12.1 Game Engine - Introduction

Game Engine

- Introduction to Game Engine
- Game Logic Screen Layout
- Game Materials
    - Game Settings
    - Material Physics
- Logic
    - Introduction
    - Sensors
    - Controllers
    - Actuators
    - Properties
    - States
- Camera
    - Camera
    - Camera Editing
    - Stereo Camera
    - Dome Camera
- Physics
    - Blender Game Physics
    - World Physics
    - Converting Game Engine Physics
    - Physics Types
- Performance
    - Introduction
    - System
    - Display
    - Introduction
- Python API
    - Introduction
    - Bullet physics Python API
    - The VideoTexture module: bge.texture
- Standalone Player
- Licensing of Blender Games
    - Standalone Games
    - More Information

Introduction to Game Engine

The Blender Game Engine (BGE) is Blender's tool for real time projects, from architectural visualizations and simulations to games.

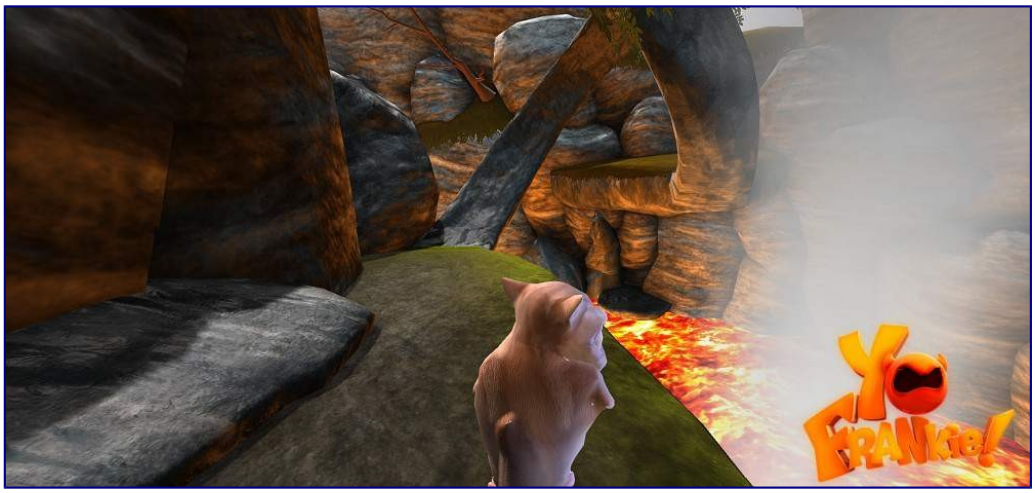
A word of warning, before you start any big or serious project with the Blender Game Engine, you should note that it is currently not very supported and that there are plans for its retargeting and refactoring that, in the very least, will break compatibility. For further information, you should get in touch with the developers via mailing list or IRC and read the development roadmap.

data-conversion –> input | animation, physics and logic | rendering

different renderer, logic is done via logic bricks or python press 'P' to play in preview mode in the embedded player

use cases and sample games

Blender has its own built in Game Engine that allows you to create interactive 3D applications or simulations. The major difference between Game Engine and the conventional Blender system is in the rendering process. In the normal Blender engine, images and animations are built off-line - once rendered they cannot be modified. Conversely, the Blender Game Engine renders scenes continuously in real-time, and incorporates facilities for user interaction during the rendering process.



Screenshot from "Yo Frankie", produced with Blender Game Engine

The Blender Game Engine oversees a game loop, which processes logic, sound, physics and rendering simulations in sequential order. The engine is written in C++.

By default, the user has access to a powerful, high level, Event Driven *Logic Editor* which is comprised of a series of specialized components called "Logic Bricks". The *Logic Editor* provides deep interaction with the simulation, and its functionality can be extended through Python scripting. It is designed to abstract the complex engine features into a simple user interface, which does not require experience with Programming. An overview of the *Logic Editor* can be found in the *Game Logic Screen Layout*

The Game Engine is closely integrated with the existing code base of Blender, which permits quick transitions between the traditional modeling feature set and game-specific functionality provided by the program. In this

sense, the Game Engine can be efficiently used in all areas of game design, from prototyping to final release.

The Game Engine can simulate content within Blender, however it also includes the ability to export a binary run-time to Linux, MacOSX and MS-Windows.

There are a number of powerful libraries the Game-Engine takes advantage of:

- Audaspace - a sound library for control of audio. Uses OpenAL or SDL
- Bullet - a physics engine featuring 3D collision detection, soft body dynamics, and rigid body dynamics
- Detour - a path-finding and spatial reasoning toolkit.
- Recast - a state of the art navigation mesh construction tool set for games.

When creating a game or simulation in the BGE, there are four essential steps:

- Create visual elements that can be rendered. This could be 3D models or images.
- Enable interaction within the scene using logic bricks to script custom behavior and determine how it is invoked (using the appropriate “sensors” such as keyboards or joysticks).
- Create one (or more) camera to give a frustrum from which to render the scene, and modify the parameters to support the environment in which the game will be displayed, such as Stereo rendering.
- Launch the game, using the internal player or exporting a runtime to the appropriate platform.