

CHECKOUT FRENZY – CONVENIENCE STORE SIMULATOR



Contents

- 1. Introduction..... 4
 - 1.1 Template Overview 4
 - 1.2 Target Audience 4
 - 1.3 Key Features and Highlights 4
 - 1.4 Support and Contact Information 5
- 2. Getting Started 5
 - 2.1 System Requirements 5
 - 2.2 Installation Instructions 5
 - 2.3 Project Structure..... 7

2.4 The Main Gameplay Scene.....	8
3. Core Gameplay Mechanics	10
3.1 Customer Interaction	11
3.1.1 Customer Spawning	11
3.1.2 Customer AI and Behavior	12
3.1.3 Checkout and Leaving	13
3.2 Checkout System	14
3.2.1 Customer Arrival and Item Placement	14
3.2.2 Scanning Items (Manual or Auto).....	15
3.2.3 Payment Processing	15
3.3 Inventory Management	17
3.3.1 Accessing the Order Terminal (PC)	17
3.3.2 Navigating the Order Program	17
3.3.3 Adding Items to the Cart	18
3.3.4 Ordering and Delivery	18
3.3.5 Product Management	19
3.4 Mission System	19
4. Customization.....	20
4.1 Adding and Configuring New Products & Licenses	20
4.1.1 Creating a New Product Asset	20
4.1.2 Configuring Product Properties.....	21
4.1.3 Creating and Configuring Licenses	22
4.2 Creating and Configuring Box Prefabs	23
4.2.1 Creating the Box 3D Model	23
4.2.2 Creating the Box Prefab	24
4.2.3 Configuring the Box Properties	25
4.3 Adding and Configuring New Shelving Units	26
4.3.1 Creating the 3D Model.....	26
4.3.2 Creating the Shelving Unit Prefab	27
4.3.3 Configuring Shelving Unit.....	28

4.4 Adding and Configuring New Customers.....	30
4.4.1 Obtaining a Rigged Humanoid Character	30
4.4.2 Importing the Character into Unity:	30
4.4.3 Creating the Customer Prefab.....	32
4.4.4 Configuring the Customer Prefab.....	32
4.4.5 Adding Customer Prefabs to the StoreManager.....	34
4.5 Customizing Store Elements.....	34
4.5.1 Checkout Counter	34
4.5.2 PC (Order Terminal)	37
4.5.3 Cashier Area.....	37
4.5.4 Expansions	38
4.5.5 Store Building.....	39
4.6 Adding and Customizing Missions	40
5. Third Party Assets	42
5.1 DOTween (v1.2.632)	42
5.2 SimpleInput (v1.1.7).....	43
5.3 Models.....	43
5.4 Audio	43

1. Introduction

Thank you for purchasing **Checkout Frenzy**! We appreciate your support and are excited to provide you with a complete convenience store simulator game template. This documentation will guide you through the setup, usage, and customization of the template to help you get the most out of this package.

1.1 Template Overview

Checkout Frenzy - Convenience Store Simulator is a complete Unity template (pack) that allows you to quickly create convenience store simulation game. From stocking shelves to serving customers, this template provides all the core mechanics and features you need to build your own unique store management experience. Easily customizable and optimized for mobile, Checkout Frenzy is the perfect starting point for both beginners and experienced developers.

1.2 Target Audience

Checkout Frenzy is designed for developers of all skill levels, from beginners to experienced creators. If you're passionate about simulation games, business management titles, or simply want to learn more about game development, this template is a great starting point. Its mobile-optimized design makes it perfect for reaching a wide audience, while its adaptability to PC or other platforms offers additional flexibility.

This template is perfect for you if:

- You want to create a mobile-friendly convenience store simulation game.
- You're a beginner or intermediate developer looking for a solid foundation.
- You want to save time and effort in game development.
- You're looking for a customizable and well-documented template.

1.3 Key Features and Highlights

Checkout Frenzy offers a range of powerful features, including:

- **Mobile-First, Cross-Platform Ready:** Optimized for Android, with easy adaptation for other platforms, maximizing your game's reach.
- **Visually Appealing & Ready to Use:** Stylized product models and diverse shelving options (general, fridge, freezer, fruit rack) allow you to quickly build a functional store.
- **Dynamic Gameplay Loop:** Engaging missions, dynamic customer AI with varied personalities, and a smooth checkout system (cash and card support included) keep players coming back for more.

- **Comprehensive Business Management:** Take control of your store with inventory management (restocking and pricing) and the ability to expand your store, unlocking new possibilities.
- **Immersive & Polished World:** A detailed environment with a fully implemented day/night cycle enhances the game's atmosphere and realism.
- **Easy Customization & Learning:** Effortlessly adjust game parameters, add new items, shelving units, customers, and missions. Learn and expand your game development skills with the well-documented source code.

1.4 Support and Contact Information

For support or inquiries, please contact us at cryingsnowstudios@gmail.com or join our Discord server: <https://discord.gg/QS6AWBbm4x>

2. Getting Started

This section will guide you through the process of setting up Checkout Frenzy and getting your project running. We'll cover everything from importing the template to configuring the game scene. By the end of this section, you'll be ready to start customizing your convenience store and building your game.

2.1 System Requirements

Before you begin working with Checkout Frenzy, please ensure your system meets the following requirements:

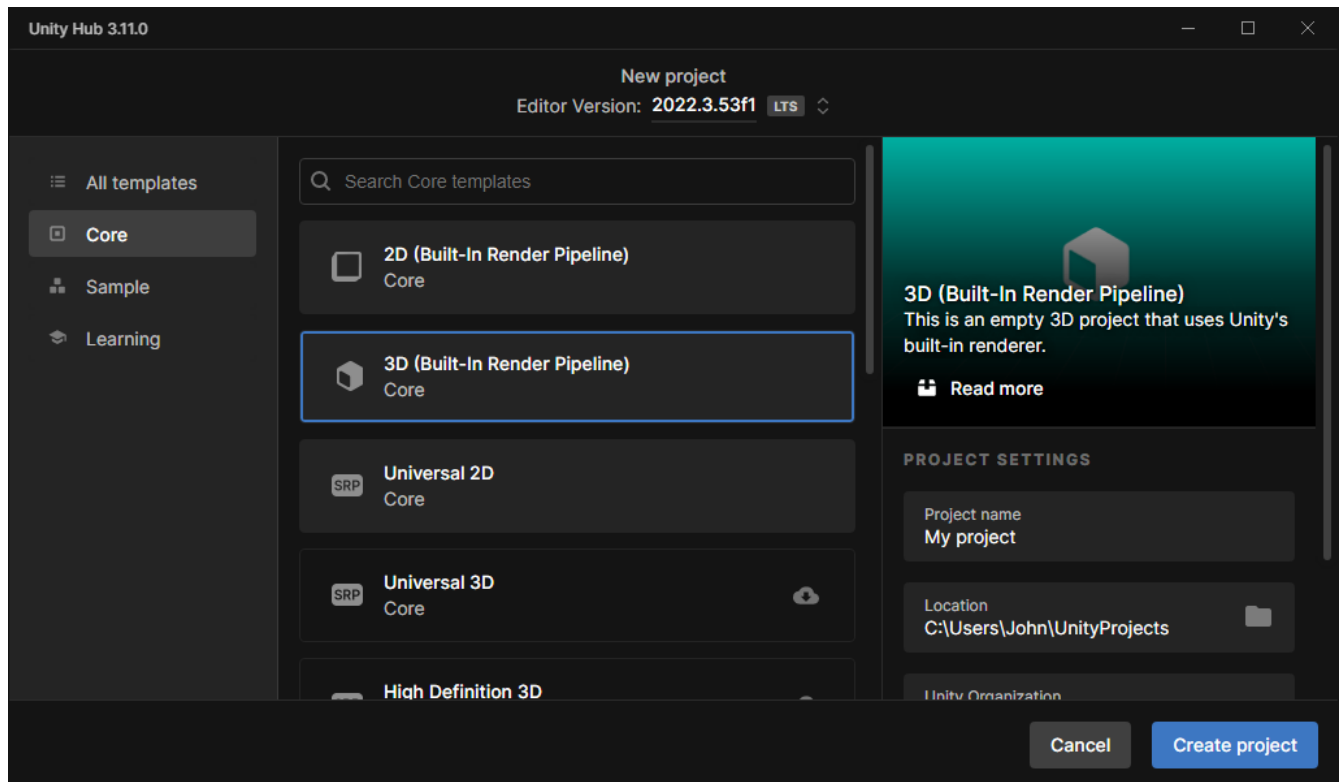
- **Unity Editor:** Checkout Frenzy is designed for use with Unity version **2022.3.53f1** or higher for optimal performance and compatibility.
- **Operating System:** The Unity Editor is supported on Windows 10 or later, macOS 10.15 or later, and Linux. For building your game to Android, you'll need a system capable of installing the Android SDK and NDK.
- **Hardware:** While the specific hardware requirements will depend on the complexity of your game, a computer capable of running the Unity Editor smoothly is recommended. A dedicated graphics card is recommended for optimal performance.

Please ensure your system meets these requirements before proceeding with the installation.

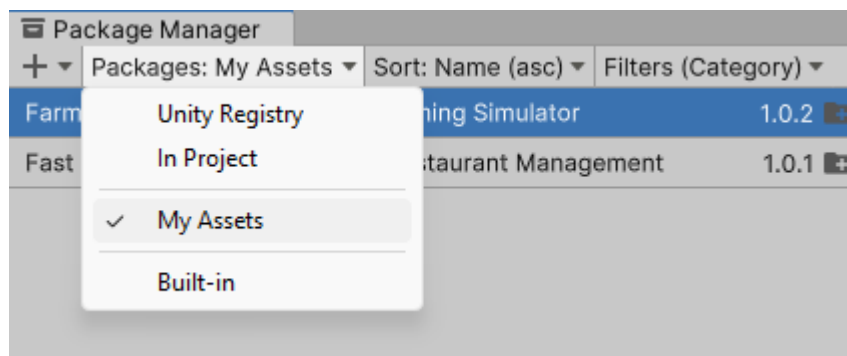
2.2 Installation Instructions

If you are a beginner, please follow these steps to install and use Checkout Frenzy:

- **Creating a New Project (Recommended):** For the best experience, we recommend starting a new Unity project specifically for Checkout Frenzy. This can be done via Unity Hub and select the Checkout Frenzy template if available.



- **Open the Package Manager:** In the Unity Editor, go to Window > Package Manager.
- **Select "My Assets":** In the Package Manager window, make sure the dropdown menu at the top left is set to "My Assets." This will display all the assets you've purchased from the Asset Store.



- **Locate Checkout Frenzy:** Search for "Checkout Frenzy" in the search bar at the top of the Package Manager window. Alternatively, you can scroll through the list of your assets.
- **Download (If Necessary):** If you haven't already downloaded Checkout Frenzy, you'll see a "Download" button. Click it to download the package. The download progress will be displayed.
- **Import:** Once the download is complete, the "Download" button will change to an "Import" button. Click "Import."
- **Import Package Dialog:** An "Import Unity Package" dialog will appear, listing all the files included in Checkout Frenzy. Generally, you should leave all items selected unless you have a specific reason not to import certain assets.

- **Click Import:** Click the "Import" button in the dialog box. Unity will then import the assets into your project.

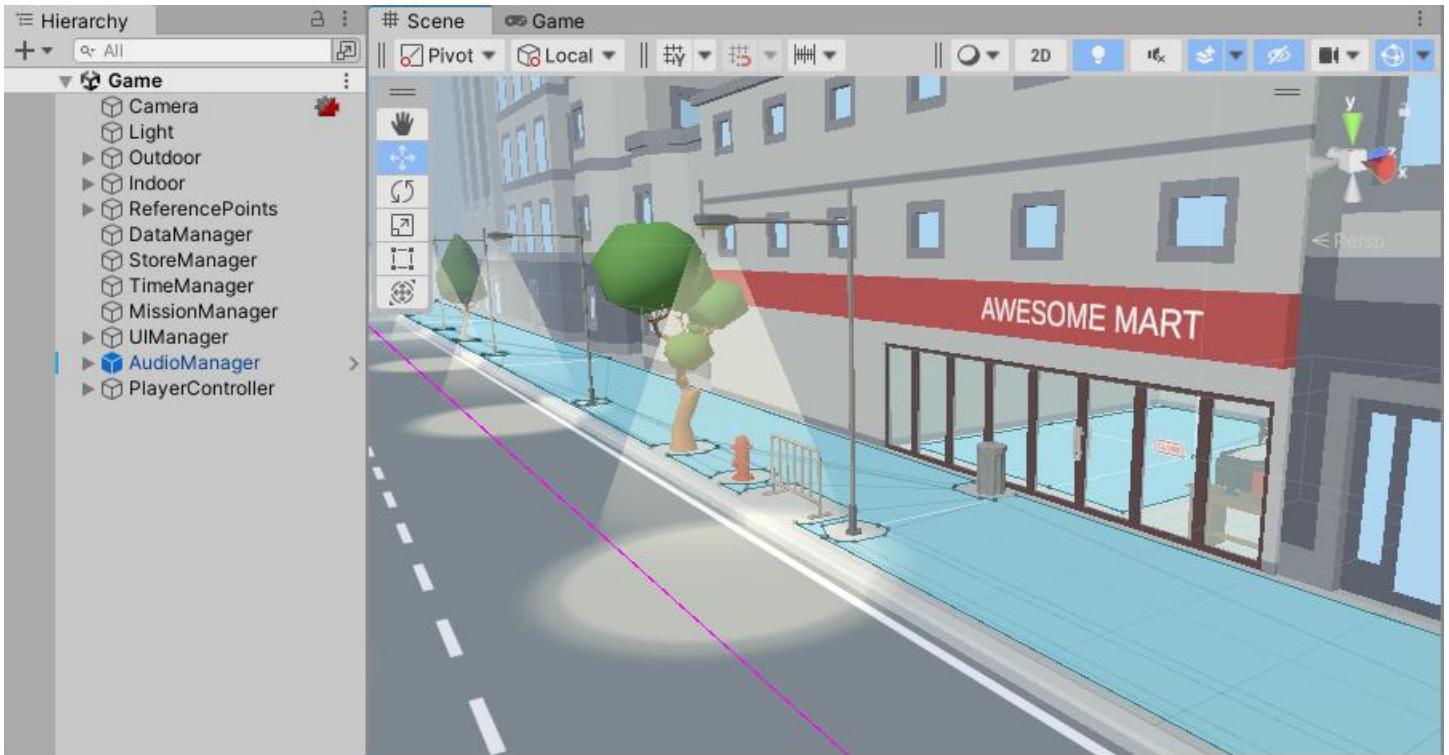
2.3 Project Structure

The Checkout Frenzy project is organized into the following key folders:

Folder	Description
Assets/CheckoutFrenzy	Main Template Folder: This folder contains the core assets and scripts that make up the Checkout Frenzy template.
Assets/CheckoutFrenzy/Animations	Contains customer animator controllers and animation clips.
Assets/CheckoutFrenzy/Dialogues	Stores the text content for customer dialogues (chat bubble popups).
Assets/CheckoutFrenzy/Materials	Contains the materials used for various 3D models and UI elements.
Assets/CheckoutFrenzy/Models	Contains the 3D models used in the game.
Assets/CheckoutFrenzy/Prefabs	Contains prefab assets, including customers, environment elements, and interactive objects.
Assets/CheckoutFrenzy/Resources	Crucial Data Assets: This folder contains Scriptable Objects that drive the game's logic and content, including box and furniture prefabs, and Scriptable Objects for Missions, Products, Product Licenses, and the main Game Configuration.
Assets/CheckoutFrenzy/Scenes	Contains the main menu scene and the core game scene.
Assets/CheckoutFrenzy/Scripts	Contains the C# scripts that control the game's logic and mechanics.
Assets/CheckoutFrenzy/Sounds	Contains audio clips for background music (BGM), sound effects (SFX), and the audio mixer settings.
Assets/CheckoutFrenzy/Sprites	Contains 2D sprites used in the game's UI and other visual elements.
Assets/CheckoutFrenzy/Textures	Contains the image textures used for models, materials, and UI elements.
Other Important Folders	
Assets/Plugins	Contains external plugins used in the template, such as DOTween (for animations) and SimpleInput (for input management).
Assets/Resources	Note: This is a separate Resources folder at the root level of the Assets directory. It specifically contains DOTween settings. It's important to distinguish this from the Resources folder inside Assets/CheckoutFrenzy.
Assets/TextMeshPro	Contains assets and resources related to TextMesh Pro, a powerful text rendering solution used for in-game text and UI.

2.4 The Main Gameplay Scene

This scene, located at `Assets/CheckoutFrenzy/Scenes/Game.unity`, is where the core gameplay takes place. It represents your convenience store and its surrounding.

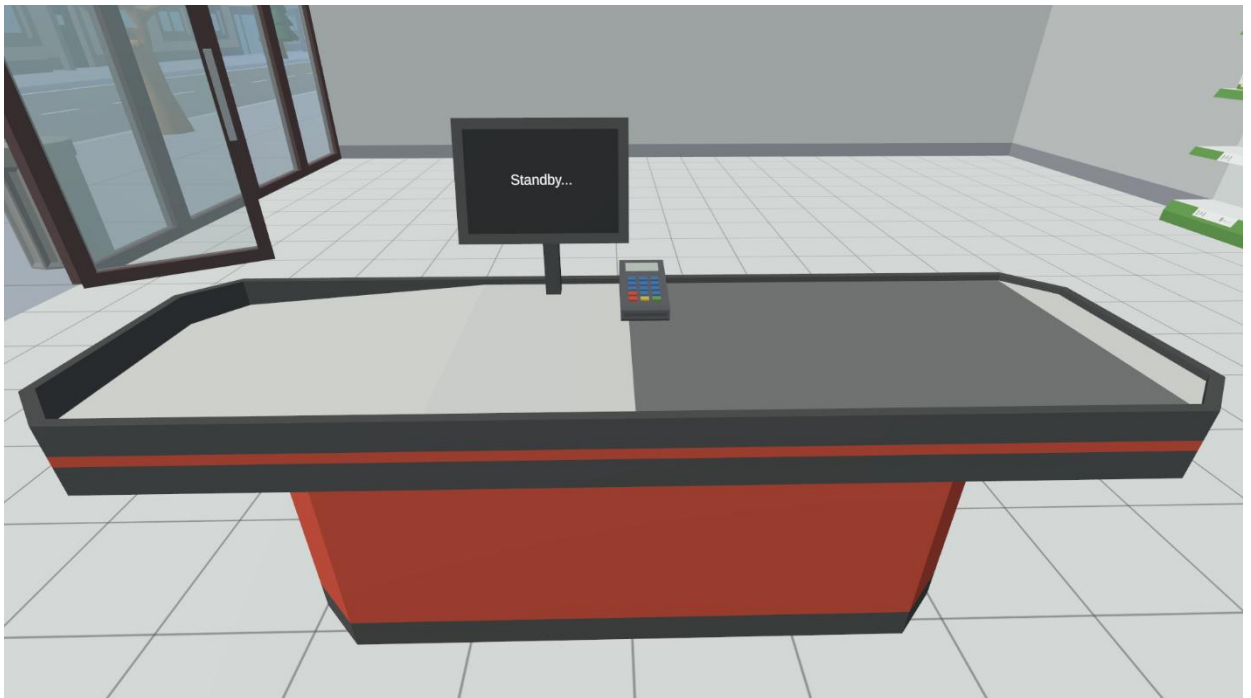


Here's a breakdown of the key components:

Scene Structure:

1. **Camera:** The main camera, utilizes Unity's `Cinemachine` to dynamically switch between virtual cameras, providing optimal views for different situations (player view, cashier view, PC screen, etc.).
2. **Light:** A `Directional Light` illuminates the entire scene.
3. **Outdoor:** This area surrounds the store and includes roads, buildings, and props. It's primarily for visual flavor and can be removed. **Important:** If removed, you must provide a collider (e.g., a plane with a `Mesh Collider`) in front of the store for `NavMesh Surface` generation. This allows customer AI to pathfind to your store.
4. **Indoor:** This section comprises the interactive elements of your store:
 - a. **Building:** The floors, walls, and ceiling of your store.
 - b. **Expansions:** Areas that can be unlocked to expand your store's size and offerings.
 - c. **CashierArea:** A `NavMeshModifierVolume` prevents customers from pathfinding to this area.

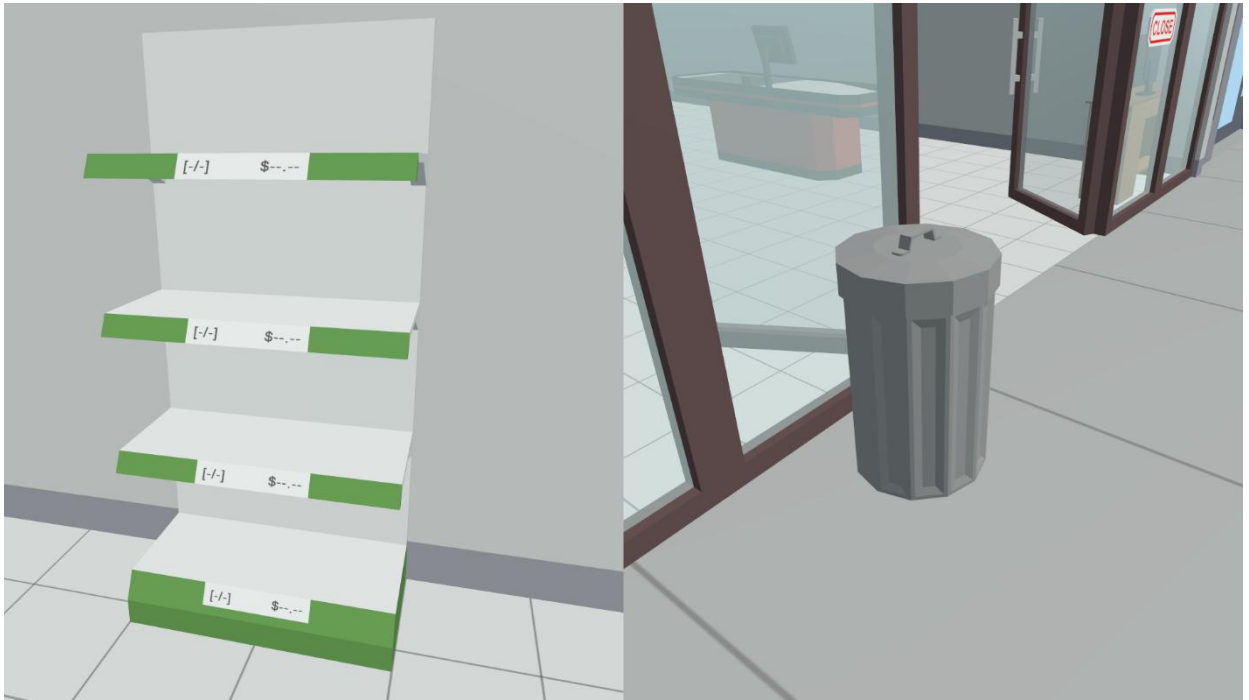
- d. **CheckoutCounter:** The player interacts with customers here.



- e. **PC (Order Terminal):** Used to order products, furniture, licenses, expansions, and services (cashier/cleaner hiring).



- f. **Default Furniture:** A shelving unit and trash can are provided to get player started.



5. **ReferencePoints:** These objects define key locations in the scene:
- DeliveryPoint:** Where ordered products and furniture are delivered.
 - SpawnPoint(s):** Customers spawn randomly at these points and then navigate to the store.
 - CarSpawner(s):** Cars spawn here, drive a set distance, and are then destroyed, adding traffic to the outdoor environment, serving no purpose other than cosmetic.

Game Logic and Management:

6. **DataManager:** Handles all databases, game data, saving, and loading.
7. **StoreManager:** Manages store operations, including opening/closing, customer spawning, and furniture placement.
8. **TimeManager:** Controls the in-game time and day/night cycle.
9. **MissionManager:** Manages the in-game mission (quest) system, including assigning missions to the player, tracking progress, and handling rewards.
10. **UIManager:** Manages all user interface elements.
11. **AudioManager:** Manages all sounds, including BGM and SFX.
12. **PlayerController:** This script controls the player character (first person character).

3. Core Gameplay Mechanics

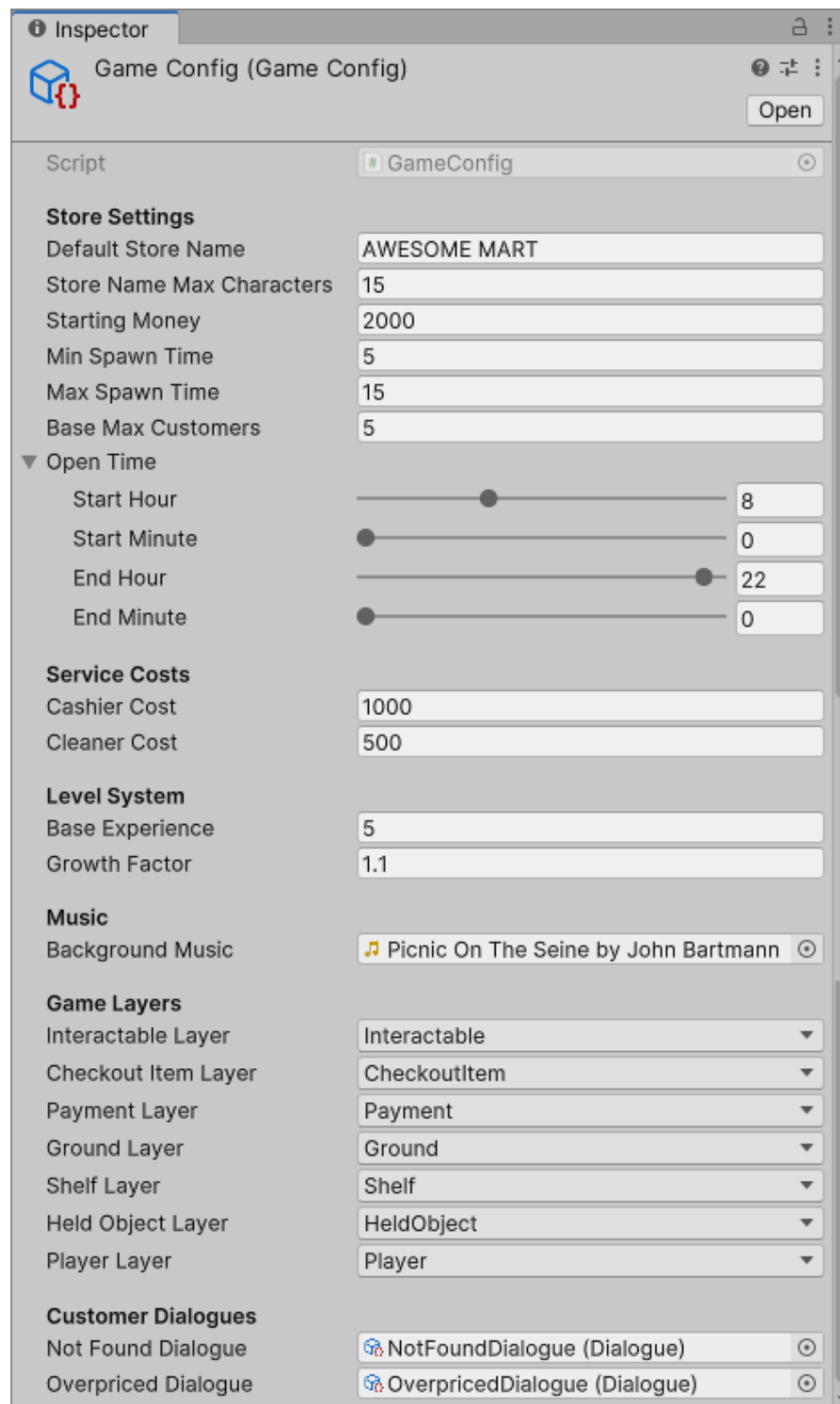
This section details the core gameplay mechanics of Checkout Frenzy, explaining how players interact with the game world and manage their store.

3.1 Customer Interaction

Customers are the lifeblood of your convenience store! Here's how customer interactions work in Checkout Frenzy:

3.1.1 Customer Spawning

Customers spawn randomly at designated points located outside the store. The time between spawns is controlled by the `GameConfig` (Tools > Checkout Frenzy > Game Config) and can be adjusted to control the flow of customers.



The maximum number of customers allowed in the store at any given time depends on your store's size (determined by `BaseMaxCustomers`) and upgrades (expansions). The `StoreManager` script manages customer spawning, ensuring that the store doesn't become overcrowded.

During the night (when in-game time reaches `OpenTime`'s `EndHour` and `EndMinute`) or when the store is closed, customers will not spawn. Player can interact with the store sign to open/close the store.



3.1.2 Customer AI and Behavior

Once a `Customer` spawns, they will attempt to enter the store. They will interact with the environment to find a `Shelving Unit` to browse and purchase products.

Customers navigate the store using Unity's `NavMesh` system, pathfinding their way to shelving units and later the `Checkout Counter`. If a shelving unit is moving (being rearranged by the player), the customer will choose a different shelving unit (if available).

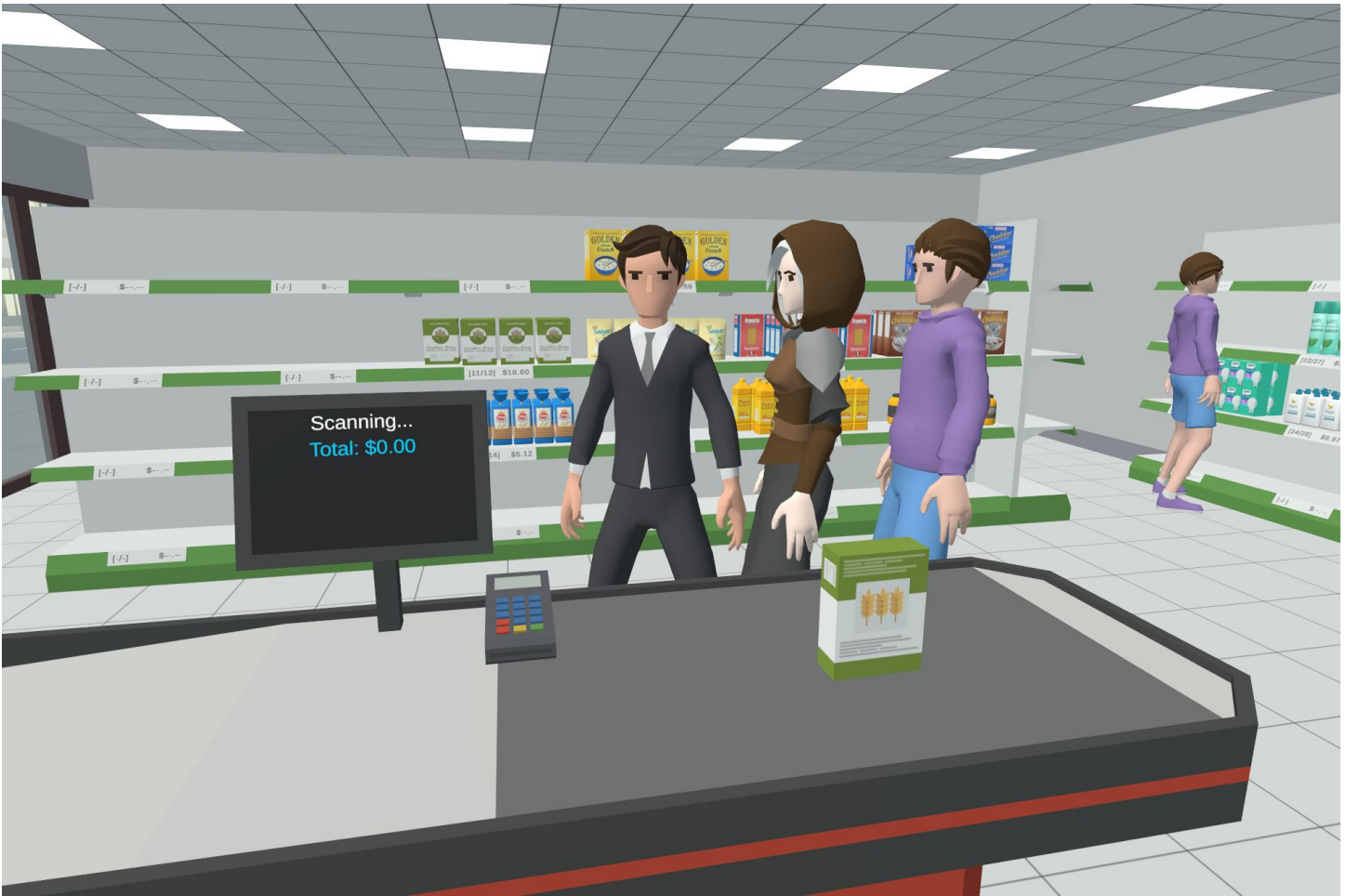


After browsing, customers have a 50% chance to continue shopping or proceed to the checkout. If they decide to buy, they will pick a product from the shelves. If the product is overpriced, they will express their discontent via a “chat bubble” before leaving without purchasing.

If they have no products on their inventory, they will leave the store without purchasing. Customers can only pick up products if the shelving unit has available stock.

3.1.3 Checkout and Leaving

Once a customer has gathered their desired products, they will queue at the checkout counter. The `StoreManager` manages the queue, assigning each customer a position and directing them to the appropriate spot.



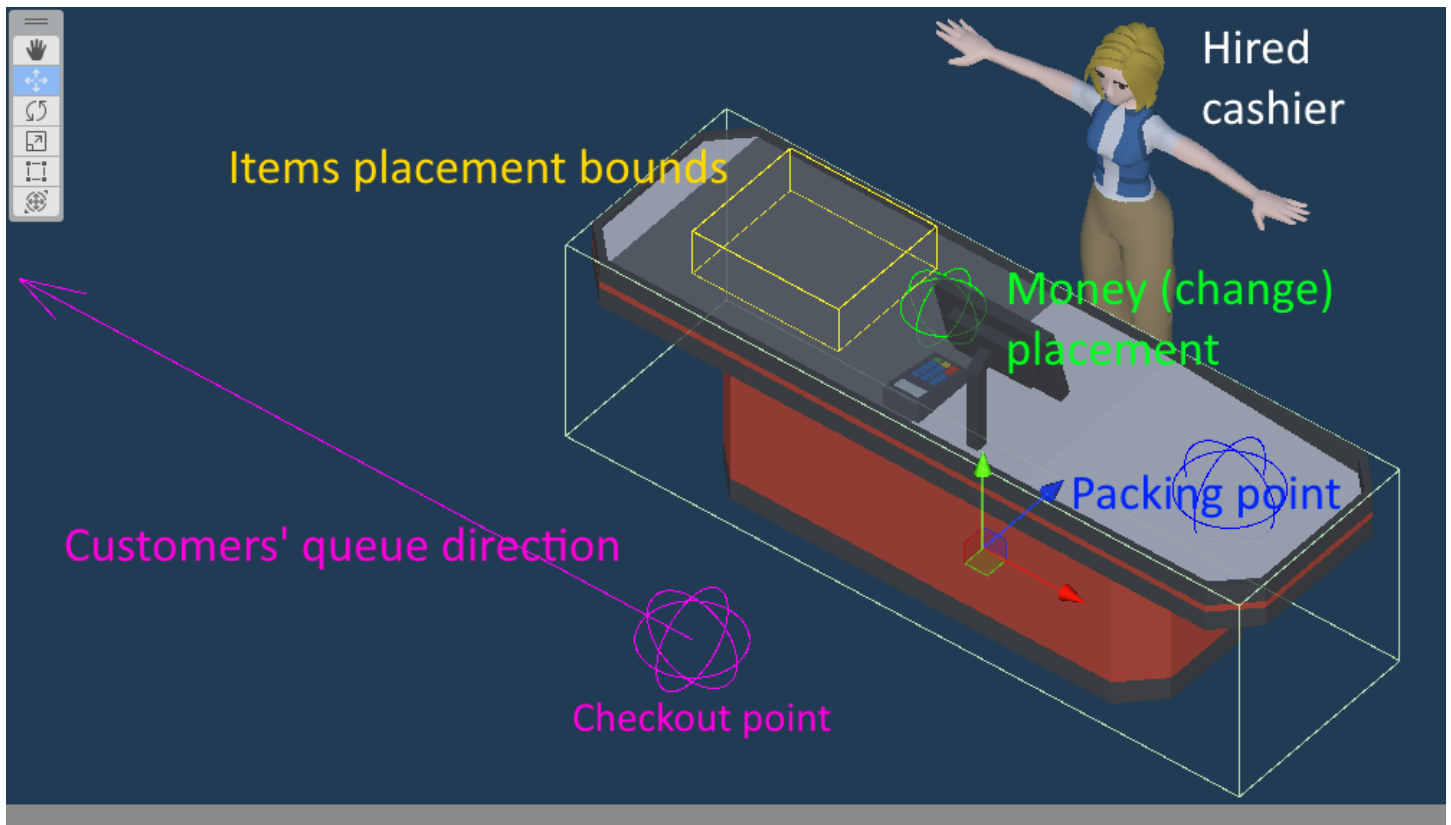
When it's their turn, the customer will place their items on the counter. The player (or a hired cashier) then processes the payment. After payment is complete, the customer leaves the store. Customers will leave the store if the store is closed and they have no products on their hands.

3.2 Checkout System

The checkout counter is where the magic happens – where you turn customer purchases into profit! Here's a detailed look at how the checkout system works in Checkout Frenzy:

3.2.1 Customer Arrival and Item Placement

When a customer is at the front of the checkout queue, they will place their selected products on the checkout counter within the designated `placementBounds` area.



The game attempts to find a valid, non-overlapping position for each item. If it cannot find a valid spot after several attempts (100 times by default), the item will be placed at the last attempted position (not to worry, this happened in a fraction of a second). This process is animated, with the items visually "jumping" onto the counter.

3.2.2 Scanning Items (Manual or Auto)

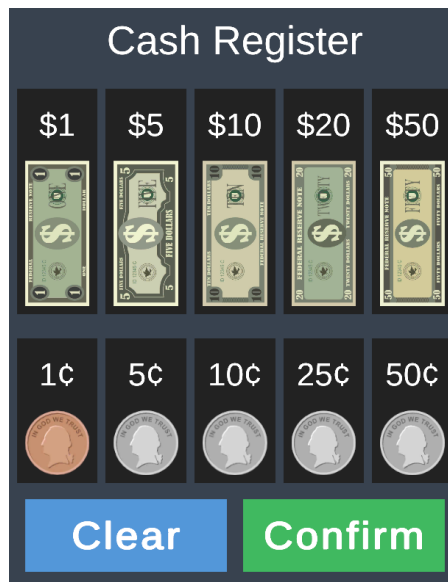
There are two ways to scan items:

- A. **Manual Scanning (Player as Cashier):** If you haven't hired a cashier, you'll need to scan the items yourself. While in "Working" state, which is triggered by interacting with the checkout counter, you can click on the items on the counter to scan them. The `CheckoutCounter` script will remove the item from the counter and add its price to the `totalPrice`. The scanned item will then be moved to the designated `packingPoint` and destroyed. The `monitorText` UI element displays running total of the scanned items. The `AudioManager` will play a scanner sound effect.
- B. **Auto Scanning (Hired Cashier):** If you've hired a cashier, they will automatically scan the items as soon as they are placed on the counter. The `autoScanTime` variable in the `CheckoutCounter` script controls the speed of the auto-scanning process.

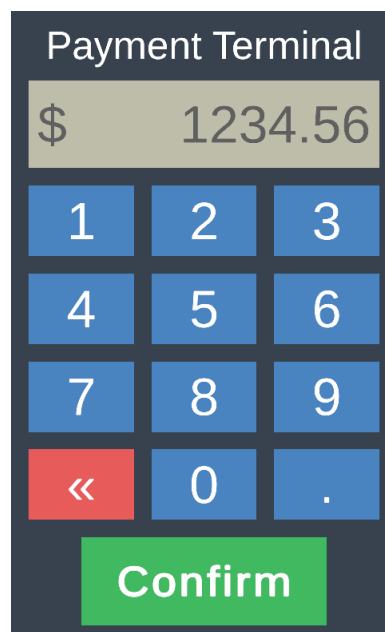
3.2.3 Payment Processing

Once all items are scanned, the customer will present their payment. Checkout Frenzy supports two payment methods:

- A. **Cash:** The customer will attempt to pay with cash. The amount the customer pays will be randomly selected from a few options. The `CheckoutCounter` script provides a few ways to determine the payment amount, including exact change, rounding up, and paying with a higher denomination. If you're acting as the cashier, the cash register UI will open. You'll need to manually provide the correct change using the cash register. The `CheckoutCounter` script will check if the change given is sufficient. If it isn't the customer will be notified with the insufficient change message. If it is correct, the transaction is complete. The `moneyPoint` transform defines the location where the change will appear. The `CheckoutCounter` script will then animate the money given to the customer by instantiating money sprites and animating them to the money point.



- B. **Card:** The customer will pay with a card. The payment terminal UI will open. You'll need to enter the correct amount. The `CheckoutCounter` script will verify the amount entered. If the amount is correct, the transaction is completed.



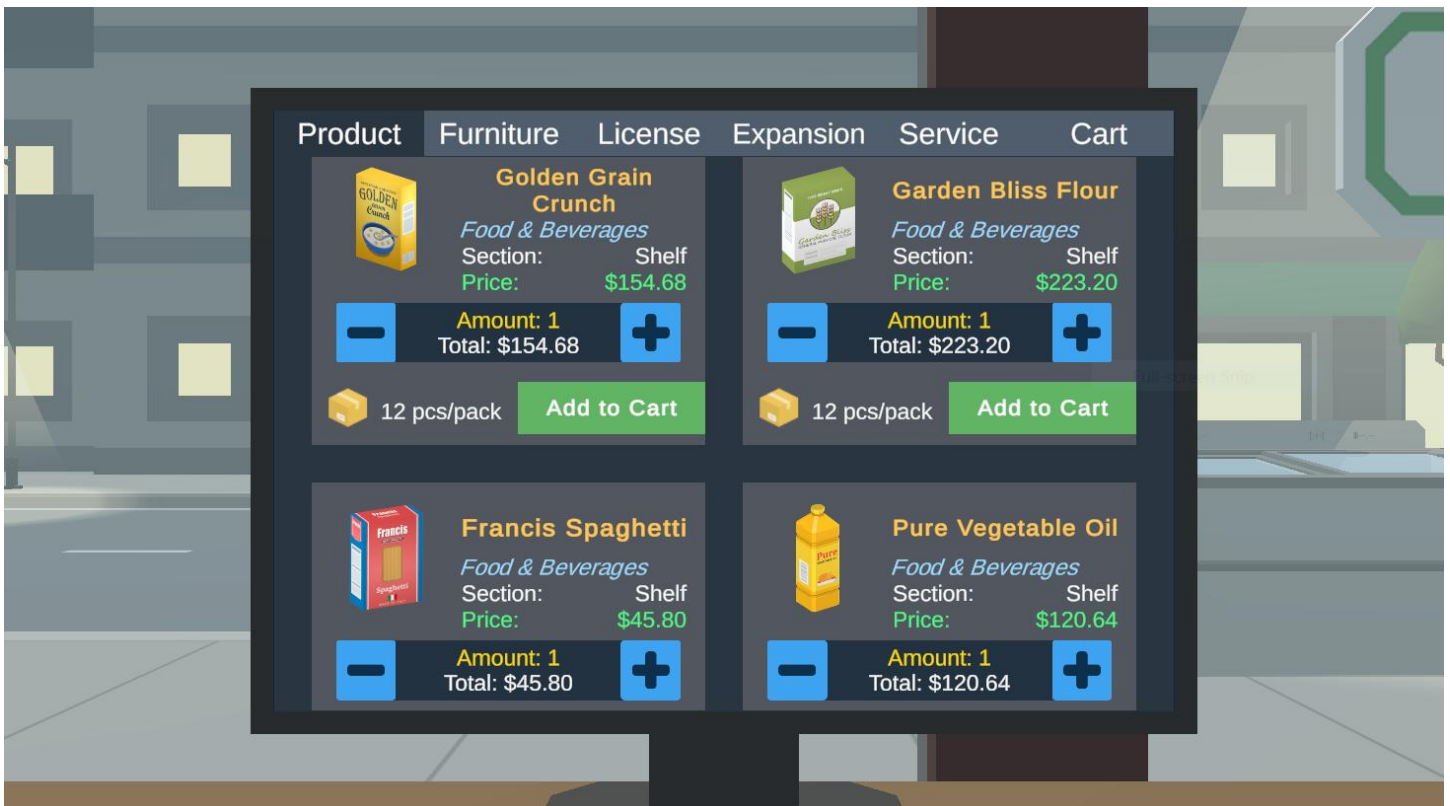
The `CheckoutCounter` script manages the payment process, including tracking the `totalPrice`, the `customerMoney` paid, and the `givenChange`. After a successful payment, the customer's money is added to your total funds, and the transaction is complete. The `MissionManager` is notified of the successful checkout.

3.3 Inventory Management

This section explains how to order new products and furniture using the in-game PC and how to manage those using boxes and shelves.

3.3.1 Accessing the Order Terminal (PC)

The order terminal is accessed via the PC located in your store. To interact with the PC, approach it and tap the interact button. This will trigger the `PlayerController`'s interaction logic and activate the PC, displaying the “fake” loading sequence and then the order program.



3.3.2 Navigating the Order Program

The PC monitor displays several tabs, allowing you to browse available products, furniture, licenses, expansions, and services. The `PCMonitor` script controls the UI and the interaction between the player and the PC.

- **Product Tab:** This tab lists all available products that you have a license to sell.
- **Furniture Tab:** This tab lists all available furniture items.
- **License Tab:** This tab displays available licenses that you can purchase to unlock new products.

- **Expansion Tab:** This tab displays available expansions that you can purchase to expand your store.
- **Service Tab:** This tab allows you to hire a cashier or a cleaner.

3.3.3 Adding Items to the Cart

To add an item to your shopping cart, select the desired product or furniture on the appropriate tab. Clicking on the "Add to Cart" button will add it to your cart. You can add multiple quantities of the same item. The `PC` script manages the shopping cart, storing the selected items and their quantities.

3.3.4 Ordering and Delivery

Once you've selected all the items you need, proceed to the checkout screen by clicking the "Cart" tab. After placing the order using Checkout button, the ordered items will be delivered to your store. The `PC` script's `ProcessOrder` coroutine simulates the delivery time. Once the delivery time has elapsed, the items (products in box containers, or furniture) will appear at the designated `deliveryPoint` in your store.



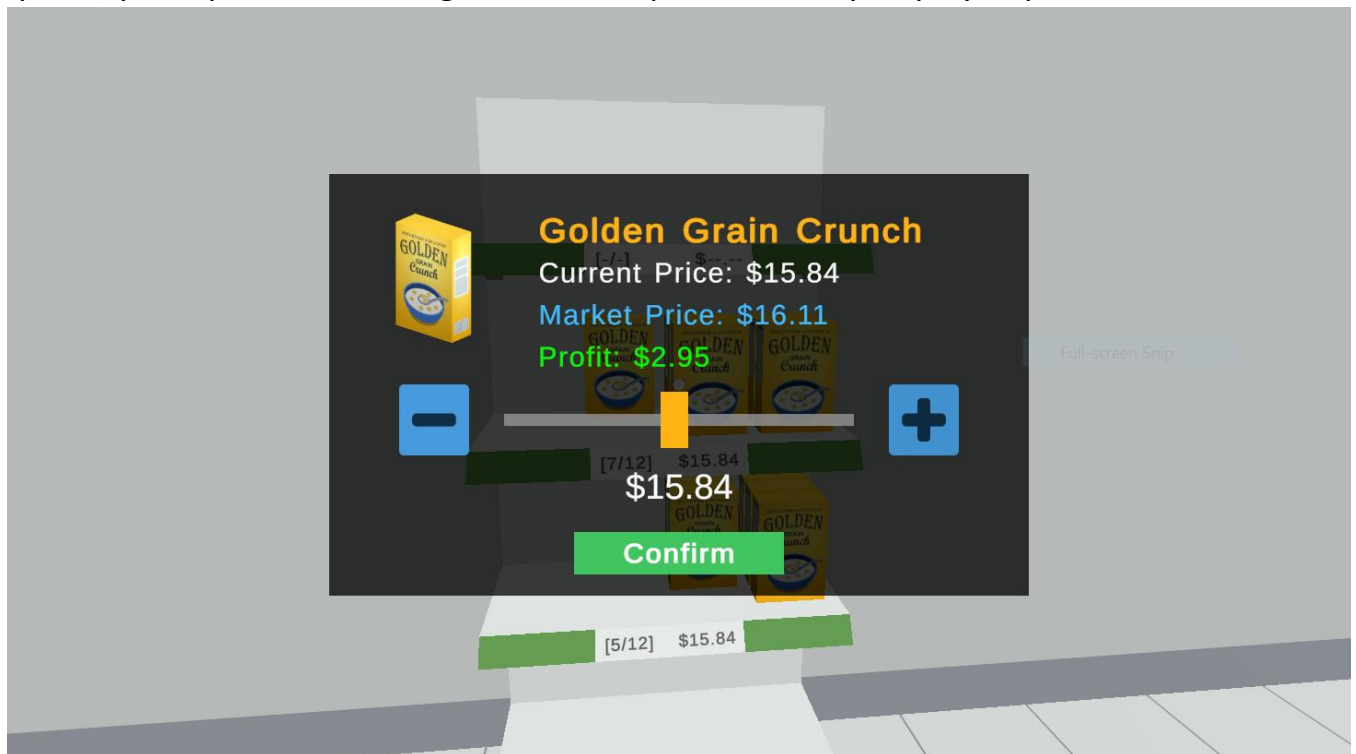
Each order contributes to the progress of "Restock" (for products) and "Furnish" (for furniture) missions. The `MissionManager` script tracks mission progress.

3.3.5 Product Management

Managing products in Checkout Frenzy is primarily handled through `ProductContainer` components. These containers provide a standardized way to store and display products.

There are two main types of product containers:

- **Boxes:** To interact with a box, approach it and tap the interact button. This allows you to pick up and carry the box. While holding an open box, you can transfer products between the box and compatible shelves (matching sections for placing, matching sizes for taking).
- **Shelves:** Shelves are used to display and sell products to customers. Shelves are grouped into `ShelvingUnits`. The `ShelvingUnit` script manages the overall behavior of the shelving unit, including opening and closing (for units with doors), and interaction with the shelves. Each shelf displays information about the product it contains, including the quantity and price. Don't forget to set the price of newly displayed products.



3.4 Mission System

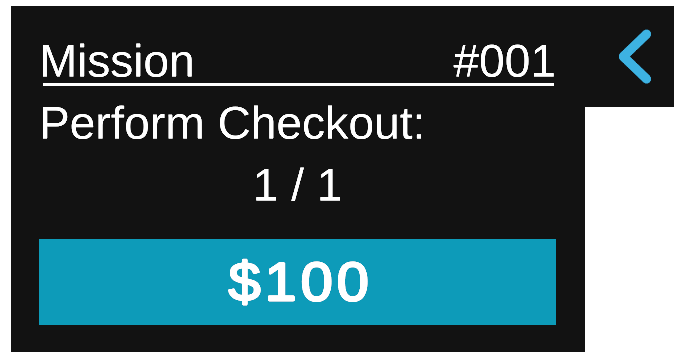
Checkout Frenzy features a mission system that provides objectives for player to complete, guiding your gameplay and offering rewards upon completion. Missions are a great way to earn extra money and progress through the game.

There are several types of mission goals:

- **Checkout:** Complete a specific number of manual checkouts.
- **Revenue:** Earn a specific amount of revenue.

- **Sell:** Sell a specific number of a particular product.
- **Restock:** Purchase and receive a specific number of a particular product.
- **Furnish:** Purchase and place a specific number of furniture items.

The `MissionManager`'s `UpdateMission` method is called whenever you perform an action that could contribute to mission progress (e.g., completing a checkout, selling a product, restocking a product, furnishing a furniture).



The `MissionTracker` displays the current mission objective, progress, and reward. The `MissionManager`'s `OnMissionUpdated` event can be used to update the UI whenever mission progress changes. The UI should also have a button to claim the reward and advance to the next mission when a mission is completed.

4. Customization

Checkout Frenzy is designed to be moddable, allowing you to add your own content and personalize the game experience. This section provides a guide for developers and modders on how to customize various aspects of the game, including adding new products, customers, shelving units, and customizing the store layout.

4.1 Adding and Configuring New Products & Licenses

This subsection details the process of creating new `Product` Scriptable Objects and configuring their properties. The `ProductEditor` script (a custom editor for Product assets) simplifies this process within the Unity Editor.

4.1.1 Creating a New Product Asset

- Ensure that your `Product` assets are inside a folder named "Resources/Products" (e.g., `Assets/CheckoutFrenzy/Resources/Products`). `DataManager` script uses Unity's `Resources.LoadAll<Product>("Products")`, which only loads assets from the `Resources` folder or its subfolders.

- Right-click in the folder and select Create > Product. This will create a new Product Scriptable Object asset.

Inspector

New Product (Product)

Product Details

Product ID: 0

Product Name:

Product Icon: None (Sprite)
 [Select](#)

Product Icon is missing. Assign an icon or generate one automatically.

[Generate Icon from Model](#)

Price (in cents): 0

Order Time (seconds): 5

Category and Section

Category: Food And Beverages

Store Section: General

Product Assets

3D Model: None (Game Object)

Product Box: None (Box)

No Product Box assigned.

4.1.2 Configuring Product Properties

- **Product ID:** A unique numerical identifier for the product. The `ProductEditor` automatically assigns the next available ID when you create a new product. It also detects and helps you resolve duplicate product IDs (e.g., when you duplicate an existing product). *Important:* Product IDs must be unique.

Product Details

Product ID: 10

Duplicate Product ID detected. Click 'Fix' to resolve.

[Fix Duplicate Product ID](#)

- **Product Name:** The name of the product (e.g., "Best Cereal", "Awesome Milk", "Toy Car"). This name is displayed in the game UI.

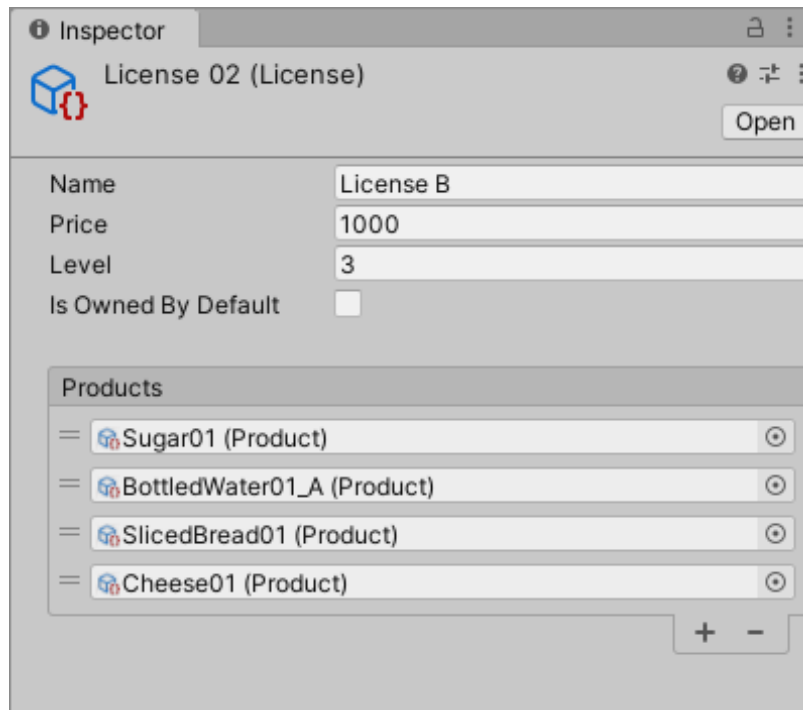
- **Product Icon:** A 2D image (Sprite) representing the product. This icon is used in the in-game UI (e.g., in the product catalog on the PC). If a 3D Model is assigned, and the Product Icon is missing, you can click the "Generate Icon from Model" button to automatically create an icon from the model. This creates a snapshot of the model and saves it as a sprite in the `Assets/CheckoutFrenzy/Sprites/Products` folder.
- **Price (in cents):** The price of the product in cents. The game internally converts this value to dollars for display.
- **Order Time (seconds):** The time it takes for the product to be delivered after it is ordered.
- **Store Section:** The section of the store where this product belongs (e.g., "Food & Beverage," "Household"). This determines which shelves the product can be placed on.
- **3D Model:** A prefab of the 3D model representing the product in the game world. This model will be instantiated and placed on shelves or in boxes.
- **Product Box:** A prefab of the Box that this product is packaged in. This box will be used when the product is delivered. The Box prefab should be configured with the appropriate size and capacity for the product.

4.1.3 Creating and Configuring Licenses

Licenses are required to make products available for purchase and restocking in the game. You can create and configure licenses as follows:

- In the Unity Editor, navigate to the folder where you want to store your `License` assets (e.g., `Assets/CheckoutFrenzy/Resources/Licenses`). Like `Product` assets, `License` assets must be placed inside "`Resources/Licenses`" folder.
- Right-click in the folder and select `Create > License`. This will create a new `License Scriptable Object` asset.

Select the newly created `License` asset to open it in the Inspector window and configure the following properties:



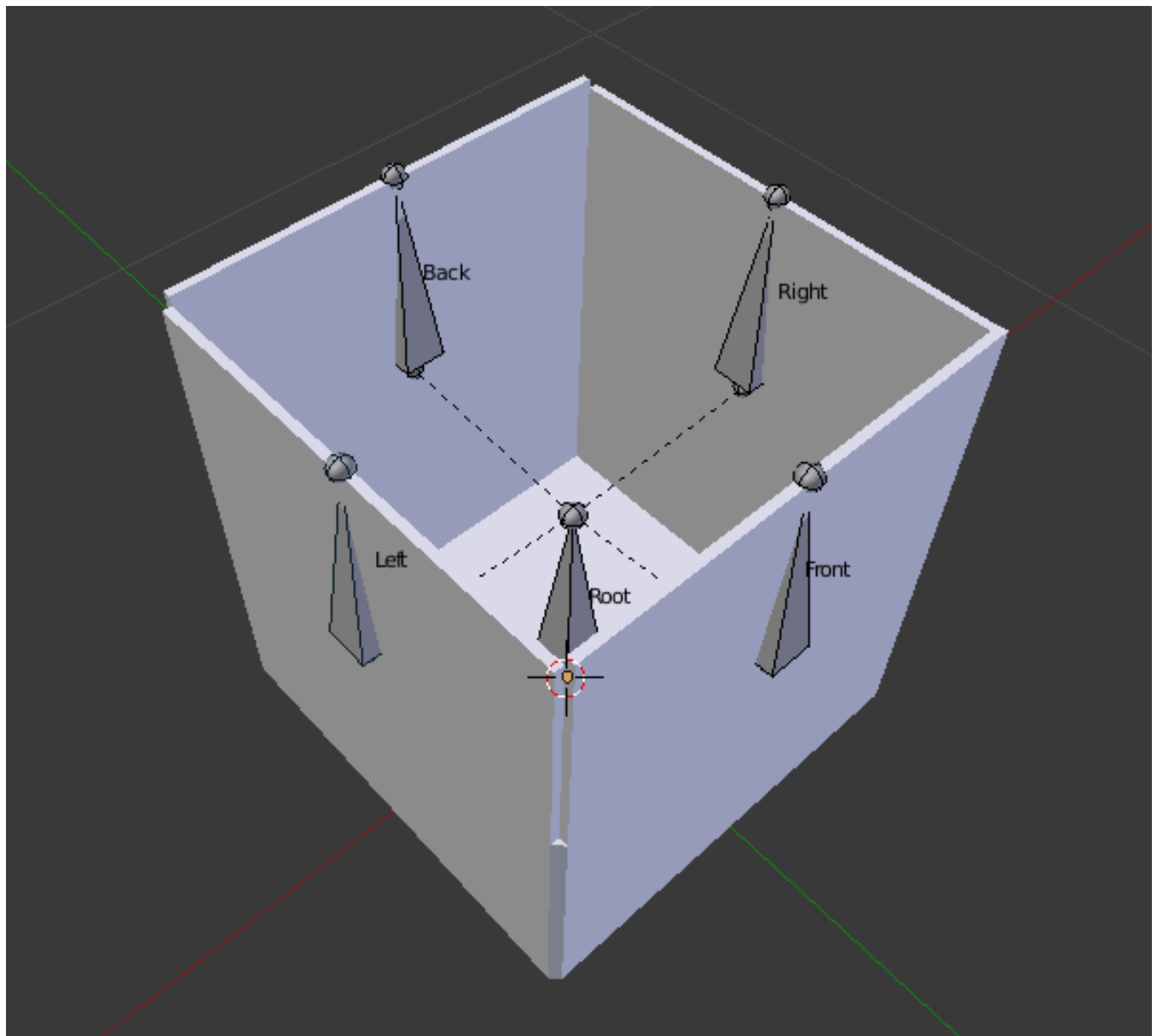
- **Name:** The name of the license (e.g., "Basic Food License," "Premium Electronics License").
- **Price:** The price (in dollars) of the license in the in-game currency.
- **Level:** Represents the required player or store level to purchase this license.
- **Is Owned By Default:** A Boolean to determine if the player owns this license by default at the start of the game.
- **Products:** A list of Product Scriptable Objects that are associated with this license. Drag and drop the desired Product assets into this list. *This is how you link products to a license.*

4.2 Creating and Configuring Box Prefabs

`Box` prefabs are essential for packaging and delivering products in Checkout Frenzy. They define the size and capacity of the boxes in which products are delivered. Follow these steps to create and configure `Box` prefabs, using rigged models for the lids:

4.2.1 Creating the Box 3D Model

- Create your box model in a 3D modeling program (e.g., Blender, Maya).
- Rig the lids to the box body (root). This will allow you to animate the lids in Unity. *Do not create animations in your 3D modeling program.* The animations will be handled by the `Box` script in Unity.



- Export then Import your model as an FBX file. *Important:* Pose the box lids to the initial “closed” position when exporting.
- Alternatively, you can just duplicate and modify the box models provided with this template (e.g., change the size, change the textures).

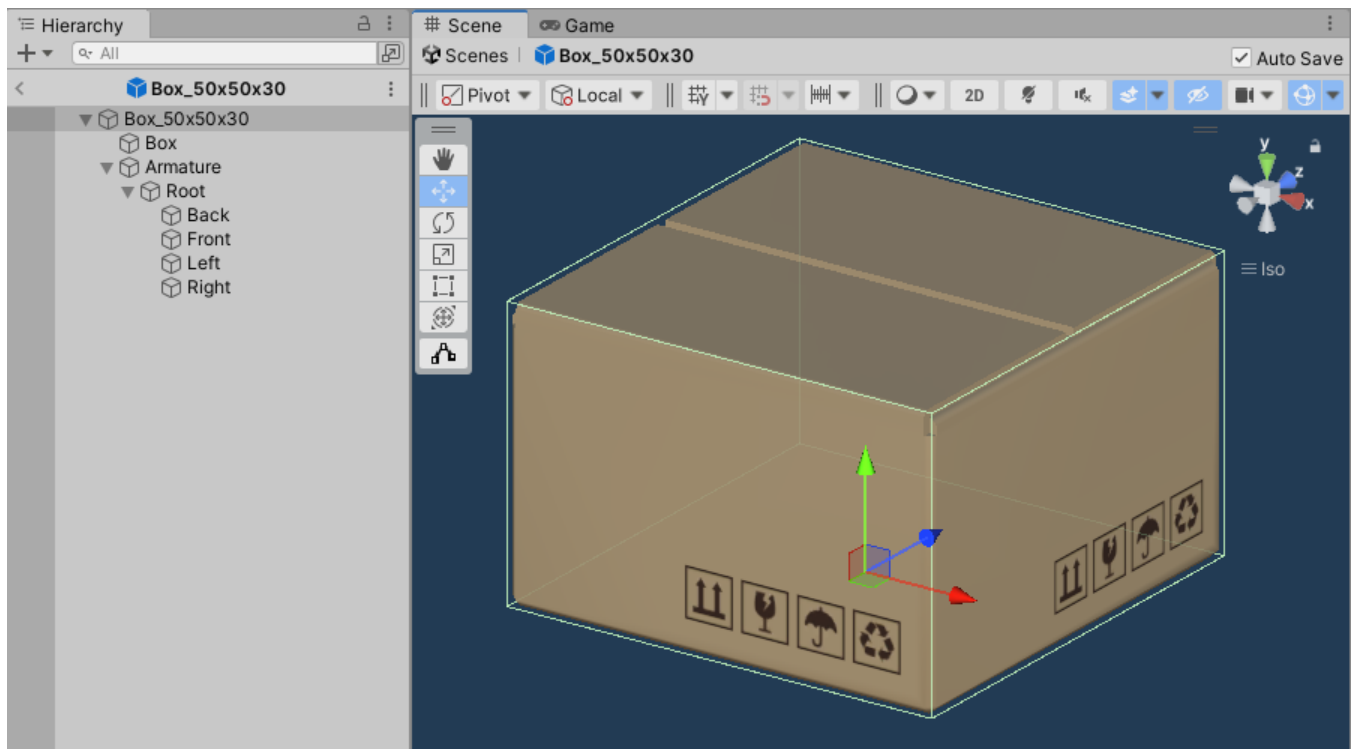
4.2.2 Creating the Box Prefab

- In the Unity Editor, navigate to the folder where you want to store your Box prefabs (e.g., Assets/CheckoutFrenzy/Resources/Boxes). Like Product and License assets, Box prefabs must be placed in a "Resources/Boxes" folder.
- Drag the imported 3D model from your project window into the scene to create an instance of the model.

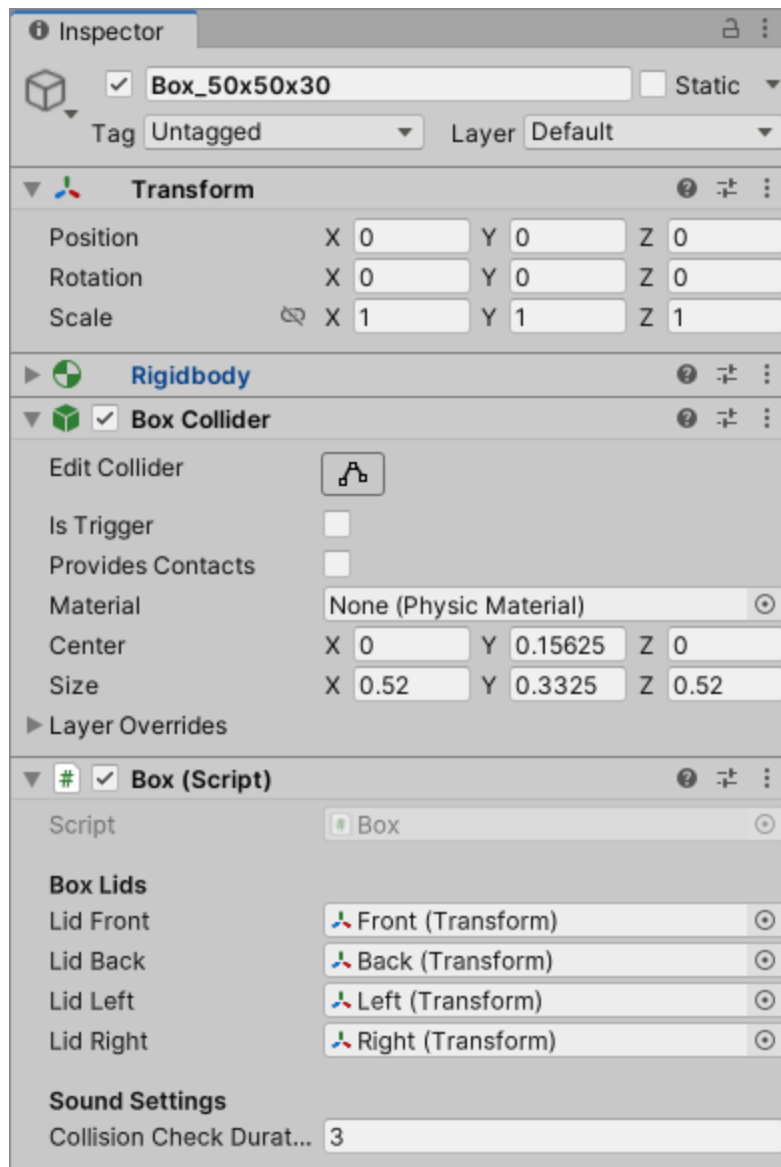
- Add a `Box` component to the model instance. You can do this by selecting the model in the Hierarchy window, then in the Inspector window, click `Add Component` and search for "`Box`." This will automatically add a `Rigidbody` and `Box Collider` as well. While the `Rigidbody` settings can remain unchanged, correctly sizing the `Box Collider` to the box's dimensions is crucial for proper interactions.
- Drag the model instance from the Hierarchy window back into your project folder to create a prefab, name it appropriately. You can now delete the instance from the scene if you wish.

4.2.3 Configuring the Box Properties

- **Open Prefab:** Double click the box prefab you just created to enter the prefab edit mode and configure its properties.



- **Lid Transforms:** Drag the bone transforms for each lid (front, back, left, and right) from the Hierarchy window to the corresponding fields in the `Box` component. This is essential for the `Box` script to animate the lids.



- **Collision Check Duration:** This is duration (in seconds) to check for collisions after throwing the box.

4.3 Adding and Configuring New Shelving Units

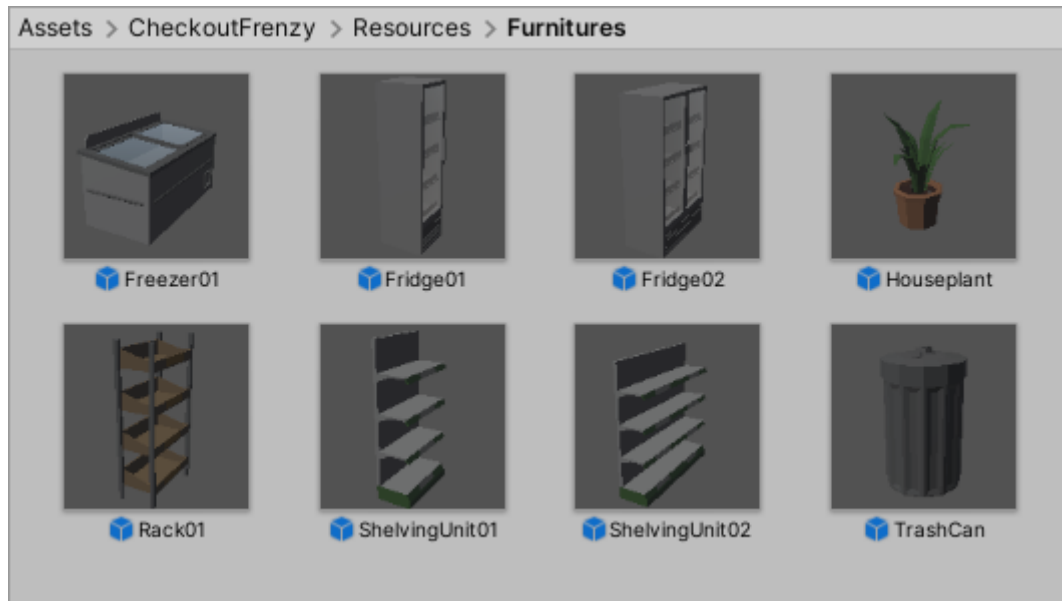
This subsection explains how to add new shelving units to the game. Shelving units are used to display products for sale and are essential for your store's functionality. Follow these steps to create and configure new shelving units:

4.3.1 Creating the 3D Model

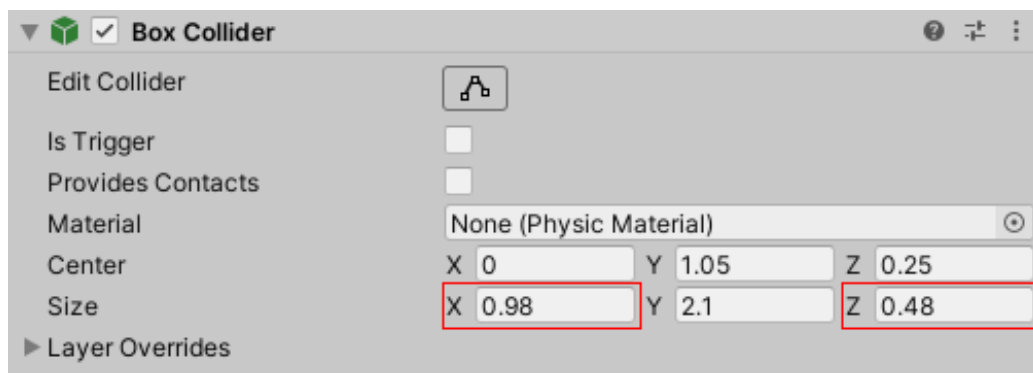
- Create your shelving unit model in a 3D modeling program (e.g., Blender, Maya). Ensure that the model includes designated areas or platforms where the shelves will be placed.
- Export your model as an FBX file, then import into your Unity project.

4.3.2 Creating the Shelving Unit Prefab

- In the Unity Editor, navigate to the `Assets/CheckoutFrenzy/Resources/Furnitures` folder. It is essential that all `Furniture` (`ShelvingUnit`'s base class) prefabs are placed in this folder (or a subfolder of it) for the game to load them correctly.



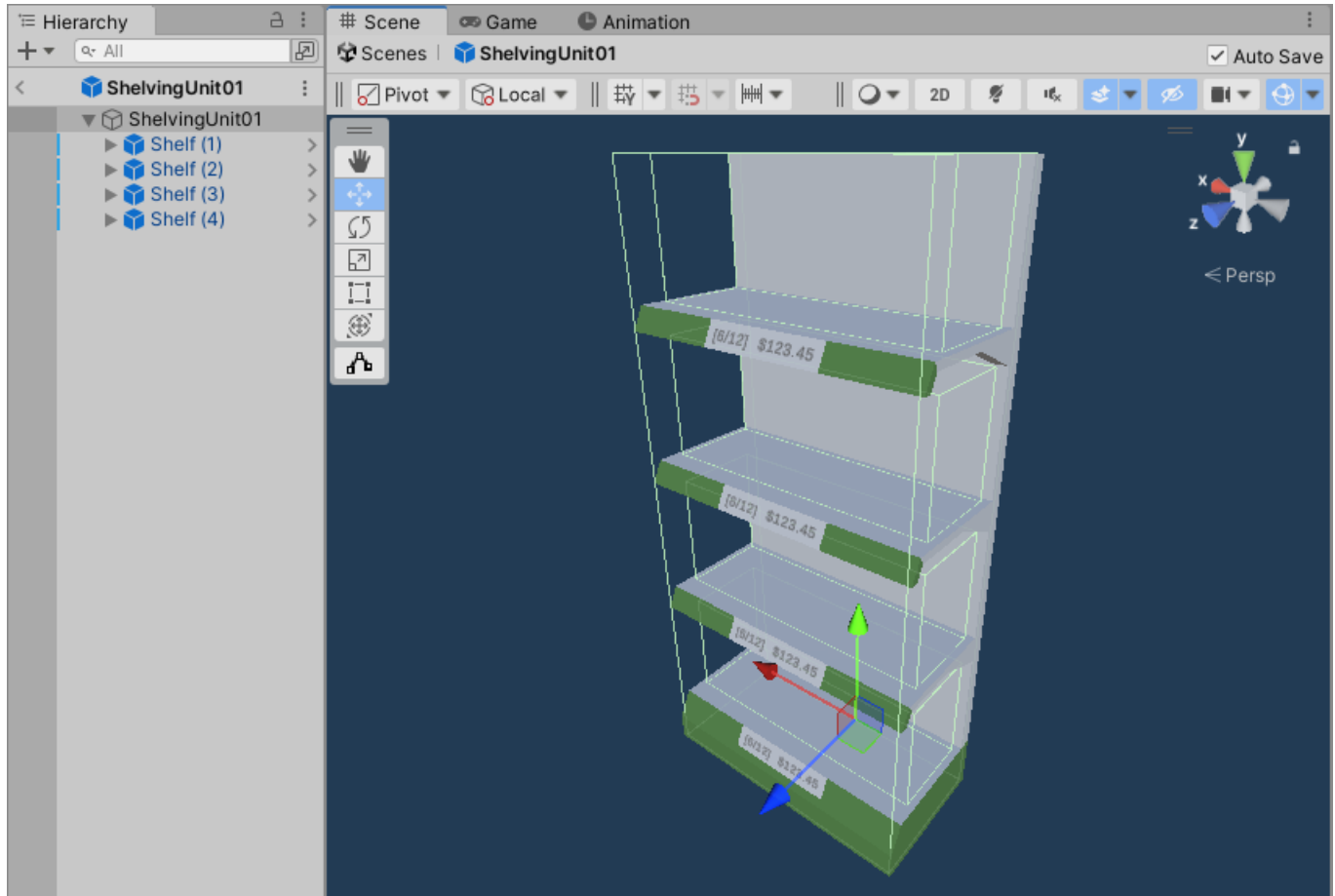
- Drag the imported 3D model from your project window into the scene to create an instance of the model.
- Add a `ShelvingUnit` component to the model instance. You can do this by selecting the model in the Hierarchy window, then in the Inspector window, click Add Component and search for "`ShelvingUnit`." This will also add a `Box Collider` component (as `ShelvingUnit` inherits from `Furniture` which requires it). *Important:* To prevent overlapping when placing furniture, slightly reduce the width (x) and length (z) of the `Box Collider`. This allows for precise placement next to other furniture. The game checks for overlapping colliders (excluding ground objects) during movement to prevent placement in invalid locations.



- Drag the model instance from the Hierarchy window back into the Assets/CheckoutFrenzy/Resources/Furnitures folder to create a prefab. Name it appropriately (e.g., "ShelvingUnitSmall," "ShelvingUnitLarge"). You can now delete the instance from the scene if you wish.

4.3.3 Configuring Shelving Unit

With the ShelvingUnit prefab selected, open the prefab in prefab mode.



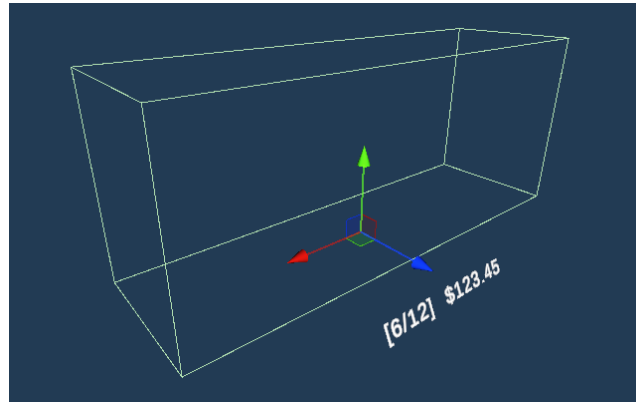
Configure the Furniture-related properties:



- **Furniture ID:** A unique numerical identifier for the furniture. Ensure that each piece of furniture has a distinct ID.
- **Name:** The name of the furniture (e.g., "Standard Shelf," "Large Fridge").
- **Icon:** A 2D image (Sprite) representing the furniture in the purchase menu.
- **Price (in cents):** The price of the furniture in cents.
- **Order Time (seconds):** The time (in seconds) it takes for the furniture to be delivered after purchase.
- **Section:** The type of product section this furniture is designed to display (e.g., Shelf, Fridge, Freezer). This is important for determining which products can be placed on the shelves of this shelving unit. The available options are:
 - **General:** For furniture that doesn't display products (e.g., decorations, trash cans).
 - **Shelf:** For standard shelving units.
 - **Fridge:** For refrigerated units.
 - **Freezer:** For freezer units.
 - **Rack:** For produce racks.
- **Main Renderer:** Drag the Mesh Renderer component of your shelving unit model to this field.
- **Moving Material:** Assign a material to be used when the furniture is being moved. This can be a semi-transparent or differently colored material to visually indicate that the furniture is being moved.

Adding Shelves to the Shelving Unit:

- In the Inspector window, you should see a button labeled "Add Shelf" provided by the `ShelvingUnitEditor` script.
- Click the "Add Shelf" button. This will instantiate a `Shelf` prefab as a child of the `ShelvingUnit`. The default Shelf prefab path is `Assets/CheckoutFrenzy/Prefabs/Shelf.prefab`. If you want to use a custom Shelf prefab, you'll need to modify the `SHELF_PREFAB_PATH` constant in the `ShelvingUnitEditor` script.



- Manually position and rotate the newly added `Shelf` within the shelving unit to your desired location.
- Adjust the `Shelf's Box Collider` to define the area where products can be placed.
- Position and rotate the `Info Text` child object on the `Shelf` to ensure proper display of product information (quantity/capacity).
- Add more shelves by repeating the "Add Shelf" process or duplicating existing shelves. Position each shelf within the shelving unit until the desired arrangement is achieved.

4.4 Adding and Configuring New Customers

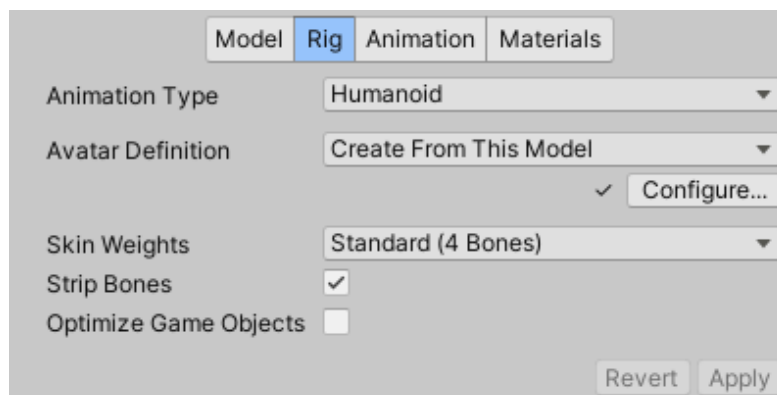
This subsection explains how to import and configure rigged humanoid characters as customers in Checkout Frenzy. This allows you to easily integrate character models from various sources.

4.4.1 Obtaining a Rigged Humanoid Character

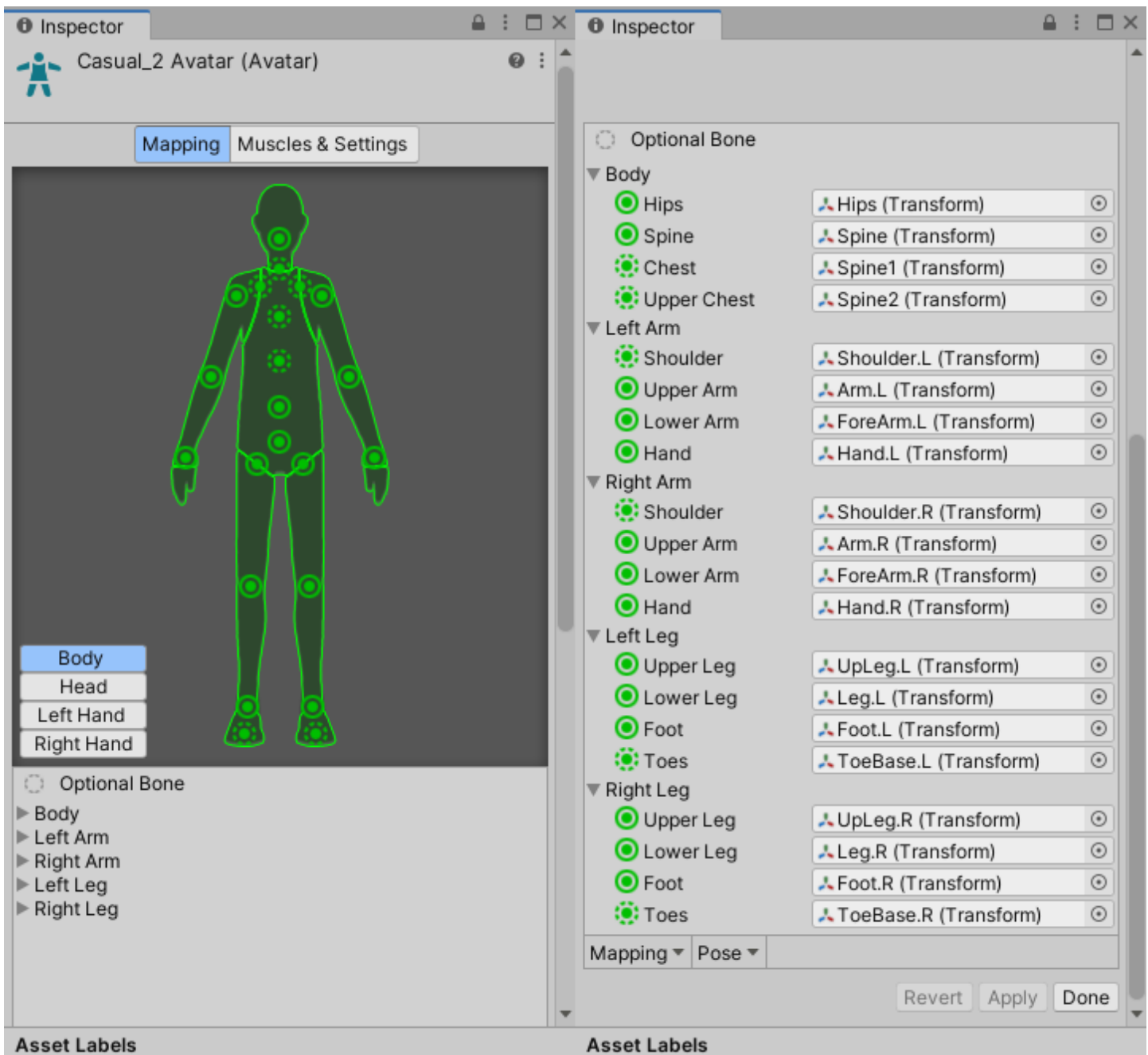
Acquire a rigged humanoid character model from your preferred source (e.g., asset store, custom creation). The model must be rigged with a humanoid rig compatible with Unity's humanoid animation system. This is essential for using the `Customer` script. Common file formats are FBX and glTF.

4.4.2 Importing the Character into Unity:

Import the character model file (FBX, glTF, etc.) into your Unity project. Pay close attention to the import settings to ensure proper functionality:



- **Animation Type:** Set this to Humanoid. This is absolutely critical for the `Customer` script to work correctly.
- **Avatar Definition:** If you're using a standard humanoid rig, you can usually leave this as "Create From This Model." If you have a custom rig, you may need to select an existing avatar or create one.
- Click **Configure...** to open the Humanoid Configuration window. Verify that all the required bones are correctly mapped to the corresponding body parts. Pay particular attention to the hands, as the `Customer` script interacts with hand bones for holding payment items (cash and card). If any bones are not mapped correctly, you'll need to adjust the mapping in this window.



- **Animations:** Unless you plan to add your own custom customer animations, leave the "Import Animations" option unchecked. This template already includes all the necessary animations.
- **Materials:** You can choose how Unity imports the materials associated with your model. You can typically leave this at the default settings unless you have specific material requirements.

4.4.3 Creating the Customer Prefab

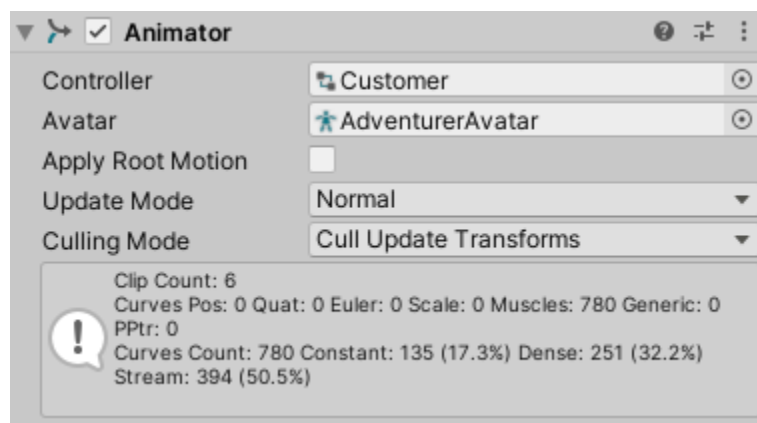
- In the Unity Editor, navigate to the folder where you want to store your Customer prefabs (e.g., `Assets/CheckoutFrenzy/Prefabs/Customers`).
- Drag the imported character model from your project window into the scene to create an instance of the model.
- Add a `Customer` component to the model instance. You can do this by selecting the model in the Hierarchy window, then in the Inspector window, click Add Component and search for "Customer." This will also add the required `Animator` (if not added yet) and `NavMeshAgent` components.
- Drag the model instance from the Hierarchy window back into your project folder to create a prefab. You can now delete the instance from the scene if you wish.

4.4.4 Configuring the Customer Prefab

With the `Customer` prefab selected, double click to open the prefab in prefab mode.

Animator:

- Assign the appropriate Customer Animator Controller to the `Animator` component. This controller should contain the animations and transitions required for customer behavior (walking, picking up items, paying, idle, etc.).



- Crucially, uncheck the "Apply Root Motion" option on the `Animator` component. The `Customer` script manages movement, so root motion should be disabled to prevent conflicts.

NavMesh Agent:

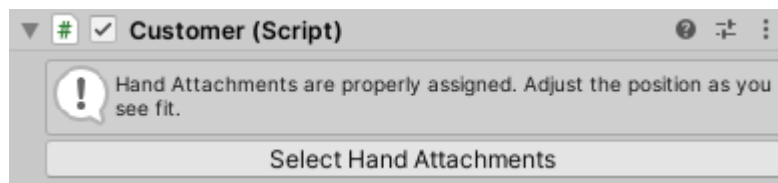
The default settings for the `NavMesh Agent` component are generally sufficient. You can leave them as they are unless you have specific requirements for customer movement speed, acceleration, or other pathfinding parameters.

Customer (Component):

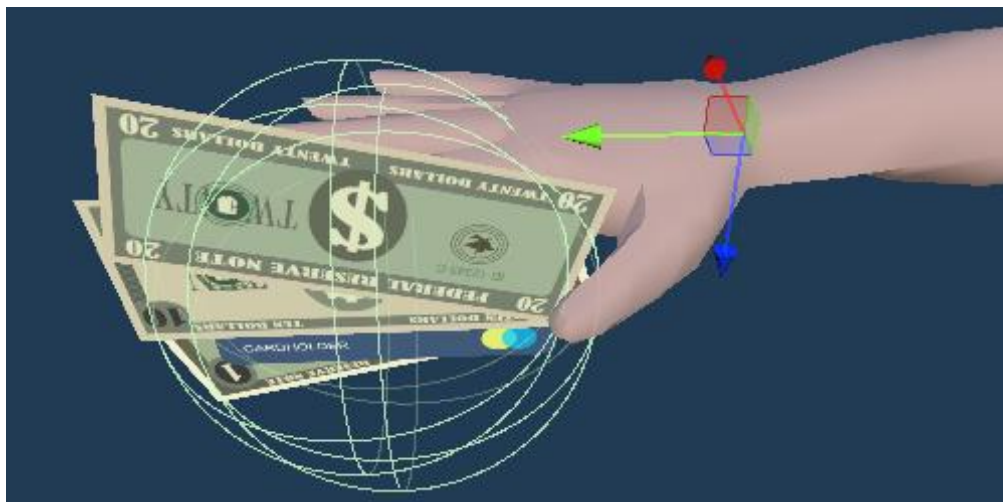
- A message will be displayed indicating whether the `HandAttachments` are properly assigned.



- A button labeled "Load Hand Attachments" will be displayed if the `HandAttachments` are missing. Clicking this button will automatically instantiate the `HandAttachments` prefab (located at `Assets/CheckoutFrenzy/Prefabs/HandAttachments.prefab`) and attach it to the customer's right hand bone.



- A button labeled "Select Hand Attachments" will be displayed if the `HandAttachments` are assigned. Clicking this button will select the `HandAttachments` object in the Hierarchy, allowing you to easily adjust its position and rotation on the customer's hand.



Hand Attachments:

This prefab manages the objects the customer holds in their hand, including products and payment methods.

The `HandAttachments` prefab should have the following structure:

- **Grip (Transform):** A transform used as the parent for products the customer is holding. This transform's position and rotation will be used to place the held product in the customer's hand.
- **Cash (GameObject):** A `GameObject` representing the cash payment option. This object should be activated when the customer is paying with cash.
- **Card (GameObject):** A `GameObject` representing the card payment option. This object should be activated when the customer is paying with a card.

Ensure both the Cash and Card `GameObjects` have colliders (e.g., a `Sphere Collider`). These colliders are used for interaction during the payment process. Players will "accept" these payment objects via raycasting.

4.4.5 Adding Customer Prefabs to the StoreManager

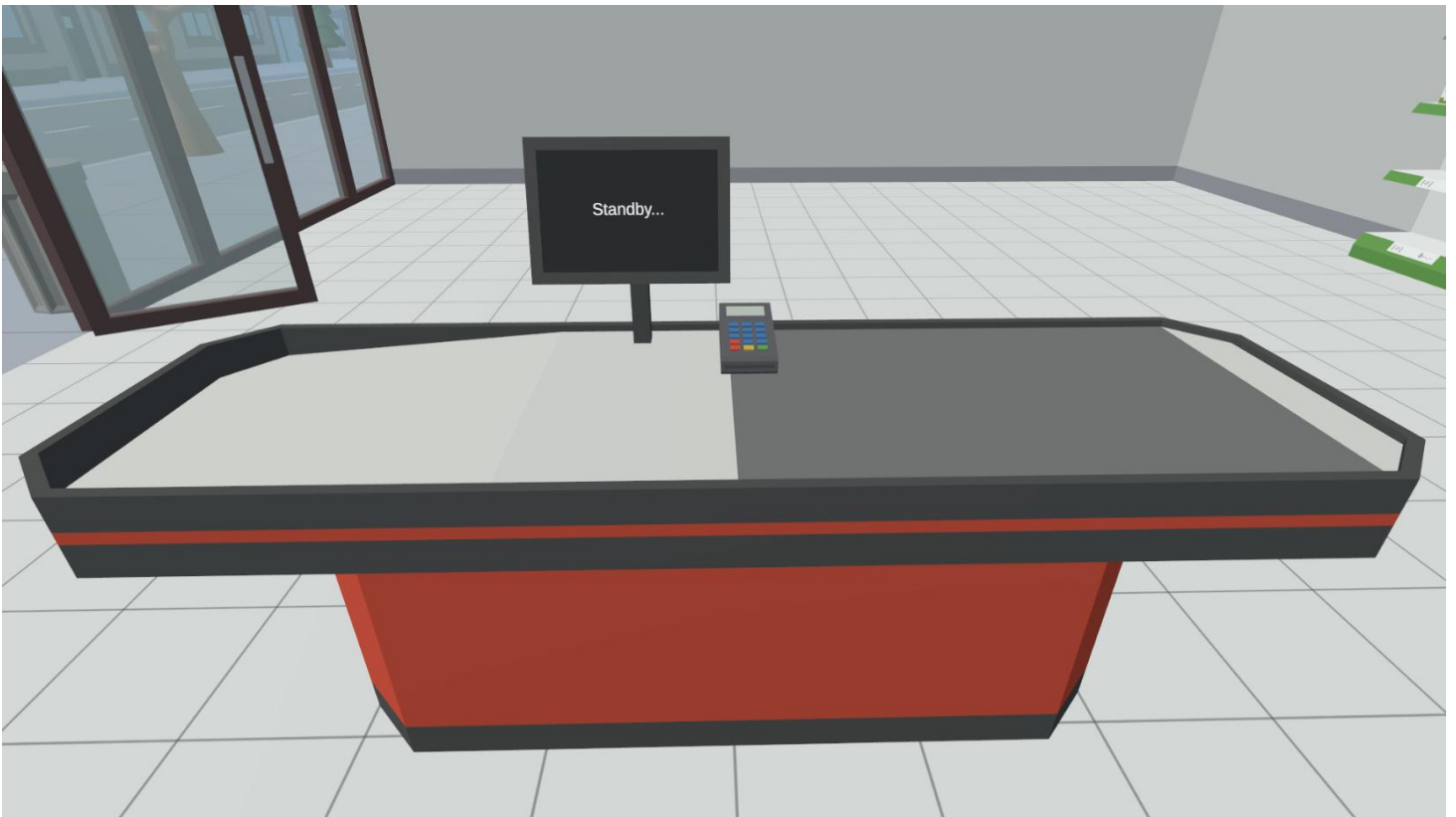
- Navigate to the `StoreManager` `GameObject` in the scene.
- Drag and drop your newly created customer prefab(s) into `customerPrefabs` list. The Store Manager will randomly select from these prefabs when spawning new customers.

4.5 Customizing Store Elements

This section details how to manage and configure key store elements, including the Checkout Counter, PC, Cashier Area, Expansions, and the building itself. These elements are crucial for customer interactions and store management.

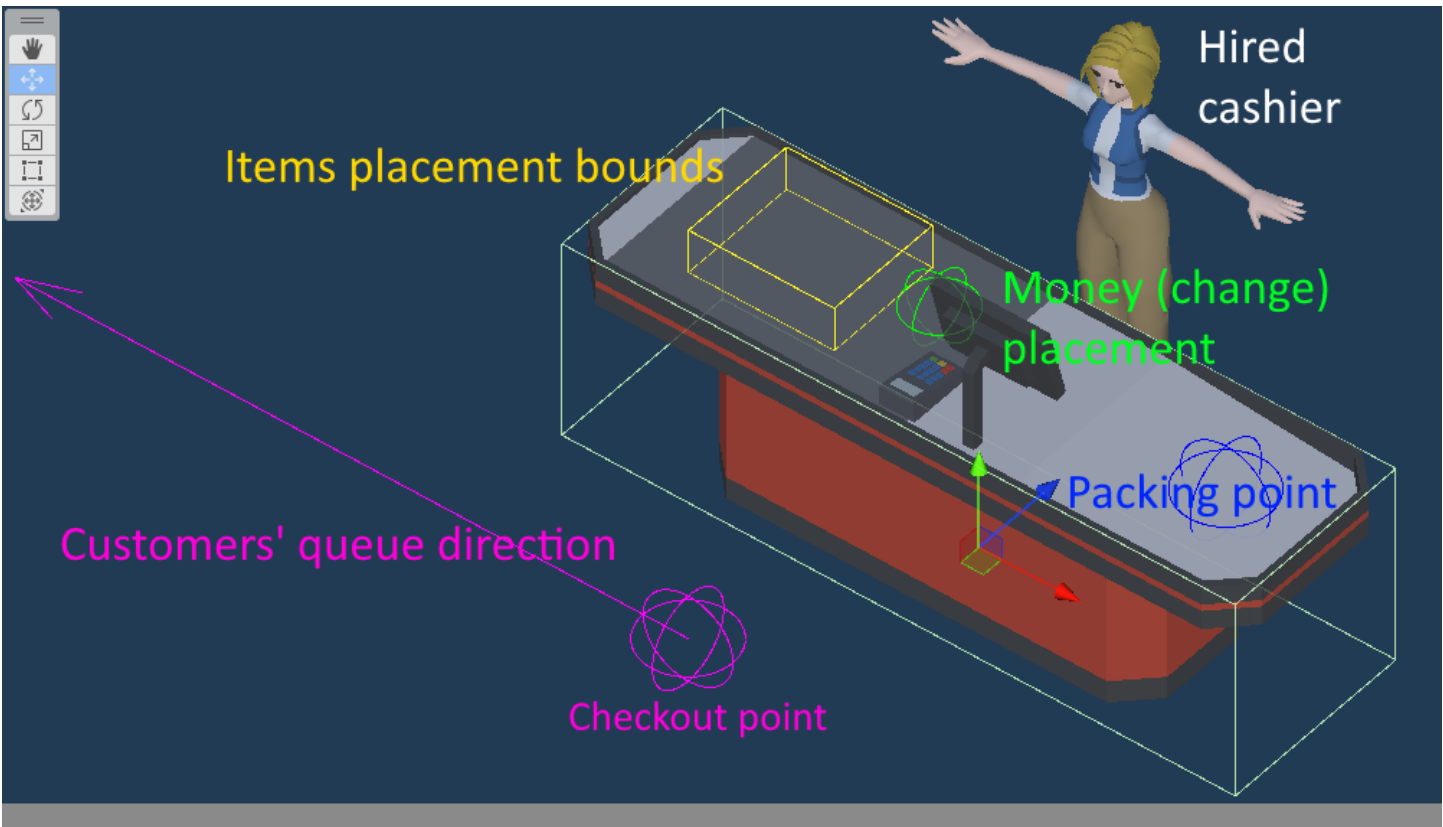
4.5.1 Checkout Counter

The Checkout Counter's appearance can be fully customized by replacing its constituent models, such as the desk, monitor, payment terminal, and even the cashier.



Customizing the Checkout Counter's Appearance:

After replacing the models, carefully adjust its properties to match your Checkout Counter's new appearance and functionality.



- **Collider:** Adjust the size and shape of the `Collider` component (e.g., `Box Collider` or `Mesh Collider`) on the Checkout Counter prefab. *This is crucial* as the Store Manager's `NavMesh Surface` generation takes the collider into account. An improperly sized collider can lead to pathfinding issues for customers. The collider should accurately represent the overall footprint of the checkout counter, including any parts that might obstruct customer movement.
- **Checkout Point:** This vector defines the local position (relative to the Checkout Counter's transform) where the first customer in line will stand. Adjust this to align with your new counter design. The gizmo in the Scene view (when the Checkout Counter is selected) will visualize this point.
- **Lining Direction:** This vector defines the direction in which the customer queue will form.
- **Packing Point:** This vector defines the local position where scanned items will be moved before being destroyed. Adjust this to match your counter's design.
- **Money Point:** This vector defines the local position where change will be given to the customer.
- **Placement Bounds:** These bounds define the area (relative to the Checkout Counter) where products can be placed for scanning. Adjust the size and center of these bounds to match the surface area of your counter. The gizmo in the Scene view (when the Checkout Counter is selected) will visualize this area as a wire cube.
- **Monitor Text:** `TextMeshPro` object that displays information on the checkout monitor. Adjust its position and rotation to match with your new monitor model.
- **Cashier Camera:** `Cinemachine Virtual Camera` that should be activated during checkout transactions to this field. This will provide a close-up view of the transaction. Adjust its position and rotation to match your new counter desk model.

Customizing the Cashier:

The cashier can also be customized by replacing the default model with a rigged humanoid character.

- **Replacing the Cashier Model:** Replace the placeholder cashier model with your custom rigged humanoid character model. Ensure the model is correctly scaled and positioned within the Checkout Counter prefab.
- **Assigning the Animator Controller:** Assign the appropriate **Cashier Animator Controller** to the cashier model's `Animator` component. This controller should contain the animations for cashier actions (e.g., scanning items, taking payments, idle animations).
- **Adding the Cashier Script:** Add the `Cashier` script to the cashier model. This script will handle the cashier's logic and interactions with customers.

4.5.2 PC (Order Terminal)

Customizing the PC's appearance follows the same general process as the Checkout Counter: replace the existing models (monitor, keyboard, case, etc.) with your custom models. Ensure the new models are appropriately scaled and positioned within the PC prefab.

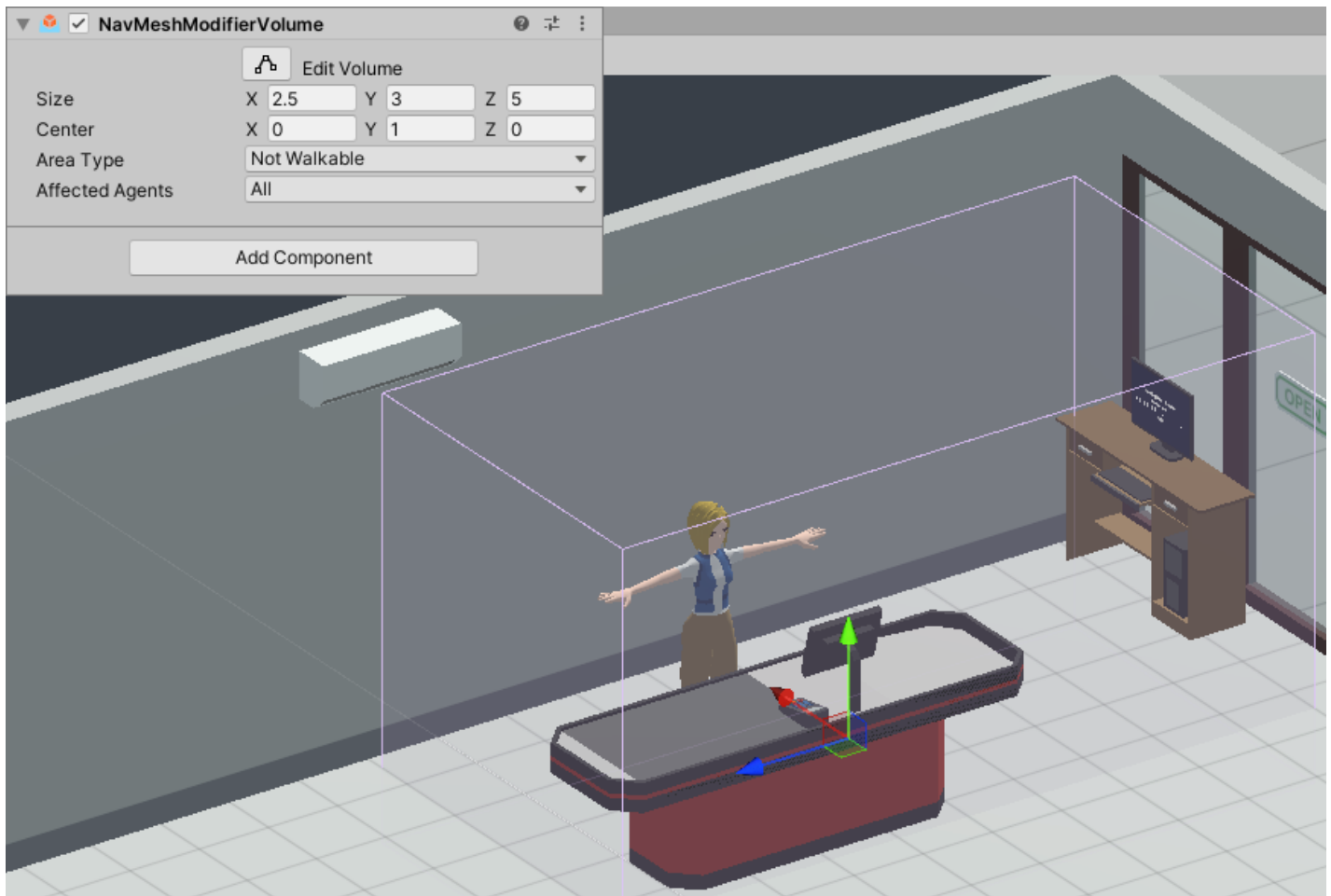


Adjusting Properties: After replacing the models, adjust the following properties in the PC component to match your PC's new look and functionality:

- **Collider:** Adjustments are exactly the same as Checkout Counter.
- **Monitor Camera:** `Cinemachine Virtual Camera` that should be activated when the player interacts with the PC to this field. This camera will provide a close-up view of the PC monitor. Adjust the position, transform, and the field of view so that PC Monitor UI will perfectly match the dimensions of the monitor model.
- **Monitor Text:** `TextMeshPro` element that simulate the loading bar on the PC monitor model when player interacts with the PC and open Order Program interface.
- **Blocker:** Collider or set of Colliders with `InteractionBlocker` component attached, placed behind the PC to prevent interactions from there.

4.5.3 Cashier Area

Defined using a `NavMeshModifierVolume` component, the Cashier Area is a designated region where customers are not allowed to pathfind. This area is typically used to define the space behind the checkout counters, preventing customers from walking through the cashier.



4.5.4 Expansions

Expansions allow you to expand your store, adding more space, and increasing customer capacity. Each expansion has specific requirements and unlocks new possibilities for your store.

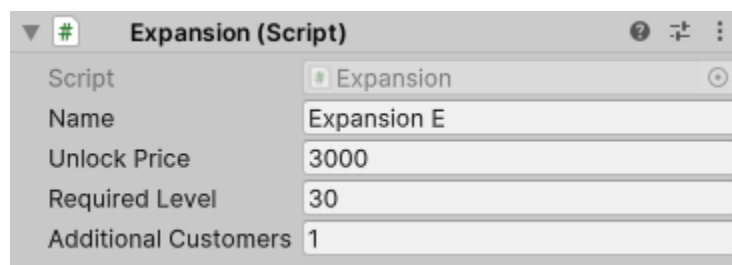
1. Creating the Expansion:

- Create a new empty `GameObject` for each expansion. This `GameObject` will serve as the parent container for all elements associated with the expansion. It's crucial to organize the expansion's contents within this parent object.
- Populate the expansion `GameObject` with its constituent elements. These elements should be categorized based on their activation state after purchase:
 1. **Initially Active Elements (Deactivated After Purchase):** Include `GameObjects` that are initially active in the scene but should be deactivated after the expansion is purchased. Examples include:
 - Colliders that block access to the expansion area before it's unlocked.
 - Visual blockers (e.g., construction barriers, temporary walls).
 - Informational signs (e.g., "Coming Soon," "Under Construction").
 - Any other elements that should be visible or interactable before the expansion is purchased.

2. **Initially Inactive Elements (Activated After Purchase):** Include GameObjects that are initially inactive in the scene but should be activated after the expansion is purchased. Examples include:

- New decorations or furniture.
 - New sections of the store (e.g., new rooms, expanded areas).
 - New customer spawn points.
 - Any other elements that should be visible or interactable only after the expansion is purchased.
- Add an `Expansion` component to the root of the expansion `GameObject`. This component will manage the activation state of the expansion's elements.

2. Configuring the Expansion Component:



The screenshot shows the Unity Inspector window for the 'Expansion (Script)' component. The component is attached to a GameObject named 'Expansion'. The configuration fields are as follows:

Property	Value
Script	Expansion
Name	Expansion E
Unlock Price	3000
Required Level	30
Additional Customers	1

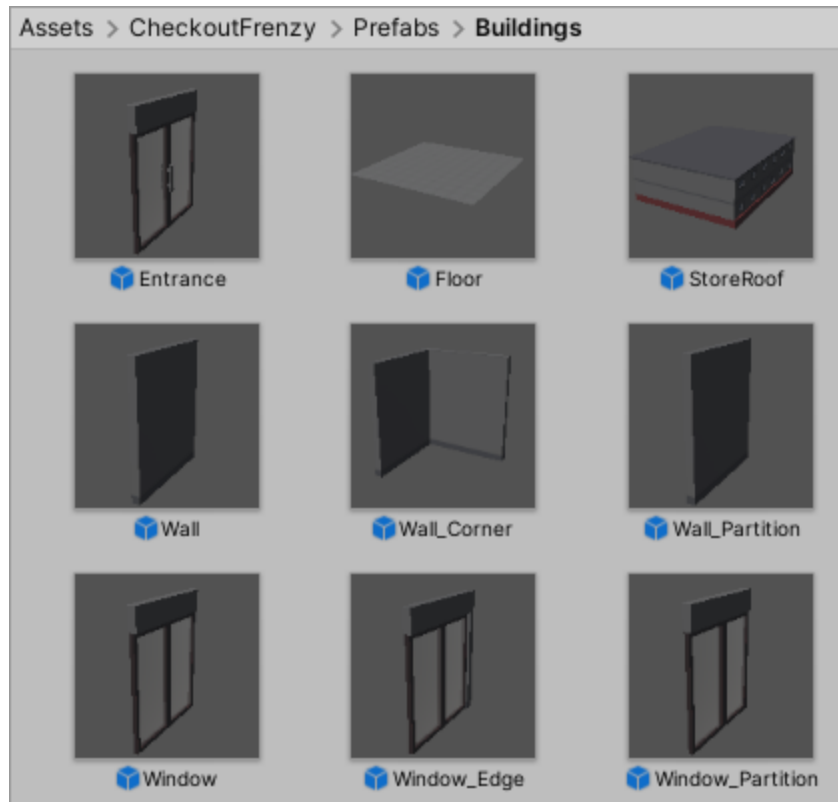
- **Name:** Enter a descriptive name for the expansion.
- **Unlock Price:** Set the price in currency required to unlock this expansion.
- **Required Level:** Specify the minimum player level / store level required to unlock this expansion.
- **Additional Customers:** Enter the number of additional customers this expansion adds in the store. This value should be added to the store's maximum customer capacity when the expansion is unlocked.

3. Assign to Store Manager:

Assign all your expansions to `StoreManager`'s `Expansions` field in the scene.

4.5.5 Store Building

The Store Building is constructed from modular parts, allowing for extensive customization. These modular parts, such as walls, windows, floors, and entrances, can be connected piece by piece to create the desired store layout. The building system uses a grid of 2.5 units to facilitate precise placement and alignment of the modular parts.



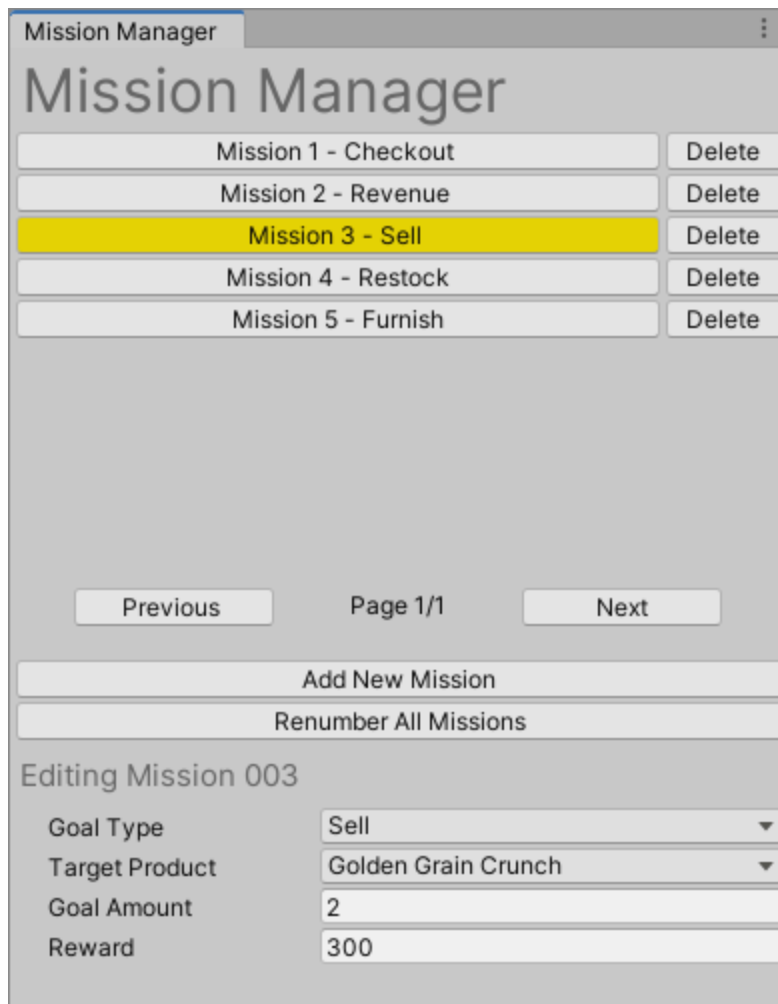
You have complete freedom to modify the appearance of these parts – change textures, swap models entirely, or even create entirely new building components.

Important Rule: The floor parts must be on the "Ground" layer. This is essential for furniture placement. The furniture placement system relies on the "Ground" layer to determine valid placement locations. If the floor is not on the "Ground" layer, you will encounter problems placing furniture correctly.

4.6 Adding and Customizing Missions

Missions provide objectives for the player to complete, rewarding them upon success. Missions are managed through the *Mission Manager Window*.

Important: Do not confuse the *Mission Manager Window* (used for creating and editing missions in the editor) with the `MissionManager` component in the scene (which handles missions at runtime). They serve different purposes.



Using the Mission Manager:

The Mission Manager window allows you to create, edit, and delete missions.

1. **Opening the Mission Manager:** Navigate to *Tools > Checkout Frenzy > Mission Manager* to open the Mission Manager Window.
2. **Mission List:** The window displays a list of existing missions, paginated if there are more missions than can be displayed at once (10 missions per page). You can navigate between pages using the "Previous" and "Next" buttons.
3. **Selecting a Mission:** Click on a mission in the list to select it for editing. The selected mission will be highlighted.
4. **Adding a New Mission:** Click the "Add New Mission" button to create a new mission. The new mission will be automatically assigned the next available `Mission ID`.
5. **Deleting a Mission:** Select a mission and click the "Delete" button to delete it. A confirmation dialog will appear to prevent accidental deletion.
6. **Renumbering Missions:** Click the "Renumber All Missions" button to renumber all existing missions sequentially. This will also update the filenames of the mission assets. A confirmation dialog will appear before renumbering.

7. **Editing Mission Details:** When a mission is selected, its details are displayed in the lower section of the window. You can edit the following properties:

- **Goal Type:** Select the type of goal for the mission (e.g., Checkout, Revenue, Sell, Restock, and Furnish). *See section 3.4 for more details.*
- **Target Product/Furniture:** If the goal type requires a specific product or furniture (Sell, Restock, and Furnish), use the dropdown menu to select the target item.
- **Goal Amount:** Enter the target amount required to complete the mission. The units for this value depend on the goal type (e.g., number of checkouts, revenue in cents, number of items sold or restocked, etc.).
- **Reward:** Enter the reward given to the player upon completing the mission.

8. **Saving Changes:** Changes to a mission are automatically saved when you edit a field.

Mission Asset Storage:

Mission assets are stored in the `Assets/CheckoutFrenzy/Resources/Missions` folder. It is recommended not to manually move or rename these assets, as this can break the Mission Manager's functionality. Use the Mission Manager Window to add, delete, and renumber missions, as it handles the asset management correctly.

5. Third Party Assets

This project utilizes several third-party assets to streamline development and enhance gameplay functionality. Below are the assets used, along with their respective roles within the game:

5.1 DOTween (v1.2.632)

DOTween is a fast and efficient animation engine used for tweening, which allows smooth transitions between values over time. It is used extensively in this game for various purposes, including:

- **UI Transitions:** Smooth animations for UI elements, such as fading and scaling.
- **Object Animations:** DOTween is used to smoothly animate various objects in the game, such as transitioning between different states or movements, rotating objects toward specific targets, and applying effects like scaling or fading.

For more information on how DOTween works, please refer to the official documentation at [DOTween](#).

5.2 SimpleInput (v1.1.7)

SimpleInput is an asset designed to handle input from touch and joystick controls, which is crucial for mobile platforms. In this game, SimpleInput provides:

- **Virtual Joystick:** Used for character movement on mobile devices, allowing players to control characters via an on-screen joystick.
- **Touch Input Handling:** Provides flexible input management for different types of touch interactions, ensuring smooth control in both gameplay and UI navigation.

You can learn more about SimpleInput at [SimpleInput GitHub](#).

5.3 Models

The following 3D models are licensed under Creative Commons Zero (CC0):

- [Ultimate Modular Men Pack](#) by Quaternius
- [Ultimate Modular Women Pack](#) by Quaternius
- [Cars Pack](#) by Quaternius
- [City Kit \(Commercial\)](#) by Kenney

5.4 Audio

The following background music tracks are licensed under Creative Commons Zero (CC0):

- [Picnic On The Seine](#) by John Bartmann

This concludes the documentation for Checkout Frenzy.