FINDING THE FUN

Damián Isla,
Moonshot Games
I DON’T KNOW WHAT I’M TALKING ABOUT
Finding the Fun

Fun is hard.

Fun lies in the gestalt.

- many interlocking systems
- the pleasingness of the art (the “appeal”)
- the “pop” of the sound
- how the controller (or phone, or tablet, or whatever) feels in your hand.
Finding the Fun: 3eC
Finding the Fun

Two Methods

1. Genius inspiration from the Heavens
2. Iteration, iteration, iteration.

“Inspiration is for amateurs – the rest of us just show up and get to work.”

- Chuck Close
Fun = Iteration

Fun is proportional to iteration count

So two approaches
  - increase total production time
  - increase iteration cycle speed
Finding the Fun: For Engineers

Here’s the hard truth.

Finding the fun is not your job.

What is:

- great tools
- fast import pipelines
- fast load-times
- stable code

Your client is not the player. It’s the design and art teams.
Designers

The hardest job in the world

- All the responsibility, none of the authority
- Everyone thinks they’re good at design
- Has to understand how everything impacts everything else.
- As much (or more) team leadership as it is creative inspiration
Agenda

- 3 Production Practices for Fun-Finding
- Technology for Fun-Finding
3 PRODUCTION PRACTICES FOR FUN-FINDING
1) Prototyping

1. Find a fun core mechanic
2. Add features to a larger game

Tends to be a certain amount of scope, complexity and risk required to warrant prototyping.

Prototyping = risk mitigation.
Answering Defined Questions

Prototypes are defined by a series of questions that need answering
- “can we do this?”
- “SHOULD we do this?”
- “what should reload delay on the shotgun be?”
- etc.

Answer the questions within the allotted time.

Once the question are answered move on.
- Does feature go into production?
- Does feature go nowhere?
- Does a new prototype suggest itself?
Prototyping is NOT about making it fun.

It *might* be about answering the question “Will it ONE DAY be fun?”

Fun comes requires polish
- visuals
- sound
- animation
- rumble

Sometimes it won’t be fun until everything is in place.
Sometimes it won’t be fun until the very last minute.
- BTW: This is terrifying.
Strike Teams

The Flip-side: if it is TOO rough, then you probably won’t learn anything.

Hence, Strike Teams

- Micro-studio to prototype high-risk features
- e.g. 1 designer, 1 engineer, 1 artist, 1 animator
- Typically these are cross-disciplinary features
- Otherwise they wouldn’t be hard or risky.
Physics

• Of the Scarab itself
• Of the things on it

AI

• Scarab Behavior
• Scarab Perception
• Perception of Riders
• Spatial Evaluation of Riders
• Etc.

Model

Design

• How do you Mount it?
• How do you Kill it?
• Why should it exist?

Production

Concept Art

Animation

Gameplay

Testing

Rendering / Visibility
2) Gray-Boxing
Gray-Boxing

- Fluid enough to iterate quickly
- Representative enough to prove the fun

→ Prototyping at the content level
3) Enforce Continuity with Production

Continuity between prototyping and production.
Continuity between gray-box and production.

The Important Part:
The Content should remain playable THROUGHOUT production.

(the production train needs it that way)
Continuity with Production

3 Things:

- Modularity helps
- Skinning
- Don’t over-engineer

Make it as simple as possible, but no simpler.
Make it as real as necessary, but no realer.
TECHNOLOGY FOR FUN-FINDING
Fun-Finding Tech

- Tools & Editors
  - Always seems like a pain in the ass. Always worth it.

- Play-in-editor

- Fast import of source content
  - and hot-loading / refreshing is MANDATORY.

- Fast boot / reset

- Should NEVER need to shut down the game.
  - context switches are mentally expensive!

- Think about usability!
  - the Designers and Artists are your users
Fun-Finding Tech

The two most important workflow / productivity technologies ever invented:

Undo / Redo

Copy / paste

(Hard to start with, very VERY hard to retro-fit.)
Scripting

Sometimes code is the best data.
“Data-Driven” Weapons System

Projectile weapons had ~ 30 parameters
  - ~15 used only for shotgun
  - e.g. the reload delay

So this was a parameter-driven system where
  - parameter-values define a complex state machine
  - certain combinations of parameters just didn’t work.

It really just wanted to be a chunk of code
Scripting is King

[Can be] Clean and explicit.

Crucial for iteration speed
  - no recompiling / restarting the game

And by the way:

Absolutely every designer on earth should know how to code (script).
Sputnik: Pervasive Scripting

The Goals

- Everything you can do in C++, you can do in Script
- To Create a Continuity between Prototyping and Production
Scripting as Architecture

LUA
- Compiled to bytecode
- Runtime recompilation
- Highly embeddable
- Highly customizable
- Dynamic typing

Hurray! 2x productivity

Kill me now
Component Soup

Perception/Ray-casting

Behavior

Target selection

A*

Animation control

Animation playback

Physics

Squad behavior

Random Script
Component Soup

- Perception/Ray-casting
- Target selection
- Behavior
- A*
- Animation control
- Animation playback
- Physics
- Squad behavior
- Random Script
Component Soup

- Perception/Ray-casting
- Target selection
- Animation script
- Locomotion control
- Animation playback
- A*

- Squad behavior
- Random Script

- Behavior

- Physics
The Result

Systems that don’t exist in Fallen Frontier:

- Health / death / damage
- Weapons / items / equipment / powers
- AI
- Behavior
- HUD / UI / Menus
- Missions / Levels / Objectives
The Rigor Onion

- Clear
- Maintainable
- Game-specific
- Iterable

Loose

- Performant, Game-agnostic
- Exquisitely Engineered
- C++
IN CONCLUSION
Fun = Iteration

3 Production Practices for Fun-Finding
- Prototyping
- Gray-boxing
- Enforce Continuity with Production Technology for Fun-Finding
- Fast import
- NEVER reboot
- Undo/redo
- Copy/paste
- Scripting
Thanks!