Software Test Plan (STP)

Modular World Generator

---

1. Introduction

This Software Test Plan outlines the testing approach for the *Modular World Generator*, a Unity Editor Tool designed to streamline game scene generation for developers. The tool provides a user interface within the Unity Editor to procedurally generate urban and non-urban layouts, buildings, NPCs, vehicles, and props such as trees. It ensures automatic integration of NavMesh and colliders and provides basic movement logic to validate generated elements.

---

2. Test Items

- **Configuration Tab:**
  Testing will focus on generating scene layouts according to user-selected density and type (urban or non-urban). This includes correct placement of roads, parks, and designated building areas.

- **Buildings Tab:**
  Tests will ensure that buildings are:

  - Placed strictly within designated building areas
  - Oriented logically (e.g., facing roads in urban layouts)
  - Not overlapping with roads, parks, or other structures

- **NPCs Tab:**
  Tests will ensure that NPCs are:
  - Spawned only on walkable areas such as parks or building areas
  - Not placed on roads
  - Not spawned on top of existing GameObjects like buildings or other NPCs
  - Distributed logically without overlapping with one another
- **Vehicles Tab:**
  Tests will ensure that vehicles are:
  - Spawned on valid road segments
  - Oriented logically to match the direction of the road on which they are placed
  - Properly spaced to avoid overlapping with other vehicles or non-road objects
- Movement validation logic for NPCs and vehicles

---

3. Features to be Tested

- Layout generation based on density and urban type
- Building placement within defined areas
- NPC and vehicle instantiation and placement validation
- NavMesh generation for NPCs and vehicles
- UI elements: sliders, checkboxes, buttons, and tab interactions
- Basic movement logic of NPCs and vehicles to verify NavMesh backing
- Visual and spatial correctness of the scene generated in the Unity Editor

4. Features Not to be Tested

- Integration with runtime game logic or scene persistence
- Advanced vehicle or NPC AI behavior
- Support for user-generated prefabs that do not follow Unity's prefab conventions
- Support for non-Unity platforms or runtime builds

5. Testing Strategy

- **Unit Testing:**
  Covers individual functions and classes for layout logic, prefab placement, and internal configuration parsing.
- **Integration Testing:**
  Focuses on ensuring full integration with the Unity Editor environment. This includes validating that all UI components (buttons, sliders, checkboxes) trigger their intended functionality, prefabs instantiate correctly in the editor, and Unity-specific components like colliders and NavMesh surfaces behave as expected.
- **System Testing:**
  End-to-end validation of the full scene creation process within Unity. This includes simulating real developer workflows using the tool from start to finish and inspecting the final scene structure.
- **Acceptance Testing:**
  Simulates usage by a typical Unity developer unfamiliar with the internal code. The test evaluates whether the tool is intuitive, behaves as described in documentation, and produces usable, coherent scenes without manual corrections. This includes verifying that the generated scene can be immediately used for prototyping or extended for gameplay development.

---

6. Test Environment

- **Unity Editor Version:** 2023.2.20f1
- **Operating System:** Windows 11
- **Testing Tools:** Unity Test Framework, Unity console logs, visual scene validation, in-editor component inspection
- **Target Environment:** Unity Editor (Editor Mode only, not runtime builds)

---

7. Responsibilities

- **[Ben Lachovitz + Guy Zvilich]:** Unit testing of scene logic and building placement
- **[Ben Lachovitz]:** Integration testing of UI controls and Unity component bindings
- **[Ben Lachovitz + Guy Zvilich]:** System testing of end-to-end workflows and layout generation
- **[Guy Zvilich]:** Acceptance testing and usability evaluation from a game developer's perspective

---

8. Schedule

- **Unit Testing:**
  Will be conducted during the development of the logic and after core logic for each tab is implemented and stabilized for final check.

- **Integration Testing:**
  Will begin once all UI elements are connected to their underlying logic and the tool is functioning interactively within the Unity Editor.

- **System Testing:**
  Will take place after all major features are integrated, and the tool can generate a full scene from start to finish.

- **Acceptance Testing:**
  Will be performed prior to final delivery, after system testing is complete and the tool is considered feature-complete and stable.

---

9. Risks and Contingencies

- **Risk:** Unity version-specific changes might break compatibility
  Contingency: Lock development and testing to Unity 2023.2.20f1

- **Risk:** NavMesh generation may fail if the terrain lacks valid walkable surfaces
  Contingency: Add pre-bake validation and fallback options in the editor

- **Risk:** Unexpected prefab structures may lack required components (e.g., missing colliders)
  Contingency: Automatically assign default components or notify the user with an in-editor warning

- **Risk:** UI elements may not trigger logic due to misbinding or Unity serialization issues
  Contingency: Implement debug logging and enable error messages in the Unity Console during development