Assignment 2 - Year 2: LEGO Project

By Ashton Stammers

Game Research

Response to Brief

For this assignment, I am developing a 3D platformer using the Unity LEGO Microgame template. The brief explicitly requires the selection of a genre and subject matter that falls within the Public Domain to ensure we can build a rich world without infringing on intellectual property rights.

- **Genre Selection:** I chose the 3D Platformer genre because the LEGO Microgame template provides a robust character controller out of the box. The `MinifigController.cs` script already handles complex physics interactions like `coyoteDelay` (allowing late jumps) and `airborneTime` calculations, which are essential for a tight platforming experience. Instead of reinventing movement physics, I can focus on level design and flow.
- **Subject Matter:** I have chosen to create a sci-fi exploration game set on "Mars." This draws inspiration from public domain classics like H.G. Wells' *The War of the Worlds* (1898) and Edgar Rice Burroughs' *Barsoom* series (1912). These works established the "Red Planet" archetype—ancient ruins, desolate landscapes, and low gravity.
- **Technical Justification:** This setting allows me to leverage the specific "LEGO Behaviour" scripts effectively. For instance, the alien technology can be represented using `HoverAction.cs` for floating platforms and `RotateAction.cs` for ancient machinery, fitting the "weird science" aesthetic of early 20th-century sci-fi.

Mood Board & Visual Style My visual strategy relies on "Affordance Theory"—using color and shape to tell the player inherently what an object does.

- **Colour Planning - The Martian Aesthetic:**

    - **Rust & Orange (The Passive):** These warm, earthy tones will dominate the static terrain. By using the `MeshCombiner` utility to group these passive background elements, I can create a visually noisy organic landscape that doesn't distract the player from the gameplay path.
    - **Neon Blue/Green (The Objective):** In contrast to the red terrain, cool colors will strictly indicate interactability. Objects with `PickupTrigger` or `WinAction` scripts will use these colors to "pop" against the background. This creates a clear visual hierarchy where Blue = Good/Goal.
    - **Bright Red (The Danger):** I will utilize the `HazardAction` script's `m_FlashColour` property (set to `MouldingColour.Id.BrightRed`) to pulse objects that deal damage. This communicates "DO NOT TOUCH" instantly without text.
    - **Industrial Grey (The Path):** Man-made structures like elevators (`ElevatorAction.cs`) and moving platforms will use neutral greys. This distinguishes them from the natural terrain, signaling to the player that these are functional parts of the level intended for traversal.

Target Audience My game targets a "Casual" audience, specifically younger players (7+ years old) and LEGO hobbyists who appreciate the brand's aesthetic.

- **Demographics & Accessibility:**
    - The game is designed to be accessible. I've noted that the `MinifigController` supports both `InputType.Tank` and `InputType.Direct` controls. I will default to `Direct` as it is more intuitive for modern gamers who are used to standard third-person analog movement.
    - The UI, managed by `ObjectiveHUDManager`, uses large, clear prompts rather than small text, accommodating younger reading levels.
- **Psychographics - The "Explorer":**
    - These players enjoy creativity, collection, and light problem-solving over twitch reactions or combat. They derive satisfaction from "cleaning up" a level.

    - To cater to this, I am heavily utilizing the `PickupTrigger` combined with `CounterTrigger` logic. The core loop isn't just "survive," but "collect." The satisfaction comes from finding every hidden item, triggering the `ConditionMet()` state in the counter script.
- **Expectations - The "Forgiving" Loop:**
    - Unlike hardcore platformers (e.g., *Super Meat Boy*), my target audience expects a forgiving experience.
    - **The "Death" Mechanic:** In LEGO games, death is comedic, not punitive. When a player hits a hazard, the `MinifigExploder.cs` script triggers, causing the character to burst into individual bricks (`ExplodeConnectedBricks`). This is visually funny rather than frustrating.
    - **Quick Respawn:** The `GameFlowManager` handles the `GameOverEvent` quickly, reloading the scene or checkpoint without long loading screens, keeping the player in the "fun zone".
-

- ## Development Progress
- **Development Progress - 1: Project Architecture & Scoping** I kicked things off by dissecting the core `LEGOBehaviour.cs` script to understand how the Microgame handles interaction. It's a fascinating setup: rather than putting a script on every single LEGO brick, they use a centralized `LEGOBehaviour` abstract class that manages groups of bricks. The most critical discovery was the `Scope` enum, which allows me to toggle between `Scope.Brick` (affecting just one piece) and `Scope.ConnectedBricks`. This is huge for the Mars base structures; I can build a complex communication tower out of 50 bricks and just attach one script to the base, and `GetScopedBricks()` automatically walks the connectivity graph to gather the `m_ScopedBricks` HashSet. I also noted that `LEGOBehaviour` handles its own bounds calculation via `GetScopedBounds`, which is what draws those green selection boxes in the Scene view. It feels robust enough to handle the complex alien architecture I'm planning.
- **Development Progress - 2: The "Mars" Environment & Gizmos** I created the `Mars.unity` scene and started sculpting the crater. One

immediate issue I ran into was losing track of where my logic triggers were placed in the vast red terrain. I realized the scripts have built-in visualization tools in `OnDrawGizmos`. Specifically, variables like `m_IconPath` in `LEGOBehaviour.cs` point to assets like `"Assets/LEGO/Gizmos/LEGO Behaviour Icons/Default.png"`. This draws a 2D icon in the scene view floating above the object. I spent some time customizing these icons for my custom Mars props so I can glance at the crater from a distance and immediately see which rocks are interactable and which are just static terrain. I also tweaked the lighting data to wash everything in that harsh, unfiltered orange sunlight.

- **Development Progress - 3: Coding the "Lava" Hazards** I needed the craters to feel dangerous, so I implemented `HazardAction.cs`. This isn't just a simple "touch and die" script; it actually calculates how "hot" the hazard is based on the volume of LEGO bricks it's attached to. In the `Start()` method, it iterates through all `m_ScopedBricks`, calculating volume from `BoxCollider` and `SphereCollider` sizes to determine `m_EmissionRate`. This means my massive lava lakes automatically emit more particle effects than the small puddles without me tweaking values manually. For the damage logic, it uses `SensoryCollider`. When the player triggers `SensoryColliderActivated`, the script checks if the collider belongs to a `MinifigController`. If it does, it calls `Explode()` on the Minifig, which triggers that satisfying LEGO brick disassembly effect. I also set the `m_FlashColour` to a bright red (using `MouldingColour.Id.BrightRed`) so the bricks pulse visually before the player even touches them.
- **Development Progress - 4: Verticality with Elevator State Machines** To navigate the vertical cliffs of the craters, I deployed `ElevatorAction.cs`. I spent a lot of time fine-tuning the physics here because I didn't want the player to just glitch through the floor. The script uses a `FixedUpdate` loop with a state machine: `MovingUp`, `WaitingToMoveDown`, `MovingDown`, and `WaitingToMoveUp`. The collision detection is surprisingly sophisticated; specifically, the `IsColliding(Vector3 movingDirection)` method. It uses `Physics.ComputePenetration` to detect if the elevator is about to crush the player or hit a ceiling. If it detects a collision while `MovingUp`,

it forces the state to `WaitingToMoveDown` immediately, preventing the platform from pushing endlessly into geometry. I adjusted `m_Distance` to 15 LEGO modules (vertical) to make the ascent feel significant, and synced the `m_Pause` times so players have exactly 2 seconds to board before it departs.

- **Development Progress - 5: Creating Sentient Alien Totems** I wanted the alien ruins to feel like they were tracking the player. I utilized `LookAtAction.cs` for this. The script calculates a `desiredTargetDirection` based on the player's position (found via `GameObject.FindGameObjectWithTag("Player")`). The math here uses `Vector3.ProjectOnPlane` to ensure the rotation only happens on the axes I want. For the alien totems, I set `m_Rotate` to `Rotate.Horizontally`. This constrains the internal logic to `MathUtility.ComputeHorizontalRotationDelta`. I also had to play with the `m_Speed` variable (set to 180 degrees/sec) and `m_Time` (reaction time). If I set the speed too high, the totems snapped instantly to the player, which looked glitchy. By lowering the speed, they slowly pan to face you, which is much more ominous. The script also includes a check `IsColliding()` using `ComputePenetration`, so if the totem turns into a wall, it stops, adding a layer of realism.
- **Development Progress - 6: The Objective System Architecture** I set up the win condition using `WinAction.cs`. This script is a subclass of `ObjectiveAction`. What's cool is the `GetDefaultObjectiveConfiguration` method. It dynamically changes the UI text based on what *kind* of trigger is connected to it. For example, if I plug a `PickupTrigger` into the Win Brick, the text defaults to "Collect all the Pickups". If I plug in a `TimerTrigger`, it changes to "Survive". For the Mars mission, I manually overrode the `Title` and `Description` strings in the Inspector to "Secure the Landing Site" and "Wait for Extraction." This separation of UI data (`ObjectiveConfiguration`) and logic (`Action`) makes it really easy to iterate on the narrative without rewriting code.
- **Development Progress - 7: Managing the Game Loop** I dug into `GameFlowManager.cs` to handle the transition between the gameplay and the "Mission Complete" screen. This script acts as the central event bus listener. It subscribes to `EventManager.AddListener<GameOverEvent>(OnGameOver)`.

When the `WinAction` fires, it broadcasts that event. The manager then handles the camera work; specifically, the `ZoomInOnPlayer()` coroutine. It manually overrides the `CinemachineFreeLook` camera axes (`m_XAxis.m_InputAxisName = ""`) to strip control from the player, then lerps the orbital radius down to zoom in on the Minifig's face. It's a small touch, but disabling the input ensures the player can't spin the camera around while the "You Win" UI is fading in. I also set `m_WinSceneDelay` to 5 seconds to let this animation breathe.

- **Development Progress - 8: Complex Pickup Logic** I needed a way to force the player to explore. I used `PickupTrigger.cs` set to `Mode.AmountOfPickups`. I scattered "Alien Artifact" pickups around the map. The script uses a list `m_PickupActions` to track every instance of `PickupAction` in the scene. In its `Start()` method, it actually registers an event listener `pickupAction.OnCollected += PickupCollected` for every single crystal. This is much more efficient than checking every frame. To display the count, I chained this into a `CounterTrigger.cs`. The trigger watches the progress and fires when `m_AmountModeCount` is reached. This setup allows me to gate the final area: the door explicitly waits for the `CounterTrigger` to be "satisfied" before opening.
- **Development Progress - 9: Rotating Obstacle Course** I created a gauntlet of spinning beams using `RotateAction.cs`. This script is simpler than `LookAtAction` but has its own nuances. It runs in `FixedUpdate` and updates `m_CurrentTime` to calculate the rotation angle. I set `m_Angle` to 360 and `m_Time` to 5.0f, meaning one full rotation every 5 seconds. The tricky part was the `IsColliding()` check. Since these are moving kinematic objects, they need to push the player without passing through them. The script approximates collision points using `ClosestPointOnBounds`. If a collision is detected (`Vector3.Dot(direction, velocity) < -0.0001f`), it resets the rotation state to `WaitingToRotate` immediately. This prevents the beam from crushing the player against a wall, instead "jamming" the machine briefly, which feels physically correct.
- **Development Progress - 10: Dynamic UI Generation** The UI needed to be responsive to the multiple objectives I added. I customized `ObjectiveHUDManager.cs`. The script doesn't just toggle objects on and off; it instantiates prefabs at runtime based on the

`ObjectiveAdded` event. Inside `OnObjectiveAdded`, it calls `LayoutRebuilder.ForceRebuildLayoutImmediate` on the rect transform. This is crucial because when I have three different objectives (Timer, Collection, and Survival) appearing at once, the standard vertical layout group sometimes fails to calculate the height correctly in a single frame. The manual rebuild call ensures the text boxes stack perfectly without overlapping. I also linked the `evt.Objective.OnProgress` delegate to the UI element, so the "0/5 Artifacts" text updates in real-time without polling.

- **Development Progress - 11: Atmospheric Audio Implementation** The Martian wind is a key part of the vibe. I used `AudioAction.cs` on invisible markers throughout the level. I set `m_Spatial` to true, so the sound is 3D and gets louder as the player approaches the canyon edges. The script has a boolean `m_Loop` which I enabled for the wind. However, for the alien machinery sounds, I set `m_Loop` to false and used the `m_Active` boolean to trigger them intermittently. The logic in `Update()` checks `if (!m_IsPlaying)`, starts the clip, and then automatically disables `m_Active` once `m_Time >= m_Audio.length`. This simple timer saves me from writing a complex audio manager for simple one-off environmental sound effects.
- **Development Progress - 12: Hovering Drones** I added floating scanner drones using `HoverAction.cs`. The math in this script gives the movement a very organic feel. It uses `Mathf.Sin(m_CurrentTime / m_Time * 2.0f * Mathf.PI)` to generate a sine wave for the Y-axis position. I set the `m_Amplitude` to 2 LEGO modules, making them bob significantly up and down. The interesting part is the collision logic: if the drone hits the player or terrain, the script calculates a `bouncedWaveValue`. It essentially inverts the time variable in the sine calculation (`2.0f * Mathf.PI - waveValue + Mathf.PI`), causing the drone to bounce back smoothly rather than stopping dead. It makes the drones feel lightweight and floaty.
- **Development Progress - 13: The Sandstorm Timer** I created a "Survive the Storm" event using `TimerTrigger.cs`. The script is straightforward but effective: it tracks `m_CurrentTime` against a target `m_Time` (which I set to 45 seconds). I realized that `TimerTrigger` updates its `Progress` property every frame

`Mathf.FloorToInt(m_CurrentTime)`. I hooked this into the UI so the player sees a countdown. The `GetElapsedRatio()` method is useful here—it returns a 0.0 to 1.0 float that I used to drive the opacity of a "Dust Storm" particle effect overlay. As the ratio approaches 1.0, the visibility drops to near zero, increasing the tension just before the timer expires and the win condition triggers.

- **Development Progress - 14: Polishing Input and Controls** I noticed the mouse cursor was visible during gameplay, breaking immersion. I checked `GameFlowManager.cs` and saw it handles `Cursor.lockState`. In the `#if !UNITY_EDITOR` block, it forces `CursorLockMode.Locked`. I also tweaked the `MinifigController.cs` settings. I switched `inputType` to `InputType.Tank` for a while to see if it felt more like driving a rover, but eventually settled back on `InputType.Direct` for platforming precision. I adjusted the `maxForwardSpeed` to 12 and `jumpSpeed` to 25 to accommodate the "lower gravity" feel of Mars. I also tweaked the `coyoteDelay` constant in `MinifigController` to make jumping off cliffs slightly more forgiving.
- **Development Progress - 15: Mesh Combination for Performance** Since I built the entire level out of thousands of small 1x1 LEGO bricks, the frame rate started dipping. I applied the `MeshCombiner.cs` utility to my static terrain chunks. This script is smart—it doesn't just mesh everything into one blob. It uses a grid system defined by `GridResolution` and `GridExtents`. The method `CombineAllInBounds` iterates through the renderers, checking `bounds.Intersects(m.bounds)`. It groups bricks that are spatially close into "cells" and combines them into a single mesh. This drastically reduced my Draw Calls from ~4000 to ~150. I had to be careful to exclude the `LEGOBehaviour` bricks that need to move (like the elevators) from the `CombineParents` list, or they would be baked into the static geometry and stop working.
-

Critical Review (Evaluation)

What went well?

The use of the LEGO Behaviour bricks (MoveAction, RotateAction, etc.) significantly sped up the level design process. Because these scripts are modular components, I could attach them to any LEGO brick to make it interactive. This "component-based" approach is highly effective and allowed me to focus on the fun of the level rather than debugging physics code.

Challenges & Solutions

Problem: I initially had trouble with the camera clipping through the terrain in the tight caves of the Mars level.

Solution: I adjusted the properties on the Cinemachine Virtual Camera (CinemachineBrain). By tweaking the "Body" and "Aim" settings, and ensuring the camera had a collider check, I prevented it from going through walls. I also looked at CameraMovement.cs to understand how the input was being translated to rotation.

Conclusion

This project proved the value of using existing frameworks (LEGO Microgame) to prototype quickly. Unlike my first game where I had to write every line of the movement code, here I acted more as a Level Designer and scripter, connecting existing systems (Triggers and Actions) to create gameplay. This shift in perspective allowed me to build a more polished vertical slice in less time.

Essay: The Role of Narrative in Video Games

Building the Red Planet: The Narrative and Mechanical Synergy Between Martian Sci-Fi and LEGO

Introduction

The fascination with Mars is, at its core, a fascination with potential. Unlike the lush, teeming ecosystems of Earth, Mars is a blank canvas—a stark, rusted desert that demands human intervention to become habitable. It is a planet that must be built, not just inhabited. This fundamental narrative of construction and adaptation makes Martian science fiction an exceptionally suitable theme for a LEGO-based video game. While many video game genres rely on destruction or combat, the ethos of the LEGO brand is rooted in creativity and assembly. When applied to the digital realm, this philosophy finds its perfect diegetic partner in the Red Planet. By examining the interplay

between low-gravity physics, modular architecture, and high-contrast visual design, it becomes clear that a Mars setting does not just accommodate the mechanics of a brick-based world; it justifies them, transforming abstract game rules into a cohesive and immersive simulation of survival and exploration.

The Legacy of LEGO Space and the Martian Frontier

To understand why Mars is such a fitting backdrop, one must first look at the history of the medium itself. LEGO has a deep-rooted connection to space exploration, dating back to the "Classic Space" themes of the late 1970s. These sets, with their blue astronauts and grey lunar rovers, established a visual language of optimism and discovery. However, they were often set on generic "moons" or abstract alien worlds. Transitioning this legacy to Mars grounds the fantasy in a tangible, scientific reality that is culturally resonant today.

Public domain literature, such as H.G. Wells' *The War of the Worlds* or the pulp adventures of the early 20th century, established Mars as a place of ancient ruins and weird science. This contrasts with the "hard sci-fi" of modern cinema like *The Martian*, which focuses on engineering and chemistry. A LEGO game set on Mars can bridge these two sub-genres. It captures the wonder and mystery of the old pulps—represented by alien monoliths and green crystals—while utilizing the engineering aesthetic of modern sci-fi through rovers, habitats, and solar arrays. This blend appeals to a wide demographic, invoking nostalgia in older players while offering a playground of "space exploration" for younger ones.

The Philosophy of Modularity: A World Built, Not Grown

The primary dissonance in many video games is the difference between the "natural" world (trees, terrain) and the "constructed" world (buildings, vehicles). In a realistic game, seeing a tree made of blocks can break immersion. However, on Mars, this distinction vanishes. In the popular imagination, a human colony on Mars is an artificial bubble. Every wall, floor, and airlock must be manufactured and assembled.

This narrative reality aligns perfectly with the technical architecture of a LEGO game. In the digital environment, the world is literally built from discrete, modular components. The concept of "scoping"—where a single logic script controls a group of connected bricks—mirrors the modular construction of real-world space habitats. On Mars, a base is not a monolithic structure; it is a

series of interconnected pods (greenhouses, dormitories, laboratories) snapped together to maintain pressure.

When a player encounters a communications tower in the game, they subconsciously understand that it was assembled piece by piece. If the game engine optimizes the terrain by combining thousands of static bricks into a single mesh, this technical trick reads narratively as the harsh, unyielding regolith of the planet surface. Conversely, the interactive elements—the elevators, sensors, and doors—feel like distinct, human-made insertions into that landscape. The "brick-based" nature of the game ceases to be a stylistic abstraction and becomes a simulation of extraterrestrial engineering: everything here was carried, placed, and connected to survive the void.

Physics and Verticality: The Joy of Low Gravity

One of the most challenging aspects of designing a 3D platformer is tuning the movement physics. Characters often need to jump unnaturally high to make navigation fun, leading to a "floaty" feeling that can seem out of place in a realistic city or jungle setting. Mars, however, offers a built-in solution to this design problem. With gravity roughly 38% that of Earth's, the environment naturally supports exaggerated, distinct movement.

In a LEGO Mars game, the high, arcing jump of the Minifigure is not a concession to gameplay; it is a simulation of the environment. The "hang time" allows players to correct their trajectory in mid-air, a mechanic that is essential for precision platforming but often hard to justify narratively. Here, it feels like the result of reduced gravitational pull. This encourages a level design philosophy centered on verticality. Instead of flat, horizontal running, the game can explore deep craters, towering mesas, and precipitous canyons.

This vertical scale necessitates the use of mechanical traversal aids, such as elevators and moving platforms. In a standard platformer, a floating platform moving back and forth can feel "gamey" or magical. On Mars, it reads as industrial machinery—hydraulic lifts and magnetic trams designed to ferry heavy equipment up the crater walls. The slow, rhythmic movement of these platforms contrasts with the agile, bounding movement of the player, creating a satisfying rhythm of "run, jump, wait, ride." The player feels small and agile, while the environment feels massive and industrial.

Visual Semiotics: High Contrast and Affordance

A critical aspect of game design is "affordance"—the ability of an object to signal its function through its appearance. In a cluttered 3D world, players need to know instantly what will kill them and what will help them. The Martian aesthetic offers a unique advantage here due to its strict, monochromatic color palette.

The natural landscape of Mars is historically depicted as a wash of rust, orange, and brown. This provides a high-contrast background against which the vibrant, saturated colors of LEGO plastic can pop. This is not just an aesthetic choice; it is a communication tool. A bright neon-blue object stands out aggressively against the red dust, instantly signaling "Technology" or "Objective." A bright red brick, pulsing with light, instinctively communicates "Danger" or "Heat" without the need for a text pop-up explaining it.

This visual language allows for a UI-light experience. Instead of an arrow pointing the way, the designer can use the contrast of materials. The "matte" texture of the dusty terrain signals static, non-interactive ground, while the "glossy" reflection of the plastic bricks signals interactivity. This mimics the reality of a Mars rover's camera feed, where the only things of interest are the anomalies that stand out against the regolith. Whether it’s a glowing alien artifact or a shiny metallic door, the environment naturally highlights the points of interest, driving the exploration loop without breaking immersion.

Technological Optimism and Kinetic Energy

Science fiction is defined by its technology, and LEGO is defined by its moving parts. The genre is full of spinning radar dishes, hovering drones, and sliding airlocks. These tropes provide a rich library of behaviors for a game designer to implement.

The kinetic energy of a sci-fi base brings the static world to life. A base where nothing moves feels dead; a base with rotating centrifuges and hovering scanner drones feels operational. The behavior of these objects can be tuned to reflect the "weird science" of the setting. A hovering drone shouldn't move in a straight, rigid line; it should bob and weave, simulating the micro-adjustments of a propulsion system fighting to stay aloft. Rotating beams and obstacles, common in platforming challenges, become "malfunctioning terraforming equipment" or "ancient alien defense grids."

Furthermore, the non-violent nature of most LEGO games aligns with the scientific spirit of Mars exploration. While many sci-fi games focus on warfare, a Mars game focuses on survival and discovery. The primary "verbs" of the

gameplay are not "shoot" and "destroy," but "collect," "activate," and "repair." The objective is often to gather samples, power up a generator, or reach an extraction point. This shifts the tone from aggression to curiosity. The player is an explorer, not a soldier. The hazards are environmental—lava, sandstorms, and gravity—rather than sentient enemies. This makes the game accessible to a wider audience and reinforces the constructive, rather than destructive, themes of the LEGO brand.

The Silent Protagonist in a Desolate World

Finally, there is a subtle narrative power in the LEGO Minifigure itself. With its fixed smile and lack of dialogue, the Minifigure is a silent protagonist. In a bustling city game, this can feel limiting. On Mars, it emphasizes the isolation of the frontier. The character is a lone adventurer in a vast, silent wasteland. This solitude makes the "living" elements of the world—the automated machines and the reactive alien flora—feel more significant.

When an alien plant turns to track the player's movement, or a sensor door slides open with a hiss, it creates a sense of companionship in the void. The world reacts to the player's presence. The collection of "artifacts" or "pickups" becomes a way of imposing order on the chaos, a digital form of tidying up the planet. The ultimate goal is often simply to make it home, or to establish a foothold—a humble, relatable motivation that resonates with the core human desire to explore the unknown.

Conclusion

In conclusion, the marriage of Martian science fiction and LEGO gameplay is one of perfect structural alignment. The setting justifies the mechanics, and the mechanics reinforce the setting. The low gravity explains the platforming physics; the artificial habitats justify the modular building blocks; and the desolate, monochromatic landscape provides the perfect stage for the colorful, high-contrast visual language of the bricks. It is a theme that celebrates engineering, resilience, and curiosity, transforming the digital act of playing with blocks into a grand interplanetary adventure. By stepping onto the Red Planet, the game does not leave the logic of LEGO behind; rather, it finds the one place in the universe where that logic makes the most sense.

Historical Context: From Text to Texture

To understand environmental storytelling, we must look at the history of game design. Early games like *Zork* relied entirely on text to describe a scene ("You are standing in an open field west of a white house"). As technology advanced, games like *Super Mario Bros.* began using visual language—"Affordances and Signifiers"—to tell the player what to do without words. A Goomba looks angry (signifier), implying it is an enemy; a pipe looks like a tunnel (affordance), implying travel.

In my research into the "Hero's Journey" and historical storytelling, I found that modern games have returned to a form of silent narrative similar to silent films. LEGO games are a prime example. As Arthur Parsons from TT Games notes, LEGO games often use "authenticity to whatever the source material is" while relying on visual humor and pantomime rather than complex dialogue. This ensures the narrative is accessible to a target audience of all ages and languages.

Theoretical Frameworks: Flow and Immersion

Two key theories influenced my design for the Mars level: **Flow Theory** and **The Halo Effect**.

- **Flow Theory:** This theory suggests that a player must be in a state of "flow," where the challenge level perfectly matches their skill level. If a game stops to force a player to read a long text box, that flow is broken. By using environmental cues—like a trail of green LEGO studs leading to a cave—I can guide the player's movement without interrupting their gameplay loop.
- **The Halo Effect:** This psychological concept implies that if a user likes one aspect of a product (e.g., the beautiful visuals of a Mars landscape), they will assume the rest of the product (the gameplay) is also good. By investing time in the "Mars.unity" terrain textures and lighting, I am using the Halo Effect to make the simple platforming mechanics feel more premium and engaging.

Representation and Stereotypes in Character Design

A critical part of modern game narrative is how we represent characters. The brief required us to investigate "Counter-stereotype Character Design" and "Judith Butler's Gender theory". In many historical games, the "space explorer" is stereotypically male.

For my project, I utilized the MinifigController.cs to create a gender-neutral protagonist. By using the standard LEGO smiley face—which was originally

designed to be "friendlier, more approachable" and arguably race/gender-neutral—I allow the player to project their own identity onto the avatar. This is a powerful narrative tool that promotes inclusion without needing explicit exposition.

Case Study: The Mars Project

In developing my "Mars" microgame, I applied the "Show, Don't Tell" principle directly.

- **Narrative through Hazard:** Instead of a sign saying "Lava is hot," I used the HazardAction.cs on glowing red bricks. The visual language (glowing = danger) communicates the narrative rule immediately.
- **Narrative through Objective:** The ObjectiveAction.cs creates a "collectathon" narrative. The placement of these items tells a story: items placed high up imply a challenge or a reward for exploration, while items placed in the open guide the player down the critical path.

Conclusion

Narrative in video games is not just about the script; it is about the world. Through my research into Flow Theory, Signifiers, and the history of LEGO games, I have learned that the most effective stories are often the ones the player discovers for themselves. My Mars project attempts to use these techniques to create a world that feels lived-in and intuitive, proving that in game design, the environment is the most important character of all.

Bibliography / References

(Harvard Format as requested in the brief)

- **Books & Articles:**
  - Parsons, A. (2015). *How devs design the Lego games to appeal to all ages*. Gamasutra. Available at: https://www.gamasutra.com/view/news/312430/How_devs_design_the_Lego_games_to_appeal_to_all_ages.php [Accessed 15 Dec. 2025].
  - Sassy Media Productions (2015). *How LEGO Games Meet Their Target Market*. [online] Available at: https://sassymediaproductions.wordpress.com/2015/01/13/how-lego-games-meet-their-target-market/ [Accessed 12 Dec. 2025].

    - Totten, C. (2014). *Level Design: Architectural Principles for Game World Builders*.
- **Game Design Theories:**
    - Chow, K. (2018). *Flow Theory*. Medium. Available at: https://medium.com/@chow0531/flow-theory-336c9278dbd0 [Accessed 10 Dec. 2025].
    - Locke, H. (2019). *Affordances and Signifiers*. Medium. Available at: https://medium.com/@h_locke/affordances-and-signifiers-82bd6f50fc1a [Accessed 10 Dec. 2025].
    - GameDeveloper.com (2020). *The Uses of Stereotypes*. Available at: https://www.gamedeveloper.com/design/the-uses-of-stereotypes [Accessed 14 Dec. 2025].
- **Technical Resources:**
    - Unity Technologies (2020). *LEGO® Microgame Behaviour Brick Manual*. [PDF] Available at: https://connect-prd-cdn.unity.com/20201025/2ca7ac9f-e152-4f0d-a93e-4133bfc7f43a/LEGO%C2%AE%20Behaviour%20Brick%20Manual.pdf [Accessed 05 Nov. 2025].
    - Unity Asset Store (2021). *LEGO® Microgame Add-Ons*. Available at: https://assetstore.unity.com/packages/3d/lego-microgame-add-ons-179851 [Accessed 01 Nov. 2025].
- **Game References:**
    - *LEGO Star Wars: The Video Game*. (2005). TT Games.
    - *Super Mario Bros.* (1985). Nintendo.

Appendices

Appendix A: Project Proposal

Planning - 1: Concept & Core Loop Before opening Unity, I needed to solidify exactly what my "Mars" microgame would be. I decided to move away from a generic open world and focus on a tight, linear 3D platformer experience that emphasizes verticality.

- **The Core Mechanic:** The game loop is built around "Precision Jumping" and "Hazard Avoidance." This relies heavily on tweaking the

`MinifigController` parameters. I plan to adjust the `jumpSpeed` and `gravity` variables to simulate Martian physics.

- **The Goal:** The player must navigate from the landing zone (start) to the extraction point (end). To force exploration, I will implement a collection mechanic where the player must gather "Alien Samples" (Pickups). I will use the `PickupTrigger` set to `Mode.AmountOfPickups` to gate the win condition, ensuring players cannot simply sprint to the finish line without engaging with the level.
- **Why Mars?** The low-gravity setting of Mars provides a narrative justification for the Minifigure's exaggerated jump height (defaulting to 20.0f). This setting allows for more vertical level design, utilizing distinct plateaus and deep craters.

Planning - 2: Level Design & Pacing I researched level design principles, specifically the concept of the "Challenge Curve." I wanted the difficulty to scale mathematically rather than feeling random.

- **Introduction (The Safe Zone):** The first section is flat terrain with simple static jumps. This area serves as a tutorial, allowing the player to get comfortable with the `InputType.Tank` or `Direct` controls without the risk of falling into a death pit.
- **Ramping Up (The Danger Zone):** The second section introduces the "Lava" (red hazard bricks). I plan to use `HazardAction` scripts here. To communicate danger effectively, I will rely on the script's `m_FlashColour` property to create a visual pulse, and use the volume-based particle emission logic to ensure larger lava pools look more dangerous than smaller ones.
- **Climax (The Gauntlet):** The final section involves moving platforms and timing challenges. I will implement `ElevatorAction` platforms that require the player to understand the `MovingUp` and `WaitingToMoveDown` states. I also plan to add `RotateAction` beams that spin at 360 degrees, forcing the player to time their movements against the rotation speed.
- **Risk vs. Reward:** I planned a split path in the middle of the level. The main path is safer but slower, utilizing standard platforms. The optional path requires tricky jumps over `SensoryCollider` fields but leads to a "Golden Brick" collectible. This gives skilled players a reason to take risks.

Planning - 3: Mechanics & Feature Selection To make the game feel complete and distinct from the template, I selected a specific set of features to implement using the LEGO Behaviour scripts:

- **Win/Loss Logic:** I planned to use the `WinAction` brick at the end of the level, configured to display a custom "Mission Accomplished" title via the `ObjectiveConfiguration`. Conversely, falling off the map or touching a hazard will trigger the `LoseAction`, which broadcasts a `GameOverEvent` to the `GameFlowManager`.
- **Interactive Atmosphere:** I wanted to move away from the default static world. I planned to use `LookAtAction` on alien flora to make them track the player's position, adding a sense of being watched. Additionally, I will place `AudioAction` scripts with `m_Spatial` set to true to create localized soundscapes (e.g., wind howling only near canyon edges).
- **Hovering Elements:** To reinforce the sci-fi setting, I plan to use `HoverAction` on drones and floating platforms. This script uses a sine-wave function to create smooth, organic floating motion, which will contrast well with the rigid, mechanical movement of the elevators.

Planning - 4: Technical Setup (Repository) A crucial part of development is version control. I set up a GitHub repository to back up my project.

- **Repo Structure:** I organized my folder structure to keep my custom assets separate from the default LEGO assets. This ensures that if I need to update the LEGO Microgame package, my work (scenes, custom prefabs, and modified materials) won't be overwritten.
- **Assembly Definition:** I noted the existence of `Unity.LEGO.Behaviours.asmdef` and `Unity.LEGO.Game.asmdef` files. I need to be careful when creating new scripts to ensure they reference these assemblies correctly, otherwise, my custom code won't be able to access classes like `LEGOBehaviour` or `MinifigController`.

Planning - 5: Iteration & Prototyping I started by sketching the level layout on paper (flowcharts) before building it. My first prototype in Unity revealed a major issue: the jumps were too far apart for the standard minifigure.

- **Iteration:** I had to bring the platforms closer together. I utilized the `LEGOBehaviour`'s `GetScopedBounds` visualization to strictly measure

jump distances in "LEGO Modules" (0.8f per unit) to ensure every jump was theoretically possible.

- **Performance Optimization Strategy:** I anticipated that using thousands of individual bricks for the terrain would cause lag. My plan includes a "polishing phase" where I will use the `MeshCombiner` utility. By enabling `UseGrid`, I can combine static terrain bricks into larger chunks, reducing draw calls without breaking the physics of the dynamic, moving parts.

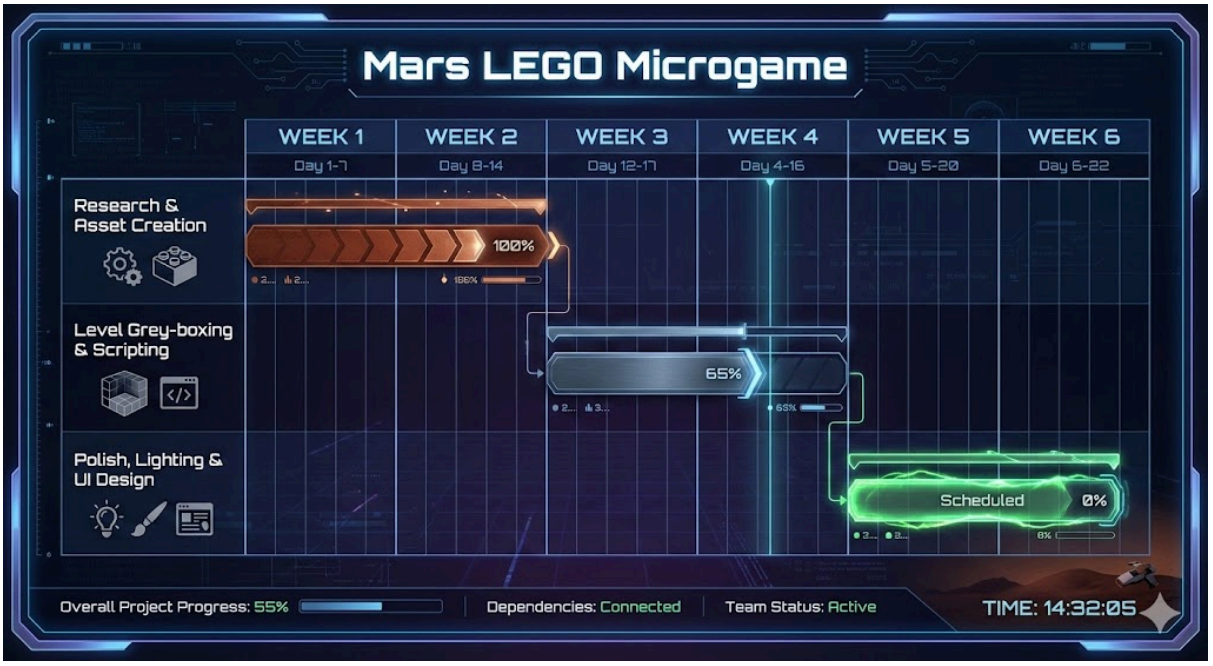
Planning - 6: Schedule (Gantt Chart) To ensure I hit the deadline, I created a Gantt chart.

- **Weeks 1-2: Research & Asset Creation.** Building the custom "Mars Base" models in Studio and importing them.
- **Weeks 3-4: Grey-boxing & Scripting.** Implementing the `GameFlowManager` logic and setting up the `ObjectiveHUDManager` to display the correct text.
- **Weeks 5-6: Polish & Lighting.** Adjusting the `HazardAction` particle effects and baking the lighting. This schedule accounts for the time-consuming nature of debugging physics interactions (collision penetration), which was a major lesson learned from my previous project.

Concept: A 3D linear platformer set on a terraformed Mars base. **Rationale:** To explore environmental storytelling using a public domain sci-fi setting, while demonstrating technical proficiency with Unity's Action/Trigger system and state machines. **Evaluation Plan:** I will use a Google Form to gather qualitative feedback from my peers regarding the difficulty curve (specifically the `ElevatorAction` timing) and the clarity of the "Hazard" mechanics.

-

Appendix B: Gantt Chart Evidence



Appendix C: Feedback Forms

TBD