hearted twist. Our goal was to fill this niche by developing a tribal management game that is both engaging and accessible. We chose a prehistoric setting enriched with a fantasy twist (dinosaurs living alongside humans) to spark players' imagination and differentiate *Tribz* from more conventional historical simulations. We also decided on a stylized, colorful visual design to ensure the game would be appealing to all ages, including younger players.

From the outset, we examined existing games and genre standards to inform our design. We looked at survival and city-building games like those mentioned in the Introduction, analyzing what made them enjoyable and what challenges they faced in gameplay design. This state-of-the-art review reinforced our motivation to focus on resource management and cooperative gameplay. For example, *Norland* emphasizes family and survival in a medieval setting, and *Manor Lords* offers detailed city-building; these inspired us to incorporate depth in managing our tribe's economy and population. At the same time, no existing game fully captured the concept of leading a primitive tribe in a world of dinosaurs, which motivated us to develop that concept in *Tribz*.

Beyond the creative concept, our motivation was also driven by what the project would mean for the team. As a final-year project, *Tribz* presented an opportunity to apply and expand our technical skills and to learn to work effectively as a team. Each member brought a different expertise (programming, AI, art, sound, etc.), and we wanted to combine these into a single cohesive project. We anticipated that working on *Tribz* would strengthen our knowledge of game development (especially

using the Unity engine and C), improve our understanding of concepts like artificial intelligence for games and user experience design, and hone our project management and collaboration skills. In essence, the project was as much about team learning and personal growth as it was about creating a fun and original game.

Objectives

We defined several key objectives and motivations for *Tribz* at the start of the project:

- **Create a compelling resource management experience:** The game should challenge players to efficiently manage natural resources (food, wood, stone, etc.) and plan the growth of their tribe. Strategic allocation of resources and planning of infrastructure are core to the gameplay.
- **Innovative prehistoric setting with fantasy elements:** Develop an interactive environment where players can tame and utilize dinosaurs. This includes the ability to domesticate certain species for the tribe's benefit (e.g., for defense or transportation) while managing the risks (a dinosaur could become unruly or attract predators).
- **Accessible and family-friendly design:** Utilize a cartoonish and colorful art style, intuitive controls, and a gradual learning curve so that players of all ages can enjoy the game. Despite involving survival elements, the game avoids excessive violence or gore, aligning with a more light-hearted tone.
- **Cooperative and social gameplay aspects:** Incorporate multiplayer elements enabling players to cooperate—such as trading resources or jointly facing challenges. Even though full multiplayer might extend beyond the initial project, laying the groundwork for player interaction was a motivating factor.
- **Team skill development:** Ensure that the project allows each team member to develop in their respective domain (programming, design, art, audio) and also learn from other domains. For instance, programmers would learn about art pipelines, artists about game engines, etc., fostering a well-rounded project experience.

Chapter 3

Requirements Specification

Based on the project objectives, we established a set of functional requirements and constraints for *Tribz*. These requirements were documented in a functional specification (Cahier des Charges) and guided our development process. Table 3.1 lists the core functional requirements of the game, each corresponding to a major gameplay feature we aimed to implement.

In addition to these functional requirements, we also outlined several non-functional requirements. These included performance goals (the game should run smoothly with dozens of active entities on screen), usability and accessibility considerations (configurable controls, color-blind friendly palette), and maintainability (well-structured code to allow future expansion of features). Meeting these requirements influenced decisions in architecture and implementation, as discussed in later sections.

Requirement	**Description**
Resource Management	The player must be able to gather, produce, and allocate resources such as wood, food, and stone. These resources are required for constructing buildings, feeding the population, and crafting other items necessary for survival.
Population Management	The game simulates villagers with individual needs (hunger, rest) and skills. The player can assign or influence villager roles (e.g., as gatherers, builders, warriors), and villagers autonomously carry out tasks. The well-being of the population (health, happiness) must affect productivity.
Construction and Upgrades	The player can construct various buildings (houses, storage, workshops, defenses) by spending resources. Buildings can be upgraded or improved over time, often requiring multiple stages of construction. Different structures provide benefits (e.g., a workshop enables new tools, a watchtower improves defense).
Defense and Threats	The game world includes dangers such as wild dinosaur attacks or raids by rival tribes/pillagers. The player must build defenses (walls, towers) and arm villagers to protect the tribe. A basic combat system is required where villagers and possibly tamed creatures can fight off threats.
Dinosaur Taming	Certain wild creatures (dinosaurs) can be tamed or domesticated. A tamed dinosaur can serve roles in the tribe (for example, as beasts of burden, combat companions, or for special tasks like scouting). Taming might involve a minigame or quest and requires ongoing care (feeding, training).
Multiplayer/Trading	The game should allow interaction between players' tribes. At minimum, this could be asynchronous trading via a neutral merchant character. Ideally, the architecture should support real-time cooperative play (multiple players in the same world) so that features like resource trading, joint defense against attacks, or competitive challenges are possible.
User Interface and Feedback	Provide an intuitive graphical user interface. The HUD must display key information (resource totals, population stats). Players should receive clear feedback on actions (e.g., placement previews for buildings, notifications if resources are insufficient, indicators when villagers are idle). Tooltips or a help system should be available to explain game mechanics.

Table 3.1: Key functional requirements for *Tribz* as identified in the project specification.

Chapter 4

Architecture and Technologies Used

Developing *Tribz* required choosing an appropriate architecture and toolset that could support the game's features. We opted to use the Unity game engine (2021.x) and C as the programming language, due to Unity's robust ecosystem for 2D game development and its built-in support for key features like physics, UI, and potentially networking. Early in the project, we designed a high-level software architecture to organize the game's functionality into modules, which would help the team work in parallel and ensure the game's complexity remained manageable.

At the core of *Tribz*'s architecture is the **Game Manager**, a central controller that maintains the overall game state (resource counts, population data, world time, etc.) and orchestrates interactions between subsystems. The major subsystems include the **AI System**, **User Interface (UI)**, **Multiplayer Network**, and **Audio Engine**, all running on top of Unity's framework for rendering and physics. We adopted a component-based design: each entity in the game (for example, a villager, a dinosaur, or a building) is a Unity GameObject with specific components (scripts) attached that define its behavior. This allowed us to encapsulate logic for different aspects of the game in separate C scripts and Unity components, improving modularity.

Figure 2 illustrates the relationships between the core components of our architecture. The Game Manager mediates between player inputs (UI) and game world changes, updates AI behavior each frame, communicates over the network to synchronize multiplayer state, and triggers audio/visual effects via Unity.

Key technologies and systems used in our architecture include:

- **Unity Engine:** Handles rendering of 2D sprites, physics collisions (used for

things like villagers colliding with objects or range detection for attacks), and the scene management. We made extensive use of Unity's scene editor to lay out environment tiles and place initial game objects.

- **C# Scripting:** All game logic is implemented in C#. We structured the code into classes such as `VillagerAIController`, `BuildingController`, `ResourceManager`, etc. Unity's MonoBehaviour lifecycle (using `Update()`, `FixedUpdate()`, etc.) was leveraged to update game logic every frame or on fixed intervals. We aimed to keep scripts focused (each script managing one aspect or entity type) to make debugging and iteration easier.
- **AI and Pathfinding:** For villager and NPC AI, we implemented our own pathfinding and decision-making logic. In particular, villagers use an A* pathfinding algorithm [5] to navigate the map grid. We chose A* over Unity's NavMesh because our game world is a dynamically changing grid (trees can be cut down, buildings erected, etc.), and we wanted fine control over walkable tiles. The AI System includes a simple planner that assigns tasks to villagers based on needs and tribe priorities (e.g., gather food if food is low, otherwise build new houses).

The networking aspect of *Tribz* was built in anticipation of future multiplayer features. We decided on a client-server model (with one player's instance potentially acting as host) to maintain an authoritative game state. Using Unity's UNet (deprecated) architecture as a reference, we started integrating the Unity Netcode for GameObjects library for synchronization of basic events. In practice, this meant creating networked object prefabs for key entities (like the merchant or players' avatars if added) and testing state synchronization for simple actions (e.g., a chat message or a trade event). While the full multiplayer gameplay is not yet realized, the architecture is in place to expand this more easily.

Finally, the audio system was structured around Unity's AudioSource and AudioMixer. We created an `AudioManager` script that keeps track of background music and can play sound effects through pooled audio source objects in the scene. This way, any part of the code (AI, UI, etc.) can request sounds by calling the AudioManager (e.g., `AudioManager.PlaySound("construction`$_{c}$*omplete*"))*, decouplingaudiologicfromgamelog*

`Overall, the chosen architecture and technologies proved effective. Unity provided a stable base and editing tools, while our modular code approach allowed team members to work relatively independently on AI, UI, and other parts, integrating their work through the Game Manager and defined interfaces. The next section discusses the design and user experience considerations which guided how we implemented these systems in practice.`

Chapter 5

Design and UI/UX Approach

The design of *Tribz* encompasses both the visual/artistic style and the user experience flow. From the beginning, we aimed for a coherent and appealing aesthetic: a bright, cartoon-like 2D art style that supports the game's accessible tone. Simultaneously, we focused on UI/UX principles to ensure the game is easy to learn and play, despite having complex strategy elements.

Visual Design and Art Style

All game art for *Tribz* was custom-made by our team's artist. The process began with hand-drawn sketches to conceptualize characters, dinosaurs, and buildings. We defined a **graphic style guide** to maintain consistency: this included a specific color palette (earthy greens, browns, and muted tones for natural elements, with brighter accent colors for highlights or important objects) and a semi-cartoony look (characters have slightly exaggerated features and proportions, outlines are clean and somewhat thick, and shading is done with simple gradients to suggest depth without realistic detail). Each asset was then digitized using a graphics tablet and software, vectorized or drawn as high-resolution sprites. By establishing this pipeline early, we ensured that as new assets (like additional dinosaurs or buildings) were added, they matched the established style.

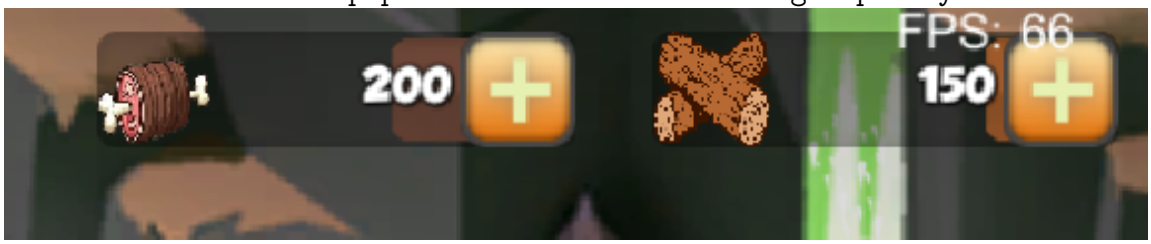
For example, our character sprites depict tribal villagers wearing simple prehistoric attire (furs, primitive jewelry) in a friendly manner (smiling faces, approachable designs). Even the dinosaurs are drawn to look somewhat charming rather than terrifying, aligning with the game's all-ages approach.

Figure 1 (earlier) gives a sense of the environment art style, showing how a village area and roaming dinosaurs appear vibrant and inviting. During development, whenever we added a new biome or object, we compared it against existing art to maintain art consistency. We also created a shared palette and references - for instance, all trees use variants of the same green shades, all buildings share architectural motifs (thatched roofs, wooden beams) to look related. This consistency was crucial as the content grew (see also the challenge of Art Consistency in Section 9).

User Interface Design

The user interface was designed for clarity and minimalism, providing essential information without overwhelming the player. We followed a classic strategy game HUD layout:

- The top of the screen displays a **resource bar**, showing icons and quantities for each resource (food, wood, stone, etc.), as well as the current population count and housing capacity.



- The bottom of the screen contains the **action panel**: buttons for building structures, managing villagers, and other actions. We implemented this as a "shop" or build menu. Initially, this menu was a single grid of all buildable structures, but after feedback, we categorized it into tabs (e.g., *Housing*, *Production*, *Defense*, *Others*) to help players find the right building type quickly. Each building icon has a tooltip that names it and shows its resource cost.

- When the player selects a building to place, a **ghost preview** of the building follows the cursor. This preview changes color or appearance to indicate whether the placement is valid (e.g., turns red if overlapping another object or if requirements aren't met). The player can click to confirm placement or right-click to cancel. This improvement made building placement much more user-friendly.

- For villager interactions, clicking a villager brings up a small **info panel** (at the bottom or side of the screen) showing that villager's stats (health, hunger) and current task. If we had implemented direct control, this panel would also allow assigning a role or task, but in our final implementation we kept villager AI autonomous, so this panel is mostly informative.

During the design, we paid attention to **UX feedback mechanisms**. For example, if the player tries to initiate a building but lacks resources, the game will display a brief message like "Not enough wood!" near the resource bar and perhaps play an error sound. Villagers in trouble (e.g., starving or under attack) flash an icon above their heads to alert the player. These cues help the player manage the tribe without requiring them to constantly dig through menus.

We conducted informal user testing of the UI by asking a few classmates to play the game (or a tutorial segment) and observing if they understood the controls and information. Based on observations, we adjusted the size and contrast of UI elements (initially, some text was too small on standard displays) and added a few tooltips where players seemed confused (like explaining what each resource icon represents, or that clicking a building opens an upgrade menu). We also considered accessibility: all important information conveyed by color (for example, the health bar color) is accompanied by an icon or text, to assist color-blind players.

Game Mechanics and Level Design

On the game mechanics side (which influences UI/UX as well), we aimed for a balance between strategic depth and simplicity. The game starts in a relatively safe state: the player has a few villagers and basic resources in a peaceful environment, giving them time to learn to gather resources and build shelters. As time progresses (or as the village grows), challenges are introduced: maybe a dinosaur wanders near the village or a neighboring tribe's scout appears. This progressive difficulty curve was part of the design to keep players engaged and gradually introduce mechanics (rather than overwhelming them at once).

We designed the **economy and build progression** to ensure the player always has short-term and mid-term goals. For instance, early game the player is prompted (through suggestions in UI or natural need) to build a fire and some tents (to house the population). Once that's done, the next obvious goal might be to secure food (build a trap or farm) before winter. After building basic facilities, the player might pursue taming a dinosaur (which requires building a pen and having a surplus of food to feed it). This progression was outlined in a simple internal design document, and we tried to support it with in-game guidance (like an optional objectives list on-screen, which we partially implemented).

The layout of the game world (level design) in our prototype consists of a single medium-sized map with varied terrain. There's a central grassland where the player starts, a forest on one side (source of wood and wild animals), a lake on another (water source, fishing potential), and some hills with stone deposits. This arrangement ensures the player must explore a bit to gather all types of resources, and it creates natural points of interest (e.g., the lake has limited shoreline for farming, the forest can hide dangers). We planned additional biomes for variety (as discussed in Future Improvements), but in the base design, we at least ensured the starting area had a bit of everything to get the gameplay loop running.

In summary, our design and UX approach was to make *Tribz* inviting visually and straightforward to play, while hiding a layer of complexity underneath for those who delve deeper. By combining a consistent art style with iterative UI improvements, we created a solid foundation that received positive feedback in play-testing and can be further polished with more time.

Chapter 6

Implementation and Development Process

The implementation of *Tribz* followed an iterative, feature-by-feature development process, aligned with Agile principles. In this section, we describe how key features were developed and integrated, and how the development process was managed over time.

Development Methodology

We organized our work into multiple sprints (typically two weeks long) with specific milestones or deliverables at each sprint's end. In the early sprints, our goal was to produce a minimum viable product-a basic playable game showcasing the core loop of gathering resources and building. Subsequent sprints focused on adding secondary features (like AI improvements, more building types, and audio) and refining existing ones (polishing UI, fixing bugs, optimizing performance).

We used a Git repository for version control, which allowed team members to work on different features simultaneously and merge changes regularly. To coordinate tasks, we maintained a task board (in the style of a Kanban board) listing tasks under categories for each domain (AI, UI, Art, Sound, Networking, etc.). Each team member would take ownership of tasks in their domain during a sprint, while also assisting others when tasks overlapped (for example, integrating art assets required both the artist and a programmer to work together).

Throughout implementation, we held brief meetings or check-ins multiple

times per week (often via an online call or group chat) to synchronize progress and address any blockers. We also did joint testing sessions at the end of major sprints to play the game together and note issues or improvements.

Core Feature Implementation

The first implemented feature was the resource gathering system. We set up the fundamental classes for resources and created gatherable objects in the world (trees for wood, berry bushes for food, rocks for stone). Villagers were programmed with simple AI: find the nearest resource, move to it, gather (by playing a gather animation and then "destroying" the resource node), and return to a stockpile to deposit. This established the basic economic loop. Once this loop was proven to work in a prototype (we could successfully collect, say, 10 units of wood and see it added to a resource counter), we moved on to construction.

For building construction, we introduced the concept of build sites. When the player selects a building to construct and confirms a location, a "construction site" object appears there. Villagers with the builder role will notice there is a construction site and bring the required resources to it. We used a state-based approach for the building process: a building can be in an "unbuilt" state (needs resources), "under construction" state (villager is actively working on it), and "completed" state (fully built and functional). Each building prefab in Unity had an associated script managing these states, as well as a list of required resources and construction time. This way, adding a new building type was as simple as defining its cost and build time in the inspector and assigning the script.

One of the challenges was implementing the **A* pathfinding** for villagers. We represented the walkable area of the map as a grid of nodes. Each node corresponded to a small area of the game world (for example, 1x1 meter in Unity units). We marked nodes as blocked if they were occupied by an obstacle (like a building or a large rock) and unblocked if free. We then wrote an A* search function that takes a start grid coordinate and a goal coordinate and returns a path (a sequence of grid points). To keep it efficient, we limited the grid size and employed some basic heuristics (Manhattan distance heuristic for A* since we allow 4-directional movement). We also made sure to recompute paths only when necessary (villagers don't constantly repath every frame unless their target or the environment changes).

Below is a simplified pseudocode of our A* pathfinding algorithm:

Listing 6.1: Simplified A* pathfinding algorithm used for NPC navigation.

```
private bool Search(Node currentNode) {
    // Current node Closed since it cannot be traversed more than once
    currentNode.State = NodeState.Closed;
    List<Node> nextNodes = GetAdjacentWalkableNodes(currentNode);

    nextNodes.Sort((node1, node2) => node1.F.CompareTo(node2.F));
    foreach (var nextNode in nextNodes)
    {
        // Check whether the end node has been reached
        if (nextNode.location == this.endNode.location)
        {
            return true;
        }
        else
        {
            // If not, check the next set of nodes
            if (Search(nextNode)) // Recurses back into Search(Node)
                return true;
        }
    }

    // The method returns false if this path leads to be a dead end
    return false;
}
```

When a villager is assigned a task that involves moving (which is most tasks), we call A* to get a path to the target. The path is then followed step by step each frame. We also implemented simple avoidance so that if two villagers meet face-to-face on a path, one will wait or choose a slightly different route around the other (to avoid infinite shoving).

With pathfinding in place, the overall AI became much more robust. Villagers could navigate around buildings and other obstacles to do their jobs. We expanded the AI with more behaviors: for example, if a villager became hungry, their AI state would switch to seek food (go to a food storage or source), eat, then resume their previous task. These behaviors were implemented in a finite-state machine manner within each villager's script.

Parallel to AI and core mechanics, we integrated the **audio and visual effects**. For instance, when a building is completed, we trigger a sound effect (a hammer sound) and a particle effect of celebratory confetti (to make the completion more satisfying). Unity's particle system was utilized for simple effects like dust when chopping trees or splashes when fishing in the lake. These touches enhanced feedback for player actions.

Networking Prototype

Late in development, we worked on a networking prototype to lay the groundwork for multiplayer. This involved a lot of architectural consideration: ensuring that game state could be cleanly separated and synchronized. We created a game scene variant that could be loaded in multiplayer mode, where each player's actions would be sent to a server (we tested with one instance acting as a host on localhost). We synchronized only a few things at first: for example, the presence and inventory of the traveling merchant character (a neutral NPC that could visit players' villages). In a multiplayer test, if Player A bought an item from the merchant, Player B's game would also reflect that the item was gone. Implementing this required us to network the merchant's inventory and use RPC (remote procedure calls) to update both clients.

We encountered some **synchronization issues** (discussed in the Challenges section) during these tests, but were able to resolve basic ones by using an authoritative approach (the host decides outcomes and informs the client). Due to time constraints, this prototype did not extend to full base-building sync or PvP interactions. However, the experience gained was valuable-our codebase is now structured in a way that many game actions are funneled through the Game Manager, which means we can insert network hooks there in the future (for example, when a building is placed, the Game Manager can not only update the local game but also send a network message).

Integration and Testing during Development

We maintained a practice of integrating features frequently to avoid late-stage merge conflicts. For example, when the artist finished a new sprite sheet for buildings, we immediately added those into the Unity project and updated the building prefabs. This close collaboration meant that for much of

the project, we always had a working build we could run to evaluate the game's current state.

To ensure quality, we performed regular testing at the end of each sprint. Each team member would play the game (often simultaneously) and note any bugs or balance issues. We also swapped roles in testing-e.g., the artist might focus on whether the UI feels intuitive, and the programmer might comment on art clarity-in order to get diverse perspectives. Some bugs we encountered and fixed during implementation included villagers getting stuck (solved by improving pathfinding and adding a timeout that teleports a villager to safety if stuck too long), resources not updating correctly in the UI (a simple bug in our UI binding that was quickly patched), and audio volume imbalance (some sounds were too loud initially, which we adjusted in the mixer).

The development process was documented through a shared journal where we logged major changes each week. This helped in preparing status updates for our project defenses and in ensuring all requirements were being addressed. By the final phases of implementation, we had a stable build of *Tribz* that included most core features and could be demonstrated to players, leaving only polish and optional features to future work.

Chapter 7

Testing and Validation

Testing and validation played a crucial role in our project to ensure that *Tribz* met its requirements and provided a smooth gameplay experience. We conducted several types of testing: functional testing of features, performance testing, and user experience testing. This chapter outlines our approach to testing and the outcomes of those efforts.

Functional Testing of Features

For each functional requirement (see Chapter 3), we devised test scenarios to validate that the implemented features worked as intended:

- **Resource Gathering and Usage:** We tested that villagers can successfully gather each type of resource and that the resources correctly register in the resource HUD. For example, in one test scenario, we depleted all nearby trees to ensure wood-gathering works from start to finish (tree object disappears, wood count increases, and the wood can then be spent on construction).
- **Building Construction:** We went through the process of placing each building type in the game, verifying that placement rules are enforced (e.g., you cannot place on water or overlapping another building), required resources are consumed upon construction, and the building transitions to a functional state when finished. We also tried cancelling constructions mid-way and confirmed resources were refunded appropriately (for unfinished projects we allowed reclaiming materials).

- **Villager AI Behavior:** We observed villagers over extended periods to ensure they follow their AI logic reliably. For instance, we intentionally created situations to test behavior: letting food run out to see if villagers starve (they do lose health over time if starving), or spawning an extra dinosaur to see if villagers appropriately flee or fight when attacked. These tests verified that the state transitions in AI (like switching to hunger state) trigger correctly.
- **Defense Mechanics:** We simulated attacks by triggering a raid event (spawning a hostile creature at the edge of the map) and ensured that any combat functions worked. In our implementation, combat is automated: if a villager with a weapon is near an enemy, they will attack. We checked that health deducted properly, death conditions worked (villagers or enemies die at 0 health with a death animation, etc.), and that the raid ends when enemies are defeated. We discovered and fixed a bug during this test where villagers would sometimes not "see" an enemy due to line-of-sight checks being too strict.
- **Trading System:** For the trading mechanic (via the merchant NPC), we tested that a player could initiate a trade, the correct amount of resources would be exchanged, and that both the player's inventory and merchant's inventory update. Since this was one of the simpler multiplayer interactions, we also tested it in a two-player session to see that both players saw consistent results of the trade.

To organize these tests, we maintained a checklist mapping tests to requirements. Most tests were done manually through gameplay, though for some repetitive tasks (like testing pathfinding on many random pairs of points), we wrote debug commands to automate the process. For example, pressing a certain key in our debug mode would spawn 50 villagers at random positions and have them all walk to a target-this stress-tested the pathfinding and AI under load.

By the final project presentation, we were able to demonstrate that each core feature was operational. Minor issues that remained were documented (for instance, occasionally two villagers might overlap if assigned to the same target resource; this was a known issue but didn't break gameplay). The functional tests gave us confidence that *Tribz* meets the intended requirements outlined initially.

Performance and Stress Testing

We also performed performance testing to ensure the game ran smoothly. Our target was 60 frames per second on a mid-range PC for a scenario with a developed village (20 villagers, 10 buildings). Using Unity's Profiler, we identified that the most CPU-intensive parts were the AI updates, especially pathfinding calculations. We addressed this partly by optimizing the pathfinding algorithm (as described in the Implementation chapter) and by limiting how often certain calculations happen. For example, we gave each villager a staggered update cycle for expensive operations: not all villagers think about their next task on the exact same frame, which smooths out CPU usage spikes.

In stress tests, we pushed the game beyond normal conditions-like spawning 100 villagers or rapidly creating and destroying buildings-to observe behavior. The game held up reasonably well up to a point: around 50 active villagers, the frame rate would start to drop on our test machine, largely due to the pathfinding and collision checks. Given that typical gameplay would rarely exceed 20-30 villagers (based on our design/balance), this performance was deemed acceptable for the scope of the project. We noted that future improvements (like more advanced pathfinding or LOD (level of detail) for AI) could extend this capacity.

Memory usage was low, and no memory leaks were observed (we checked that objects were properly destroyed when no longer needed, using Unity's tools to monitor object counts).

User Experience Testing

To validate the user experience, we conducted small playtesting sessions. We invited a few fellow students who were not intimately familiar with our game to play a build of *Tribz* for about 15-30 minutes while we observed quietly. Afterward, we asked them to provide feedback on what was confusing or what felt smooth.

From these sessions, we gathered valuable feedback: • New players appreciated the concept and found the game's premise fun, but some were unsure what to do first when the game started. This indicated the need for better guidance at the very beginning (which we then added by making the starting villagers automatically begin gathering and by adding a message like "Build

a fire pit to cook food!" as an initial hint). • The build menu categorization (added later in development) significantly helped players find the buildings they wanted; earlier testers had struggled when all buildings were in one uncategorized list. • The testers liked the visual style and particularly commented that the dinosaurs were "cute and not scary," which confirmed our art direction was effective for approachability. • One common point of confusion was how the trading with the merchant worked (testers didn't immediately realize they needed to click on the merchant when he visited). In response, we adjusted the merchant's design to make him more visually distinct (bigger sprite, an icon above him when he arrives) and added a notification, "A merchant has arrived! Click on him to trade."

We also incorporated feedback on controls-for example, adding a keyboard shortcut for opening the build menu, and enabling the Escape key to close any open window or cancel placement (standard conventions that testers expected).

Finally, we ensured that the game was stable during these playtests. Over multiple sessions, we didn't encounter any crashes or severe bugs, indicating good overall stability. Minor issues (like overlapping UI elements at certain screen resolutions) were noted and fixed by adjusting our Canvas settings in Unity (using anchor presets to ensure UI scales correctly).

Validation Against Requirements

By the end of our testing phase, we validated that *Tribz* met the majority of its initial requirements. Appendix A provides a summary of the implementation status of each planned feature, showing that most were completed or partially implemented. In summary:

- All essential gameplay mechanics (resource management, building, basic AI) were fully implemented and verified.
- Certain stretch goals (full multiplayer co-op, elaborate dinosaur taming mechanics) remained as future work, but a foundation for these was in place.
- The user interface and experience were refined to a point where new players could understand and play the game, satisfying our accessibility and usability goals.

- The performance was within acceptable bounds for the defined scope, and the game ran stably without critical errors.

Overall, testing and validation gave the team confidence in the quality of the product we are delivering. It also highlighted areas for future improvement, but importantly confirmed that the core experience of *Tribz* is functional and enjoyable.

Chapter 8

Project Management and Team Contributions

Developing *Tribz* was not only a technical endeavor but also a lesson in teamwork and project management. In this chapter, we describe how we organized our team, the methodologies we used to manage progress, and each member's contributions to the project.

Team Organization and Workflow

Our team ("The Tribe") consisted of five members, and we assigned roles based on each person's strengths and interests. However, we maintained flexibility, with members frequently collaborating across role boundaries. The nominal roles were:

- **Martin Doillon** - *AI Programmer Game Balancer*
- **Emile Lassalle** - *UI/UX Designer Network Engineer*
- **Adrien Le Berre** - *Lead Game Designer (Gameplay Mechanics)*
- **Thomas Chen** - *Sound Designer Audio Engineer*
- **Marie-Jasmine Andres-Yosrungruang** - *2D Artist Asset Creator*

From the beginning, we adopted Agile practices, version control (Git), and team communication tools to keep development organized. Key elements of our workflow included:

- **Bi-Weekly Sprints:** We planned the project in two-week sprints. At the start of each sprint, we held a planning meeting to choose which features or tasks to focus on. Each day, we did asynchronous check-ins via a group chat to update each other on progress or blockers. At the end of the sprint, we held a review where we collectively tested new features and gathered feedback for the next sprint.
- **Kanban Board:** We maintained a task board (using an online tool) categorized by domains-AI, UI, Art, Sound, Game Design. Tasks would move from "To Do" to "In Progress" to "Done." This provided transparency for everyone to see who was working on what and what tasks were upcoming. It also helped in ensuring all aspects (for example, sound or art) were getting attention in each sprint.
- **Regular Builds and Integrations:** After each sprint (and sometimes more often), we produced an internal playable build of the game. By continuously integrating and testing, we caught integration issues (like a new feature breaking an old one) early. This practice also meant that we always had an up-to-date version of the game ready to showcase, which was useful for interim presentations to our advisors.

Using these methods, we were able to manage the project effectively. Of course, we encountered the typical challenges of scope management-deciding what features to cut when we were short on time, and ensuring that no single person was overloaded while others were idle. We mitigated these by reprioritizing tasks in each sprint planning (for example, pushing non-essential polish tasks to the backlog) and by pair-programming or collaborative work when a task was large (e.g., Emile and Martin worked together on integrating the AI with the UI notifications).

Individual Contributions

Every team member significantly contributed to *Tribz*, each in different areas. Below we summarize the primary contributions and responsibilities of each member:

Martin Doillon (AI Development Game Balancing): Martin was responsible for the artificial intelligence systems that drive villager behavior and game balancing. He implemented the core AI logic, including the villager

finite state machines (idle, gather, build, eat, etc.) and the A* pathfinding algorithm. Martin also introduced advanced AI features in later stages, such as dynamic role assignment (villagers can change jobs depending on needs) and basic enemy AI for raiders. In terms of balancing, Martin tuned parameters like resource yields, villager hunger rates, and enemy strength to ensure the game provided a steady challenge without becoming unmanageable. He conducted many of the internal playtests specifically to adjust these values for a fun gameplay experience.

Emile Lassalle (UI/UX Design Networking): Emile took charge of designing and implementing the user interface. He created the HUD layout, the building menu system, and various in-game panels (such as the villager info panel and notifications). Emile's focus was on making the interface intuitive: he implemented features like the building placement preview and category tabs in the build menu following tester feedback. Later in the project, Emile shifted attention to the networking prototype. He set up the initial multiplayer framework, figuring out how to sync game objects and writing the code for networked trading. Emile effectively bridged the game's presentation layer and its underlying systems, ensuring that the player's interactions were captured and reflected in the game world (both locally and across the network in multiplayer tests).

Adrien Le Berre (Game Mechanics Design): Adrien was the lead game designer, crafting the rules and mechanics that define *Tribz*. He specified how the economy works, what resources are needed for each building, and how the progression unfolds. Adrien implemented the construction system, including multi-stage building upgrades (for example, a hut can be upgraded to a cottage with more resources). He also worked on world design, setting up the terrain and resource node distribution in the map. When it came to gameplay features, Adrien was often the one to prototype new ideas: for instance, he scripted a simple event system for random occurrences (a small chance each day of a special event like "bountiful harvest" or "dino sighting" which we experimented with). Many of these were tunings or optional mechanics that could be toggled. Throughout the project, Adrien kept the team focused on the overall player experience, ensuring that all individual features meshed well together to create engaging gameplay.

```
void OnMouseOverTile(Tile tile)
{
    if (tile.IsOccupied)
        tile.SetColor(Color.red);
    else
        tile.SetColor(Color.green);
}
```

Thomas Chen (Sound Design Audio Implementation): Thomas handled everything audio-visual that had to do with sound. He composed a set of short musical loops for the game using digital audio software (including an ambient daytime theme and a tense combat theme). He also gathered or created sound effects for various actions: chopping wood, construction sounds, dinosaur roars, etc. In Unity, Thomas set up AudioSources for background music and one-shots for effects, and wrote the AudioManager code that other developers could call. He paid special attention to the mixing of sounds-ensuring that background music volume allowed effects to be heard clearly, and that multiple effects playing together still sounded pleasant. Thomas also did a pass on optimizing audio, such as disabling certain 3D sound settings for 2D sounds to reduce unnecessary processing. His work greatly increased the immersion of the game and provided feedback cues that were vital for UX (like a sound when an objective is completed or when an error occurs).

```
public class BackgroundMusic : MonoBehaviour
{
    public AudioSource musicSource;

    void Start()
    {
        musicSource.loop = true;
        musicSource.Play();
    }
}
```

```
using UnityEngine;

public class SoundManager : MonoBehaviour
{
    public AudioSource audioSource;
    public AudioClip constructionSound;

    void Start()
    {
        audioSource = GetComponent<AudioSource>();
    }

    public void PlayConstructionSound()
    {
        audioSource.PlayOneShot(constructionSound);
    }
}
```

```
[SerializeField] AudioSource buildSound;
public void PlayBuildSound()
{
    if (!buildSound.isPlaying)
        buildSound.Play();
}
```

```
public void AssignRole(Villager villager)
{
    if (NeedWood())
        villager.SetRole(VillagerRole.Lumberjack);
    else if (NeedStone())
        villager.SetRole(VillagerRole.Miner);
    else if (NeedBuilder())
        villager.SetRole(VillagerRole.Builder);
}
```

Marie-Jasmine Andres-Yosrungruang (2D Art Animation): Marie-Jasmine was the creative force behind the game's visuals. She designed and drew all characters (male and female villager variants, the trader Naël, enemy raiders), dinosaurs (a few species such as a triceratops-like herbivore and a small raptor-like predator), and environmental tiles (trees, bushes, rocks, water, ground textures).

After sketching concepts and getting team feedback, she produced the final digital art and exported them as sprites with transparency to be used in Unity. For animations, Marie-Jasmine created frame-by-frame animations for basic movements (villager walking, chopping, building hammer motion, etc.) and ensured they integrated correctly with Unity's animator. Additionally, she prepared concept art for potential future assets-like sketches of a mammoth or additional building types-some of which were used in our report and presentations to illustrate future plans. Marie-Jasmine also contributed to UI artwork, drawing custom icons for resources and actions, which helped maintain a consistent style across the game. [a4paper,12pt]article graphicx caption

It is worth noting that while each member had clear primary responsibilities as outlined, the development process was highly collaborative. We often debugged issues together (for example, when a bug involved both AI and UI, Martin and Emile would pair up to solve it). Similarly, design decisions were discussed as a group to reach consensus, rather than individually in isolation. This collaborative spirit is what allowed us to integrate such diverse elements (AI, UI, art, sound, etc.) into a cohesive final product.

Chapter 9

Challenges Faced and Solutions

Over the course of the project, we encountered a number of challenges, both technical and organizational. In this chapter, we detail the major challenges and the solutions or mitigations we applied for each.

AI Complexity vs. Performance

One significant challenge was ensuring that the AI system remained efficient as we increased its complexity. Initially, with only a few villagers and simple behavior, performance was not an issue. However, as we added pathfinding and scaled up the number of AI entities, we observed higher CPU usage and occasional frame-rate drops. The worst-case scenario tested was having many villagers and enemies moving simultaneously, which could result in dozens of pathfinding computations in a short time.

Solution: We optimized the AI in several ways. First, we introduced the concept of an AI update manager that staggers heavy computations. Instead of all 20 villagers computing a new path in the same frame, we distribute these calculations over several frames (e.g., 5 per frame over 4 frames). Human players typically do not notice if one villager starts moving a fraction of a second later than another, but this greatly smooths out CPU load spikes. Second, we looked into pathfinding optimizations: Martin experimented with a hierarchical pathfinding approach for the map. The idea was to have a coarse grid (for example, 4x4 chunks of the map) to first plan a broad route,

then refine it in local areas with the normal A*. This can avoid exploring unnecessary nodes far from the path. We partially implemented this and saw improved performance in large distances. Additionally, we made sure to cache and reuse paths when appropriate. For example, if multiple villagers are all heading to the same resource stockpile, they can share a path or use each other's trails instead of computing separate paths.

Another AI-related performance fix was simplifying collision checks. We initially allowed Unity's physics to handle villager collisions (to prevent overlapping), but we found that at larger scales, these physics calculations became costly. We solved this by turning off full physics for villagers and instead handling avoidance in code (as described earlier, a simple rule to not move if someone is directly in front). This reduced the overhead of using the physics engine unnecessarily.

These measures combined ensured that our AI could handle moderate loads. In the final testing, the game runs at the target frame rate under expected conditions. Should we push to even larger scales in the future, more advanced optimizations or multithreading the AI (using Unity's Job System) could be explored, but for the scope of our project, the solutions implemented sufficed.

Network Synchronization Issues

When building the multiplayer prototype, we encountered synchronization problems between clients. For instance, during a test, one player built a structure that did not appear on the other player's screen, or a dinosaur would be in a different location on each machine. These inconsistencies were due to the difficulty of keeping a complex simulation in lockstep, especially since we retrofitted networking onto a project originally coded for single-player.

Solution: We adopted an authoritative server model to maintain a single source of truth. In our tests, one instance of the game is the host/server and the others are clients. We structured network messages so that clients would send an intention (e.g., "Player wants to build X at location Y") to the host, and the host would then validate and broadcast the result ("Build X at Y successful, here is

the object ID"). By doing this, we ensured that crucial state changes occur in one place (the server) and are simply mirrored to clients. This solved issues like double-counting of resources or divergent game states.

We also added confirmation steps for important actions. For example, in the trading system, if Player A initiates a trade, their UI will show a "pending" state until a confirmation comes back from the server that the trade went through. This prevents the scenario where Player A's game thinks the trade happened but Player B's does not.

Due to the limited time, we did not fully eliminate all minor glitches (sometimes there is a slight delay or stutter when syncing many events at once), but we managed to synchronize all the key game elements that we attempted. The experience taught us much about Unity's networking and the need to design with multiplayer in mind from the start to avoid such issues. If multiplayer becomes a main focus in future development, we would likely refactor certain systems and possibly integrate a dedicated networking framework (like Mirror or Photon for Unity) to handle low-level sync more robustly.

Art Consistency Across Assets

As the number of art assets grew, maintaining a consistent look and feel was an artistic challenge. Early on, some assets created at different times or by different methods (scanned sketches vs. digital drawing) did not perfectly match in style. For example, the first few building sprites had thicker outlines and darker colors than the later ones, which could look slightly jarring when all are seen together in-game.

Solution: To address this, Marie-Jasmine established a more detailed art style guide and went back to adjust assets that were outliers. The guide included specifics like line thickness (e.g., all outlines should be 2 px at game resolution), shadow direction (bottom-right for all objects, for consistency with a fixed light source), and color swatches for common materials (all wooden structures use the same base brown tones, stone uses the same grey palette, etc.). She also decided on a standard scale: how big a human is relative to a building or a dinosaur in the world, and made sure all sprites were drawn to that scale to avoid mismatched proportions.

We scheduled an "art polish" sprint toward the end where the primary goal was consistency. In this sprint, older assets were re-exported after minor tweaks. We also did side-by-side comparisons in a test scene (placing all units and buildings next to each other) to visually inspect for any that stood out. This process significantly improved the overall visual cohesion.

The outcome is that *Tribz* now has a unified aesthetic: one can tell everything belongs to the same game world. Consistency in art not only improves visual appeal but also gameplay-players can more easily distinguish interactable objects if they follow a consistent style (for instance, all resource nodes have a subtle glow effect that was uniformly applied so players know they can be clicked on).

Audio Overlap and Volume Balancing

With many sound effects integrated (for building, gathering, alerts, etc.), we ran into an issue where sounds could overlap or play too frequently, resulting in a noisy or unpleasant audio experience. This was especially noticeable during hectic moments: if five villagers completed tasks at the same time, five sound effects might play simultaneously, causing a cacophony. Additionally, some sounds dominated others, and the balance between game sounds and music needed refinement.

Solution: Thomas used Unity's audio mixing tools to manage this. We grouped similar sounds into mixer groups (e.g., all ambient/environment sounds in one group, all UI/click sounds in another). By doing so, we could collectively lower the volume of a group or set a limit on how many instances of a sound can play at once. For example, we set a cap so that no more than 3 "construction hammer" sounds can play in any one second-if additional ones trigger, they are either skipped or queued faintly. This required some scripting but Unity's audio system has built-in features for voice limits.

We also carefully adjusted relative volumes: background music was set to a lower level and given a slight low-pass filter during critical sound moments (Unity can do snapshot transitions in the mixer, so when an attack happens, the music ducks a bit to let alert

sounds through). Each sound effect's volume was tuned; loud ones like the dinosaur roar were brought down just enough to not spike, while softer ones like footsteps were increased slightly but also set to not spam if many villagers walk together.

Additionally, we introduced randomness in pitch/volume for repetitive sounds to avoid them sounding too identical and robotic. A wood-chopping sound might play at a subtly different pitch each time, which is less fatiguing on the ears when heard repeatedly.

Through playtesting with these audio tweaks, we found the soundscape much improved. Players can still hear important cues (an alarm horn when an attack occurs, etc.) without being drowned out by unrelated noises. The volume balancing ensured that the audio contributed positively to the experience and did not become a distraction.

User Onboarding and Tutorial

Despite our efforts to make the game intuitive, we recognized that completely new players might struggle with understanding all mechanics (especially in a strategy game with multiple systems). Implementing a full tutorial mode was a challenge we couldn't fully solve within the project's timeframe. A guided tutorial or campaign introduction would ideally teach players step-by-step, but creating that requires significant scripting and possibly bespoke scenarios.

Solution: As an interim measure, we added in-game guidance tools. One such tool is the "Advisor tips" system: a small text box near the top of the screen that occasionally provides suggestions, e.g., "Your people need shelter: try building a Tent from the build menu." These tips are triggered by game state conditions (like if it's been X minutes and the player hasn't built any house, or if food is very low). They serve to nudge the player in the right direction. We kept them simple and not too frequent, so as not to annoy experienced players.

We also wrote a concise game guide in the pause menu, accessible at any time. It's basically a scrollable text that explains the basic controls and goals. It covers topics like how to assign roles to villagers (or rather how villagers automatically assign based on needs, in our implementation) and what to do if attacked. This guide is a stop-gap for not having an interactive tutorial.

While these additions were helpful, we consider onboarding still an open challenge. The ideal solution, which we propose for future work, is a dedicated tutorial scenario that introduces concepts one by one (for example, a scenario where you must build a fire to survive the night, teaching resource gathering and building in a focused context). Since we couldn't implement this fully, we at least laid the groundwork (the game's code is structured to allow disabling certain features for a tutorial and guiding the player without threats in the beginning).

In summary, our solution for onboarding was to provide as much information as possible through the UI and optional guides, acknowledging that the full tutorial experience would be a next step beyond this project.

Chapter 10

Future Improvements

Although *Tribz* is fully playable and meets the major goals set out at the project's start, there are many features and enhancements we envision adding with more time. Some of these are improvements that were planned but fell outside the time scope, while others are new ideas that emerged during development and testing. This chapter outlines the most significant future improvements and additions that could enhance *Tribz*.

Immediate Development Goals

In the near term, focusing on the next phase of development, we have identified several high-priority improvements:

- **Advanced AI Enhancements:** We plan to enrich the AI by introducing more complex behaviors and agents. This includes implementing raider AI that can coordinate attacks (instead of just random single enemies) and specialized wildlife behavior (for example, predators that hunt other animals or threaten the village at night, and prey animals that villagers could hunt). These will add depth to the survival aspect. We also aim to keep improving the efficiency of pathfinding, possibly using techniques like hierarchical grids or flow fields for crowds, to handle the advanced AI without performance loss.
- **Fuller Multiplayer Features:** With the groundwork laid for networking, a next immediate step is to implement true cooperative gameplay. This

means multiple players controlling their own tribes in the same world, able to assist each other or compete in a balanced way. Key features would be alliances (players formally teaming up, maybe sharing vision or resources), a trading system UI for player-to-player exchange, and possibly small-scale PvP conflicts for those who want a competitive edge (e.g., the ability to challenge another tribe or steal resources in a controlled manner). We would also integrate chat or communication tools for players in-game to coordinate.

- **Biome Diversity:** Currently the game has a single biome (temperate forest/plains). We intend to add at least 2-3 distinct biomes, such as a desert biome, a swamp/jungle biome, or a volcanic region. Each biome would come with unique resources (e.g., cactus fruit in the desert, medicinal herbs in the swamp), unique challenges (deserts might have scarcity of water, volcano areas might have periodic lava flows or earthquakes), and distinct flora/fauna. This will not only make the game world more interesting to explore but also add strategic decisions-perhaps encouraging trade, as one biome's resources might be valuable to a tribe in another biome.

- **Expanded Sound Design:** We plan to compose additional music tracks to provide variety (different themes for daytime, nighttime, combat, and peaceful village management). Each biome could also have its own ambient soundscape (for instance, desert winds or jungle birds). Moreover, we want to refine event-based audio triggers: more nuanced sound cues for things like a villager leveling up in skill, a building being upgraded, or a warning drum sound when a large threat is approaching. We may also incorporate basic voice lines or vocalizations for villagers (even if just simlish or murmurs) to give them more personality.

- **Visual Effects and Polish:** There are numerous visual improvements to pursue. One is implementing a day-night cycle with smooth lighting transitions (which could also affect gameplay, like certain dinos appearing only at night). Weather effects are another goal-rain could water crops but slow movement, thunderstorms might randomly start fires, etc., adding emergent challenges. We also plan to add small animations and effects to existing mechanics: for example, buildings visually showing upgrades (adding new parts as they level up), villagers visibly carrying resource bundles when hauling items, and more elaborate

particle effects for magic or special events (if any fantasy elements like tribal rituals are added).

These immediate goals are aimed at enhancing the core experience that already exists in *Tribz*, making the world more dynamic, the gameplay deeper, and the presentation more polished.

Long-Term Vision

Looking further ahead, we have a broader vision for what *Tribz* could become if development continued beyond the scope of an academic project. Some of the long-term ideas include:

- **Endgame Objectives and Narrative:** Currently, *Tribz* is a sandbox with implicit goals (survive, grow your tribe). In the future, we'd introduce explicit endgame objectives to work towards, providing a satisfying conclusion or win-state. For example, an endgame goal might be constructing a great "Wonder" for your tribe (like a monumental temple or wonder of the world) which requires massive resources and cooperation. Another concept is introducing legendary boss creatures (perhaps a giant dinosaur or mythical beast) that require a well-prepared tribe (or multiple tribes in multiplayer) to defeat, effectively serving as a final challenge.
- **Modular Building and Technology Progression:** We would like to expand the building system into a more complex technology tree. Rather than a linear upgrade (hut -> cottage -> house), players could customize or choose paths: e.g., turn a basic hut either into a defensive bunker or a larger communal house depending on need. Modular add-ons for buildings could be a feature (for instance, add a watchtower module on a palisade wall to give archers a perch). Over time, the tribe might progress from the stone age towards a proto-civilization (introducing early technologies like metal working). We must be careful to still keep the prehistoric theme consistent, but a bit of progression keeps long-term play interesting.
- **Robust Player Onboarding and Difficulty Scaling:** In the long run, we envision a multi-stage tutorial or story-driven campaign that can also double as an introduction to the game. New players could play

a short campaign of scenarios (each focusing on a mechanic, such as hunting, then one on defense, etc.) with narrative elements (perhaps following the story of a tribe leader). This would ease players into the full sandbox mode. Additionally, we'd implement difficulty levels or adaptive difficulty-so that both casual players and hardcore strategy fans can enjoy the game. Adaptive events could ramp up or down based on how well the player is doing.

- **Community and Modding Support:** If *Tribz* were to grow into a larger project, supporting player creativity could be a long-term goal. This might involve exposing modding tools or at least scenario editors where players can create their own maps or events. While this is beyond a student project scope, it's part of the vision to have a game that lives beyond the content we developers put in, thriving on community-driven content.

These long-term features paint a picture of *Tribz* evolving from a solid base (which we have built) into a rich, potentially extensible game. Many of these ideas come from brainstorming sessions we had when imagining "wouldn't it be cool if...". We recognize that implementing them would require careful planning and likely a lot more development resources, but they serve as an aspirational roadmap for what *Tribz* could become.

In conclusion, *Tribz* has ample room to grow. The future improvements listed here, both short-term and long-term, would significantly expand the content and capabilities of the game. We hope that either through our own continued efforts or by sharing the project with the community, these improvements can be realized, bringing *Tribz* to its full potential.

Chapter 11

Conclusion

Developing *Tribz* has been a challenging and rewarding journey. In this final chapter, we reflect on what has been achieved, what we have learned, and how the project could evolve moving forward.

By implementing the core gameplay systems-resource management, base building, and AI-driven population simulation-we turned our initial vision into a working game. The prehistoric-fantasy world of *Tribz* comes to life with villagers collecting resources, constructing a thriving settlement from nothing, and facing the dangers of a land where dinosaurs roam. We successfully met the key objectives laid out in our design: the game provides an engaging strategic experience that is accessible to a broad audience thanks to its intuitive UI and charming art style. We demonstrated these features in our final defense presentation, where *Tribz* ran smoothly, and viewers could watch a tribe survive and grow under player guidance.

Each member of "The Tribe" team contributed their expertise to reach this point, and in doing so, we all grew our skillsets. We learned a great deal about working collaboratively on a software project. Using Agile methods and frequent communication, we managed to keep the project on track and integrate a wide range of components (from pathfinding algorithms to hand-drawn artwork) into one coherent product. We faced and overcame technical challenges such as optimizing AI performance and ensuring consistent game state in our networking tests, which gave us deeper insight into game development pitfalls and best practices.

Importantly, the project taught us about the balance between vision

and practicality. In the beginning, our list of ideas was very long (perhaps even too ambitious), but as we progressed, we became adept at prioritizing features that were essential to gameplay and deferring those that were "nice-to-have." This project management insight-knowing how to cut or postpone features while preserving the core experience-is a lesson we value and will carry into future endeavors. Even with these cuts, we delivered a project that feels complete and fun in its current state, which speaks to good scope management.

Looking ahead, we see *Tribz* not as a static endpoint but as a foundation. In the Future Improvements chapter, we outlined many enhancements, from richer AI to multiplayer co-op, that could be built on this foundation. We believe that the architecture and codebase we designed can accommodate these new features, should the opportunity arise to continue development. In fact, one of the successes of our architecture is its flexibility: the modular design means adding a new resource type or a new building or even a new game mode (like a scenario) would be relatively straightforward, without needing to overhaul what's already built.

On a personal note, completing *Tribz* as our final year project is a source of pride. We managed to create not just a document or a design, but an actual playable game-something we can showcase and perhaps even let others enjoy. It encapsulates the multidisciplinary nature of modern game development: art, music, programming, design, all coming together. This holistic experience is perhaps the most significant outcome of the project for the team. Each of us can point to *Tribz* and say, "we built that together," understanding all the collaboration and effort that went into it.

In conclusion, *Tribz* meets the goals we set out to achieve and has been validated through testing and user feedback as an enjoyable prototype of a game. While there will always be more features to add and polish to apply, the project in its current form demonstrates a successful integration of concept, technical implementation, and teamwork. We are satisfied with the final result and excited about the possibilities it holds for the future. Whether *Tribz* remains a project within academia or grows into something more, it has already fulfilled its purpose as a learning vehicle and a capstone accomplishment for our team. Thank you for following our journey

through this report, and we hope you found *Tribz* as interesting to read about as we did to create it.

Bibliography

[1] Long Jaunt. *Norland* (Video Game, in development). Strategy game blending city-building and narrative, 2022. Official site: https://long-jaunt.itch.io/norland.

[2] Slavic Magic. *Manor Lords* (Video Game, Early Access). A medieval city-building strategy game, 2023.

[3] Ubisoft Blue Byte. *Anno 1800* (Video Game). City-building economic simulation set in the Industrial Revolution, Ubisoft, 2019.

[4] Frontier Developments. *Jurassic World Evolution 2* (Video Game). Theme park simulation with dinosaurs, 2021.

[5] Hart, P. E., Nilsson, N. J., Raphael, B. *A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics*, SSC-4(2), pp. 100-107, 1968.

[6] Unity Technologies. *Unity Official Documentation* (Manual Scripting API). Online: https://docs.unity3d.com/, accessed 2025.

Appendix A

Feature Implementation Status

Table A.1 summarizes the implementation status of major features in *Tribz* as of the final project submission. This provides a quick overview of which features are complete, which are partially implemented, and which are planned for future work.

As shown above, the foundational features of *Tribz* are complete, providing a playable and enjoyable experience. Some features remain in a prototype or partially-done state, offering clear avenues for future development beyond this project.

Feature	**Status**	**Notes**
Resource Management	*Completed*	Players can gather wood and food. Resource inventory updates and resources are consumed by building and maintenance (food consumption by villagers). Balancing of production/consumption rates achieved.
Villager AI (Basic Tasks)	*Completed*	Villagers autonomously perform tasks: gathering, building, and self-care (eating/resting). AI includes dynamic role switching and A* pathfinding for movement.
Construction System	*Completed*	All planned building types can be constructed and have effects. Multi-stage upgrades implemented for several buildings (e.g., houses). Builders deliver resources and construct over time.
Defense Mechanics	*Partial*	Basic combat is in place (villagers and tamed dinos will fight enemies, enemies cause damage to villagers/buildings). However, only a couple of enemy types (one dino species, one human raider) are implemented. No complex raiding AI yet.
Multiplayer Co-op	*Prototype*	Networking code allows two instances to connect and play a mini-game together.
User Interface & UX	*Completed*	Core UI (HUD, build menu, info panels) is implemented and refined based on feedback. Lacks a dedicated tutorial, but contextual tips and a help menu are provided. Some advanced feedback (detailed tooltips, etc.) could be added.
Audio Implementation	*Completed*	Background music and key sound effects are integrated. Volume levels balanced; spatial audio for certain sounds. Additional content (more music tracks, more voice cues) can be added as enhancement.

Table A.1: Implementation status of main features in *Tribz* at project completion.