Smart Beasts
Development Report

Smart Beasts is a **logic-based puzzle game** where players must create rules to guide their character to the target object.

The game takes place on a **10x10 grid-based map**, where players interact with various elements to solve puzzles. To progress, players must **fill in logic cards** that define how the character moves and reacts to obstacles.

Core Gameplay Mechanics:

- Players use a **drag-and-drop system** to place logic cards, which dictate movement and interactions.
- Characters move **based on predefined rules** (e.g., moving twice in one direction when selected).
- Upon colliding with objects, characters can **change direction** (up, down, left, or right) according to the game logic.
- Objects may disappear upon interaction, **creating new paths** for the player.
- Some objects can be **pushed by the player**, triggering **chain reactions** where multiple objects shift to reveal solutions.

Each puzzle challenges players to **think critically and experiment with different rule combinations** to achieve the objective. For example, in the scenario shown, the dog must **navigate obstacles and reach the bone** by carefully structuring its movement logic.

Smart Beasts offers an engaging way to develop problem-solving skills and logical thinking through interactive gameplay!

The game was developed using the following technologies:

- **Unity 2021.3.30 (LTS)** – main game engine.
- **C#** – scripting language for logic implementation.
- **Unity WebGL** – for web deployment.
- **Optimized PNG textures** (RGBA Crunched DXT5, BS3) – used to reduce build size while maintaining image quality.
- **Spine 2D Animation** – used for creating all character animations, ensuring smooth and dynamic motion
- **Asset Bundles with Amazon Web Services (AWS) & Web3** – Enables dynamic asset loading to reduce initial game size.
  AWS & Web3 are used for hosting and fetching resources efficiently, ensuring an optimized user experience.
- **External API Communication** – The simulation interacts with an external API written in TypeScript.
  Uses System.Runtime.InteropServices to send messages between C# (Unity) and JavaScript/TypeScript.
  Receives instructions from the API, allowing dynamic simulation behavior based on user-defined parameters.

The Main Interactive Features of Smart Beasts:

Smart Beasts is a logic-driven puzzle game where players define movement rules through a **drag-and-drop logic system** and execute them in a **real-time simulation**. The game emphasizes **strategic planning, interaction with dynamic objects, and cause-and-effect logic chains** to solve puzzles.

Key Interactive Features:

Rule-Based Execution & Real-Time Simulation

- Players create **logic sequences** using a **drag-and-drop card system** in the web interface.
- The game **executes these rules dynamically**, influencing how characters behave within the 10x10 grid.

Character Interaction & Object Reactions

- Players **click on characters** to activate their movements according to **predefined rules**.
- Characters can interact with **objects, walls, and obstacles**, which may **alter movement patterns** or trigger environmental changes.

Dynamic Puzzle Elements & Logical Triggers

- Some objects **disappear** upon interaction, **opening new paths**.
- Objects can be **pushed by characters**, causing **chain reactions** where multiple elements shift in sequence.

Conditional Logic & Strategy Development

- Players **test different rule combinations**, adjusting their strategies based on **real-time feedback**.
- The ability to **manipulate environmental elements** introduces multiple ways to **approach a single puzzle**.

Gradual Complexity & Progressive Challenge System

- Early levels introduce **basic movement and rule-building**.
- Advanced puzzles feature **complex logic chains, conditional actions, and multi-step interactions**.

Smart Beasts provides an engaging, hands-on way to develop problem-solving skills through logic-based interactions, player experimentation, and strategic thinking!

Challenges and Solutions

1. Handling a Large Number of Objects and Animations

Problem:
The game features **a large number of animated objects and assets**, which led to increased **build size and memory usage**, especially for **WebGL deployment**. Loading all assets at once would result in **longer load times and higher RAM consumption**, negatively impacting performance.

Solution:
To optimize asset management, we implemented:

- **Asset Bundles with Amazon Web Services (AWS) & Web3** – This allows for **dynamic asset loading**, reducing the **initial game size** while ensuring smooth performance.
- **AWS & Web3 for Hosting & Fetching Resources** – Assets are **retrieved on demand**, meaning only the necessary elements are loaded at a given moment, minimizing memory overhead.

This system significantly **improved performance, reduced loading times, and ensured scalability**, making the game more accessible for web-based play.

---

2. Iterative Animation Development & Collaboration with an Animator

Problem:
The game's animations play a crucial role in making interactions feel smooth and responsive. However, during development, we found that **certain animations did not align well with gameplay mechanics**, requiring multiple revisions. Ensuring that animations **reacted correctly**

to character movement, environmental interactions, and rule execution was a challenge.

Solution:

- **Collaboration with a Professional Animator** – The animations were created using **Spine 2D Animation**, and throughout development, we **continuously refined movement cycles and transitions** to match the game logic.
- **Iterative Testing & Refinement** – Animations were repeatedly tested and adjusted to ensure **seamless integration** with **player inputs and logic execution**.
- **Optimization of Animation Files** – Sprite-based animation sequences were **compressed and structured efficiently**, keeping the WebGL build lightweight.

Through **close collaboration and multiple refinements**, we achieved **smooth, visually appealing animations that seamlessly fit into the game's mechanics**.