ISOMAKER

BETA TEST PLAN

Epitech - Class of 2026

HD-2D Isometric Game Engine

Alessandro Tosi
Aether Stoll-Metz
Tom Bariteau-Peter
Léa Guillemard

"A simple tool to create complex worlds."

Context

IsoMaker is a lightweight, specialized game engine designed to enable rapid and seamless creation of HD-2D isometric games. It aims to empower both developers and creative users by providing an intuitive map editor, a low-code scripting system, and a streamlined export pipeline, allowing them to move quickly from concept to a playable project.

At its core, IsoMaker offers tools tailored specifically for isometric game development:

- A 3D tile-based map editor using modular shapes and texture mapping
- A custom scripting system that allows easy definition of game logic
- A build and export pipeline that produces playable games on both Windows and Unix platforms

Technically, IsoMaker leverages Raylib (C++) for graphics, input, and audio management, alongside CMake for reliable cross-platform builds. This technology stack was carefully chosen to balance high performance with long-term maintainability.

Certain features were intentionally excluded during development—such as built-in low-poly 3D modeling tools and 2D sprite editors—as they were deemed redundant and outside the project's focused scope. This specialization ensures a clear vision and allows IsoMaker to stand out as an efficient tool, especially for indie developers and game designers who want to prototype and build without the complexity of large, general-purpose engines.

IsoMaker's design is inspired by recent industry successes that employ the isometric tile-map formula combined with the popular "HD-2D" visual style, exemplified by titles like *Tunic* (Isometricorp Games), *Hades* (Supergiant Games), and remakes such as *Dragon Quest 3 HD-2D* (Square Enix). IsoMaker naturally fits within this trend, offering a highly specialized, user-friendly toolset that enables anyone to create similar projects.

This beta release establishes a solid foundation for future growth, including planned features such as multiplayer support and genre-specific modules (e.g., turn-based tactics and cozy cooperative games). It already provides a full creative platform through its core gameplay systems, marking the first step toward building a focused ecosystem where both technical and non-technical users can unleash their creativity.

1. Core Functionalities for Beta Version

Feature Name	Description	Priority (High / Medium / Low)	Changes Since Tech3
Compile and Export Projects	Compile the modifications made by the user in the editor into an executable	HIGH	Export map and compile a new game project with a generic project
Import Projects	Read the data from a project exported by the software	HIGH	Import map
Rendering Engine	Encapsulated Raylib graphics library to render both 2D and 3D, UI and game graphics	HIGH	Abandoned SDL2 library for input and audio handling
Map Editor	Intuitively build a world map out of blocks and other simple shapes. Add decorative elements, NPCs	HIGH	The map itself is composed of a predefined selection of shapes to which we can apply textures rather than importing full 3D models
Scripting Engine	Write additional scripts to add custom features to the game	MEDIUM	Implement a scripting system that allows game logic to be modified using user-written scripts
Physics Engine	Simulate realistic physics, including collision detection, gravity, and object dynamics	LOW	Implement physics calculations for predefined shapes, applying forces, friction, and constraints

2. Beta Testing Scenarios

2.1 User Roles

Role Name	Description
Game designer	They love video games and want to create their own but don't know how to program. IsoMaker allows them to build levels, design game mechanics using visual tools, and customize assets without coding.
Game developer	A programmer who wants to create and customize game mechanics, assets, and features using IsoMaker. They can write scripts, optimize performance, and extend functionalities through code integration.

2.2 Test scenarios

Scenario 1: Import a project

Objective: Ensure that users can successfully import existing projects into IsoMaker without losing assets, settings, or configurations.
Test Steps:
1. Open IsoMaker and navigate to the project import section.
2. Select an external project file from IsoMaker or another compatible format.
3. Verify that all assets (sprites, models, scripts, textures) are correctly loaded.
4. Check that project settings, such as resolution, physics, and game logic, remain intact.
5. Attempt to run the imported project within IsoMaker and confirm it works as expected.

Expected Results:
- The project imports successfully without missing assets.
- No errors occur during the import.
- The project is editable and functional after import.

Scenario 2: Compile and Export Project

Objective: Ensure that projects can be compiled and exported into a playable format.
Test Steps:
1. Open an existing game project in IsoMaker.
2. Navigate to the export menu.
3. Run the compilation process and monitor for errors.

4. Launch the exported game and verify that it runs properly on the target platform.

Expected Results:

- The game compiles successfully without critical errors.
- The exported game maintains all assets, logic, and interactions.
- The exported file is playable.

Scenario 3: Rendering Engine

Objective: Validate the rendering performance and visual quality of the engine.

Test Steps:

1. Load a complex scene with multiple 2D and 3D assets.
2. Observe texture details, lighting, and shadow rendering.
3. Adjust rendering settings, like the resolution and the shader effects, and check their result.
4. Test the engine by adding more objects and animations.
5. Measure performance using FPS counters and resource usage.

Expected Results:

- Objects render correctly without artifacts.
- Adjusting settings dynamically affects rendering as expected in preview.
- The engine maintains stable performance even with high object counts.

Scenario 4: Map Editor

Objective: Test the usability and functionality of the map editor for creating game levels.

Test Steps:

1. Open the map editor and create a new map.
2. Place objects, structures, and entities within the map
3. Modify terrain, adjust object positions, and apply textures.
4. Save and reload the map to check if changes are applied.
5. Test gameplay within the edited map.

Expected Results:

- The editor allows smooth placement and modification of objects.
- Changes save correctly and reload without issues.
- The map integrates properly into gameplay without errors.

Scenario 5: Scripting Engine

Objective: Ensure that the scripting engine correctly executes game logic and custom behaviors.

Test Steps:

1. Open the scripting editor and write a simple game interaction, in that scenario, make a character move when a key is pressed.
2. Attach the script to an object in the scene.
3. Run the game and test if the script executes correctly.
4. Test error handling by introducing faulty scripts.

Expected Results:

- Scripts execute as expected without delays or errors.
- Error messages provide clear debugging information.

Scenario 6: Physics Engine

Objective: Ensure that the scripting engine correctly executes game logic and custom behaviors.

Test Steps:

1. Place objects with different physical properties (solid, dynamic, bouncy).
2. Test gravity by dropping objects and checking their movement.
3. Observe object collisions and interactions.
4. Adjust physics parameters (friction, mass, velocity) and analyze their effects.
5. Test the engine with many physics-enabled objects.

Expected Results:

- Objects behave according to real-world physics laws.
- Collisions are detected and processed correctly.
- The engine maintains performance even with multiple physics interactions.

Scenario 7: Hello World

Objective: Ensure we can create our own game with a character movable in a created map without errors.

Test Steps:

1. Open the map editor and create a new map.
2. Place objects, structures, and entities within the map.
3. Add a playable character with a chosen sprite.
4. Export the game project
5. Start the game executable
6. Test the game by moving the player in the map

Expected Results:

- Collisions are detected and processed correctly between the player and the map.
- Player movable: walking, jumping and attack with no error.

3. Success Criteria

Criterion	Description	Threshold for Success
Stability	IsoMaker must remain stable under real-world use and during high-load processes	No crashes or memory leaks detected during sessions >1h, including compiling projects with big map and custom scripts
Usability	New users (game designers or devs) should be able to prototype a basic game independently	80% of testers can create a playable map with a character, collisions, and export it within 45 minutes without external help
Performance	The engine must support real-time editing and gameplay testing with heavy asset use	Maintains 60 FPS with 200+ active entities, dynamic lights, and scripted behaviors on a standard laptop (Intel i5 + 8GB RAM).
Accuracy	Ensure the game engine correctly processes code, assets, and game logic, preventing compilation errors and file-related issues	No compilation or file errors reported during build and export processes, exported maps match in-editor layout exactly, with no data loss.
Robustness of Exports	Projects exported from IsoMaker must behave identically to their in-editor versions	The exported game looks and works exactly like it does in the editor, on both Windows and Unix
Extensibility	The scripting system must allow meaningful gameplay customization	Users can implement at least 3 unique game mechanics using scripts only

4. Known Issues & Limitations

Feature Name	Description	Changes
3D Modeling Tool	Create low-poly 3D models to be added into the project's library	Too ambitious Redundant Not competitive
2D Drawing Tool	Create pixel-art 2D models to be added into the project's library	Redundant Not competitive