YRKESHÖGSKOLAN
NOVIA

Development of an Interactive Educational Game for the Brewing Process

Kaleab Shumet Kebede

Degree Thesis

Thesis for a Bachelor of Engineering (UAS) - degree

Degree Programme in Information Technology

Vaasa 2025

BACHELOR`S THESIS

Author: Kaleab Shumet Kebede

Degree Programme and place of study: Information Technology, Vaasa

Supervisor(s): Susanne Österholm, Novia University of Applied Sciences

Title: Development of an Interactive Educational Game for the Brewing Process

---

Date: 18.5.2025 Number of pages: 35 Appendices: 1

Abstract

This thesis describes the development of an Android game using Unity, Blender, and various other tools that transform a complex industrial process into an interactive experience. The game aims to provide a simplified and engaging simulation of the beer brewing process, which is inspired by Bock's Corner Brewery located in Vaasa, Finland. The game intends to entertain and educate brewery visitors, offering them an interactive and informative means of exploring the brewing process.

The game covers the fundamental brewing process from milling to sampling and packaging. It has simplified tap-based mechanics, low-poly isometric visuals, and is optimized for Android devices. Additionally, at the end of the game, a reward system is adopted in which successful players are given a QR code for a discount at the brewery.

This thesis demonstrates how game-based learning can effectively simplify and communicate complex industrial processes through interactive digital media while also serving as a practical example for the development of similar instructional games within various industry sectors.

The development process relies on the preservation of the brew concepts while sustaining engaging gameplay interactions through the balance between the educational aspect and simplicity. The result is an engaging and interactive tool that integrates the brewing process into modern digital learning.

Language: English
Keywords: game-based learning, brewing simulation, unity game development

Table of Contents

List of Figures

List of Codes

Note: All figures/codes are created by the author unless otherwise stated.

1 Introduction

This section highlights Bock's Corner Brewery's story, whose historic legacy and brewing techniques serve as inspiration for the game. It outlines the game's purpose to create an interactive experience that makes the brewing process both educational and practical. By gamifying the complex steps of brewing, users can gain an understanding and insight into the challenges of beer production. The section also covers the game's background, goals, scope, and limitations, focusing on improving knowledge and user engagement.

1.1 The Story of Bock's Corner Brewery

According to A. Maier (personal communication, April 7, 2025), Bock's Corner Brewery was originally constructed by Lahti Bryggeri in the 1860s and is now found at Gerbyntie 18 in Vaasa, Finland. Bock purchased the building and continued to use it for lagering once operations began. When ice-making technology did not exist, the building was a place where lager beers were cooled using ice from the nearby lake.

The brewery was officially opened in 1890. It was then constructed across the street from the bar that is still a part of Bock's Corner Village. In 1892, the first Bock beer was brewed and introduced.

When Hartwall purchased Bock in the 1960s, a growth period for the brewery followed, making it the seventh largest in Finland. However, in 1987, commercial brewing at the Vaasa facility stopped, which meant no more brewing in the area for the following 30 years.

In 2015, Bock's Corner Brewery Oy was established, showing that brewing was returning to the historic site located at Gerbyntie 18. The first beer, which was Pilsner, was brewed on February 8, 2015, and then the brewpub was officially opened a month later, on March 27, 2015. This event marked the return of brewing operations to a site that carries significant historical value.

The name *Bock* comes from the German word for goat and identifies a lager beer that packs a strong flavor in brewing. Many believe that the town of Einbeck, a town near Hannover, is where this style first began. Bock's Corner Brewery Oy continues to uphold the technical processes and cultural heritage of Finland's brewing tradition.

1.2 Background and Motivation

Beer is one of the most widely consumed alcoholic drinks in the world. Beer comes in a variety of styles, from crisp and light lagers to robust and rich stouts. Despite the rise in popularity of beer culture, **many** people are still unclear about the process of making beer, which is called brewing. Brewing is complex, and it passes through multiple industrial processes and quality control. Traditional methods of explaining brewing, such as brochures, films, or guided tours, often fail to provide engaging, dynamic learning experiences, as they can be static and lack interactive components. For breweries, developing a game helps to address this limitation by

making learning interactive, creative, and engaging. The development of such a game helps to simplify and visualize the important steps and processes that are used to produce a beer. This makes learning easily understandable and enjoyable, especially for customers and students who are curious about the steps and techniques of the brewery process and beer production.

1.3 Objective

The main goal is to design and develop an Android game that can educate and engage brewery visitors about the main brewing process through interactive gameplay. The game aims to address the main operations, ranging from the milling process to fermenting to the final sampling and packaging. This game makes the brewery process easy, engaging, and understandable by breaking down each unit operation into a series of simplified, easy-to-visualize operations.

1.4 Scope and Limitations

The game is developed as a simplified and minimal representation of the brewery process, which focuses on providing basic knowledge and engagement rather than an accurate simulation of the process and realism. The game scope includes the primary steps in beer production, which are milling, mashing, lautering, boiling, cooling, fermentation, maturation, and the final **packaging**. Each unit operation is gamified as simple tap-and-hold or multiple-tap game mechanics tailored to mimic real-world actions in a playful format. The game's visuals are built on a 3D isometric perspective with a low-poly style, which is ideal for targeting low-end devices like Android phones and tablets.

The game tries to cover most of the brewing processes, however, it has several limitations in this game, for example, the game does not include the business or marketing part of beer production such as pricing, branding, or distribution; Additionally, for simplicity and to maintain minimality, some operations such as harvesting and malting are not considered. Furthermore, it does not include customization features, such as choosing different ingredients and changing the recipes of the different varieties of beers, which would be part of overly complex and dynamic simulation games.

This game project is not developed as a professional tool for brewers or engineers. It is limited to giving the basic knowledge of the brewery process by way of a simple, interactive, and engaging method for brewery visitors.

2 Theoretical Background

This chapter serves as the basis for developing the interactive brewery game by providing its theoretical framework. It includes a brief discussion of the brewing process, and the relevance of Game-Based Learning on understanding the brewing process, while also focusing on its conventions and how these principles contribute to creating engaging and effective learning environments. Additionally, it briefly explains game design, the development process, and development tools.

2.1 Overview of the Brewing Process

Brewing is a multi-stage biochemical process that converts raw ingredients, primarily malted barley, hops, water, and yeast into beer (Boulton, 2017). Figure 1 illustrates the fundamental processes of brewing.

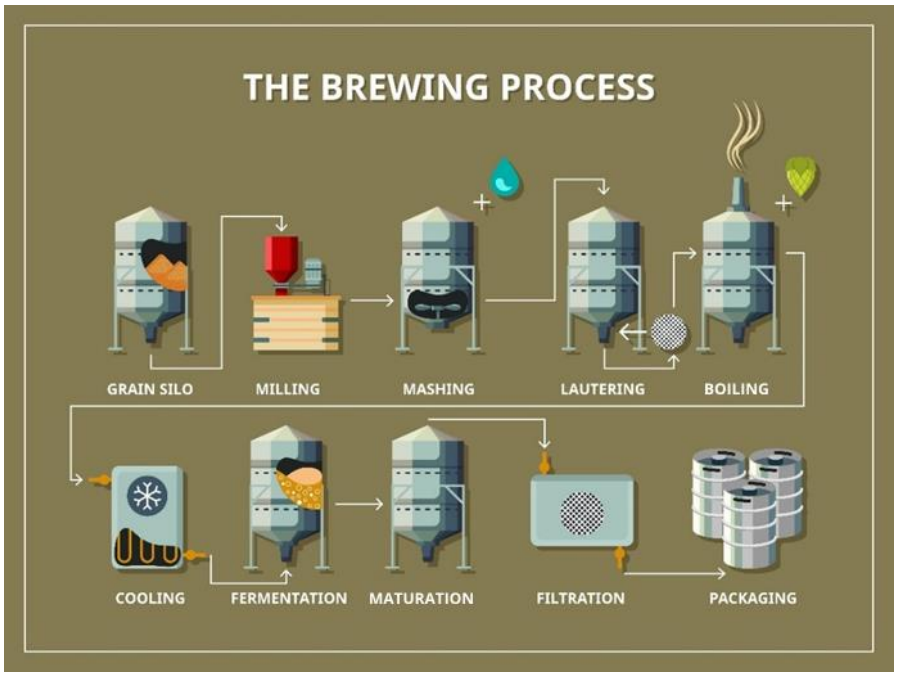

Figure 1: Brewing process flow chart adapted from (Macrovector, 2025)

As illustrated in Figure 1 and detailed by Boulton (2017), the principal processes involved are as follows:

Malting: a process that involves soaking grains in water to germinate them, activating enzymes that break down starches, and then drying the grains to halt germination, preparing them for milling.

Milling: a process of breaking down malted grains by grinding them to reveal starches inside.

Mashing: a process of mixing crushed grains with hot water to activate enzymes that turn starch into fermentable sugars, thereby producing a sweet liquid known as wort.

Lautering: a process of separating the spent grain husks from the liquid wort, thereby clarifying the wort and preparing it for the next stage.

Boiling: a process of boiling the wort, with a fermentable liquid obtained from mashing and lautering, it is boiled with hops to get the appropriate flavor, bitterness, and aroma. Boiling sterilizes the wort, eliminates undesirable chemicals, and evaporates surplus water to concentrate the sugars and other constituents in the wort.

Cooling: a step in the beer brewing process where the wort is rapidly cooled down from boiling to a temperature suitable for yeast to be added.

Fermentation: this process involves the addition of yeast to the cooled wort where the yeast metabolizes the sugars present in the wort, resulting in the production of alcohol and carbon dioxide. This phase is essential for developing the distinctive flavor and aroma of beer.

Maturation: a stage that occurs when fermented beer is transferred to a different vessel for conditioning and carbonation. This phase is crucial for enhancing the flavors and carbonation of the beer, and it can last for several days or even months.

Filtration: the process of removing unwanted particles, yeast, and other suspended solids from beer after maturation, to achieve clarity and stability, and improve the overall quality of the beer.

Bottling or packaging: a process by which conditioned beer is bottled, canned, or kegged for distribution and consumption. This phase is the concluding part of the beer brewing process and is essential for maintaining the beer's freshness and flavor for an extended duration.

2.2 Game-Based Learning (GBL)

A game involves following certain rules and aiming to accomplish a certain goal, while Game-Based Learning (GBL) involves integrating elements of games such as mechanics, dynamics, and storytelling into educational settings to enhance the effectiveness of learning (Plass et al., 2015). GBL raises motivation, problem-solving, and experiential involvement to help learners acquire knowledge and skill development to enable learners to experiment (Al-Khayat et al., 2023). GBL has shown success in teaching procedural knowledge, systems thinking, and complicated workflows in science, technology, engineering, and mathematics (STEM) education (Gui et al., 2023). It replicates real-world events in risk-free surroundings (Gui et al., 2023). For example, games that simulate industrial processes make learners experiment with process parameters, track results, and improve their understanding (Charland, 2014).

The developed brewery game aligns with GBL ideas because it turns the complex brewing process into an interactive, purposeful exercise by gamifying brewery processes such as milling, fermentation, and packaging, which helps users engage with brewery concepts while preserving the integrity of fundamental ideas. According to Kienitz et al. (2024), simulation games were indeed motivating for participants and encouraged their repeated use. This

capacity to capture and maintain learner engagement is a valuable attribute of educational tools.

2.3 Core GBL Principles

Several key principles make Game-Based Learning effective for educational goals. The following are among those principles.

Simplification of Complex Concepts

GBL simplifies and visualizes complex processes, making them in a way that helps users understand them easily (Pacheco-Velazquez et al., 2024). The brewing process includes several complex processes that are difficult to understand. The brewery game simplifies this complex process and helps players understand the basic concept.

Experiential Learning and Procedural Knowledge

Learners get a better understanding when they apply their knowledge in practice, instead of remembering the basic concepts (Deslauriers et al., 2019). Repeatedly performing different steps of brewing helps practice how things are done in the real world. Engaging in this form of learning helps acquire more knowledge and skills.

Intrinsic Motivation

Effective learning games engage players by making them feel accomplished, independent, and purposeful (Li et al., 2024). The brewery game motivates players by making them feel like accomplished brewers, producing high-quality results.

Experimentation Without Risks

Games provide a risk-free environment for learners to test and experiment with, enabling players to try each brewing process and its variables without any risk and danger (Léger et al., 2011).

The design method of the brewery game is based on the GBL principles. It ensures that a better learning outcome is achieved through engaging gameplay rather than traditional learning methods.

2.4 Game Design and Development Process

Game design is the development of fundamental components such as rules, systems, objectives, and overall player experience (Schell, 2008). Game design combines creative thinking and logical planning to produce engaging interactions, challenges, and narratives that appeal to players (Fullerton et al., 2008). A well-crafted game is not merely a form of entertainment; it is a planned experience based on thoughtful decisions about player engagement with the game environment (Rollings & Adams, 2003).

Game design serves as a step-by-step guide that outlines the essential features and their intended functionalities for the development team (Schell, 2008). The main objective is to facilitate the player's advancement from the initial stage to the final challenge interestingly and equitably (Fullerton et al., 2008).

2.4.1 Key Components of Game Design

A comprehensive understanding of a game's internal structure is crucial. According to (Ahmad, 2019), the main components that a game designer should consider include:

- **Game mechanics:** are the fundamental guidelines and systems controlling game operation. Mechanics control what players can do, how challenges are presented, and what results from interactions among players.
- **Objective and Goals:** every game should have well-defined goals for players to aim for. These goals provide motives, purpose, and structure for the players.
- **Player agency:** is the ability of a player to make significant game decisions. Strong degrees of agency help players to feel in control, which improves immersion and satisfaction.
- **Progression and Rewards:** games typically include mechanisms to make progress, such as unlocking levels, acquiring abilities, or earning in-game currency. These rewards reinforce player effort and encourage continued play.
- **Balance and Difficulty:** A well-balanced game offers challenges that are neither too easy nor excessively difficult. Proper tuning ensures players remain engaged and motivated without feeling frustrated or bored.
- **Feedback**: Visual, audio, and haptic responses within the game that communicate the effects of player actions, helping to guide their understanding and subsequent decisions.
- **Immersion and Narrative**: Storytelling, character development, and environmental design contribute to the immersive quality of a game. A strong narrative can foster emotional connections and enrich the player’s experience.

2.4.2 Game Development Processes

For the development of the 3D brewery game, the ADDIE model (Braad et al., 2016) was employed. This model's name is an acronym for its five main stages: Analysis, Design, Development, Implementation, and Evaluation. This structured development process is used fully when experts from various fields are involved, and the design undergoes multiple reviews. According to Braad et al. (2016), the model benefits from testing, as the results can be fed back into the early **stages** of development.

In the Analysis stage, the focus is on understanding the users’ needs, defining the target audience, and assessing the environment in which the game will be implemented. (Braad et al., 2016).

The design phase focuses on the arrangement of the key game mechanics, narrative, content, and various feedback options. All game elements should align with the defined educational goals. User-centered design techniques are often applied during this phase to ensure that user needs are prioritized from the beginning of the design and development process (Braad et al., 2016).

The Development stage deals with building the actual game, usually following the Scrum and other agile approaches. At this point, game designers, educators, and experts in the field should join forces. Braad et al. (2016) emphasize that the developers should focus on creating a game equally fun and educational at this stage of creating the game (Braad et al., 2016).

Implementation involves deploying the game within real-world environments, for instance in classrooms or even healthcare settings. The extent to which this phase succeeds is often determined by the degree the game becomes integrated with its environment (Braad et al., 2016).

Evaluation is the examination of usability, learning outcomes, as well as player experience. Formative testing is included throughout development, while summative assessments are conducted after implementation has finished. The need for both qualitative and quantitative data to be used for validating game effectiveness and for guiding iteration is highlighted by (Braad et al., 2016)

The 3D Brewery Game was developed utilizing the ADDIE model. During the Analysis phase, the focus was on identifying the target audience and defining the game's educational goals. User-friendly mechanics and visual aids were prioritized throughout the Design phase. Unity 3D and Blender were used for the development of the game. Then, it is implemented in real-world settings, with a QR code reward system being included. The Evaluation involved playtesting and feedback so usability could be refined, and the game met its educational objectives.

2.5 Game Development Tools and Technologies

Creating modern video games involves diverse tools and technologies that help streamline development and facilitate multi-platform support. In developing the brewery-themed educational game, a customized selection of industry-standard tools was chosen to match the project's objectives, technical limits, and budget.

2.5.1 Game Engines

Unity and Unreal Engine are popular game engines with a large community, providing foundational tools that support rendering, physics simulation, input processing, and scripting.

Unity, a product of Unity Technologies, is used for the development of 2D, 3D, AR, and VR experiences. Its integration of C# makes learning the engine simple. It has lightweight runtime and memory efficiency, especially ideal for mobile and low-poly games. Its cross-platform capabilities support development for mobile devices, PCs, consoles, and web platforms. The

engine also supports fast workflows through its rich asset store, modular system, and availability of numerous plugins. It is commonly used by Indie developers, professionals, and educators (Unity Technologies, 2025).

Unreal Engine, designed by Epic Games**, is** a recommended tool for producing real-time experiences, simulations, and high-quality 3D games. The engine supports PC, console, mobile, VR, and AR among other platforms. It is popular for its innovative graphics and photorealistic rendering, which is suitable for AAA games and movie productions. Unreal Engine is often preferred by developers seeking scalability, graphical realism, and high performance. Its strong development tools, built-in multiplayer framework, and vast asset market suit both indie projects and big-scale production. Unreal develops sophisticated gameplay systems using C++ and Blueprints (Epic Games, 2025).

For the development of the 3D brewery game, Unity was chosen for its mobile-optimized, lightweight engine, fast prototyping with C#, and extensive Asset Store. Unreal Engine, despite offering high-end graphics, is less appropriate for smaller mobile projects due to its complexity, larger file sizes, and more demanding system requirements.

2.5.2 Tools and Resources for Game Asset Creation and Integration

In addition to the game engine, external tools are used to create and manage 3D models, textures, and user interfaces.

Blender is an open-source 3D modeling suite that supports modeling, texturing, rigging, animation, and rendering. It is commonly used alongside Unity to create custom low-poly assets for mobile games. Blender compatibility with Unity (via FBX or glTF exports) ensures a seamless workflow (Blender, 2025).

FAB.com and Unity Asset Store are utilized to source various ready-made 3D models, textures, and environment assets, significantly accelerating development and ensuring visual consistency (Fab.com, 2025; Unity Technologies, 2025).

Adobe Photoshop is used to design **UI** elements, such as buttons, icons, and overlays. It provides pixel-perfect control and layer-based design, enabling the efficient creation of high-resolution, optimized 2D assets (Adobe Inc, 2025).

Additionally, character animations are imported from **Adobe Mixamo**, which offers a wide range of rigged and animated character motions (Adobe Mixamo Team, 2025). This combination of asset sources allows for rapid iteration and high-quality visual fidelity without the need for extensive custom animation work.

3 Game Design & Implementation

The technical implementation and design logic of the game with a brewery theme are described in this chapter. It explains the main idea of the game, the technologies employed in its creation, the interactive and visual elements produced, and the difficulties encountered in its execution. The goal is to present a thorough understanding of the game's conception, design, and development.

3.1 Concept and Mechanics

The game's central concept is to replicate the beer-making process in an understandable, interesting, and instructive fashion. Acting as a brewer, players guide the beer through all production phases from milling to the final, bottled result. Every stage in the linear, task-driven gameplay reflects a real-world phase of the brewing process.

Players control variables including temperature and time to interact with the brewing equipment. To turn on the equipment, they can touch or multi-tap it; visual indicators let them track development. The game uses a win/lose mechanism whereby the player receives a discount QR code then the brewery verifies and approves the discount at the end.

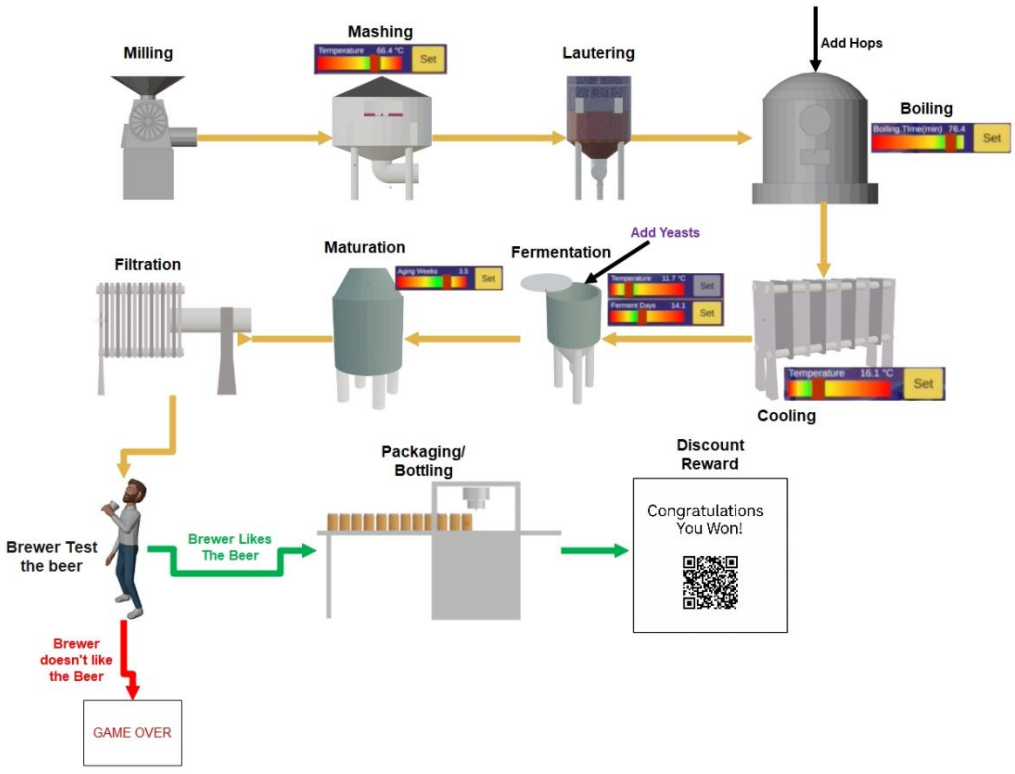

Figure 2: Brewing Game Flow Diagram

As illustrated in Figure 2: Brewing Game Flow Diagram, it begins with milling, where barley is added at the top, ground, and then transferred to the mashing stage. In this stage, the player sets the temperature. In this stage, the player sets the temperature using an interactive slider.

A handle on this slider moves continuously across a color gradient bar. Players must time their click to stop the handle's movement. This click locks in the desired mash temperature. The game then manages this temperature during the mixing of milled barley with water. Next comes lautering, where the grain husks are separated from the liquid wort. The wort then moves to the boiling stage, where the temperature is adjusted and hops are added and boiled. After boiling, the wort is cooled and pumped into the fermentation tank. At this point, yeast is added, and the fermentation process is actively monitored by setting proper temperature and ferment days. Once fermentation is complete, the beer enters the maturation phase for aging. The player sets the required number of weeks for maturation. After this process, the beer is transported to the filtration stage, and following filtration, it enters the evaluation phase.

The Brewer Test is the evaluation method for this game. The brewer takes a sip of the beer and then if the beer is not good enough, the player loses. However, if the brewer approves, the beer is packaged and bottled, and the player receives a QR code with a discount.

This progression guarantees that most of the brewing journey is included/displayed and gamified. In most of the stages, players need to control certain parameters, it is both interactive and informative.

3.2 Implementation of the game

This subsection looks at the implementation of the brewery game, starting from milling to packaging. It shows how both technical and conceptual ideas were realized in a working digital product. This section includes 3D asset creation using Blender, exporting to Unity and the implementation of the game mechanics using C# scripting, and finally the designing of the user interface for the game.

This game project demonstrates various stages of brewing with the help of a variety of mechanics integrating Blender models and Unity's developmental tools which are flexible enough to develop this project. This section explains how everything in the development stack came together to deliver an engaging and educational game environment.

However, this section represents a small part of the overall code used in the entire implementation. Instead of including the entire codebase, only selected snippets are included to emphasize specific mechanical and design choices. This helps the reader from being overwhelmed by technical detail while keeping the focus on the core concepts and features.

3.2.1 Milling

In the milling process, the milling machine is designed in Blender. The mill consists of a cylindrical midsection with a visible radial mill wheel for grinding, a rectangular base housing the internal mechanisms, a funnel-shaped hopper at the top for grain loading, and a cylindrical output chute on the right side where processed grain exits after milling.

Each mill component is modeled independently to produce a modular design. This method allows scripts to be attached to individual components and enables accurate collision detection. Also, it allows components to be easily adjusted and improves development efficiency.

Figure 3: Mill Model.

The mill model was exported as an FBX file that included default materials. Then the FBX model was imported and added to the Unity scene, and saved as a Prefab. A Prefab (short for "prefabricated object") is a saved GameObject complete with its components, properties, and child objects. Saving as a prefab makes it simple to reuse and make consistent updates throughout the development process.

As shown in Figure 4: Mill Components in Unity, the mill is a parent of multiple components, it has a box collider attached, and its script is called “Milling”. This script is used to handle touch or click events as shown in Code 1: Mill Click Handler. When the mill is clicked the milling process starts and all the milling components such as the rotator, GroundBarleyGenerator, and BarleyHider start their actions.

Figure 4: Mill Components in Unity.

```
0 references
void OnMouseDown()
{
    if (!isEnabled) return;
    if (handPointer.IsActive())
        handPointer.HidePointer();
    if (UIInteractionGuard.IsPointerOverUI) return;

    // Set isClicked to true when the GameObject is clicked
    if (!OverClicked) isClicked = true;
}
```

Code 1: Mill Click Handler.

Code 2: Outline Animate controls the outline effect animation, which serves as a visual hint for the player to interact with the GameObject. It is a custom script that relies on an external script called "Quick Outline" (Nolet, 2022), available for download from the Unity Asset Store.

```
0 references
void Start()
{
    outlineAnimate = new OutlineAnimate(gameObject);
}
```

Code 2: Outline Animate.

The mill contains a child component, GroundBarleyGenerator, which has an associated script and it spawns ground barley into the mill chute upon clicking the mill.

```
void Start()
{
    Transform grandParentTransform = transform.parent?.parent;
    if (grandParentTransform != null)
    {
        milling = grandParentTransform.GetComponent<Milling>();
        if (milling == null)
        {
            Debug.LogError("Milling script not found on grandparent G
        }
    }
    else
    {
        Debug.LogError("Grandparent GameObject not found");
    }

    StartCoroutine(SpawnGrains());
}
```

Code 3: Snippet for grain spawning in the mill chute.

```
IEnumerator SpawnGrains()
{
    bool shouldWait = true;
    int spawnedCount = 0;
    while (spawnedCount < spawnCount)
    {
        // Pause spawning if milling is null or milling.IsClicked
        while (milling == null || !milling.IsClicked())
        {
            if(!shouldWait) shouldWait = true;
            yield return null; // Wait for the next frame
        }

        if(shouldWait){
            shouldWait = false;
            yield return new WaitForSeconds(1f);
        }


        for (int i = 0; i < grainsPerInterval && spawnedCount < sp
```

Code 4: Snippet for the SpawnGrains.

According to Code 3, GroundBarleyGenerator starts coroutine functions called SpawnGrains. This coroutine function waits until the GameObject is clicked as shown in Code 4. Then it spawns a ground barley prefab inside the chute of the mill.

The mill GameObject includes components such as "Rotator," "BarleyHider," as well as "ProgressCanvas." When the mill is clicked, it allows its rotator to rotate, which gives a visual cue for the mill being in operation. BarleyHider, which is placed at the bottom part of the mill funnel, hides the barley so it appears as if entering the mill.

```
void Update()
{

    if ( milling != null && milling.IsClicked())
    {
        transform.Rotate(Vector3.up  * speedFactor

    }
}
```

Code 5: Mill Rotator code snippet.

```
if (!milling.IsClicked())
    return;
if (barleyColliders.Count > 0)
{
    timer += Time.deltaTime;
    if (timer >= interval)
    {
        for (int i = 0; i < batchSize && barleyColliders.Count > 0;
        {
            Collider barleyCollider = barleyColliders.Dequeue();
            barleyCollider.gameObject.SetActive(false);
        }
        timer = 0f;
    }
}
```

Code 6: Barley Hider code snippet.

On the top of the mill, an empty GameObject with its script is created, the script spawns the raw barleys on the top of the mill funnel, making it ready to be milled.

The other component of the mill is ProgressCanvas, a UI component that displays the progress of the operation. The progress operation is time-based, and it is defined to run for specific time intervals.

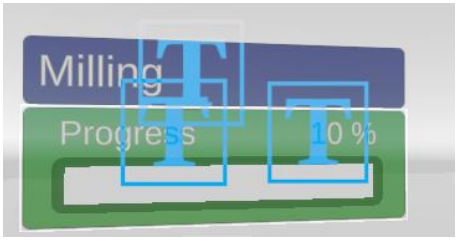

Figure 5: Progress Canvas.

```
void Update()
{
    if (isClicked && CurrentClickTime < TotalClickTime)
    {
        CurrentClickTime += Time.deltaTime;

        float currentProgress = CurrentClickTime / TotalClickTime;
        progressBar.SetValue(currentProgress, true);
    }
```

Code 7: Updating the progress value.

3.2.2 Conveyor and Pipes

The conveyor belt is an essential element of the brewery, which connects the mill to the mash tun, allowing ground barley to pass along its top surface. This is an automated process in which the ground barley is transported and enters the mash tun.

The conveyor model features a simple gray frame supporting a distinctive red belt surface, creating visual contrast in the brewery setup. The model is created in Blender and exported as an FBX file with default materials.

A script attached to the conveyor model detects ground barley colliding with it. This script then moves the colliding barley GameObjects along the conveyor's surface at a steady velocity, as illustrated by the logic in Code 8 which modifies the barley's Rigidbody velocity.

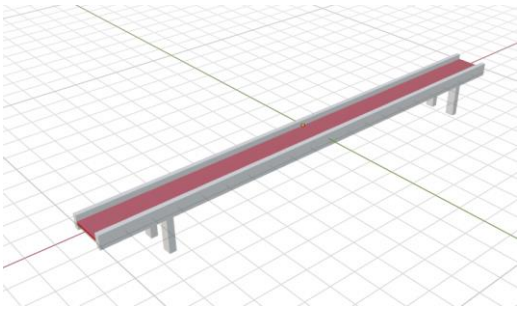

Figure 6: Conveyor model.

```
public float speed = 5f;

private void OnCollisionStay(Collision collision)
{
    if (collision.gameObject.CompareTag("GroundBarley"))
    {
        Rigidbody rb = collision.gameObject.GetComponent<Rigidbody>();
        if (rb != null)
        {
            rb.linearVelocity = new Vector3(speed, 0, 0);
        }
    }
}
```

Code 8: Snippet used to move grains on the top of the conveyor.

Pipes are very important in the liquid transport system of the brewery game and connect different brewing vessels for easy transfer of liquids between equipment. They move important liquids like wort, and beer, ensuring that the brewery runs smoothly from beginning to end.

A pipe model is created in Blender with a cylindrical design and an angled elbow, typically used for piping in the brewery process of the game. The elbow is a separate component that can be customized in the game engine. Its transparent appearance provides a visual clue that the fluid is flowing inside the pipe.

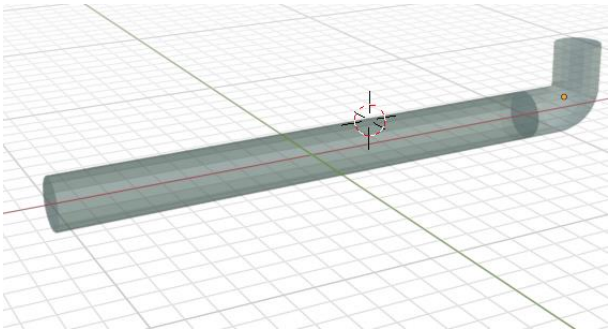

Figure 7: Pipe model.

In Unity, the pipe model gets imported and then Pipe System GameObject is added as a child, where this PipeSystem is a particle effect together with a script named ParticlesPath which simulates fluid flow. The script moves particles along a defined path. This path is typically determined by a series of child GameObjects (waypoints), and the script updates particle positions by interpolating between these waypoint positions. The pipe system supports play, pause, and stop features, also it can auto-pause when reaching the path's end. This script made it possible to visualize and control fluid flow throughout the brewery process.

3.2.3 Mashing

Following the milling process, the ground barley is moved to the Mash tun where the mashing process is done. The mash tun is used to mix water with the ground barley, it is designed in Blender to have a cylindrical body and transparent conical top. A rectangular window reveals the water inside. This opening is where the ground barley is added to the water, while a drainage pipe extends from the bottom. Mixers are also present inside, used to mix the ground barley with water.

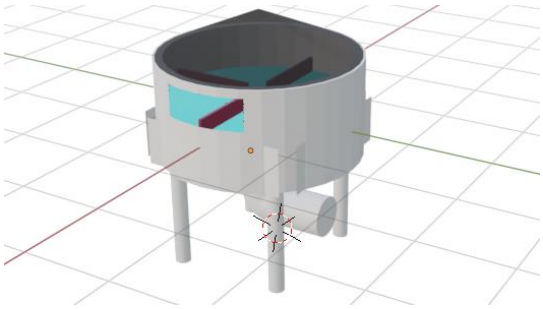

Figure 8: Mash tun model.

The model is imported to Unity, and a box collider is attached to key components of the mash tun. The mash tun has a script called Mashing, which is used to control the mashing process. The script has similar functionality to the previous milling script; it listens to click events and controls the outline animation.

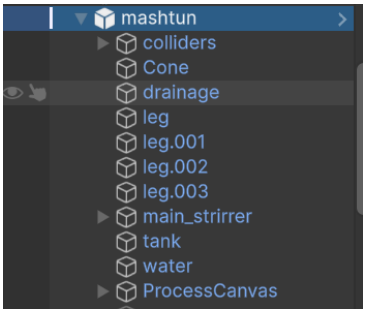

Figure 9: Components of mash tun.

```
void OnMouseDown()
{
    if (!isEnabled) return;
    if (UIInteractionGuard.IsPointerOverUI) return;
    if (!OverClicked) isClicked = true;
}

void OnMouseUp()
{
    isClicked = false;
}
```

Code 9: Click handler for mash tun.

The mash tun has different components which have different tasks. When the mash tun is clicked, its main script ("Mashing.cs") detects this event and then initiates the operations of its relevant child components. One of the components of the mash tun is main_stirrer which is a set of GameObjects with their colliders. As illustrated in Code 10 When the mash tun is clicked, main_stirrer starts rotating and mixes the barley with the water.

```
void Update()
{
    if (mashing != null && mashing.IsClicked())
    {
        // Rotate the GameObject around the Z-axis
        transform.Rotate(Vector3.forward, rotationSpeed * Time.deltaTime);

    }
}
```

Code 10: Snippet used to rotate mash tun stirrer.

Another component inside the mash tun with its script is “water”. When the mash tun is clicked, the stirrer rotates (Code 10), causing the water to appear to swirl and mix with barley. Eventually, the water changes its color from blue to brown as seen in Code 11, which gives a better visual of mixing water with barley.

```
IEnumerator ChangeColorOverTime()
{
    isChangingColor = true;

    while (elapsedTime < colorChangeDuration)
    {
        waterMaterial.color =
        Color.Lerp(
            startColor, endColor,
                elapsedTime / colorChangeDuration);
        yield return null;
    }

    // Ensure the final color is set
    waterMaterial.color = endColor;
    isChangingColor = false;
}
```

Code 11: A Snippet used to change the color of water while mixing.

The mash tun has a ProcessCanvas as displayed in Figure 10, It is a user interface component that includes texts, buttons, and a slider that enables players to set parameters such as temperature and time. This UI component has an indicator that is constantly moving and oscillating through a color gradient bar. The main goal is to click the “Set” button when the indicator is in the green (optimal) zone. The value is recorded and used as evaluation criteria at the end of the brewing process.

The UI slider is customized by the “CustomSlider” script, as it makes it possible for a dynamic gradient fill to change the color of the slider according to value ranges (red, yellow, green), and controls the indicator to oscillate back and forth.

Figure 10: UI Slider used to set parameters.

```
3 references
private Texture2D GenerateGradientTexture(int width, int height)
{
    Texture2D texture = new Texture2D(width, height, TextureFormat.RGBA32, false);
    float greenMidpoint = (greenRangeMin + greenRangeMax) / 2f;
    Color colorGreen = Color.green;
    Color colorYellow = Color.yellow;
    Color colorRed = Color.red;
    float normGreenStart = Mathf.InverseLerp(minValue, maxValue, greenRangeMin);
    float normGreenEnd = Mathf.InverseLerp(minValue, maxValue, greenRangeMax);
    float normGreenMid = Mathf.InverseLerp(minValue, maxValue, greenMidpoint);
    for (int x = 0; x < width; x++)
    {
        float t = (float)x / (width - 1);
        Color pixelColor;

        if (t < normGreenStart)
        {
            float localT = Mathf.InverseLerp(0, normGreenStart, t);
            pixelColor = Color.Lerp(colorRed, colorYellow, localT);
        }
        else if (t < normGreenMid)
        {
            float localT = Mathf.InverseLerp(normGreenStart, normGreenMid, t);
            pixelColor = Color.Lerp(colorYellow, colorGreen, localT);
```

Code 12: A snippet used to generate gradient texture for UI Slider.

As shown in Figure 11, at the bottom of the mash tun there is a drainage connected to a pipe system. Once the mashing process has finished, the wort is released through the pipe, simulating fluid flow inside. The wort moves through the pipe and accumulates at the end of the pipe for a short period, waiting for the next process, which is lautering.

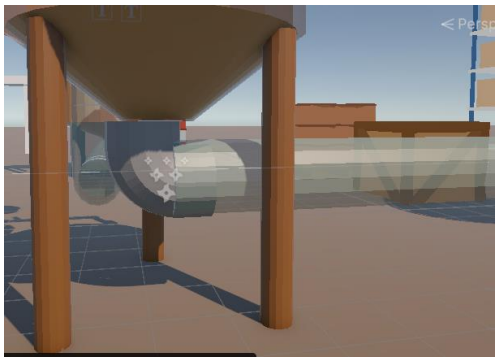

Figure 11: Connection of mash tun drainage to a pipe system.

3.2.4 Lautering

Lautering is a process in which the mixed barley and wort are separated, this operation is performed in an equipment called Lauter Tun. For this game project, a lauter tun is modeled as a cylindrical vessel with a brown lower section that can hold the wort. It comprises vertical bars for the agitation system and a perforated false bottom mesh for filtration. Also, a drainage pipe runs out from the base.

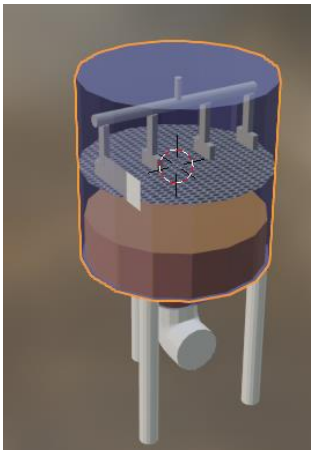

Figure 12: Lauter tun model.

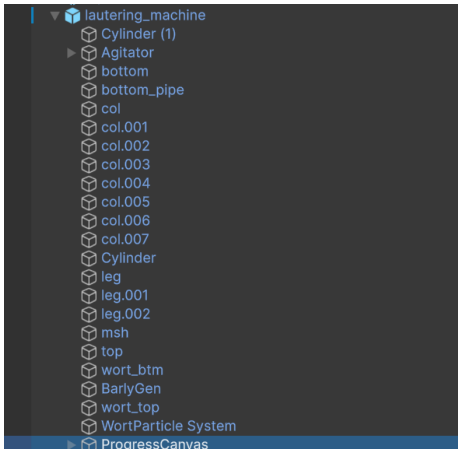

Figure 13: Lauter tun components.

The lauter tun is imported to Unity, and a box collider and script are attached to handle click events, when the lauter tun is clicked it activates its components to do their operations. For example, when the lauter tun is clicked, the wort (The PipeSystem particles effect) starts flowing into the vessel. It passes through the perforated false bottom (mesh) and the agitator starts rotating, providing a better visual effect.

The implementation of most component scripts for the Lauter tun follows similar patterns to those previously described. However, the lauter tun includes an additional feature: as the wort enters, it is filtered through the false bottom and collects at the base of the tun. At this point, the visual representation of the wort level begins to rise, simulating the filling process. The effect of filling is made by scaling the wort_top GameObject that is inside the lauter tun as provided in Code 13.

```
if (lauterTun.IsClicked())
{
    // Increase the local y-scale over time, clamped to maxYScale.
    float newYScale = Mathf.Min(
        transform.localScale.y + growthRate * Time.deltaTime,
        maxYScale
    );
    if (newYScale > maxYScale)
        return;

    // Update the scale.
    transform.localScale =
        new Vector3(initialScale.x, newYScale, initialScale.z);

    // Adjust the object's position so that its bottom remains fixed.
    float newCenterY = initialBottomY + newYScale * 0.5f;
    transform.position =
        new Vector3(transform.position.x, newCenterY, transform.position.z);
}
```

Code 13: A snippet to simulate the wort-filling process.

At the end of the lautering process, the wort is automatically discharged through its drainage pipe and moves to the next stage.

3.2.5 Boiling

Boiling is a process where the wort and the hops are heated and boiled inside a vessel. In Blender, a boiling vessel with a cylindrical body and open dome is designed as shown in Figure 14.

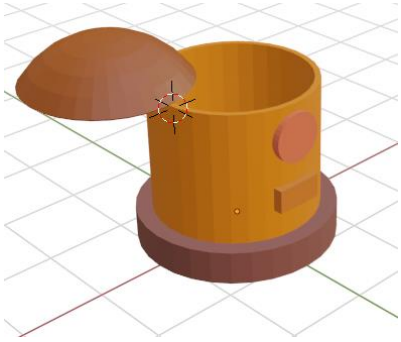

Figure 14: Boiler model.

The model is imported into Unity, and then a box collider and script are attached. The boiler has an additional Animator component and script code which is used to play animation as shown in Code 14. Additionally, as shown in Figure 15, an animation controller is created to manage and control the animation, this animation is used to automatically close the boiler after the player adds the hops.

```
public void CloseBoiler()
{
    if (isOpen)
    {
        PlayAnimation("boiler_close");
        isOpen = false;
    }
}

// Method to play animation
public void PlayAnimation(string animationName)
{
    if (animator != null)
    {
        animator.Play(animationName);
    }
}
```

Code 14: A snippet used to play boiler animation.

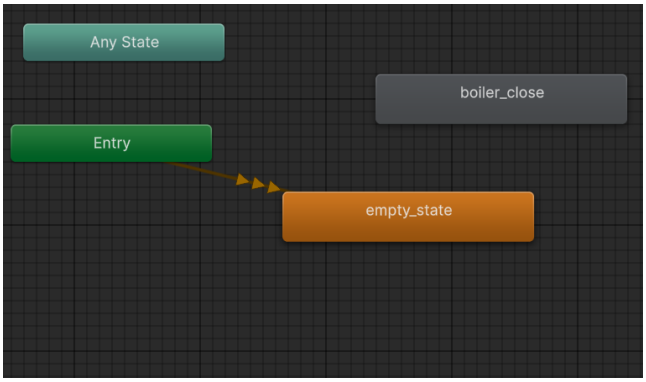

Figure 15: Boiler animation controller in Unity.

In the game, hops are added to the boiler using a shooting mechanism from the cannon, which makes the gameplay more engaging and dynamic. The cannon was modeled in Blender as displayed in Figure 16, and then imported into Unity for use. The cannon has its particle system along with a script that controls the firing mechanism. Code 15 shows that when tapping the cannon, hops will be shot and added to the boiler. Additionally, the cannon includes a progress bar and outline animation to give a visual cue for the players to interact with it.

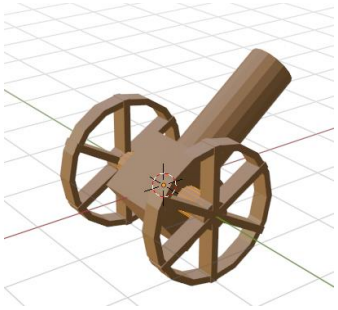

Figure 16: Cannon Model.

After adding hops to the boiler, the boiler will be closed automatically. The player then sets the boiling time using the color gradient UI slider and clicks the boiler to start the boiling process. Once the process is complete, the wort drains through the pipe and moves to the next stage.

```
void OnMouseDown()
{
    if(boiler.handPointer != null)
        if(boiler.handPointer.IsActive())
            boiler.handPointer.HidePointer();


    if (clickCount < requiredClicks)
    {
        gunPS.Emit(1);
        if (++clickCount >= requiredClicks)
        {
            gunPS.Stop();
            StartCoroutine(CloseBoilerAfterDelay(1f));
        }
        float progress = (float)clickCount / requiredClicks;
        hopGunSlider.SetValue(progress, true);
    }
}
```

Code 15: A snippet used to shoot hops particles.

3.2.6 Cooling

Cooling reduces the temperature of the wort, making it suitable for fermentation. It is held in the heat exchanger commonly known as a cooler. Figure 17 shows a Blender cooler model with parallel vertical plates connected by a horizontal bar.

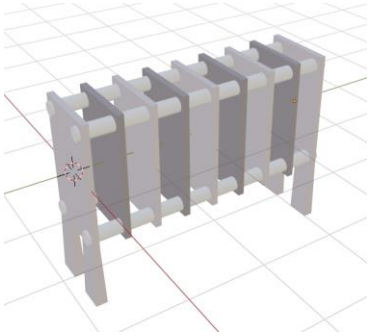

Figure 17: Cooler model.

The model is imported into Unity and then a collider and script are attached. The script controls the outline effect animation and manages the cooling process, including triggering the fluid flow animations into and out of the cooler via the connected pipe systems. The cooler contains a UI component which is used to set temperature as a parameter. The player sets the temperature and clicks the GameObject to complete the cooling process.

3.2.7 Fermentation

Fermentation is a process that changes the wort's sugar into alcohol and other byproducts. This process takes place inside a fermenter. Figure 18 shows a fermenter model consisting of a cylindrical vessel with an open top for yeast addition.

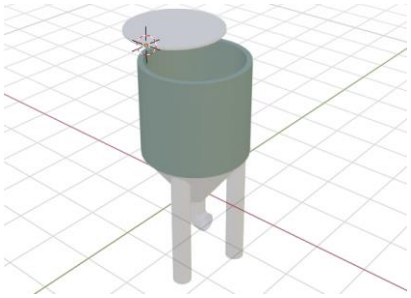

Figure 18: Fermenter Model.

Yeast should be added to the fermenter to make it operational. The same technique that is used on the boiler can be implemented, so the cannon prefab can be used again. However, the particle effect should be adjusted to represent the yeast particles. The fermenter also needs an Animator component which is used to close the fermenter. Furthermore, the fermenter consists of UI components to set parameters such as temperature and ferment days.

An additional feature for the fermenter is a day-night simulation as shown in Code 16. This simulation can be achieved easily by cycling the floor color between light and dark shades.

After the player added yeast and clicked the fermenter, the day and night cycle continued to iterate.

```
private void UpdateDayNightCycle()
{
    if (planeRenderer == null) return;

    float progress = (CurrentClickTime / TotalClickTime) * 100f;
    Debug.Log("Progress: " + progress);

    if (progress <= 0f || progress >= 100f)
    {
        planeRenderer.material.color = lightColor;
        return;
    }

    float cycleProgress = Mathf.PingPong(progress + 10f, 20f) / 20f;

    Color newColor = Color.Lerp(lightColor, darkColor, cycleProgress);

    planeRenderer.material.color = newColor;
}
```

Code 16: A snippet used to simulate a day-night cycle.

Following the completion of the fermentation process, the wort moves to the next stage, which is called maturation.

3.2.8 Maturation

Maturation is the conditioning period where beer develops its final flavors and carbonation while aging in tanks. The maturation vessel model shown in Figure 19 is designed in Blender, featuring a cylindrical body with a conical top, supported by legs.

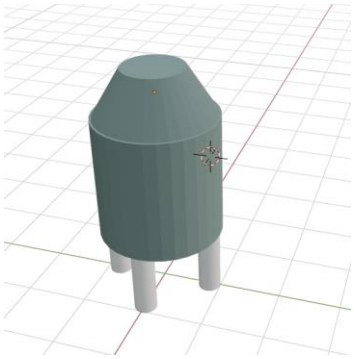

Figure 19: Maturation vessel model.

The maturation vessel has a parameter for the aging duration (set in weeks) that players can adjust. It also reuses the day-night cycle simulation logic from the fermenter to visually represent the passage of aging time.

3.2.9 Filtration

Filtration is the process of removing suspended solids and yeast from the beer. A filter is designed using Blender as shown in Figure 20, containing alternating plates and frames set on a central rod.

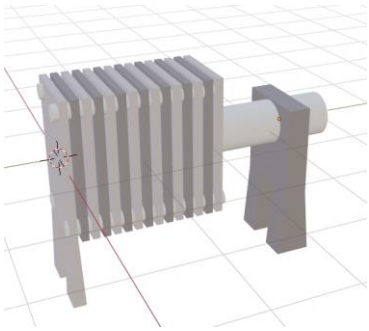

Figure 20: Filter model.

The model is imported in Unity, and then a box collider and filtration script are attached to it. Most of the code and functionality in the filtration script are like the cooling script. The only difference is that this process does not record any parameters such as temperature. The beer passes through the filter and then becomes ready for the test.

3.2.10 Testing the Beer

Testing the beer is the final stage of the brewery game. A **sampling tank** is designed as shown in Figure 21, allowing the brewmaster to pour and drink the beer to evaluate its quality, which is determined by the brewing parameters set by the player. The brewmaster determines whether the batch meets the desired standards or not.

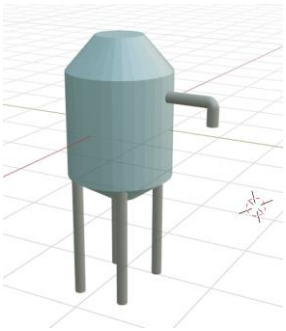

Figure 21: Sampling tank model.

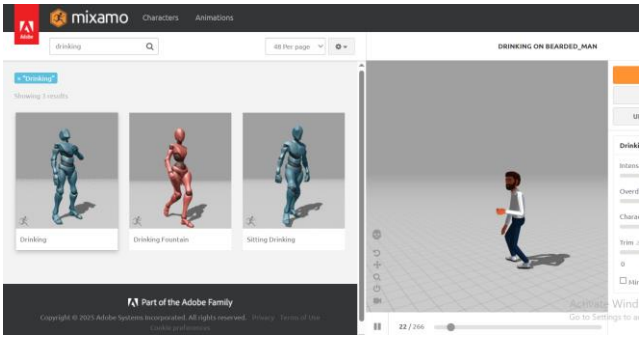

Figure 22: Animations from Mixamo.

Additionally, the brewmaster character model was acquired from **Fab.com**(Agor2012, 2025) to represent the brewmaster during the beer testing stage. The character did not include a drinking animation, therefore, Mixamo animations were applied instead as shown in Figure 22. Blender was used to modify these animations and to add other actions, such as dancing and yelling. Ultimately, the animated character was exported as FBX.

The character was imported and placed beside the sampling tank so the drinking animation could be implemented. Then, an animation controller(shown in Figure 23) is created to manage and transition between the character animations smoothly during gameplay.

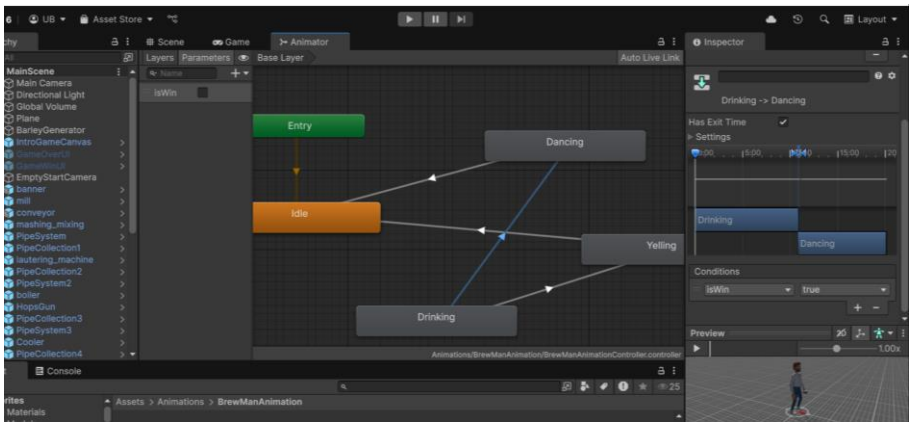

Figure 23: Animator Controller in Unity for the Brewmaster character.

An additional script (Code 17) is attached to the Brewmaster character. This script manages the playback of the character's animations.

```
public class BrewMaster : MonoBehaviour
{
    6 references
    private Animator animator;
    0 references
    void Start()
    {
        animator = GetComponent<Animator>();
        if (animator == null)
        {
            Debug.LogError("Animator component not found on BrewMaster!");
        }
    }
    0 references
    public void PlayAnim(string animationName)
    {
        if (animator != null)
        {
            animator.Play(animationName);
        }
    }
    0 references
    public void SetWinAnim(bool isWin)
    {
        if (animator != null)
        {
            animator.SetBool("isWin", isWin);
        }
    }
```

Code 17: A snippet used to animate Brewmaster's character.

```
if (GameController.Instance == null)
{
    Debug.LogError("GameController.Instance is null. Make sure GameControl
    return;
}

bool isWin = GameController.Instance.EvaluateBreweryParameters();

if (brewMan != null)
{
    brewMan.SetWinAnim(isWin);
}
StartCoroutine(DelayAction(2f, () =>
{
    if (brewMan != null) brewMan.PlayAnim("Drinking"); // 10
}));

StartCoroutine(DelayAction(3f, () =>
{
    if (beerPouringParticleSystem != null) beerPouringParticleSystem.Play(
}));
```

Code 18: Sampling tank code that manages particle system and Brewmaster animation.

The Sampling Tank contains a particle effect as its child, for simulating the beer pouring effect. To integrate this particle effect with the character animations, a custom SamplingTank script was created and attached to the sampling tank as shown in Code 18.

A static instance of a class GameController is used to manage the entire game flow including user interfaces. All game parameters are stored within this controller, making it easier to control the game. The method EvaluateBreweryParameters shown in Code 19 is included in this GameController to assess whether the player has set the correct parameters and met the required standards.

```
public bool EvaluateBreweryParameters()
{
    float mashingMinTemp = 52f, mashingMaxTemp = 78f;
    float boilingMinuteMin = 60f, boilingMinuteMax = 90f;
    float coolingMinTemp = 5f, coolingMaxTemp = 20f;
    float fermentationMinTemp = 6f, fermentationMaxTemp = 20f;
    float minFermentationDays = 7f, maxFermentationDays = 14f;
    float minMaturationWeeks = 2f, maxMaturationWeeks = 4f;
    LogProcessParameters();

    if (ProcessParameters["Mashing_Temperature"] < mashingMinTemp || ProcessParameters["Mashing_Temperature"] > mashingMaxTemp)
    {
        return false;
    }
    if (ProcessParameters["Boiling_Minute"] < boilingMinuteMin || ProcessParameters["Boiling_Minute"] > boilingMinuteMax)
    {
        return false;
    }
    if (ProcessParameters["Cooling_Temperature"] < coolingMinTemp || ProcessParameters["Cooling_Temperature"] > coolingMaxTemp)
    {
        return false; // Fail: Cooling Temperature out of range
    }
```

Code 19: Game controller snippet to evaluate brewing parameters.

After the brewmaster drinks the beer, if EvaluateBreweryParameters returns true, he performs a dance animation and the player wins. Otherwise, a yelling animation is played, indicating that the player has lost.

3.2.11 Packaging or Bottling

The last stage in the brewing process is packaging the final product. For this stage, a bottling machine model is created in Blender as seen in Figure 24, and then brought into Unity. The packaging model acts as a visual indicator. Therefore, if the player successfully brews quality beer, this packaging stage follows. It lets the players know that winning will bring them to the final phase in the process.

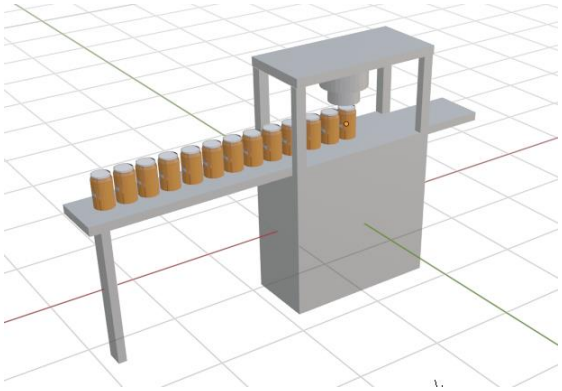

Figure 24: Bottling machine model.

3.2.12 Game Environment

The game environment shown in Figure 25 was created using assets from the Unity Asset Store to reduce the development time. To attain a factory or warehouse aesthetic, the Low Poly FPS Map Vol. 4 (JustCreate, 2023) asset was selected due to its key components such as boxes, shelves, barrels, and containers, being ideal for the creation of an industrial-style setting.

Figure 25: Game environment.

3.2.13 Game UI Elements

The 3D brewery game UI was designed in Adobe Photoshop to achieve visual consistency and appeal. Figure 26 shows the implemented three screens, which are Main UI, Game Over UI, and Game Success UI. The Main UI serves as the starting point from where gameplay can be initiated. Game Over UI appears when the player loses, showing a message to retry. The Game Success UI displays when the player wins, containing a reward coupon print button.

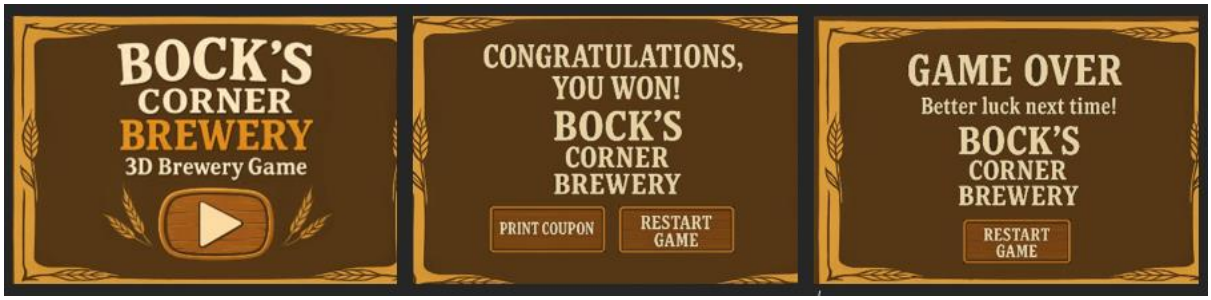

Figure 26: Game Screens.

3.2.14 Game Camera

The Isometric Camera script provides a fixed top-down view for a brewery simulation game, so players can observe the brewing process from a steady perspective. The camera changes its position for every brewing stage according to predefined camera offsets as shown in Code 20. When the player proceeds to the game process, the camera transitions to the next offset. This makes the camera flow smoothly and improves the transition between unit operations. Code 21 is used for this smooth camera flow and transition.

```
public class IsometricCamera : MonoBehaviour
{
    public Transform target;  // Assign the player in the Inspector
    private float smoothSpeed = 1f;  // Adjust for smoother movement

    List<Vector3> predefinedOffsets = new List<Vector3>
{
    //new Vector3(-17.2f, 21.5f, -47.6f), // Empty Camera Start 0
    //new Vector3(-19.2f, 15.6f, -41.7f), // Banner   1
    new Vector3(-16.9f, 14.39f, -22.4f), // Milling 2
    new Vector3(-8.15f, 14f, -22.4f), // Mashing 3
    new Vector3(6.29f, 3.81f, -20.8f), // Lautering 4
    new Vector3(22.76f, 0.23f, -25.8f), // Boiling 5
    new Vector3(42.4f, 1f, -21.9f), // Cooling 6
    new Vector3(47.9f, 0.23f, -39.7f), // Fermenting 7
    new Vector3(65.0f, 0.23f, -39.7f), // Conditioning 8
    new Vector3(81.5f, 0.23f, -39.7f), // Filtration 9
    new Vector3(102.5f, 0.23f, -39.7f), // SimpleStorageTank 10
};
    public Vector3 offset = Vector3.zero; // Initialize with zero
```

Code 20: Predefined offsets of different brewery components

```
void LateUpdate()
{
    if (target == null) return;

    if (isTransitioning)
    {
        transitionTime += Time.deltaTime;
        float t = Mathf.Clamp01(transitionTime / transitionDuration);
        offset = Vector3.Lerp(startOffset, targetOffset, t);

        if (t >= 1f)
        {
            isTransitioning = false;
        }
    }

    Vector3 desiredPosition = target.position + offset;
    transform.position
        = Vector3.Lerp(transform.position, desiredPosition, smoothSpeed * Time.de
}
```

Code 21: Smooth Camera flow.

3.2.15 QR Code Generation

Players get a discount QR Code from the brewery if they set the proper parameters and win the game. This QR code is generated by encrypting a secret string with a personalized message. Unity does not natively support QR code generation. Therefore, this project utilizes the ZXing.Net library (Jahn, 2025). With this library, a QR code is generated and then rendered onto a UI canvas that contains an empty RawImage (as seen in Figure 27), This rendered QR code is subsequently captured using a dynamic temporary camera and converted to a PNG image. Currently, printing functionality exists only for Android, where it utilizes native components for Android such as PrintHelper for the printing process.

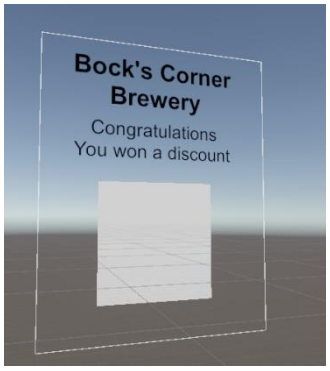

Figure 27: Canvas used to render QR code.

3.3 Implementation of QR Code Verification App

A separate Android app was created to validate QR codes given to the winners. To create the Android app, Android Studio with Java programming is used. The app reads QR codes using the device's camera via a third-party scanning library called "Code Scanner" (Budiyev, 2025). Every QR code contains encrypted string data. This data is decrypted in the validation app using a secret key that is also embedded within the game (for encryption) and the validation app (for decryption). After scanning and decrypting (as highlighted in Code 22), it checks for specific suffix and verifies if the win is legitimate. Every scanned code is entered into a local database (using the Android Room library) and duplicate scans are automatically invalidated

with a warning message. This guarantees an offline-capable and tamper-proof mechanism to prevent fraud.

```
1 usage
private void handleScannedResult(String scannedText) {
    String decrypted = decryptString(scannedText, PASSWORD);
    if (decrypted != null && decrypted.endsWith("[redacted]")) {
        validateAndStoreCode(scannedText);
    } else {
        runOnUiThread(this::showInvalidCodeDialog);
    }
}
1 usage
private void validateAndStoreCode(String scannedText) {
    new Thread(() -> {
        ScannedCodeDao dao = db.scannedCodeDao();
        ScannedCode existing = dao.getCode(scannedText);

        runOnUiThread(() -> {
            if (existing != null) {
                showAlreadyScannedDialog();
            } else {
                insertCodeAndShowWinner(scannedText);
            }
        });
```

Code 22: Java code used to validate scanned QR code.

3.4 Development Challenges and Solutions

During game development, significant challenges need to be addressed. The primary issue was turning the real brewing process's complexity into a playable experience. For this challenge, experts from Bock's brewing company gave support in choosing which aspects to include and not to consider. Finally, the complex operations were transformed into simplified visual representations, allowing players to engage with the core of brewing steps without the complexity.

Another challenge arose during playtesting: users sometimes found it difficult to identify which interactive elements to select. To address this, guidance systems were introduced. These systems utilize animated hand icons and outline animations to offer clear visual cues and suitable guidance, making it easier for players to interact with the game.

4 Results and Discussion

This chapter describes the results after the development of the 3D Brewery Game, which aims to enhance users' understanding of brewing. As described in Section 1.3, the principal aim of this digital simulation game is to be interactive, engaging, and educational. The results reported in this chapter cover the main aspects of the game, including the implemented features and its alignment with Game-Based Learning (GBL) Principles.

4.1 Result of 3D Brewing Game

The game simulates the key processes of the brewery, allowing the player to engage with interactive elements that provide progress updates and visual feedback.

At the start of the game, the main user interface appears, presenting a play button to start the game as shown in Figure 26 in section 3, Once the button is clicked, the actual game environment is displayed. This environment features a low-poly and industrial aesthetic that is populated with barrels, shelves, and brewing equipment. To eliminate any potential confusion for the players, the game employs animated hand icons and an outline effect that gives clear guidance as to which object the player needs to interact with.

4.1.1 Milling

In this 3D brewery game, the milling gameplay is provided in Figure 28. The player begins beer production by operating a grain mill. To operate the mill, the player must tap on it. When the player taps the mill, grains are loaded into the mill, the mill wheel starts rotating, and at the bottom of the mill chute, the ground barley is discharged into the conveyor. The progress bar shows the player how much of the task has been completed, and the conveyor carries the ground barley to the mash tun.

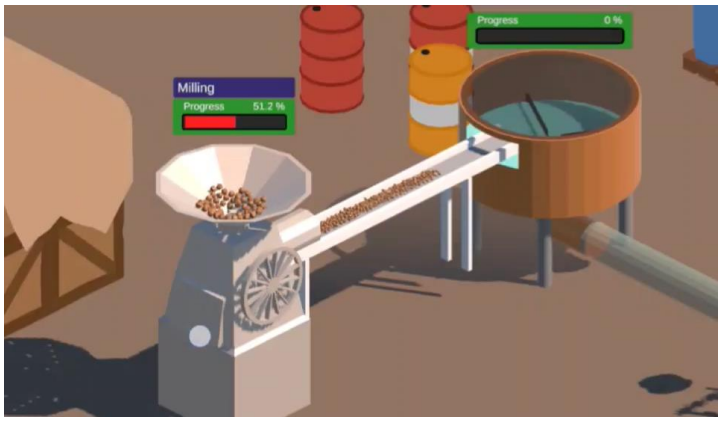

Figure 28: Milling process gameplay.

4.1.2 Mashing

After milling, as seen in Figure 29, the ground barley grain is added to the mash tun. The player sets the mash temperature on a color-coded slider, which has an indicator that oscillates left and right. The player must focus on the slider and click a button marked "Set" when the

indicator is in the optimal zone (green region). Next, the player must tap the mash tun to start the mixing process. A progress bar shows the progress of the mashing operation. Finally, the wort is discharged and transferred to the next stage.

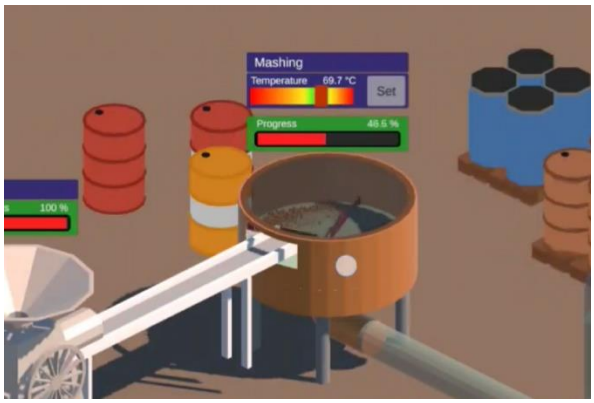

Figure 29: Mashing process gameplay.

4.1.3 Lautering

The wort is transferred to the lauter tun, waiting for the user to initiate the lautering process. Figure 30 shows the lautering process: when the user taps on the lauter tun, the agitator rotates. Simultaneously, the wort begins to flow into the lauter tun; the grain husks remain on top of the perforated mesh, while the clear wort passes through and is collected at the bottom.

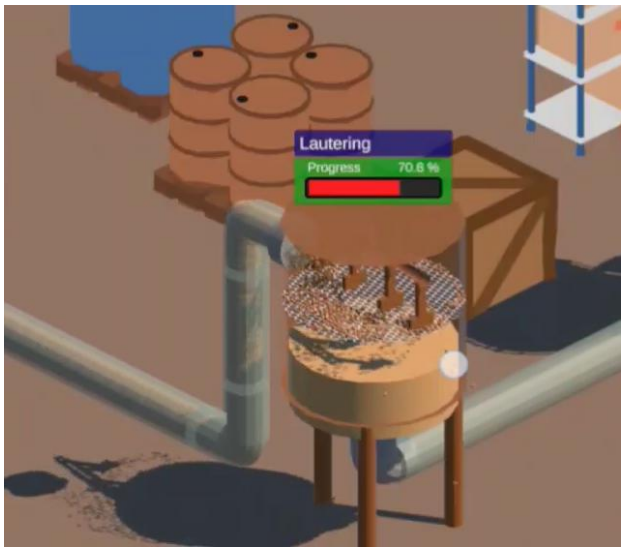

Figure 30: Lautering process gameplay.

4.1.4 Boiling and Hop Addition

Wort is sent into the boiling kettle. By tapping on the hops cannon, the player adds hops into the boiler as seen in Figure 31. The boiler automatically closes, and then the player needs to set a proper boiling time using the color-coded slider. The player should press and hold the boiler GameObject to start the boiling process. Upon completion of the process, the wort is automatically discharged.

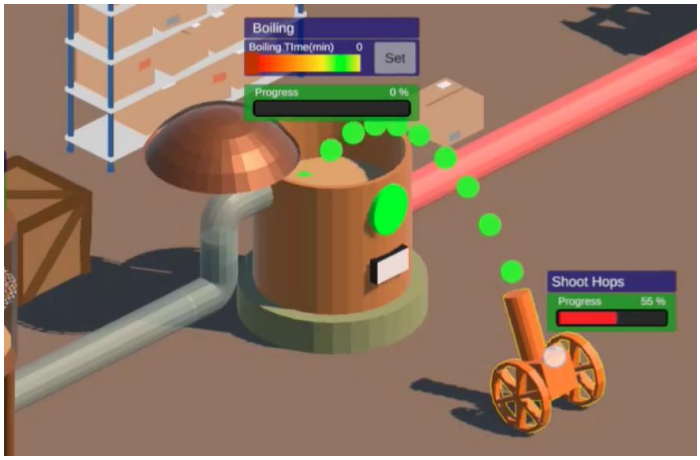

Figure 31: Adding hops to the boiler - gameplay.

4.1.5 Cooling

Following the boiling process, the wort passes through the cooler as seen in Figure 32. The player first sets the target cooling temperature (using a UI slider) to start the process. Once the desired temperature is set, the process is initiated by pressing and holding the cooler GameObject. The wort passes through the cooler and moves to the fermenter.

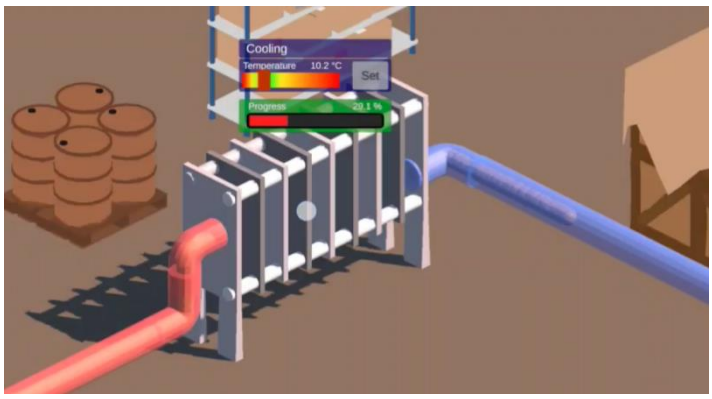

Figure 32: Cooling process gameplay.

4.1.6 Fermentation and Yeast Addition

After the wort is transferred to the fermenter, yeast is added by shooting from the cannon. The player adjusts the temperature and number of days for fermentation using UI elements. Then, the player taps and holds the fermenter GameObject to start the fermentation process. A day-night cycle is then shown to the passage of the set aging period (weeks). When the process is done, the wort moves to the maturation stage.

4.1.7 Maturation

In the maturation process, the player sets the number of weeks for the beer to age. While it is maturing, a day-night cycle is shown to simulate the passing of days.

4.1.8 Filtration

Following the maturation process, the beer moves to the filter. The filtration process starts when a player taps on it, the beer then gets filtered and as it passes through the equipment, a progress bar tracks the progression of this process.

4.1.9 Testing and Packaging

The brewmaster character conducts a final quality control test to determine whether the final beer meets the required quality standards. The brewmaster takes the beer from the sampling tank and drinks it to evaluate it. If the beer is satisfactory, he celebrates by dancing. If not, the brewmaster reacts by yelling. Figure 33 below illustrates the gameplay for the fermentation, maturation, filtration, and testing processes.

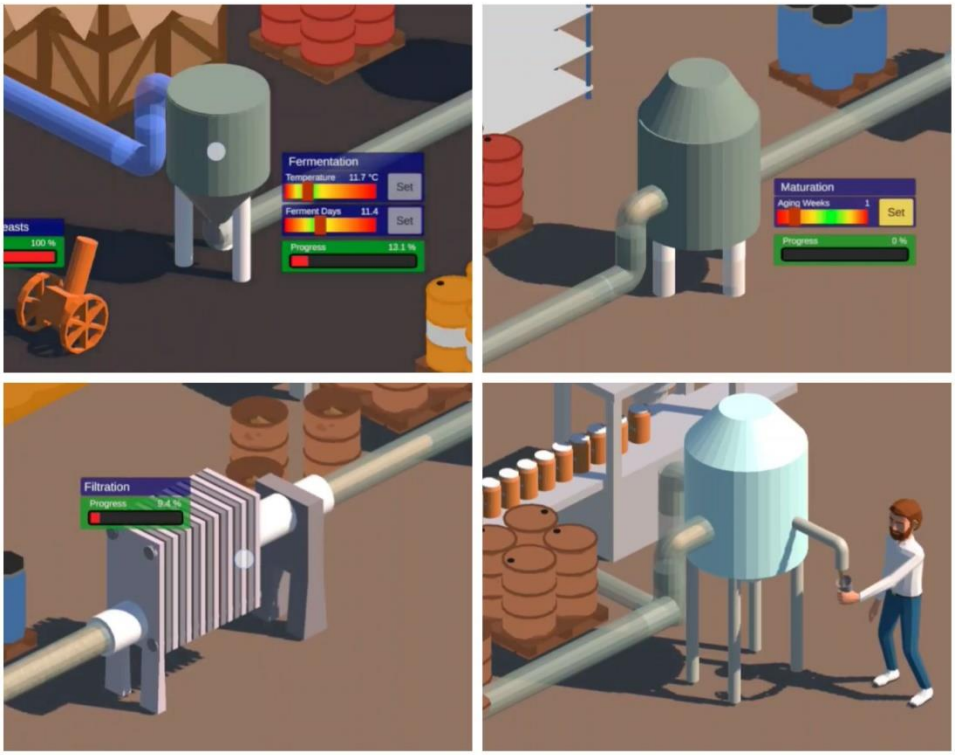

Figure 33: Fermentation, maturation, filtration, and Brewmaster testing the beer.

4.2 Endgame Rewards and QR Code Validation

Following the brewmaster evaluation, if the player gets a satisfactory result, the game moves into a win state. A game success UI appears, showing a message of congratulations and giving the player an option to print a coupon as a reward. The player can print and show the coupon to the brewery to redeem the discount.

To avoid fraudulent activities, a separate Android app is developed to validate the QR codes as seen in Figure 34. This verification app is built in Java using the “Code Scanner” library. This app serves to decode and decrypt the QR code data and check its authenticity.

Security measures include:

- A predefined key and suffix embedded in the code are used to decrypt and validate the string encoded in the QR code.
- Validated data will be saved in a local database to prevent repeated redemption or fraudulent attempts.

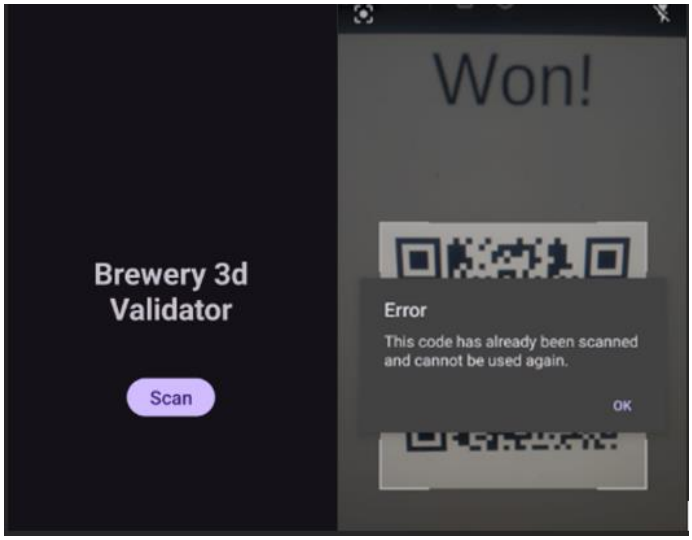

Figure 34: QR code validator app.

Summary

The 3D Brewery Game has successfully satisfied the core objectives of this project. The application presents an engaging simulation of the beer production process. It employs simplified brewing steps and interaction methods, such as slider adjustments for setting parameters and tapping on process equipment, to create a user-friendly and educational experience.

5 Conclusion and Recommendations

5.1 Conclusion

The goal of the project was to develop an interactive 3D Android game based on the brewing Bock's Corner Brewery. The game featured a low-poly 3D isometric design optimized for Android devices. It complies with GBL (Game-Based Learning) by simplifying the complex brewing process into interactive gameplay. Additionally, players who win the game get a QR code discount redeemable at the brewery. A separate QR code validation Android application was developed to check the validity of the QR code. This validation is done using encryption along with the local database check to ensure no fraudulent use. The project effectively blends gameplay, learning, and rewards, all centered on the Bock's Corner Brewery experience.

5.2 Recommendations

To improve engagement with the 3D brewery game, as well as educational impact, the following changes are proposed.

- Adding the ability to customize ingredients and beer styles would help encourage experimentation and increase engagement.
- Including an interactive in-game Brewing Encyclopedia, with resources detailing the brewing process and ingredients, would add a fundamental learning element.
- Adding sound effects, enhancing animations, and improving user interface would increase the user experience of the game.

6 References

Adobe Inc. (2025). *Adobe Photoshop*. https://www.adobe.com/products/photoshop.html

Adobe Mixamo Team. (2025). *Mixamo - 3D Character Animation*. https://www.mixamo.com

Ahmad, M. (2019). *Categorizing Game Design Elements into Educational Game Design Fundamentals*. https://doi.org/10.5772/intechopen.89971

Al-Khayat, M. R., Gargash, M. U., & Atiq, A. F. (2023). The Effectiveness of Game-based Learning in Enhancing Students' Motivation and Cognitive Skills. *Journal of Education and Teaching Methods ISSN*, *2*, 50–62. https://doi.org/10.58425/jetm.v2i3.199

Agor2012. (2025). Bearded Man Low-Poly Stylized. https://www.fab.com/listings/3d25f12e-bdb2-4f3d-9e2b-fb80c58f86ba

Blender. (2025). *Blender 4.4 Reference Manual*. https://docs.blender.org/manual/en/latest/

Boulton, Chris. (2017). *Encyclopaedia of brewing*. Wiley-Blackwell, Credo Reference.

Braad, E., Žavcer, G., & Sandovar, A. (2016). Processes and models for serious game design and development. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *9970 LNCS*, 92–118. https://doi.org/10.1007/978-3-319-46152-6_5

Budiyev, Y. (2025). *code-scanner: Code scanner library for Android, based on ZXing*. https://github.com/yuriy-budiyev/code-scanner

Charland, P. (2014). *Business Simulation Training in Information Technology Education: Guidelines for New Approaches in IT Training*. https://www.researchgate.net/publication/233911667

Deslauriers, L., McCarty, L. S., Miller, K., Callaghan, K., & Kestin, G. (2019). Measuring actual learning versus feeling of learning in response to being actively engaged in the classroom. *Proceedings of the National Academy of Sciences of the United States of America*, *116*(39), 19251–19257. https://doi.org/10.1073/pnas.1821936116

Epic Games. (2025). *Unreal Engine 5.5 Documentation*. https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-5-5-documentation

Fab.com. (2025). *Fab.com*. https://www.fab.com

Fullerton, T., Swain, C., & Hoffman, S. S. (2008). *Game Design Workshop: A Playcentric Approach to Creating Innovative Games* (2nd ed.). Morgan Kaufmann.

Gui, Y., Cai, Z., Yang, Y., Kong, L., Fan, X., & Tai, R. H. (2023). Effectiveness of digital educational game and game design in STEM learning: a meta-analytic review. *International Journal of STEM Education*, *10*(1). https://doi.org/10.1186/s40594-023-00424-9

Jahn, M. (2025). *ZXing.Net: .NET Port of the ZXing Barcode Library*. https://github.com/micjahn/ZXing.Net

JustCreate. (2023). *Low Poly FPS Map Vol.4*. https://assetstore.unity.com/packages/3d/environments/industrial/low-poly-fps-map-vol-4-254007

Kienitz, A., Eitel, A., & Krebs, M. C. (2024). Press START to Teach – Can Simulation Games Close the Theory-Practice Gap? *Simulation and Gaming*. https://doi.org/10.1177/10468781241252521

Léger, P.-M., Charland, P., Feldstein, H. D., Robert, J., Babin, G., & Lyle, D. (2011). Business Simulation Training in Information Technology Education: Guidelines for New Approaches in IT Training. In *Journal of Information Technology Education* (Vol. 10).

Li, L., Hew, K. F., & Du, J. (2024). Gamification enhances student intrinsic motivation, perceptions of autonomy and relatedness, but minimal impact on competency: a meta-analysis and systematic review. *Educational Technology Research and Development*, *72*(2), 765–796. https://doi.org/10.1007/s11423-023-10337-7

Macrovector. (2025). *Brewery Process Infographic Flat Style. Production Beer, Alcohol and Grain, Silo and Milling, Mashing and Lautering*. https://www.freepik.com/free-vector/brewery-process-infographic-flat-style-production-beer-alcohol-grain-silo-milling-mashing-lautering_10601143.htm

Nolet, C. (2022). *Quick Outline*. https://assetstore.unity.com/packages/tools/particles-effects/quick-outline-115488

Pacheco-Velazquez, E., Rodés, V., & Salinas-Navarro, D. (2024). DEVELOPING LEARNING SKILLS THROUGH GAME-BASED LEARNING IN COMPLEX SCENARIOS: A CASE IN UNDERGRADUATE LOGISTICS EDUCATION. *Journal of Technology and Science Education*, *14*(1), 169–183. https://doi.org/10.3926/jotse.2219

Plass, J. L., Homer, B. D., & Kinzer, C. K. (2015). Foundations of Game-Based Learning. *Educational Psychologist*, *50*(4), 258–283. https://doi.org/10.1080/00461520.2015.1122533

Rollings, A., & Adams, E. (2003). *Andrew Rollings and Ernest Adams on Game Design*.

Schell, J. (2008). *The Art of Game Design: A Book of Lenses*. Morgan Kaufmann.

Unity Technologies. (2025). *Unity Asset Store*. https://assetstore.unity.com