

# 人海戦術からAI戦術に！ テストイングソリューション「Playable!」が目指す次世代QAスタイル

**Playable!**  
Advanced Game Testing Solution

# ソリューション・コンセプト

## ゲームQAをより高度に、より迅速にし、 クリエイターの挑戦の機会を増やす

Playable!は、ゲームQAに掛かる多くの時間、判断をコンピュータ・AIに代替させ、人間はより高度で創造的なQAにリソースを使うことにより、ゲーム開発の多くの部分を占めるQAを高度化・効率化します。

Playable!開発チームはゲームAIに関する研究部門を擁し、自分たち自身の研究、あるいは世界中のQA、ゲームに関する最新の論文や事例発表から着想を得た「ゲームQA」x「自動化」をソリューションとして表現、提供してまいります。

# ソリューション・コンセプト

ゲームQAをより高度に、より迅速にし、  
**クリエイターの挑戦の機会を増やす**

Playable!は、レベルやパラメータ変更によるゲームタイトルへの影響をクリエイターにすばやくフィードバックすることで、ローンチまでの限られた時間におけるクリエイターの挑戦、試行錯誤の機会を増やすことに貢献します。

## Playable! Collision Check

ゲーム内に存在するPCの侵入不可地点(壁等)に対して、角度や移動方式によって侵入を許してしまう地点がないか、あるいは意図しない奈落落ちが発生しないかを総当たりで検出。

ツールによって予めマップを解析、および衝突試験を施行することで、コリジョン抜け、奈落落ちが懸念されるポイントを絞り込みます。人手による任意の総当たり試験に比べて劇的な効率向上が期待できます。

また、一般的な操作ではPCがその場から動けなくなってしまう、いわゆる「ハマリ」ポイントの検出にも対応しています。





## Playable!

# Playthrough

ゲームをクリアするための最小限のイベントフラグを定義し、そのフラグを攻略するお手本を作成。お手本ファイルを参考にAIがゲーム開始からエンディングまでプレイします。

ゲームタイトルが、スタートからエンディングまで間違いなく完了できることを継続的かつ省リソースで検証可能。継続的インテグレーションにも活用できます。また、シナリオを増幅させることで、実際の人間によるプレイのように幅をもたせることも可能です。

統計  
合計

## Playable! ItemCollector

特定の領域内に配置されているアイテム全数をスキャンし、それぞれのアイテムがPCの移動/特殊移動によって本当に回収可能であるかを検証。特定領域におけるアイテム配置の不具合検出、および、アイテム図鑑等収集要素の検証等に活用可能です。

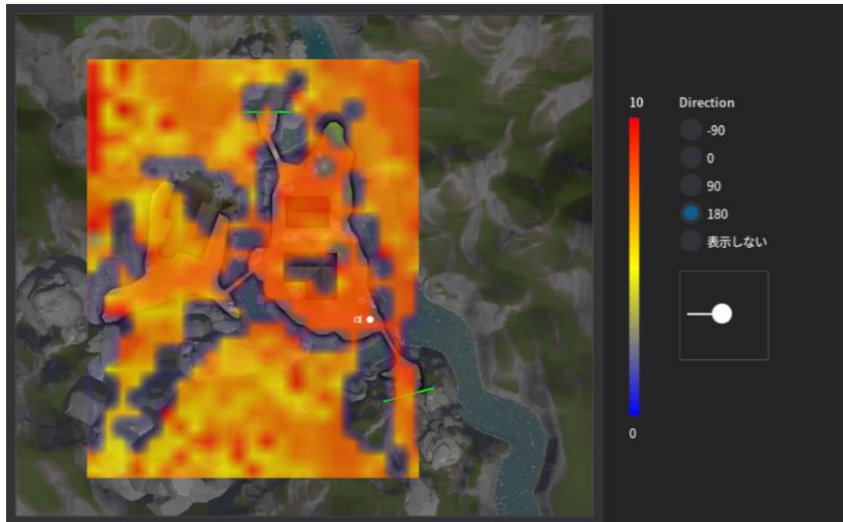
また、本ツールは「ひたすらXX」系のタスクのインスタンスです。同様の種別のテストタイプにも対応可能な場合があります。

✓ フィールド移動で回収 ✓ デバッグ移動で回収 ✓ アイテム未入手 ✓ 回収失敗



# Coming soon & Challenge !

オートプレイとエージェントを組み合わせ、マップの各所、各アングルでのリソース負荷を自動計測

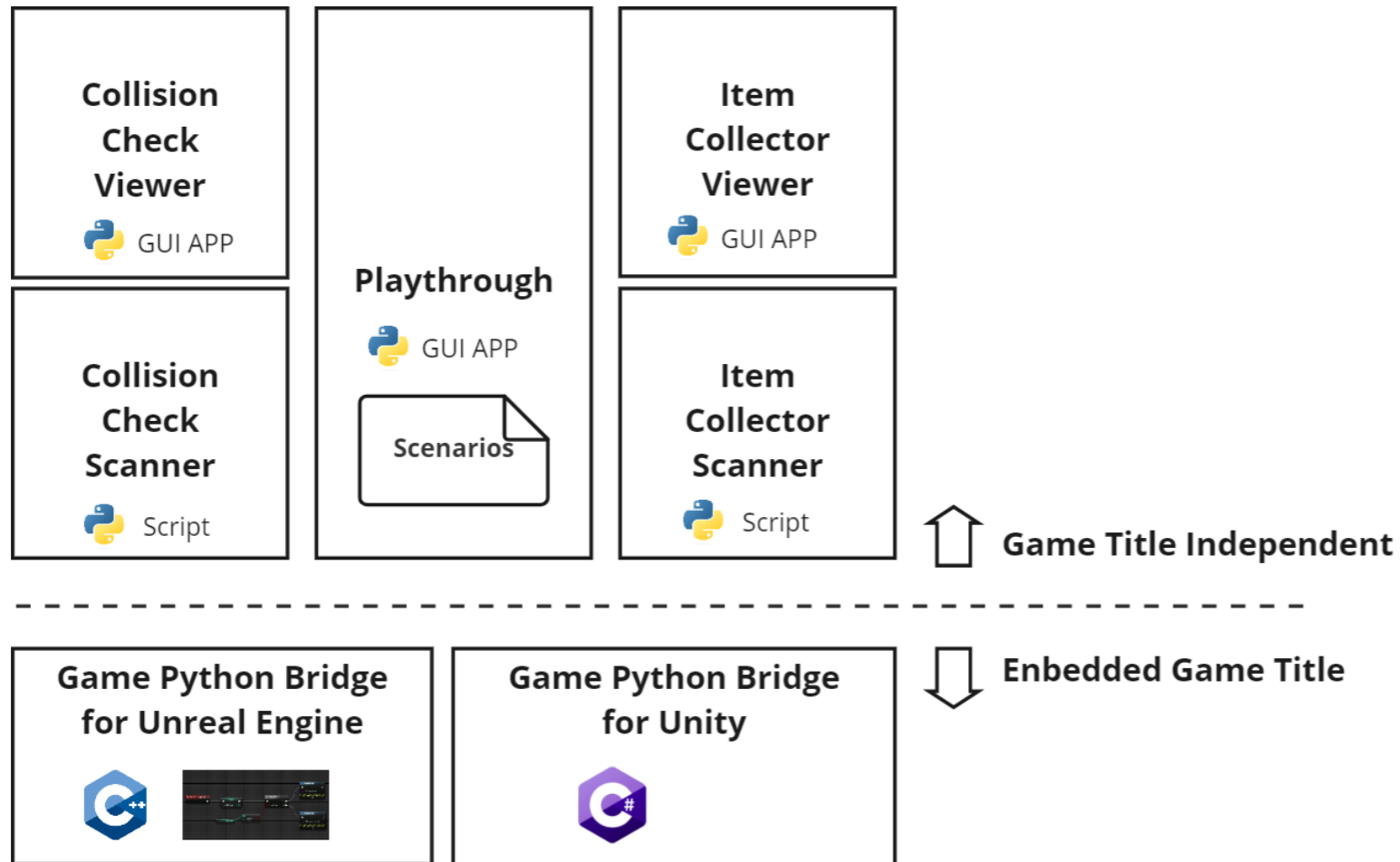


AI自然言語処理によるビヘイビアツリーの動的な生成により、さまざまなタスクをこなす「汎用エージェント」





# 製品アーキテクチャ



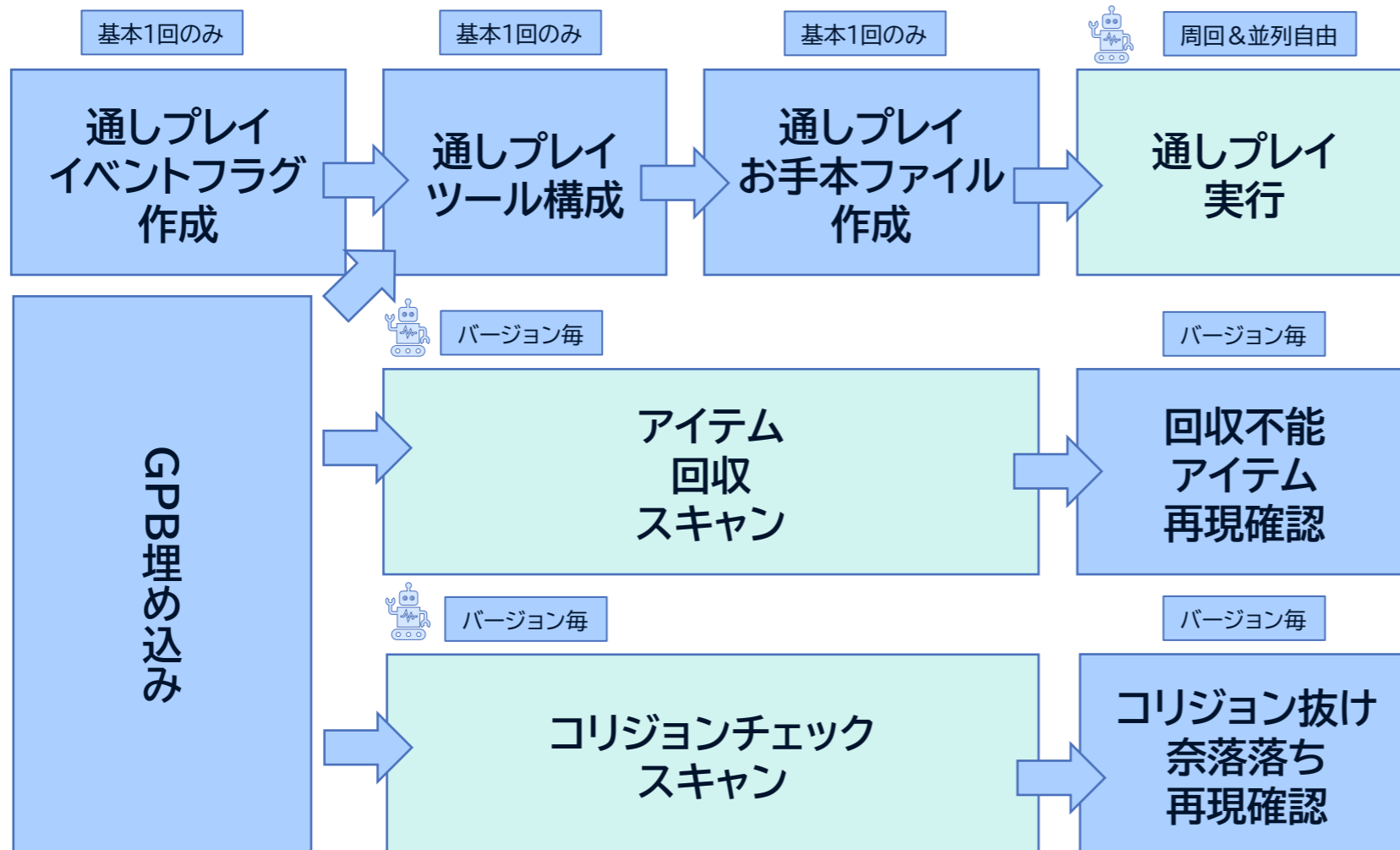
## タイトルへの影響を最小限に。

Playable!は独自開発のゲームI/Oライブラリ「Game-Python-Bridge」を組み込むことにより、ゲームからの最小限の情報の読み取り\*1(PCの位置情報、カメラの向き等)、ゲーム側への最小限の情報送信(コントローラーの操作情報のみ)にて、ゲームQAに不可欠ないくつかのテストの実行を完全自動化します。

※1.「ゲーム画面のみを入力として自動プレイ」を行う汎用的な技術は、私達の観測範囲では、まだ確立されていません。私達も積極的にこの技術の確立と提供を目指してまいります。



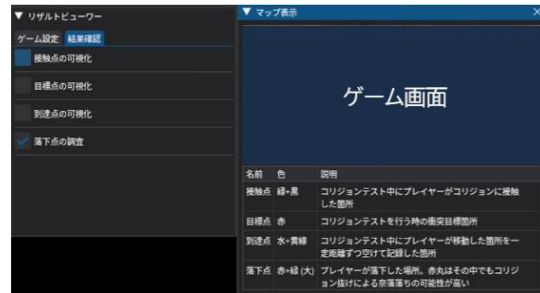
# モデル業務プロセス



Playable! 導入時に想定される基本的な業務プロセス例です。赤い四角が実際にテストを実行している部分となります。

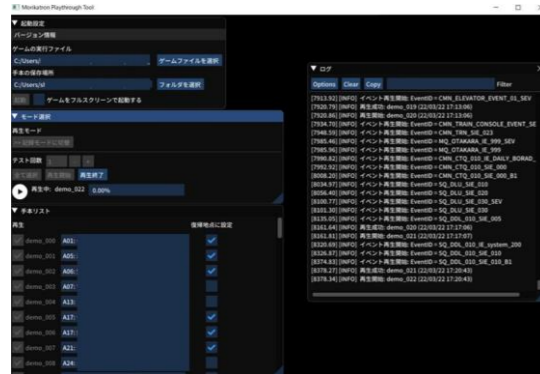
左図は現在提供されている3つのツールの運用例となります。今後追加されるツールについては個別にご案内を差し上げます。

# モデル業務プロセス



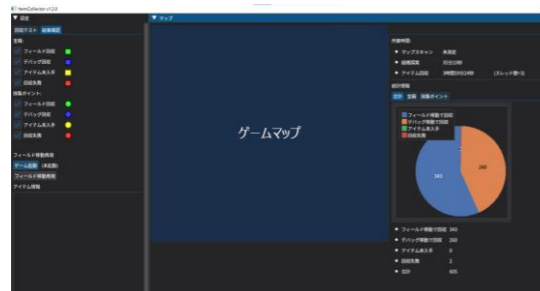
## コリジョンチェックのテスト結果

タイトル、品質状況によりますが、概ね1回に100件~200件程度の疑わしいポイントを検出します。プロセスにより、そのまま開発チームに渡す場合と、テストチームで再現確認を行って起票する場合があります。また、修正後に同条件でチェックを行い、改修確認を実施します。



## プレイスルーのテスト結果

実行したシナリオすべてのログを確認します(ログの内容はゲームエンジンによります)原則、選択したシナリオがすべて問題なく完了していれば、通しプレイ1周としてはPASSとなります。



## アイテムコレクターのテスト結果

経路探索によるアイテム回収、デバッグ移動によるアイテム回収、どちらのルートでも失敗の3種類の結果が表示されます。デバッグ移動、回収不可についてはバグである可能性が高いため、再現確認を実施します。

# FAQ

Q1. Game-Python-Bridgeの組み込みにはどれぐらいの期間がかかりますか？

A1. ゲームの規模によらず、概ね2週間～1ヶ月程度で完了します(採用されているゲームエンジンにもよります)

Q2. コリジョンチェック、アイテム回収にはどれぐらいのコンピューティングリソースが必要になりますか？

A2. 2022年にSwitchで発売されたAAA級のアクションRPGタイトルにおいて、コリジョンチェックを一周するために要したコンピューティング費用が概ね25万円程度です。諸条件によって変わります。

Q3. 組み込みライブラリはUEとUnityに対応とありますが、自社エンジンでも利用可能ですか？

A3. はい、ゲームから取得する情報の種類はわずかですので、御社エンジニアの方と連携しつつ、組み込み可能です。ライブラリが組み込めれば上位層のツールは動作します。

# 出展・メディア掲載

東京ゲームショー2022

週刊ファミ通 2022年12月8日号  
西川善司の「ゲームのムズカシイ話」



**FAMITSU COLUMN**

**■AIならではの並列デバッグ処理**

たとえば、「人間のテスターがひとりで9台のマシンを使って別々のシーンを並行してデバッグする」という種別は到底できるものではないが、AIはそれをやってのける。右の画面は、本文で取り上げているゲーム世界における「衝突判定チェック」(Collision Check)を9台の仮想マシンを使い、並列で実践している様子。

●AIデバッグの便利ところは、文字通りの「放置プレイ」ができること。人間はマシンの前にいる必要すらなく、テスト結果を待つだけでいいのだ。

えるべきだと考える「人間では時間が掛かりすぎる/人間がやるには非効率すぎる」デバッグ工程は3つあるという。ひとつは、広大なゲーム世界におけるオ

のか……というチェックだ。ゲームをプレイしていて「あの崖っぷちに置いてある宝箱、本当に取れるの?」と思ったことはないだろうか。もし、それが実際に回収不可能だった

ェックしてくれる「Playthrough」モジュールだ。こちらは、ゲームがある程度の完成状態になってから、「バランス調整」や「地形調整」といった最終的な修正を行った際に、全

ほか多数



# Playable!は本日よりお試し可能です

Playable!サービスサイトはこちら！

<https://playable.qa>

ユーザ登録のうえ、お試しく下さい！



# Playable!は本日よりお試しいただけます

## オススメのステップ

### STEP1

Playable!  
が  
組み込み済みの  
サンプルゲームで  
自動テストの実行の  
様子を確認する



### STEP2

素のサンプルゲーム  
プロジェクト(UE4)に  
対してPlayable!  
を組み込み、手順を  
確認する



### STEP3

実プロジェクトへ  
組み込む!

# Playable!は本日よりお試しいただけます

## 組込み済みサンプルゲームで動作を確認

最初に、Getting Started でソリューションの概要を確認

例えば Playthroughを試したい場合、サイドバーから当該ツールを開き、操作マニュアルを確認

**対応の流れ**

新規ゲームの自動テスト対応を行う際は大体か以下のような流れで進めます。

1. ゲームプログラムの改造
  - プラグインの導入
  - ゲーム内情報を送信する処理の実装
2. 各自動テストツールの対象ゲーム対応
3. 各自動テストツールによる自動テストの実行

各ツールごとの対応の流れを示したものが下図になります。

ゲームプログラム側	Map Scanner	Playthrough Tester	Item Collector	Collision Checker
ゲーム改造 (プラグインの導入)				
ゲーム改造 (対象ゲーム対応)				
ゲーム改造 (Playthrough Tester対応)	対象ゲーム対応			
	マップスキャン	対象ゲーム対応		
			対象ゲーム対応	対象ゲーム対応
		自動テスト	自動テスト	自動テスト

**操作マニュアル**

**ツールの起動**

1. ドキュメントサイトよりツールをダウンロードし、展開します。
2. ツールディレクトリ直下で、python playthrough.py を実行します。

ツールが起動すると以下のウィンドウが表示されます。

One More Thing ...

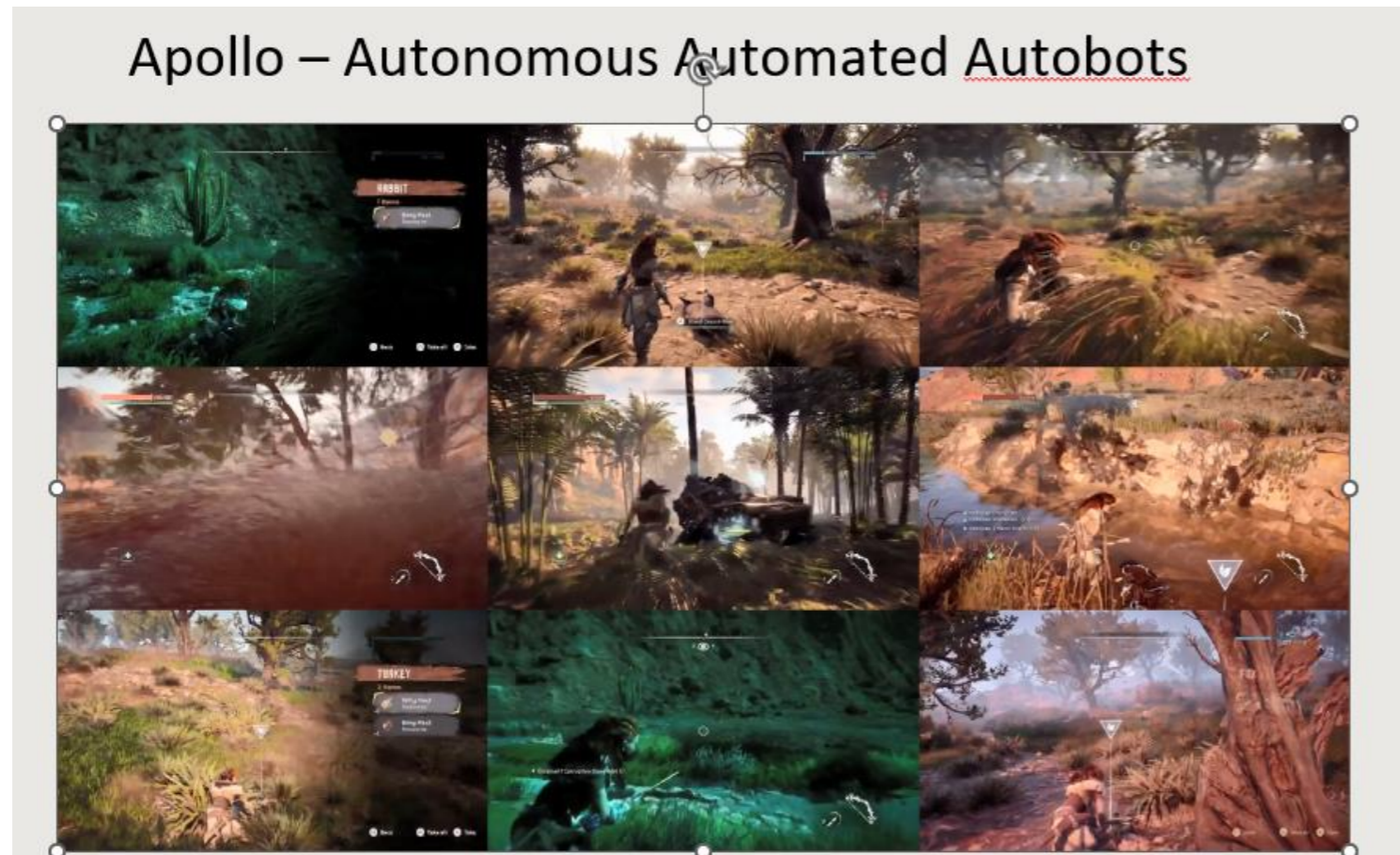


# 汎用エージェント、動き出しました

The screenshot displays the Playable! Chat Gameplay Agent (Ver. 0.9.3a) interface, which is divided into several panels:

- Gameplay AI Chat Window:** Shows a conversation between the AI (PI) and the user. The AI provides instructions and status updates based on the user's commands.
  - AI: あなたの代わりに私がゲームを操作します 私にやってほしいことを教えてください
  - AI: その前に... まずはゲームを操作可能な状態にしますので少々お待ちください
  - AI: ゲームが操作可能な状態になりました 指示をどうぞ
  - User: 平原街道へ移動して
  - AI: 平原街道(PL\_030GRP)に行きます
  - AI: タスク「平原街道(PL\_030GRP)に行きます」を完了しました 現在の予定されているタスクはありません 指示をどうぞ
  - User: ポーションを1つ回収してからダンジョンへ行って
  - AI: ポーション(HealPotion)を1個回収します 壱蔵(PL\_050GRH)に行きます 以上のタスクを登録しました 現在のタスクは
    - 1: ポーション(HealPotion)を1個回収します
    - 2: 壱蔵(PL\_050GRH)に行きます
 です
  - AI: タスク「ポーション(HealPotion)を1個回収します」を完了しました 残りのタスク:
    - 1: 壱蔵(PL\_050GRH)に行きます
 未実行のタスクがあるので、続けて「壱蔵(PL\_050GRH)に行きます」を実行します
- Game Map Window:** Displays a top-down map of the game environment with various locations labeled, such as 'ルーン村', '守り岩の洞', '平原街道', and '壱蔵'.
- Behavior Tree Window:** Shows a visual programming interface for the AI's behavior. It features a 'node editor' with various nodes like 'JsonAgent', 'ExnTreeSuccessEnd', 'ExnContinuousCloseMsg', 'Wait200FrameEnd', 'ExnT', 'CloseMsgQuickly', 'MoveToParentMap', 'MoveToParentPosFailureReset', 'MoveToParentPosFailureReso', 'MovingOverObstacles', 'MoveToParentPos', 'PatrolCollect', 'SceneTransferHitBox', and 'MoveToParentPos'.
- Game Play Window:** Shows a first-person view of a character in a dark, gothic-style dungeon. The character's health (HP) is 100/100. The interface includes a '回避' (Evasion) button, '回復アイテム使用' (Use Recovery Item), 'カメラリセット' (Reset Camera), and 'メニュー' (Menu) options.

# 余談：海外AAAタイトルでのQA



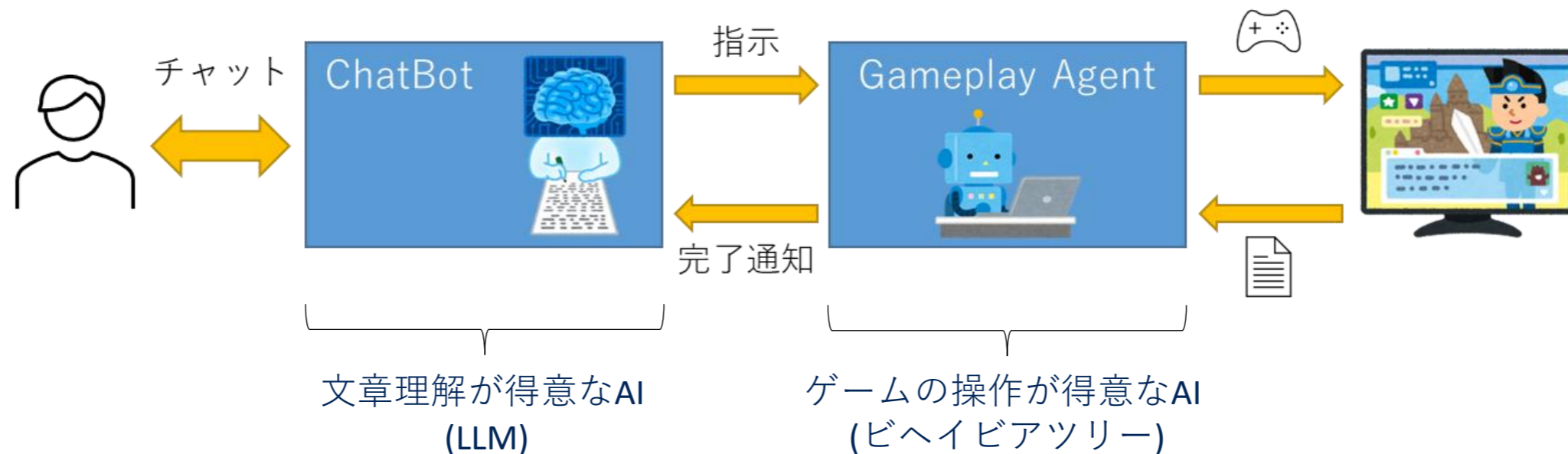
海外AAAタイトルの開発では、QAチームが無数のボットをオーケストレーションし、アサートやログを解析する、というアプローチが取られています

今回、わたしたちは  
**その斜め上のアプローチ**  
を実現しました

GDC2018, Ana\_Barbuta, Horizon Zero Dawn: An open world QA case study

# 汎用エージェントとは？

Playable!のシリーズのひとつ。人間の自然言語の入力によりテスト対象のゲームを操作し、結果を返します。自然言語の解釈にLLMを活用し、独自開発した変換エンジンと協調することによりビヘイビアツリーを動的に生成します。



# 汎用エージェントは ゲームQAにとって何が嬉しいのか？

- 人間の手を介さず、「文字列」でゲームを操作できるため、例えば1人の指揮により、100台のエージェントをいっぺんに動作させる、といった超並列テストが可能になる
- 特定のタスクを達成させるための手順をAI自身に考えさせることで、抜け道の発見や難易度調整をある程度自動化することができる
- 既存のテストケースのテスト手順と期待動作を入力して、スクリプティング無しでテストを自動化することができる

夢ありすぎ



# 汎用エージェントの課題

- アイテムやNPC、エネミー、建造物の名称や大まかな出現位置は予め教えておく必要がある  
→ ゆくゆくはゲームデザインドキュメントをそのまま入力して解釈できるようにする
- ある特定のタスクが「完了」したことを現状はゲームタイトル側からコールバックする必要がある  
→ インベントリやステータスの情報をPlayable!からポーリングすることでコールバックを不要にする

課題はありますが、どれも解決の道筋がついています

# 汎用エージェント

AIQVE ONEブースでムービー公開中！  
ぜひ、お立ち寄りください！

Playable!サービスサイトはこちら！



資料ダウンロードいただいたかたへのおまけ  
(当日お話していない)  
ゲームのテスト設計のチャレンジ

# ゲームのテストアーキテクチャ



# ゲームのテストアーキテクチャ

テトリスで例えると...



## インゲーム:コア

AIデバッグ  
(帰納的なテスト)

- ・どれぐらいの速度までラインを消し続けられるのか
  - ・規定のゲームクリアまで到達できるか
  - ・T-スピンによるライン消去の処理に不具合はないか(さまざまな状況におけるT-スピン)
  - ・対戦の場合、勝ち確パターンが存在しないか
- などなど

## インゲーム:ゲームメカニクス

テスト設計&MBT  
(演繹的なテスト)

- ・テトリミノは法則に従って無限に供給される
- ・テトリミノは一定タイミングで落下する
  - ・落下速度は進行に従って向上していく
- ・テトリミノは積み上がる
- ・テトリミノが横一列そろったらその列は消える
- ・四列いっぺんに消すことを特別に「テトリス」と呼称する
- ・一部のテトリミノは特殊な回転が行える

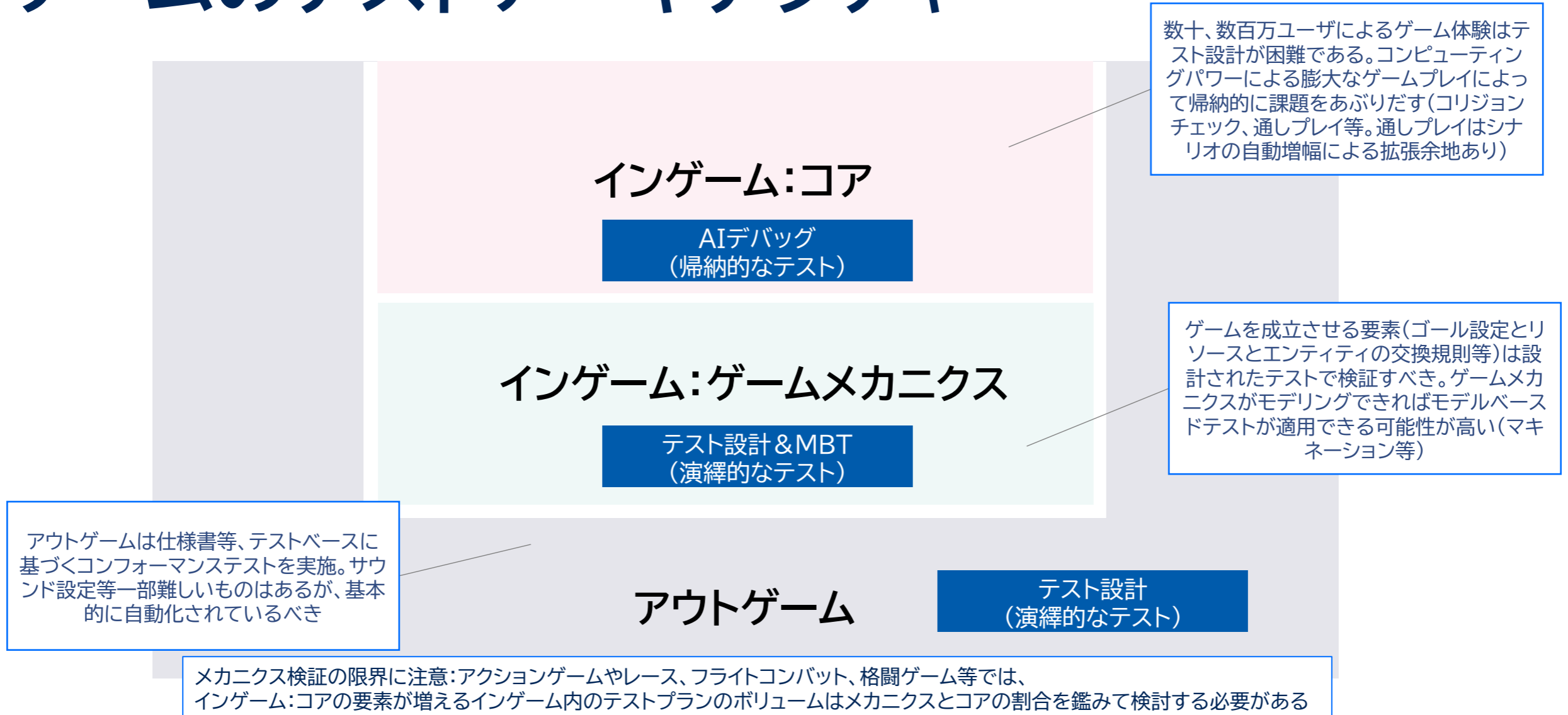
## アウトゲーム

テスト設計  
(演繹的なテスト)

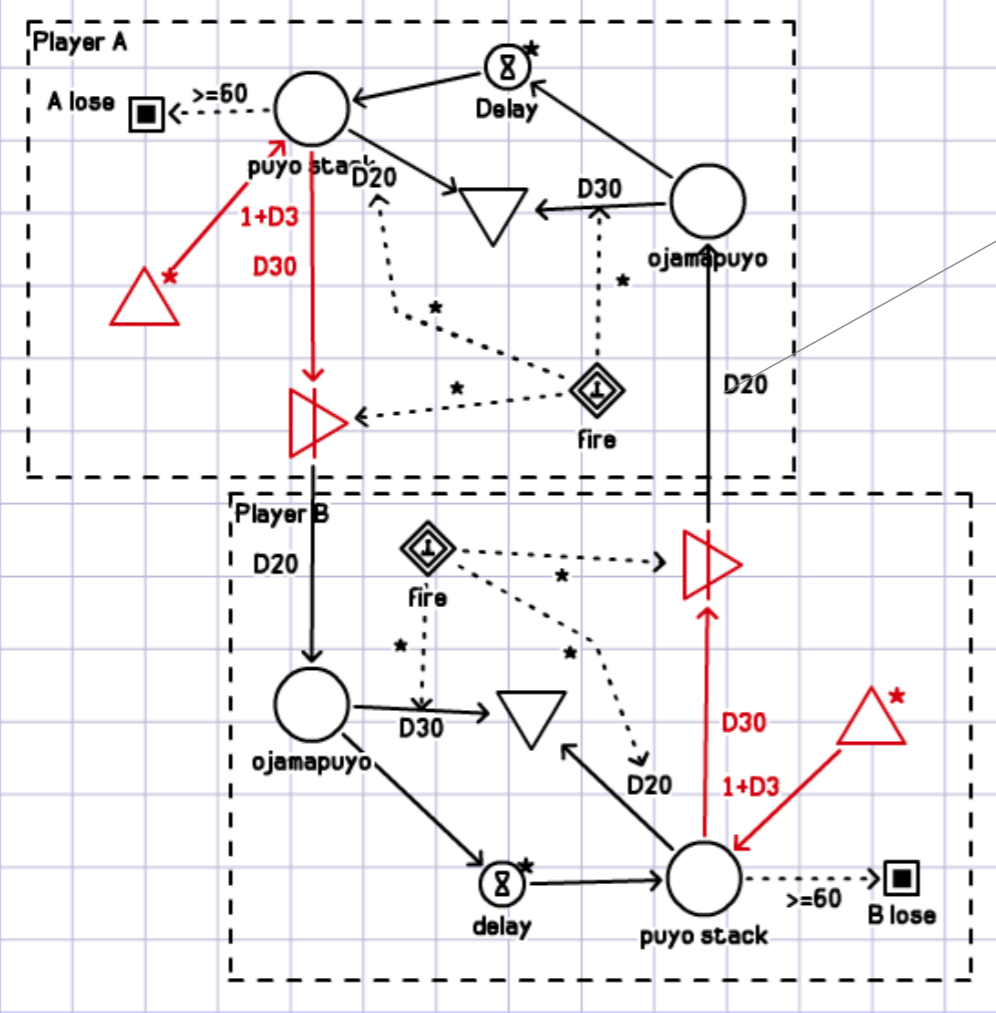
オプション、開始時レベル設定、  
エンドレス有無、など



# ゲームのテストアーキテクチャ



# ゲームメカニクスのMBT



パスが提示されている時点で、状態遷移テストのNスイッチカバレッジは適用可能。

単純なルールの確認はデシジョンテーブルも利用できる

トリガー項目の発火のさせかたは増幅が効く

csvダウンロード

	有効/無効	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
条件		1	2	3	4
i mod 3 == 0	+	Y	N	N	N
i mod 5 == 0	+	N	Y	N	N
i mod 3 == 0 && i mod 5 == 0	+	N	N	Y	N
動作					
buzz		X	-	-	-
buzz		-	X	-	-
fizzbuzz		-	-	X	-
number		-	-	-	X

GIHOZ側のモデルで振る舞い変更されたので、これと連動してテストがコケます。ポイントは、GIHOZ側のモデルを変更しただけで、テストコードは一切触っていない、点です。TestOpsばいですね。

```

test session starts
platform win32 -- Python 3.6.7rc2, pytest-6.2.5, py-1.11.0, pluggy-1.0.0
rootdir: C:\project\gihozops
collected 4 items

gihoz_testops.py F... [100%]

===== FAILURES =====
test_fizzbuzz[3-buzz]
num = 3, expect = 'buzz'
@pytest.mark.parametrize("num, expect", test_cases)
def test_fizzbuzz(num, expect):

```

GIHOZのデシジョンテーブルからMDTしている例

出典：<https://yosukem.hatenablog.com/entry/2013/07/02/103009> (2023.3.24閲覧)