🎯 Core Gameplay Loop – *Hotdog Hustler*

Author: Game Design
Version: 1.0
Linked Systems: CartStateMachine, OrderManager, CustomerSystem, ScoreSystem, CurrencyManager
Scene Context: Single scene ("Hotdog Cart") with sub-states (Toasting → Grilling → Toppings → Serving)

🧩 1. High-Level Loop Overview

The core gameplay loop defines what the player repeatedly does to progress in the game.
In *Hotdog Hustler*, every loop represents **serving one or more customers** efficiently while managing time, accuracy, and resources.

Loop Summary

> *"Receive, Cook, Assemble, Serve, Earn, Upgrade — repeat."*

🧱 2. Step-by-Step Player Flow (Loop Cycle)

Step	Player Action	System Triggered	Feedback & Outcome
1. Receive Order	Customer arrives and requests a specific hotdog configuration (bun type, sausage, condiments).	`CustomerSpawner`, `OrderManager`	Order Ticket appears on HUD with timer + patience meter.
2. Toast Bun	Player interacts with the toaster station to toast buns. Timing impacts quality (undercooked, perfect, burnt).	`CartStateMachine` (Toasting State), `StationManager`	SFX: sizzle + visual cue; progress bar fills.

3. Grill Sausage	Player moves to the grill station and cooks the sausage. Must flip or remove before burning.	`CartStateMachine` (Grilling State)	Grill sizzle, smoke intensity increases; “Perfect!” popup on success.
4. Add Toppings	Player selects toppings from available condiments and ingredients (limited by unlocks).	`CartStateMachine` (Toppings State), `OrderValidator`	Visual topping layering; clicking SFX; topping icons added to ticket UI.
5. Serve Customer	Player confirms order submission to serve the customer.	`OrderValidator`, `CustomerBehavior`, `ScoreSystem`	Score calculated → Patience, Accuracy, Combo Bonus.
6. Receive Feedback & Tips	Customer reacts based on satisfaction (happy, neutral, angry). Earns tips accordingly.	`ScoreSystem`, `CurrencyManager`, `ComboManager`	Animated reaction + currency popup; Combo meter increments.
7. Prepare for Next Order	Next customer spawns. Player resets workflow or handles overlapping orders.	`CustomerSpawner`	Loop restarts automatically.

🔁 3. Meta-Loop (Progression Layer)

After several order loops (e.g., 10–15 customers per “day”), the meta loop triggers.

Phase	Action	System
End of Day Summary	Show performance metrics (customers served, accuracy, money earned).	`DayCycleManager`, `SummaryPanel`
Upgrade Shop	Player spends earnings to unlock new toppings, sauces, and cart improvements.	`ShopPanel`, `UpgradeManager`, `CurrencyManager`
Start Next Day	Difficulty scales — more customers, faster patience decay, more complex orders.	`DayCycleManager`, `CustomerSpawner`, `EventManager`

🧠 4. Player Motivation Loop

The design must emotionally engage players by rewarding **speed, mastery, and creativity**.

Motivation Type	Player Emotion	Reward Type	Design Method
Competence	“I’m getting faster and better.”	Score streaks, Combo bonuses.	Responsive feedback loops + visible tips counter.
Progression	“I’m upgrading my cart.”	Unlockable ingredients, visual upgrades.	Persistent economy + upgrade milestones.
Expression	“I like customizing my dog.”	Ingredient variety.	Optional toppings and cosmetic choices.
Flow	“I’m in the zone.”	Continuous action rhythm.	Balanced pacing, satisfying SFX + animations.

⚙️ 5. Gameplay Loop Logic Diagram (Simplified)

```
[Customer Arrives]
        ↓
[Order Ticket Appears]
        ↓
[Toast Bun] → [Grill Sausage] → [Add Toppings]
        ↓
[Serve Order]
        ↓
[Score + Tip + Combo]
        ↓
[Next Customer]
        ↓
[End of Day] → [Summary + Upgrades] → [Next Day Start]
```

🔄 6. Supporting Systems Overview

Each gameplay step corresponds to a system controlled by the developer’s architecture:

System	Purpose	Event / Trigger
CartStateMachine	Handles transitions between Toasting, Grilling, Toppings, Serving.	`OnStateChanged`
OrderManager	Creates orders and validates served hotdogs.	`OnOrderCreated`, `OnOrderCompleted`
CustomerSpawner	Controls arrival frequency and patience timers.	`OnCustomerSpawned`, `OnCustomerDeparted`
OrderValidator	Calculates accuracy and satisfaction.	Internal validation check.
ScoreSystem	Awards points/tips/combos.	`OnOrderCompleted`
CurrencyManager	Tracks earnings and upgrades.	`OnCurrencyChanged`
UpgradeManager	Unlocks new toppings and speed boosts.	`OnUpgradePurchased`
DayCycleManager	Defines session length, starts/ends the “day.”	`OnDayStarted`, `OnDayEnded`

🎮 7. Gameplay Balancing Variables (Design Notes)

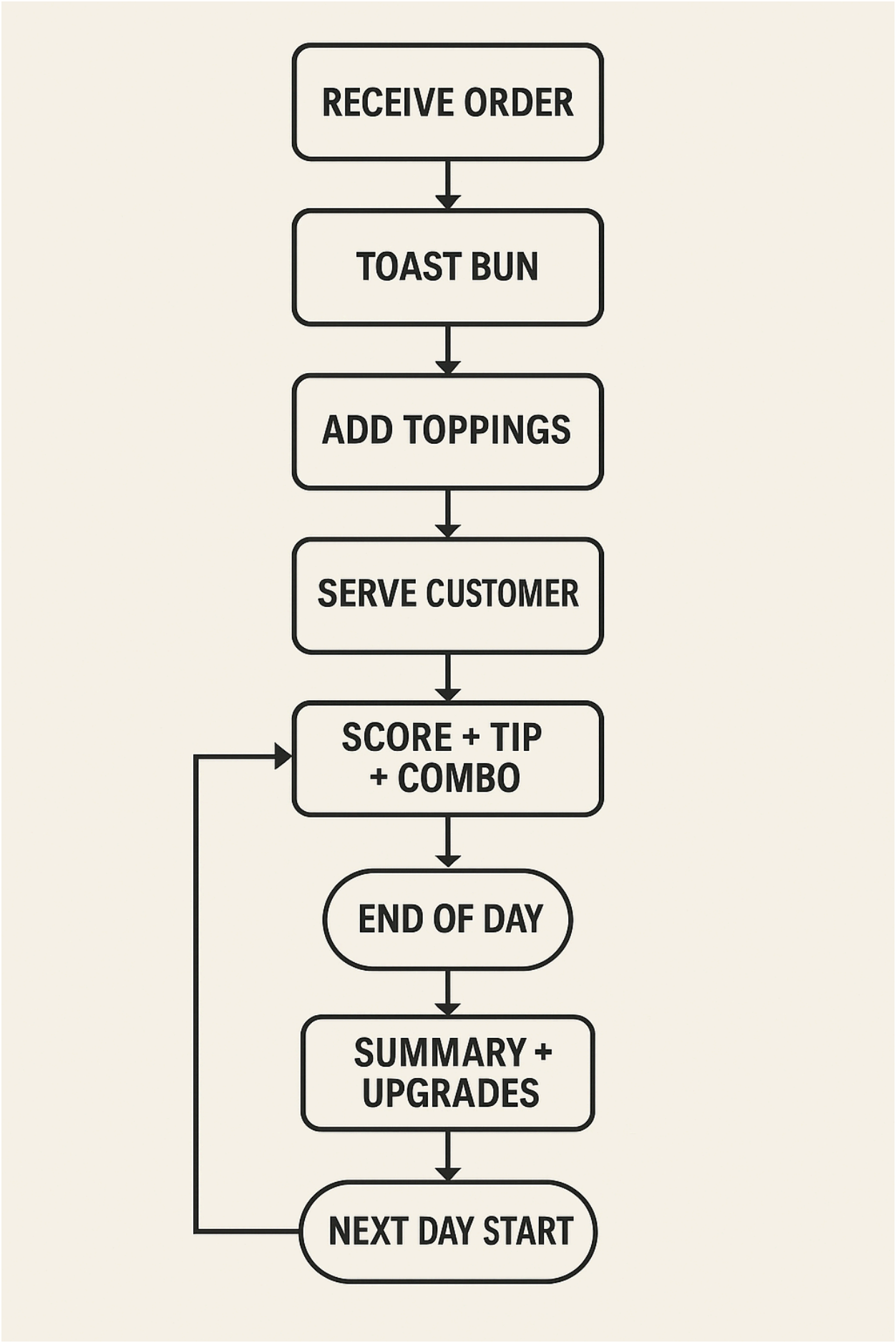
Variable	Range	Purpose	Design Rule
Customer patience time	20–45s	How long before leaving.	Decreases 10% per day.
Cooking window	±2s	Time to hit “Perfect” cook.	Reward accuracy with +10% tip.
Order complexity	1–6 toppings	Defines difficulty scaling.	+1 topping every 2 in-game days.
Combo timer	5s between perfects	Encourage pace mastery.	Resets on miss.
Daily goal	$200–$1000	Target to progress.	Scales 1.25× per day.

🧾 8. Deliverables for GD

- ✅ Core Gameplay Loop Diagram (Flow + Logic Mapping)
- ✅ Step-by-step loop write-up (player-facing & system-facing)
- ✅ Motivation & Meta-loop breakdown
- ✅ Event/System crosswalk table
- ✅ Balancing variable table (early tuning pass)

🧩 9. Hand-off Notes for DEV

- **CartStateMachine** will control station progression and state transitions (Toasting → Grilling → Toppings → Serving).
- **All timing and patience variables** should be exposed in `ScriptableObjects` for rapid tuning.
- **Order complexity scaling** must be data-driven, not hardcoded.
- **Customer behavior feedback** (emote sprites, SFX) ties to `OrderResult` accuracy (Perfect, OK, Fail).


RECEIVE ORDER
TOAST BUN
ADD TOPPINGS
SERVE CUSTOMER
SCORE + TIP + COMBO
END OF DAY
SUMMARY + UPGRADES
NEXT DAY START