INVENTORY AND SHOP SYSTEM

PRESENTED BY ACCARDO CHENG

THE PROBLEM SPACE: INVENTORY & SHOP IN GAMES

- Inventory and item shops are core systems in many games.
- They strongly influence:
  - Player progression
  - Game pacing
  - Player engagement
- Especially common in:
  - RPGs
  - Platformers
  - Adventure games

Inventory from The Elder Scrolls V: Skyrim(2011)

CHALLENGES IN EXISTING INVENTORY & SHOP DESIGNS

- Systems often become **overly complex**
- Common issues for **players**:
  - Multiple nested menus
  - Managing duplicate items
  - Tedious item selling/buying
- Common issues for **developers**:
  - Hard to add or edit items
  - Logic duplicated between inventory and shop
  - UI and data easily become inconsistent

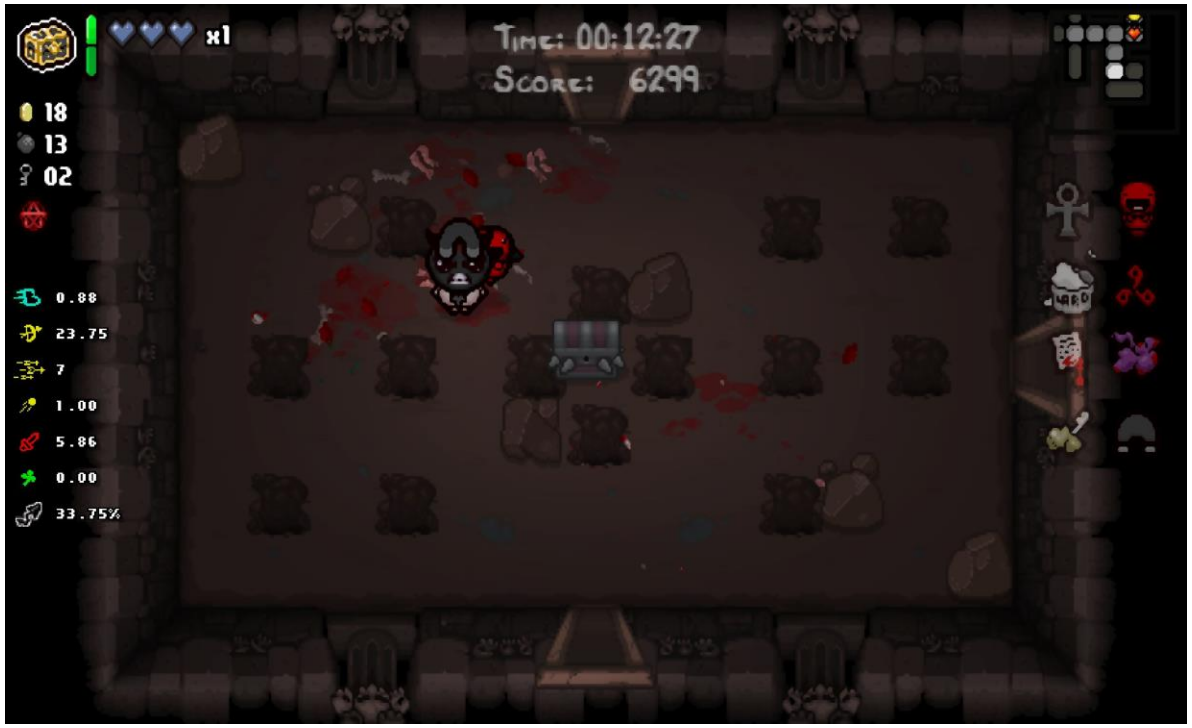
Inventory and Shop menu from Escape from Tarkov(2017)

WHY THIS COMPLEXITY MATTERS

- For players:
  - Slower gameplay
  - Reduced enjoyment
  - Friction during core actions
- For developers:
  - Increased maintenance cost
  - Higher chance of bugs
  - Difficult to scale or reuse systems

MOTIVATIONS & DESIGN GOALS

- Reduce unnecessary complexity
- Keep core interactions fast and intuitive
- Avoid duplicated logic between systems
- Make the system easy to extend and reuse



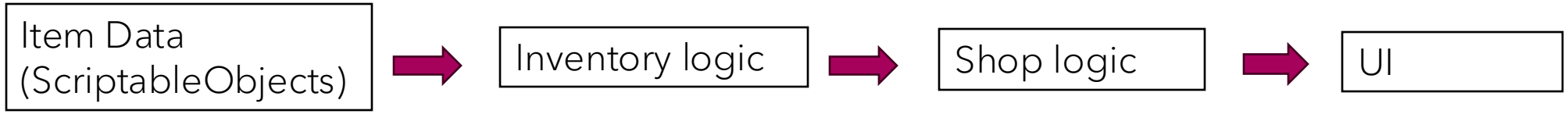
HUD inventory from The Binding of Isaac: Rebirth(2014)

MOTIVATED BY PLAYER EXPERIENCE AND DEVELOPER USABILITY

- For Players
  - Fewer menus and clicks
  - Clear feedback when actions succeed or fail
  - Faster selling and upgrading
- For Developers
  - Centralized item definitions
  - No duplicated inventory/shop logic
  - Easy to add new items or upgrades

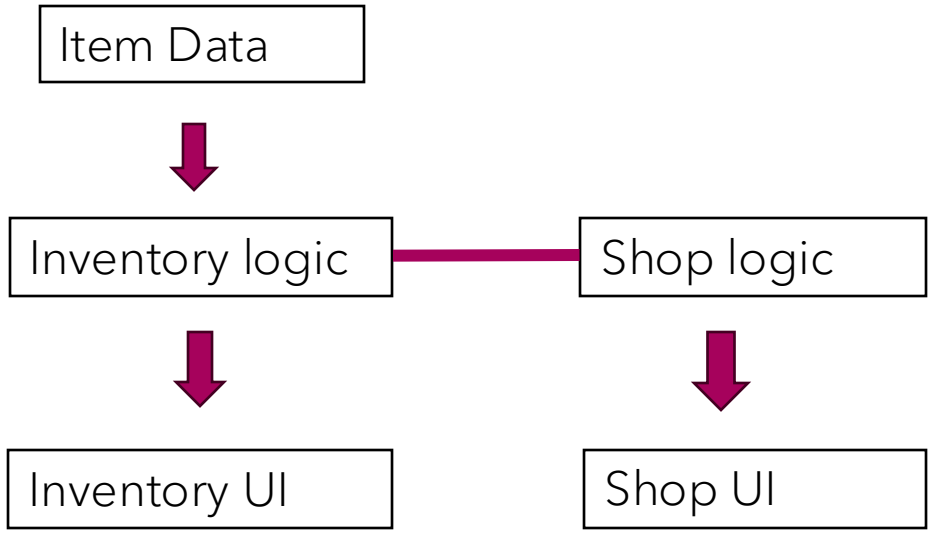
WHY A MODULAR, DATA-DRIVEN APPROACH

- Inventory and shop operate on the same data model
- UI reacts to state changes instead of controlling logic
- Item behavior defined through data, not hardcoded logic



SOLUTION OVERVIEW: INVENTORY & SHOP SYSTEM

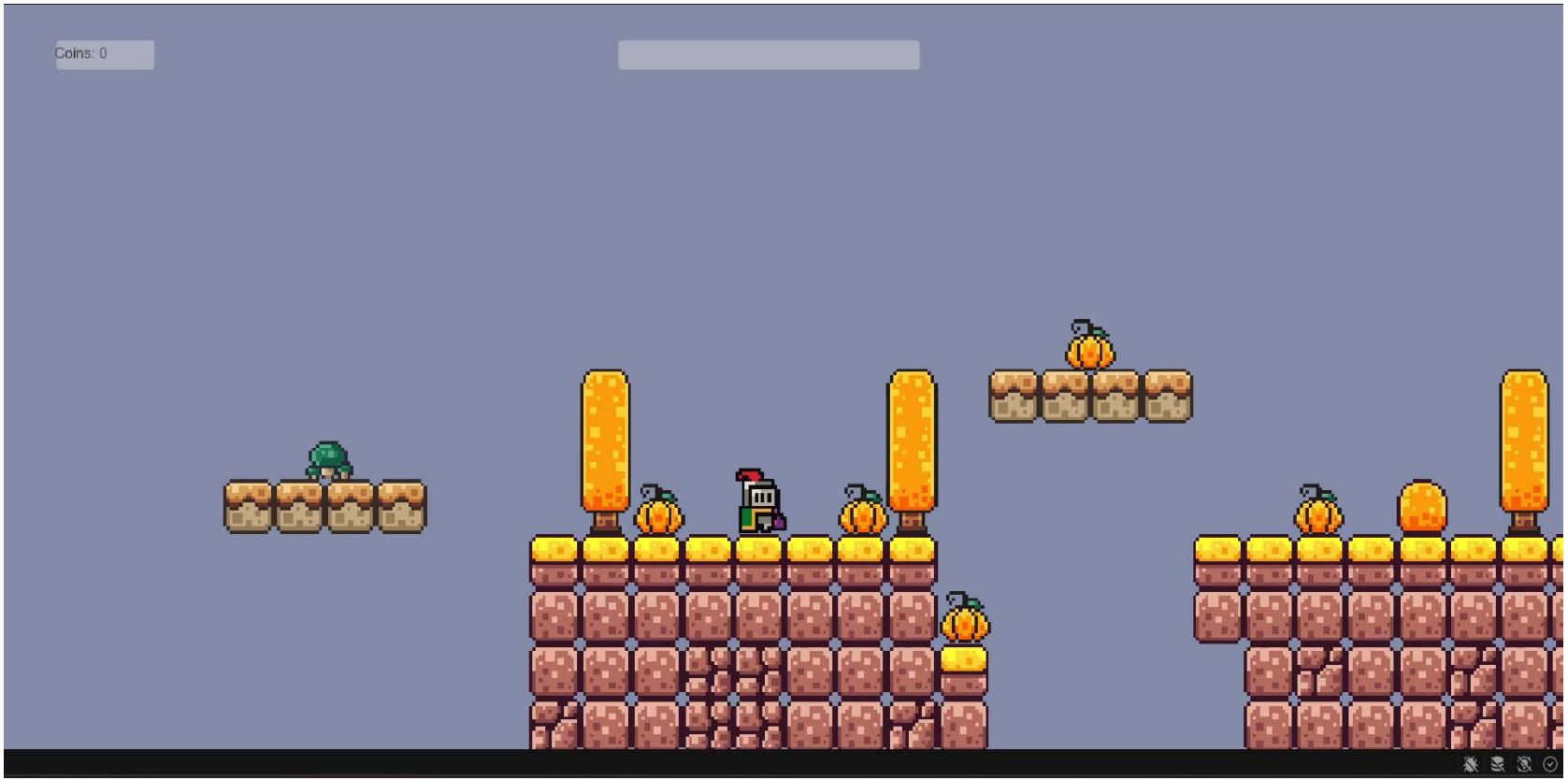
- A unified inventory and shop system
- Single source of truth for item data
- Inventory and shop operate on shared state
- UI reacts automatically to state changes



KEY DESIGN DECISIONS

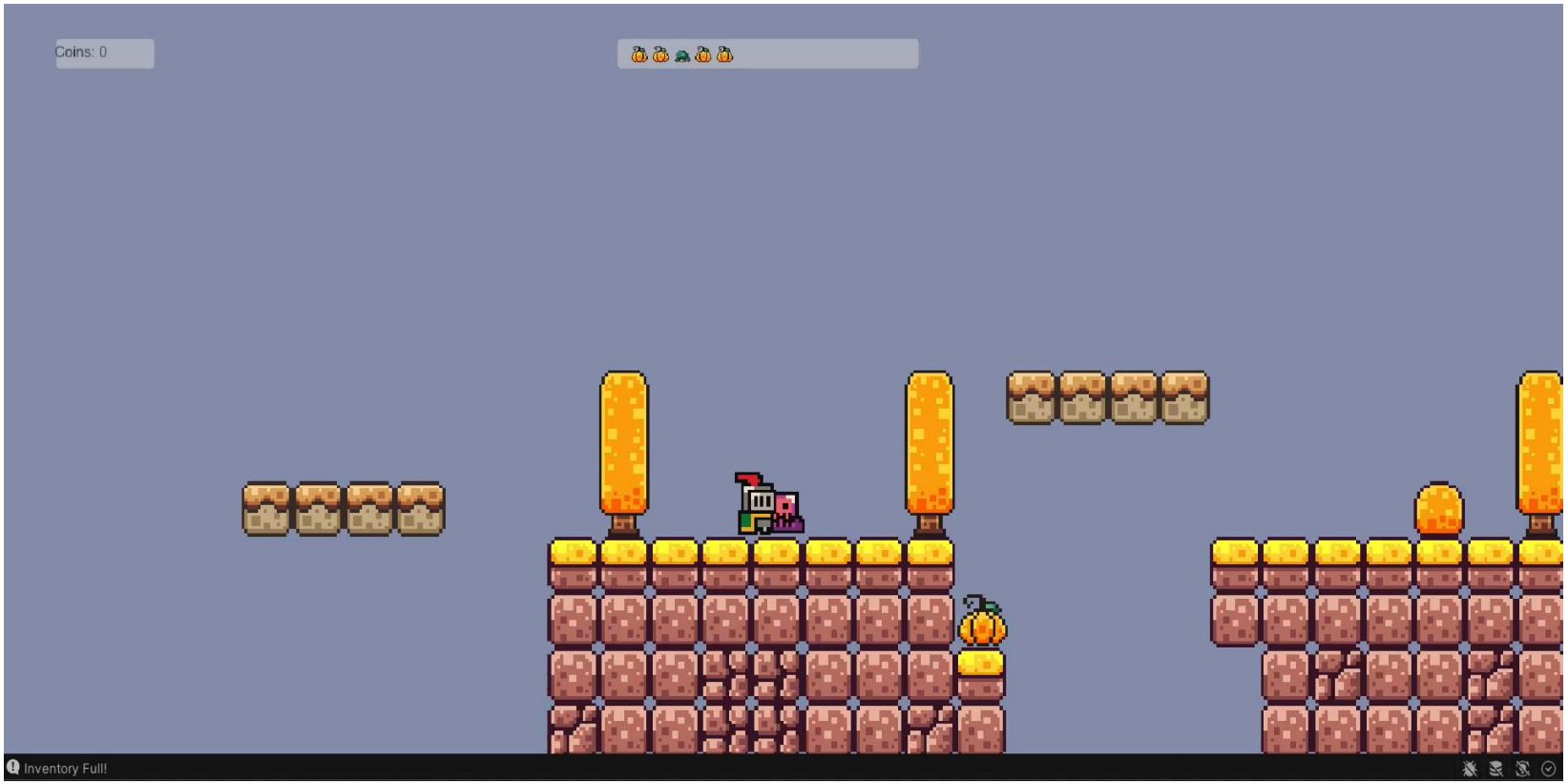
- Data-driven items and upgrades
- Centralized inventory logic
- Event-driven UI updates
- No duplicated logic between inventory and shop

DEMO: CORE INVENTORY FUNCTIONALITY



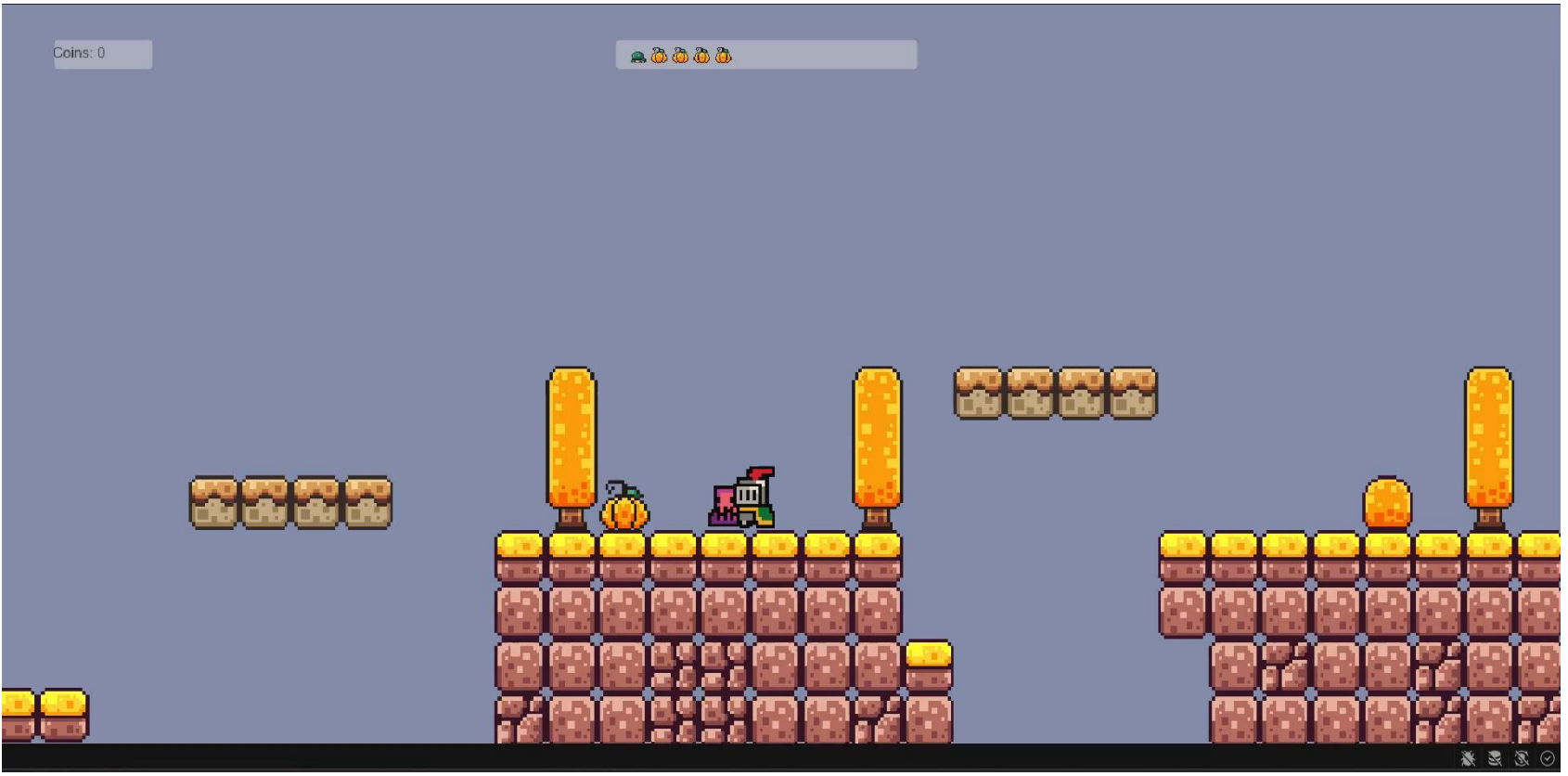
- Items are added to inventory
- UI updates automatically via callbacks
- Inventory enforces capacity constraints

INTERESTING CASES: DYNAMIC SHOP INTERACTIONS



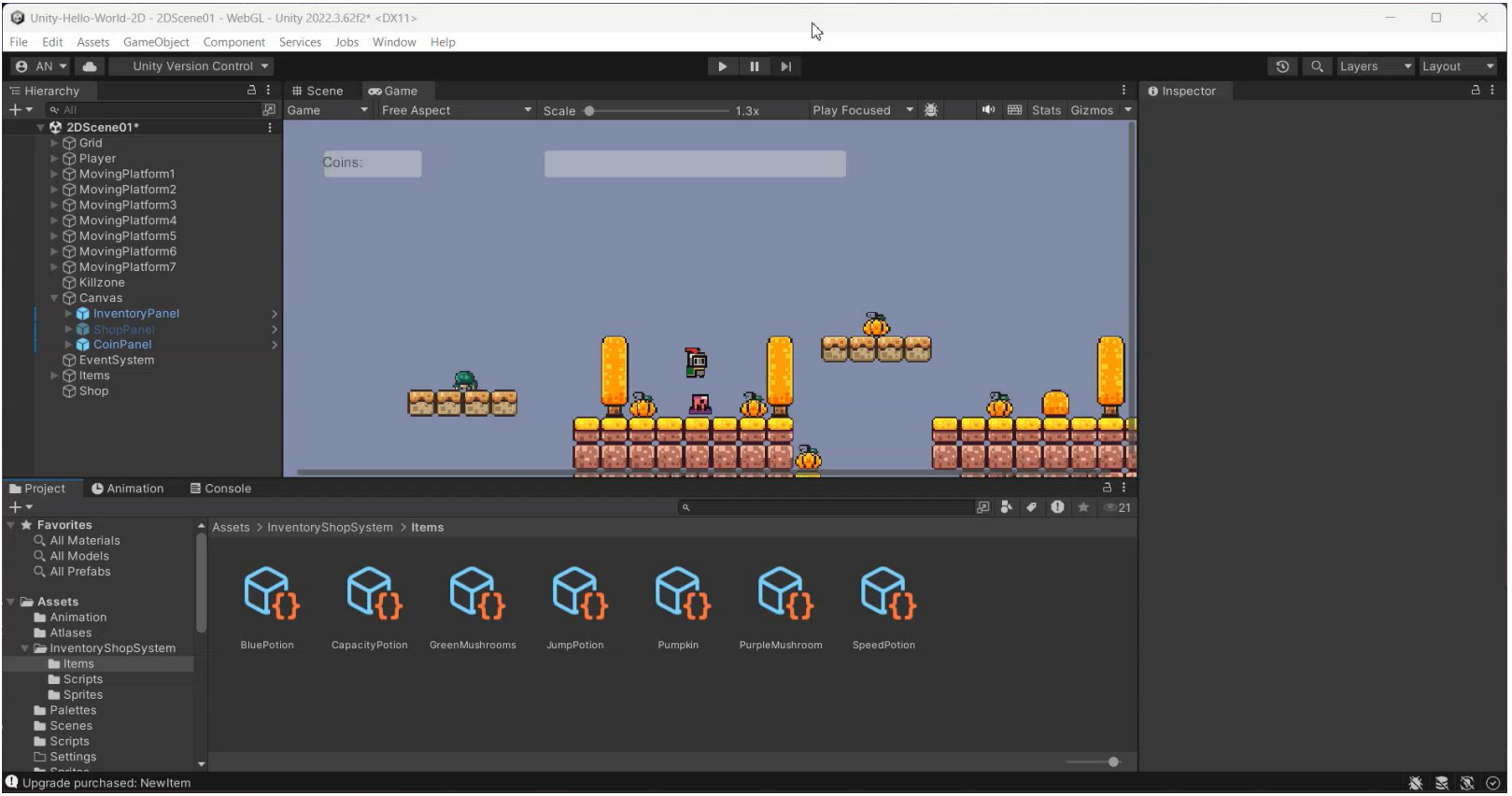
- Shop UI is generated from inventory state
- Items are grouped by type
- UI updates correctly when quantities change and purchase succeed
- Upgrade effects apply after purchase

FAILURE & EDGE CASES



- Failure case:
  - Capacity limits enforced
  - Invalid purchases are rejected
- Edge case:
  - Removal of the UI row when the item quantity reduces to 0.
- System remains in a valid state

DEMO: SYSTEM PARAMETERIZATION



- New items created via data, not code
- No changes to inventory or shop logic
- System is reusable and extensible

QUALITATIVE RESULTS

- Normal and Interesting Cases
  - Automatic UI synchronization
  - Dynamic item grouping in shop
  - Upgrades applied without modifying player logic
- Fail and Edge Cases
  - Inventory capacity reached
  - Insufficient currency for upgrades
  - Selling last item removes UI row

SYSTEM LIMITATIONS & CHALLENGES

- Limitations:
  - Inventory operations scale linearly with item count, only fit for small to medium size datasets
  - UI regeneration tied to inventory state changes
- Challenges:
  - Asset pipeline requires careful sprite organization
  - Upgrade effects logic is hardcoded and requires programming to expand on

DISCUSSION: FINDINGS AND FUTURE WORK

- Findings
  - Unified inventory and shop logic reduces complexity
  - Event-driven UI ensures consistent state representation
- Future Work
  - Optimized data structures for large inventories
  - UI virtualization for scalability
  - Extended upgrade and pricing logic

THANK YOU FOR WATCHING