無料で始める！
「龍が如く」を面白くするための
高速デバッグログ分析と自動化
（JaSST’18 Tokyo 完全版）

株式会社セガゲームス
第１CSスタジオ
アドバンスト・テクノロジー開発チーム
（ドラゴンエンジン開発チーム）
阪上 直樹

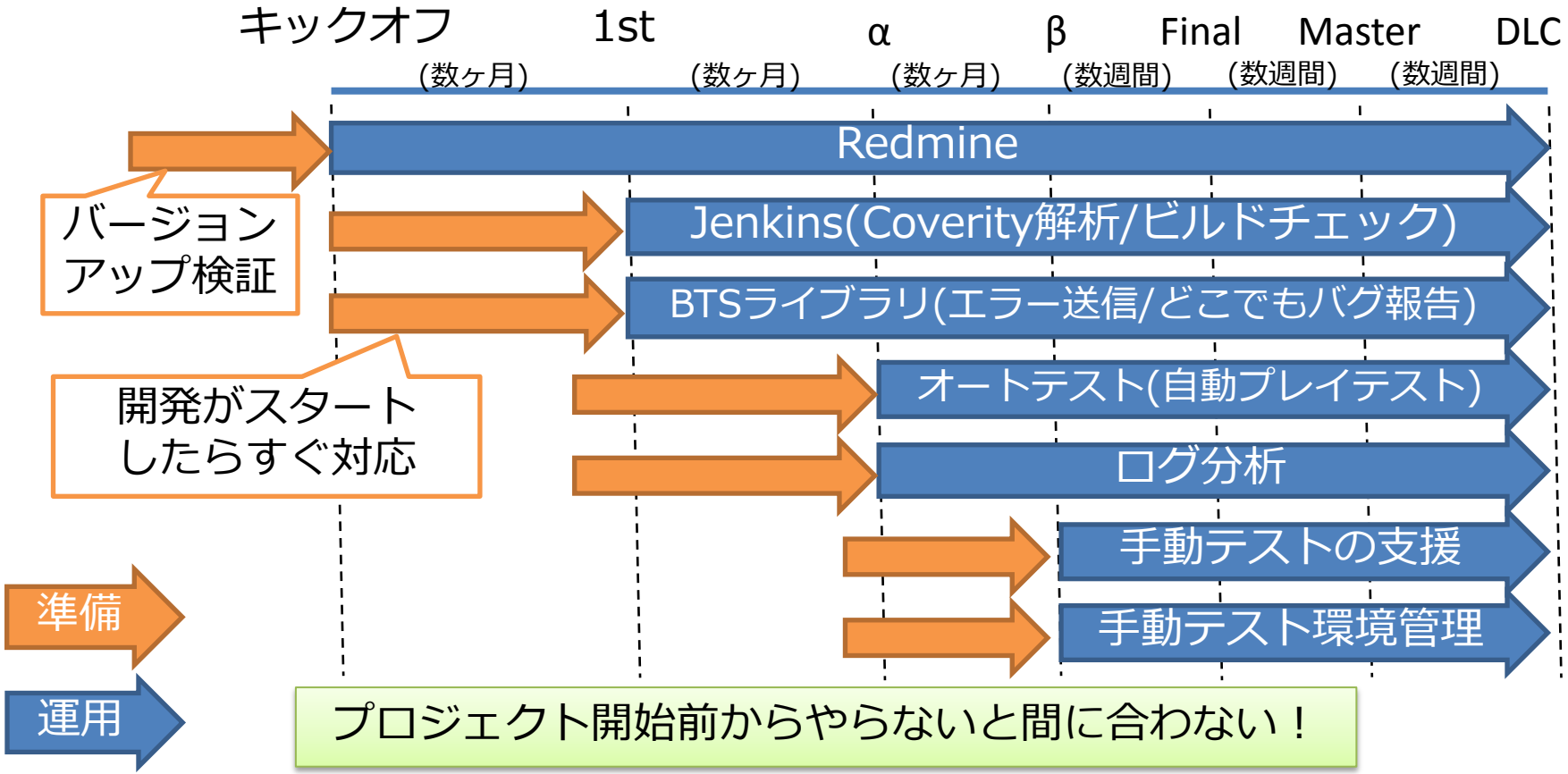
Japan Symposium on Software Testing JaSST

自己紹介

- 「龍が如くスタジオ」専属QAエンジニア
  - 元々ゲームプログラマ
  - プログラムをガリガリ書いて、開発環境を幅広くサポート
  - アプリ側にも手を入れる
    - アプリに書いたソースは「#if DEBUG」内なので、製品には1行も入らない！

ゲームプログラムを直接いじるQAエンジニアリング ＝ Embedded QA

龍が如くスタジオ専属QAエンジニアの一年



本セッションのゲームタイトル

©SEGA

龍が如く６ 命の詩。
2016年12月8日発売
(略称：龍6)

©SEGA

龍が如く 極2
2017年12月7日発売
(略称：極2)

©SEGA ©武論尊・原哲夫/NSP 1983 版権許諾証GA-217

北斗が如く
2018年3月8日発売
（好評発売中です！）

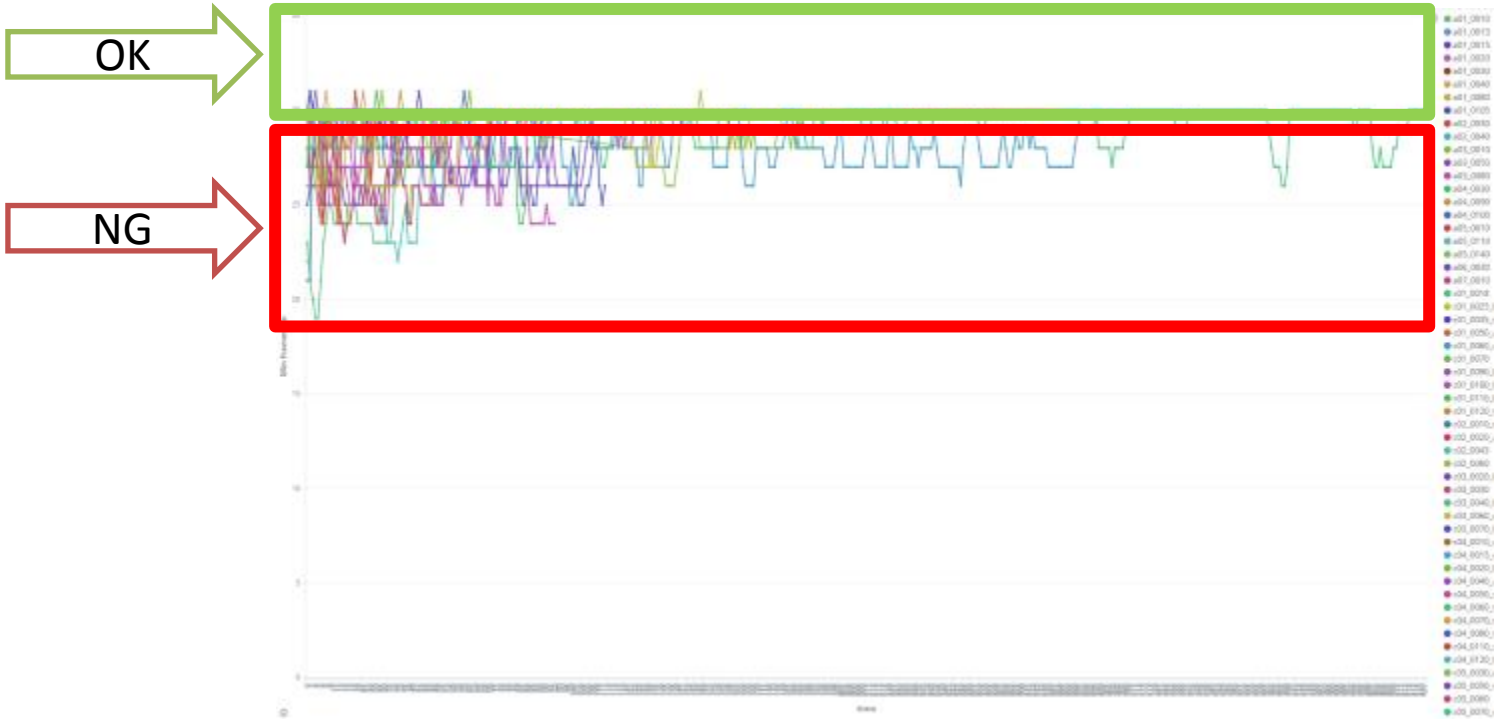
「龍が如く６ 命の詩。」のログ分析

VRAMヒートマップ

桐生が死んだマップ

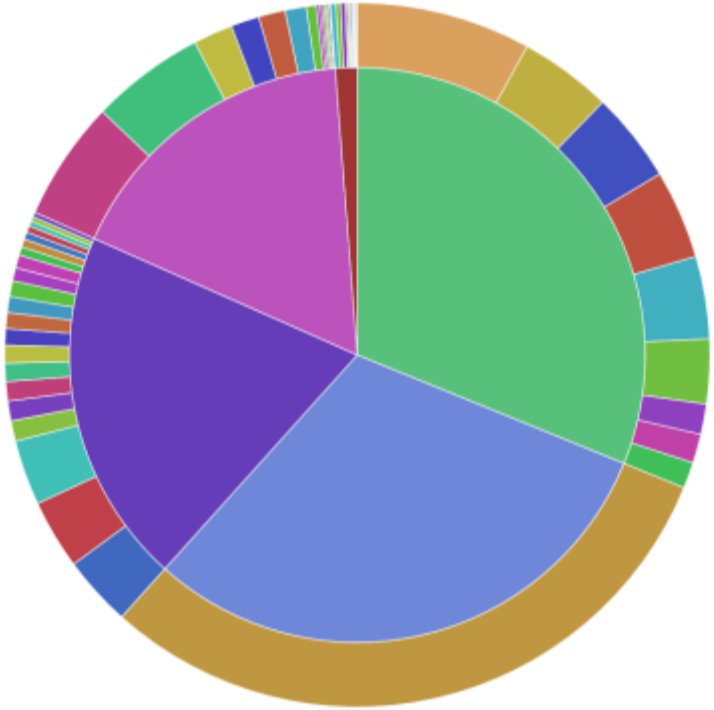
「龍が如く 極2」のログ分析

フレームレート(処理落ち)分析



「北斗が如く」のログ分析

アイテムドロップ率



コリジョン抜け

©SEGA ©武論尊・原哲夫/NSP 1983 版権許諾証GA-217

本セッションの概要

1. 高速デバッグログ分析の環境構築
2. 龍が如くスタジオの自動化環境
3. デバッグログ分析と自動化の連携

目次

1. 高速デバッグログ分析の環境構築
   - デバッグログ分析とは
   - 極での導入実験
   - オープンソースの組み合わせによるサーバ構成
   - 「どこでもログ送信」とデータ構成
2. 龍が如く開発チームの自動化環境
3. デバッグログ分析と自動化の連携

デバッグログ分析とは

- 本セッションのデバッグログの定義
  - 開発期に発生するログ
    - 製品版には搭載されていない
  - printf出力したものやプレイログ(移動履歴やダメージ量)が含まれる
- 開発期のデバッグログ分析の話
  - 課金の分析の話は出てきません。
- ゲームの品質を上げたい、面白くしたいというQAと開発者のためのログ分析

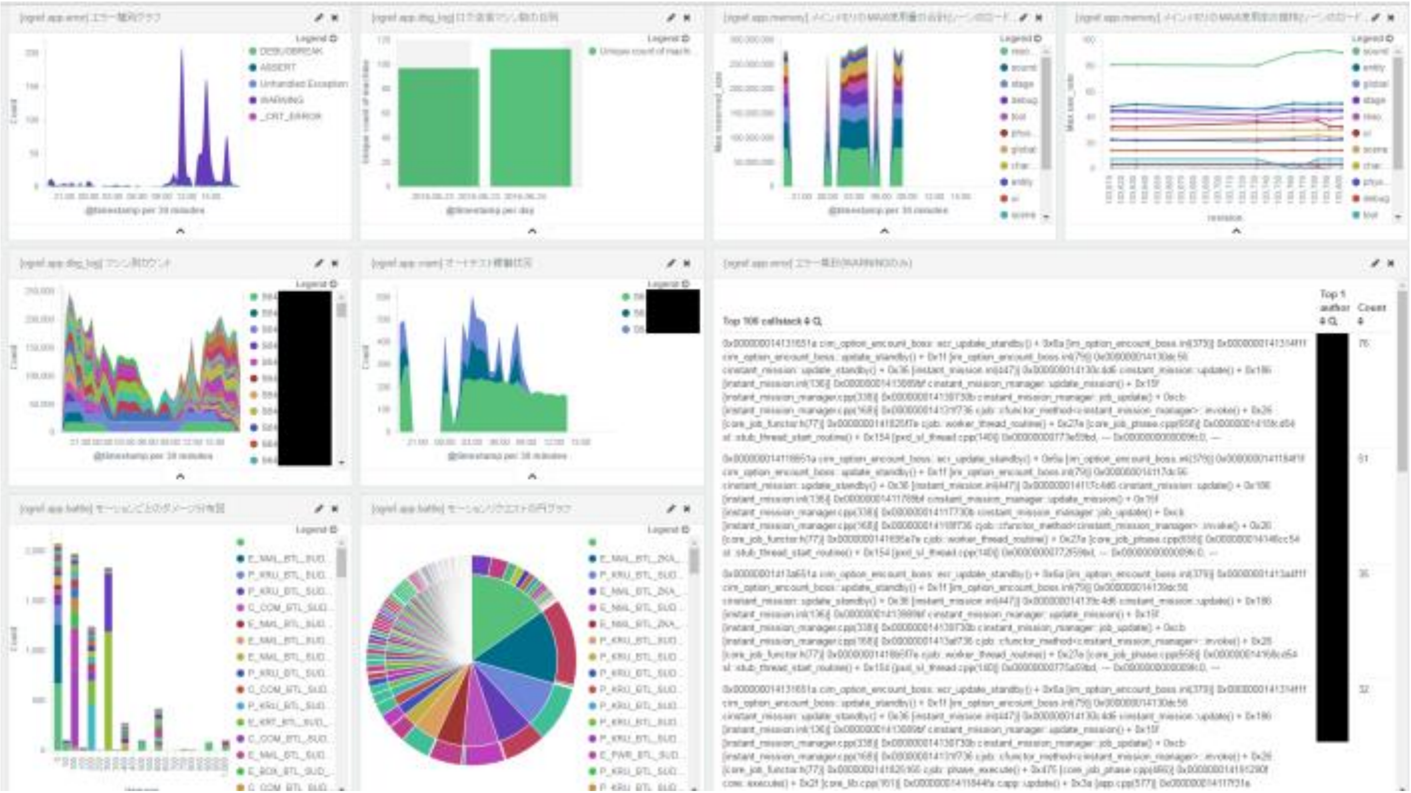
なぜデバッグログ分析が必要？

- 個々のログファイルを検索するのが面倒なので、一括で検索できない？
- 強制終了するほどではない警告レベルの問題発生を確認したい
- 実行頻度が低い関数の動作実績を知りたい
  - どういう引数が渡ってくるのか
  - 結果戻り値がどうなったか

面白くするためのデバッグログ分析

- 開発中のデバッグログを収集
- エラー情報以外にゲームの進行度、経験値などを送信
- 分析したら、もっと客観的にゲームバランスを調整できるかも？

kibana



「龍が如く 極」で導入実験

- ファイル出力していたデバッグログ(printf)の書き込みを監視して、ログサーバに送信する実験
- 用意した仮想環境のスペックが足らず、送信データをさばき切れない、検索が遅いなど、問題が多発
  - ログ収集サーバは、CPUとIOがボトルネック
  - ログデータサーバは、メモリ不足がボトルネック
- デバッグログ(printf)を検索できるだけでは、データの加工が大変で、分析できることも限られていた

デバッグログ(printf)の例

```
[!DIV                ] p_auth_playが消える予定なのでリソースの処理はしない
[!ina                ] cscene_info_handover construct
[!TIMELINE           ] 不明なtimeline_clock()です。dispose_fileの更新で治る場合があります。
[!SCENE              ] scene chgreq 302 empty
[!SAVEDATA           ] load : system
[!SOUND              ]>ボリュームプリセットが変わりました  → common
[!CCC                ] refresh_ec_ccc_tree(0)
[!SOUND              ]>ボリュームプリセットが変わりました common → system_def
[!SAVEDATA           ] check_series_save_data : system
[!SCENE              ] scene chgreq 008 system_selector
[!TIMELINE           ] set_timeline() : c01 [1章開始前] -> [1章終了]
[!TIMELINE           ] set_timeline() : title [開始前] -> [ゲーム開始後]
[!ina                ] cscene_info_handover construct
[!db                 ] load(593,[redacted]/media/data/db/ja/particle_oneshot_se.bin)
[!PARTICLE           ] load par_cmn
[!db                 ] load(85,[redacted]/media/data/db/ja/sound_se_name_table.bin)
[!ui                 ]!scene loaded.[smartphone]
[!timeline change    ]>シーン変更で切り替わりました padv
[!TIMELINE           ] set_timeline() : padv [開始前] -> [padv]
[!DISPOSE_MANAGER    ] [redacted]¥src¥dispose¥dispose_manager.inl(442): timeline_change: load_entity
[!DISPOSE_MANAGER    ] [redacted]¥src¥scene¥lua¥scene_lua_command.cpp(2953): timeline_change: load_entity
[!SOUND              ]>ボリュームプリセットが変わりました  → common
```

人が読むのに適しているが、
機械的にデータ処理しにくい構造

龍６でのログデータの流れ

debug_printf

SEND_LOG_XXX

ファイル出力
dbg_log_XXX.txt
log_send_XXX.txt

どこでも
ログ送信

LogSender.exe

ファイルを監視して
１行ずつ送信
(JSON形式)

Win/PS4
両対応

ログ収集サーバ

fluentd

登録

データベース

elasticsearch

分析

グラフ化

kibana

オープンソース(OSS)の組み合わせ

- Fluentd (https://www.fluentd.org/)
  - ログ収集・加工サーバ
- Elasticsearch (https://www.elastic.co/jp/products/elasticsearch)
  - 全文検索エンジンで使われているNoSQLデータベース
  - JSON形式のデータを蓄積
  - スキーマレスでもOK（DB経験がない人向き！）
- Kibana (https://www.elastic.co/jp/products/kibana)
  - グラフ作成・可視化ツール
  - Elasticsearchのプラグイン(最新の5.Xでは単体動作可能)
  - DBやSQLに詳しくない人でもグラフが作れる
- すべて無料(OSS)、すべてWindows版あり

龍６でのサーバ構成

- サーバ2台構成
  - 開発用のHP Z420を流用
    - Windows7 64bitで運用
    - PC-A：Fluentd/Elasticsearch/Kibana
    - PC-B：Elasticsearchのスレーブ
  - Elasticsearch
    - 情報の多い1.7系を使用
    - ログのバックアップと負荷分散になる設定 (shard 2、replica 1)
    - Java 64bitで16GBを割り当て
- 詳しいセットアップ方法(Windows版)
  - Fluentd (https://docs.fluentd.org/v1.0/articles/install-by-msi)
  - Elasticsearch (https://www.elastic.co/guide/en/elasticsearch/reference/master/zip-windows.html)
  - Kibana (https://www.elastic.co/guide/en/kibana/master/windows.html)
  - データ分析基盤構築入門[Fluentd、Elasticsearch、Kibanaによるログ収集と可視化]（技術評論社）

LogSender

- ログファイルを監視して差分をFluentdに送信する常駐アプリを作成
- fluent-logger-csharpを使ったC#製
- ゲーム側はログファイルに書き込むだけなので、ゲーム側の実装やプラットホーム依存が最小限

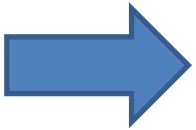
どこでもログ送信

キー・バリュー形式（マクロ→JSON）

```
SEND_LOG_SAKAUE_COLLISION(
  "fall",
  "桐生が奈落へ落ちた",
  LOG_BOOL("is_player", true),
  LOG_STR("kind", "fall"));
```

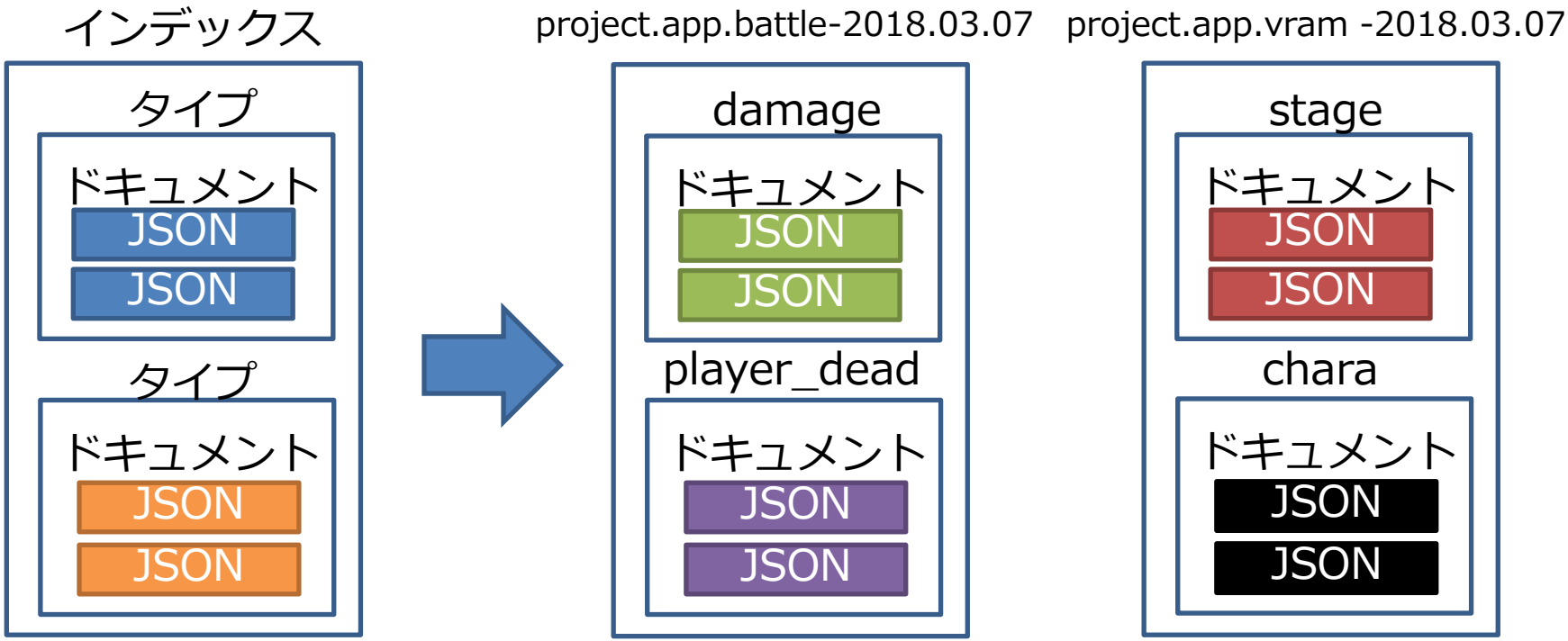
キー・バリューの数は任意に増やせる

LogSender.exe



```
{
    "author": "【AT開発】阪上 直樹",
    "category": "COLLISION",
    "is_player": true,
    "kind": "fall",
    "message": "桐生が奈落へ落ちた",
    "player_pos": [
        -170.705993652344,
        -81.4550018310547,
        -64.0670013427734,
        -20641
    ],
    "revision": 140600,
    "type": "fall",
    "machine": "SAKAUE-PC",
    "@timestamp": "2016-10-17T02:18:41.630"
}
```

Elasticsearchのデータ構造(龍６の場合)

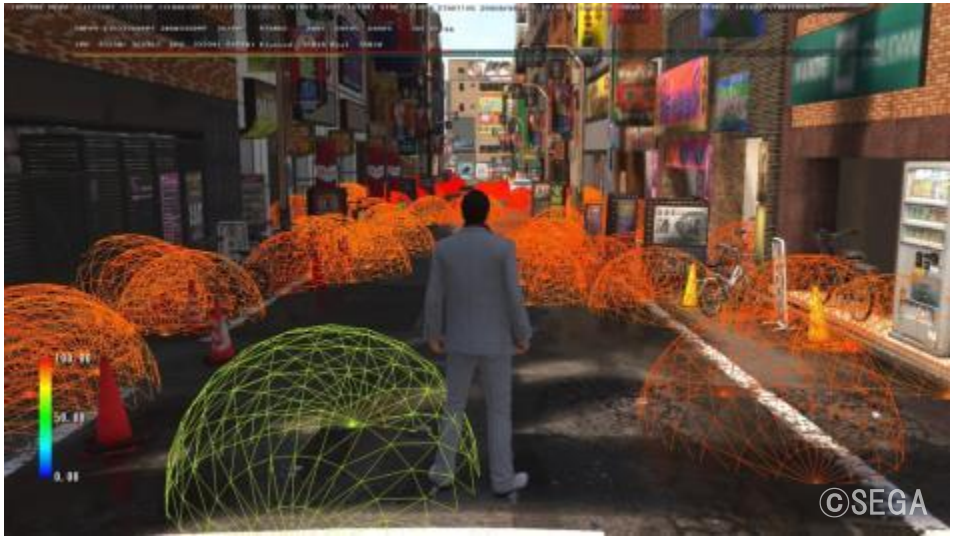


※typeは将来的になくなります。

オリジナルのログ分析可視化機能

どこでもログ分析(3D版)を準備

ElasticsearchからC#アプリ経由でゲーム実行中にその場で分析可能！



色々準備した結果

使われなかった…（涙）

※αの時点

使われない原因

- 従来のデバッグログ(printf)は、検索して警告等が出ているか確認できるが、それ以上の分析は難しい
- 「どこでもログ送信」が、なかなかアプリ側に実装してもらえない
- 「どこでもログ分析」は、見た目は面白いけど、起動に時間がかかるし、条件設定が面倒
- データの信頼性が低い
  - デバッグ機能でプレイヤーをワープさせて落下しても、エラーじゃない
  - プレイヤーが想定より強いけど、これはデバッグ機能でレベルを上げたからじゃないのか？という疑念
- そもそも何を分析したらいいのか分からない

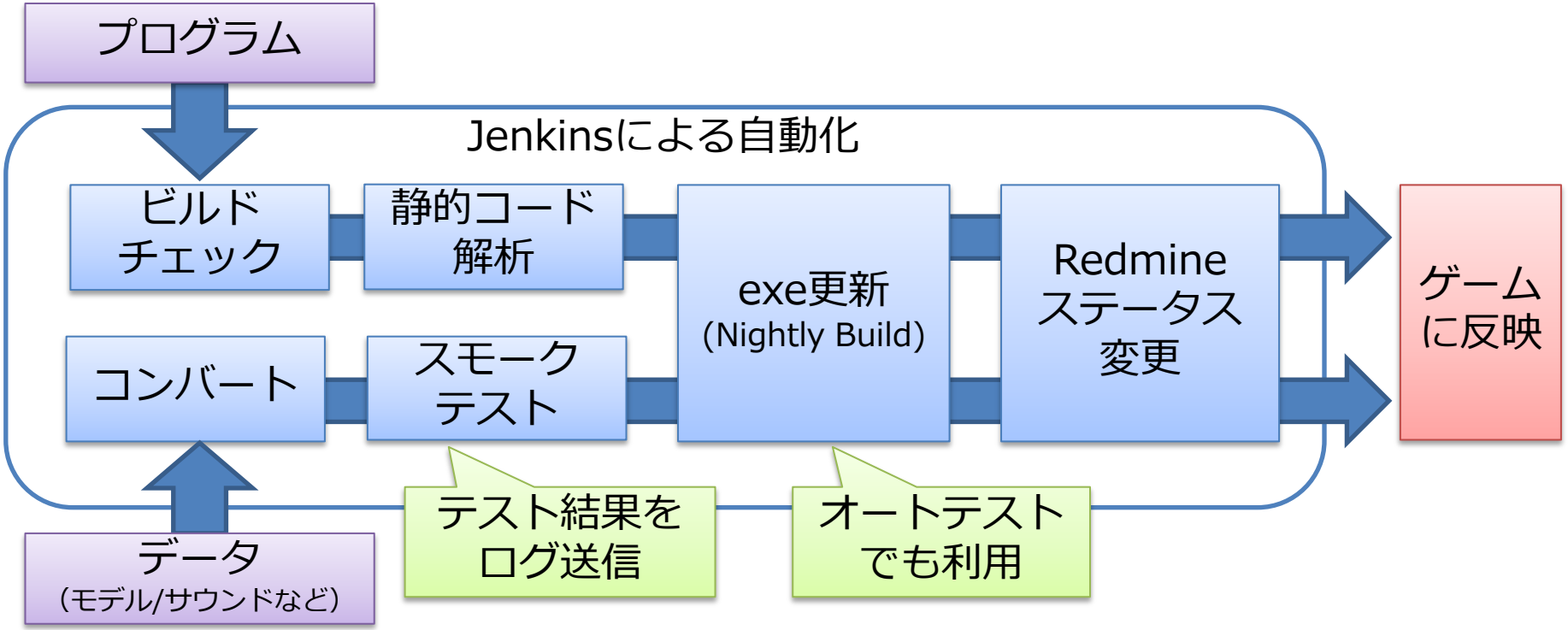
使ってもらうために

- 「どこでもログ送信」を組み込んでもらえない
  - プログラマに聞いてQAエンジニア(私)が直接実装
- ログ分析ツールの起動が面倒
  - URLにアクセスするだけですぐ閲覧できる環境を準備
- データの信頼性の向上
  - 手動テストのログをフィルタできるようにis_checkフラグを追加
  - 自動化のログを活用できないか

目次

1. 高速デバッグログ分析の環境構築
2. 龍が如くスタジオの自動化環境
   - Jenkins
   - 独自の自動化ツール
     - BTSライブラリ
     - オートテスト
3. デバッグログ分析と自動化の連携

Jenkinsの活用ワークフロー



詳しくはJaSST'16 Tokyoの講演資料を見てね！！
http://jasst.jp/symposium/jasst16tokyo/pdf/E2.pdf

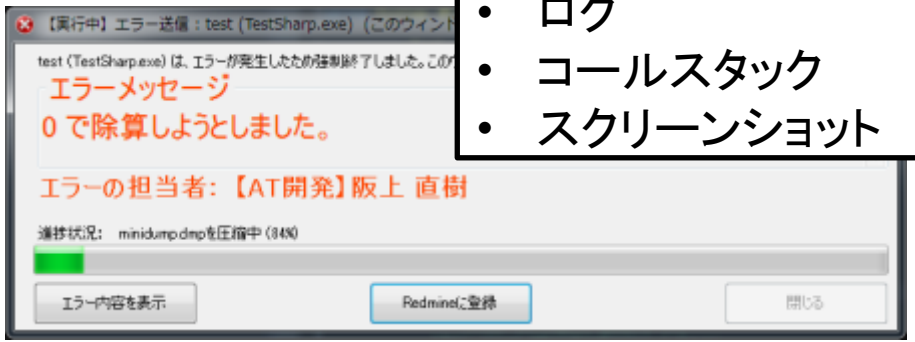
独自の自動化ツール

- BTSライブラリ
  - エラー送信(クラッシュレポートの自動送信)
  - どこでもバグ報告(バグ報告の自動入力)
- オートテスト(自動プレイテスト)
  - 仕組み（ランダム・パス移動）
  - エラー検出結果

エラー送信

龍が如くスタジオのクラッシュレポート機能

ゲームやツール
実行中に
例外発生！



- ダンプ
- ログ
- コールスタック
- スクリーンショット

アップロード → ネットワークドライブ（1エラー数百MB～数GB）

メール送信 →
- ダンプ表示batのURL
- コールスタック
- リビジョン

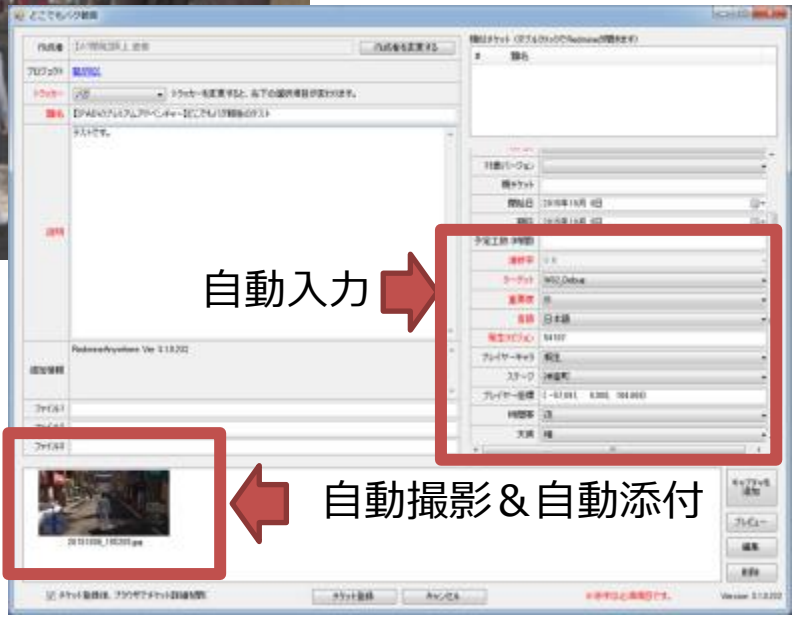
バグ報告 → バグ管理システム (Redmine)

ログ送信 → Fluentdサーバ

どこでもバグ報告

バグ報告に必要な項目の自動入力機能

©SEGA



独自の自動化ツール

- BTSライブラリ
  - エラー送信(クラッシュレポートの自動送信)
  - どこでもバグ報告(バグ報告の自動入力)
- オートテスト(自動プレイテスト)
  - 仕組み（ランダム・パス移動）
  - エラー検出結果

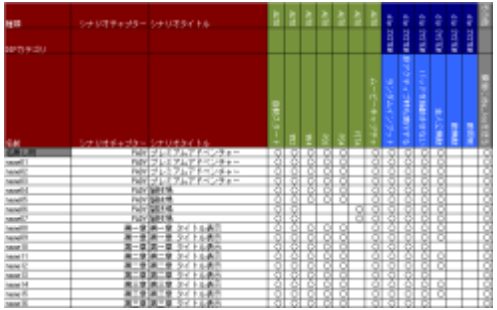
オートテスト

- 龍が如くスタジオの自動プレイテスト
  - 夜間の開発環境を活用して、各PCごとに設定された条件でゲームを起動してエラーを検出する仕組み
  - オートインプット(自動パッド入力)
    - ゲームパッドの擬似入力以上のことをしない
    - ランダム（異常系テスト）
    - パス移動（正常系テスト）**NEW!!**

オートテストの仕組み

帰宅前に各PCで
オートテストクライアントを実行

設定ファイル(Excel)から
iniを生成



最新ビルドを取得

ゲームを自動起動
(iniファイルのシナリオ/条件)

©SEGA

エラー送信

ランダム移動の仕組み

- ゲーム内容に合ったガチャ押し
  - バトル中は必殺技が出やすいボタンを連打
  - UI起動中は、上下左右/○×連打
  - コリジョン抜けを見つける動き
    - 一定時間同じ方向に移動を続けて壁にぶつかる
    - 半径数メートルの範囲内で動き回る

パス移動の仕組み

ゲームプレイ中の入力を
パスリストに変換

オートテストでパスを再生

シナリオ切り替え時に
直前のシナリオの
クリアパスを送信

シナリオ切り替え時に
一番新しいリビジョン
のパスを取得

オートテストでクリア
したパスも送信

Elasticsearch

JSON形式で
パスを保存

パス作成(ゲームプレイ中に自動生成)

パス再生(オートテスト)

オートテストのパス移動導入の結果（龍6）

メインシナリオの全自動クリアを達成！

※複数マシンを使用

パスも自動生成へ

- プレイするだけでパスを作成できるが、それも面倒なので完全自動生成を目論む
  - オートテストで次のパスに進めないときに試行錯誤する処理を追加

    →パスが長くなるだけで思うように補正されず…
  - ゲームをプレイしている全員のパスを送信

    →クリアを目指してプレイしていないことが多くてダメ…

メインストーリーをクリアしまくって
自力でパスを作成・修正した！

龍が如く 極2/北斗が如くでの改善点

- 章ごとに分担してパスを作成・修正した
- 基本的にはゲームを進めるだけなので、プログラマじゃなくてもパス作成が可能
- 一部強引に突破したが、極2と北斗が如くでも、メインシナリオの全クリアを達成

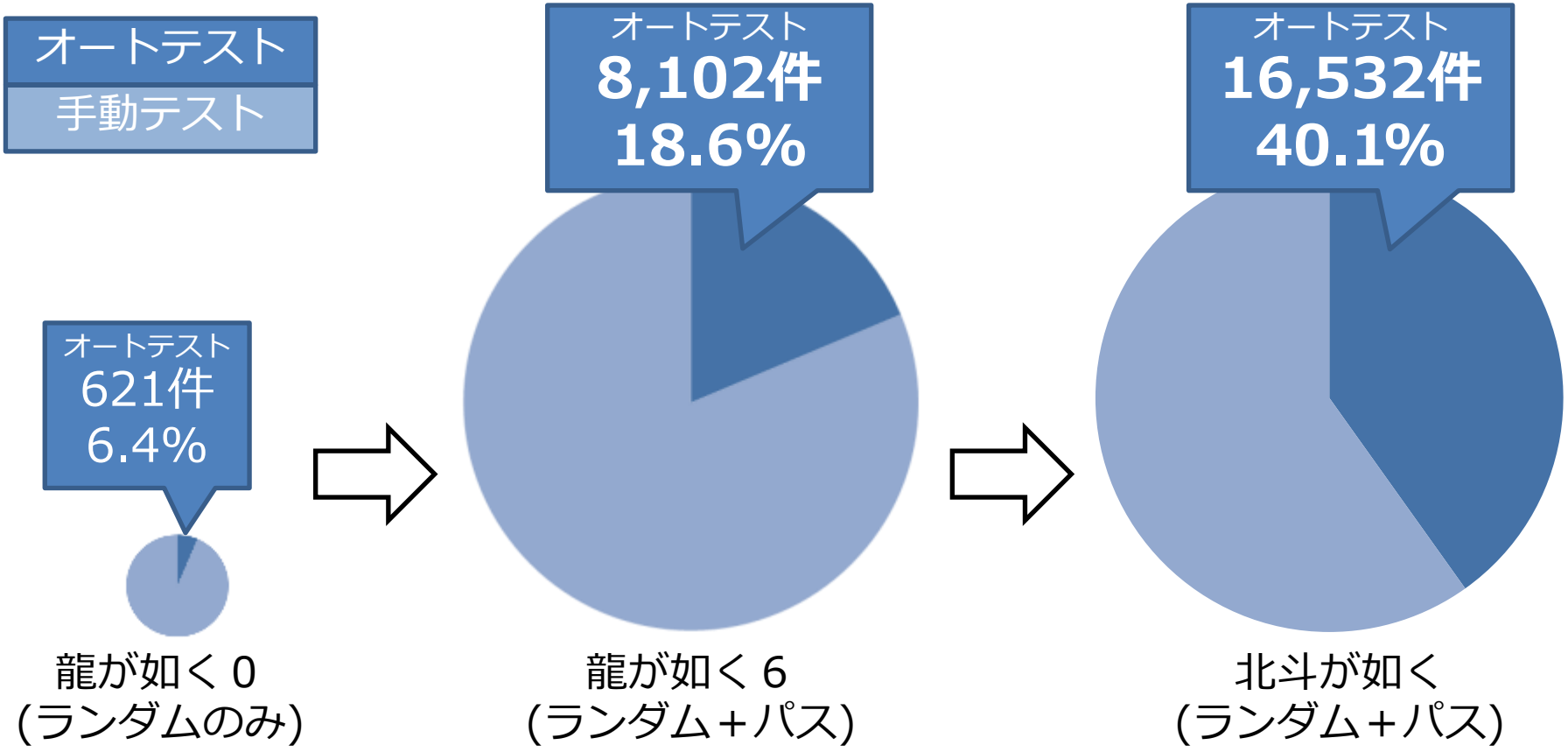
パス再生（北斗が如く・体験版）

会話中は○連打

バトル中は敵を倒す
操作に自動切換

©SEGA ©武論尊・原哲夫/NSP 1983 版権許諾証GA-217

オートテストのエラー検出数と割合



オートテストの運用結果

		龍0	龍6	極2	北斗が如く
規模	オートテスト稼動数	100台未満	150台	180台	210台
	24時間稼働数	0台	41台	30台	45台
エラー送信	総数	9,664件	43,369件	20,675件	41,190件
	オートテスト	621件	8,102件	8,645件	16,532件
	オートテスト率	6.4%	18.6%	41.8%	40.1%
	コリジョン抜け	-	1,910件	145件	6,435件
価値	のべ稼働日数※	-	10,263日	9,770日	14,351日
	時給1000円換算	-	8,210万円	7,816万円	11,481万円

オートテストのポテンシャル(潜在価値)

※1日8時間換算

オートテストの有用性と課題

- 有用性
  - 低コスト（開発機材＋電気代のみ）
  - デバッグ期間の短縮（スケールアップが容易）
  - コリジョン抜けの検出（単純作業からの解放）
  - 再帰テストとしての活用
  - 再現性の低いバグの検出と検証
- 課題
  - パス移動でのUI（買い物や選択肢）の突破率の向上
  - 各種ミニゲームのクリア

オートテストと手動テストは競合？

- オートテストが得意なところ
  - メインシナリオの最速動作チェック（再帰テスト）
  - 広いマップ上のコリジョン抜け検出
  - 組み合わせが多すぎて手動で確認できないもの
- 手動テストが得意なところ
  - バグかどうかの判断（見た目など）
  - ゲームバランスや操作性の指摘
  - 臨機応変なテスト（バグが出やすい場所を推測）

オートテストと手動テストのいいとこ取りが目標

デバッグログ分析に使える自動化

オートテストのログを
活用しよう！

目次

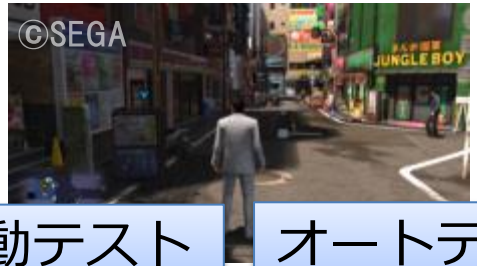
1. 高速デバッグログ分析の環境構築
2. 龍が如くスタジオの自動化環境
3. デバッグログ分析と自動化の連携
   - デバッグログ分析のワークフロー
   - 活用事例(エラー検出)
   - 活用事例(ゲームバランス調整)
   - 運用結果と課題

自動化とデバッグログ分析の連携

- オートテスト(自動プレイテスト)を使って信頼できるログを集める(is_autotest)
- 気軽に閲覧できる環境を整える
- 手動テストのログを自動分析して、ゲームバランス調整のワークフローに乗せる

デバッグログ分析活用ワークフロー

日ごと or リアルタイムで修正＆確認



手動テスト

オートテスト

ログ送信

Elasticsearch

自動集計

Jenkinsで1日1回

Kibanaで即確認

調整

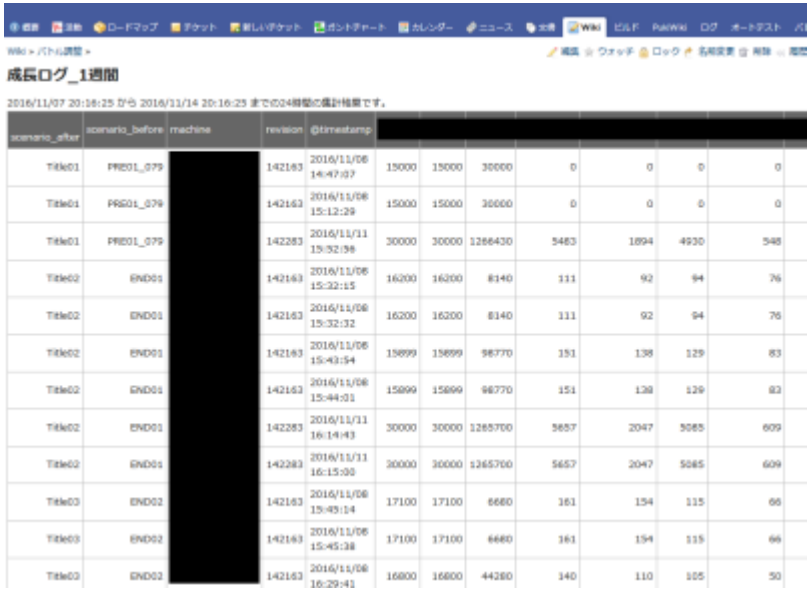
開発者

集計結果を確認

Redmine内のWikiに公開

RedmineのWikiで即時確認

Redmine APIで自動更新！すぐ確認できる！



成長ログ_1週間

2016/11/07 20:16:25 から 2016/11/14 20:16:25 までの24時間の統計結果です。

scenario_after	scenario_before	machine	revision	@timestamp							
Title01	PRE01_079	[redacted]	142163	2016/11/08 14:47:07	15000	15000	30000	0	0	0	0
Title01	PRE01_079		142163	2016/11/08 15:12:29	15000	15000	30000	0	0	0	0
Title01	PRE01_079		142283	2016/11/11 15:52:58	30000	30000	1266430	5483	1894	4930	548
Title02	END01		142163	2016/11/08 15:32:15	16200	16200	8140	111	92	94	76
Title02	END01		142163	2016/11/08 15:32:32	16200	16200	8140	111	92	94	76
Title02	END01		142163	2016/11/08 15:43:54	15899	15899	98770	151	138	129	83
Title02	END01		142163	2016/11/08 15:44:01	15899	15899	98770	151	138	129	83
Title02	END01		142283	2016/11/11 16:14:43	30000	30000	1265700	5657	2047	5085	609
Title02	END01		142283	2016/11/11 16:15:00	30000	30000	1265700	5657	2047	5085	609
Title03	END02		142163	2016/11/08 15:45:14	17100	17100	6660	161	154	115	66
Title03	END02		142163	2016/11/08 15:45:38	17100	17100	6660	161	154	115	66
Title03	END02		142163	2016/11/08 16:29:41	16800	16800	44280	140	110	105	50

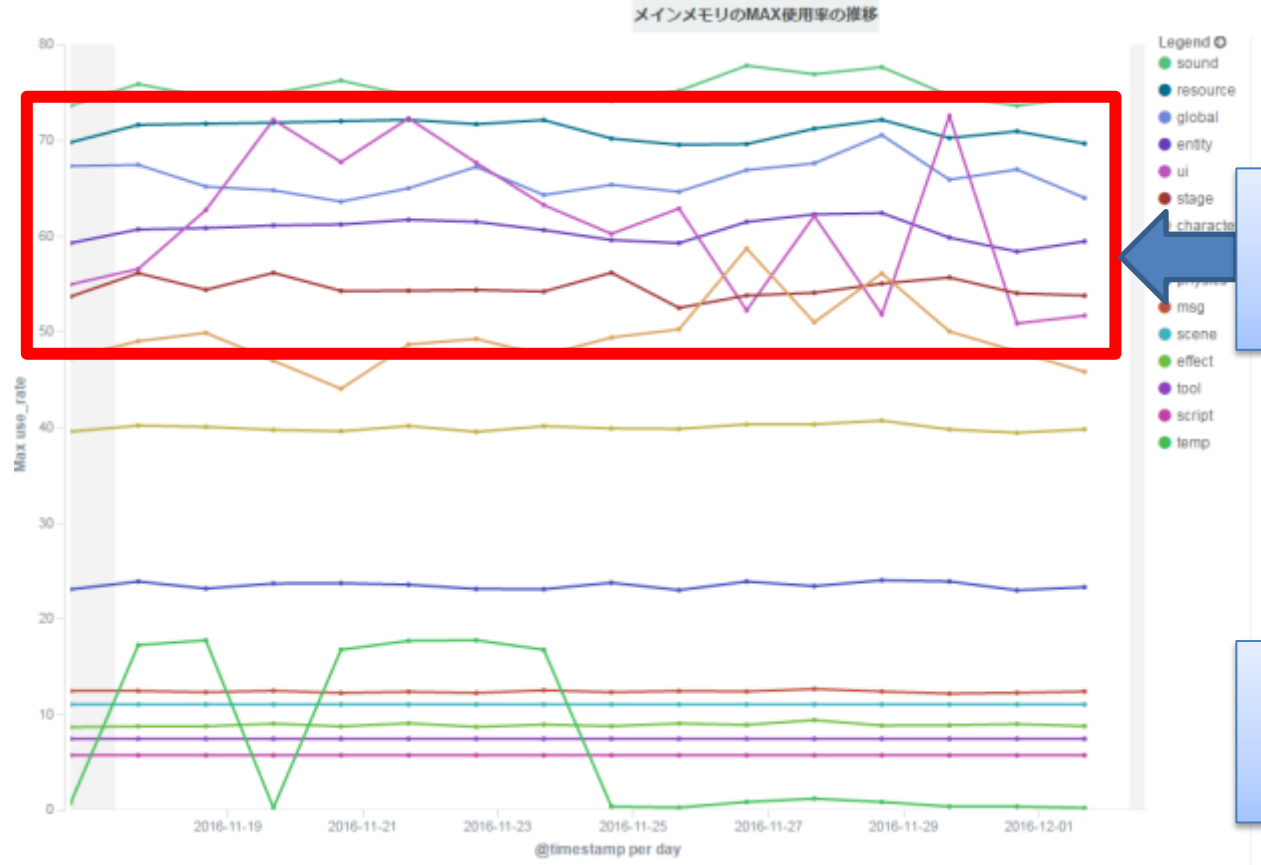
デバッグログ分析活用事例

- エラー検出
  - 主にオートテストのログを活用
- ゲームバランス調整
  - 主に手動テストのログを活用
  - 面白くするためのログ分析

デバッグログ分析の活用事例

- エラー検出
  - メモリ使用率（グラフ）
  - 演出シーンのフレームレート分析
  - メモリ使用率（ヒートマップ）
  - コリジョン抜け
  - オートテスト結果分析
- ゲームバランス調整
  - 桐生が死んだマップ
  - 成長ログ
  - 死亡回数と蓄積ダメージ
  - アイテムドロップ率

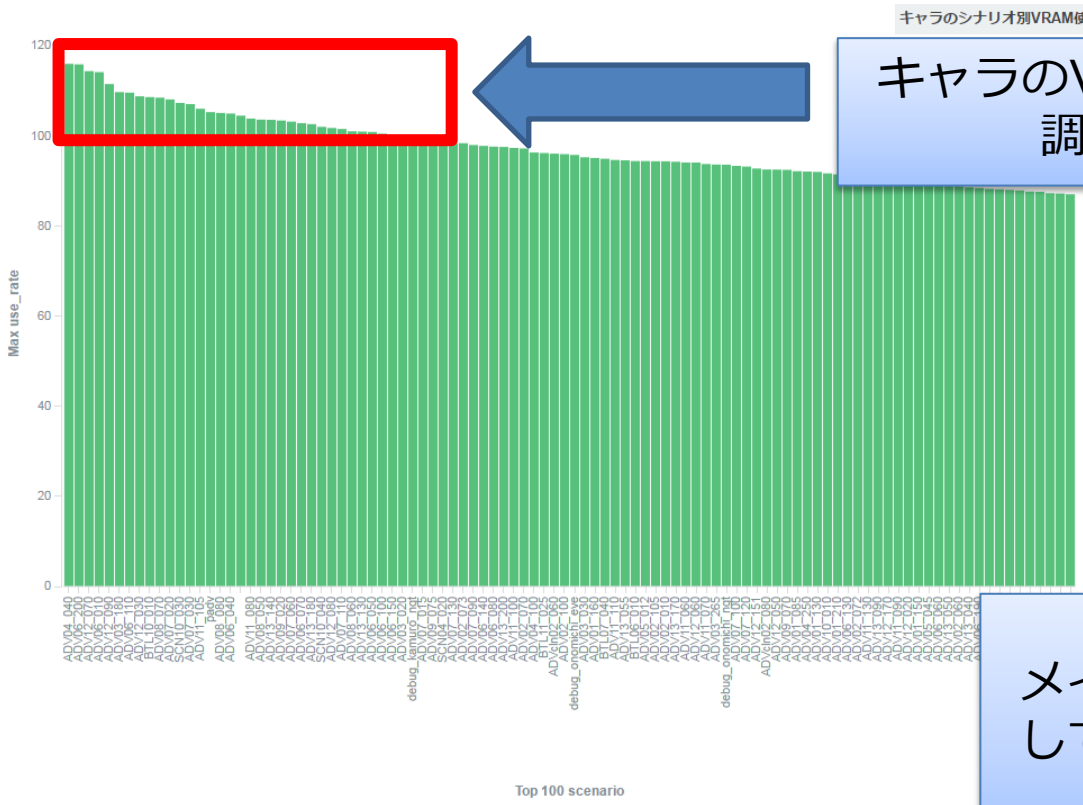
メインメモリ使用率（グラフ）



UI(紫色)の使用率が変動

オートテストの
特定条件下カテゴリ別
メインメモリ使用率

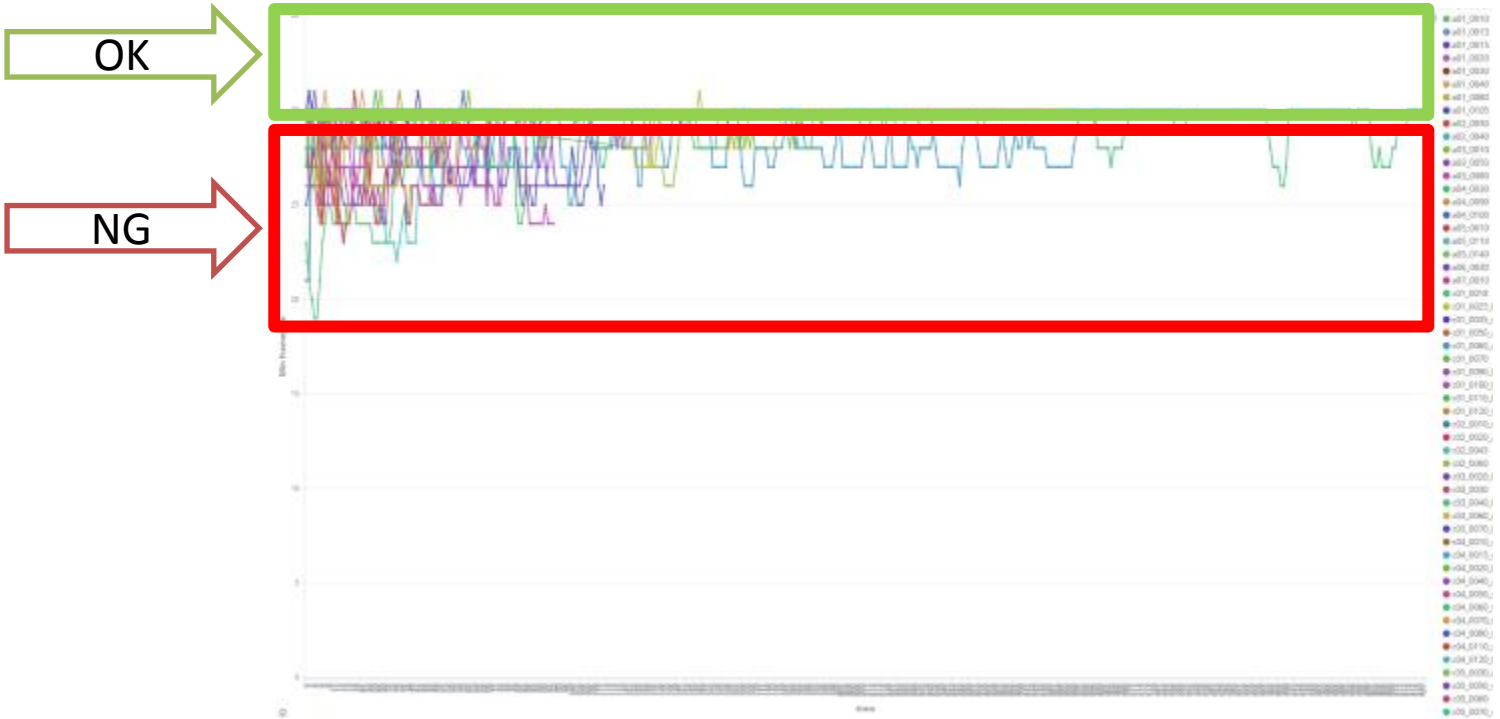
キャラのVRAM使用率（グラフ）



キャラのVRAMが100%を超えていて
調整が必要なシナリオ

オートテストが
メインストーリーをすべて周回
しているのですべてのシナリオ
を計測可能

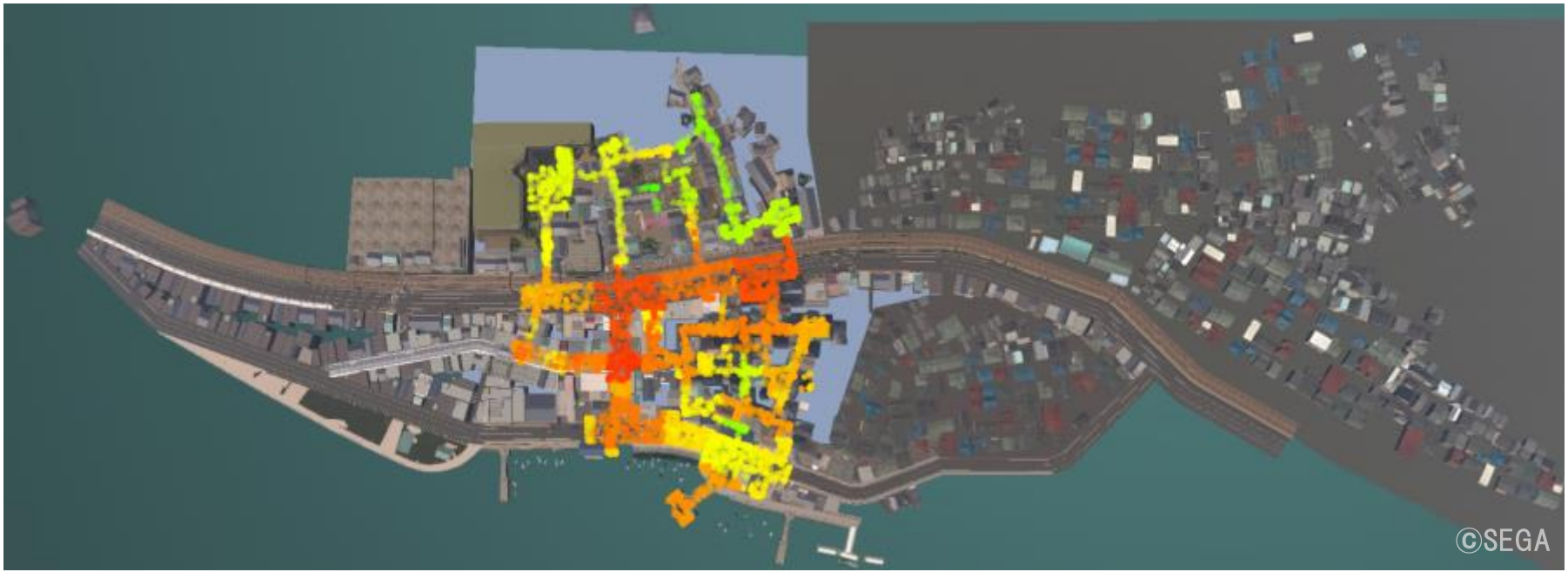
演出シーンのフレームレート分析(極2)



30fpsを下回って処理落ちしているシーンを見える化

役に立ったオリジナルのログ分析ツール

どこでもログ分析(2D版)



VRAMヒートマップ(龍６の神室町・修正前)

オートテストで神室町の
巡回パスを作成して毎日実行

× →100%を超えている
赤点→ギリギリ
緑点→大丈夫

静的にもチェックしていたが
自転車などのオブジェクトの
配置で変動していたので
動的確認が必要になった

VRAMヒートマップ(龍６の神室町・修正後)

真っ赤だが、
ぎりぎり範囲内に収まった

見える化したので
ギリギリを狙うことも可能
（どの範囲に収めるかの
申し合わせが必要）

コリジョン抜け（龍6の神室町）

オートテストの
ランダム移動を活用

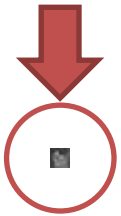
再現しにくいレアケースは
落ちる直前の60秒動画を確認

コリジョン抜け（北斗が如くの荒野）

神室町

荒野

これ



1600倍の面積

広い荒野をどのようにテストすればいいのか…

©SEGA ©武論尊・原哲夫/NSP 1983 版権許諾証GA-217

任意の場所までパス移動後ランダム

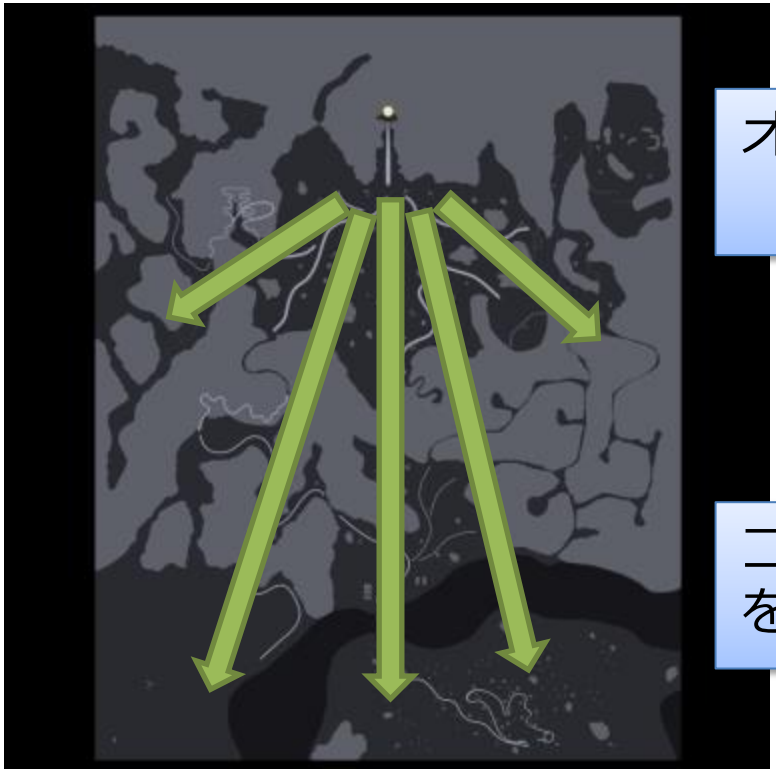
パス移動

目的地に到着

ランダム移動

©SEGA ©武論尊・原哲夫/NSP 1983 版権許諾証GA-217

コリジョン抜け（北斗が如くの荒野）

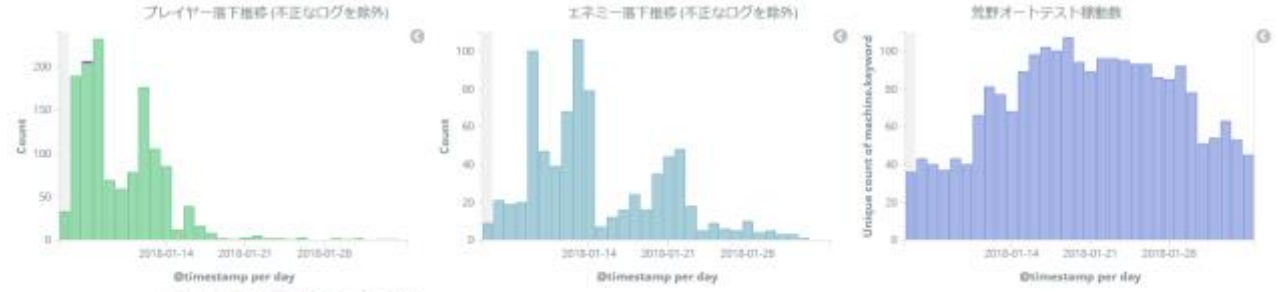


オートテストを
100台投入

コリジョン抜け
をくまなく検出

©SEGA ©武論尊・原哲夫/NSP 1983 版権許諾証GA-217

コリジョン抜け（北斗が如くの荒野）



コリジョン抜け動画（強制ワープを除外）

1-50 of 1,151

Time	message	player_pos	stage	movie_capture_path	revision	machine
	【FALL】プレイヤーのパギーが落ちた！(-3921.2378, 377.9391, -4154.6479) (STID_ST_WILDLAND)	-3920.404, 378.434, -4154.824, 1	STID_ST_WILD AND		50,752	
	【FALL】プレイヤーのパギーが落ちた！(-3108.3022, 248.3661, -3908.8767) (STID_ST_WILDLAND)	-3108.761, 249.066, -3908.37, 1	STID_ST_WILD AND			
	【FALL】プレイヤーのパギーが落ちた！(-3155.5469, 289.8750, -4229.4946) (STID_ST_WILDLAND)	-	-		50,737	
	【FALL】プレイヤーのパギーが落ちた！(-3276.1001, 211.0465, -3791.6287) (STID_ST_WILDLAND)	-3276.805, 211.814, -3791.966, 1	STID_ST_WILD AND		50,168	
	【FALL】プレイヤーのパギーが落ちた！(-3944.2256, 105.2687, -3671.7129) (STID_ST_WILDLAND)	-3943.696, 106.065, -3671.951, 1	STID_ST_WILD AND		50,038	
	【FALL】プレイヤーのパギーが落ちた！(-3936.5239, 353.7408, -4152.6182) (STID_ST_WILDLAND)	-3935.679, 354.128, -4152.948, 1	STID_ST_WILD AND		49,994	

日時

動画ファイルのパス

PC名

落下直前の60秒動画を確認

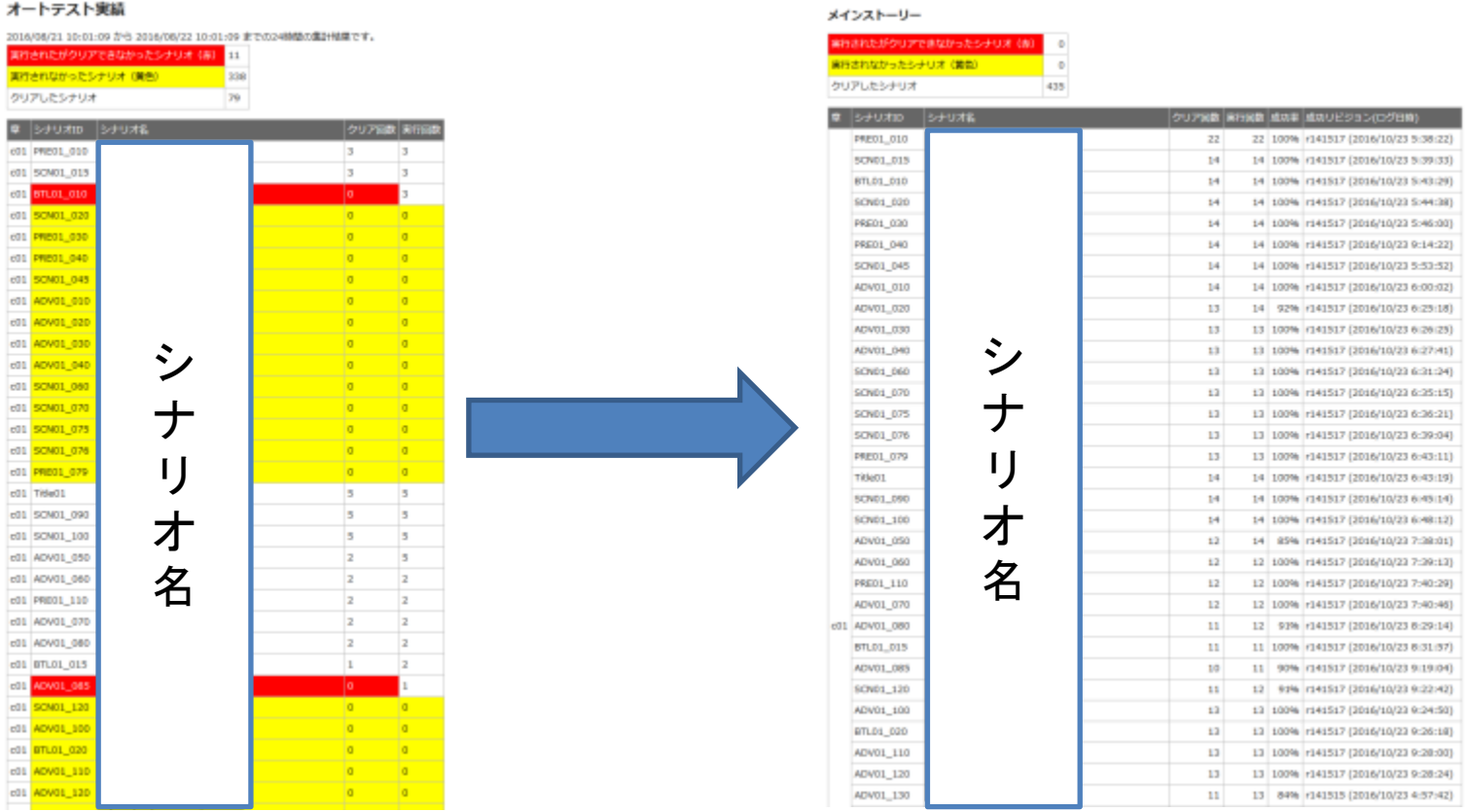
コリジョン抜け（北斗が如くの荒野）

©SEGA ©武論尊・原哲夫/NSP 1983 版権許諾証GA-217

オートテストの結果分析

- オートテストの実行結果の定量化
- シナリオのクリア状況の集計に活用
  - 進行不能のシナリオがあればチェック
  - パスに問題があれば逐次修正
- 組み合わせテストの結果確認
  - 要素×要素×要素＝数千通りの全組み合わせを実行
  - テスト結果を逐次ログ送信して確認

オートテストのシナリオクリア分析



オートテスト実績

2016/08/21 10:01:09 から 2016/08/22 10:01:09 までの24時間の集計結果です。

実行されたがクリアできなかったシナリオ（赤）	11
実行されなかったシナリオ（黄色）	338
クリアしたシナリオ	79

章	シナリオID	シナリオ名	クリア回数	実行回数
c01	PRE01_010		3	3
c01	SCN01_015		3	3
c01	BTL01_010		0	3
c01	SCN01_020		0	0
c01	PRE01_030		0	0
c01	PRE01_040		0	0
c01	SCN01_045		0	0
c01	ADV01_010		0	0
c01	ADV01_020		0	0
c01	ADV01_030		0	0
c01	ADV01_040		0	0
c01	SCN01_060		0	0
c01	SCN01_070		0	0
c01	SCN01_075		0	0
c01	SCN01_076		0	0
c01	PRE01_079		0	0
c01	Title01		5	5
c01	SCN01_090		5	5
c01	SCN01_100		5	5
c01	ADV01_050		2	5
c01	ADV01_060		2	2
c01	PRE01_110		2	2
c01	ADV01_070		2	2
c01	ADV01_080		2	2
c01	BTL01_015		1	2
c01	ADV01_085		0	1
c01	SCN01_120		0	0
c01	ADV01_100		0	0
c01	BTL01_020		0	0
c01	ADV01_110		0	0
c01	ADV01_120		0	0

シナリオ名

メインストーリー

実行されたがクリアできなかったシナリオ（赤）	0
実行されなかったシナリオ（黄色）	0
クリアしたシナリオ	435

章	シナリオID	シナリオ名	クリア回数	実行回数	成功率	成功リビジョン(ログ日時)
	PRE01_010		22	22	100%	r141517 (2016/10/23 5:38:22)
	SCN01_015		14	14	100%	r141517 (2016/10/23 5:39:33)
	BTL01_010		14	14	100%	r141517 (2016/10/23 5:43:29)
	SCN01_020		14	14	100%	r141517 (2016/10/23 5:44:38)
	PRE01_030		14	14	100%	r141517 (2016/10/23 5:46:00)
	PRE01_040		14	14	100%	r141517 (2016/10/23 9:14:22)
	SCN01_045		14	14	100%	r141517 (2016/10/23 5:53:52)
	ADV01_010		14	14	100%	r141517 (2016/10/23 6:00:02)
	ADV01_020		13	14	92%	r141517 (2016/10/23 6:25:18)
	ADV01_030		13	13	100%	r141517 (2016/10/23 6:26:25)
	ADV01_040		13	13	100%	r141517 (2016/10/23 6:27:41)
	SCN01_060		13	13	100%	r141517 (2016/10/23 6:31:24)
	SCN01_070		13	13	100%	r141517 (2016/10/23 6:35:15)
	SCN01_075		13	13	100%	r141517 (2016/10/23 6:36:21)
	SCN01_076		13	13	100%	r141517 (2016/10/23 6:39:04)
	PRE01_079		13	13	100%	r141517 (2016/10/23 6:43:11)
	Title01		14	14	100%	r141517 (2016/10/23 6:43:19)
	SCN01_090		14	14	100%	r141517 (2016/10/23 6:45:14)
	SCN01_100		14	14	100%	r141517 (2016/10/23 6:48:12)
	ADV01_050		12	14	85%	r141517 (2016/10/23 7:38:01)
	ADV01_060		12	12	100%	r141517 (2016/10/23 7:39:13)
	PRE01_110		12	12	100%	r141517 (2016/10/23 7:40:29)
	ADV01_070		12	12	100%	r141517 (2016/10/23 7:40:46)
c01	ADV01_080		11	12	91%	r141517 (2016/10/23 8:29:14)
	BTL01_015		11	11	100%	r141517 (2016/10/23 8:31:57)
	ADV01_085		10	11	90%	r141517 (2016/10/23 9:19:04)
	SCN01_120		11	12	91%	r141517 (2016/10/23 9:22:42)
	ADV01_100		13	13	100%	r141517 (2016/10/23 9:24:50)
	BTL01_020		13	13	100%	r141517 (2016/10/23 9:26:18)
	ADV01_110		13	13	100%	r141517 (2016/10/23 9:28:00)
	ADV01_120		13	13	100%	r141517 (2016/10/23 9:28:24)
	ADV01_130		11	13	84%	r141515 (2016/10/23 4:57:42)

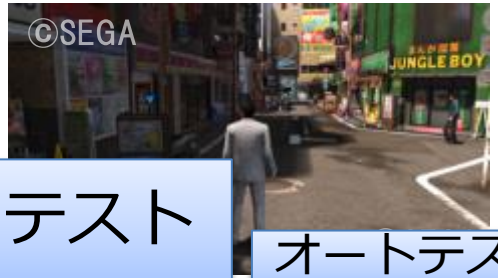
シナリオ名

デバッグログ分析活用事例

- エラー検出
  - メモリ使用率（グラフ）
  - 演出シーンのフレームレート分析
  - メモリ使用率（ヒートマップ）
  - コリジョン抜け
  - オートテスト結果分析
- ゲームバランス調整
  - 桐生が死んだマップ
  - 成長ログ
  - 死亡回数と蓄積ダメージ
  - アイテムドロップ率

ゲームバランス調整フロー

日ごと or リアルタイムでトライ＆エラーできる！



手動テスト

オートテスト

調整

ログ送信

Elasticsearch

自動集計

Jenkinsで1日1回

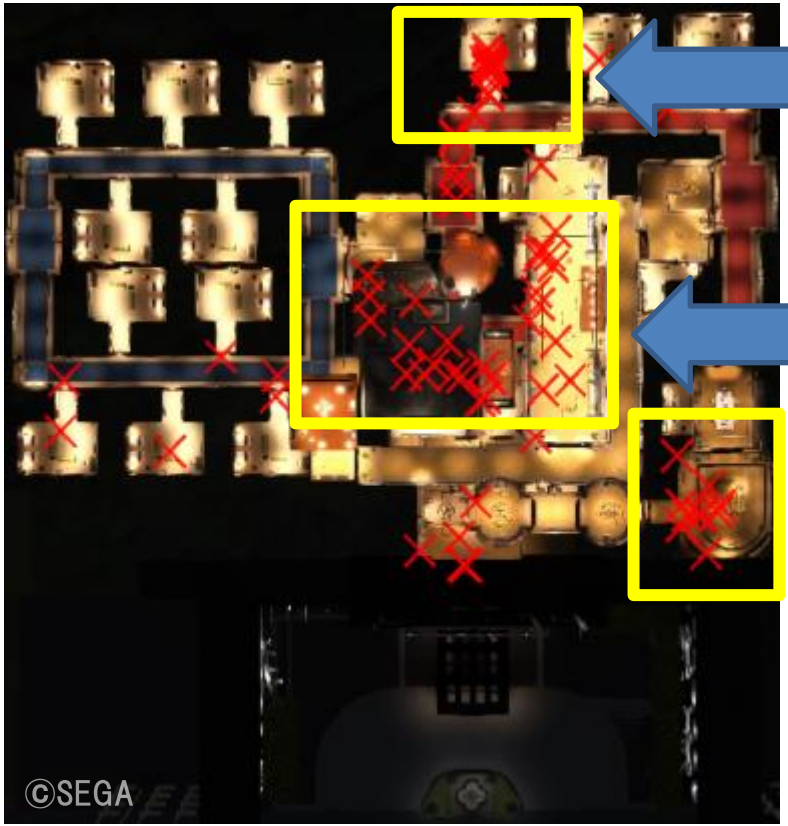
Kibanaで即確認

プレイヤーが中ボス戦で死にすぎてる。中ボスを少し弱くしてみよう。

集計結果を確認

Redmine内のWikiに公開

桐生が死んだマップ（手動テスト）



あれ？
ここのザコ敵
強くしすぎた？

ボスや中ボス戦
なので想定内

成長ログ（手動テスト）

Wiki » バトル調整 »

成長ログ_1週間

2016/11/07 20:16:25 から 2016/11/14 20:16:25 までの24時間の集計結果

scenario_after	scenario_before	machine	revision	@timestamp							
Title01	PRE01_079		42163	2016/11/ 14:47:07	15000	15000	30000	0	0	0	0
Title01	PRE01_079		42163	2016/11/ 15:12:29	15000	15000	30000	0	0	0	0
Title01	PRE01_079		42283	2016/11/ 15:52:56	30000	30000	1266430	5483	1894	4930	548
Title02	END01		42163	2016/11/ 15:32:15	16200	16200	8140	111	92	94	76
Title02	END01		42163	2016/11/ 15:32:32	16200	16200	8140	111	92	94	76
Title02	END01		42163	2016/11/ 15:43:54	15899	15899	98770	151	138	129	83
Title02	END01		42163	2016/11/ 15:44:01	15899	15899	98770	151	138	129	83
Title02	END01		42283	2016/11/ 16:14:43	30000	30000	1265700	5657	2047	5085	609
Title02	END01		42283	2016/11/ 16:15:00	30000	30000	1265700	5657	2047	5085	609
Title03	END02		42163	2016/11/ 15:45:14	17100	17100	6680	161	154	115	66
Title03	END02		42163	2016/11/ 15:45:38	17100	17100	6680	161	154	115	66
Title03	END02		42163	2016/11/ 16:29:41	16800	16800	44280	140	110	105	50

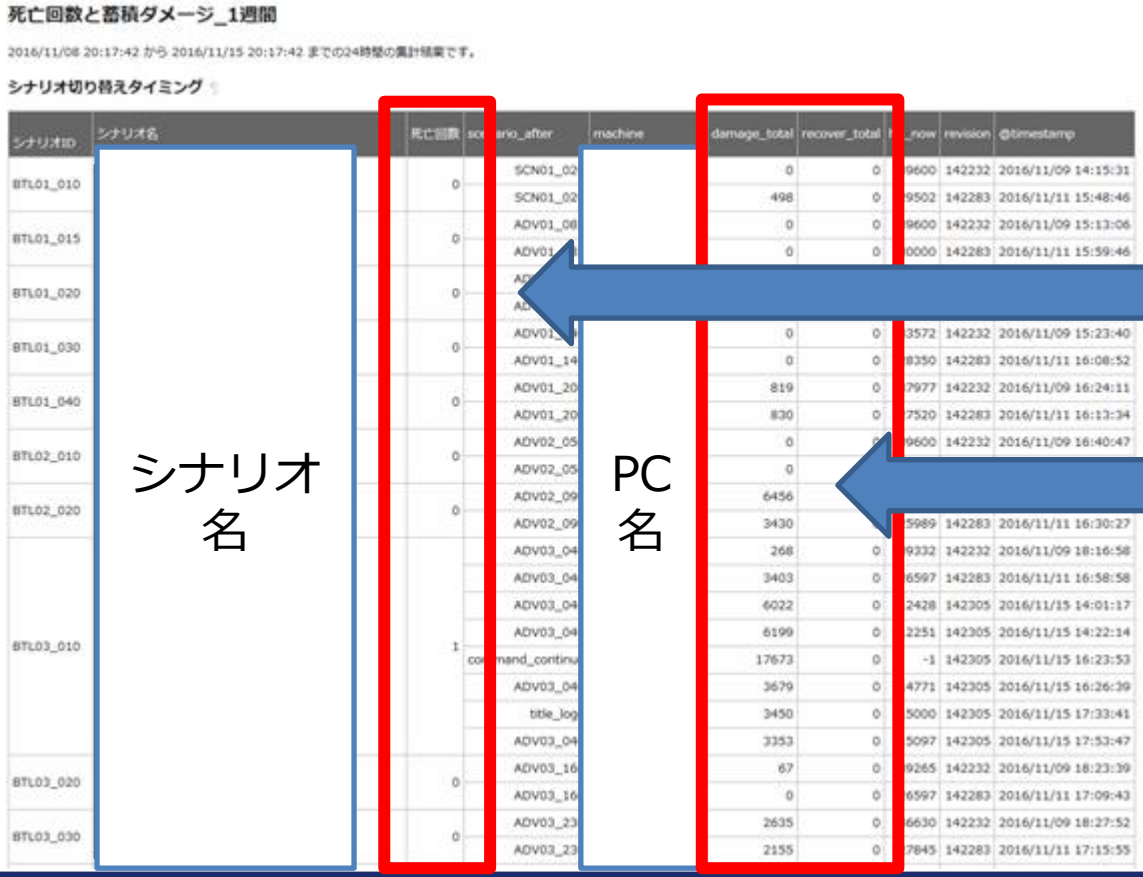
PC名

経験値やレベル等

各章ごとの経験値を集計

想定どおりに
経験値を取得しているか
スキルレベルが適正か

死亡回数と蓄積ダメージ(手動テスト)



死亡回数と蓄積ダメージ_1週間

2016/11/08 20:17:42 から 2016/11/15 20:17:42 までの24時間の集計結果です。

シナリオ切り替えタイミング

シナリオID	シナリオ名	死亡回数	sc	ario_after	machine	damage_total	recover_total	_now	revision	@timestamp
BTL01_010		0		SCN01_02		0	0	9600	142232	2016/11/09 14:15:31
				SCN01_02		498	0	9502	142283	2016/11/11 15:48:46
BTL01_015		0		ADV01_08		0	0	9600	142232	2016/11/09 15:13:06
				ADV01		0	0	0000	142283	2016/11/11 15:59:46
BTL01_020		0		AD						
				AD						
BTL01_030		0		ADV01		0	0	3572	142232	2016/11/09 15:23:40
				ADV01_14		0	0	8350	142283	2016/11/11 16:08:52
BTL01_040		0		ADV01_20		819	0	7977	142232	2016/11/09 16:24:11
				ADV01_20		830	0	7520	142283	2016/11/11 16:13:34
BTL02_010		0		ADV02_05		0	0	9600	142232	2016/11/09 16:40:47
				ADV02_05		0				
BTL02_020		0		ADV02_09		6456				
				ADV02_09		3430		5989	142283	2016/11/11 16:30:27
BTL03_010		1		ADV03_04		268	0	9332	142232	2016/11/09 18:16:58
				ADV03_04		3403	0	6597	142283	2016/11/11 16:58:58
				ADV03_04		6022	0	2428	142305	2016/11/15 14:01:17
				ADV03_04		6199	0	2251	142305	2016/11/15 14:22:14
			co	mand_continu		17673	0	-1	142305	2016/11/15 16:23:53
				ADV03_04		3679	0	4771	142305	2016/11/15 16:26:39
				title_log		3450	0	5000	142305	2016/11/15 17:33:41
				ADV03_04		3353	0	5097	142305	2016/11/15 17:53:47
BTL03_020		0		ADV03_16		67	0	9265	142232	2016/11/09 18:23:39
				ADV03_16		0	0	6597	142283	2016/11/11 17:09:43
BTL03_030		0		ADV03_23		2635	0	6630	142232	2016/11/09 18:27:52
				ADV03_23		2155	0	7845	142283	2016/11/11 17:15:55

シナリオ名

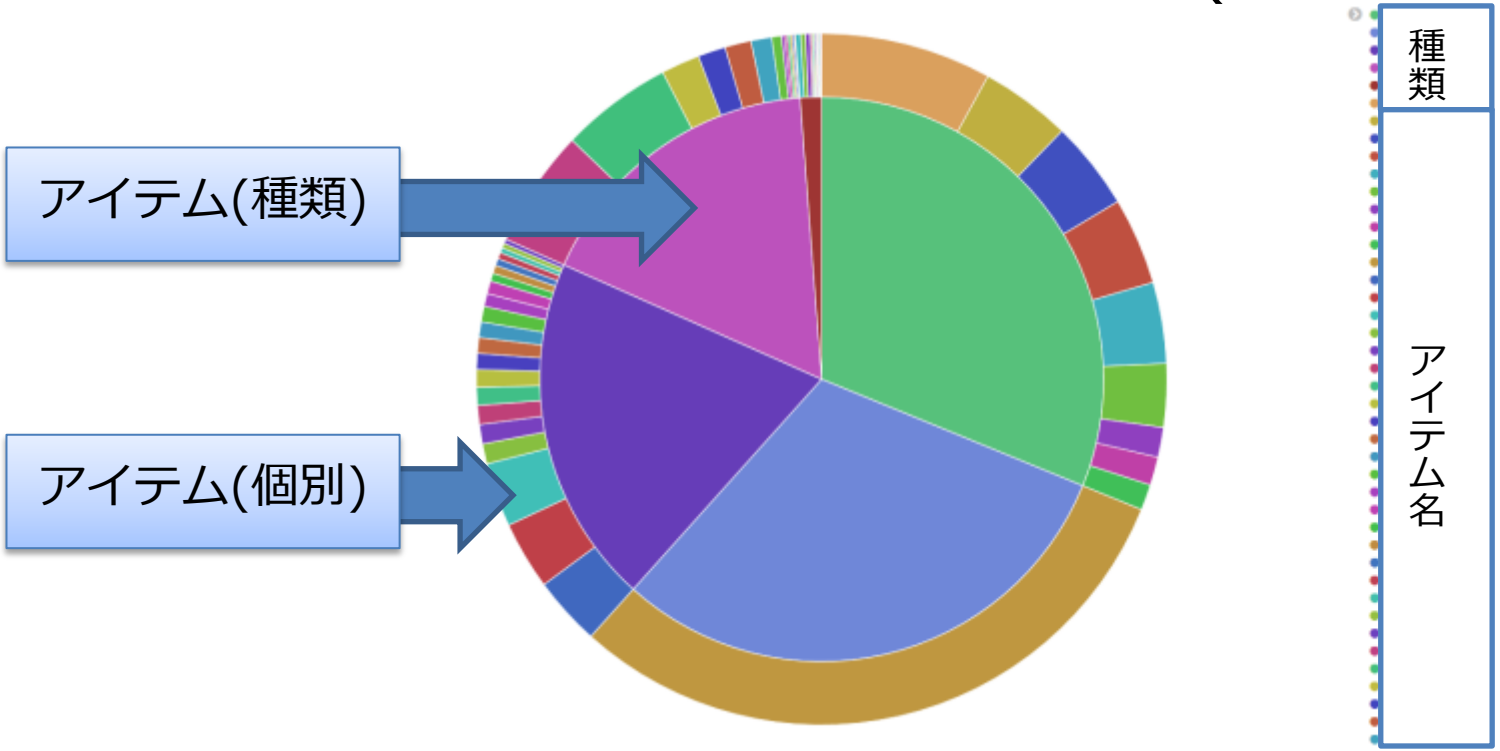
PC名

各シナリオで
プレイヤーが
死んだ回数

各シナリオで受けた
ダメージの合計値
回復アイテムの合計値

アイテムドロップ率（オートテスト）

敵が落とすアイテムの種類と確率の確認(北斗が如く)



リアルタイムログ分析の活用事例

- 手動テストでのプレイヤー死亡回数をリアルタイムで監視
- 想定以上にバトルで死にすぎているときに、すぐにチェック席に行ってヒヤリングする
- 各テストプレイヤーの特性を発見
  - アクションゲームが苦手なタイプ
  - 敵の攻撃を避けるのが上手い

ログサーバ運用結果（龍6）

- ログ送信量
  - ピーク時に200万行/時間くらい
  - デイリーだと2000万行/日程度
- ログサーバ運用
  - たまにFluentdとElasticsearchの接続が切れたままになる
- ログサーバのデータ量
  - デバッグログ(printf)はでかいので、２週間分だけ残してcuratorというツールで削除
  - 「どこでもログ送信」はすべて残して、最終的にログデータは150GBほどになった
- 速度
  - 古いログはあまり分析しないので、それほど重くならなかった
  - 重くなりそうな分析は、夜中にJenkinsで定期的に集計してRedmineのWikiにアップロードして対処

かかった工数（龍6）

- ログサーバ設置
  - 2～3日
- ログ送信とログ分析（Kibana）
  - 2～3週間
- ログ分析結果を見やすくする努力
  - 2～3ヶ月

ログを収集してKibanaでグラフ化するだけなら簡単で低コスト

分析結果を見たい人に、どう見える化して伝えるかが大事

デバッグログ分析の課題（龍6）

- ログ分析ツールの周知＆活用
- プログラマを介さず誰でも気軽にログ分析できるようにしたい
- 「どこでもログ送信」の積極的な追加
  - プログラマが気軽に組み込んでいく
  - 企画やデザイナからの要望も受ける
- 「どこでもログ送信」の高速化
  - マルチスレッド化

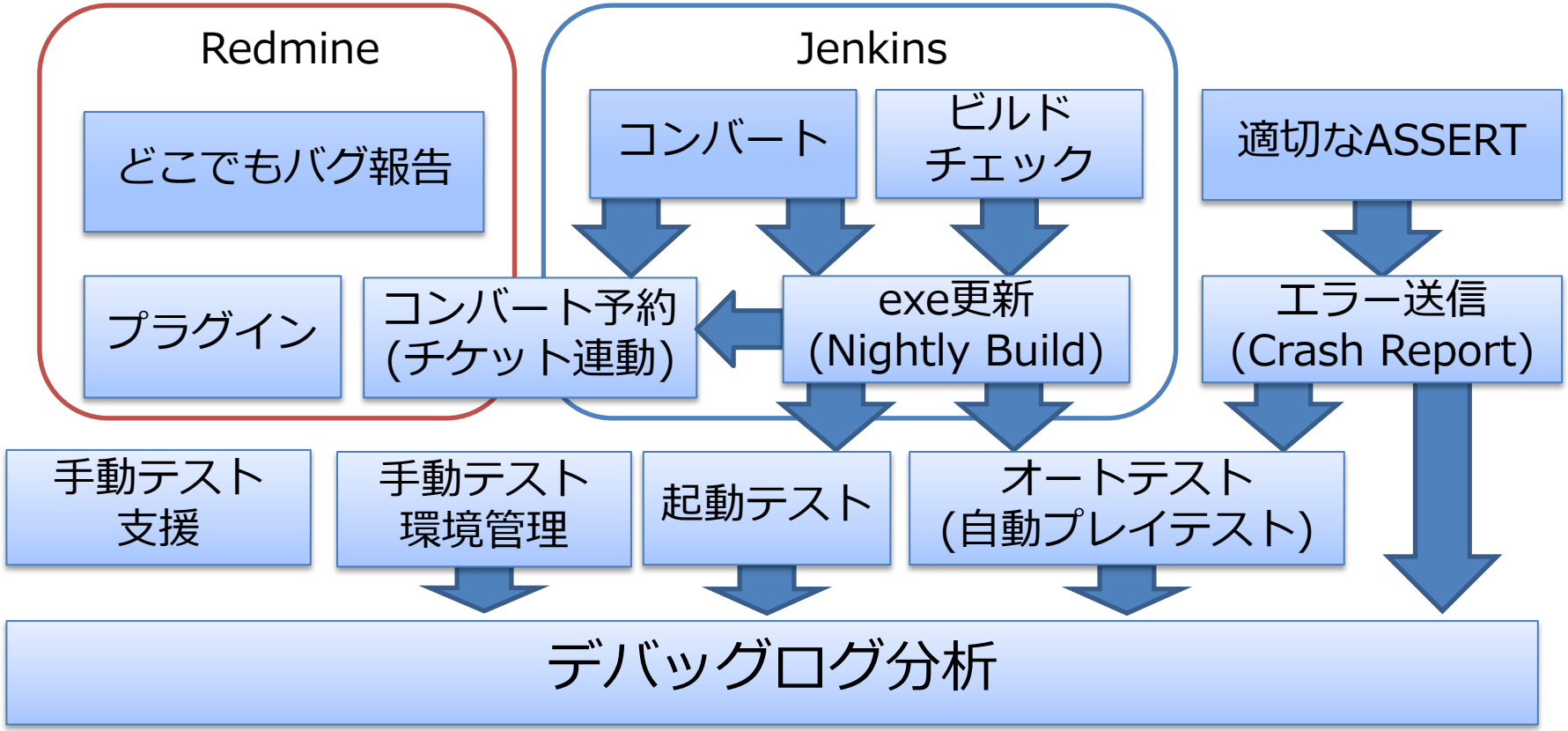
現在の進捗（極２/北斗が如く）

- ログ分析の活用人数と事例の増加
- どこでもログ送信
  - マルチスレッドに完全対応して高速化
  - 各プログラマが組み込んで独自に分析
- BTSライブラリ(ドラゴンエンジン内)に統合
  - 他のプロジェクトでも利用可能に
  - BTS部分を移植して、北斗が如くでも活用
- Elastic Stack
  - Elasticsearch/Kibanaは、5.4にバージョンアップ
  - ログの量が増えてサーバ負荷が増えたので、サーバの台数を追加
  - 今から始めるなら最新の6.X系がお勧め

まとめ

- デバッグログ分析の導入は無料(OSS)でできる
- 見える化にコストがかかるがやる価値はある
- 自動化あってのデバッグログ分析
  - 信頼できるログがなければ用意するのが先！
  - 分析の目的（課題）を見つけるのが先！

自動化あってのデバッグログ分析



デバッグログ分析とは

自動化の果てにたどり着いたご褒美

※個人の感想です。

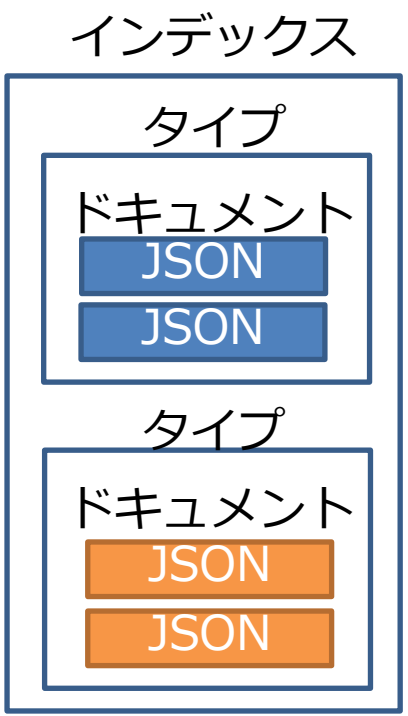
ゲーム開発におけるQAエンジニアとは

楽しい開発環境を作って
ゲーム開発を楽しくする仕事

参考資料．書籍・スライド資料

- ログ分析
  - CEDEC 2017
    - 無料で始める！「龍が如く」を面白くするための高速デバッグログ分析と自動化
      - https://cedil.cesa.or.jp/cedil_sessions/view/1621
  - サーバ／インフラエンジニア養成読本 ログ収集～可視化編（技術評論社）
  - 高速スケーラブル検索エンジン ElasticSearch Server（KADOKAWA / アスキー・メディアワークス）
  - Elastic Stackで作るBI環境　誰でもできるデータ分析入門（インプレスR&D）
  - データ分析基盤構築入門[Fluentd、Elasticsearch、Kibanaによるログ収集と可視化]（技術評論社）
- 自動化
  - KYUSHU CEDEC 2015
    - 「龍が如く」の高速デバッグ術 ～そびえ立つバグの山を踏破するための弾丸ワークフロー～
      - https://cedil.cesa.or.jp/cedil_sessions/view/1398
  - JaSST'16 Tokyo
    - 「龍が如く」の高速デバッグ術 ～そびえ立つバグの山を踏破するための弾丸ワークフロー～**（完全版）**
      - http://jasst.jp/symposium/jasst16tokyo/pdf/E2.pdf

参考資料. Elasticsearchのデータ構造



Elasticsearch	説明	MySQLだと
index	documentの集合体	database
type	documentの分類	table
document	データの最小単位	record
field	カラム。fieldごとに型を設定できる	column
mapping	各fieldの値の定義(string/integerなど) 未定義は自動推測	テーブル定義

※無理やりMySQLと比較しているので、全く同じではないです。
※typeは将来的になくなります。

参考資料. 龍６でのデータ構成(補足)

- インデックスは、１日単位
  - 古い日付のログを削除するため
- カテゴリ別にインデックスも分ける
  - stage, scene, battle, motion …
  - カテゴリごとに削除する範囲を分けるため
- タイプを必ず指定
  - 別のデータが紛れ込まないようにするため

参考資料. 龍が如くのJenkins活用例

- ビルドエラー検出の自動化
- 静的コード解析の自動化
  - Visual Studio コード分析
  - Coverity
- exe更新(Nightly Build)の自動化
  - バージョン管理サーバに自動コミット
  - PKG/ISOファイルを作成して自動アップロード
- 各種コンバートの自動化
  - 背景
  - キャラ
- ゲーム起動テスト（スモークテスト）の自動化
  - 背景の描画テスト
  - イベントシーンの再生テスト
- Redmineワークフローの自動化
  - コンバート結果に応じてチケットのステータスを自動的に変更

参考資料. 開発環境の監視への活用

- MetricBeat
  - Jenkins PCの稼動状況の確認
  - マシンリソースに余剰があるのか、足りなくなっているのかの指標
  - メモリの使用量、プロセスを監視して、Jenkins PCの異常を監視
- Jenkinsのログを送信
  - 実行時間やエラー率を分析