
GDC
09
learn
network
inspire
www.GDConf.com
Game Developers Conference®
March 23-27, 2009 | Moscone Center, San Francisco

Robotic Testing

(to the rescue)

Bert Chang and Paul Du Bois
Double Fine Productions

About us

- Paul: Senior Programmer
- Bert: Software Test Engineer
- RoBert:
  Robot brainchild
  Automated tester

120-second pitch

- Unit testing is well understood
- “But how do we test game logic…”
- We implemented a prototype
- “Hey, it works…”

120-second pitch

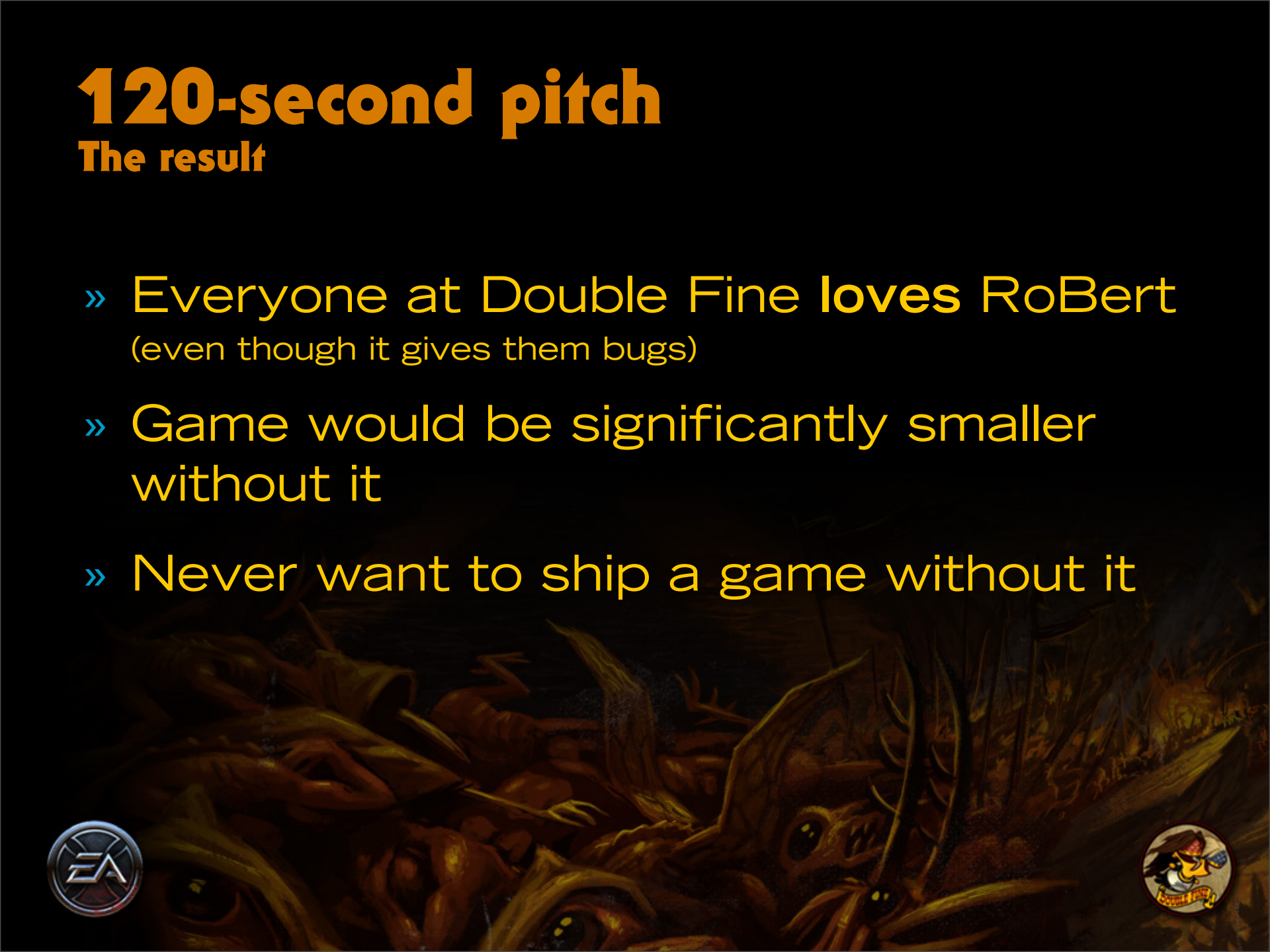
- Unit testing is well understood
- “But how do we test game logic…”
- We implemented a prototype
- “Hey, it works… really well!”

120-second pitch
The result

- Framework for writing very high-level code to exercise game
- Runs on any idle devkit
- Used directly by
  - Test
  - Gameplay, System programmers
  - Designers

120-second pitch
The result

- Everyone at Double Fine **loves** RoBert
  (even though it gives them bugs)
- Game would be significantly smaller without it
- Never want to ship a game without it



60-second pitch

The result

Demo time!

60-second pitch

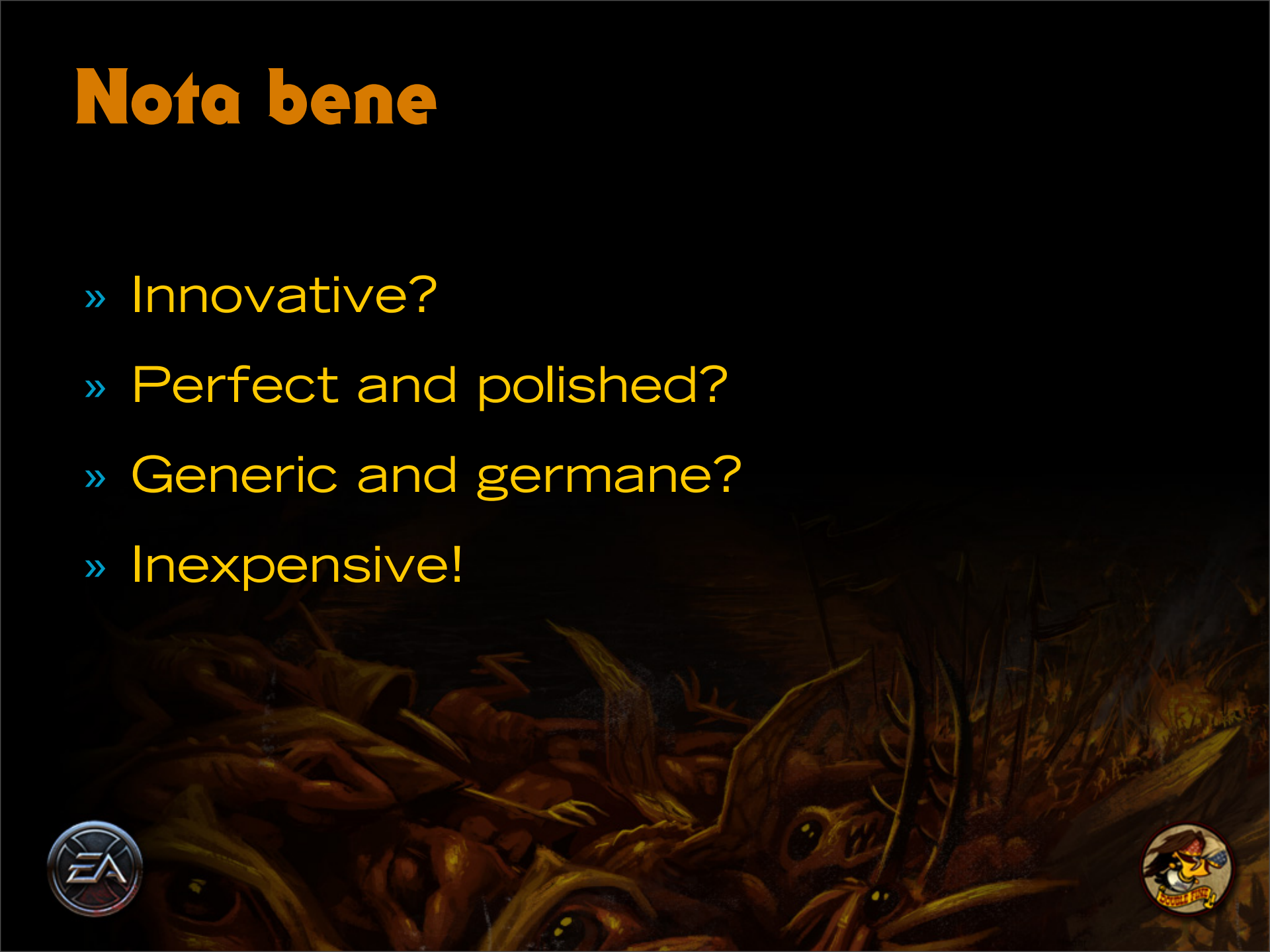
(video)

Overview of talk

- Motivation
- Implementation
- Uses and examples
- Analysis and future work
- Q&A + discussion period

Nota bene

- Innovative?
- Perfect and polished?
- Generic and germane?
- Inexpensive!



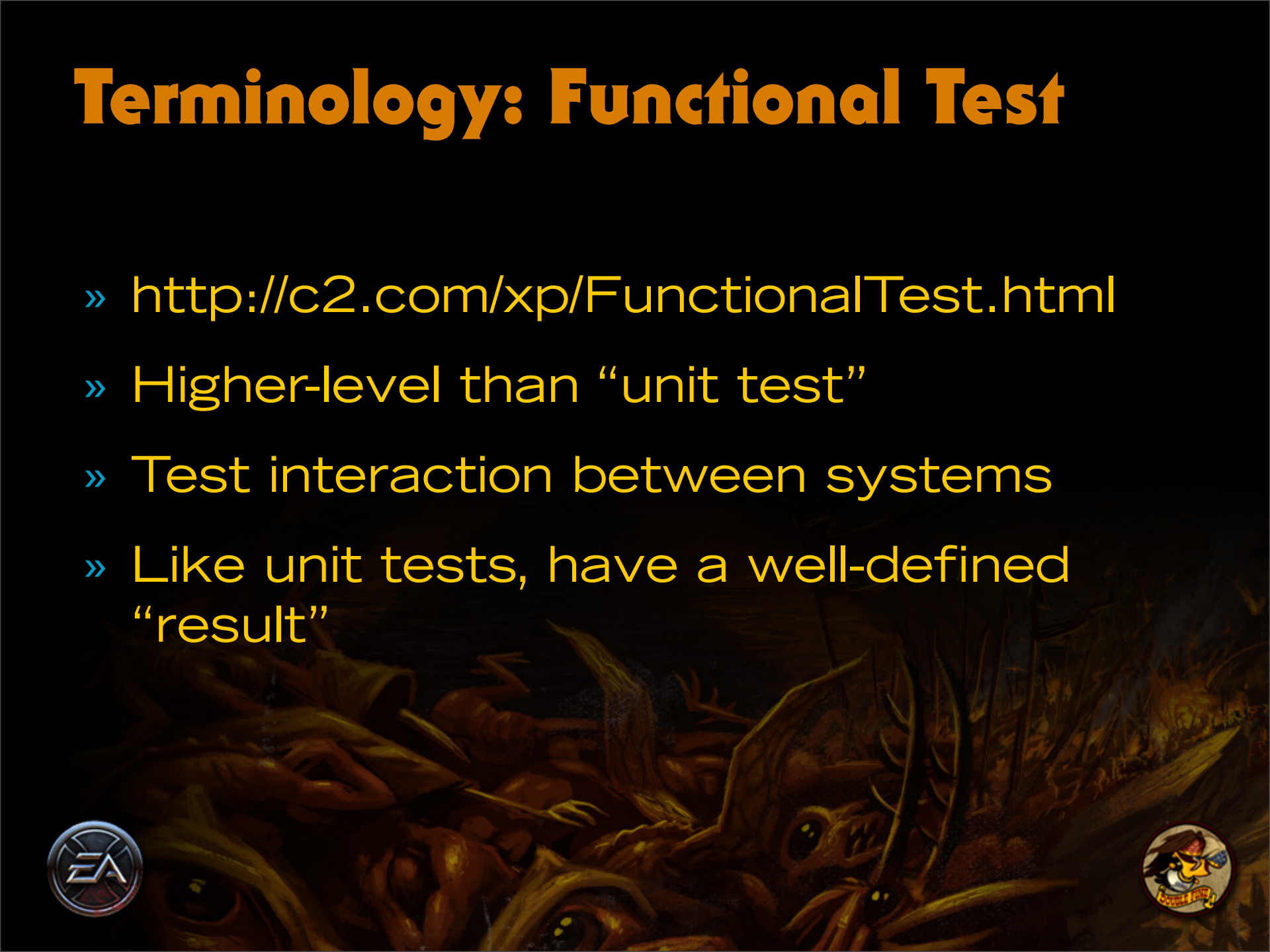
Mötivation

Terminology: Unit Test

- http://c2.com/xp/UnitTest.html
- Individual “unit” of functionality
- Tests should run quickly
- Doesn't tend to test interaction between systems

Terminology: Functional Test

- http://c2.com/xp/FunctionalTest.html
- Higher-level than “unit test”
- Test interaction between systems
- Like unit tests, have a well-defined “result”



Problem summary

Problem summary

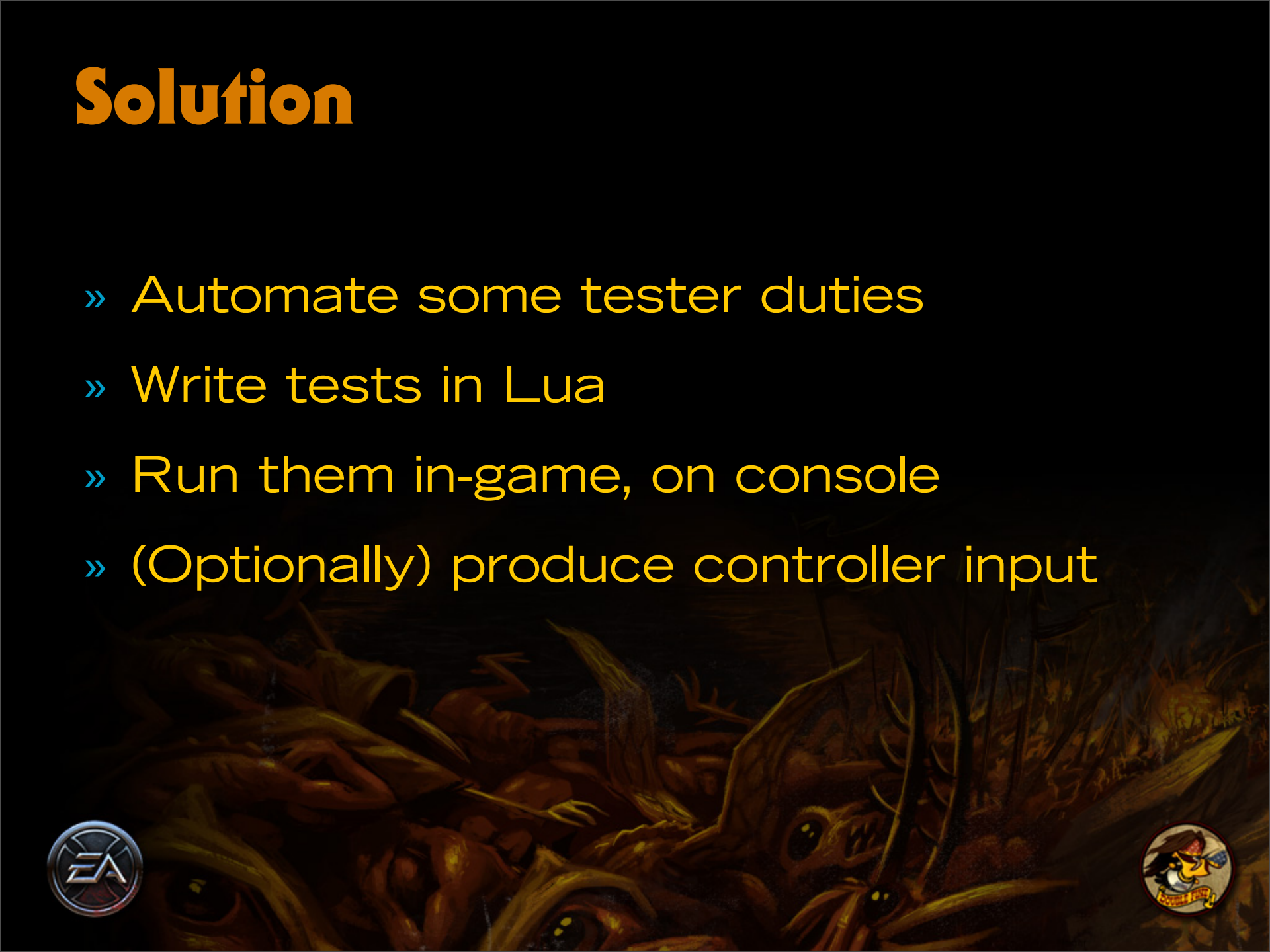
- Brütal Legend is big
- ...big technical challenge
- ...big design
- ...big landmass

Problem summary

- Double Fine is small
- Test team is very small
- Build breakages (theoretical)

Solution

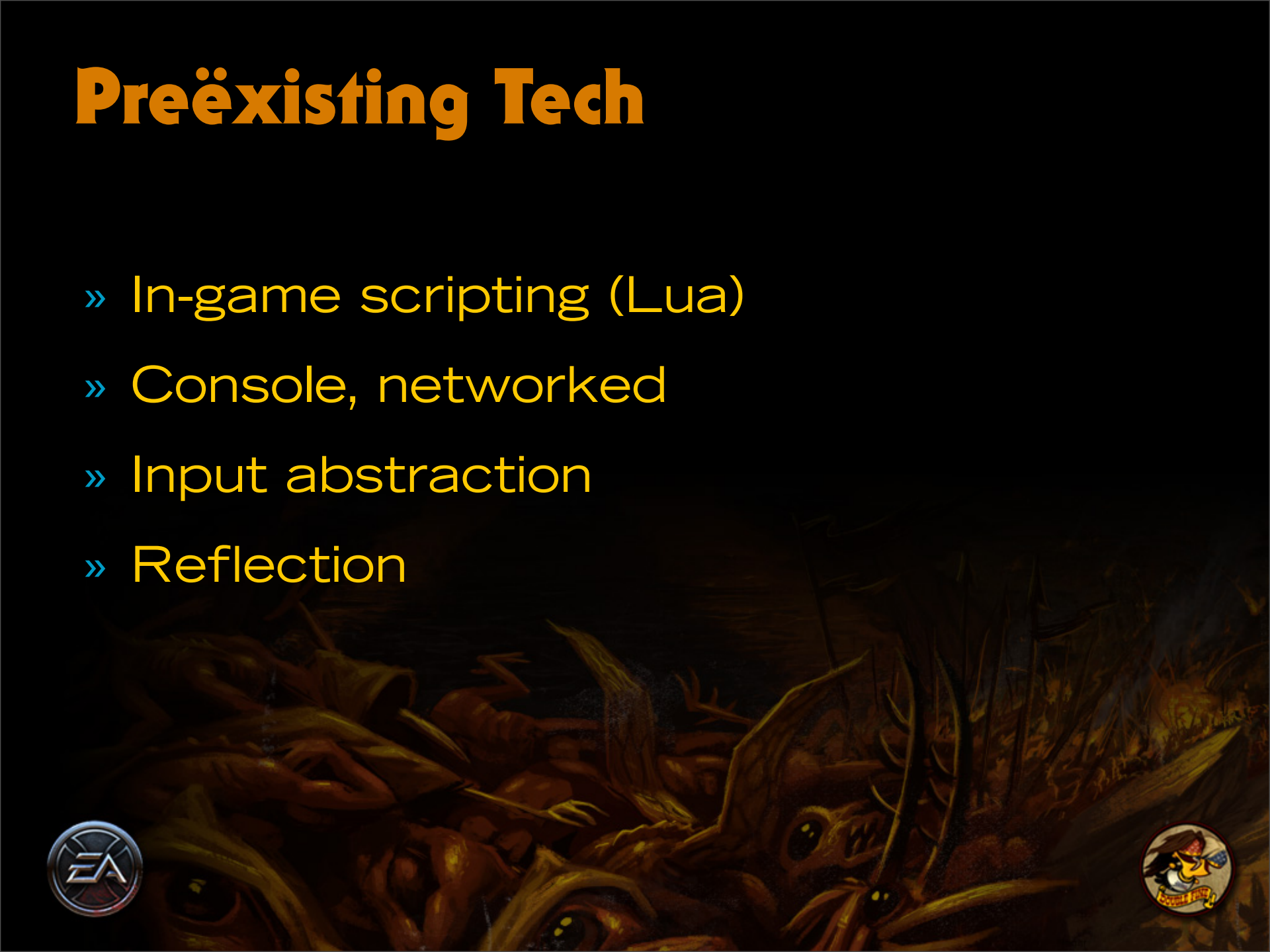
- Automate some tester duties
- Write tests in Lua
- Run them in-game, on console
- (Optionally) produce controller input



Implementation

Preëxisting Tech

- In-game scripting (Lua)
- Console, networked
- Input abstraction
- Reflection



In-game scripting

- We use Lua 5.1 (http://www.lua.org)
- Tiny code footprint
- Reasonable memory footprint
- Compiler and interpreter
- Also used for console commands

Console, networked

- Simple TCP-based messaging
- Game sends debug output
- Game receives and executes commands
- Host-side tools in C# and Python

Input abstraction

- Multiple possible input sources
  - From file
  - From network
  - From device
  - From script

Reflection

Entity A02_Headbanger2F3

CoPhysics

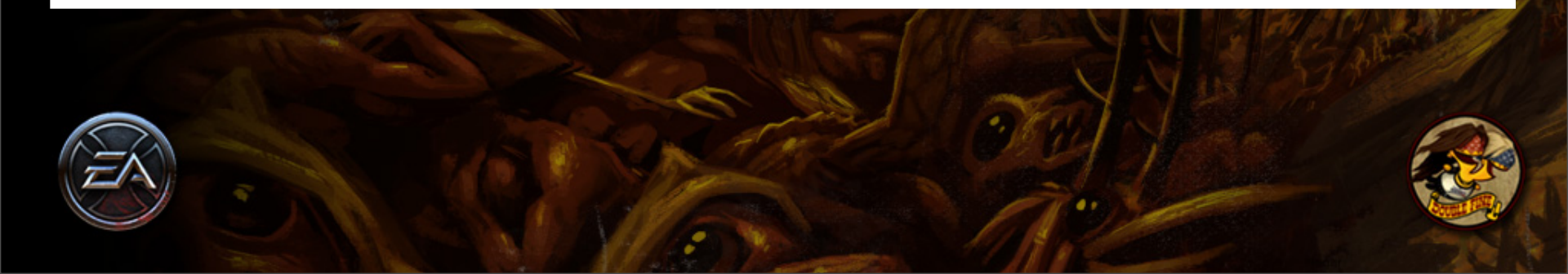
Pos: (3,4,5)
Mass: 10

CoController

State: Idle

CoDamageable

Health: 30
Ragdoll: true



Reflection + Lua

```
function Class:waitForActiveLine(self, ent)
  while true do
    self:sleep(0)
    if ent.CoVoice.HasActiveVoiceLine then
      return
    end
  end
end
```

New tech

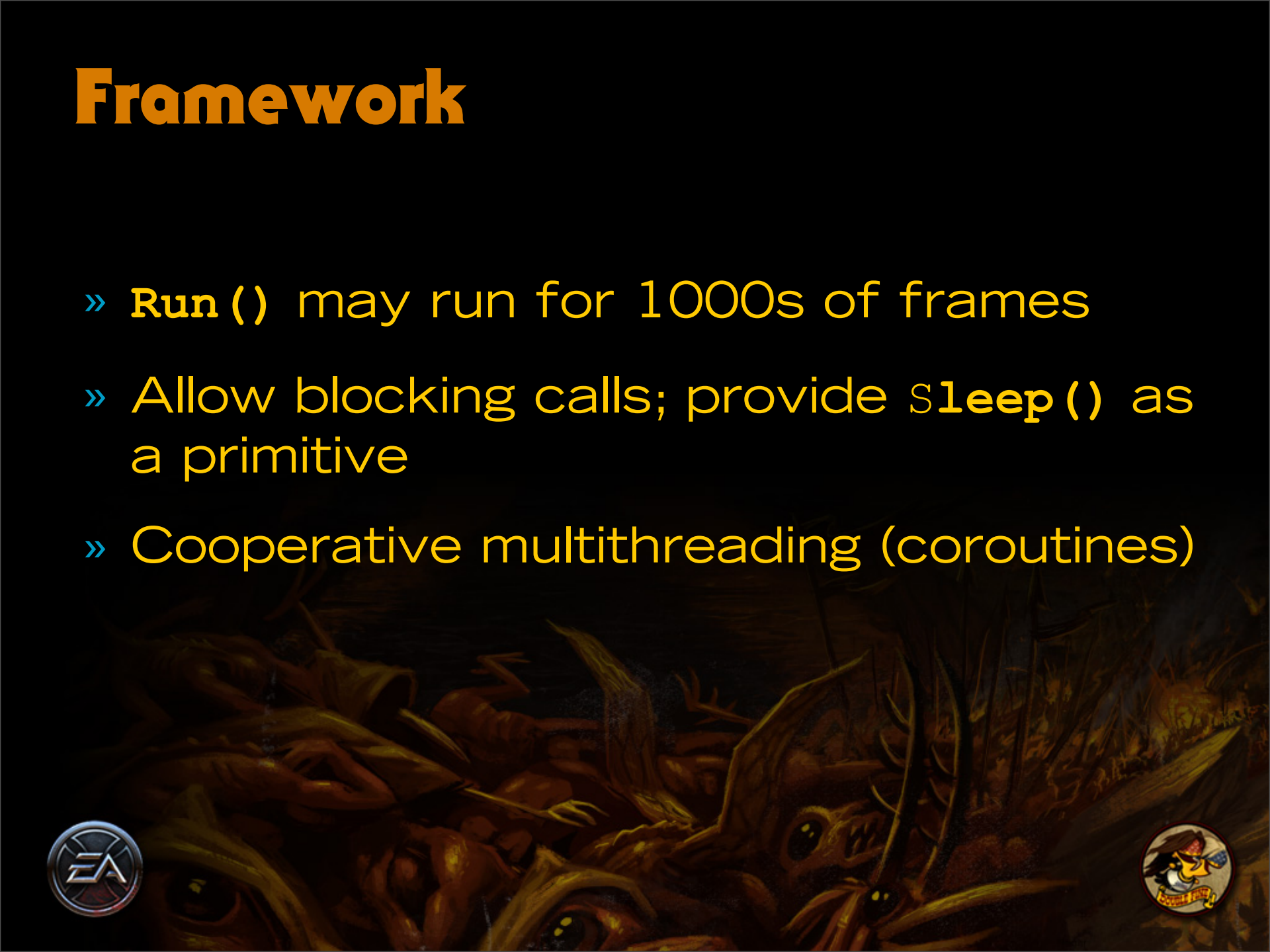
- Test framework (on console)
- Test runner (on host PC)
- “Bot Farm”

Framework

- Similar to unit test framework
- Create class, implement **`Setup()`**, **`Teardown()`**, **`Run()`**, ...
- Call **`ASSERT()`** method on failure
- Return from **`Run()`** signals success

Framework

- `Run()` may run for 1000s of frames
- Allow blocking calls; provide `Sleep()` as a primitive
- Cooperative multithreading (coroutines)



Framework

- Test can function as input source
- Mutate a state block
- Use blocking calls to make API convenient
- Manipulate joystick in “world coordinates”

Example: providing input

```
-- push some button for time t1

self.input.buttons[btn] = true
self:sleep(t1)
self.input.buttons[btn] = false

-- move towards world-space pos x,y,z

self.input.joy1 = test.GetInputDir(x,y,z)
```

Example: simple mission

```
function Class:Run()
  function fightSpiders(entity)
    self:attackSmallSpiders()
    self:killHealerSpiders()
    self:basicFightFunc(entity)

  self:waypointAttack(
    "P1_050_1", "Monster", 40, fightSpiders)
  self:attackEntitiesOfTypeInRadius(
    "Monster", 50, fightSpiders)
  self:attackBarrier("A_WebBarrierA", 100)
  self:waypointTo{"P1_050_ChromeWidowLair"}
```

Example: reproduce a bug

```
function Class:Run()
  function waitForActiveLine()
    while true do
      self:sleep(0)
      if player.CoVoice.HasActiveVoiceLine then
        return

  streams = sound.GetNumStreams()
  while true do
    game.SayLine( 'MIIN001ROAD' )
    game.SayLine( 'MIIN001ROAD' )
    waitForActiveLine()
    if sound.GetNumStreams() > streams then
      self:sleep(1)
      self:ASSERT(sound.GetNumStreams() <= streams)
```

Test runner

- Launch test
- Watch output stream for messages (start, fail, heartbeat)
- Watch for warning, assert, stack dump
- Exceptional results are reported via email

Dynamic Bot Farm

- Find unused devkits and run tests on them
- Perform intelligent test selection
- Record results

Role of the human

- Initially, start tests by hand
- Bot farm means more time writing bugs
- Half time writing new tests, updating old tests, writing/regressing bugs
- Half time on infrastructure work

Uses and Examples

Not built in a day

- Will quickly go over the various uses we found for the framework
- Not all uses are related to testing
- Please note down which ones you're interested in and ask!

Initial tests

- Before controller interface was written
- Convinced us that project was useful
- Does the game start/quit/leak memory?
- Do these entities spawn properly?
- Can this unit pathfind properly?

More tests

- Can player interact with this unit?
- Can bot fly across the world without the game crashing?
- Can bot join a multiplayer game with another bot?
- Are any desyncs generated?
- Do “debuffs” work properly?

More tests

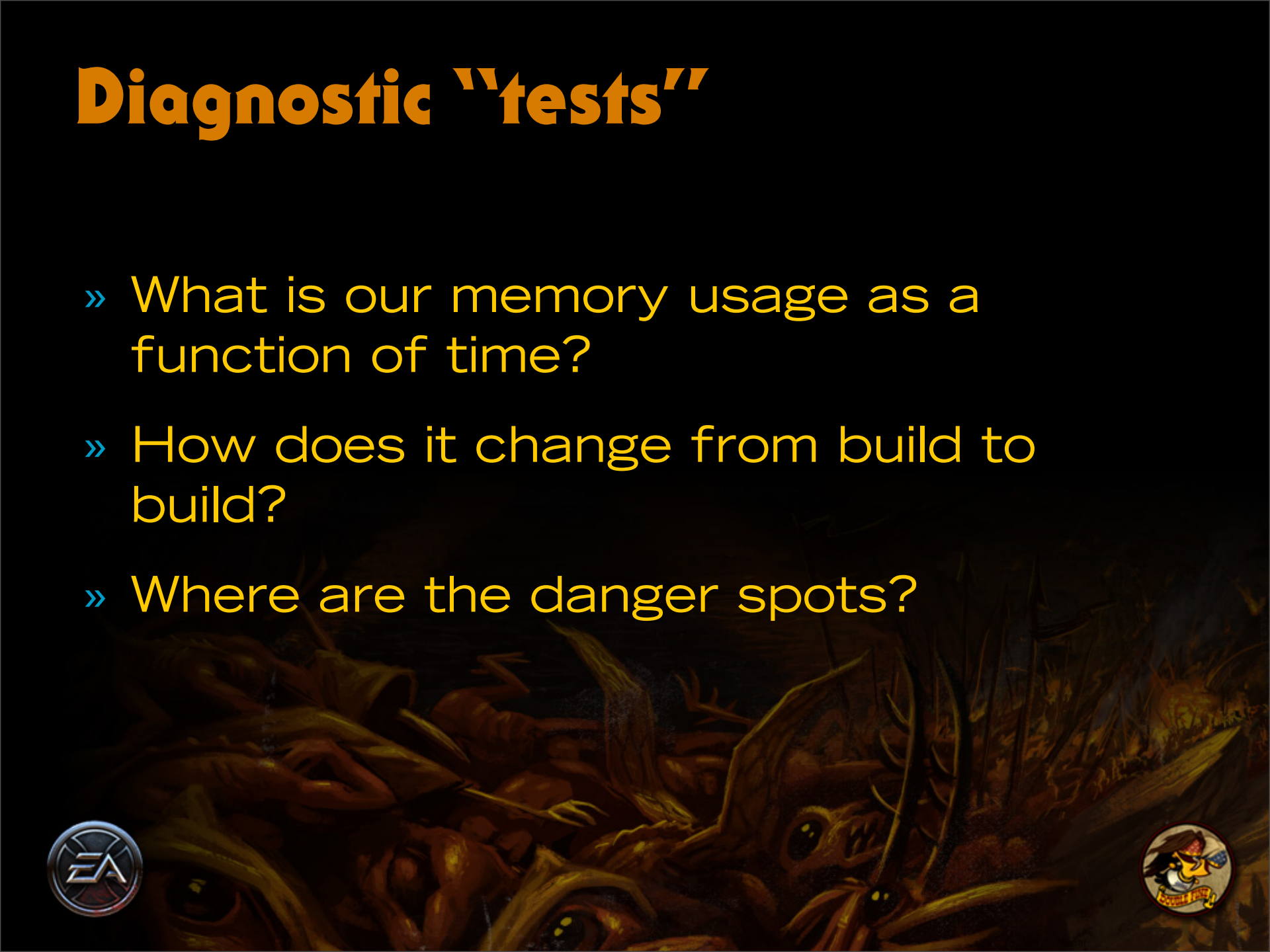
- Can I go to each mission contact and talk to them?
- Can I complete each contact's mission?
- Can I successfully fail the mission?
- Multiplayer!

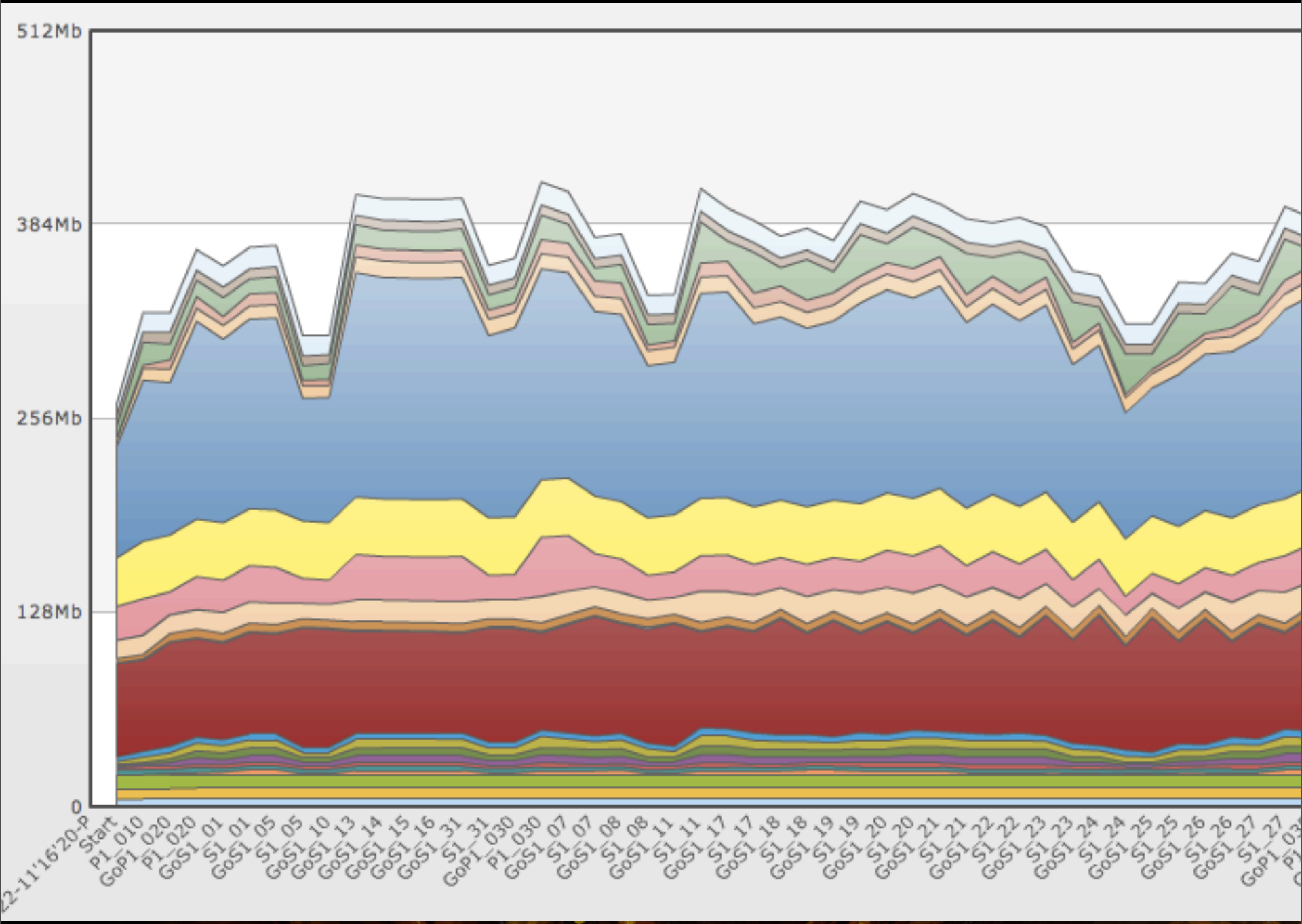
Test-writing strategies

- Bot is not sophisticated
- Means lower impact when missions change
- Means less-precise diagnostic when test fails
- Not a big deal in practice

Diagnostic "tests"

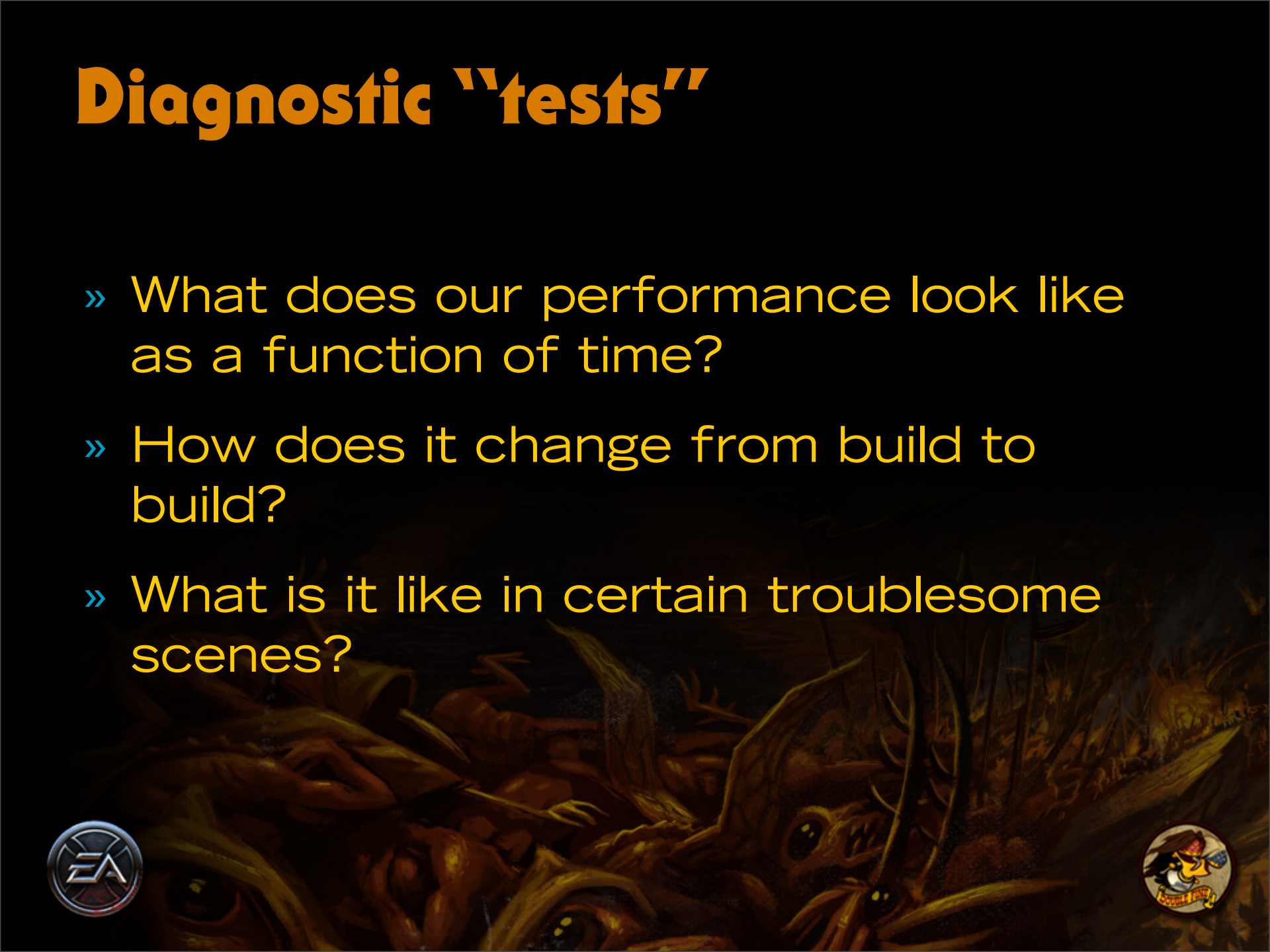
- What is our memory usage as a function of time?
- How does it change from build to build?
- Where are the danger spots?

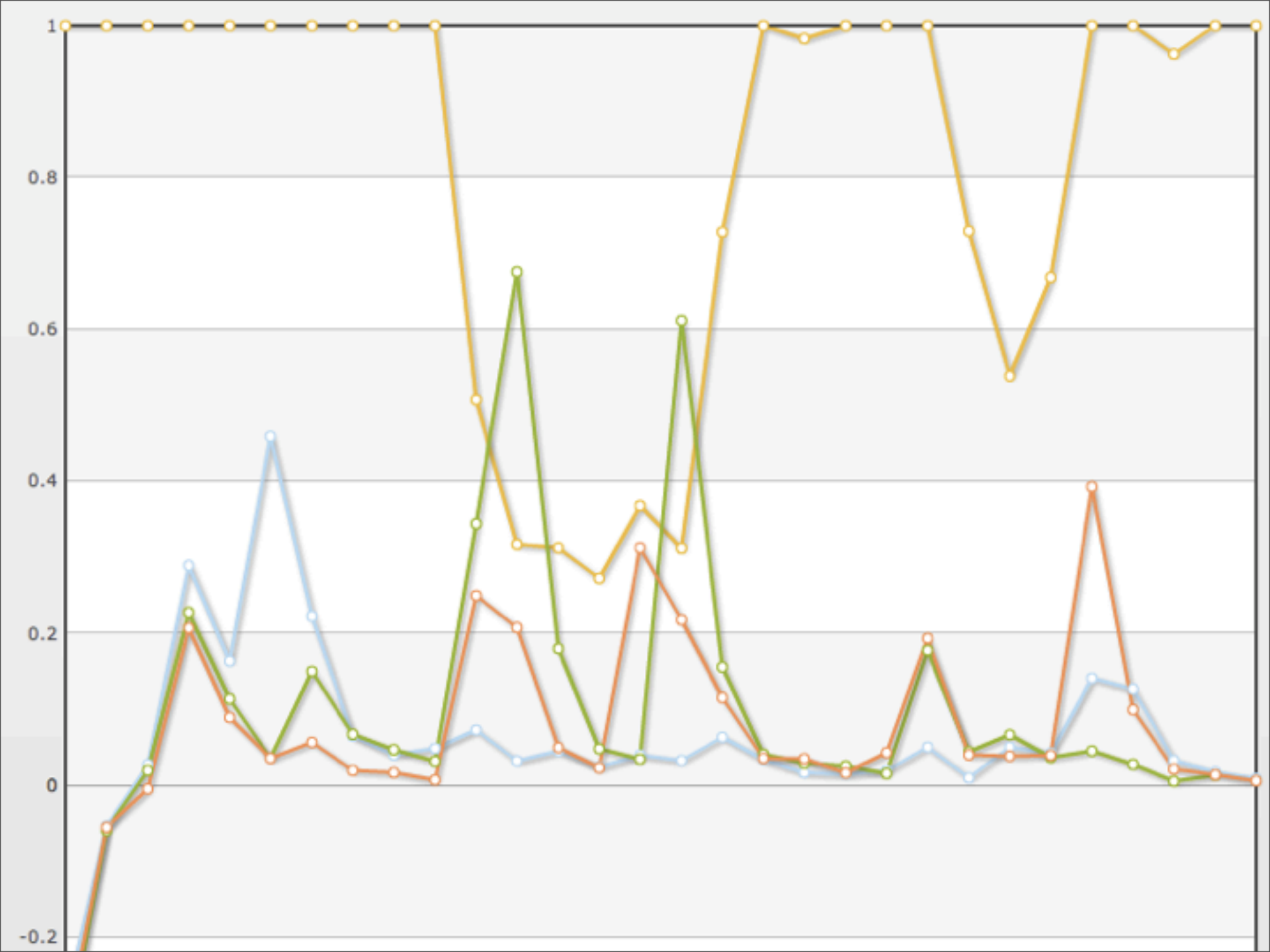



512Mb
384Mb
256Mb
128Mb
0
22-11'16'20-P
Start
P1_010
GoP1_020
P1_020
GoS1_01
S1_01
GoS1_05
S1_05
GoS1_10
GoS1_13
GoS1_14
GoS1_15
GoS1_16
GoS1_31
S1_31
GoP1_030
P1_030
GoS1_07
S1_07
GoS1_08
S1_08
GoS1_11
S1_11
GoS1_17
S1_17
GoS1_18
S1_18
GoS1_19
S1_19
GoS1_20
S1_20
GoS1_21
S1_21
GoS1_22
S1_22
GoS1_23
S1_23
GoS1_24
S1_24
GoS1_25
S1_25
GoS1_26
S1_26
GoS1_27
S1_27
GoP1_03

Diagnostic "tests"

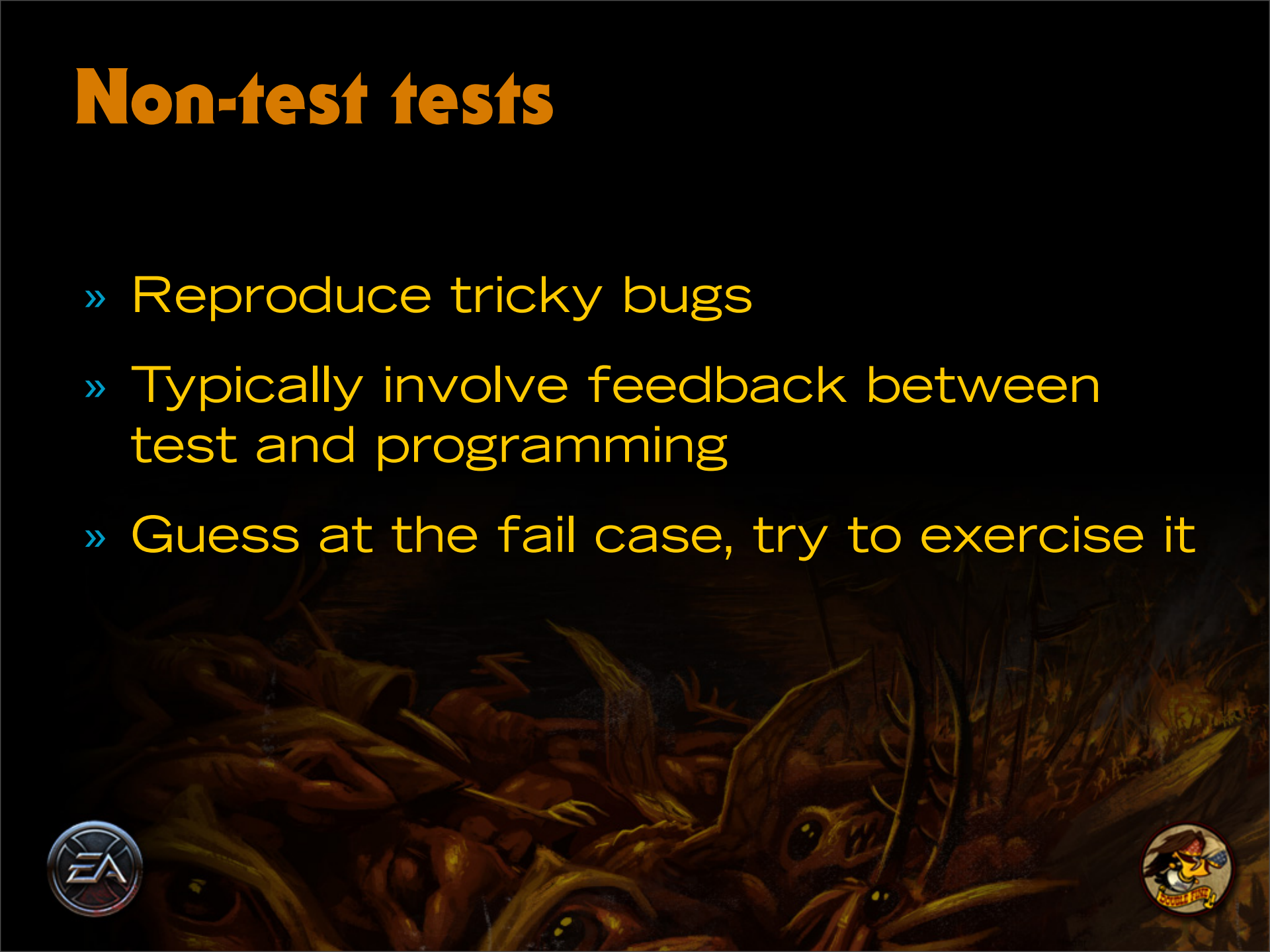
- What does our performance look like as a function of time?
- How does it change from build to build?
- What is it like in certain troublesome scenes?




1
0.8
0.6
0.4
0.2
0
-0.2

Non-test tests

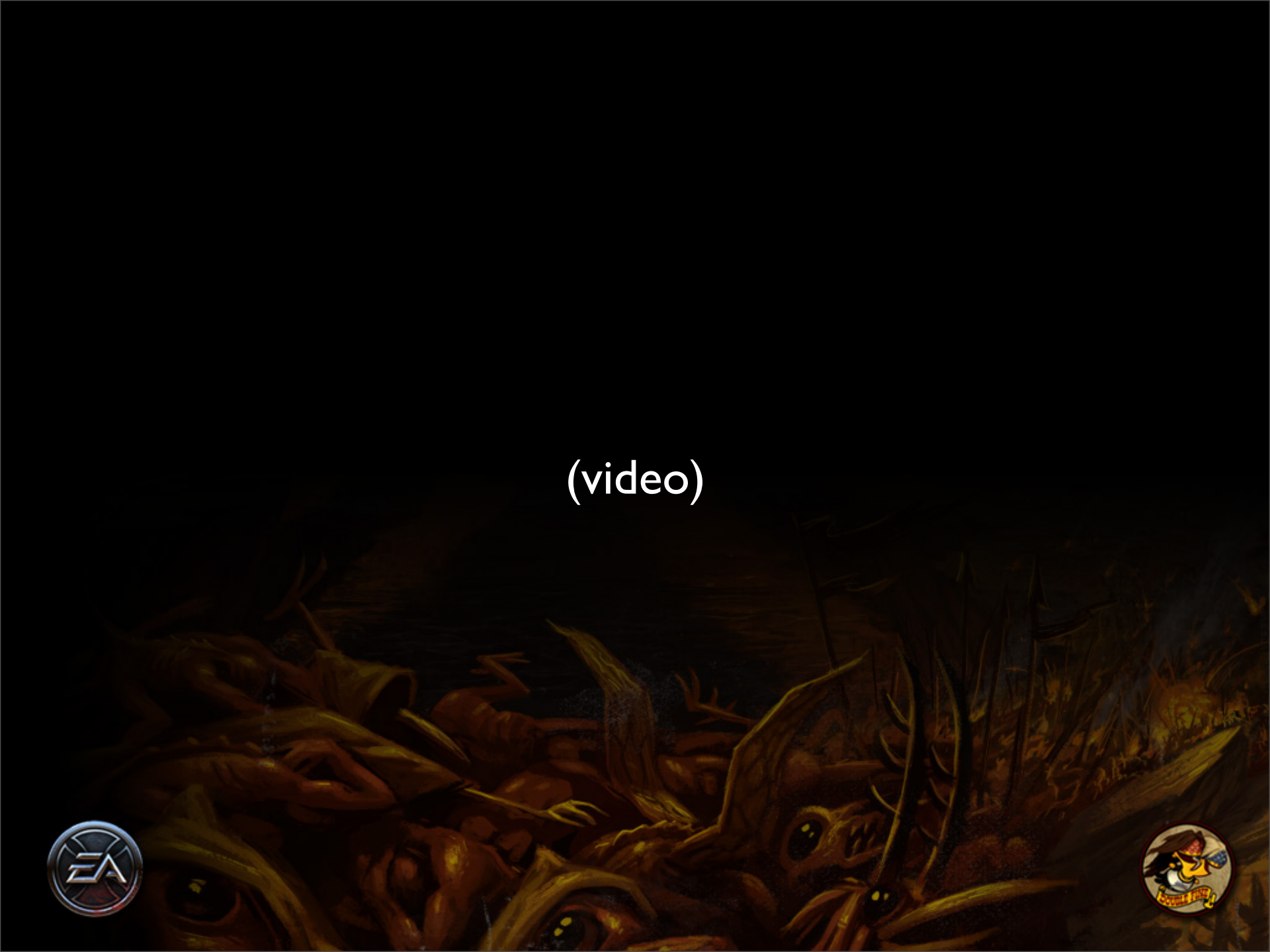
- Reproduce tricky bugs
- Typically involve feedback between test and programming
- Guess at the fail case, try to exercise it



Use by programmers

- Pre-checkin verification
- Soak testing for risky changes
- Can use Debug builds!

(video)



Use by designers

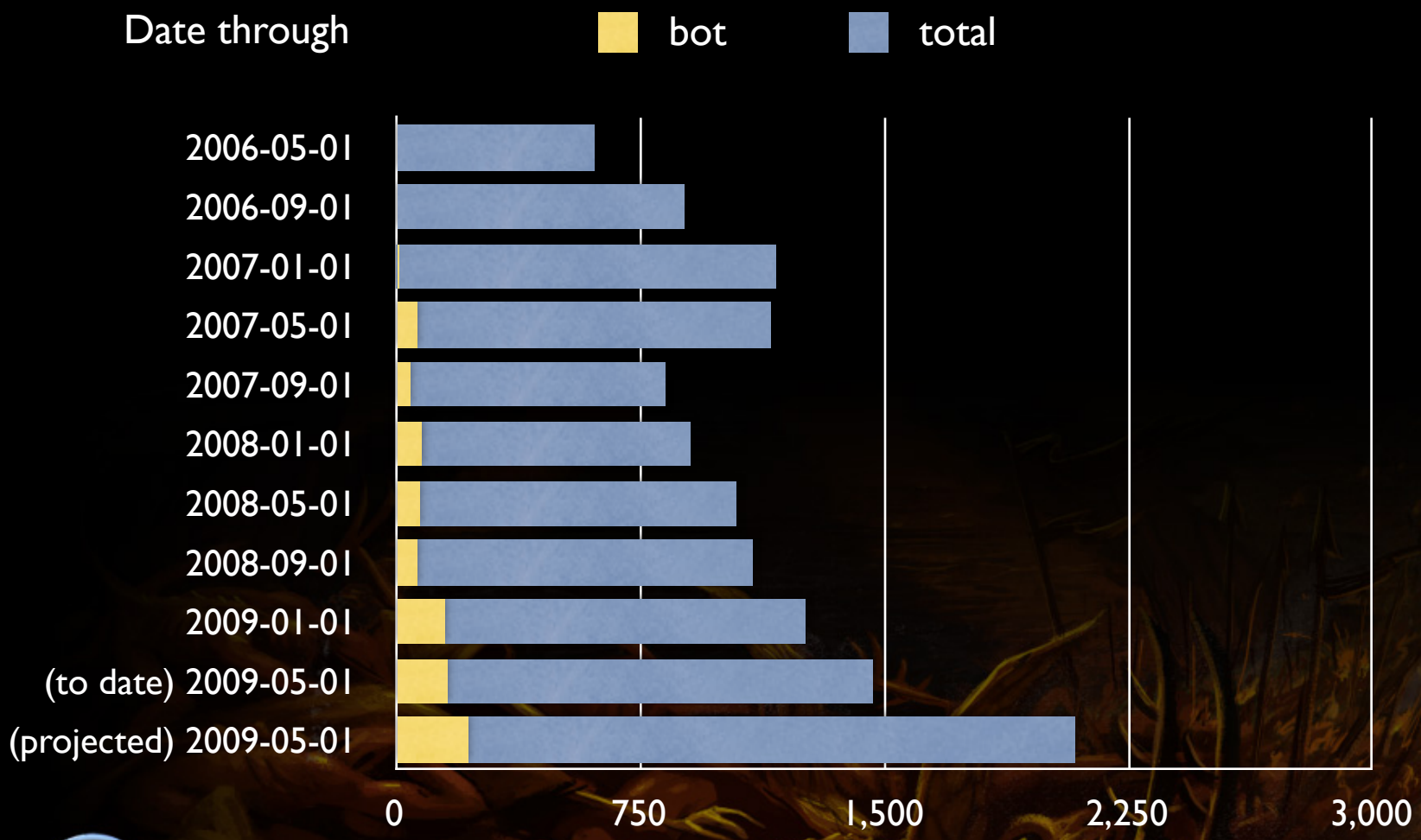
- Write a series of balance “tests”
- Throw permutations of unit groups at each other
- Print out results in a structured fashion
- Examined by a human for unexpected results

Use by artists

- They don’t run it themselves…
- …but they do see it running
- See parts of the game they normally wouldn’t
- Notice things that don’t look right

Analyšis

Number of bugs found



Number of bugs found

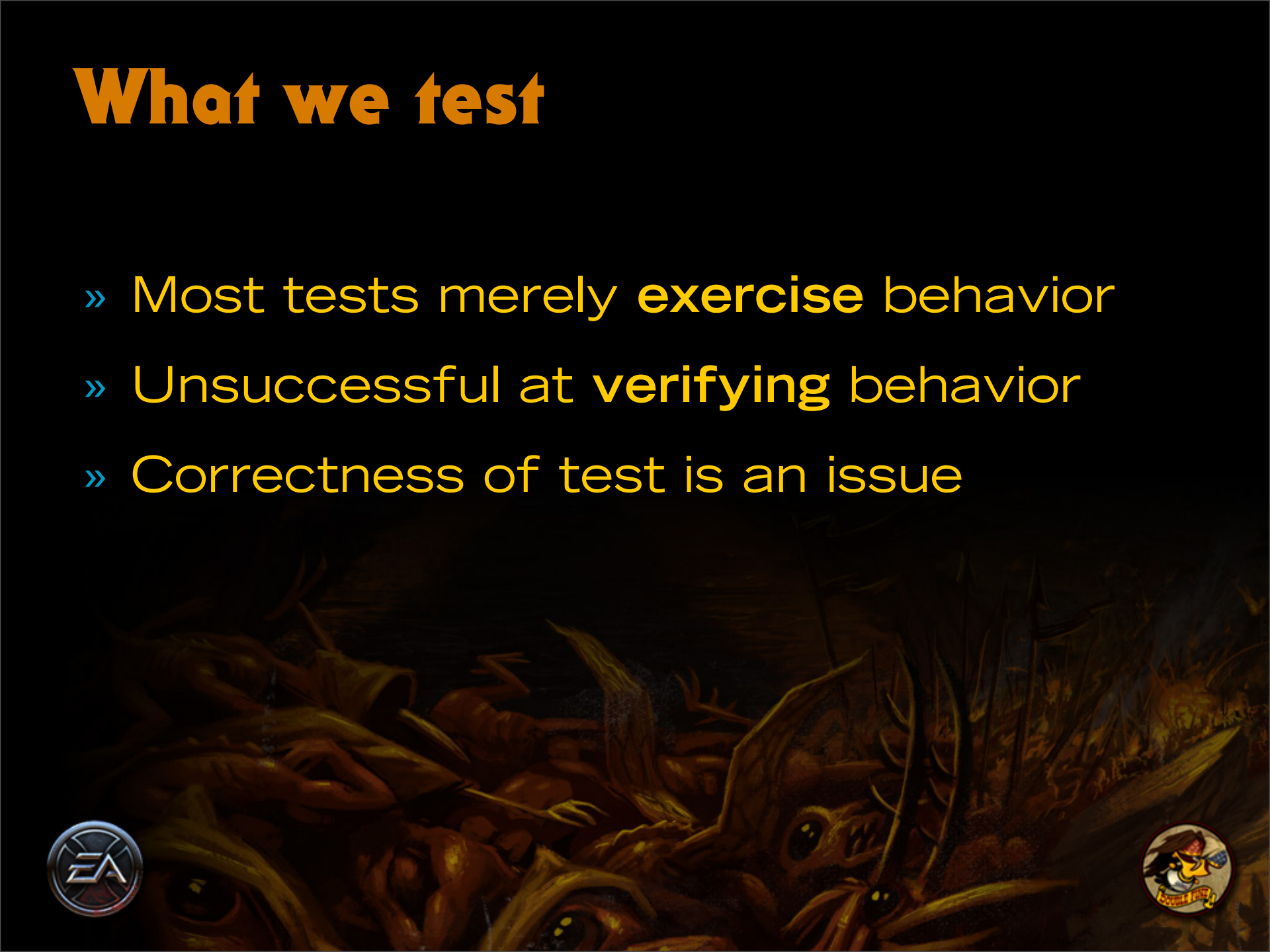
- Raw bug count undersells RoBert
- Query didn't catch all RoBert bugs
- Not all problems found get entered

Types of bugs found

- Almost all crashes and asserts
- Middleware bugs
- Logic bugs manifest as “Bot stuck in mission” failures
- Complementary to bugs found by human testers

What we test

- Most tests merely **exercise** behavior
- Unsuccessful at **verifying** behavior
- Correctness of test is an issue



What we don't test

- No testing of visuals
- Limited testing of performance
- Specific behaviors, game logic

Problems and future work

- Big tests can take a long time to complete
- Still a lot of human-required work
- May be guiding us to non-optimal solutions
- Bot cheats a lot

Our takeaway

- Doesn't replace a test team
- Does take tedious work off their plate
- Hillclimbing development strategy worked well
- Very curious what others are doing!

Questions?

dubois@doublefine.com

Fill out forms!