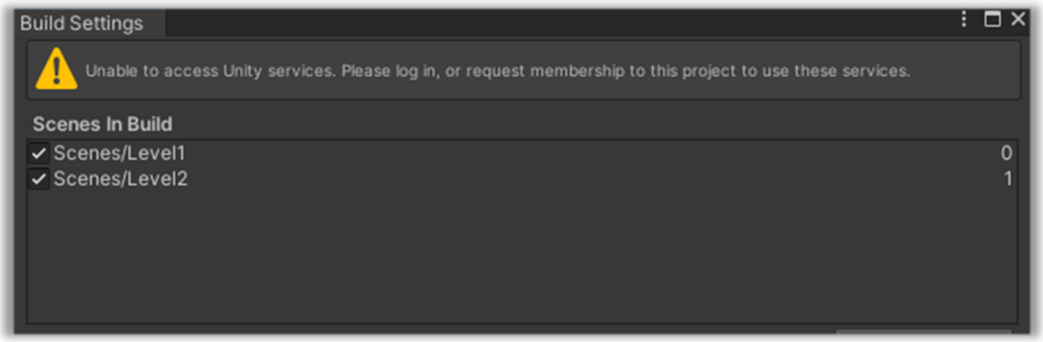
Fast Food Universe

Thank you for purchasing „Fast Food Universe”, I hope that you will enjoy using it. If you have any questions or issues, please don’t hesitate to contact me.

CONTENTS

1. Setup

- **Use 2021.3.1 version of Unity** for the project to avoid issues caused by version difference
- **Drag&Drop** the **.unitypackage** file to your „Assets" folder inside „Projects", or go to „**Assets**" window -> „**Import New Asset**" -> then select the **.unitypackage** file
- Or extract the **.zip** root folder, and open it inside **Unity Hub**
- Open **„Level1"** scene (***Assets -> Scenes***). Make sure to **add all levels** in the **build settings** before building the game



Build Settings

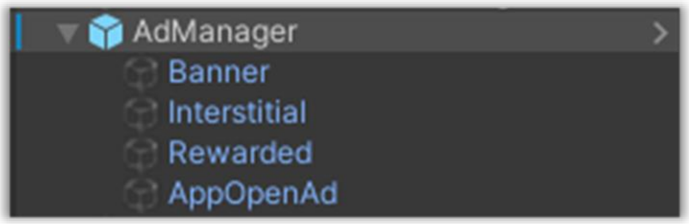
Unable to access Unity services. Please log in, or request membership to this project to use these services.

Scenes In Build

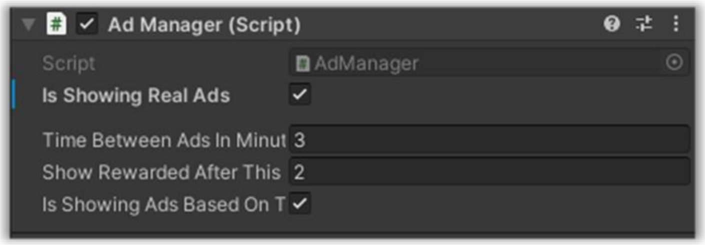
```
✓ Scenes/Level1    0
✓ Scenes/Level2    1
```

2. Adding your Admob Ads to the game

- **„AdManager”** gameobject has all the ad types as children. Make sure to **ENABLE** this gameobject for testing the ads, and for making the final build.



- Set how frequently are the ads are being showed, at „AdManager” gameobject. Also set if you want to show the test ads or real ads (make sure to set it to show real ads for the final build)


Ad Manager (Script)

Script AdManager

Is Showing Real Ads ✓

Time Between Ads In Minut 3

Show Rewarded After This 2

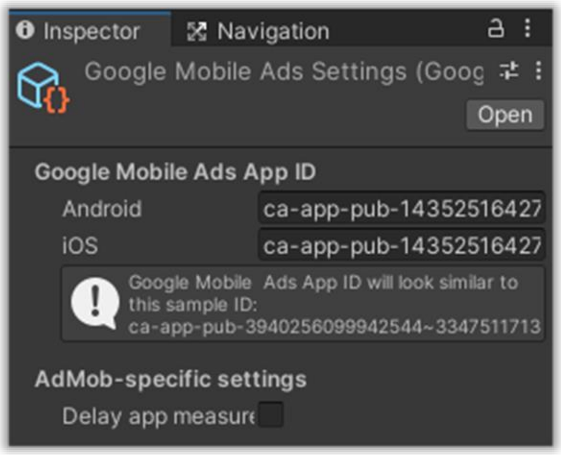
Is Showing Ads Based On T ✓

- Set up your test and real ad IDs at Banner, Interstitial, Rewarded, and AppOpenAd gameobjects (children of AdManager gameobject)
- **How to create your own Admob Ad IDs (CLICK HERE)**

Common issues with AppOpenAd

- *Please note that testing „AppOpenAd" in Unity does not behave like the real App Open ad. Also some times its buggy and you cannot close the ad in the Unity Editor, but it will work properly in when testing on device.*
- *„AppOpenAd" is displayed when the game opens. However, on iOS, the game is not approved if the player cannot play the game before seeing an ad. So at the very first time the player opens the game, the AppOpenAd should not show.*

- You need to set up your Android/iOS **App IDs** at: *Assets -> Google Mobile Ads -> Settings*



Google Mobile Ads App ID

Android ca-app-pub-14352516427

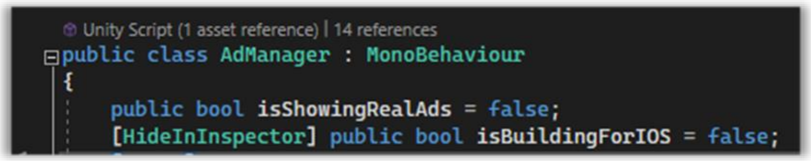
iOS ca-app-pub-14352516427

Google Mobile Ads App ID will look similar to this sample ID: ca-app-pub-3940256099942544~3347511713

AdMob-specific settings

Delay app measure

- In **AdManager**, set **isBuildingForIOS** to true if you are building for iOS release, set it false if building for Android



```
Unity Script (1 asset reference) | 14 references
public class AdManager : MonoBehaviour
{
    public bool isShowingRealAds = false;
    [HideInInspector] public bool isBuildingForIOS = false;
```

3. Explanation of scripts

VibrationManager.cs

- This script can be used to add haptic feedback to your application. It provides three types of vibration: LightImpact, SoftImpact, and Error. You can turn on or off the vibration using the canVibrate variable. The vibration strength is automatically adjusted based on the device platform. The script also includes a cooldown time and a function for restoring the vibration amplitude to its original value. You can call the Vibr_LightImpact(), Vibr_SoftImpact(), and Vibr_Error() functions to trigger the respective types of vibration.

PlayerMcDonalds.cs

- Includes various functions for the player, such as activating and deactivating a scooter, moving away from a certain position, and handling collisions with certain objects.
- The script starts by defining a public static instance of the player character, which is used to reference the player throughout the game.
- The script also defines private variables for the player's rigidbody, animator, and joystick (which is a user interface component that allows for movement control). There is also a public float variable for the player's movement speed.
- The DeactivateScooter and ActivateScooter functions handle the player's scooter state, changing the player's movement speed, adjusting their position and collider, and playing a particle effect when the scooter is activated or deactivated.
- The MoveAway function moves the player away from a given position by gradually adjusting their position until they reach a target position (This is used when a table gets bought, player gets pushed away).
- The OnTriggerEnter function handles collisions with certain objects, such as arrows (tutorial) and chip spawners (food spawners), and calls other functions accordingly. The OnTriggerExit function is called when the player stops colliding with certain objects.

- The RigOn_Chips and RigOff_Chips functions turn on and off an animation for the player's hand holding chips (foods), and the RigOn_Cheater and RigOff_Cheater functions do the same for a different animation for the player's hand.
- The Update function handles the player's movement, setting the player's velocity based on the joystick direction and updating the player's rotation accordingly. The function also handles the player's animation, turning it on or off depending on the joystick's direction. Finally, the function includes some time scaling code for testing purposes.

Guest.cs

- The guest has a cup holder and can be a VIP or a regular guest. The code initializes the guest's appearance, assigns an animator and a cup holder, and determines if the guest needs food. If the guest is a VIP, it is seated at the VIP table, otherwise, it joins a waiting line to be seated.
- There are also functions for the guest to pick up and put down the cup holder, move to different locations, and check if another waiting line is better.es various functions for the player, such as activating and deactivating a scooter, moving away from a certain position, and handling collisions with certain objects.
- Includes a function to find an available seat, move the character to the seat, and deliver food to the customer. It also includes functions to move the character forward, move the character to a target position, and rotate the character towards its parent.
- The script has a few private variables to keep track of the parent object and instances of components in the game, and a few public variables to determine if the character can move and hold a cup. The script is triggered when the game is running and updates the position and behavior of the character based on various conditions.
- The FindSeat method uses a random selection process to find an available seat for the character. If the seat is available, the character is moved to the seat by setting its parent to the seat transform and invoking the MoveTo method.
- The MoveTo method is responsible for moving the character to the target position, and the MoveForward method is called to move the character forward towards the target.

- The Update method is called every frame and is used to check if the character has reached its destination. If the character has reached its destination, the character is rotated to face the parent transform, and the LocalPosChange method is called to adjust the character's position.
- The RotateToParent method is used to rotate the character to face the parent transform, and the LocalPosChange method is used to adjust the character's position.

SpawnMoneyPos.cs

- The script is used to spawn and manage the money
- It has a variable called "spawnMoneyPosIndex" which is used to identify the specific location of the money spawn point in the game.
- It also has an "AnimationCurve" variable called "spawnTimeCurve" which determines the speed at which the money will spawn.
- The script has a bool variable called "canSpawn" which is used to check if the money can be spawned at the current moment or not.
- There's another bool variable called "isBarSpawner" which is not currently being used in the script.
- The script contains a function called "CountMoney" which counts the number of money stacks and updates the player preferences with the count.
- In the Start() function, the script finds the "MoneyStack" object and calls the "SpawnMoney" function with a delay of 1 second if the spawnMoneyPosIndex is not 0.
- If there is already money on the table, the script spawns new money and adds it to the stack in a random location. It also updates the player preferences with the new count.
- The script has a function called "CallGuests" which calls the "FindSeat" function for each guest in the scene (except the MC).
- There's a public GameObject variable called "moneyPrefab" which is used as the base for creating new money.
- The script also has a public integer variable called "playersAtTable" which is used to determine the spawn time based on the value of the "spawnTimeCurve".

- The "SpawnMoney" function is used to spawn money on the table. It checks if "canSpawn" is true and if there are any players at the table before spawning money.
- The function then determines a random location to spawn the money and instantiates a new money object with the "moneyPrefab". It also updates the player preferences with the new count.
- The "SpawnMoney" function is recursively called with a delay based on the "spawnTimeCurve" value.
- Lastly, there's a "SpawnMoneyOnce" function which spawns money on the table once and takes an optional delay value.

MoneyStack.cs

- Controls a money stack in a game. It resets the collider every 0.05 seconds to prevent objects from getting stuck in the stack.
- When the player touches the stack, it activates the money destroyer. When the stack collides with a delivery object, it spawns money and sets the delivery object's moneyToSpawn variable to 0.

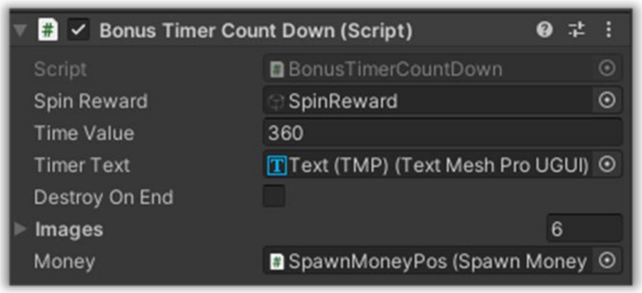
LockedCapsule.cs

- The script has a public integer variable "placeIndex" and several private variables, including a TextMeshProUGUI component for displaying the price of the locked capsule, an Image component for displaying the progress of unlocking the capsule, and a particle system for displaying a poof effect when the capsule is unlocked.
- In the Start() method, the script checks if the locked capsule has been purchased before by the player by checking the value of a PlayerPrefs integer variable called "PlaceX", where X is the value of placeIndex. If the value is 0, the script sets the price of the capsule to the default price of 50 and displays it on the screen using the TextMeshProUGUI and Image components. If the value is 1, the script destroys the animation component of the capsule, displays the unlocked capsule model, plays the poof particle effect, and performs other game logic related to unlocking the capsule.
- The script also has two trigger methods, OnTriggerEnter() and OnTriggerExit(), which are used to start and stop a coroutine called

CountDown(). When the player enters the trigger zone of the locked capsule, CountDown() is started, which decreases the price of the capsule over time and updates the TextMeshProUGUI and Image components accordingly. When the player exits the trigger zone, CountDown() is stopped.

- The script also has a method called CapsuleBuy(), which is called when the price of the capsule reaches zero. This method performs various game logic related to unlocking the capsule, such as setting PlayerPrefs variables, playing the poof particle effect, and destroying the locked capsule model.

BonusTimerCountDown.cs



```
Bonus Timer Count Down (Script)
Script          BonusTimerCountDown
Spin Reward     SpinReward
Time Value      360
Timer Text      Text (TMP) (Text Mesh Pro UGUI)
Destroy On End  
Images                                  6
Money           SpawnMoneyPos (Spawn Money
```

- Controls controls a timer countdown mechanism for a bonus feature in the game. The timer is displayed using a TextMeshProUGUI component and a fill image component.
- The script also handles interactions with a collider and trigger. When the timer reaches zero, the collider and trigger are enabled, and the "SPIN" text is displayed. If it's the first spin, a box collider is also enabled. If the bonus is destroyed at the end, a particle system is played and the parent game object is destroyed.
- If a player collides with the bonus object and it can't countdown anymore, and it's not destroyed at the end, and it's the first spin, the player is allowed to spin the wheel. The wheel is spun using AddTorque method, and after a set duration, a function is invoked to decide the bonus.
- The decide function selects the bonus by choosing the highest image position on the wheel and playing the corresponding animation. The switch statement handles the different bonuses that can be awarded, including spawning money, activating a scooter, and enabling a UI

element. The collider and trigger are disabled, the fill image is reset, and the timer starts again.
- Finally, the script also includes a function to add a reward spin, which is called externally when a reward is earned outside of the bonus feature.

ChipMan.cs

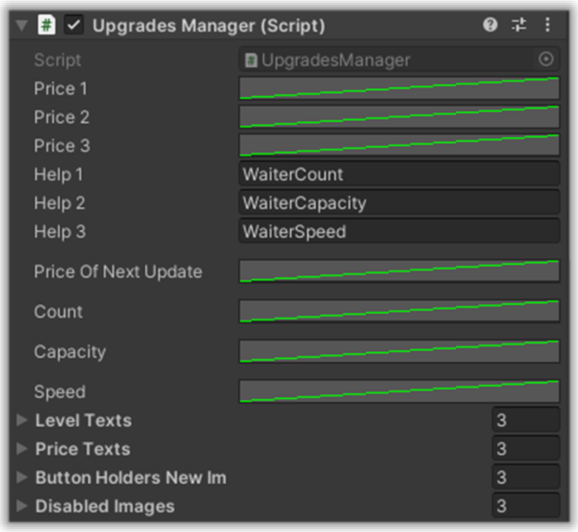
- Controls Chefs and Delivery Guys
- The script starts with declaring private variables for the ChipMan character including NavMeshAgent, Transform, and Animator. The public Transform chips variable is also declared.
- The CheckShelfs function checks the number of game objects with the tag "Chips" and sets PlayerPrefs to a certain value depending on how many chips are found. Then it calls the CheckWhatToEnable function from the EnableShelfs script.
- The OnEnable function checks the name of the game object that the script is attached to and invokes the CheckShelfs function if the name contains "CHEF". It also sets the capacity of the food holder for the chef if the ChefCapacity PlayerPrefs has a value. If the name contains "WAITER", it sets the speed and capacity of the NavMeshAgent if the PlayerPrefs ChipManSpeed and ChipManCapacity have values.
- The KeepResetingCollider function repeatedly disables and enables the Collider of the game object with a delay.
- The Start function sets the Collider, Animator, exit, and NavMeshAgent variables. It then waits for 0.05 seconds and sets the chips variable to a random game object with the tag "Chips" or to the first one if isRandom is false. If the name contains "WAITER", the KeepResetingCollider function is invoked and chips is set to a random game object with the tag "BoxesUp". The GetChips function is then called.
- The OnVespa function runs indefinitely while the ChipMan character is on a Vespa, setting its local position and rotation. The VespaBack function destroys the food, plays a Vespa animation, and invokes the GetOffVespa function after 3.4 seconds. The GetOffVespa function resets the position and rotation of the character, stops all coroutines, sets the "motor" and "motorOff" Animator parameters, unparents the character from the Vespa, and invokes the GoToPay function after 1.8 seconds. The GoToPay function then invokes the GetChips function.

- The OnTriggerEnter function checks for collisions with game objects named "ChipSpawner" or "PutDown". If it collides with a ChipSpawner, it sets the hasChips variable to true and invokes the RigOn_Chips function, then disables the Collider if the game object name contains "WAITER". If it collides with a PutDown object and the character has chips, it sets hasChips to false, checks if the character is within a certain distance from the object, then invokes the DropChipsAtTable or DropChipsAtDelivery function depending on the game object name. If it collides with a PutDown object and the character doesn't have chips, it returns.

Waiter_Mc.cs

- This script is attached to the Waiters and is responsible for controlling their behavior. The character can move around using the NavMeshAgent component and has the ability to choose a random table or food shelf to go to.
- The script keeps track of whether the character is currently going to get food or not and has several functions related to that. It also has some animation-related functions for handling the character's hands and an OnTriggerEnter function for detecting when the character has reached their destination.
- The script uses several other components like Collider, FoodHolderPlayers, and Animator to achieve its functionality. It also has some PlayerPrefs-related logic to retrieve player settings like speed and capacity.

UpgradesManager.cs



Upgrades Manager (Script)

Script	UpgradesManager
Price 1	
Price 2	
Price 3	
Help 1	WaiterCount
Help 2	WaiterCapacity
Help 3	WaiterSpeed
Price Of Next Update	
Count	
Capacity	
Speed	
Level Texts	3
Price Texts	3
Button Holders New Im	3
Disabled Images	3

- The script defines several variables, including price curves for upgrades, upgrade prices, and text objects that display the level and cost of upgrades.
- The script initializes values for some of these variables in the Start() method, including temporary values for count, capacity, and speed, as well as upgradable names based on the name of the object the script is attached to.
- The script also defines a method called UpdateTexts(), which is called by the script when it needs to update the text on the level and cost text objects. This method calculates the cost of upgrades based on the price curve, checks if the player has enough money to buy an upgrade, and updates the text objects accordingly.
- The script includes a method called GetTempPriceOfUpdate(), which returns the current cost of an upgrade based on its index in the upgrade array.
- The script also includes a method called Upgrade(), which is called when the player wants to buy an upgrade. This method subtracts the cost of the upgrade from the player's money, updates the player's level for the relevant upgrade, and plays an animation to indicate that the upgrade was successful. Finally, this method updates the tempcount, tempcapacity, or tempspeed variable for the relevant upgrade and adjusts other game objects based on the upgrade.

RandomReward.cs

- The script is responsible for spawning a random reward and animating it.
- The script uses an array of possible spawn positions and a set of game objects that will be disabled when the reward is finished.
- When the game object is first enabled, it waits for 2 seconds before calling the OfferReward() method. This method chooses a random position, enables the game objects to be disabled at the end, and calculates the amount of money to spawn.
- The amount of money spawned is based on an animation curve, which maps a value between 0 and 300 to a range of possible values. If the player's "Eat" value is over 300, a fixed value of 82 is used. The final amount of money spawned is then randomized slightly.
- The money is spawned by calling the SpawnMoneyOnce() method on a child game object multiple times. The reward is then animated by enabling a particle system.
- The reward is set to stop after 20 seconds, or when the DontDestroy() method is called. If the reward is not destroyed by DontDestroy(), it will be automatically restarted after a random delay between 25 and 55 seconds.
- Overall, this script allows for the creation of random rewards with various animations and spawn rates, and it can be customized easily by changing the values of the various parameters.

ArrowTutorialHolder.cs

- The script is responsible for showing the tutorials.
- The script has a public static instance variable and an Awake method that sets this instance to the current object. This allows other objects to access and modify properties of the ArrowTutorialHolder object.
- The script also has several public methods that are called from other objects. For example, EnableNextTutAfter() is a method that is called to enable the next tutorial after a specified amount of time. EnableNextTut() is a method that is called to enable the next tutorial immediately.
- The script has a private DynamicJoystick variable and several private variables used to keep track of the current tutorial index and to manage tutorial object activation.

- In the Start method, the script disables all tutorial objects and sets the current tutorial index based on the value stored in PlayerPrefs. It also renames previously shown tutorials to "VOOTMAR".
- When a tutorial is enabled, the script sets the ArrowController to look at the tutorial object, enables the tutorial object, and disables its collider or sets its trigger to false. It also enables a child object called CamCasino_Tut and invokes a method called CamOff after 2 seconds.
- The CamOff method disables the CamCasino_Tut child object, sets the tutorial object's collider to trigger mode, increments the tutorial index, and stores the new index in PlayerPrefs.

4. Tutorials for customizing the game