



Advanced Engineering Techniques for Large Web Player Games

Cliffhanger Productions



- Founded as Production Company, Development Studio since 2010
- Offices:
 - Vienna, Austria
 - Frankfurt, Germany
- >20 employees and freelancers



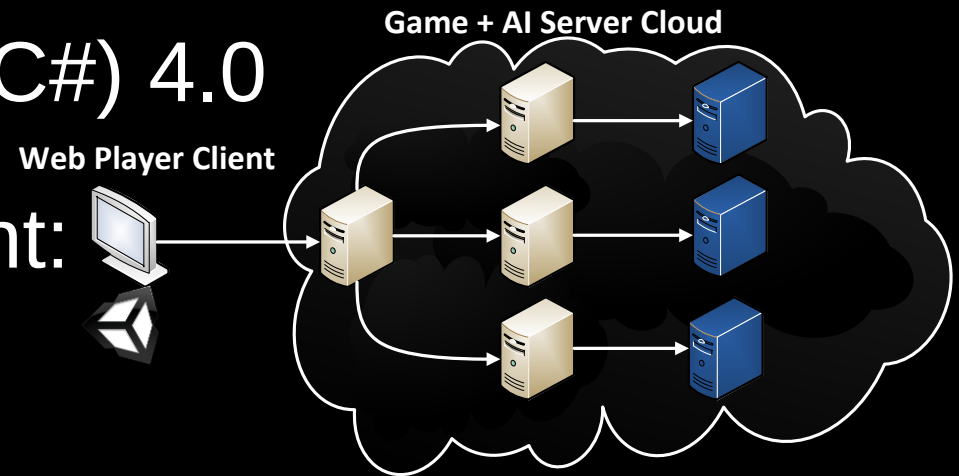
Jagged Alliance Online

- Turn based, tactical browser game
- State of the art 3D graphics
- > 100 unique missions



Development conditions

- Client developed for Unity 3.x Web Player
 - Has to look good on latest hardware
 - Must be playable on low-end laptops/netbooks
- Game server in .NET (C#) 4.0
- 14 months development:
 - 6 programmers
 - 6 designers
 - 3 artists



Challenges 1/2

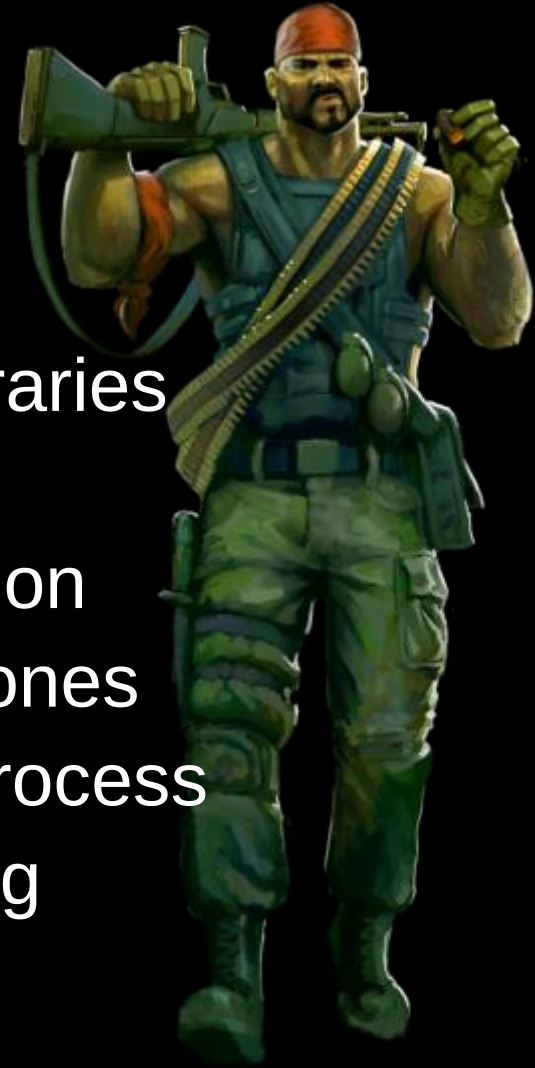
- Organize codebase:
 - About 100.000 lines of logic code, > 1.000 classes
 - Share code between Unity client and game server
 - Unit tests
- Manage huge amounts of content:
 - >10.000 assets, 100 scenes, various game data
 - Importing costs a lot of time
 - Minimize download size + optimize performance

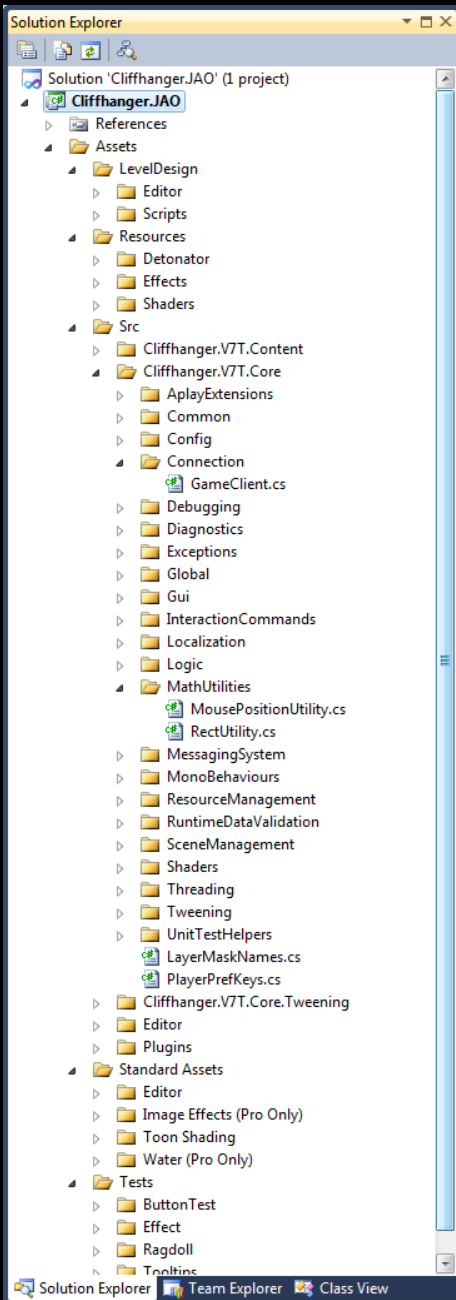
Challenges 2/2

- Create regular deployments for QA, reviews
 - Daily testing in browser
 - Builds have to be available fast
 - Manual build is complicated and error-prone
 - Light map baking takes a lot of time

Approaches

- Coming up in the next 45 minutes:
 1. Visual Studio, Unity and multiple libraries
 2. Client/server code sharing
 3. Unit testing and continuous integration
 4. Splitting Unity projects into smaller ones
 5. Reliable and fast automated build process including distributed light map baking
 6. Optimizing graphics performance



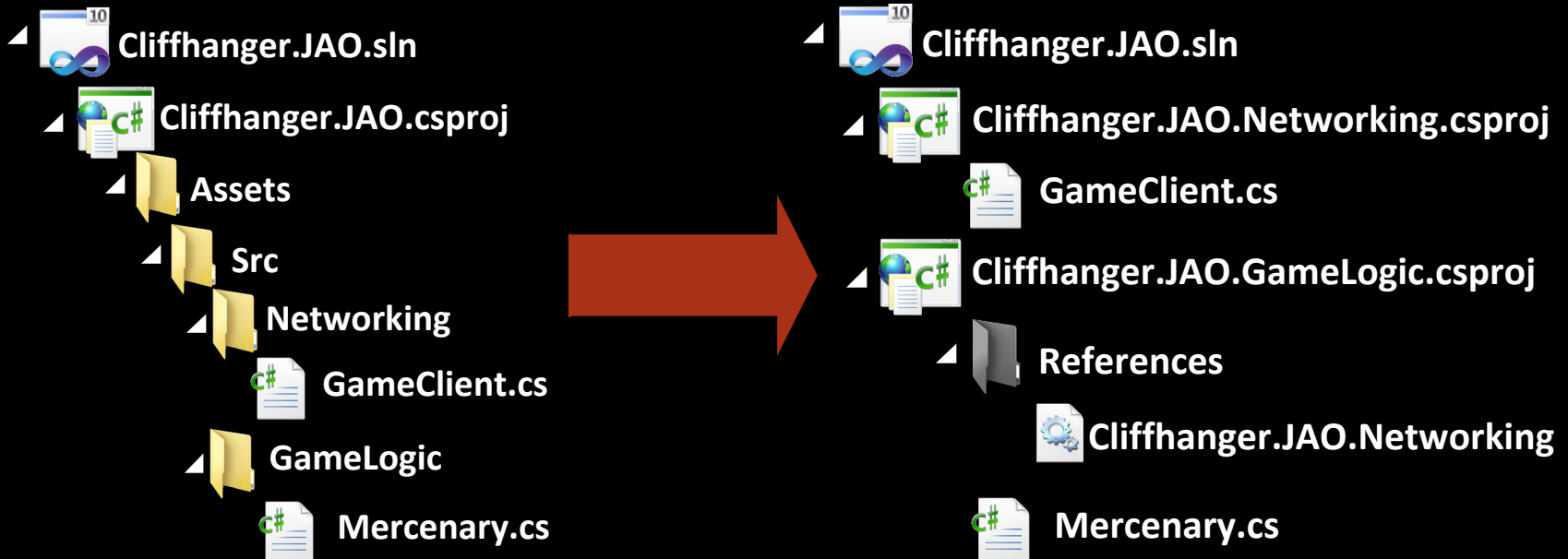


1 Visual Studio and Unity

- Visual Studio is a great tool for C# Unity development
- VS solution synced automatically, BUT:
 - All scripts in single project and namespace
 - Changes in solution or project overwritten
 - Not possible to split project into multiple libraries!

1 Why multiple libraries per Solution?

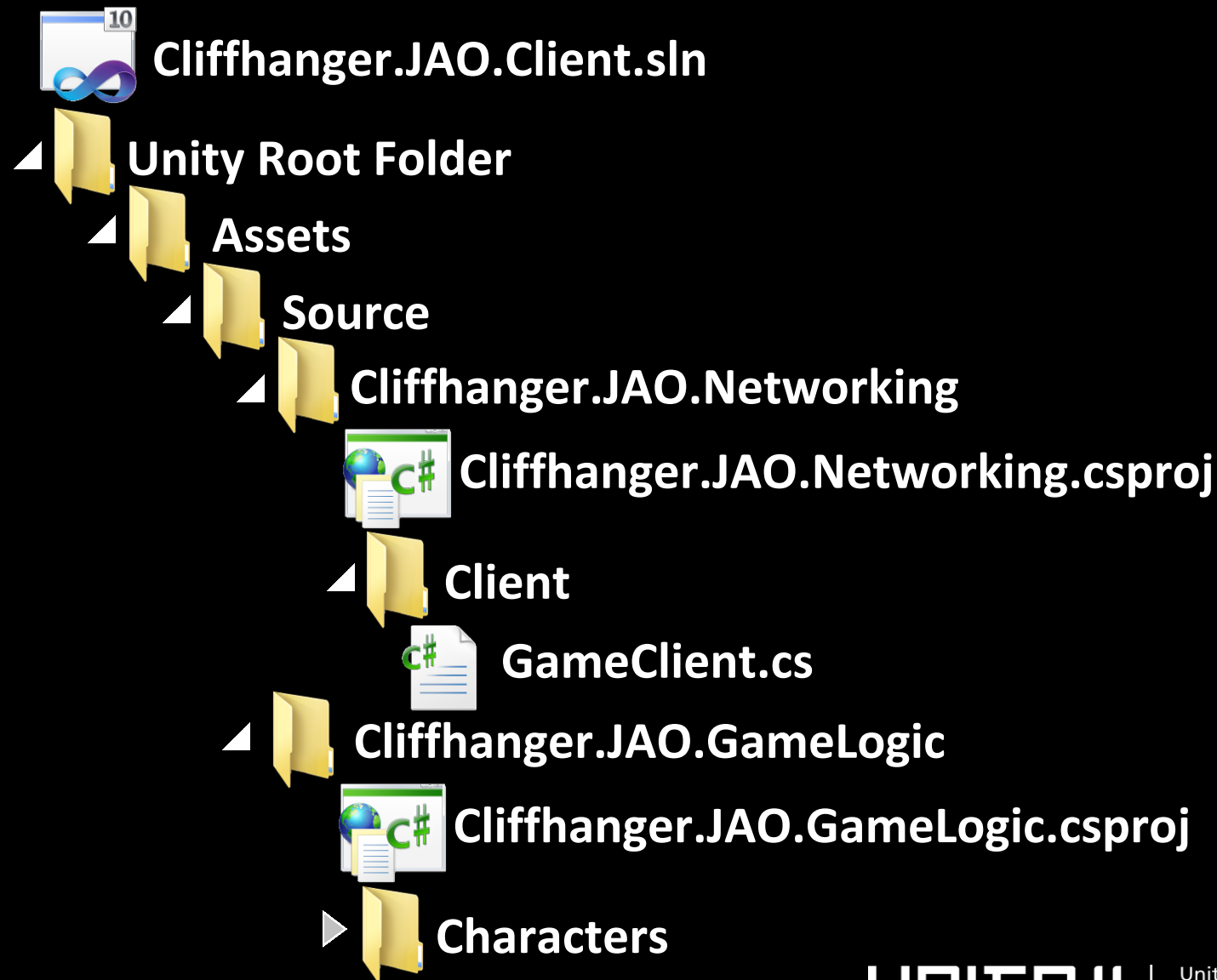
- Breaks codebase into smaller pieces
- Encapsulate certain responsibilities
- Decouple functionality from implementation
- Makes code easily reusable!



1 Custom Visual Studio Solution

- Custom solution outside of Unity asset folder
- Organize classes into multiple libraries in solution and according folders in asset structure
- Some even outside of asset folder (Unit Tests...)
- Namespaces work (except for MonoBehaviours)
- Debugging still possible in MonoDevelop

1 Custom Visual Studio Solution



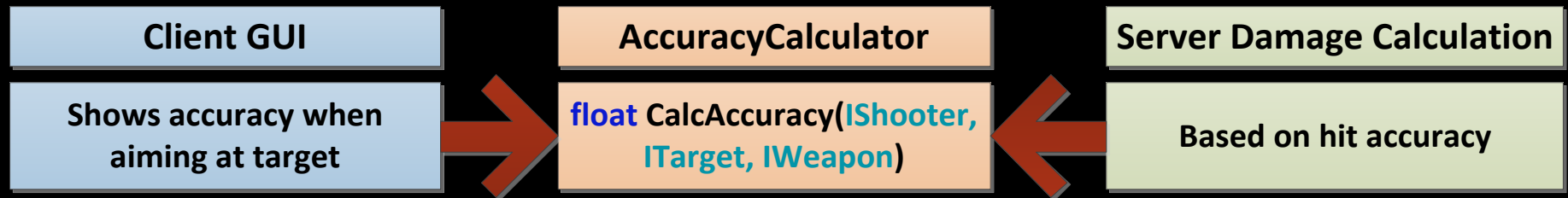
1 Problems with custom VS solutions

- AssemblyInfo.cs generated for each library
 - Class name conflicts in Unity
- Compiled dlls are created in project subfolder
 - Again: class name conflicts in Unity
 - Solution: compile dlls to UnityRoot\Temp folder

```
<PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">
  <BaseIntermediateOutputPath>..\..\..\Temp\obj\Debug\</BaseIntermediateOutputPath>
  <OutputPath>..\..\..\Temp\bin\Debug\</OutputPath>
</PropertyGroup>
<PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Release|AnyCPU' ">
  <BaseIntermediateOutputPath>..\..\..\Temp\obj\Release\</BaseIntermediateOutputPath>
  <OutputPath>..\..\..\Temp\bin\Release\</OutputPath>
</PropertyGroup>
```

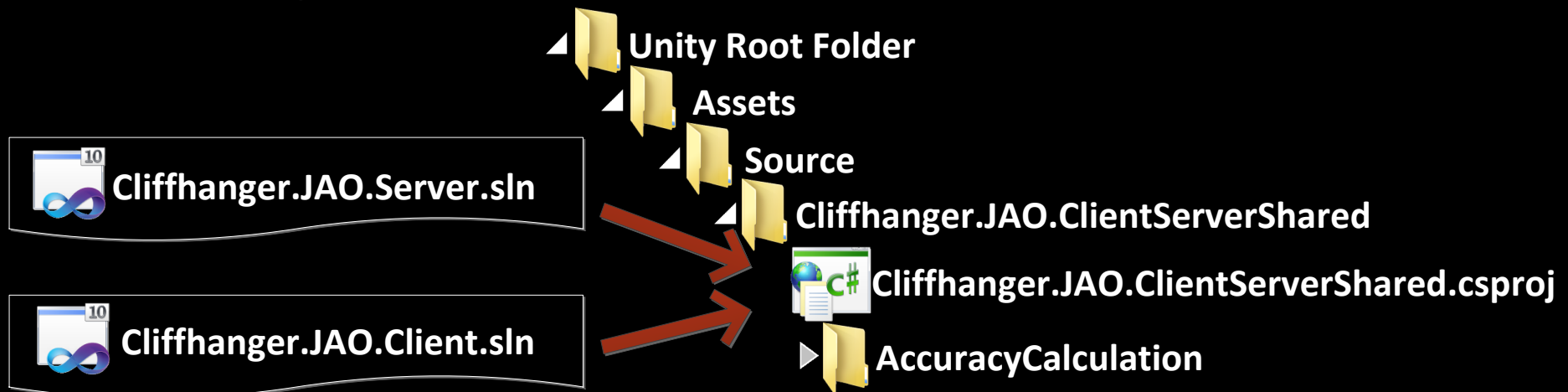
2 Server/Client Code Sharing

- Parts of code and logic same as on server
 - Game data definitions (weapon strength, armor...)
 - Damage calculations, path finding, action points
- Game server developed in C# 4.0
- How to share code without duplicating it?



2 How to share code?

- Move shared code to VS project in Unity folder
- Target framework .NET 3.5 (Unity compatible)
- Can be used in 4.0 applications!
- Add shared project to server solution
- Or: single solution for server + client!



3 Unit Testing your Unity Project

- What is unit testing all about?
 - A method to test „individual units of source code“
 - In OOP, a unit = interface or class
 - Each class is tested separately by multiple test cases to verify functionality
 - Test boundary cases that seldom occur in game
 - Expose found bugs in tests to reproduce them

3 How do Unit Tests work?

```
public class Multiplier
{
    // multiplies two float values and returns the result
    public float Multiply(float a, float b)
    {
        return a * b;
    }
}
```

```
[TestFixture]
class TestMultiplier
{
    Multiplier mMultiplier = new Multiplier();

    [Test]
    public void TestMultiplyIntegers()
    {
        Assert.AreEqual(6, mMultiplier.Multiply(2, 3));
    }

    [Test]
    public void TestMultiplyFloats()
    {
        Assert.AreEqual(1.3f, mMultiplier.Multiply(2.6f, 0.5f));
    }
}
```

3 Unit Testing your Unity project

- „Why would I do that, unit testing is hard and tedious?“
 - Unit tests reduce bugs
 - Ideal code examples = documentation
 - Tests show when functionality has been broken
 - Improves design – decoupled code easier to test
 - Think about the test while implementing code
 - Even better: prior to implementation!

3 How are Unit Tests executed?

- Tests are written in frameworks like NUnit
- Regularly tested using Continuous Integration

```
[TestFixture]
class TestMultiplier
{
    Multiplier mMultiplier = new Multiplier();

    [Test]
    public void TestMultiplyIntegers()
    {
        Assert.AreEqual(6, mMultiplier.Multiply(2, 3));
    }
}
```

Projects | My Changes | Agents (3) | Build Queue (0)

Welcome, Johannes Scharl

Administration > V7T Project > Continuous Integration Windows Code Configuration

Build Steps

There are 2 build steps defined.

Build Step	Description	edit	more
Visual Studio (sln)	Build file path: nonUnity/source/solutions/All.sln Targets: Rebuild Configuration: Debug Platform: x86		
NUnit	Runtime : NUnit-2.5.9 v4.0 x86 Run tests on : unity/**bin/x86/**/*.UnitTests.dll nonUnity/**bin/x86/**/*.UnitTests.dll Collect .NET code coverage data with NCover (3.x)		

V7T > Continuous Integration Windows Code

Overview | History | Change Log | Issue Log | Statistics | Compatible Agents (1) | Pending Changes (41) | Settings

#	Results	Artifacts	Changes	Started
#2177	Tests passed: 729, ignored: 10	View	jscharl@cliffha... (1)	09 Sep 11 19:34
#2176	Tests passed: 729, ignored: 10	View	Changes (2)	09 Sep 11 18:44
#2175	Tests failed: 8 (8 new), passed: 707, ignored: 10	View	Changes (3)	09 Sep 11 18:16

3 Test code that uses Unity Objects?

- Unity objects can't be used outside of Unity!

```
using NUnit.Framework;
using UnityEngine;

namespace Cliffhanger.JAO.UnitTests
{
    [TestFixture]
    class TestGameObject
    {
        [Test]
        public void TestGameObjectCreation()
        {
            var gameObj = new GameObject { active = true };
            Assert.AreEqual(true, gameObj.active);
            GameObject.DestroyImmediate(gameObj);
        }
    }
}
```

100 %

Error List

1 Error | 1 Warning | 0 Messages

	Description
2	Test 'Cliffhanger.JAO.UnitTests.TestGameObject.TestGameObjectCreation' failed: System.Security.SecurityException : ECall methods must be packaged into a system module. at Cliffhanger.JAO.UnitTests.TestGameObject.TestGameObjectCreation()

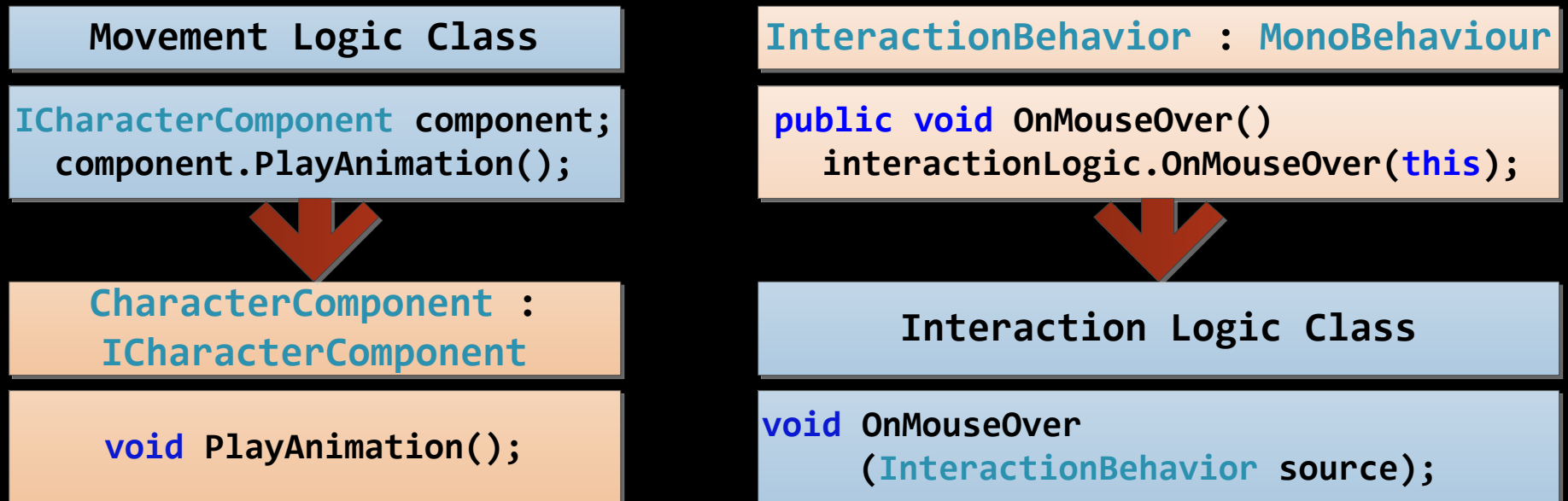
3 How test code depending on Unity?

- Our solution: modified SharpUnit framework (<http://www.unifycommunity.com/wiki/index.php?title=SharpUnit>)
- Allows to write tests very similar to NUnit
- Can be run from Continuous Integration



3 Unit testing - Advice

- Test as much code as possible with NUnit!
- If not possible: code depends on Unity classes
- Decouple logic code from Unity dependencies
 - Easier to test and reuse!

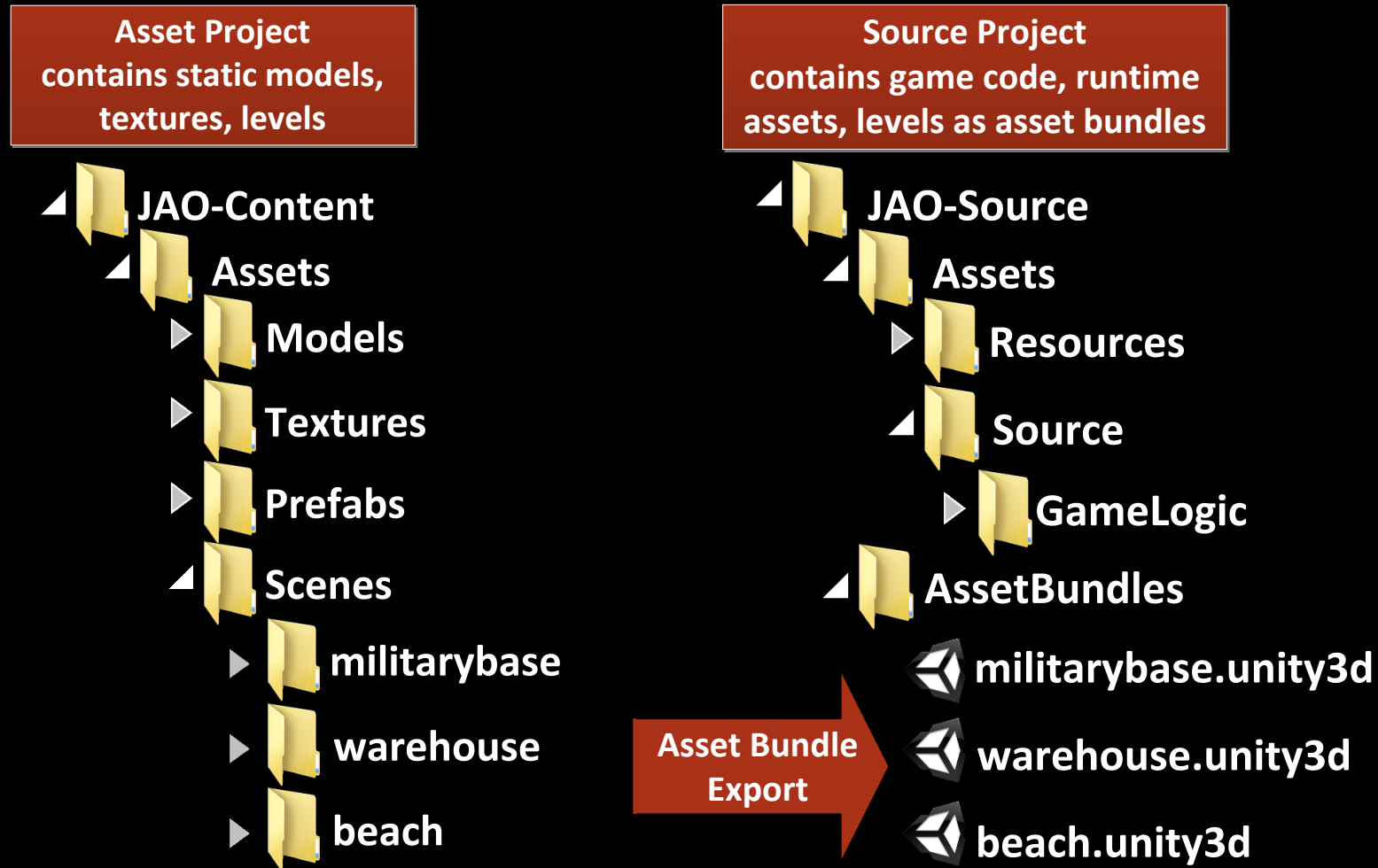


4 Splitting up Unity projects

- Hundreds of assets created/modified per day
- Programmers update and import them
- Our solution: Split up Unity project
 - Source project: code and resources
 - Asset project: all static assets and levels



4 Splitting up Unity projects



4 Advantages of multiple projects

- Less overhead for updating and importing assets
- Initial download smaller, no levels or static assets
- Forces developers to decouple code from scenes
 - Shared code should be minimal

4 Splitting up Unity projects

- What to do with Code used in both projects?
 - Custom components on objects used in code
- Solution: move components to separate library
 - Copy compiled dll to both projects
 - post build event



5 Deploying Jagged Alliance Online

- Deployments needed daily for QA and reviews
- Deployment consists of many different parts
- Some solve this with a 10h+ manual routine
- We don't.



5 Deployment: Challenges

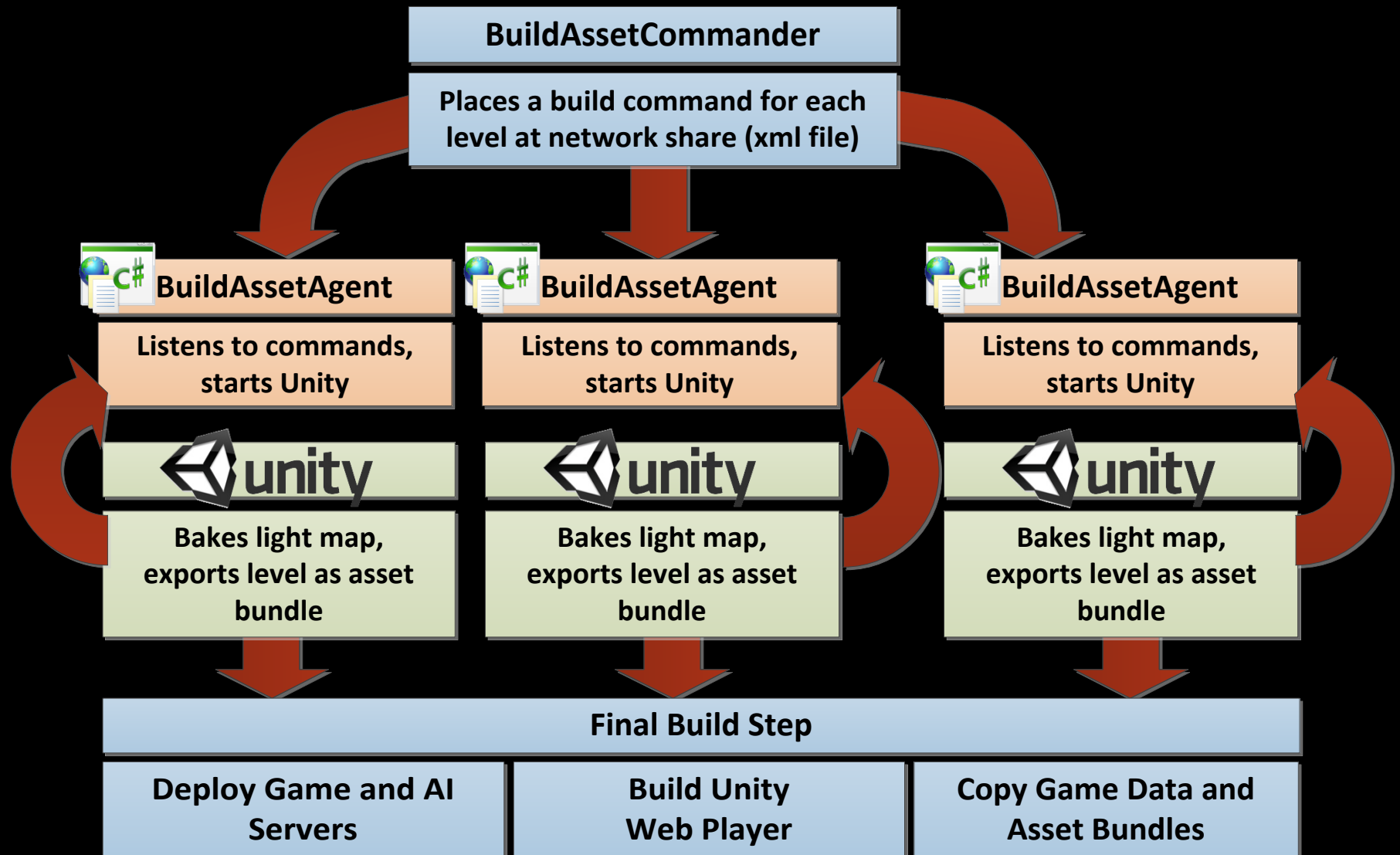
- Problem 1: Exporting Levels
 - Bake light map + exporting asset bundle: 1h/level
 - 100+ levels = 100h = 4 days!
- Problem 2: Stability
 - Unity started in headless mode from CI (TeamCity)
 - Asset project: >7GB
 - Occasionally freezes or crashes
 - Needs to be fail-safe

5 Basic deployment process

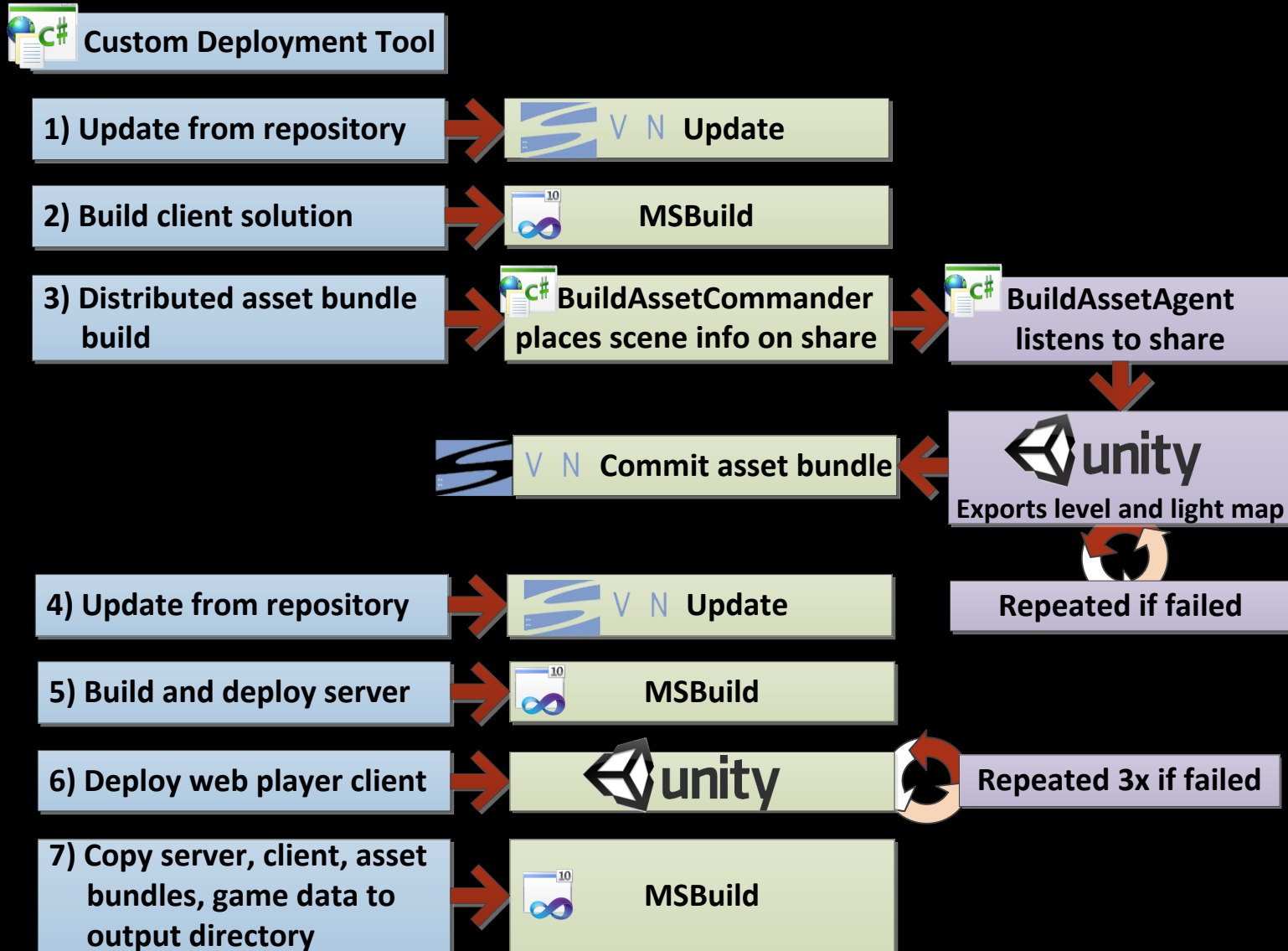
- Idea: Distribute deployment
 - Build agents run on multiple machines
 - Export levels from a list in parallel
 - Can easily be scaled up
 - Failed exports are repeated
- Final web player build step
 - Compiles web player
 - Collects asset bundles and balancing data



5 Basic deployment process



5 Deployment Implementation

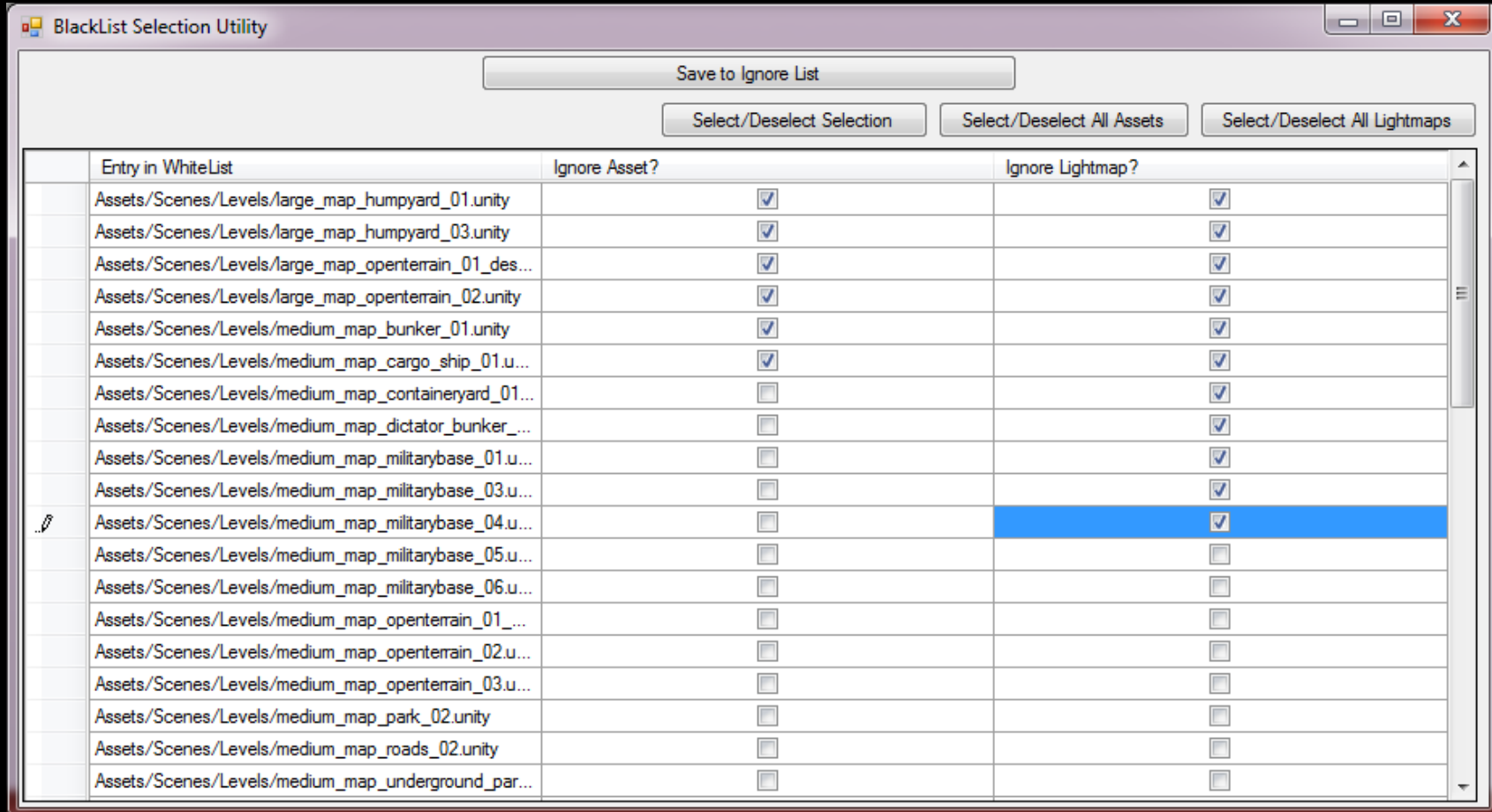


5 Can this process be optimized?

- Building 100 maps still takes a long time
- Not all of them change every day
- Detecting changes automatically is difficult
- Our solution: level blacklist

5 Level Deployment Blacklist

- List is saved as XML, used by Build Asset Commander



5 What about quickly available Builds?

- Nightly build great for daily tests, but:
 - Fast solution necessary for milestones
 - Should run locally on one machine
- Manually triggered deployment process
 - No light map baking
 - Select steps to build and deploy



5 Manual triggered Deployment

- Same technology as automatic deployment
- Asset bundle export step different



6 Graphics Performance

- Biggest challenge:
 - Game should look great on cutting edge hardware
 - Game must perform well on low-end laptops



6 Great graphics, but slow hardware?

- Our approach: Extended quality levels
 - 3 configurations triggering Unity quality levels: Good, Beautiful & Fantastic
 - Different rendering paths (Forward/Deferred)

```
// Set the correct rendering path for this quality setting
Camera.mainCamera.renderingPath = currentQualitySettings.RenderingPath;
```

- Toggle layers and effects

6 Quality Levels

	Fantastic	Beautiful	Good
Rendering Path	Deferred	Deferred	Forward
Shadows	Hard + Soft Shadows High Resolution	Hard Shadows Only Medium Resolution	No Shadows
Lights	20 Pixel Lights, Mood Lights	5 Pixel Lights no Mood lights	Vertex Lights only no Mood Lights
Anti Aliasing	2x	✗	✗
Particle Effects	✓	✓	✗
Decals	✓	✓	✗
Water	High Quality Prefab	High Quality Prefab	Low Quality Prefab
Normal Maps	✓	✓	✗
Post Processing	✓	✗	✗
Detonator Quality	1.0	0.5	0.0

6 Low End: Dual Core Netbook, 20fps



6 High End: Quad Core, 8800 GT, >60fps



Thank you for your attention!



JAGGED ALLIANCE ONLINE



Questions?

Johannes Scharl

jscharl@cliffhanger-productions.com