Game Engines

Technical Game Development II

Professor Charles Rich
Computer Science Department
rich@wpi.edu

Pedagogical Goal

- Your technical skills should not be tied to any particular game engine
- Just like your programming skills should not be tied to any particular programming language
- Use the best tools for each job
- ... or the tools you were given 😊
Definition

Game Engine
A series of modules and interfaces that allows a development team to focus on product game-play content, rather than technical content. [Julian Gold, OO Game Dev.]

- But this class is about “the technical content”! 😊

Buy versus Build

- Depends on your needs, resources and constraints
  - technical needs (e.g., “pushing the envelope”?)
  - financial resources (e.g., venture capital?)
  - time constraints (e.g., 1 mo. or 2 yr.?)
  - platform constraints (e.g., Flash?)
  - other factors (e.g., sequel?)
- Most games commonly built today with some sort of “engine layer”
Types of Engine Architectures (Roughly)

- **Monolithic** (e.g., Unreal Engine)
- **Modular** (e.g., C4 Engine)
- **Tool Kit** (e.g., jME)

Monolithic Engines (e.g., Unreal)

- “old style”—typically grew out of specific game
- tend to be genre-specific
- difficult to go beyond extensions/modifications not *anticipated* in (e.g., scripting) API
- proven, comprehensive capabilities
Modular Engines (e.g., C4)

- "modern"—often developed by game engine company
- use object-oriented techniques for greater modularity
- much easier to extend/replace components than monolithic engines
- architecture a bit more "bundled" (IDE-like) than tool-kit engines (see next)

Tool Kit Engines (e.g., jME)

- highly object-oriented
- designed for maximum modifiability
- typically open source
- may not be as complete or mature
Basic Game Engine Architecture Blocks

- ai
- animation
- camera
- controllers
- effects
- game
- geometry
- graphics
- gui
- input
- level builder
- modeler
- physics
- scene graph
- sound
- renderer
- operating system

Choices: “It’s a Jungle Out There”

- 290 3D engines reviewed at DevMaster.net

Most Reviewed Open Source Engines
1. Ogre
2. Irrlicht
3. Crystal Space
4. Panda3D
5. lib
6. Reality Factory
7. Render Game Engine
8. The Nebula Project
9. RealmForge
10. OpenSceneGraph

Most Reviewed Commercial Engines
1. Torque Game Engine
2. Tycoon SDK 6.5
3. 3DgameStudio
4. C4 Engine
5. Unity
6. Leadworks Engine 2
7. Neocore Engine
8. Max Studio
9. Visual3D.NET Game Engine
10. Esperant Creator

Latest Engines XNAL
- Bauerbraten
- ArxEngine
- MultiVerse
- AGENT
- 3D Real v6
- Elemental Engine II
- StormCell Game Engine
- Visual3D.NET Game Engine
- Leadworks Engine 2
- Reactor 3D Engine

- We are not going to try to review them all here
Many Evaluation Dimensions/Features

<table>
<thead>
<tr>
<th>General Info</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphics API</td>
<td>Windows</td>
</tr>
<tr>
<td>D3D</td>
<td>Linux</td>
</tr>
<tr>
<td>OpenGL</td>
<td>MacOS</td>
</tr>
<tr>
<td>GI</td>
<td>Linux</td>
</tr>
<tr>
<td>Software</td>
<td>Windows</td>
</tr>
<tr>
<td>Other</td>
<td>Windows</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operating Systems</th>
<th>Misc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Win32</td>
<td>Documentation</td>
</tr>
<tr>
<td>MacOS</td>
<td></td>
</tr>
<tr>
<td>FreeBSD</td>
<td></td>
</tr>
<tr>
<td>Linux</td>
<td></td>
</tr>
<tr>
<td>Solaris</td>
<td></td>
</tr>
<tr>
<td>WindowsCE</td>
<td></td>
</tr>
<tr>
<td>WindowsCE .NET</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Programming Language</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C++</td>
<td></td>
</tr>
<tr>
<td>Java</td>
<td></td>
</tr>
<tr>
<td>Python</td>
<td></td>
</tr>
<tr>
<td>C#</td>
<td></td>
</tr>
<tr>
<td>Smalltalk</td>
<td></td>
</tr>
<tr>
<td>Ruby</td>
<td></td>
</tr>
<tr>
<td>Perl</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Game Features</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Networking System</td>
<td></td>
</tr>
<tr>
<td>Unity/Netcode</td>
<td></td>
</tr>
<tr>
<td>Unreal Engine</td>
<td></td>
</tr>
<tr>
<td>Windows Server</td>
<td></td>
</tr>
<tr>
<td>Hammer/Server</td>
<td></td>
</tr>
<tr>
<td>Game Features</td>
<td></td>
</tr>
<tr>
<td>Networking System</td>
<td></td>
</tr>
<tr>
<td>Unity/Netcode</td>
<td></td>
</tr>
<tr>
<td>Unreal Engine</td>
<td></td>
</tr>
<tr>
<td>Windows Server</td>
<td></td>
</tr>
<tr>
<td>Hammer/Server</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tools &amp; Editors</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Modules</td>
<td></td>
</tr>
<tr>
<td>Add-ons</td>
<td></td>
</tr>
<tr>
<td>UI</td>
<td></td>
</tr>
<tr>
<td>Shaders</td>
<td></td>
</tr>
<tr>
<td>Textures</td>
<td></td>
</tr>
<tr>
<td>Rendering</td>
<td></td>
</tr>
<tr>
<td>Scene Management</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sound &amp; Video</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio</td>
<td></td>
</tr>
<tr>
<td>Video</td>
<td></td>
</tr>
<tr>
<td>Sound</td>
<td></td>
</tr>
<tr>
<td>Scene Management</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Special Features</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Lighting</td>
<td></td>
</tr>
<tr>
<td>Shadows</td>
<td></td>
</tr>
<tr>
<td>Texturing</td>
<td></td>
</tr>
<tr>
<td>Shaders</td>
<td></td>
</tr>
<tr>
<td>Rendering</td>
<td></td>
</tr>
<tr>
<td>Scene Management</td>
<td></td>
</tr>
</tbody>
</table>

If there's a feature term here you don't know, you should look it up!

Best Choice is Relative to Situation

- Similar issues of needs, resources and constraints (as in buy vs. build)
  - platform, programming language constraints
  - cost constraints (commercial run $ to $$$)
  - specific technical features required (e.g., MMO)
  - previous experience of staff
  - support from developers, user community (e.g., forums)
  - pedagogical goals (e.g., this course)
Choice of C4 and jME for IMGD 3000/4000

- C4 Engine
  - modular
  - C++ language (industry standard)
  - reasonable cost
  - technically sophisticated
  - good support community (forum)
  - http://www.terathon.com/c4engine

- jME (jMonkeyEngine)
  - tool kit
    - API to ODE and PhysX: https://jmepysics.dev.java.net
    - JGN for lightweight networking: http://code.google.com/p/jgn
    - FengGUI for HUD’s and other GUI’s: http://www.fenggui.org
  - Java language
    - “up and coming”, especially for mobile
    - ties in with Darkstar assignment
    - much less error-prone than C++
  - free, open source
  - technically sophisticated
  - good support community (forum)
  - http://www.ardor3d.com
C4 and jME Comparison

- Architecture
- Guided Tour of Tutorial Examples
- Feature comparison

<table>
<thead>
<tr>
<th>Base Services</th>
<th>System Managers</th>
<th>Large-Scale Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Manager</td>
<td>Graphics Manager</td>
<td>World Manager</td>
</tr>
<tr>
<td>- Allocating memory</td>
<td>- Renderable objects</td>
<td>- Hierarchy</td>
</tr>
<tr>
<td>- Organizing multiple heaps</td>
<td>- Light objects</td>
<td>- Geometries, lights, cameras</td>
</tr>
<tr>
<td>File Manager</td>
<td>- Camera objects</td>
<td>- Sound sources</td>
</tr>
<tr>
<td>- Reading and writing files</td>
<td>- Texture maps</td>
<td>- Zones and portals</td>
</tr>
<tr>
<td>- Building fbx files</td>
<td>- Shading attributes</td>
<td>- Triggers, markers</td>
</tr>
<tr>
<td>Resource Manager</td>
<td>Sound Manager</td>
<td>- Entities</td>
</tr>
<tr>
<td>- Resource loading</td>
<td>- Playing sounds</td>
<td></td>
</tr>
<tr>
<td>- Multiple resource catalogs</td>
<td>- Streaming music</td>
<td></td>
</tr>
<tr>
<td>Time Manager</td>
<td>Network Manager</td>
<td>Controller System</td>
</tr>
<tr>
<td>- Frame timing</td>
<td>- Low-level network access</td>
<td>- Using controllers</td>
</tr>
<tr>
<td>- Deferred events and timers</td>
<td>- Establishing connections</td>
<td>- Collision detection</td>
</tr>
<tr>
<td>Math Library</td>
<td>- Domain name resolution</td>
<td>- Characters and projectiles</td>
</tr>
<tr>
<td>- Vectors, matrices, quaternions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- RGB colors, complex numbers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Linear algebra routines</td>
<td>- System Utilities</td>
<td></td>
</tr>
<tr>
<td>- Polygon manipulation</td>
<td>- Multithreading</td>
<td>- Particle systems</td>
</tr>
<tr>
<td>Utility Library</td>
<td>- Persistent variables</td>
<td>- Surface markings</td>
</tr>
<tr>
<td>- Strings</td>
<td>- Console commands</td>
<td>- Fire effects</td>
</tr>
<tr>
<td></td>
<td>- Logging tools</td>
<td>- Fluid simulation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Cloth simulation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Interface Manager</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Creating and displaying windows</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- User interface elements</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Mouse and keyboard events</td>
</tr>
</tbody>
</table>
jMonkey Engine Architecture

Application: Game, Modeler, Level Builder, etc.

Graphics  Effects  Physics  AI
Input  Camera  Controllers  Animation
Geometry  GUI  Sound
Scene Graph  Renderer

Java Native Interface (JNI):  LWJGL  JOGL  ???

OpenGL / OpenGL ES

Operating System:  Windows  Linux  OSX

jMonkey Engine Architecture

Large-scale Architecture:
- World Manager
  - Node hierarchy
  - Nodes, objects, cameras
  - Lights, materials, nodes, camera
  - Scene and world
  - Nodes, hierarchy
  - Node hierarchy

Message Manager
- Parsing, serialization
- Sending and receiving messages
- Events and messages

Effect Manager
- Textures, materials
- Shadow maps
- Light sources
- Gaussian blur
- DOF simulation

Renderer
- Lighting and shading
- Camera and projection
- Scene and node hierarchy

Operating System:  Windows  Linux  OSX

System Utilities
- Multithreading
- Performance
- GUI
- Logging

Editor Framework
- Scene
- Scene Management
- Scene export
- Scene import
- Scene management

Input Manager
- Event handling
- Input handling
- Keyboard handling
- Mouse handling

View Manager
- Camera control
- Perspective
- Projection
- Camera handling

Resource Manager
- Resource management
- Loading and unloading
- Resource caching
- Node management

Video Library
- Video handling
- Video playback
- Video display
- Video effects

Manager Library
- Manager handling
- Manager creation
- Manager destruction
- Manager management

Memory Manager
- Memory allocation
- Memory deallocation
- Memory management

Math Library
- Math operations
- Math functions
- Math constants
- Math utilities

Utility Library
- Utility functions
- Utility tools
- Utility classes
- Utility methods

Edition:
- Development
- Production
- Beta
- Alpha

IMGD 4000 (D 09)
Guided Tour of Tutorial Examples

- Why are we doing this?
  - *not* to save you the trouble of reading the documentation! (You will need to anyways :-)
    - leaving out many details (e.g., error checking)
    - reordering for clarity (e.g., combining .h and .cpp files)
  - *not* interested in low-level C++ vs. Java coding differences
  - goal is to better understand the *design space* of engines by looking closely at different choices made
  - more generally, *thoughtful reading* of other people’s code is an important skill for software developers
    - paying close attention to modularity and architecture

```cpp
module C4::Application *ConstructApplication(void) // called by C4 engine
{ return (new Game); }

class Game : public Application {
private:
  EntityRegistration ballEntityReg; // for World Editor
  MovementAction *forwardAction; // typical input control

  Game() :
    ballEntityReg(kEntityBall, "model/Ball", kEntityPrecache, kControllerBall)
  {
    ballEntityReg.SetEntitySize(0.125F, 0.125F, 0.125F);
    ballEntityReg.SetEntityColor(ColorRGB(0.0F, 1.0F, 0.0F));
    TheWorldMgr->SetWorldConstructor(&ConstructWorld);
    // create and register movement actions
    forwardAction = new MovementAction(kActionForward, kSpectatorMoveForward);
    TheInputMgr->AddAction(forwardAction);
  }

  World *ConstructWorld(const char *name, void *data) // called by TheWorldMgr
  { return (new GameWorld(name)); }
```

---

```cpp```

```cpp```
class GameWorld : public World {
    private: SpectatorCamera spectatorCamera;

    GameWorld(const char *name) :
        World(name),
        spectatorCamera(2.0F, 1.0F, 0.3F) {} 

    WorldResult Preprocess(void) {
        Zone *zone = GetRootZone();
        const Marker *marker = zone->GetFirstMarker();
        while (marker) // find World Editor marker for camera placement
        {
            MarkerType type = marker->GetMarkerType();
            if (type == kMarkerLocator)
            {
                if (static_cast<const LocatorMarker *>(marker)->GetLocatorType() == kLocatorSpectator)
                {
                    spectatorCamera.SetNodePosition(marker->GetWorldPosition());
                    const Vector3D direction = marker->GetWorldTransform()[0];
                    float azimuth = Atan(direction.y, direction.x);
                    float altitude = Atan(direction.z, sqrt(...));
                    spectatorCamera.SetCameraAzimuth(azimuth);
                    spectatorCamera.SetCameraAltitude(altitude);
                }
            }
            marker = marker->ListElement<Marker>::Next();
        }
        SetCamera(&spectatorCamera); // set world's camera for rendering
        return (kWorldOkay);
    }
};

public abstract class AbstractGame { // in com.jme.app
    protected DisplaySystem display;

    public final void start() {
        initSystem();
        initGame();
        while (!finished && !display.isClosing()) {
            InputSystem.update();
            update();
            render();
        }
    }
}

public class SimpleGame extends AbstractGame {

    public static void main(String[] args) { // called by JVM
        new SimpleGame().start();
    }

    protected Camera camera;
    protected InputHandler input;
    protected LightState lightState;
    protected Node rootNode; // NB

    protected final void update() {
        timer.update(); // recalculate frame rate
        float tpf = timer.getTimePerFrame();
        Input.update(tpf); // check for key/mouse events
        rootNode.updateGeometricState(tpf, true);
    }

    protected final void render() {
        display.getRenderer().clearBuffers();
        display.getRenderer().draw(rootNode);
    }
}
protected final void initSystem() {
    display = DisplaySystem.getDisplaySystem(properties.getRenderer());
    display.createWindow(...);
    camera = display.getRenderer().createCamera(...);
    camera.setFrustumPerspective(...);
    camera.setFrame(...);
    camera.update();
    display.getRenderer().setCamera(camera);
    // setup input controls
    input = new FirstPersonHandler(camera);
}

protected final void initGame() {
    rootNode = new Node("rootNode");
    // create ZBuffer
    ZBufferState buf = display.getRenderer().createZBufferState();
    buf.setEnabled(true);
    buf.setFunction(ZBufferState.CF_LEQUAL);
    rootNode.setRenderState(buf);
    // set up basic default light
    PointLight light = new PointLight();
    light.setDiffuse(new ColorRGBA(1.0f, 1.0f, 1.0f, 1.0f));
    light.setAmbient(new ColorRGBA(0.5f, 0.5f, 0.5f, 1.0f));
    light.setLocation(new Vector3f(100, 100, 100));
    light.setEnabled(true);
    // attach light to a lightState and the lightState to rootNode
    lightState = display.getRenderer().createLightState();
    lightState.setEnabled(true);
    lightState.attach(light);
    rootNode.setRenderState(lightState);
    // attach example box to root node
    rootNode.attachChild(new Box("my box",new Vector3f(0,0,0),new Vector3f(1,1,1)));
    // update geometric and rendering information
    rootNode.updateGeometricState(0.0f, true);
    rootNode.updateRenderState();
}

Some Observations from Code Tour

- Code is overall more similar than different
  - systematic separation of node vs. state (to allow reuse of state descriptions)
    - C4: Light/LightObject, etc.
    - jME: Light/LightState, etc.
  - controllers associated with nodes for response to events
Some Observations from Code Tour

- Examples of how C4 more bundled, IDE-like:
  - C4 makes heavier use of singleton “managers”
    - C4 has single root node in WorldManager
    - any jME program can call updateGeometricState on any node
  - World editor more tightly integrated
    - “markers” installed in world editor and searched for by game initialization
    - level editor not bundled into jME (cf. MonkeyWorld3D)

Detailed Feature Comparisons

- From DevMaster.net

- Caveats:
  - Info may not be up-to-date (especially for jME)
  - I have added a few comments of my own
  - Let’s not get bogged down in the details---the idea is to get overall sense of emphasis
General Features

Object-Oriented Design, Plug-in Architecture, Save/Load System:
• Extremely clean class hierarchy for scene graph nodes, including geometries, cameras, lights, sounds, zones, portals, triggers, markers, and special effects
• General state serialization support for saving worlds
• Quick save and quick load capabilities
• Separation between per-instance and shared data
• External scene graph referencing from within another scene graph
• Support for pack files and a virtual directory hierarchy
• Skinable GUI’s

Modular OO based design with abstract interfaces for all low level APIs:
• 3D Text Generation
• Binding system for input controls
• Support for using jME in a Java Applet
• New Importer and Exporter System giving a standard framework for loading and saving jME scenegraphs
• A Binary Format implementation for the new import/export system that is more compact and faster than standard Java serialization
• Control Binding Management

Scripting

Graphical script editor

Efforts underway to add scripting extensions:
• Current JVM’s include JavaScript and LiveConnect (easy api between Java and JS)

[CR]
Built-in-Editors

- Full-featured integrated cross-platform world editor
- Interface panel editor
- Complete built-in windowing system
- Powerful and intuitive interface design
- Advanced surface attribute manipulation and material management

Level editor considered separate project:
- e.g., MonkeyWorld3D

Physics

Basic Physics, Collision Detection, Rigid Body:
- Built-in character controller.
- Built-in projectile controller.
- Real-time fluid surface simulation.
- Real-time cloth simulation.

Collision Detection:
- Triangle accurate collision detection

Physics considered separate project:
- e.g., jME Physics interface to ODE (Open Dynamics Engine), PhysX and others
Lighting

Per-vertex, Per-pixel, Lightmapping, Radiosity, Gloss maps, Anisotropic:
• Support for fully dynamic infinite, point, and spot lights
• Gloss-mapped specular reflections
• Ambient radiosity
  • Projected cube and spot textures
  • Cook-Torrance microfacet shading

Shadows

Shadow Mapping, Projected planar, Shadow Volume:
• All shadows are rendered in real time at global scale
• Three types of shadows are seamlessly combined in one world
• True penumbral soft shadows for area light sources

Shadow Volume:
• Z-Pass shadow volumes
Texturing

Basic, Multi-texturing, Bumpmapping, Mipmapping, Projected:
- Comprehensive bump mapping capabilities
- Enhanced parallax mapping
- Ambient occlusion channels
- Emission/glow maps
- Horizon mapping
- Realistic water shading

Basic, Multi-texturing, Mipmapping, Procedural:
- Support for simple texture based dot3 bump mapping

Shaders

Vertex, Pixel, High Level:
- Extensive support for vertex programs and pixel shaders

Vertex, Pixel, High Level:
- Support for OpenGL Vertex Programs.
- Support for OpenGL Fragment Programs
- Support for GLSL (cf. lecture on Fri, April 10)
Scene Management

General, Portals, Occlusion Culling, LOD:
- Efficient large-scale visibility determination
- Advanced inter-zone lighting analysis at runtime
- Special support for mirrors and remote portals
- Object instancing and external scene referencing
- Scene data can be imported from Collada format

General, Octrees, LOD:
- Scene graph based architecture
- Scene data can be imported from Collada format

Animation

Skeletal Animation, Animation Blending:
- Full skeletal hierarchy support for deformable meshes
- Powerful hierarchical animation blending system

Keyframe Animation, Skeletal Animation:
- A Skin and Animatable Bone System enabling realistic representation of models and motion
**Meshes**

Mesh Loading, Progressive:
- Support for the Collada scene format, enabling models to be imported from 3D Studio MAX, Maya, XSI, Blender, and other content creation packages

Mesh Loading, Skinning:
- Handles its internal format (.jme) and converts from/exports to ASE, 3DS, MD2, MD3, Milkshape, Obj and Collada
- Support for importing files in the COLLADA format
- New extension providing the ability to generate 3d meshes from text

**Special Effects**

Environment Mapping, Lens Flares, Billboard, Particle System, Motion Blur, Sky, Water, Fire, Decals, Fog, Mirror:
- Cube environment mapping
- Environment-mapped bump mapping
- Fully extensible particle systems
- Surface markings on arbitrary geometry
- Bump-mapped (fully lit) surface markings
- Real-time fire and electrical effects
- Transparent warping effects (heat haze, etc.)
- Bumpy reflection and refraction
- Postprocessed glow
- Fog volumes
- Full-scene cinematic motion blur
- Interactive in-game interface panels

Environment Mapping, Lens Flares, Billboard, Particle System, Sky, Water, Fire, Explosion, Fog:
- Cloth Simulation
- Water, with configurable reflection, refraction, wave generation and more
- Bloom, with configurable intensity, blurring, resolution and more-Dot3 Bumpmapping
Networking

Client-Server:
- Fast, reliable network implementation using UDP/IP
- Solid fault tolerance and hacker resistance
- Advanced security measures, including packet encryption
- Automatic message distribution to entity controllers

Networking viewed as separate project:
- e.g., see JavaGameNetworking, Darkstar

Sound and Video

2D Sound, 3D Sound, Streaming Sound:
- Fully spatialized 3D sound effects
- Unlimited streaming music channels with seamless looping and concatenation
- Doppler shift and other frequency effects
- High-precision sound travel delay
- Atmospheric absorption effects
- Reverberation with multiple simultaneous environments
- Directional sounds with cone attenuation
- Obstruction attenuation applied to direct and reflected paths
- Frequency-dependent volume settings for all effects
- Permeation system determines how far sounds travel through interiors
- Apple’s QuickTime technology can be used to play movies or soundtracks from numerous formats

3D Sound:
- OpenAL support with 3D position
Rendering

Fixed-function, Render-to-Texture:
• Antialiasing (up to 8x)
• Bilinear and trilinear filtration
• Anisotropic filtration (up to 16x)
• Vertical Sync control

Fixed-function, Render-to-Texture, Fonts, GUI:
• Rendering system supports both rendering to a screen context as well as rendering to a texture.
• Implements a Rendering Queue that automatically sorts opaque, transparent and screen objects and renders them in the correct order
• Multipass rendering system
• Supports rendering into a web-page via applets
• FBO support
• Support for rendering to Framebuffer Objects

Summary Ratings (5 star scale)

<table>
<thead>
<tr>
<th>Category</th>
<th>Rating</th>
<th>Votes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall</td>
<td>4.5</td>
<td>56</td>
</tr>
<tr>
<td>Features</td>
<td>4.5</td>
<td></td>
</tr>
<tr>
<td>Ease of Use</td>
<td>4.0</td>
<td></td>
</tr>
<tr>
<td>Stability</td>
<td>4.5</td>
<td></td>
</tr>
<tr>
<td>Support</td>
<td>4.5</td>
<td></td>
</tr>
</tbody>
</table>

Overall: 4.0 (30 votes)
Features: 4.0
Ease of Use: 4.0
Stability: 4.0
Support: 4.5