事前条件と事後条件に着目した First Person Shooting Gameに対するキーワード駆動テストの提案

電気通信大学　大学院
情報理工学研究科
菊池文矩　西康晴

ゲームの開発規模の増大

記録媒体の容量の増加や計算機の性能向上に伴い
ゲームの開発規模が大きくなっている

表1：一般的なアクションゲームの開発費の推移

世代	発売年	予算	工数(人月)	実期間
PS	1994	2億	350	6ヶ月
PS2	2000	7～8億	1000	2年
PS3	2005	14～16億	2000	3年

出典：デジタルコンテンツ制作の先端技術応用に関する調査研究(2010)[2]

テスト工程の分離

テストにかける時間が増え、
ゲームを開発しているプログラマだけではテストを
行うのが難しくなった

テストエンジニアがテストを単独の業務として行うようになった

テストエンジニアは仕様に関する知識はあるが、
実装の知識は求められず、コーディングスキルも必要ない

コーディングスキルのないテストエンジニアが
ゲームのテストを行っている

テスト自動化の必要性

ゲームの品質管理のために多くのテストが行われる
多くのテストが人力で行われている

- 通しプレイ
  - ツールを使わずに通常と同じ環境でプレイしゲームがクリア可能であることを確認する
- コリジョン抜けのチェック
  - 床や壁などに当たり判定の張り忘れがないかを実際にキャラクターをぶつけていくことでテストする

テストを人力で行うのは多くのコストが
かかるためテストの自動化が進められている

テストの自動化の難しさ

Webシステムのログインテストを行うスクリプト
（Selenium IDE[4]にて作成）

```
def test_login(self):
    driver = self.driver
    driver.get(self.base_url + "/")
    driver.find_element_by_link_text("server.py").click()
    driver.find_element_by_link_text("html/").click()
    driver.find_element_by_name("username_field").clear()
    driver.find_element_by_name("username_field").send_keys("菊池")
    driver.find_element_by_name("password_field").clear()
driver.find_element_by_name("password_field").send_keys("pass")
    driver.find_element_by_name("login_button").click()
```

問題点：
- コーディングができなければ
  テストの自動化を行うことができない

コーディングスキルのないテストエンジニアが
自動テストを行う事は困難である

データ駆動テスト

Webシステムのログインテストを行うスクリプト
（Selenium IDE[4]にて作成）

```
def test_login(self):
    driver = self.driver
    driver.get(self.base_url + "/")
    driver.find_element_by_link_text("server.py").click()
    driver.find_element_by_link_text("html/").click()
    driver.find_element_by_name("username_field").clear()
    driver.find_element_by_name("username_field").send_keys(name)
    driver.find_element_by_name("password_field").clear()
driver.find_element_by_name("password_field").send_keys("pass")
    driver.find_element_by_name("login_button").click()
```

外部ファイルから入力データを読み込む

name
“伊藤”
“小沢”
“菊池”

入力データを外部に切り出すことで、同じスクリプトで
違う入力を行うテストを実行することができる

キーワードによるテストケースの記述

Webシステムのログインテストを行うスクリプト
（Selenium IDE[4]にて作成）

```
def test_login(self):
    driver = self.driver
    driver.get(self.base_url + "/")
    driver.find_element_by_link_text("server.py").click()
    driver.find_element_by_link_text("html/").click()
    driver.find_element_by_name("username_field").clear()
    driver.find_element_by_name("username_field").send_keys("菊池")
    driver.find_element_by_name("password_field").clear()
    driver.find_element_by_name("password_field").send_keys("pass")
    driver.find_element_by_name("login_button").click()
```

スプレッドシートに記述したテストケース

keyword	target	value
テキスト入力	ユーザネーム	"菊池"
テキスト入力	パスワード	"pass"
クリック	ログイン	"login"

ユーザの操作内容も外部ファイルに記述することで、
簡単なキーワードを使って違う操作内容を行うテストケースを
スクリプトを編集することなく作成することができる

キーワード駆動テスト(KDT)

**コーディングスキルのないテストエンジニアが
テストケースを作成するための
既存手法としてキーワード駆動テスト(KDT)が
Buwaldaら[5]によって提案されている**

自然言語の命令を「キーワード」と呼ぶ

キーワードを並べることで自動テストを作成する

FIRST PERSON SHOOTING GAME

**First Person Shooting(FPS)は最も開発規模の
大きいゲームジャンル**

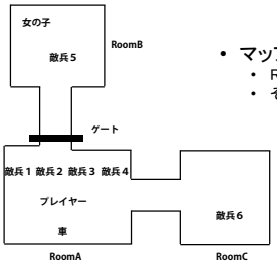
- FPSのタイトルの一つである「Call of Duty:Modern Warfare 2」は
  開発費が約200億円[3]

出典: http://chaos-lab.jp/archives/597「counter-strike1.6」

FPSに対するKDTの適用例

**サンプルのFPSに対して既存のKDTを適用した例を示し
既存手法の問題点を説明する**

サンプルのFPSのマップ



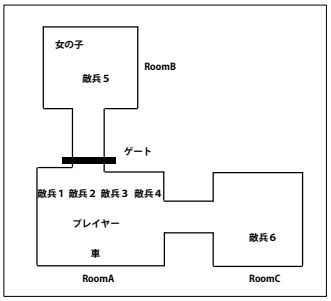
サンプルFPSのマップ

- マップは3つのエリアに分かれている
  - RoomA,RoomB,RoomC
  - それぞれのエリアに敵が配置されている

テストケースの例

テストケースの例

キーワード	ターゲット
移動	RoomA
攻撃	army1
攻撃	army2
移動	RoomC
攻撃	army6

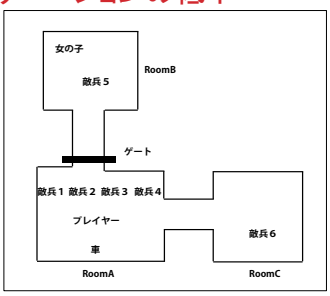


左の表はキーワード駆動テストを用いて作成したテストケースである
実行すると以下の動作をする
- RoomAに移動し敵兵1,2 を倒す
- RoomCに移動して敵兵6を倒す

手動作成によるバリエーションの低下

実行不可能なテストケース

キーワード	ターゲット
移動	RoomA
攻撃	army6
攻撃	army2
移動	RoomC
攻撃	army1

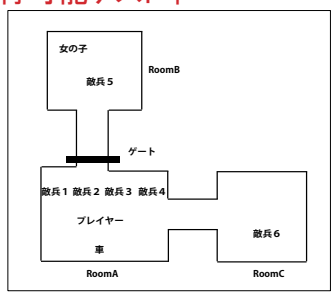


- 手動で実行可能なテストケースを作っていくと、テストケースのバリエーションが少なくなってしまう
- キーワードの順番のミスなどが発生する

自動生成による実行可能テスト率

実行不可能なテストケース

キーワード	ターゲット
移動	RoomA
攻撃	army6
攻撃	army2
移動	RoomC
攻撃	army1



単純にキーワードの順列を自動で生成すると
上のような実行できないテストケースが大量に含まれる

関心のないキーワードの混入

実行不可能なテストケース

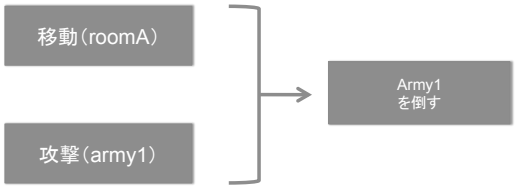
キーワード	ターゲット
移動	RoomA
攻撃	army6
攻撃	army2
移動	RoomC
攻撃	army1

関心のないキーワード

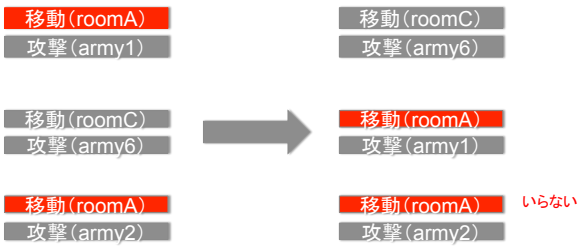
- 敵を倒す順番を変えたテストケースを書きたい時に「attack」以外のキーワードを気にしなければならないのは面倒くさい
- テスト開発期間中にはレベル調整のためにイベントの順番などが入れ替わるため変更しにくいのは問題である

抽象的なキーワード

既存手法では具体的なキーワードを組み合わせて抽象的なキーワードを定義することでができる
こうすることで、移動をいちいち書かなくていいように見える



抽象的なキーワードの入れ替え



- キーワードを入れ替えた際に入れる必要のないキーワードが入ったりする
- 抽象的なキーワードの動作が決定的だと余計なキーワードが挿入されてしまったりして場合によってはうまく動作しない

既存手法の問題点

実行可能なテストケース率が低くなる

- ツールでは実行可能なテストケースを判断して生成することができない

テストケースのバリエーションが少なくなる

- テストエンジニアが手動で多くのバリエーションのテストケースを作成するのは難しい

重要なキーワードのみを使ってテストケースを作成することができない

- 実行可能なテストケースを作成するために関心のないキーワードを大量に記述しなければならない

着眼点

キーワード	ターゲット
移動	RoomA
攻撃	army2
攻撃	army1
移動	RoomC
攻撃	army6

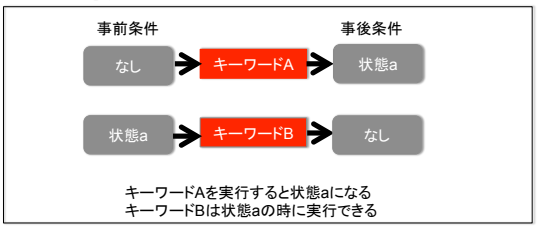
キーワード	ターゲット
移動	RoomA
攻撃	army1
攻撃	army2
移動	RoomC
攻撃	army6

攻撃(army1)実行する際には必ずRoomAにいる必要がある
RoomAに行くには移動(RoomA)を実行する

つまり、移動(RoomA)実行後のゲームの状態が
攻撃,army1を実行するために必要な条件である

キーワードによって変化する状態を明示的に表す

- キーワードが実行された後にゲームの状態がどう変化するか、キーワードを実行する際にどのような状態である必要があるかを明示的に表すことで、キーワードが正しい前後関係かどうか判定することができる



キーワードAを実行すると状態aになる
キーワードBは状態aの時に実行できる

提案手法:キーワードの拡張

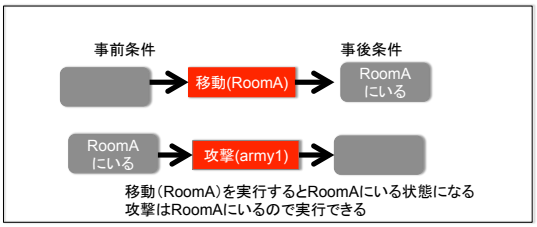
キーワードによって変化するゲームの状態を明示的に定義する

- キーワードを実行する際に満たされていなければならない状態を事前条件として定義する
- キーワードが実行された後に満たされている状態を事後条件として定義する

既存手法のキーワードを拡張し
事前条件と事後条件を加える

キーワードによって変化する状態を明示的に表す

- キーワードが実行された後にゲームの状態がどう変化するか、キーワードを実行する際にどのような状態である必要があるかを明示的に表すことで、キーワードが正しい前後関係かどうか判定することができる



移動(RoomA)を実行するとRoomAにいる状態になる
攻撃はRoomAにいるので実行できる

テストケースの補完ツール

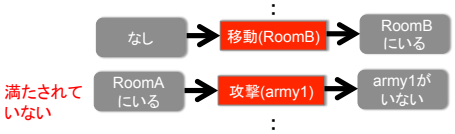
事前条件と事後条件を定義したことでテストツールが自動でテストケースを補完することができる

以下の動作をする補完ツールを作成した

- テストケースのキーワードを順番に読み込む
- キーワードの事前条件が満たされていない場合は該当する事後条件を持つキーワードを挿入する
- 条件を満たすためのキーワードが複数必要な場合や複数の候補がある場合は挿入するキーワードのバリエーションを網羅した
テストケースを生成する

キーワードを読み込み事前条件が満たされているかを確認する

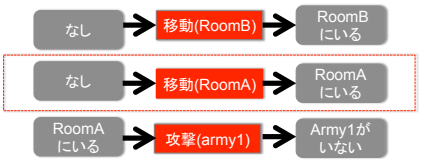
①キーワードを上から順番に読み込み
事前条件が満たされているか確認する



キーワードCを読み込んだ際に事前のキーワードAによって
キーワードCの事前条件が満たされていないのなら、必要なキーワードを
事前に挿入する必要がある

必要なキーワードを キーワードライブラリから選択し 挿入する

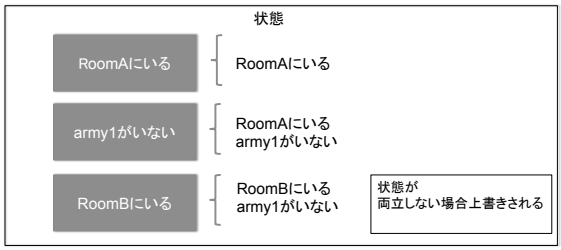
②該当するキーワードをキーワードライブラリ
から探し出し、挿入する



挿入したキーワードBに対しても事前条件が満たされているかどうかの
チェックをし、必要があればキーワードを挿入する

事後条件の継承と 両立しない条件

- **予置した事後条件を持つ場合に状態を新しい事後条件で 上書きされる**



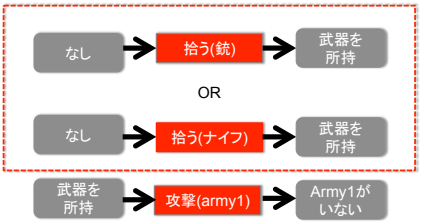
複数のキーワードの挿入が 必要な場合

挿入しなければならないキーワードが複数ある場合
順序を変えたテストケースを作成する

複数のキーワードの挿入が 必要な場合

挿入しなければならないキーワードが複数ある場合
順序を変えたテストケースを作成する

キーワードの候補が複数ある 場合



挿入するキーワード候補が複数ある場合、
異なるキーワードを選択した場合のテストケースを作成する

キーワードの候補が複数ある 場合

挿入するキーワード候補が複数ある場合、
異なるキーワードを選択した場合のテストケースを作成する

提案手法を用いた
テストケース作成手順

提案手法を用いたテストケース作成手順は以下のようになる

1. **具体的なキーワードの作成**
2. **事前条件と事後条件を持つ抽象的なキーワードの作成**
    1. テスト対象を分解しキーワードを作成する
3. **テストケースの作成・改修**

具体的なキーワードの定義

キーボードやマウスの入力を元にキーワードを定義する

- 左クリック→「射撃」、R→「リロード」

視点の移動やキャラクターの移動など移動系の操作は抽象的なものをこの段階で用意する

- ～まで移動する、～を見るなどのキーワードを定義する

抽象的なキーワードの定義

キーワードを定義するにはキーワードの振る舞い、事前条件、事後条件を決める必要がある。

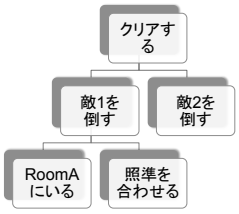
- テストに使いそうなキーワードをいきなり作り出すのは難しい

テスト対象のゲームを分解しキーワードの候補になるものを抽出しキーワード間の関係を明らかにしキーワードを定義する

ゲームの分解

ゲームをゴールで分解しキーワードの候補を抽出する

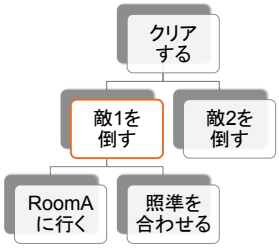
- ゴールとはプレイヤーが達成しようとするゲームの目的
- ゴールは親子関係があり、子のゴールを達成することで親のゴールを達成することができるという関係になる



敵1と敵2を倒すことで
トップのゴールである「クリア」を
達成することができる

キーワードの定義

「敵1を倒す」ゴールは「RoomAに行く」というキーワードと「照準を合わせる」という子ゴールを達成する必要がある
子ゴールを達成する順番は「RoomAに行く」が必ず先に来るので、
別々のキーワードではなく「敵1を倒す」というキーワードの振る舞いにしてしまう

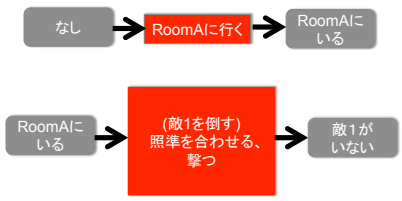


キーワードの定義

「敵1を倒す」ゴールは「RoomAに行く」というキーワードと「照準を合わせる」という子ゴールを達成する必要がある
子ゴールを達成する順番は「RoomAに行く」が必ず先に来るので、
別々のキーワードではなく「敵1を倒す」というキーワードの振る舞いにしてしまう

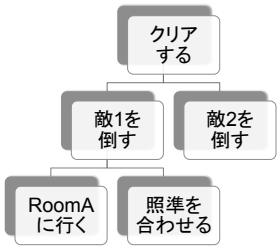
キーワードの定義

「RoomAに行く」という操作はRoomAにいれば実行する必要がないので他のキーワードにしてしまっても良い



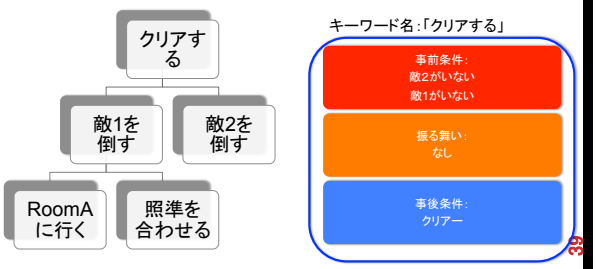
キーワードの定義

子同士に優先順位がないなら、子のキーワードを達成することで親の事前条件が満たされるような関係になるように条件を設定していく



キーワードの定義

子同士に優先順位がないなら、子のキーワードを達成することで親の事前条件が満たされるような関係になるように条件を設定していく



テストケースの作成

テストエンジニアは「attack」順番に関心があるなら、「attack」のみでテストケースを作成する

既存手法

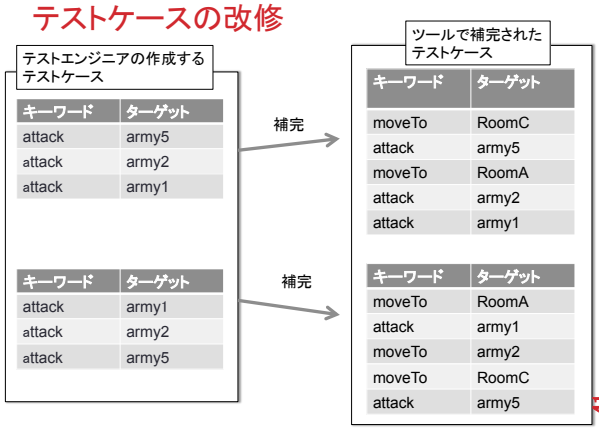
キーワード	ターゲット
moveTo	RoomC
attack	army5
moveTo	RoomA
attack	army2
attack	army1

提案手法

キーワード	ターゲット
attack	army5
attack	army2
attack	army1

テストエンジニアは関心の実行する順序に
関心のあるキーワードのみをスプレッドシートに記述すれば良い

テストケースの改修



検証

サンプルのFPSにたいしてテストケースの作成と改修を行ってもらう

被験者はテスト作成経験のない8名

- 既存手法4名　提案手法4名

実行可能なテストケース率とテストケースのバリエーションを測定した

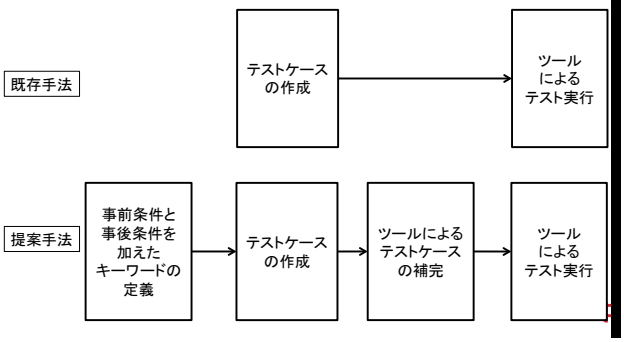
既存手法

FPSに適用するためのKDTツールは存在していない

本手法のために作成したツールのサブセットを使う

事前条件と事後条件を定義していないキーワードを使いテストケースを作成する

既存手法と提案手法の手順の違い

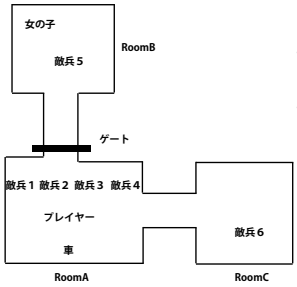


テスト対象の仕様

一般に販売されているFPSは外部のツールで操作することができなかったので
FPSの基本的な機能をもつサンプルを作成し検証に使用した

入力	動作
マウスの移動	視点の移動
WASD	キャラクターの移動
左クリック	攻撃
右クリック	武器の変更

サンプルのFPSのマップ



サンプルFPSのマップ

- クリア条件
  - 女の子を車まで連れていく
  - 敵兵6を倒す
- その他
  - 敵兵1～4を倒すとゲートが開く
  - 10分程度でクリア可能

検証手順

1. **被験者全員にサンプルのFPSの説明とKDTの説明を行う**
2. **被験者全員にキーワードのリストを渡し、説明する**
3. **提案手法の被験者は準備として事前条件と事後条件を加えたキーワードを新たに定義してもらう**
4. **キーワードを使い目的通りのテストケースを作成してもらう**
5. **作成してもらったテストケースを実行する**
6. **サンプルのFPSに変更を加え、変更箇所を説明する**
7. **変更が反映されるようにテストケースやキーワードを改修してもらう**
8. **変更されたテストケースを実行する**

被験者の作成した実行できないテストケース

moveTo	RoomC
attack	army6
attack	gate-army3
attack	gate-army4
attack	gate-army1
attack	gate-army2
moveTo	RoomB
attack	army5
moveTo	RoomA
restart	

RoomAに移動するためのキーワードを入れ忘れてしまったため、attack,gate-army3を実行することができず、下のキーワード全てが正しく実行されない

被験者の作成した実行できないテストケース

テストケース作成結果

	被験者	準備時間（分）	作成時間（分）	テストケース数（件）	実行できないテストケース数（件）	実行可能なテストケース率（%）
既存手法	A	なし	27	48	0	100
	B	なし	28	19	3	84
	C	なし	33	23	6	74
	D	なし	45	24	1	96
提案手法	E	133	1	48	0	100
	F	20	1	48	0	100
	G	45	1	48	0	100
	H	50	1	48	0	100

- 既存手法では実行できないテストケースを作成してしまっている
- 既存手法ではテストケースの件数が少ない
- 被験者Eは準備に多くの時間がかかっている

テストケース改修結果

	被験者	作成時間（分）	テストケース数（件）	実行できないテストケース数（件）	実行可能なテストケース率（%）
既存手法	A	25	24	0	100
	B	11	19	3→5	74
	C	20	23	6→8	65
	D	13	24	1	96
提案手法	E	2	48	0	100
	F	15	48	0	100
	G	20	48	0	100
	H	22	48	0	100

- 実行できないテストケースの件数が増えている
- 作成結果では準備に時間がかかっていた
  被験者Eの作成時間が最も少ない

検証結果まとめ

既存手法に比べ提案手法では実行可能なテストケース率が高くなった

提案手法では既存手法に比べて多くのバリエーションを持つテストケースを作成することができた

改修結果では既存手法に比べ,1件あたり半分以下の時間で改修を行うことができた

- 既存手法: 1件あたり0.7分
- 提案手法: 1件あたり0.3分

考察

被験者Eの改修時間が短いのはキーワードの作成に多くの時間を使い、変更を見越したキーワードの作成ができていたからだと思われる

- テストの変更に強いキーワードを作成する指針が必要である

テスト対象の規模が小さい場合は既存手法を用いたほうがコストが低い可能性がある

まとめ

本研究では既存のKDTを拡張し、FPSに適用した

本手法を用いるためのテストツールを作成した

検証ではサンプルのFPSに対してテストケースの作成と改修を行い、実行可能なテストケースをより網羅的に作成することができることを示した

今後の課題として、ゲームの開発に応じてキーワードの作成工数がどのように変わるか検証する必要がある

参考文献

[1]wazap,"Call of Duty 4: Modern Warfare":http://jp.wazap.com/game/,2015/09/20

[2]デジタルコンテンツ協会,"デジタルコンテンツ制作の先端技術応用に関する調査研究",2010

[3] gamespot,http://www.gamespot.com/,2015/09/29

[4]selenium,http://docs.seleniumhq.org/download/,2015/09/30

[5] HansBuwalda,"ActionBasedTesting",stickyminds.com ,http://www.sticyminds.com/better-software-article/not-just-number-real-value-metrics.

[6].RobotFreamwork,http://robotframework.org/ ,2015/09/30

[7]EA,"BattleField",www.battlefield.com,2015/09/29

御清聴ありがとうございました

事前条件と事後条件を追加したキーワード

提案手法の
「moveTo,RoomA」キーワード

事前条件：
なし

振る舞い：
RoomAに移動する

事後条件：
RoomAにいる

提案手法の
「attack,army1」キーワード

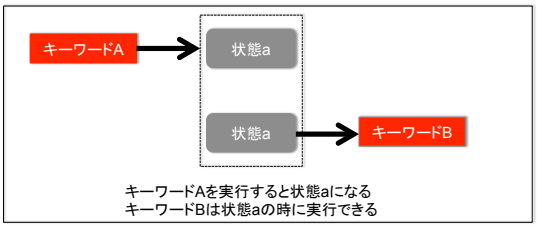
事前条件：
RoomAにいる

振る舞い：
照準を合わせる
銃を撃つ

事後条件：
敵兵1が死んでいる

キーワードによって変化する状態を明示的に表す

- キーワードが実行された後にゲームの状態がどう変化するか、キーワードを実行する際にどのような状態である必要があるかを明示的に表すことで、キーワードが正しい前後関係かどうか判定することができる



キーワードAを実行すると状態aになる
キーワードBは状態aの時に実行できる