Cooking Star

Fast food & burger delivery

Unity Asset

Requires Unity 2020.3.30f1+
Works on Android, iOS, WebGL, Windows and Mac

Dear Customer,

Thank you for purchasing this game template. Here you can find the most important information on how to use this project efficiently. All classes are fully commented, but if you ever needed a hand on anything, feel free to contact us at www.finalbossgame.com . We'll try our best to support you with your questions as soon as possible.

Overview

Cooking star template is an easy to use and learn project that can greatly speed up your prototypes, and can be used as a strong base for further development and experiments. It has all nuts and bolts to build a fully functional cooking game. It came with:

- **8** premade customers (with different textures and patience level)
- **15** predefined ingredients (can be raw or processed)
- **15** predefined products (based on the available ingredients)
- **4** predefined Side-Request items
- Candies/Salads to relax customers and refill their patience bar.
- Full game flow with menu, shop, coins, missions, tutorial & game scenes.

However, you are not limited by these numbers and can easily define and add more items to the game to deliver a rich game play experience to your players.

The game accepts both touch and mouse inputs, and thus, can be tested on **Android**, **iOS**, **WebGL** and **Stand-Alone** platforms.

Classes

This Starter kit uses a few separate classes to control the game's main routine. All these classes are fully commented and you can easily guess the dataflow. But I try to introduce them here also:

- **MainGameController:** This is the main controller of the game. It holds all static variables, manages game time settings, money, score, and also is in charge of random customer creation. This class will also manage game's start/finish/completed/failed states.

- **CustomerController:** This is the main customer controller. It controls customer events, animations and is responsible for time (patience) management of each customer. It also determines each customer's wish and its ingredients and then show them in a bubble over customer's head. It also can receive delivery from other classes and check them to see if they match the given order or not and then respond accordingly.

- **ProductManager:** This simple class is the heart of the game. Here you can define and edit products and set their special attributes. Each product has Price and Ingredients and, in this class, you can easily define these values. We show you how to add new products, later on.

- **IngredientsController:** This class controls the behavior of available ingredients on screen, if they can be picked and dragged into delivery Plate, and also manages their movements. This also handles ingredients status, whether they should be processed before landing in delivery plate, or if they can be used right away.

- **PlateController:** This class is responsible for all things related to cooking and delivery Queue. It will allow player to drag delivery plate to towards customers, and towards trash bin in order to deliver or discard it. This is also responsible for freeing and resetting the main Queue array, whenever needed.

- **ProductMover:** This class will manage products movements (from ingredients panel to delivery Plate) and let player drag and move them with mouse or touch. It also checks if the products are dragged over delivery Plate and if so, add their ID to main delivery queue. If player pick a product and release it without landing it over something interactive, this class will automatically destroy that instantiated game Object.

- **TrashBinController:** This class is responsible for receiving and destroying wrong deliveries. Player can move their undesired cookings to this Object to discard them.

- **CandyController:** Candies are used to entertain bored customers and refill their patience bars. This class manages the drag, drop and message handling of this routine.

- **SideRequestController:** Side Request items are used besides the main order of customers. This class handles the creation and delivering of these items.

How to add new Products, Ingredients, Side-Reqs, Customers & Levels?

To add a new customer to the game:

1. Pick a customer prefab from folder “Prefabs/Customers” and duplicate it with Ctrl+D;
2. Rename the new customer prefab to something appropriate.
3. In the inspector, set the desired **customerPatience** (in seconds) and their specific needs via **customerNeeds** (ID: int). Eventually you can leave the customerNeeds at 0 to let the controller pick a random product from products Array every time.
4. You can set to show/hide customer’s Wishlist ingredients by setting/unsetting its value from inspector.
5. Set the available products for this customer by carefully choosing products within **AvailableProducts** array.
6. Set the available ingredients for the selected products (in step 5) via **AvailableIngredients** array. Please note that if you add a product to **AvailableProducts** array, and forget to add it’s required ingredients to **AvailableIngredients**, it leads to an error.
7. Define new material and texture for this new customer and set them via prefab’s inspector.
8. Go to GameController object in the scene and update its **customers** Array with your newly created customer prefabs.
9. Repeat these steps to add as many customers as you like.
10. You are done!

To add a new product:

1. Pick a product prefab from “Prefabs/Products” and duplicate it with Ctrl+D;
2. Rename the new product prefab to something appropriate.
3. From the inspector, set the **Price** (Int), **TotalIngredients** (int, range from 1 to 6) and **ingredientsIDs** (Int, with index from **1** to **TotalNumberOfAvailableIngredients [currently 15]**)
4. For example, the definition for a new product is like this:
    a. Price = 150
    b. TotalIngredients = 6
    c. IngredientsIDs: [1, 3, 8, 10, 12, 15]
5. Define new material and texture for this new product and set them via prefab’s inspector.
6. Select all customers prefabs in “Prefabs/Customers” and update their AvailableProducts array with the newly created product. This will enable them to be able to select this new product as their wishlist.
7. Repeat above steps to add as many products as you like.
8. You are done!

To add a new normal ingredient (which doesn't need processing before use):

1. Pick a normal ingredient prefab from "Prefabs/Ingredients" and duplicate it with Ctrl+D;
2. Rename the new ingredient prefab to something appropriate.
3. Define new material and texture for this new ingredient and set them via prefab's inspector.
4. Do not edit prefab's FactoryID. (Must be left at 0).
5. Do not touch other public variables in inspector. Leave them at their default state.
6. We are done with Prefab.
7. Go to Hierarchy and select **IngredientsHolder** game object. It contains 15 ingredients already.
8. Duplicate one of the available **Ingredient-xx** child objects and position it properly in the editor.
9. Set the newly created **ingredient-xx** child object's Factory ID to an incremented size (16, 17, 18 , … and so on).
10. Select all children **ingredient-xx** objects (from hierarchy) and update their Ingredients array with the newly created ingredient prefab from step 1.
11. Select all customer prefabs in "prefabs/Customers" and update their **availableIngredients** array with the newly created prefab from step 1.
12. You are done!

To add a new raw ingredient (which needs processing [in a machine] before use):

1. Pick a raw ingredient prefab from "Prefabs/Ingredients" and duplicate it with Ctrl+D; (For example pick "*Ingredient-Type-02*" which is a raw ingredient)
2. Rename the new ingredient prefab to something appropriate.
3. Define new material and texture for this new ingredient and set them via prefab's inspector.
4. Define 2 new "Raw" and "Processed" materials for this ingredient which contain appropriate image for each state of the ingredient. Then set the "BeforeAfterMat" array size to 2 and set the *Raw* mat as the first and *Processed* mat as the second parameter.
5. Do not edit prefab's FactoryID. (Must be left at 0).
6. Do not touch other public variables in inspector. Leave them at their default state. (Just edit the *beforeAfterMat* in step 4).
7. We are done with Prefab.
8. Go to Hierarchy and select **IngredientsHolder** game object. It contains 15 ingredients already.
9. Duplicate one of the available **Ingredient-xx** child objects and position it properly in the editor.
10. Set the newly created **ingredient-xx** child object's Factory ID to an incremented size (16, 17, 18 , … and so on).
11. Set the "needsProcess" value to true, for this raw ingredient.
12. Set the processor machine's tag as the "ProcessorTag" value. By default, this will be set to "*grill*".
13. Select all children **ingredient-xx** objects (from hierarchy) and update their Ingredients array with the newly created ingredient prefab from step 1.
14. Select all customer prefabs in "prefabs/Customers" and update their **availableIngredients** array with the newly created prefab from step 1.
15. You are done!

To add a new Side-Request item:

1. Pick a side-request prefab from “Prefabs/ SideRequests” and duplicate it with Ctrl+D;
2. Rename the new side-request prefab to something appropriate.
3. Define new material and texture for this new side-request and set them via prefab’s inspector.
4. Do not edit prefab’s sideReqID. (must be left at 0).
5. We are done with Prefab.
6. Go to Hierarchy and select **SideRequestsHolder** game object. It contains 3 sideRequest already.
7. Duplicate one of the available **SideRequest-xx** child objects and position it properly in the editor.
8. Set the newly created **SideRequest-xx** child object’s sideReqID to an incremented size (4, 5, 6 , … and so on).
9. Select all children **SideRequest-xx** objects (from hierarchy) and update their sideRequest array with the newly created sideRequest prefab from step 1.
10. Select all customer prefabs in “prefabs/Customers” and update their **availableSideRequest** array with the newly created prefab from step 1.
11. You are done!

To add a new Side-Request item that needs processing before use (like coffee):

1. Pick a side-request prefab (preferably *sideRequest-Type-04-C#*, as it is already configured) from “Prefabs/ SideRequests” and duplicate it with Ctrl+D. Please note that siderequest items that needs processing uses two materials (one for raw and one for processed state) in their sideRequestMover class.
2. Rename the new side-request prefab to something appropriate.
3. Define two new materials and textures (raw and processed) for this new side-request and set them via prefab’s inspector.
4. Do not edit prefab’s sideReqID or needsProcess value. (Both must be left at 0).
5. We are done with Prefab.
6. Go to Hierarchy and select **SideRequestsHolder** game object. It contains 4 sideRequest already and the forth one is a sidereq that needs processing before use. Look at the controller for this game object with marked “Needs Process” and two before/after material. You also have to set the tag of the machine/gameObject that has to process this item via “Processor Tag” field.
7. Duplicate one of the available **SideRequest-xx** child objects (preferably *sideRequest-04,* as it is already configured) and position it properly inside the editor.
8. Set the newly created **SideRequest-xx** child object’s sideReqID to an incremented size (4, 5, 6 , … and so on). Set the “Needs Porcess” flag to true and also set the tag of the processor machine/game object in the respective field. As the last step, set two before/after materials for this item.
9. Select all children **SideRequest-xx** objects (from hierarchy) and update their sideRequest array with the newly created sideRequest prefab from step 1.
10. Select all customer prefabs in “prefabs/Customers” and update their **availableSideRequest** array with the newly created prefab from step 1.
11. You are done!

To add a new mission to Level-Selection scene (extending the career mode):

1. Select and duplicate (ctrl + d) one “MissionHolder” object from the available buttons on “Level-Selection” scene. You may need to duplicate a new row (ButtonsRow object) in order to spawn two new mission holder objects.
2. Rename this cloned object to something appropriate like “Level-9” …
3. Select the button and configure its class properties (Career-Level-Setup) via inspector:
    a. **MissionID**: index of the level we want to load. Must be unique.
    b. **MissionName**: Optional name for this mission that shows beneath the mission number.
    c. **Mission Reward**: Money that will be given to player after beating the level.
    d. **Mission Target**: mission of the level to beat.
    e. **Mission Time**: time given to player to beat the level mission money. (In seconds)
    f. **CanUseCandy**: set if player is allowed to use candy to refill customer’s patience bars.
    g. **AvailableProducts**: array containing available indexes for products that are available in this level. Customers will always check their wishes with this array and choose from available products you carefully select for each level.

*Note:

You can select a few simpler products for the starting levels and then increase the difficulty of your game by choosing complex products (consisting of many ingredients) in larger quantities. Look at the default configuration for levels 1 to 8 to get familiar with the setup.

Support

If you have any questions, feel free to send us a message at https://www.finalbossgame.com and we will get back to you as soon as possible.