Breakthrough to Excellence

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

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

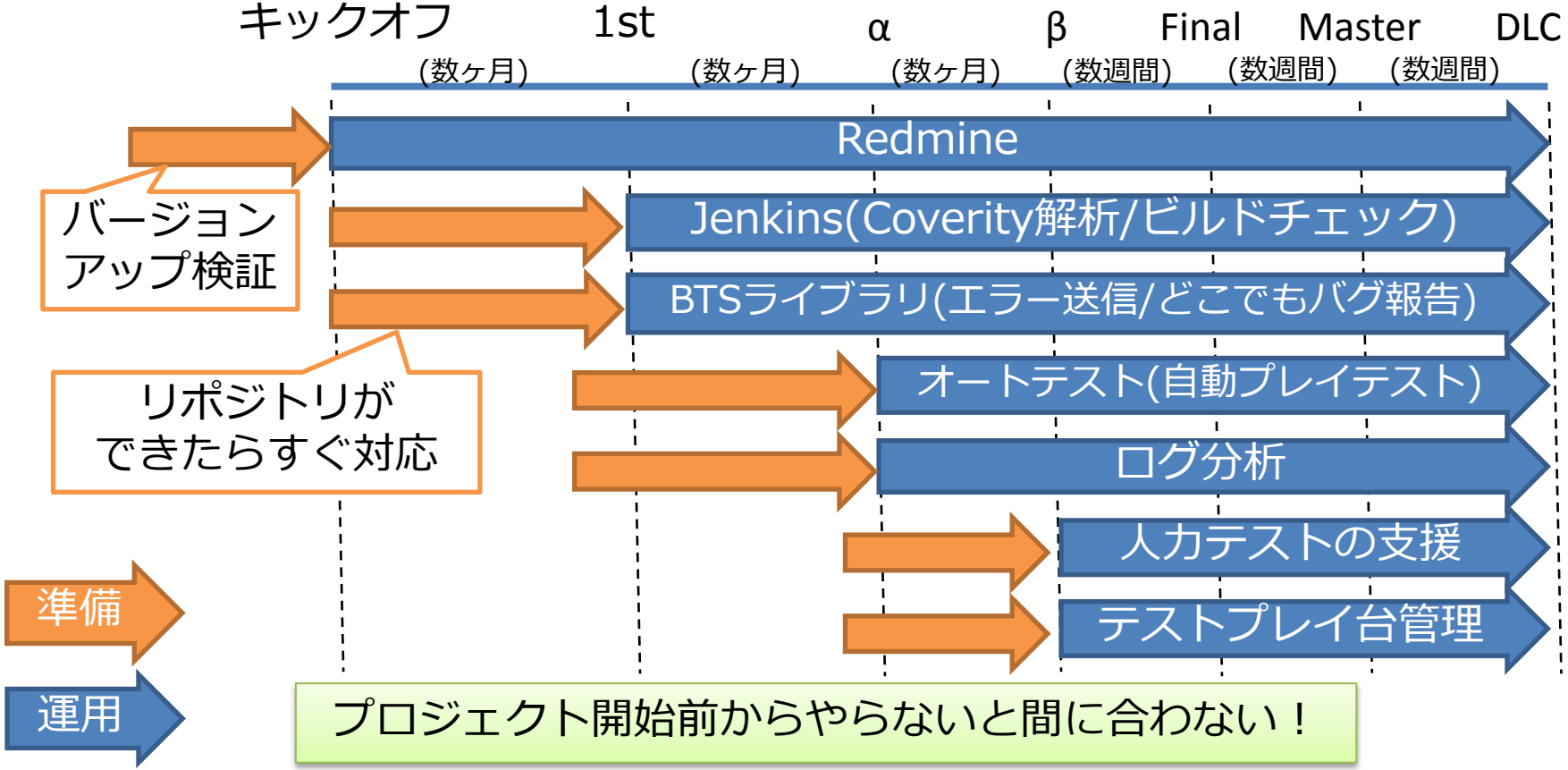
CEDEC 2017
Computer Entertainment Developers Conference

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れる
QAエンジニア（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

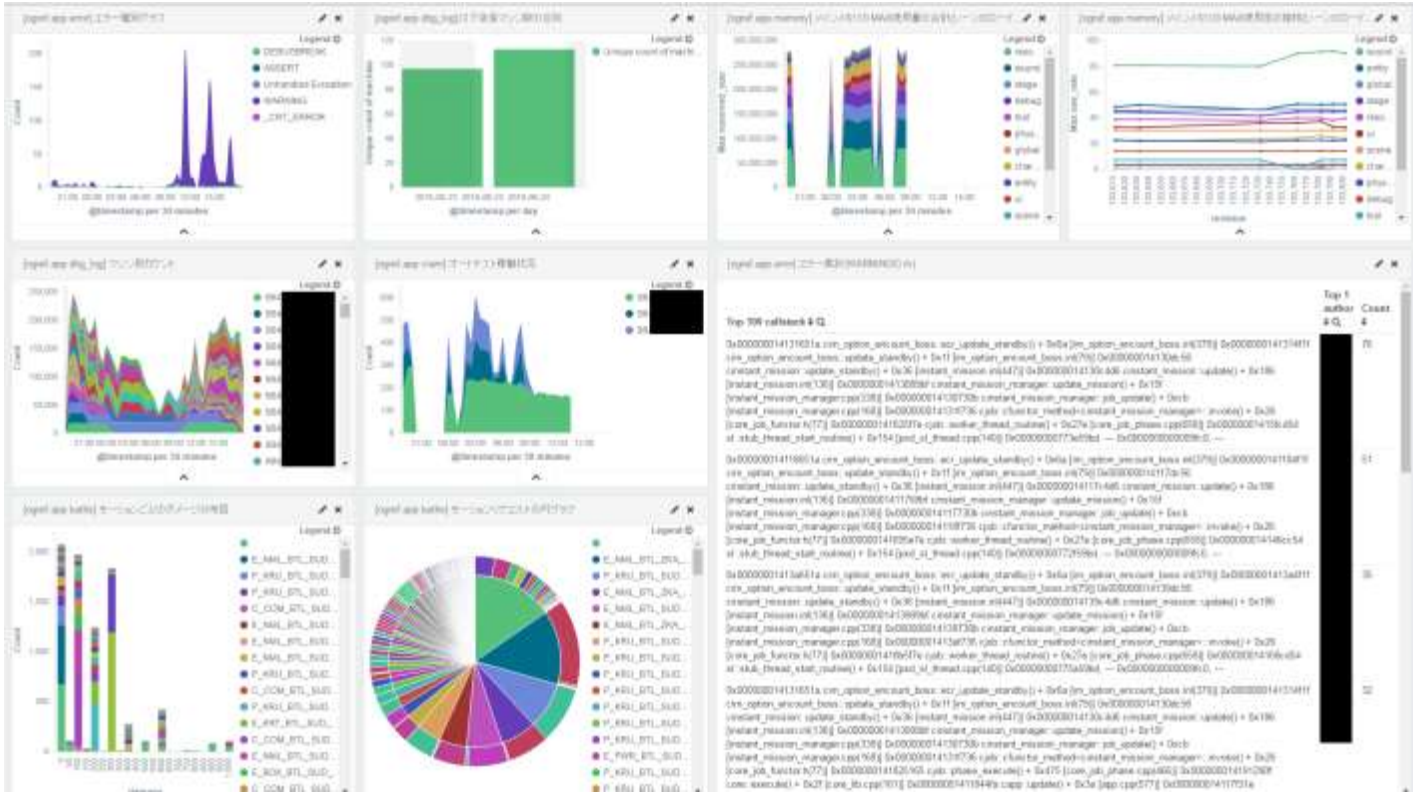
なぜデバッグログ分析が必要？

- 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

龍が如く６でのログデータの流れ

debug_printf

SEND_LOG_XXX

どこでも
ログ送信

ファイル出力
dbg_log_XXX.txt
log_send_XXX.txt

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)
  - 可視化・BI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

龍が如く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 (http://docs.fluentd.org/v0.14/articles/install-by-msi)
  - Elasticsearch (https://www.elastic.co/guide/en/elasticsearch/reference/5.x/windows.html)
  - Kibana (https://www.elastic.co/guide/en/kibana/5.x/windows.html)
  - 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/石井葵 著）

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

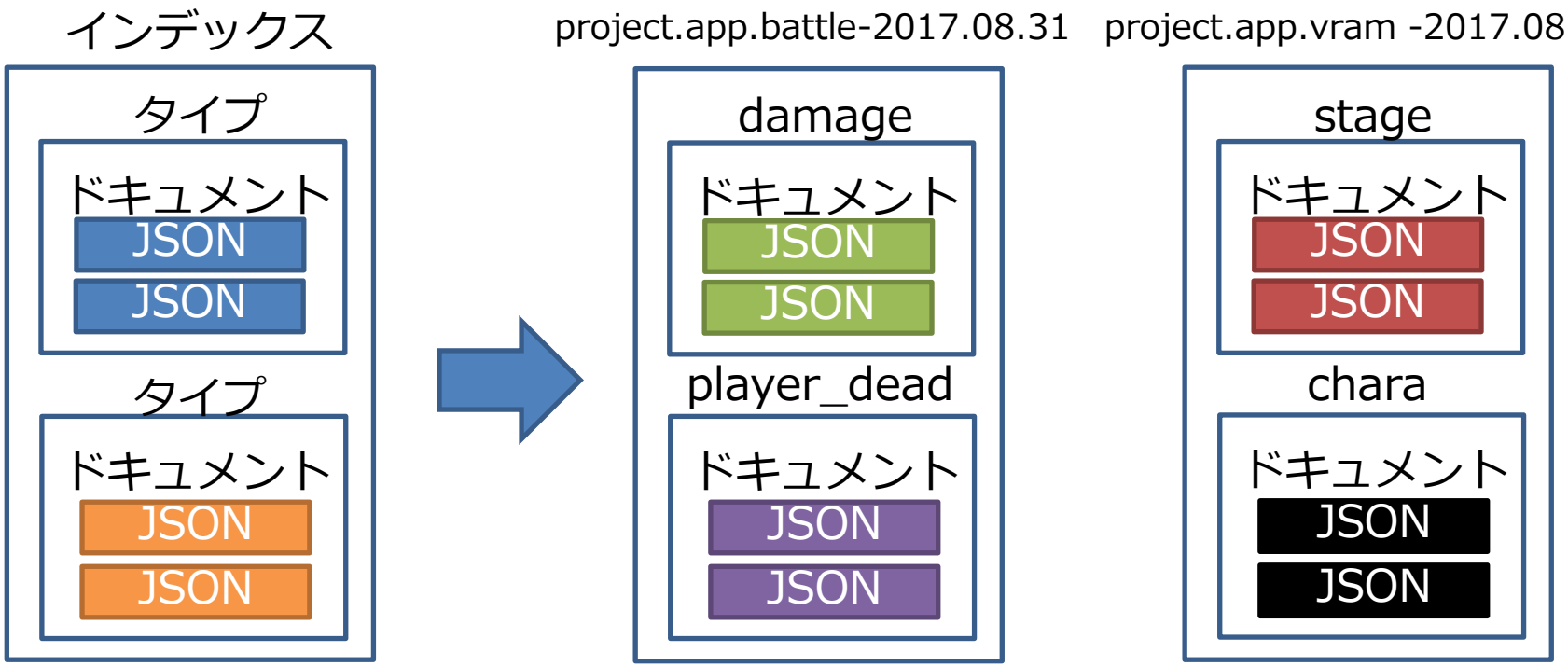
キー・バリュー形式（マクロ→JSON）

```
SEND_LOG_SAKAUE_COLLISION(
  "fall",
  "桐生が奈落へ落ちた",
  2, /*キー・バリューの数*/
  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


```
{
    "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)



龍が如く６でのデータ構成

- 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

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

使ってもらえなかった…（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

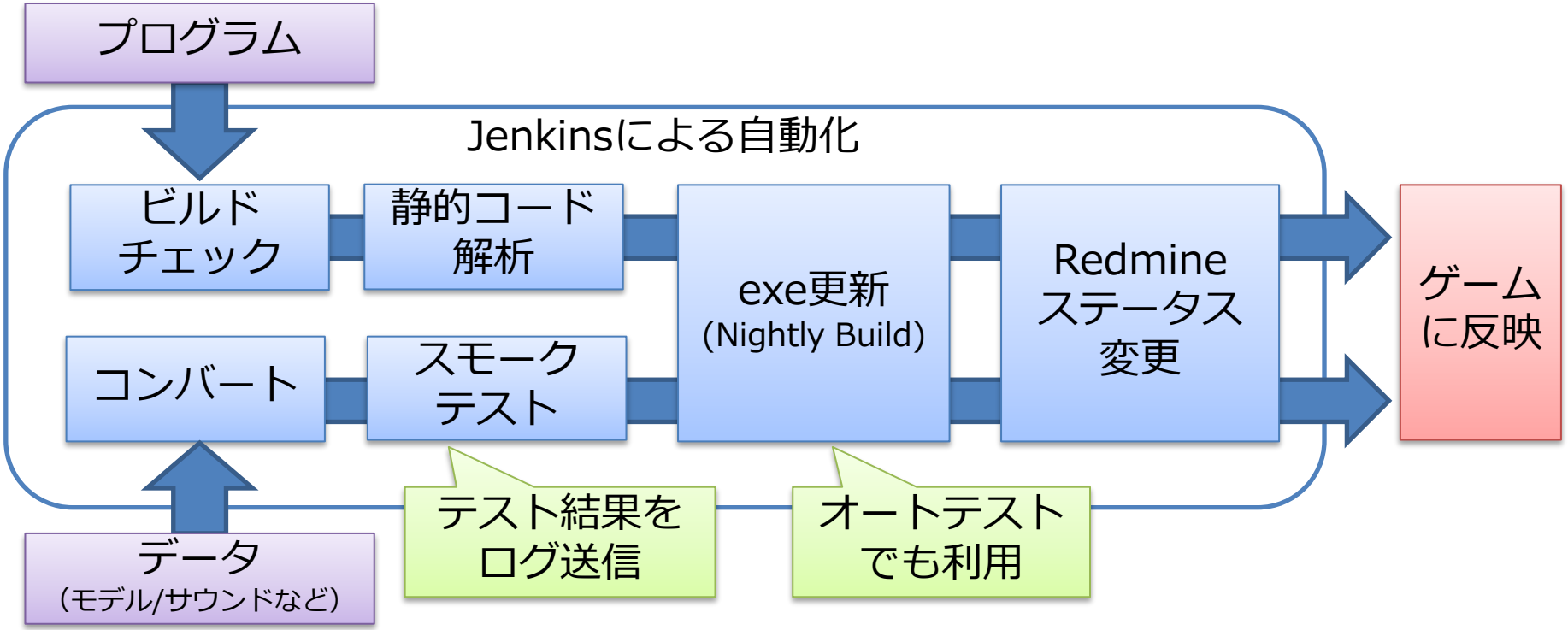
使ってもらうために

- 「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
  - テストプレイ台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
   - 独自ツール
3. デバッグログ分析と自動化の連携

Jenkins



詳しくはKYUSHU CEDEC 2015の講演資料を見てね！！
https://cedil.cesa.or.jp/cedil_sessions/view/1398

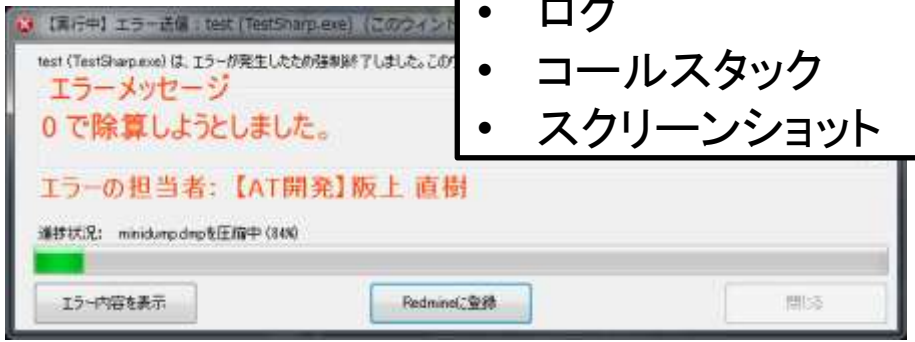
独自ツール

- 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)

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

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

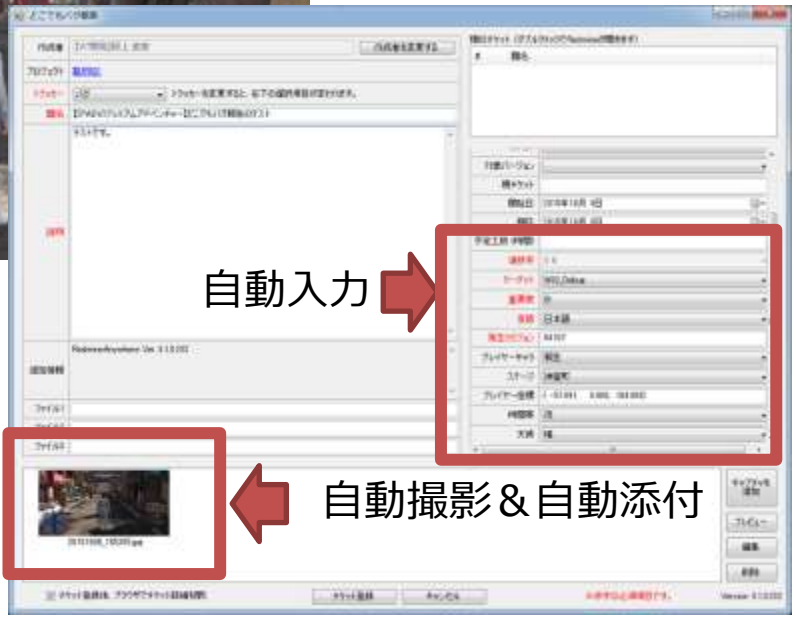
バグ報告 → バグ管理システム (Redmine)

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



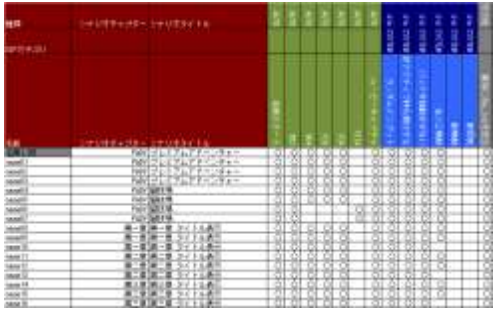
オートテスト

- 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（モンキーテスト）
    - 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

パスの作成と再生のデモ

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

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

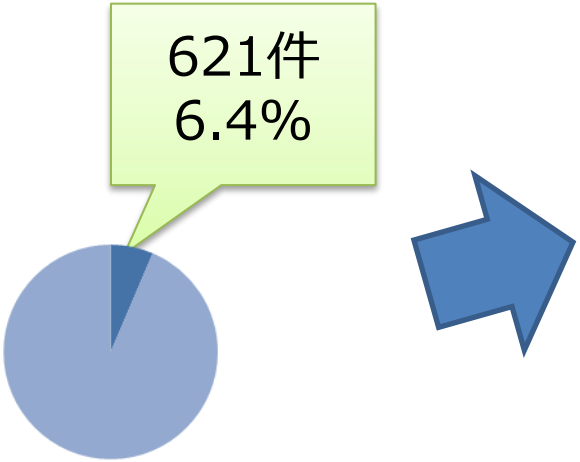
パスも自動生成へ

- 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…

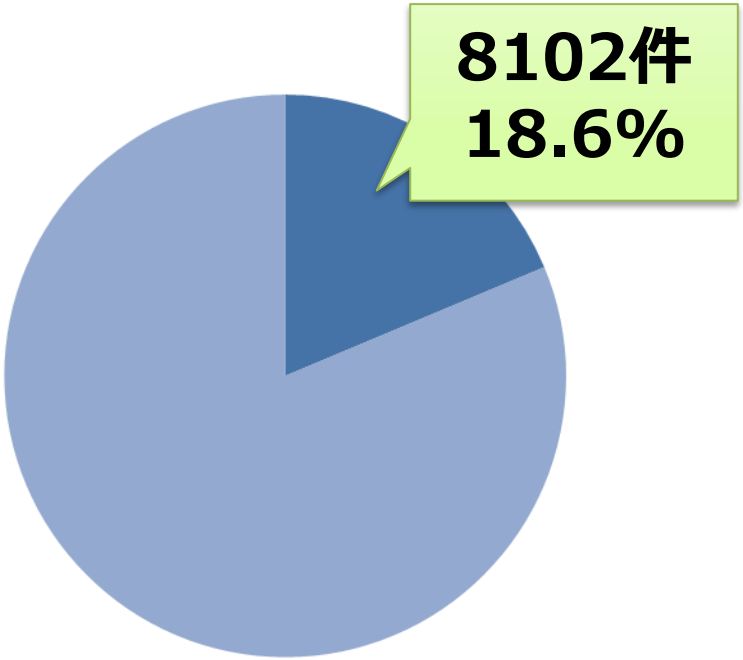
メインストーリーをクリアしまくって
自力でパスを作成・修正した！

オートテストのエラー検出数

オートテスト
人力



龍が如く０(ランダムのみ)



龍が如く６(ランダム＋パス)

エラー検出数は10倍以上！割合も３倍に増えた！

オートテストの運用結果

- 毎日150ターゲットを稼動
  - のべ10263日分のチェック実績(1日8時間換算)
- 24時間稼動PCを41台用意
  - リース切れ目前や在庫になっているPCをフル活用
  - 8時間ごとに最新ビルドを取得して再実行
- ランダム移動の活用
  - コリジョン抜けの30%をオートテストで検出
- パス移動の活用
  - 複数マシンを使ってメインストーリーを全クリア
  - リグレッションテストにも活用
  - 未再現バグの検証用パス（店の出入りを繰り返す等）
- 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

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
- テストプレイ台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

テストプレイ　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

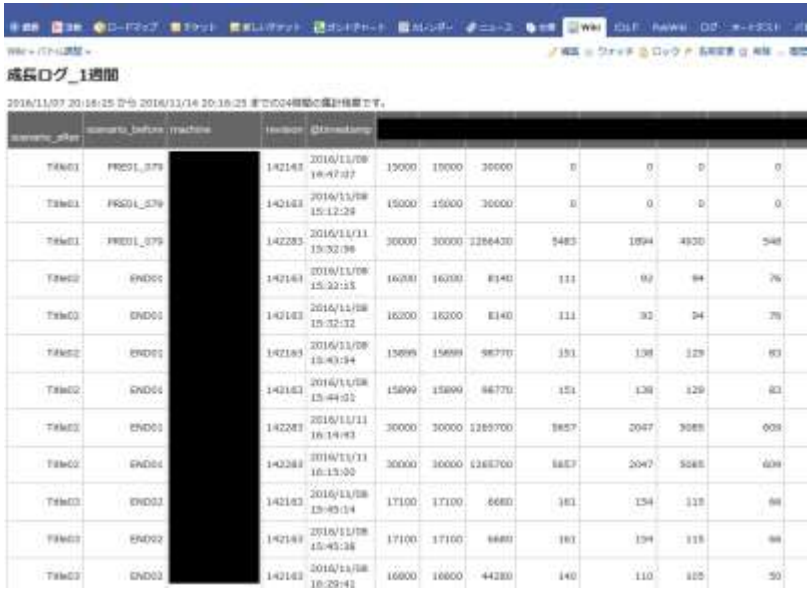
開発者

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

scenario_after	scenario_before	machine	revision	@timestamp							
Title01	PRE01_079	[redacted]	142163	2016/11/09 16:47:07	15000	15000	30000	0	0	0	0
Title01	PRE01_079		142163	2016/11/09 15:12:29	15000	15000	30000	0	0	0	0
Title01	PRE01_079		142283	2016/11/11 15:32:36	30000	30000	1286430	5483	1894	4930	548
Title02	END01		142163	2016/11/09 15:22:15	16200	16200	8140	111	92	94	76
Title02	END01		142163	2016/11/09 15:32:32	16200	16200	8140	111	92	94	76
Title02	END01		142163	2016/11/09 15:43:34	15899	15899	98770	151	138	129	83
Title02	END01		142163	2016/11/09 15:44:02	15899	15899	98770	151	138	129	83
Title03	END01		142283	2016/11/11 16:14:43	30000	30000	1283700	3857	2047	3085	609
Title03	END01		142283	2016/11/11 16:15:00	30000	30000	1283700	3857	2047	3085	609
Title03	END02		142163	2016/11/09 15:45:14	17100	17100	6680	161	154	115	98
Title03	END02		142163	2016/11/09 15:45:38	17100	17100	6680	161	154	115	98
Title03	END03		142163	2016/11/08 16:29:41	16800	16800	44280	140	110	105	50

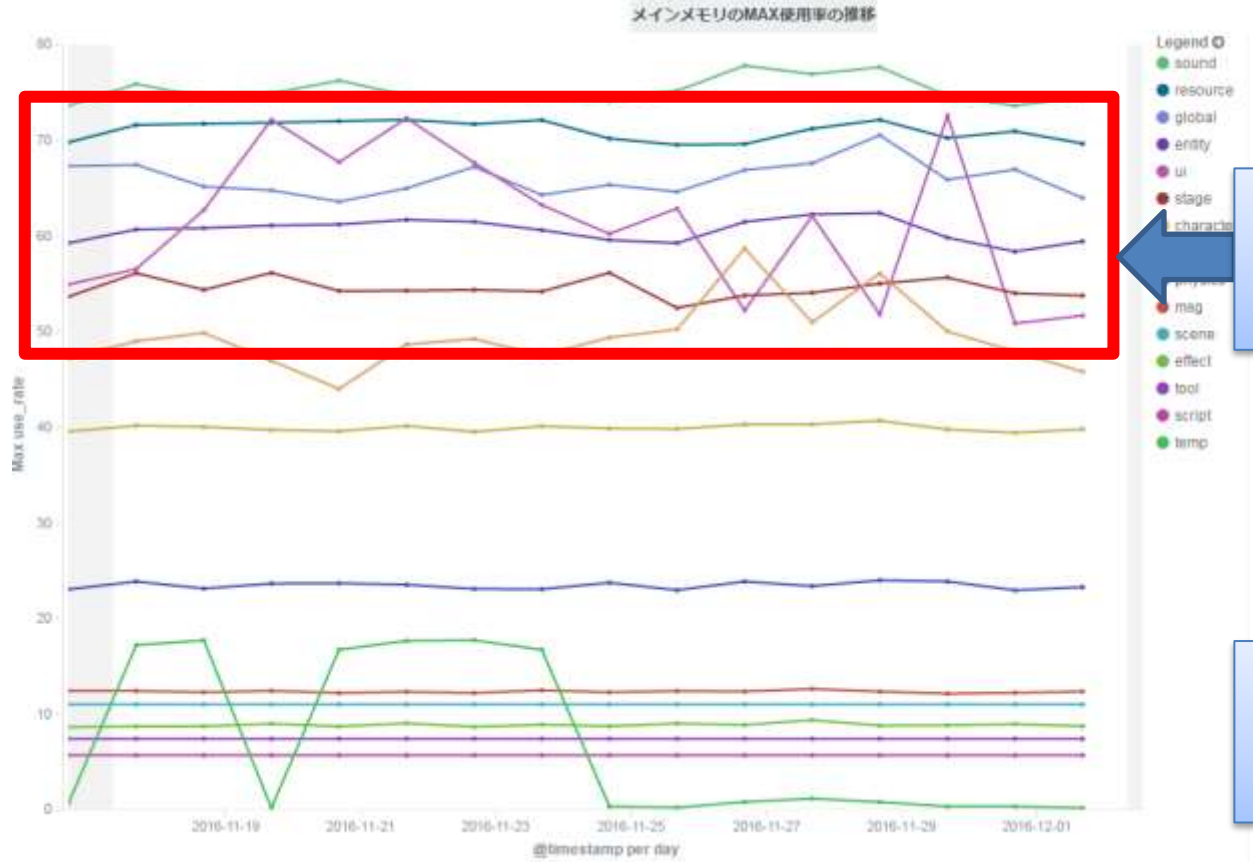
デバッグログ分析活用事例

- エラー検出
  - オートテストのログを活用
- ゲームバランス調整
  - テストプレイのログを活用
  - 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）
  - メモリ使用率（ヒートマップ）
  - コリジョン抜け
  - オートテスト結果集計
- 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

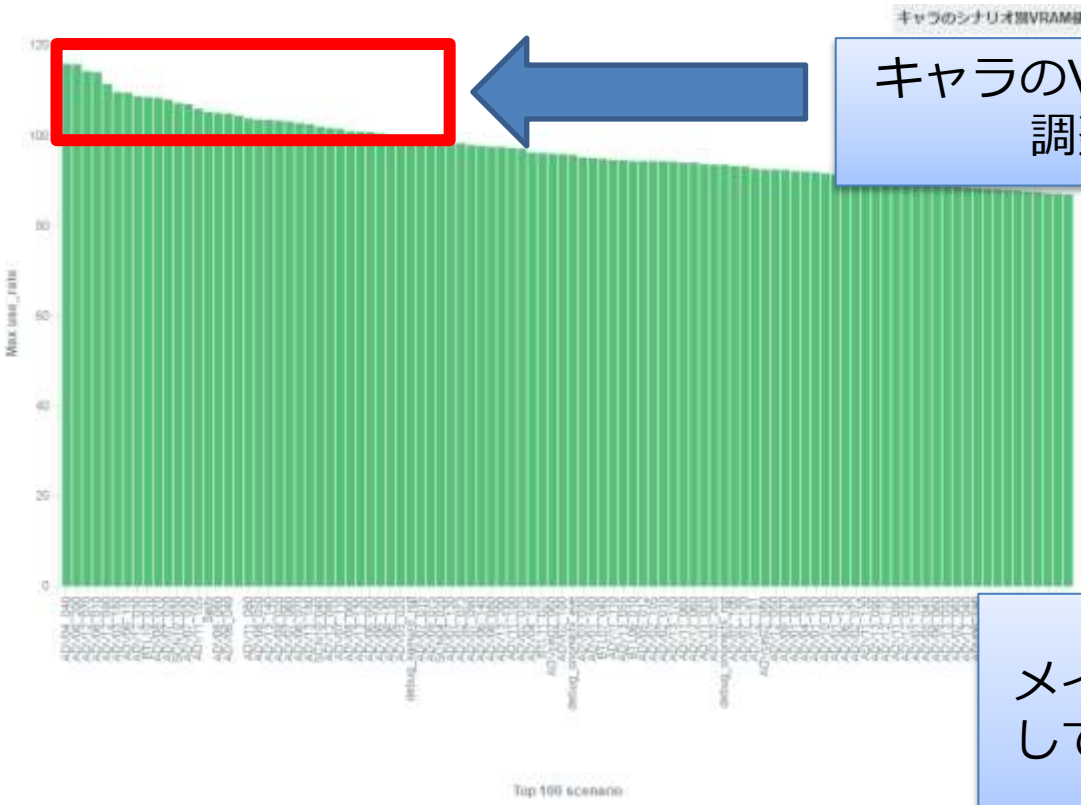
メインメモリ使用率（グラフ）



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）

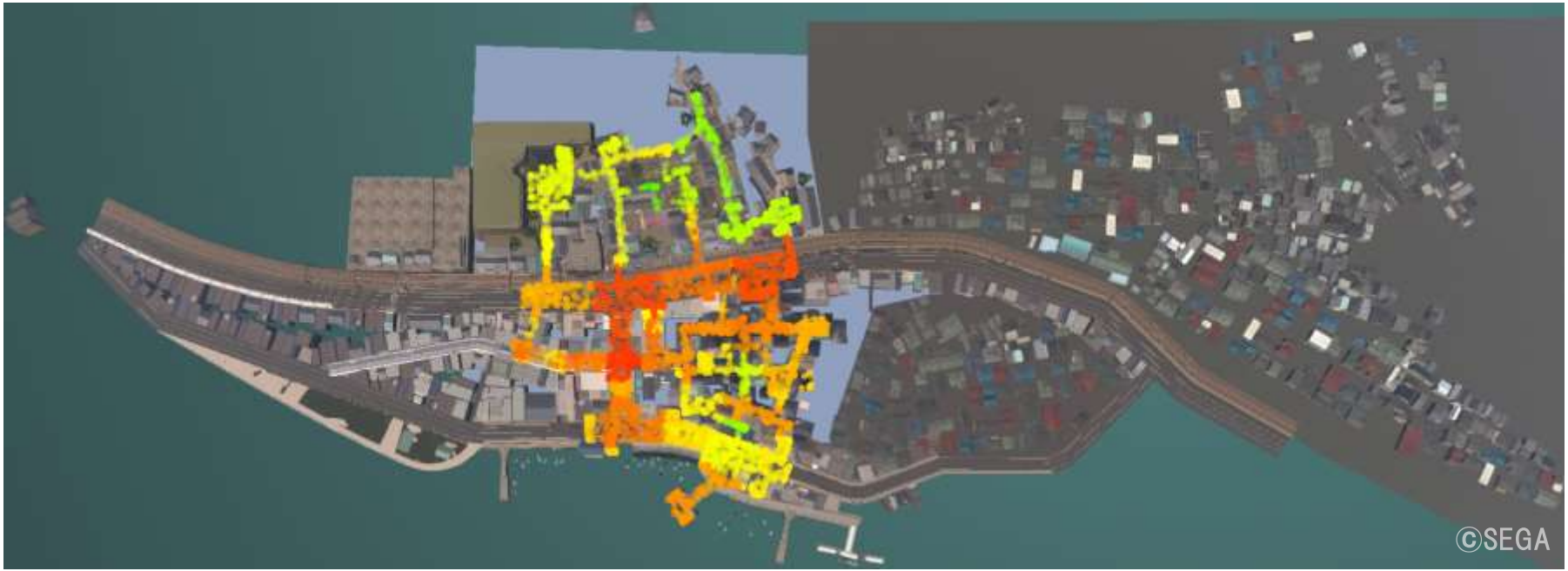


キャラのVRAMが100%を超えていて
調整が必要なシーンNo

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ーンを
計測可能

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)



背景のVRAMヒートマップ(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%を超えているところは
×で表示

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

背景のVRAMヒートマップ(修正後)

真っ赤だが、
ぎりぎり範囲内に収まった

見える化したために、
ギリギリを攻めるように
なってしまった…

コリジョン抜け

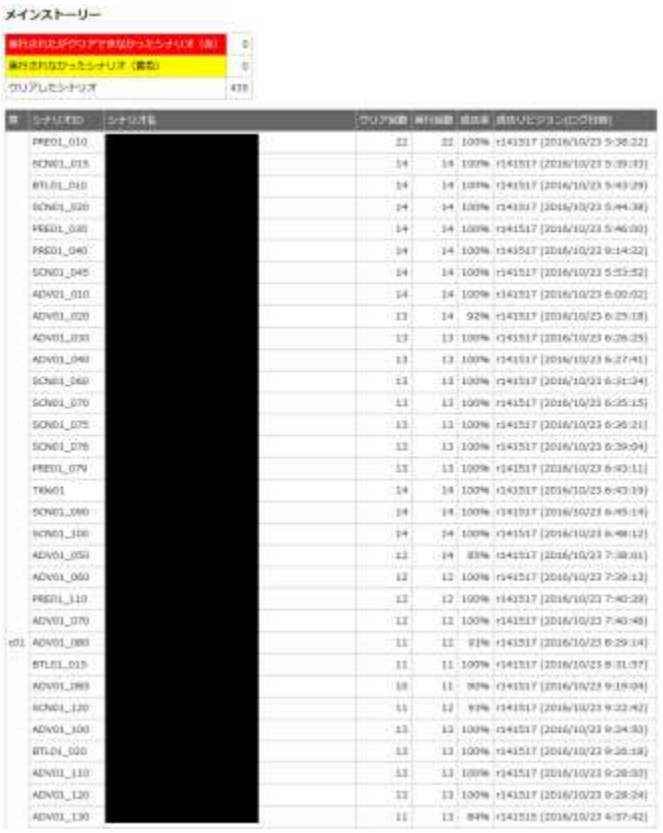
オートテストの
ランダム移動を活用

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

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
- クリアシナリオの集計に活用
- 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

オートテストのシナリオクリア結果



デバッグログ分析活用事例

- エラー検出
  - VRAM使用率グラフ
  - VRAMヒートマップ
  - コリジョン抜け
  - オートテスト結果集計
- 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

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！

テストプレイ

ログ送信

Elasticsearch

自動集計

Jenkinsで1日1回

Redmine内のWikiに公開

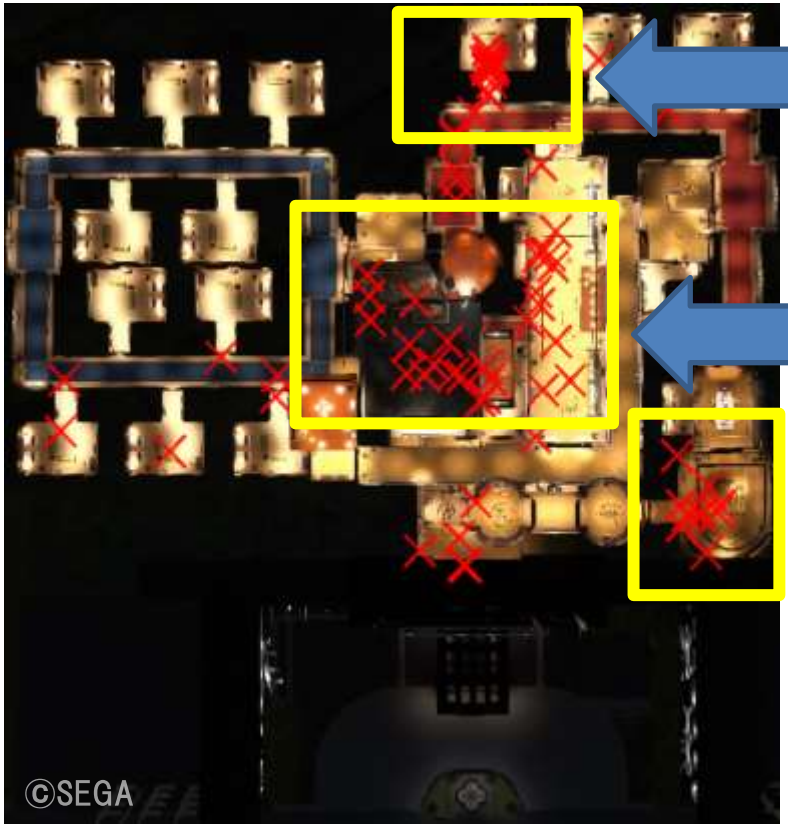
集計結果を確認

Kibanaで即確認

調整

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。

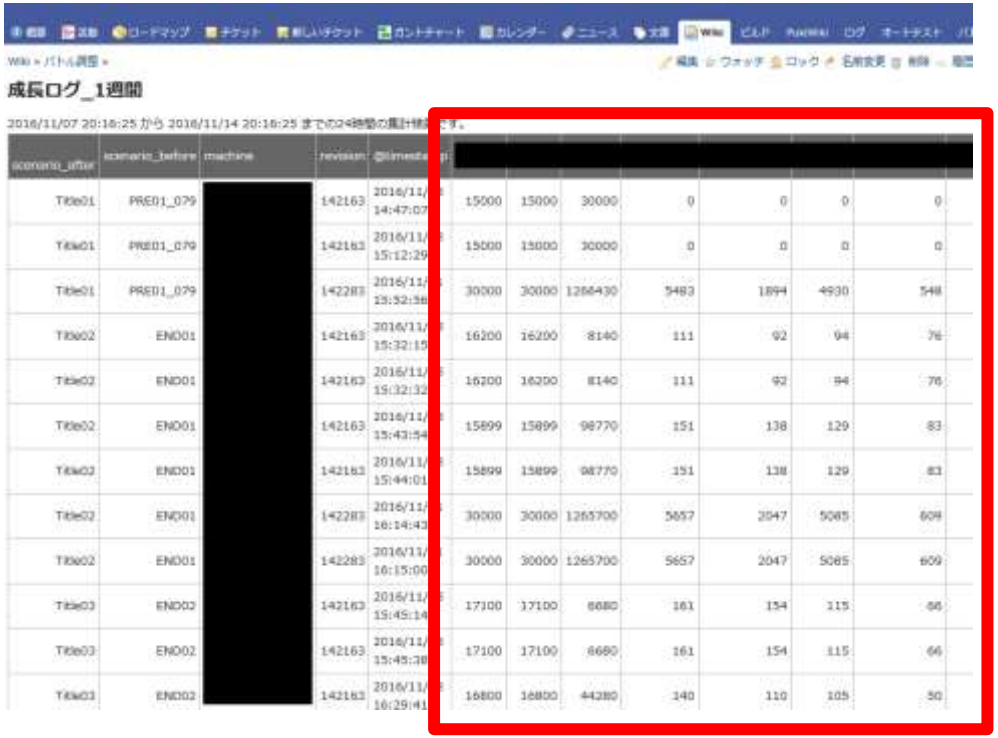
桐生が死んだマップ



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



成長ログ_1週間

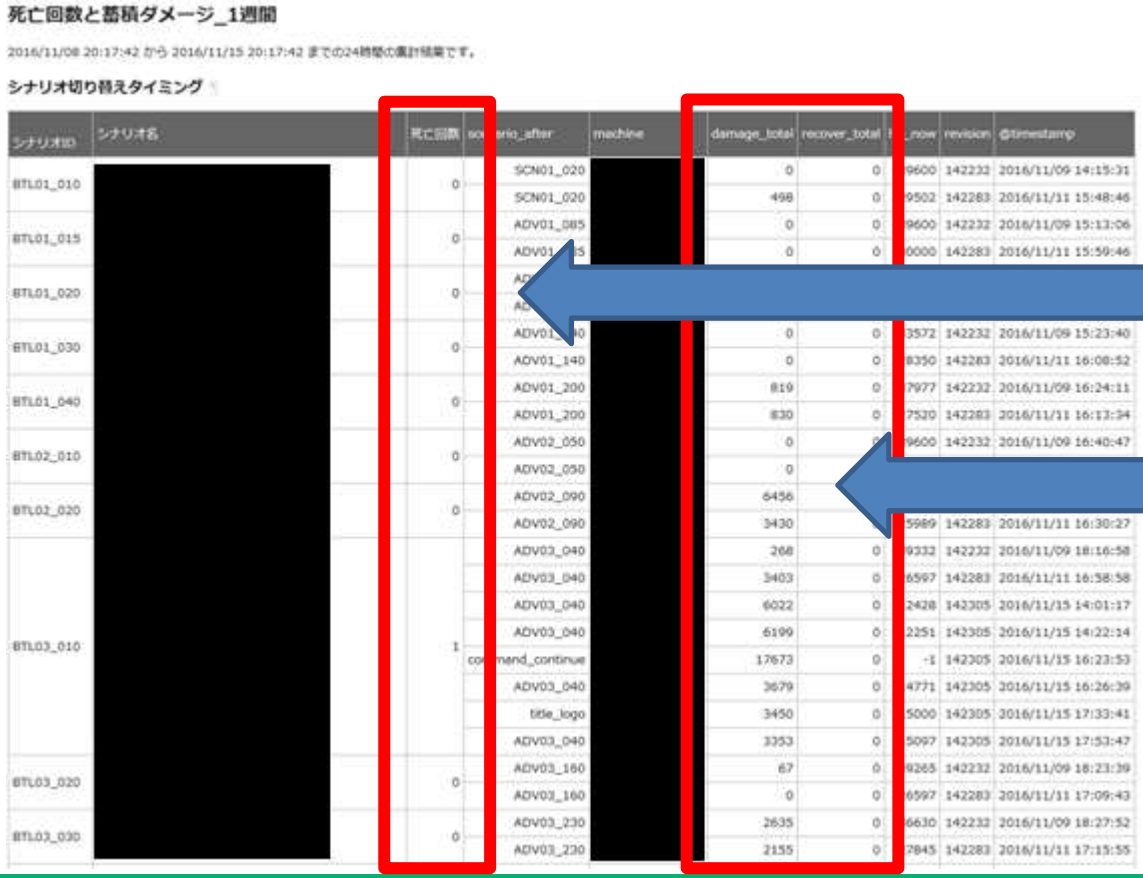
2016/11/07 20:16:25 から 2016/11/14 20:16:25 までの24時間の集計結果です。

scenario_after	scenario_before	machine	revision	@timestamp							
Title01	PRE01_079	[redacted]	142163	2016/11/[redacted] 14:47:07	15000	15000	30000	0	0	0	0
Title01	PRE01_079		142163	2016/11/[redacted] 15:12:29	15000	15000	30000	0	0	0	0
Title01	PRE01_079		142283	2016/11/[redacted] 15:52:36	30000	30000	1266430	5483	1894	4930	548
Title02	END01		142163	2016/11/[redacted] 15:32:15	16200	16200	8140	111	92	94	76
Title02	END01		142163	2016/11/[redacted] 15:32:32	16200	16200	8140	111	92	94	76
Title02	END01		142163	2016/11/[redacted] 15:43:54	15899	15899	98770	151	138	129	83
Title02	END01		142163	2016/11/[redacted] 15:44:01	15899	15899	98770	151	138	129	83
Title02	END01		142283	2016/11/[redacted] 16:14:43	30000	30000	1265700	5657	2047	5085	609
Title02	END01		142283	2016/11/[redacted] 16:15:00	30000	30000	1265700	5657	2047	5085	609
Title03	END02		142163	2016/11/[redacted] 15:45:14	17100	17100	6680	161	154	115	66
Title03	END02		142163	2016/11/[redacted] 15:45:38	17100	17100	6680	161	154	115	66
Title03	END02		142163	2016/11/[redacted] 16:29:41	16800	16800	44280	140	110	105	50

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



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	scenario_after	machine	damage_total	recover_total	_now	revision	@timestamp
BTL01_010	[redacted]	0	SCN01_020	[redacted]	0	0	9600	142232	2016/11/09 14:15:31
			SCN01_020		498	0	9502	142283	2016/11/11 15:48:46
BTL01_015		0	ADV01_085		0	0	9600	142232	2016/11/09 15:13:06
			ADV01_085		0	0	0000	142283	2016/11/11 15:59:46
BTL01_020		0	AD						
			AD						
BTL01_030		0	ADV01_140		0	0	3572	142232	2016/11/09 15:23:40
			ADV01_140		0	0	8350	142283	2016/11/11 16:08:52
BTL01_040		0	ADV01_200		819	0	7977	142232	2016/11/09 16:24:11
			ADV01_200		830	0	7520	142283	2016/11/11 16:13:34
BTL02_010		0	ADV02_050		0	0	9600	142232	2016/11/09 16:40:47
			ADV02_050		0				
BTL02_020		0	ADV02_090		6456				
			ADV02_090		3430		5989	142283	2016/11/11 16:30:27
BTL03_010		1	ADV03_040		268	0	9332	142232	2016/11/09 18:16:58
			ADV03_040		3403	0	6597	142283	2016/11/11 16:58:58
			ADV03_040		6022	0	2428	142305	2016/11/15 14:01:17
			ADV03_040		6199	0	2251	142305	2016/11/15 14:22:14
			co mend_continue		17673	0	-1	142305	2016/11/15 16:23:53
			ADV03_040		3679	0	4771	142305	2016/11/15 16:26:39
			title_logo		3450	0	5000	142305	2016/11/15 17:33:41
			ADV03_040		3353	0	5097	142305	2016/11/15 17:53:47
BTL03_020		0	ADV03_160		67	0	9265	142232	2016/11/09 18:23:39
			ADV03_160		0	0	6597	142283	2016/11/11 17:09:43
BTL03_030		0	ADV03_230		2635	0	6630	142232	2016/11/09 18:27:52
			ADV03_230		2155	0	7845	142283	2016/11/11 17:15:55

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

リアルタイムログ分析の活用事例

- テストプレイ台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

- 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
- ログサーバ運用は、特に問題なし
  - たまにFluentdとElasticsearchの接続が切れたままになる
  - 2ヶ月１回くらいの頻度でFluentdが「Too many open files」で死ぬ
    - CPU負荷が高くなるのが問題なので、FluentdとElasticsearchはサーバを分けたほうがいい
- 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（おおまか）

- 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

- 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」（カスタムログ）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
  - ファイル出力→TCP/IP通信
  - 共通項目（リビジョン、ターゲット、シナリオ、ステージ、プレイヤー座標、is_debugger/is_autotest）の高速送信
    - 値が変わったときだけ送信して、後はLogSender側で追加するようなイメージ
    - または、毎回送信はするが、ランタイム内で値はプールする

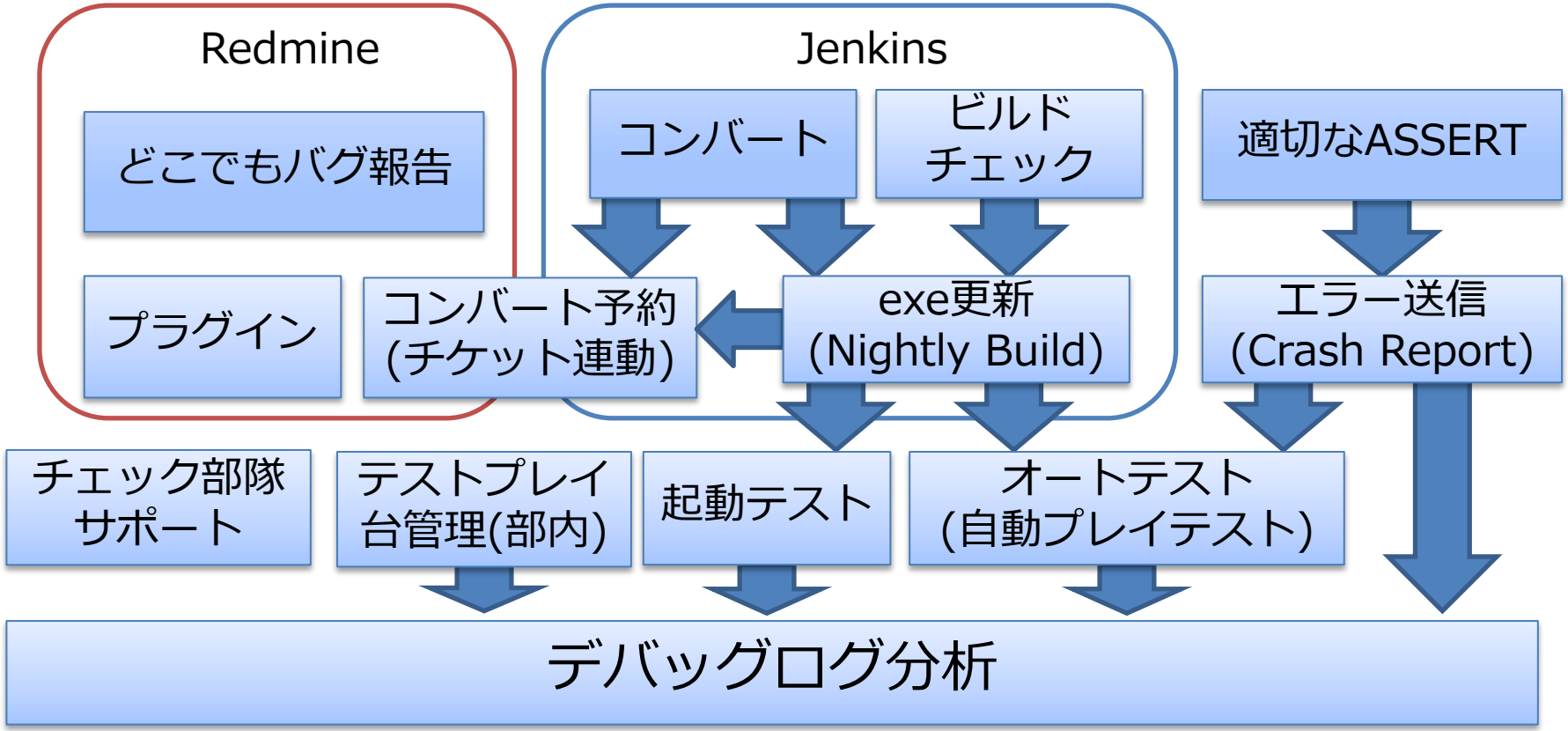
現在の進捗

- 「どこでもログ送信」を高速化
  - マルチスレッド完全対応
  - 一時バッファにためて、まとめて非同期書き込み
  - ファイル出力のままだが、毎フレーム送り続けるわけではないので問題なし
- 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
- Elastic Stackのバージョンアップ
  - Elasticsearch/Kibanaは、5.X系を利用

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。

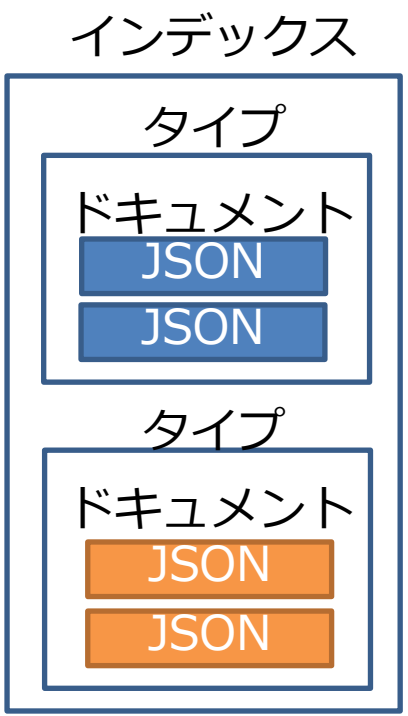
参考資料．書籍・スライド資料

- ログ分析
  - 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/石井葵 著）
- 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．龍６での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．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。