



公式説明書

(202509更新)

目次

1. ゲームの試し方…P2
2. スモウルビー甲子園のルール…P5
3. スモウルビーの使い方…P8
 - (1)スモウルビー・エディタの起動…P8
 - (2)スモウルビー・エディタの画面構成…P8
 - (3)ブロックの種類と使い方…P9
 - (4)プログラムの保存…P12
 - (5)プログラムの実行…P13
4. AIプログラムの作り方…P14
 - (1)最初に覚えておくこと…P14
 - (2)テンプレートの使い方…P15
 - (3)スモウルビー甲子園メソッドの種類と使い方…P16
 - (4)簡単なAIプログラムを作ってみる…P25
5. スモウルビー3.0の使い方…P30
 - (1)スモウルビー3.0版メソッドの種類と使い方…P30
6. 便利なメソッドの紹介…P39
 - (1)草薙の剣の座標を調べる…P40
 - (2)配列に含まれる座標の数を確認する…P41
 - (3)一番近い加点アイテムを目指して移動する…P42
 - (4)ターン数を管理する…P44
 - (5)ダイナマイトを置く…P45
7. 作戦の考え方…P47
 - (1)やり方1…P47
 - (2)やり方2…P47
 - (3)優勝者からのアドバイス…P47
8. その他の説明…P51
 - (1)スモウルビー甲子園の構成…P51
 - (2)マップエディタの使い方…P51
 - (3)ログの確認方法…P53
 - (4)エラーの調べ方…P54
9. 参考…P56

1. ゲームの試し方

- デスクトップ上のスモウルビー甲子園20xxアイコンをダブルクリックしてください。
- ゲームビューアが起動したら、「START」ボタンをクリックしてください。



- ゲームは2ラウンドを行います。(2ラウンドの合計得点で勝敗を決定します。)
- ROUND1が終了したら、もう一度「START」ボタンをクリックしてください。



- ゲーム終了後、再度ゲームを試したい場合は「NEW GAME」ボタンをクリックしてください。

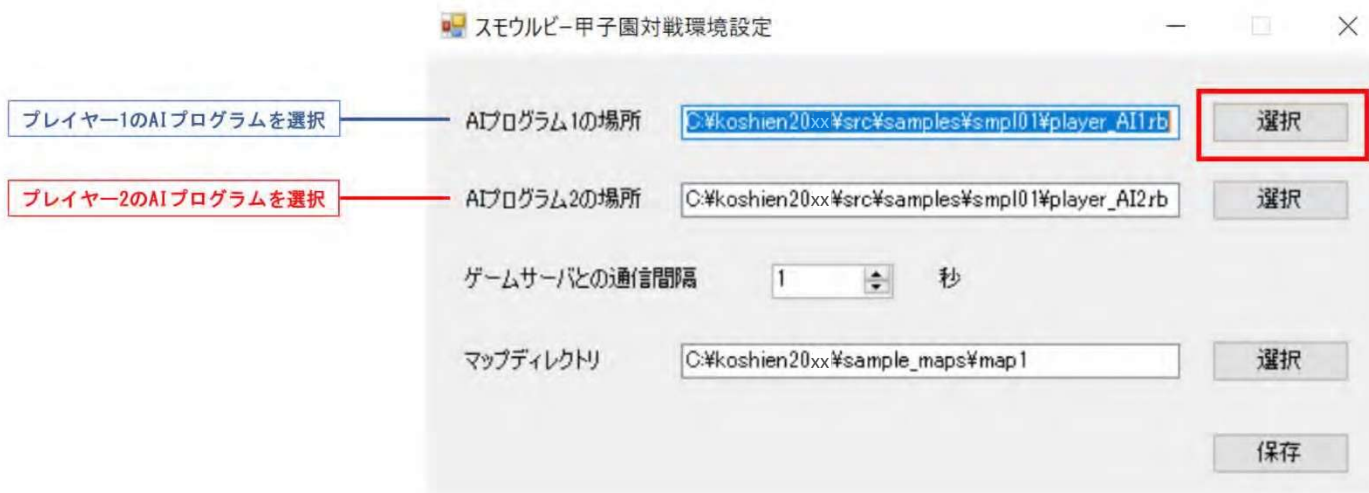


[使用するAIプログラムを変更する場合]

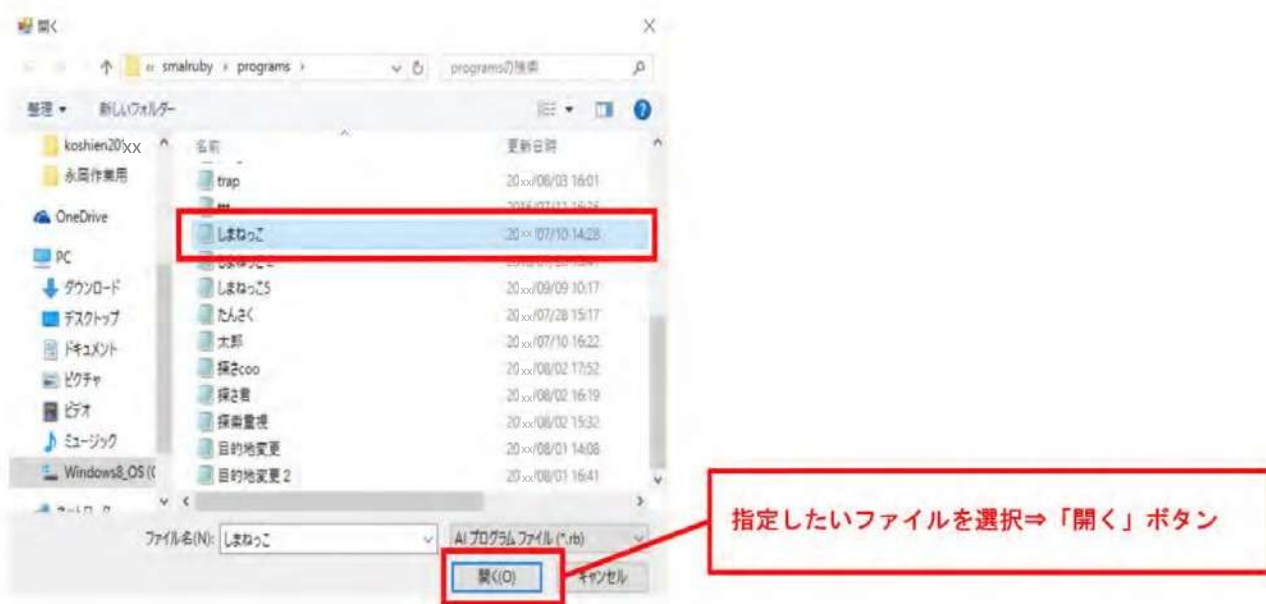
- 「CONFIG」ボタンをクリック



- 変更したいプレイヤーの「選択」ボタンをクリック



- ファイルを選択し、「開く」ボタンをクリック



[スマウルビーで作ったプログラムの保存場所]
Cドライブ > koshien20xx > smalruby > programsの中

[サンプルプログラムの格納場所]
Cドライブ > koshien20xx > src > samples > の中

※順番に高度な内容になっています。⇒

smpl01
smpl02
smpl03

- 「ゲームサーバとの通信間隔」を設定
(秒数を少なくするとゲームの進行速度が速くなります。)

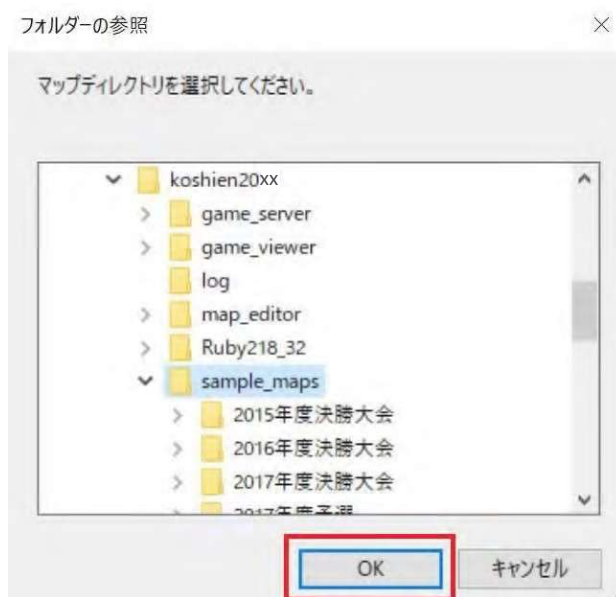


[対戦マップを変更する場合]

- マップディレクトリの「選択」ボタンをクリック



- フォルダを選択し、「OK」ボタンをクリック



[マップエディタで作ったプログラムの保存場所]
 Cドライブ > koshien20xx > map_editor > data の中
 ※間違っても編集しないよう保存したファイルはどこか別の場所に移して置きましょう。
 (koshien20xx > map_editor > data に新しくフォルダを作って保存しておくとも便利です。)

[サンプルマップ及び過去の決勝大会のマップ格納場所]
 Cドライブ > koshien20xx > sample_maps の中

以上で設定は完了です。

2. スモウルビー甲子園のルール

(1) 基本ルール

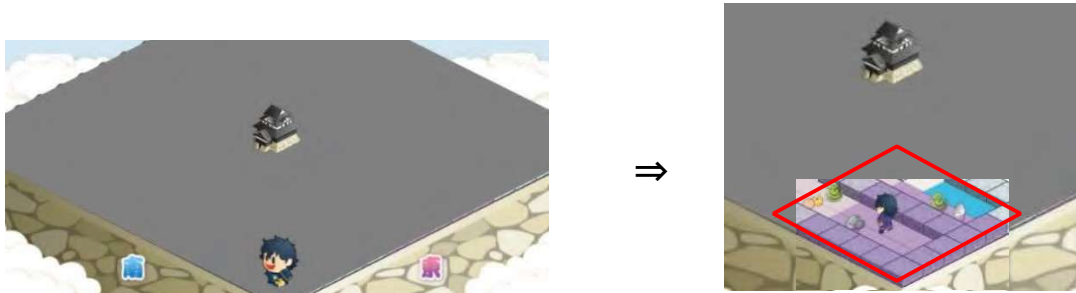
- 17マス×17マスのマップ上でゴールを目指すゲーム
- 最大50ターンのターン制で進行
- プレイヤーは1ターンに1マスだけ移動可能
- 1ターンにAIプログラムを実行できる時間は10秒以内




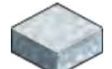


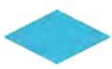

(2) マップ情報について

- ゲーム開始時点で、プレイヤーは、自分の位置とゴールの位置を知っていますが、それ以外のマップ情報(マップ上に何があるか、などの情報)を持っていません。
- マップ情報は、「マップ情報を取得」の命令を実行した5マスx5マスの範囲の情報を取得することができます。「マップ情報を取得」の命令を実行していない範囲のマスは「未探索のマス」と認識します。

x座標が [キャラクターのx座標]、y座標が [キャラクターのy座標] 付近のマップ情報を取得



- マスには、移動可能な「空間」マス、移動できない「壁」マスなど複数種類あり

種類	値	内容
未探索のマス	-1	マップ情報が不明なマス。最短経路を検索するときは移動可能なマスとみなす。
	0	空間(自由に移動可能なマス)
	1	壁(移動できないマス)
	2	蔵(移動できないマス)
	3	ゴール(到着するとラウンドが終了し、通過はできないマス)
	4	水たまり(移動命令が1回実行されないマス)
	5	壊せる壁(移動できないマス。ただし、ダイナマイトで壊すと空間に変わり移動可能になる)

- ・大会ルールとしてマップは壁または蔵に囲まれているものとします。
- ・プレイヤー1とプレイヤー2が同時に同じマスに入ることができます。
- ・早いターン数でゴールすれば高い得点を獲得(ゴールボーナス)

ターン数	1-10	11-20	21-30	31-40	41-50	未ゴール
得点	100点	90点	80点	70点	60点	0点

- ・マップ上には加点アイテムと減点アイテムが設置されています。アイテムが設置されているマスにプレイヤーが入ると、アイテムを取ったことになり、アイテムの点数が加点または減点されます。

[加点アイテム]

種類	お茶 	和菓子 	丁銀 	シロイルカ 	草薙剣 
値	a	b	c	d	e
得点	10点	20点	30点	40点	60点



- ・大会ルールとして、草薙剣はマップ上に1個のみ出現するものとします。
(対戦環境のマップ上に草薙剣を2個以上設置することは可能ですので、マップを自作される場合はご注意ください。)

[減点アイテム]

種類	毒キノコ 	蛇 	トラバサミ 	爆弾 
値	A	B	C	D
得点	▲10点	▲20点	▲30点	▲40点

- ・アイテムが設置されているマスにプレイヤー1とプレイヤー2が同時にいったときは、アイテム点数の半分の点数が両方のプレイヤーに加点または減点されます。

[使えるアイテム]

種類	内容
	ダイナマイト。 空間または水たまりの上に置いて、隣にある「壊せる壁」を壊すことができる。
	爆弾。 減点アイテムとして空間におくことができる。

・妨害キャラクタ



(1～40ターン)

(41～50ターン)

- ・1回の接触につき10点減点(何度でも接触します。)
- ・プレイヤーキャラクタがゴールしたとき、ゴール上に妨害キャラクタがいるときも減点されます。
- ・1～40ターンはランダムで移動しますが、プレイヤーキャラクタが3マス以内に入ると、プレイヤーキャラクタへ向かって移動します。
- ・41～50ターンは最寄りのプレイヤーキャラクタに向かって移動します。プレイヤー1とプレイヤー2が等距離にいる場合は、ラウンド1はプレイヤー1へ、ラウンド2はプレイヤー2 へ向かって移動します。

・妨害キャラクタの撃退

草薙剣を保持した状態で妨害キャラクタに接触すると30点加点。妨害キャラクタは画面からいなくなります。

・歩行得点

5マス移動するたびに3点が加点されます。

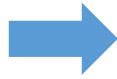
(3) タイムアウトについて

- ・1ターンの間にAIプログラムを実行することができる時間は10秒以内です。
- ・10秒を超えると「タイムアウト」になります。
- ・タイムアウトになると、画面に「タイムアウトしました」と表示され、プレイヤーキャラクタが画面からいなくなります。
- ・タイムアウトになった場合の得点は、タイムアウトになるまでに取った加点アイテムの得点、減点アイテムの得点、および歩行得点の合計点です。

3. スモウルビーの使い方

(1) スモウルビー・エディタの起動

デスクトップ上にあるスモウルビーAI作成のアイコンをダブルクリックするとスモウルビー・エディタが起動します。



(2) スモウルビー・エディタの画面構成

スモウルビー・エディタの画面構成は次の通りです。

画面の説明



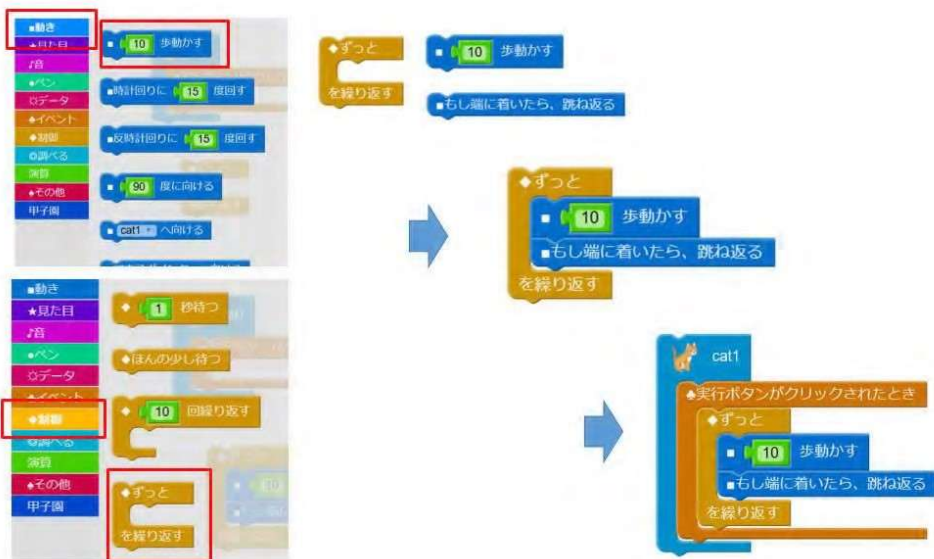
プログラムエリア	スモウルビーのプログラムを組み立てるキャンパスとなる領域
ジャンル	プログラムエリアに配置できる各種命令ブロックをグルーピングした領域
ゴミ箱	プログラムエリアで不要になったブロックを廃棄するためのツール (廃棄したい命令ブロックをゴミ箱にドラッグ&ドロップ)
キャラクターリスト	スモウルビーの実行画面に表示するキャラクタの画面を管理する領域
操作メニュー	作成したプログラムの実行や、保存・読み込みなどの各種操作を行う領域
ブロック	ジャンルより選択した命令ブロックがプログラムエリアに表示
モード切替	「スモウルビー」と「Ruby」モードの切り替え画面 (「Ruby」モードではRubyのコードに変換されて表示)

(3) ブロックの種類と使い方

・ブロックの種類と格納場所

ブロックはジャンルの中に格納されています。

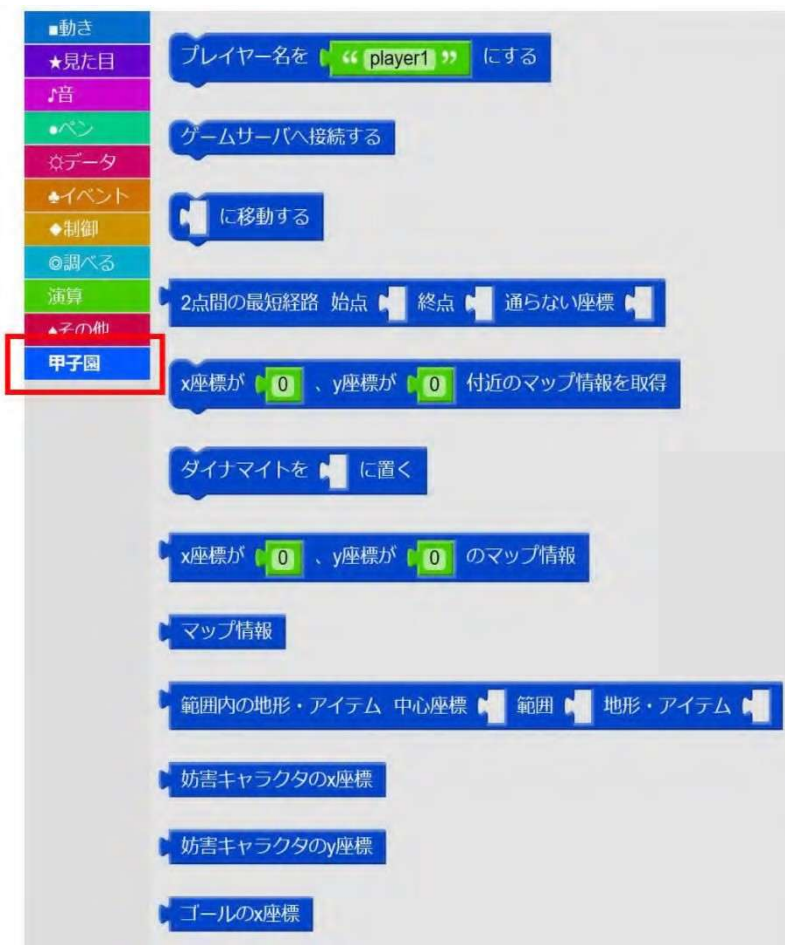
このブロックを組み合わせることにより、プログラムを作ります。



・AIプログラム作成では次のジャンルのブロックをよく使います。

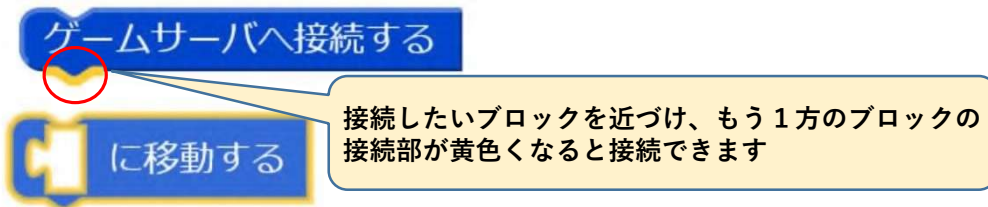
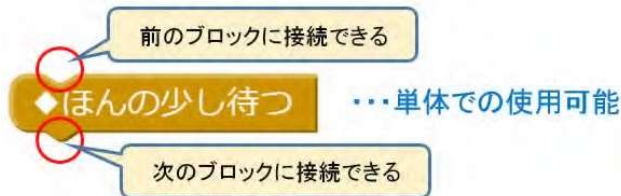
[甲子園]

▶ スモウルビー甲子園専用を用意したブロックを格納



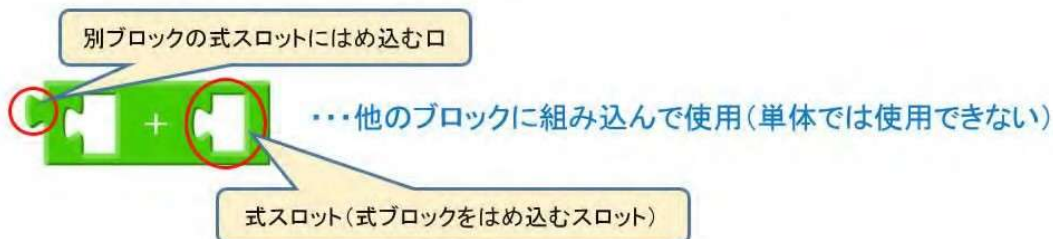
[文ブロック]

- ▶プログラムの処理を構成する主要なブロック
- ▶プログラムは、結合された文ブロックを上から下へ処理



[式ブロック]

- ▶式ブロックは、戻り値を伴うRubyの式を実行するためのブロック
- ▶他の文ブロックや式ブロックの中にある「スロット」に組み込んで使用



[参考]

- ▶文ブロック3つの組み合わせ ⇒ この内容が上から下に実行される
- ▶式ブロックは文ブロックに組み込んで使用(緑囲い部分)



(5) プログラムの実行

・作成したプログラムは、「実行」ボタンを押すことにより、実際に動かすことができます。

※AIプログラムは「実行」ボタンで動かすことはできません。

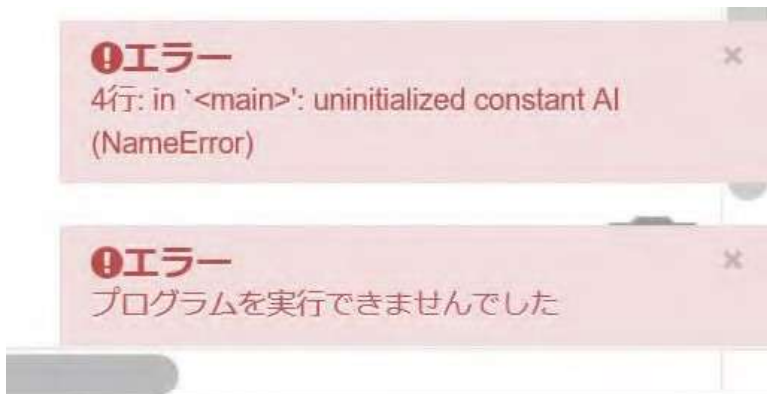


※プログラム実行画面は「ESC」キーは「×」ボタンで終了できます。

・AIプログラムを動かすことはできませんが、デバックに利用することができます。

⇒AIプログラムを実行した場合、必ず2つのエラーが表示されます。

⇒これ以外のエラーが表示された場合はプログラムに問題があります。



4.AIプログラムの作り方

(1)最初に覚えておくこと

- マップにはx軸とy軸があり、0から16までの番号が付いています。



- このx軸、y軸の番号を使って、特定のマスを表示することができます。

マスの位置を座標と言い、[x座標,y座標]で表します。

x座標は、東に行くと増え、西に行くと減ります。(算数・数学と同じ向き) y座標は、北へ行くと減り、南へ行くと増えます(算数・数学と反対向き) マーカー部の座標を表すと下のようになります。

		X軸																
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
y軸	0																	
	1																	
	2			[2, 2]					[7, 2]					[12, 2]				[16, 2]
	3																	
	4																	
	5																	
	6																	
	7			[2, 7]					[7, 7]					[12, 7]				[16, 7]
	8																	
	9																	
	10																	
	11																	
	12			[2, 12]					[7, 12]					[12, 12]				[16, 12]
	13																	
	14																	
	15																	
	16			[2, 16]					[7, 16]					[12, 16]				[16, 16]

・「配列」について

Rubyには「配列」というデータ形式があります。
複数のデータをひとまとまりにして扱うことができる形式です。
Rubyの配列は、データをコンマで区切り、角かっこで囲みます。

例[データ1,データ2,データ3]

実は、スモウルビー甲子園において座標は配列です。

例[7,7]

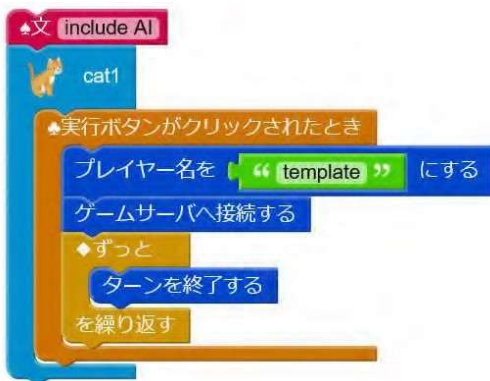
複数の座標を扱う場合、【座標の配列】で表します。

例[[7,7],[7,8],[7,9]]

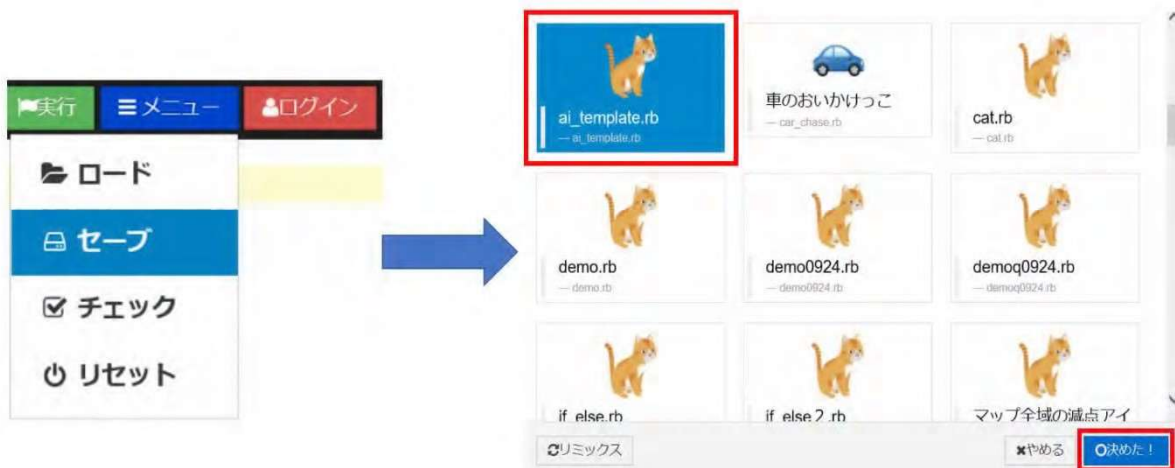
※座標の取り扱い方座標が配列であること、いろいろな命令の実行結果が【座標の配列】で得られることを覚えてください。

(2)テンプレートの使い方

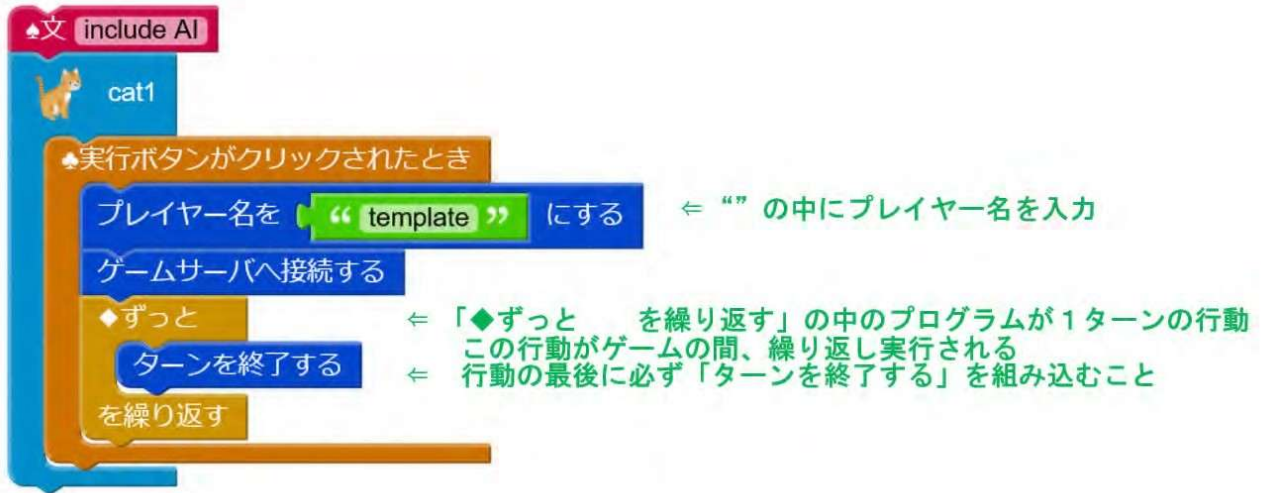
・AIプログラムは次のテンプレートを使って作成します。



・テンプレートの読み込みスモウルビー・エディタを立ち上げ、「メニュー」―「ロード」
「ai_template.rb」を選択し「決めた」をクリック



・テンプレートの構成



- ・テンプレートに含まれるブロックは消さないように気を付けてください。消すと動かなくなります。消してしまったときは「甲子園」ジャンルの中から取り出して入れてください。
- ・「ずっと～を繰り返す」の中に命令ブロックを入れていきます。
- ・「ずっと～を繰り返す」のブロックは、「実行ボタンがクリックされたとき」のブロックの内側に入れたままにしてください。外側に出すと動かなくなります。
- ・「ずっと～を繰り返す」の内側に「ターンを終了する」入れてください
- ・ターンが始まって10秒以内に「ターンを終了する」を実行しないとタイムアウトになり、そのラウンドの行動は終了する、というルールです。
- ・「ターンを終了する」が無いと、10秒を過ぎても「ターンを終了する」を実行することができないので、タイムアウトします。


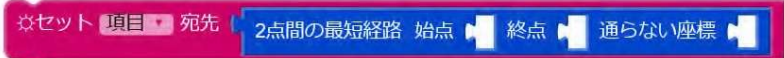
(3) スモウルビー甲子園メソッドの種類と使い方

① 1マス移動する

ブロック		1回のみ ①③④⑭のいずれか2回まで
コード	move to(引数)	
実行内容	<ul style="list-style-type: none"> ・指定した座標にプレイヤーが1マス移動します。 ・指定できるのは現在地から東西南北の1マスです。(斜めには移動できません) 	
引数	<ul style="list-style-type: none"> ・移動先の座標を配列または変数で指定します。 	
解説	<ul style="list-style-type: none"> ・移動できるのは空間と水たまりだけで、壁には移動できません。 ・移動できない座標を指定した場合、使用回数はカウントされますが、実行は無視されます。 ・水たまりに移動した場合は、次回の移動命令が無視されます。(使用回数はカウントされます。) 	


※1ターンの使用回数に制限のある命令があります。詳細はP24をご覧ください。

②最短経路を検索する

ブロック	
コード	calc_route(引数)
実行内容	<ul style="list-style-type: none"> 指定した2点間の最短ルートを検索します。 戻り値として、始点から終点までの座標配列が返されます。
引数	<ul style="list-style-type: none"> 引数としてハッシュを指定します。 指定できるハッシュは、「始点」「終点」「通らない座標」です。 それぞれのハッシュは省略も可能です。 <p>[ハッシュを省略した場合の設定]</p> <ul style="list-style-type: none"> ▶ 始点: 現在地 ▶ 終点: ゴール ▶ 通らない座標: 未設定
解説	<ul style="list-style-type: none"> 戻り値の座標配列は次の順番に並んでいます。 [[始点],[次の移動先],...経路順の座標...,[終点]] 戻り値の中の各座標は並び順に0,1,2,3...の番号で指定できます。 指定した条件での経路がない場合は始点の座標が1つだけ返されます。 マップ情報を取得していない範囲のマス(未探索のマス)は全て移動可能とみなして経路探索するので注意が必要です。ただし、判明している空間マスがあれば、そちらを通る経路を優先します。 関数ブロックとセットで使用します。  <ul style="list-style-type: none"> コード入力する場合のハッシュ指定は次のとおりです。 ⇒ src(始点)、dst(終点)、except_cells(通らない座標) ⇒ 入力例1 全てのハッシュを指定する場合 calc_route(src:[13,9],dst:[7,7],except_cells:[9,9]) 入力例2 通らない座標のみを指定する場合 calc_route(except_cells[9,9]) 入力例3 ハッシュを省略する場合 calc_route


③5マス×5マスのマップ情報を取得

①③④⑭のいずれか2回まで

ブロック	
コード	<code>get_map_area(引数)</code>
実行内容	<ul style="list-style-type: none"> 指定した範囲(5マス×5マス)のマップ情報を取得します。 戻り値として、指定した範囲のマップ情報が返されます。
引数	<ul style="list-style-type: none"> 指定したい範囲の中心座標を配列または変数で指定します。
解説	<ul style="list-style-type: none"> 取得できるマップ情報は、指定した範囲の以下の情報です。 <ul style="list-style-type: none"> ▶マップ構成(空間・壁・水たまり・ゴール・壊せる壁) ▶加点アイテム、減点アイテムがある場合は、その座標と種類解説 ▶対戦相手が指定範囲内にいる場合はその座標 ▶妨害キャラクタの現ターン開始時点の座標と前ターン開始時点の座標(妨害キャラクタのみは指定範囲内にいなくても情報取得が可能) 取得した情報はAIライブラリに保存され、後から呼び出すことができます。

④ダイナマイトを置く

①③④⑭のいずれか2回まで

ブロック	
コード	<code>set_dynamite(引数)</code>
実行内容	<ul style="list-style-type: none"> ダイナマイトを現在地又は隣接する東西南北のマ스에設置します。
引数	<ul style="list-style-type: none"> ダイナマイトを設置したい座標を配列または変数で指定します。 引数を省略した場合は現在地にダイナマイトを設置します。
解説	<ul style="list-style-type: none"> ダイナマイトは、空間または水たまりの上に置くことができます。アイテムがあるマスには設置できません。 ダイナマイトは1ラウンドに2回まで設置できます。 無効な座標を指定した場合、設置されませんが、ダイナマイトは1つ消費され、使用回数もカウントされます。 両プレイヤーが同じ座標に同時にダイナマイトを設置した場合、両プレイヤーとも設置は成功しますが、ダイナマイトは1つだけ設置されたこととなります。 ダイナマイトは設置したターンの終了時に爆発します。 ダイナマイトが爆発すると、ダイナマイトのマ스에隣接する「壊せる壁」は「空間」になり、次のターンから移動可能になります。 ダイナマイトの爆発は「壊せる壁」以外の地形、プレイヤー、妨害キャラクタ、アイテムに影響を与えません。ダイナマイトが爆発したマスや隣接したマ스에プレイヤーがいても減点されることはありません。





⑤ 指定した座標のマップ情報を呼び出す

ブロック	
コード	map(引数)
実行内容	<ul style="list-style-type: none"> 指定した座標のマップ情報を呼び出します。 指定した座標のマップ情報が返されます。
引数	<ul style="list-style-type: none"> 呼び出したい座標を配列で指定します。
解説	<ul style="list-style-type: none"> 呼び出せるマップ情報は「③マップ情報の取得」で取得した情報です。 マップ情報の取得を実行していない座標を指定した場合は、-1が返されます。 マップエリア外を指定した場合は、nilが返されます。



⑥ マップ全体のマップ情報を呼び出す

ブロック	
コード	map_all
実行内容	<ul style="list-style-type: none"> マップ全体のマップ情報を呼び出します。 マップ全体のマップ情報が返されます。
引数	-
解説	<ul style="list-style-type: none"> 呼び出せるマップ情報は「③マップ情報の取得」で取得した情報です。 マップ情報の取得を実行していない座標は、-1が返されます。



⑦指定した範囲に、指定した要素が存在するかどうかを確認する

ブロック	
コード	locate_objects(引数)
実行内容	<ul style="list-style-type: none"> 指定した範囲に、指定した要素が存在するかどうかを確認します。 指定した要素がある座標が返されます。
引数	<ul style="list-style-type: none"> 引数としてハッシュを指定します。 指定できるハッシュは次のとおりです。 <ul style="list-style-type: none"> ▶ 中心座標: 指定する範囲の中心座標(省略時は現在地) ▶ 範囲: 指定する範囲の縦横のマス長さ(省略時は5) ▶ 地形・アイテム: 確認したい要素の値(省略時は減点アイテム全種類)
解説	<ul style="list-style-type: none"> 戻り値は座標の配列で返されます。 配列の中の各座標はy座標が小さい順に並んでいます。(y座標が同じ場合はx座標の小さいほうが先になります。) マップ情報を取得していない座標の情報は確認できません。 関数ブロックとセットで使います。  <ul style="list-style-type: none"> マップ全体を範囲指定したい場合は次のようになります。 ⇒減点アイテム  <ul style="list-style-type: none"> ⇒加点アイテム  <ul style="list-style-type: none"> コード入力する場合のハッシュ指定は次のとおりです。 ⇒ cent:[中心座標] sq_size:[範囲(数値)] objects:["地形・アイテムをあらわす文字"]の配列 ⇒ 入力例 1 マップ全体の加点アイテムの座標を確認する場合 locate_objects(cent: [8,8],sq_size: 17,objects: ["a","b","c","d","e"]) ⇒ 入力例 2 マップ全体の水たまりの座標を確認する場合 locate_objects(cent: [8,8], sq_size: 17, objects: [4])



⑧対戦キャラクタのx座標、y座標

ブロック	 
コード	<code>other_player_x</code> <code>other_player_y</code>
実行内容	<ul style="list-style-type: none"> 対戦相手の座標を呼び出します。 最後にマップ情報の取得で把握した対戦相手の座標(x,y)を返します。
引数	-
解説	<ul style="list-style-type: none"> 得られる情報は、マップ情報の取得で把握した時点の情報です。 対戦相手の座標は、マップ情報の取得の範囲に対戦相手がいないと把握できません。 対戦相手の座標を把握していない場合は nil が返されます。 マップ情報の取得を繰り返し行っている場合、情報が上書きされていくため、一度把握した対戦相手の座標を見失う場合があります。



⑨妨害キャラクタのx座標、y座標

ブロック	 
コード	<code>enemy_x</code> <code>enemy_y</code>
実行内容	<ul style="list-style-type: none"> 妨害キャラクタの座標を呼び出します。 最後にマップ情報の取得で把握した妨害キャラクタの座標(x,y)を返します。
引数	-
解説	<ul style="list-style-type: none"> 得られる情報は、マップ情報の取得で把握した時点の情報です。 妨害キャラクタの座標は、マップ情報の取得の範囲に妨害キャラクタがいなくても把握できます。

⑩ゴールのx座標、y座標

ブロック	 
コード	<code>goal_x goal_y</code>
実行内容	<ul style="list-style-type: none"> •ゴールの座標を呼び出します。 •ゴールの座標が返されます。
引数	-
解説	<ul style="list-style-type: none"> •ゴールの座標は、マップ情報の取得に関わらず呼び出し可能です。

⑪キャラクタのx座標、y座標

ブロック	 
コード	<code>player_x player_y</code>
実行内容	<ul style="list-style-type: none"> •キャラクタの座標を呼び出します。 •キャラクタの座標が返されます。
引数	-
解説	<ul style="list-style-type: none"> •キャラクタの座標は、マップ情報の取得に関わらず呼び出し可能です。

⑫ プレイヤーの名前をつける


ブロック	
コード	set_name(プレイヤー名)
実行内容	・プレイヤー名で設定した名前がゲーム時にプレイヤー名として表示されます。
引数	・引数としてプレイヤー名を入力します。
解説	・文字列14文字まで入力できます。

⑬ ターンを終了する

ブロック	
コード	turn_over
実行内容	・現在のターンを終了させ、次のターンを待ちます。
引数	-
解説	・必ずターンの最後に1回実行する必要があります。 (ターン終了を実行しないとタイムアウトでゲームが終了します。)

⑭爆弾を置く

①③④⑭のいずれか2回まで

ブロック	 文 <code>set_bomb([0, 1])</code>
コード	<code>set_bomb(引数)</code>
実行内容	<ul style="list-style-type: none"> 爆弾を現在地又は隣接する東西南北のマスを設置します。
引数	<ul style="list-style-type: none"> 爆弾を設置したい座標を配列で指定します。 引数を省略した場合は現在地に爆弾を設置します。
解説	<ul style="list-style-type: none"> 爆弾ブロックはスモウルビー甲子園専用ブロックとしては用意していません。[その他]の中の文ブロックを利用してください。 爆弾は、他アイテムのない空間に置くことができます。 爆弾は1ラウンドに2回まで設置できます。 無効な座標を指定した場合は設置されませんが、爆弾は1つ消費され、使用回数もカウントされます。 両プレイヤーが同じ座標に同時に爆弾を設置した場合、両プレイヤーとも設置は成功しますが爆弾は1つだけ設置されたことになります。

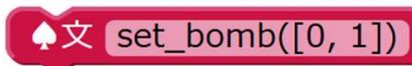
[甲子園メソッドの使用制限]

• 次の命令には1ターンでの使用回数に制限があるので、注意してください。
(使用回数を越えた命令は無視されます。)

→1ターン1回



→次の4つの使用回数はいずれか2回
(移動する以外は同じ命令を2回使用することも可能です。)



(4) 簡単なAIプログラムを作ってみる

・困ったときは

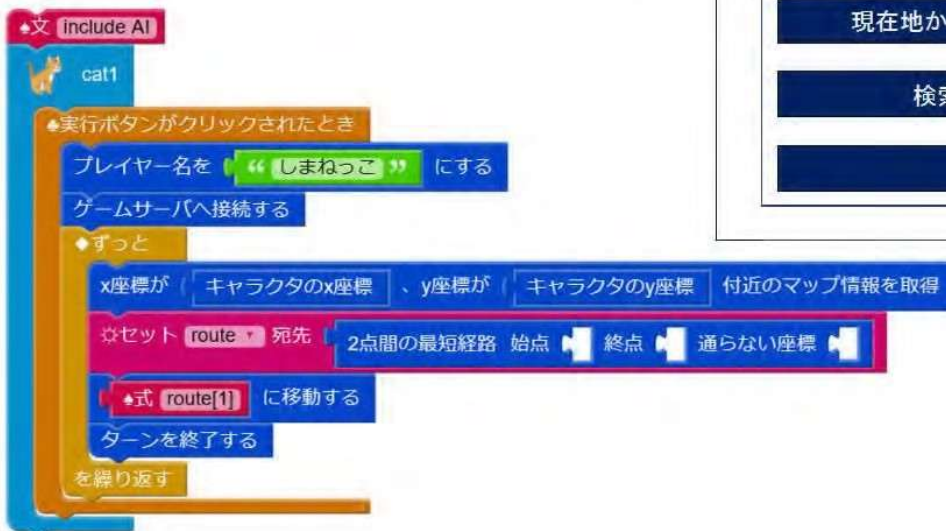
- ・思い通りに動かない
- ・タイムアウトしてしまう
- ・エラーが出た

○プログラムの実行の記録が「ログファイル」というファイルに書き出されています。それを見ると動かなかった原因がわかるかもしれません。うまくいかないときはログファイルを見てみましょう。

○プレイヤー1の結果は player1.log というログファイルに記録されています。メモ帳で開いて見ることができます。ログファイルをダブルクリックすると開きます。

・次のプログラムを作ってみましょう

最短経路でゴールを目指す



最初にファイル名を変更してください。

ファイル名には、アルファベットか数字を使ってください。

「.rb」を消さないでください。消してしまったときは、半角で入力してください。

変更したらエンターキーを押してください。

次にプレイヤー名を変更してください。

ひらがな、カタカナ、漢字、アルファベット、数字が使えます。

14文字まで(15文字を超えるとエラーが発生します。)

「ゴールを目指す」=ゴールに到着するように移動することです。

キャラクタを移動するには「移動する」というブロックを使います。甲子園ジャンルの中に入っています。



「移動する」のブロックの空欄には、移動先の座標を入れます。

移動先の座標を調べるために「2点間の最短経路」のブロックを使います。



ただし、正しい座標を調べるためには、マップ情報を取得していなければなりません。そのために「マップ情報を取得」のブロックを使います。



「マップ情報を取得」を実行したあと、「2点間の最短経路」を実行すると、正しい座標を調べることができます。

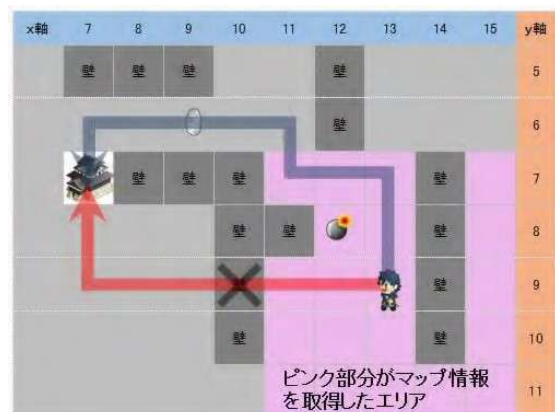
「マップ情報の取得」のブロックに、以下のように「キャラクタのx座標」と「キャラクタのy座標」のブロックを組み合わせると、「今、自分がいる座標を中心として5マス×5マスの範囲」のマップ情報を取得します。



「2点間の最短経路」のブロックは、以下のように「セット」のブロックと組み合わせます。



- 戻り値は、経路順の座標が返される(0からの番号で指定可能)
- 引数を省略した場合は次の設定で経路を検索
 - ▶ 始点: プレイヤーキャラクタの現在地
 - ▶ 終点: ゴール
 - ▶ 通らない座標: 設定なし
- 未探索エリアは通行可能として経路検索(通れない経路を返す場合がある)
- 変数をつけることで後から経路検索の結果の呼び出し可能



この例では、変数「route」の中に、「今、自分がいる座標からゴールまでの経路順の座標の配列」が格納されます。図の座標の場合、以下のような座標の配列が入ります。

```
[[13,9],[13,8],[13,7],[12,7],[11,7],[11,6],[10,6],[9,6],[8,6],[7,6],[7,7]]
```

今、自分がいる座標が[13,9]で、移動先の座標は[13,8]です。

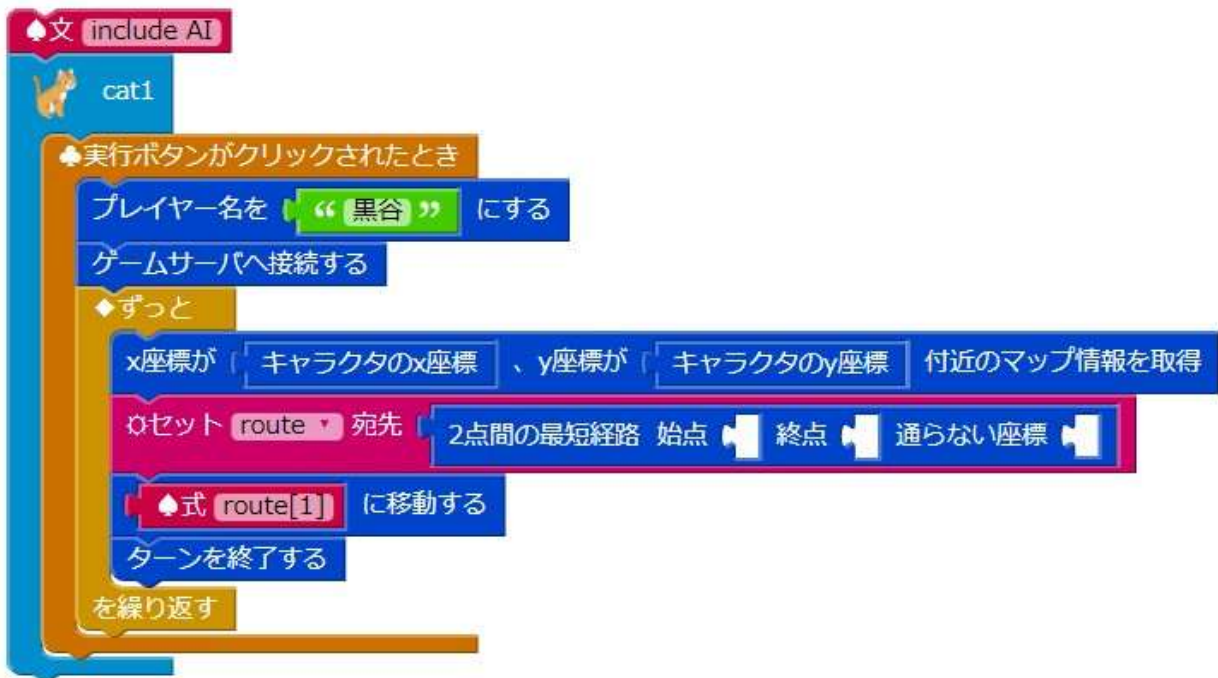
route の中身の座標配列から[13,8]を取り出せれば良いということになります。

そのためには route[1] という式を使います。

式ブロックの空欄に「route[1]」と入力し、移動するブロックと組み合わせ、以下のようなブロックを作ります。



ブロックを順番に組み合わせて、以下のように組み立てます。

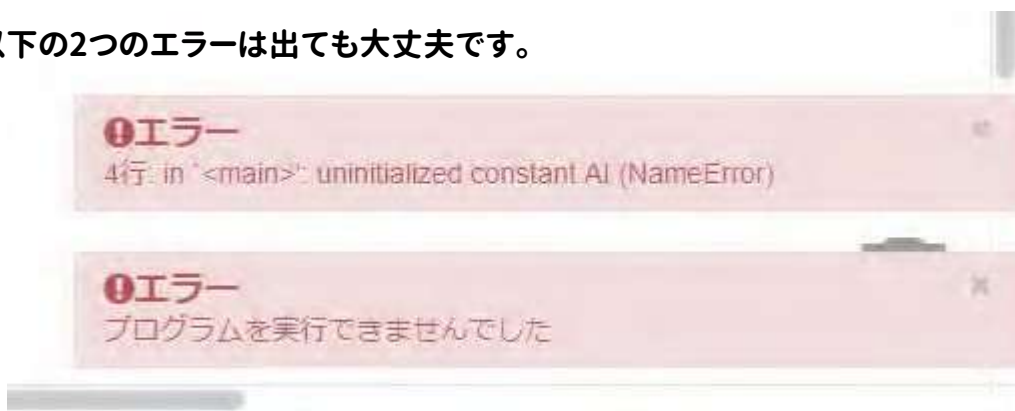


組み立てができたなら右上 「実行」 ボタンをクリックします。



プログラムの保存と文法チェックをしてくれます。

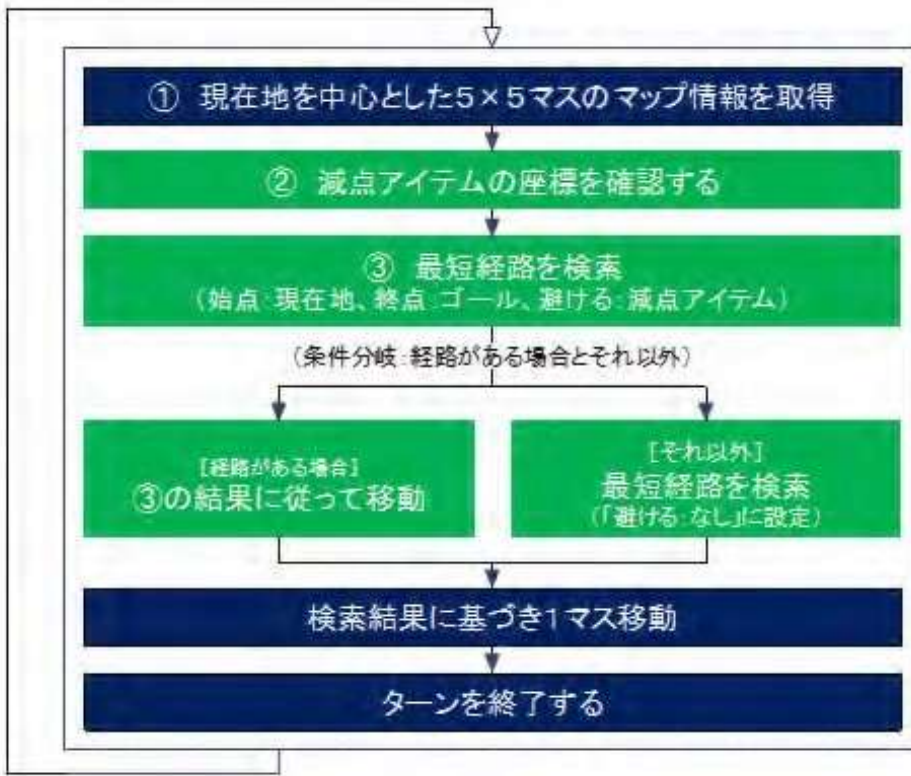
以下の2つのエラーは出ても大丈夫です。



対戦環境を起動し、プレイヤー1のAI プログラムを自分のAIプログラムに取り換えて、対戦をスタートしてみてください。ゴールまで進むことを確認してください。

- 先ほど作ったプログラムを改良します。
ファイル名を変更し、エンターキーを押してください。

減点アイテムを避けてゴールを目指す



```

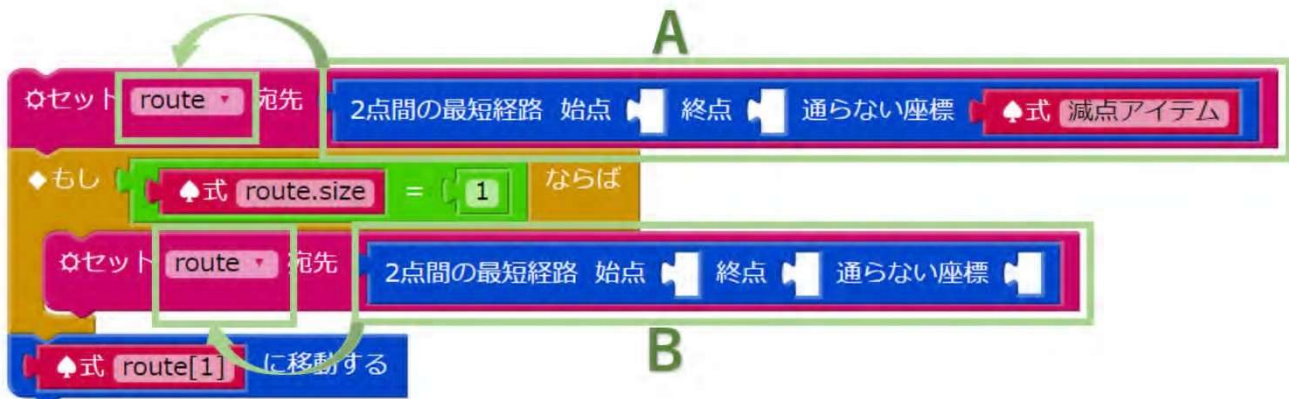
    include AI
    cat1
    実行ボタンがクリックされたとき
      プレイヤー名を "しまねっこ2" にする
      ゲームサーバへ接続する
      ずっと
        x座標が キャракタのx座標、y座標が キャラクタのy座標 付近のマップ情報を取得
        減点アイテム宛先 範囲内の地形・アイテム 中心座標 式 [8,8] 範囲 式 17 地形・アイテム
        route宛先 2点間の最短経路 始点 終点 通らない座標 式 減点アイテム
        もし 式 route.size = 1 ならば
          route宛先 2点間の最短経路 始点 終点 通らない座標
        route[1] に移動する
        ターンを終了する
      を繰り返す
  
```



- 指定したマップ要素やアイテムの座標を確認するためのメソッド
- 取得したマップ情報から情報を呼び出すため、未探索エリアの情報は取得できません。
- 引数として以下を指定
 - ▶ 中心座標: 呼び出す範囲の中心座標を指定
⇒ マップ全体の場合は [8,8]
 - ▶ 範囲: 呼び出す範囲のマップサイズ指定
⇒ マップ全体の場合は 17
 - ▶ 地形・アイテム: 呼び出す地形・アイテムの値を指定
⇒ 省略した場合は 減点アイテム を呼び出す
⇒ 値については別途説明します
- 変数をつけることで後からこの情報を呼び出すことが可能



- 減点アイテムを避けてゴールする経路がない場合もあり
- このため、経路がなかった場合の行動もプログラムする必要あり
⇒ ある場合とない場合の2種類の行動が必要
- こうした場合の条件分岐を使用
- 条件に合致した場合(真)のみ、ブロックの中のプログラムを実行し、合致しない場合は、このブロックの処理は実行されない



- 上のブロックでは変数「route」が2つあり、それぞれ中身が違う
- プログラムは上から順番に処理
⇒ 最初にAの内容が変数「route」となる
⇒ 減点アイテムを避ける経路がない場合は、Bが実行され、Bの内容が変数「route」になる
⇒ 減点アイテムを避ける経路がある場合は、Bの内容は実行されないため変数「route」はAのまま

- プログラムが完成したら、「実行」ボタンをクリックしてプログラムの保存と文法チェックをします。
- 対戦環境を起動し、プレイヤー1のAIプログラムを取り換えます。
- 減点アイテムを避けていることが確認できるように、マップを map3 に変更します。
- 対戦をスタートして、減点アイテムを避けながらゴールまで進むことを確認してください。


5. スモウルビー3.0の使い方

スモウルビープログラミング甲子園20xxではスモウルビー3.0(<https://smalruby.app>)を用いてAIプログラムを作成することができます。詳しくは以下の動画を参照ください。


<https://www.youtube.com/watch?v=eyaDyHGs41Q>

(1) スモウルビー3.0版メソッドの種類と使い方

① 1マス移動する


ブロック	
コード	<code>koshien.move_to("x:y")</code>
実行内容	<ul style="list-style-type: none">指定した座標にプレイヤーが1マス移動します指定できるのは現在地から東西南北の1マスです。(斜めには移動できません)
引数	移動先のX座標、Y座標を "x:y" 形式の文字列で指定します
解説	<ul style="list-style-type: none">移動できるのは空間と水たまりだけで、壁には移動できません。移動できない座標を指定した場合、使用回数はカウントされますが、実行は無視されます。水たまりに移動した場合は、次回の移動命令が無視されます。(使用回数はカウントされます。)

②最短経路を検索する

ブロック	
コード	<pre>koshien.calc_route(result: list("\$最短経路"), src: "0:0", dst: "0:0", except_cells: list("\$通らない座標"))</pre>
実行内容	<ul style="list-style-type: none"> 指定した2点間の最短ルートを探します。 座標(`"x:y"`形式)のリストを通らない座標として指定すると、その座標を通らない経路を探します。 始点から終点までの座標(`"x:y"`形式)のリストを指定したリストに保存します。
引数	<ul style="list-style-type: none"> <code>src</code>: 始点の座標`"x:y"`形式を指定します。 <code>dst</code>: 終点の座標`"x:y"`形式を指定します。 <code>except_cells</code>: 経路を探すときに通ってほしくない座標(`"x:y"`形式)のリストを指定します。 <code>result</code>: 探した最短経路の座標(`"x:y"`形式)のリストを保存するリストを指定します。
解説	<ul style="list-style-type: none"> 最短経路の座標のリストは「始点, 次の移動先, …経路順の座標…, 終点」の順番に並んでいます。 最短経路の座標のリストから取得するにはリスト内の位置を0,1,2,3…の番号で指定します。 指定した条件での経路がない場合は始点の座標が1つだけのリストが保存されます。 <code>get_map_area</code> 命令でマップ情報を取得していない範囲のマス(未探索のマス)は全て移動可能とみなして経路探索するので注意が必要です。 ただし、判明している空間マスがあれば、そちらを通る経路を優先します。
引数の指定例	<ul style="list-style-type: none"> 例1: 始点、終点、通らない座標を指定して、ある座標(13, 9)からアイテムのある(7, 7)への最短経路を探し、その経路で1マス移動する <pre>list("\$通らない座標").clear list("\$通らない座標").push("9:9") koshien.calc_route(result: list("\$最短経路"), src: "13:9", dst: "7:7", except_cells: list("\$通らない経路")) koshien.move_to(list("\$最短経路")[1])</pre> 例2: 通らない座標のみを指定して、ゴールまでの最短経路を探し、その経路で1マス移動する <pre>list("\$通らない座標").clear list("\$通らない座標").push("9:9") koshien.calc_route(result: list("\$最短経路"), except_cells: list("\$通らない経路")) koshien.move_to(list("\$最短経路")[1])</pre> 例3: ゴールまでの最短経路を探し、その経路で1マス移動する <pre>koshien.calc_route(result: list("\$最短経路")) koshien.move_to(list("\$最短経路")[1])</pre>


③5マス×5マスのマップ情報を取得

①③④⑭のいずれか2回まで


ブロック	
コード	<code>koshien.get_map_area("x:y")</code>
実行内容	・指定した範囲(5マス×5マス)のマップ情報を取得します。
引数	・マップ情報を取得したい範囲の中心座標のX座標、Y座標を "x:y" 形式の文字列で指定します
解説	<ul style="list-style-type: none"> ・取得できるマップ情報は、指定した範囲の以下の情報です。 <ul style="list-style-type: none"> ▶ マップ構成(空間・壁・水たまり・ゴール・壊せる壁) ▶ 加点アイテム、減点アイテムがある場合は、その座標と種類解説 ▶ 対戦相手が指定範囲内にいる場合はその座標 ▶ 妨害キャラクタの現ターン開始時点の座標と前ターン開始時点の座標(妨害キャラクタのみは指定範囲内にいなくても情報取得が可能) ・取得した情報はAIライブラリに保存され、後から呼び出すことができます。

④ダイナマイトを置く


①③④⑭のいずれか2回まで

ブロック	
コード	<code>koshien.set_dynamite("x:y")</code>
実行内容	・ダイナマイトを現在地又は隣接する東西南北のマスを設置します。
引数	<ul style="list-style-type: none"> ・ダイナマイトを設置したい座標のX座標、Y座標を "x:y" 形式の文字列で指定します。 ・引数を省略した場合は現在地にダイナマイトを設置します。
解説	<ul style="list-style-type: none"> ・ダイナマイトは、空間または水たまりの上に置くことができます。アイテムがあるマスには設置できません。 ・ダイナマイトは1ラウンドに2回まで設置できます。 ・無効な座標を指定した場合、設置されませんが、ダイナマイトは1つ消費され、使用回数もカウントされます。 ・両プレイヤーが同じ座標に同時にダイナマイトを設置した場合、両プレイヤーとも設置は成功しますが、ダイナマイトは1つだけ設置されたことになります。 ・ダイナマイトは設置したターンの終了時に爆発します。 ・ダイナマイトが爆発すると、ダイナマイトのマスに隣接する「壊せる壁」は「空間」になり、次のターンから移動可能になります。 ・ダイナマイトの爆発は「壊せる壁」以外の地形、プレイヤー、妨害キャラクタ、アイテムに影響を与えません。 ・ダイナマイトが爆発したマスや隣接したマスにプレイヤーがいても減点されることはありません。


⑤ 指定した座標のマップ情報を呼び出す

ブロック	
コード	<code>koshien.map("x:y")</code>
実行内容	<ul style="list-style-type: none"> • 指定した座標のマップ情報を呼び出します。 • 指定した座標のマップ情報が返されます。
引数	<ul style="list-style-type: none"> • 呼び出したい座標のX座標、Y座標を“x:y”形式の文字列で指定します。
解説	<ul style="list-style-type: none"> • 呼び出せるマップ情報は「③マップ情報の取得」で取得した情報です。 • マップ情報の取得を実行していない座標を指定した場合は、-1が返されます。 • マップエリア外を指定した場合は、nilが返されます。


⑥ マップ全体のマップ情報を呼び出す

ブロック	
コード	<code>koshien.map_all</code>
実行内容	<ul style="list-style-type: none"> • マップ全体のマップ情報を呼び出します。 • マップ全体のマップ情報が返されます。
引数	-
解説	<ul style="list-style-type: none"> • 呼び出せるマップ情報は「③マップ情報の取得」で取得した情報です。 • マップ情報の取得を実行していない座標は、-1が返されます。


⑦指定した範囲に、指定した要素が存在するかどうかを確認する

ブロック	
コード	<pre>koshien.locate_objects(result: list("\$リスト名"), sq_size: 範囲, cent: "x:y", objects: "アイテム、地形を示す記号")</pre>
実行内容	<ul style="list-style-type: none"> 指定した範囲に、指定した要素が存在するかどうかを確認します。 指定した要素がある座標がリストに保存されます。
引数	<ul style="list-style-type: none"> result: 結果を保存するリスト。あらかじめ作成しておく必要があります。 sq_size: 指定する範囲の縦横のマス目の長さ(省略時は5) cent: 指定する範囲の中心座標(省略時は現在地) objects: 確認したい要素の値。複数の場合は区切り文字なしで指定(省略時は減点アイテム全種類 `ABCD`)
解説	<ul style="list-style-type: none"> 結果は指定したリストに保存します。 リストの中の各座標はy座標が小さい順に並んでいます(y座標が同じ場合はx座標の小さいほうが先になります)。 マップ情報を取得していない座標の情報は確認できません。
引数の指定例	<ul style="list-style-type: none"> 例1 マップ全体の加点アイテムの座標を確認する場合 <pre>koshien.locate_objects(result: list("\$加点アイテム"), sq_size: 17, cent: "8:8", objects: "abcde")</pre> 例2 マップ全体の水たまりの座標を確認する場合 <pre>koshien.locate_objects(result: list("\$水たまり"), sq_size: 17, cent: "8:8", objects: "4")</pre>


⑧対戦キャラクタ座標

ブロック	
コード	koshien.other_player
実行内容	<ul style="list-style-type: none"> ・対戦相手の座標を呼び出します。 ・最後にマップ情報の取得で把握した対戦相手の座標("x:y"形式)を返します。
引数	-
解説	<ul style="list-style-type: none"> ・得られる情報は、マップ情報の取得で把握した時点の情報です。 ・対戦相手の座標は、マップ情報の取得の範囲に対戦相手がいないと把握できません。 ・対戦相手の座標を把握していない場合は nil が返されます。 ・マップ情報の取得を繰り返し行っている場合、情報が上書きされていくため、一度把握した対戦相手の座標を見失う場合があります。


⑨妨害キャラクタの座標

ブロック	
コード	koshien.enemy
実行内容	<ul style="list-style-type: none"> ・妨害キャラクタの座標を呼び出します。 ・最後にマップ情報の取得で把握した妨害キャラクタの座標("x:y"形式)を返します。
引数	-
解説	<ul style="list-style-type: none"> ・得られる情報は、マップ情報の取得で把握した時点の情報です。 ・妨害キャラクタの座標は、マップ情報の取得の範囲に妨害キャラクタがいなくても把握できます。

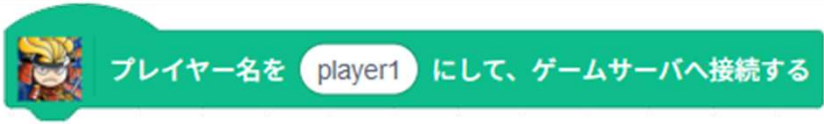
⑩ゴールの座標

ブロック	 ゴール ▼ の 座標 ▼
コード	koshien.goal
実行内容	<ul style="list-style-type: none"> •ゴールの座標を呼び出します。 •ゴールの座標("x:y"形式)が返されます。
引数	-
解説	<ul style="list-style-type: none"> •ゴールの座標は、マップ情報の取得に関わらず呼び出し可能です。


⑪キャラクタの座標

ブロック	 プレイヤー ▼ の 座標 ▼
コード	koshien.player
実行内容	<ul style="list-style-type: none"> •キャラクタの座標を呼び出します。 •キャラクタの座標("x:y"形式)が返されます。
引数	-
解説	<ul style="list-style-type: none"> •キャラクタの座標は、マップ情報を取得していなくても参照できます。

⑫プレイヤーの名前をつける


ブロック	
コード	<code>koshien.connect_game(name: "プレイヤー名")</code>
実行内容	<ul style="list-style-type: none"> ・ゲーム時に指定したプレイヤー名が表示されます。 ・ゲームサーバへ接続します。
引数	<ul style="list-style-type: none"> ・name: プレイヤー名を指定します。
解説	<ul style="list-style-type: none"> ・文字列14文字まで入力できます。

⑬ターンを終了する


ブロック	
コード	<code>koshien.turn_over</code>
実行内容	<ul style="list-style-type: none"> ・現在のターンを終了させ、次のターンを待ちます。
引数	-
解説	<ul style="list-style-type: none"> ・必ずターンの最後に1回実行する必要があります。(ターン終了を実行しないとタイムアウトでゲームが終了します。)

⑭爆弾を置く

①③④⑭のいずれか2回まで

ブロック	
コード	<code>koshien.set_bomb("x:y")</code>
実行内容	<ul style="list-style-type: none"> 爆弾を現在地又は隣接する東西南北のマ스에設置します。
引数	<ul style="list-style-type: none"> 爆弾を設置したい座標のX座標、Y座標を“x:y”形式の文字列で指定します。 引数を省略した場合は現在地に爆弾を設置します。
解説	<ul style="list-style-type: none"> 爆弾は、空間または水たまりの上に置くことができます。アイテムがあるマスには設置できません。 爆弾は1ラウンドに2回まで設置できます。 無効な座標を指定した場合は設置されませんが、爆弾は1つ消費され、使用回数もカウントされます。 両プレイヤーが同じ座標に同時に爆弾を設置した場合、両プレイヤーとも設置は成功しますが爆弾は1つだけ設置されたことになります。

②-2 ゴールまでの最短経路を探索する

ブロック	
コード	<code>koshien.calc_route(result: list("\$最短経路"))</code>
実行内容	<ul style="list-style-type: none"> 通らない座標を考慮しないゴールまでの最短経路を探索します。 最短経路はリストに保存されます。
引数	-
解説	<ul style="list-style-type: none"> ゴールまでの最短経路を探索します。 「②最短経路を探索する」の、始点を現在地、終点をゴール、通らない座標の指定なし、に固定したものです。

6. 便利なメソッドの紹介

[紹介する内容]

- (1) 草薙剣の座標を調べる
- (2) 配列に含まれる座標の数を確認する
(例: 経路検索した経路があるかを確認する)
- (3) 一番近い加点アイテムを目指して移動する
- (4) ターン数を管理する
- (5) ダイナマイトを置く
(例: プレイヤーが壊せる壁の隣のマスに来たらダイナマイトを置く)

[参考]

- ・ スモウルビー甲子園で用意した命令ブロックは、ジャンル「甲子園」の中に入っています。
- ・ また、ジャンルとブロックの色は一緒になっていますので、色を参考にブロックを探すこともできます。
- ・ よく使う関数ブロックや文・式ブロックは次のジャンルに入っています。

■ 動き				
★ 見た目				
♪ 音				
● ペン				
⊙ データ	⇒ 関数ブロック			など
♣ イベント				
◆ 制御	⇒ 条件分岐ブロック			など
◎ 調べる				
演算				
♠ その他	⇒ 文・式ブロック	 		など
甲子園	⇒ 甲子園ブロック	 		など

(1) 草薙剣の座標を調べる

<ブロックの例>

セット kusanagi 宛先 範囲内の地形・アイテム 中心座標 式 [8,8] 範囲 式 17 地形・アイテム 式 ["e"]

- ・「範囲内の地形・アイテム」のブロックを使うと、指定したマスの種類(壁や水たまりなど)の座標や、指定したアイテムの座標を取得することができます。

範囲内の地形・アイテム 中心座標 範囲 地形・アイテム

- ・中心座標[8,8]、範囲17を指定すると、マップ全体から指定したアイテムなどの情報を取得することができます。以下のように式ブロックを利用します。

式 [8,8]

式 17

- ・草薙剣の座標を確認したい場合は、地形・アイテムの-slotに以下のような式ブロックをあてはめます。

式 ["e"]

[地形・アイテムの-slotにあてはめる式ブロックの例]

▶水たまりの場合

地形・アイテム 式 [4]

▶加点アイテム全部の場合

地形・アイテム 式 ["a","b","c","d","e"]

▶シロイルカ(加点アイテム40点)の場合

式 ["d"]

▶減点アイテム全部の場合

地形・アイテム (空欄のまま)

▶爆弾(減点アイテム40点)の場合

地形・アイテム 式 [D]

▶壊せる壁

地形・アイテム 式 [5]

⇒引数は配列で指定するため、必ず[]の中に指定したい値を入力してください。

また、文字の場合は" "を必ず付けてください。

⇒指定できるマップ要素、アイテムの種類と値とは次のとおりです。

⇒引数を省略した場合は減点アイテム4種類の場所が返されます。

[アイテム等の種類と値]

加点アイテム	値	得点
お茶	a	10
和菓子	b	20
丁銀	c	30
シロイルカ	d	40
草薙剣	e	60

減点アイテム	値	得点
毒キノコ	A	△10
蛇	B	△20
トラバサミ	C	△30
爆弾	D	△40

マスの種類	値
未探索のマス	-1
空間	0
壁	1
蔵(外壁)	2
ゴール	3
水たまり	4
壊せる壁	5

- ・「セット ○○ 宛先 □□」というブロックを用意して、変数名を付けます。変数名は自由に付けることができます。ここでは「kusanagi」とします。宛先に、「範囲内の地形・アイテム」のブロック、中心座標の式ブロック、範囲の式ブロック、草薙剣を表す式ブロックを組み合わせたものを入れます。これで後から「kusanagi」という変数を参照することで、草薙剣の座標を取り出すことができます。



[注意]

- ・マップ探索を実施したエリアの情報のみが確認できます。(マップ探索をしていないエリアの情報は確認できません。)
- ・また、確認できる情報はマップ探索を行った時点での情報です。アイテムを対戦相手が取得してなくなっているなど、情報が古くなっている場合がありますので、注意してください。

(2) 配列に含まれる座標の数を確認する

(例: 経路検索した経路があるかを確認する)

<ブロックの例>



- ・Rubyのsizeメソッドを使用することにより、配列に含まれる座標の数を確認することができます。(sizeメソッドでは、指定した配列に含まれる要素の数を返します。)
 - ・例えば、経路検索では、指定した条件の経路がない場合は、始点の座標を1つだけ返します。このことから、sizeメソッドを使うことにより、経路のありなしを確認できます。(戻り値が1の場合は経路がないことが分かります。)
- また、sizeメソッドを使って、加点アイテムの有無なども確認できます。

[使用例]

“減点アイテムを避けてゴールする経路を検索し、もし、この経路がない場合は、ゴールまでの最短経路を検索する”



[ブロックの処理内容]

- ・減点アイテムの座標を確認し、その結果に「traps」と名前を付ける
- ・始点: 現在地、終点: ゴール、通らない座標: 減点アイテムで、経路を検索し、その結果に「route」と名前を付ける
- ・もし、「route」に含まれる座標の数が1であれば(経路がない場合は)、始点: 現在地、終点: ゴール、通らない座標: 未設定で経路検索する


(3) 一番近い加点アイテムを目指して移動する
 <ブロックの例>



- マップ上の指定した要素(アイテムマップ要素)の座標を確認する方法は(3)でやったとおりですが、今度はマップ上の加点アイテムの座標を確認し、一番近い加点アイテムを終点として移動するブロックを作ります。
- 最初に加点アイテムの座標を確認し、この結果に「items」という名前を付けます。



- 次に「items」に含まれる座標を現在地から近い順に並べ替えます。
- (「items」に含まれる座標は、y軸の座標数が小さい順に並んでいます。)この処理を「sort_by!」「calc_route」「size」メソッドを使って行います。

<p>sort by!</p>	<p>{ } で囲まれたコードを使って配列に含まれる要素を並べ替え、配列自身を変更するメソッドです。({ } で囲まれた部分を「ブロック」と呼びます)</p> <p>配列.sort_by!{ ブロックパラメータ 繰り返したい処理 }</p> <p>ブロックパラメータに、配列に含まれる各要素を入れながら繰り返したい処理を実行し戻り値を集めます。集めた戻り値を「<=>」演算子で比較して、小さい順に要素を並べ替えます。</p> <p>例)</p> <pre>animals = ["mouse", "cat", "elephant", "lion"] animals.sort_by! { animal animal.size }</pre> <p>⇒ animals の要素が次のように「文字数の小さい順」に並べ替えられます。 ["cat", "lion", "mouse", "elephant"]</p> <p>※ 文字列(“と”で囲まれたもの)に対して size メソッドを実行すると、その文字数を返します。</p>
<p>calc_route</p>	<p>スモウルビー甲子園で用意した 2 点間の最短経路を検索するメソッドです。(下のブロックと同じ内容です。)</p>  <p>引数として始点の座標(src)、終点の座標(dst)、通らない座標の配列(except_cells)を指定できます。</p> <p>なお、引数を省略した場合は、始点:現在地、終点:ゴール、通らない座標:未設定で経路検索します。</p> <p>calc_route(src: [始点の座標], dst: [終点の座標], except_cells: [通らない座標の配列])</p> <p>例)</p> <pre>calc_route(src: [9,9], dst: [7,7], except_cells: [[9,7]])</pre> <p>⇒ 次のような経路順の座標配列が返されます。 [[9, 9], [9, 8], [8, 8], [8, 7], [7, 7]]</p>

size	<p>配列の要素の数を返すメソッドです。</p> <p>配列.size</p> <p>例)</p> <p>route = [[11,8],[11,7],[10,7],[9,7],[8,7],[7,7]] route.size</p> <p>⇒座標の個数6が返されます。</p>
------	--

• 上で説明した「sort_by!」「calc_route」「size」メソッドを次のように組み合わせます。

```
♣文 items.sort_by!{|item| calc_route(dst: item).size}
```

[ブロックの処理内容]

- 「items」に含まれる加点アイテムの各座標を「item」というブロックパラメータに入れてブロックを繰り返し実行する(加点アイテムの座標が5つであれば5回繰り返す)。
- ブロックの内容は、以下の座標を設定し calc_route を実行すること 始点:現在地 終点:「items」に含まれる各加点アイテムの座標通らない座標:未設定
- calc_route の結果(現在地から各加点アイテムまでの座標の数)の小さい順に「items」の中の座標を並べ替える

• 並べ替えの結果、一番近い加点アイテムの座標が「items」の最初の要素にあるので、その要素を取り出して、target という変数に入れます。

```
○セット target ▾ 宛先 ♣式 items.first
```

first メソッドは、配列の最初の要素を返します。
items.first では、「items」の先頭の座標を返します。

• 一番近い加点アイテムを取得すると、次に近い加点アイテムを目指して移動する必要があります。この処理を正しく行うためには、マップ情報を更新し、「items」の情報を更新していく必要があります。このため、この一連のブロックの先頭にマップ情報の取得ブロックを組み込んでおきます。⇒これにより、ターンのたびに、まずは現在地周辺のマップ情報を取得し、「items」の情報が更新されます。⇒ただし、情報が更新されるのは現在地周辺(5マス×5マス)だけのため、この範囲外で対戦相手が加点アイテムを取得した情報は取得できません。

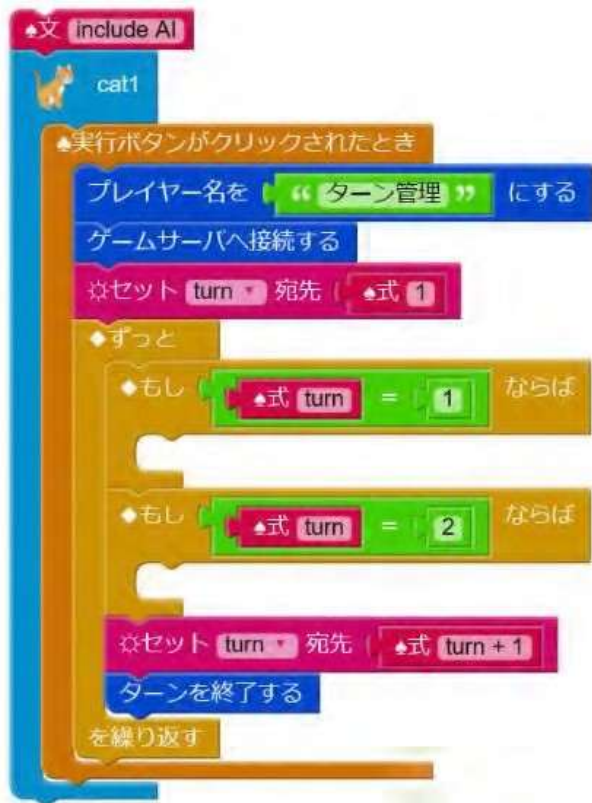
```
x座標が [キャラクタのx座標]、y座標が [キャラクタのy座標] 付近のマップ情報を取得
○セット items ▾ 宛先 範囲内の地形・アイテム 中心座標 ♣式 [8,8] 範囲 [17] 地形・アイテム ♣式 ["a","b","c","d","e"]
♣文 items.sort_by!{|item| calc_route(dst: item).size}
○セット target ▾ 宛先 ♣式 items.first
```

• 最後に「target」を終点に設定して移動するブロックを組み込んで完成です。

```
x座標が [キャラクタのx座標]、y座標が [キャラクタのy座標] 付近のマップ情報を取得
○セット items ▾ 宛先 範囲内の地形・アイテム 中心座標 ♣式 [8,8] 範囲 [17] 地形・アイテム ♣式 ["a","b","c","d","e"]
♣文 items.sort_by!{|item| calc_route(dst: item).size}
○セット target ▾ 宛先 ♣式 items.first
○セット route_target ▾ 宛先 2点間の最短経路 始点 [ ] 終点 [target] 通らない座標 [ ]
♣式 route_target[1] に移動する
```

※「便利なメソッド」の中の「座標を自分に近い順に並べ替える」ブロック
Ruby言語の命令を手入力しなければなりません。打ち間違いが起きやすいので、ゆっくり気を付けながら入力してください。

(4) ターン数を管理する <ブロックの例>



- 1ターンの行動として、AIプログラムの「◆ずっと を繰り返す」の中のプログラムが1回実行されます。よって、「◆ずっと を繰り返す」の実行回数がターン数と同じ数になります。このことから、「◆ずっと を繰り返す」が実行された回数を変数に入れることによりターン数を管理することができます。
- 最初に「◆ずっと を繰り返す」より前に「turn = 1」を設定します。(これで最初のプログラム実行は「turn = 1」としてスタートします。)



- 次に「ターンを終了する」の前に「turn = turn + 1」を組み込みます。(ターン終了時に「turn」に1を加えることにより、ターン数が加算されていきます。)



・変数「turn」を参照することで現在のターン数がわかりますので、以下のようなブロックを作ることによって、ターンごとの行動を変えることができます。



※「便利なメソッド」の中の「ターン数の管理」

「もし～ならば」ブロックがたくさん出てきますので、入れ間違えないように注意してください。最初に紙に書いておくと良いかもしれません。

(5) ダイナマイトを置く

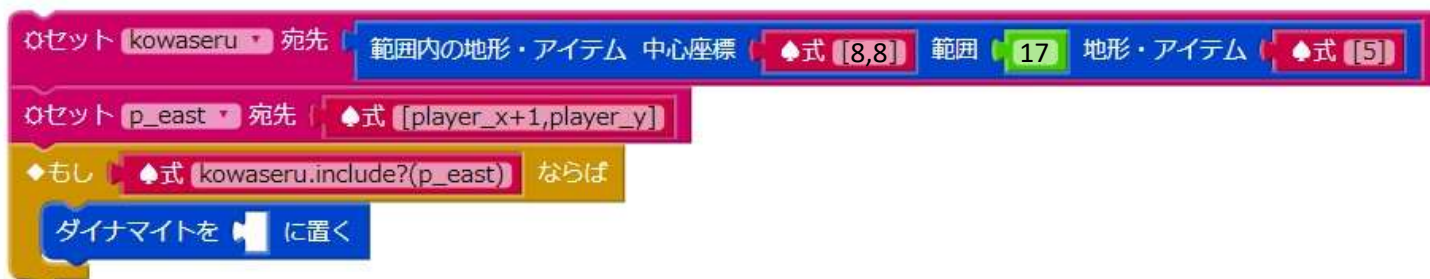
(例: プレイヤーが壊せる壁の隣のマスに来たらダイナマイトを置く)

<ブロックの例>

・プレイヤーが壊せる壁の西隣のマスに来た場合にダイナマイトを置くブロック例

<ブロックの例>

・プレイヤーが壊せる壁の西隣のマスに来た場合にダイナマイトを置くブロック例



「プレイヤーが壊せる壁の西隣のマスに来た」という状況は「プレイヤーの東側の隣のマスが壊せる壁になった」という状況であると読みかえることができます。

「プレイヤーの東側の隣のマス」の座標を調べるためには、プレイヤーの座標を調べる必要があります。

「プレイヤーの座標」は

[player_x, player_y]

と表すことができます。(5.AI プログラムの作り方(3)スモウルビー甲子園メソッドの種類と使い方

①①キャラクターのx座標、y座標 参照)

「プレイヤーの東側の隣のマス」の座標は、x座標が1多く、y座標は同じなので、

[player_x+1, player_y]

と表すことができます。

「プレイヤーの東側の隣のマスが壊せる壁になった」という条件は、

「複数ある「壊せる壁」の座標のうち、「プレイヤーの東側の隣のマス」の座標と一致するものがある」と言い換えることができます。

Ruby には、複数のものの中から一致するものを見つけることができる「include?」というメソッドがあります。このメソッドを使います。

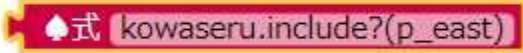
変数「kowaseru」を作り、壊せる壁の座標をセットします。



変数「p east」を作り、プレイヤーの東側の隣のマスの座標をセットします。



式ブロックに以下の式を入力します。
kowaseru.include?(p_east)



「もし ならば」のブロックに式ブロックを入れます。



これらのブロックを順番につなぎます。



「もし ならば」のブロックの中に「ダイナマイトを置く」のブロックをあてはめたら完成です。



プレイヤーが壊せる壁の北隣や南隣、東隣に来た場合のブロックはどうすれば作れるか、皆さん考えてみてください。

7. 作戦の考え方

(1) やり方1

- ▶ やりたいことを書き出す
- ▶ 実装するためのプログラムを考える
- ▶ ゲームで試す
- ▶ 改良する

例)

- やりたいこと
「加点アイテムを取ってからゴールに向かう」
- 実装するためのプログラムを考える
「最初に全マップの情報を取得する」
「加点アイテムの座標を確認する」
「加点アイテムの座標を近い順に並べ替える」
「一番近い加点アイテムに向かう」
「加点アイテムがなくなったらゴールに向かう」
- ゲームで試す
「加点アイテムを全部とっているとゴールできないなあ(50ターン経過してしまう)」
- 改良する
「草薙剣の獲得を優先しよう!」
「30ターン経過したらゴールに向かおう!」など

(2) やり方2

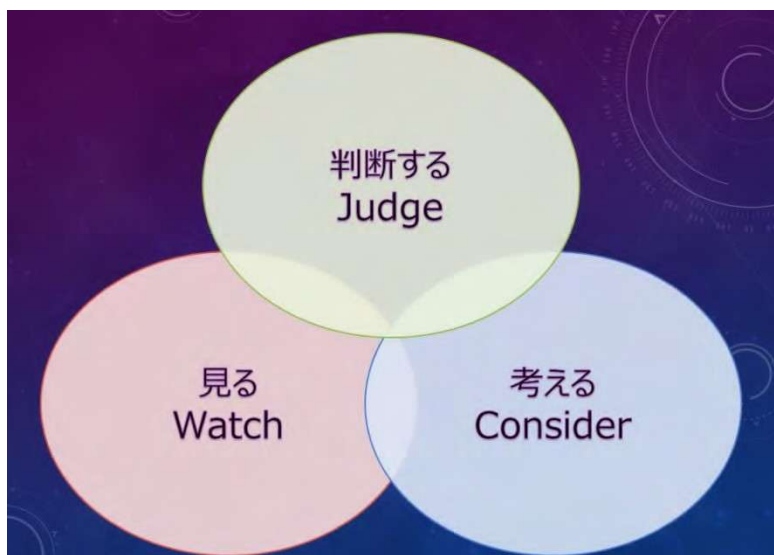
- ▶ サンプルプログラムを流用する
- ▶ サンプルプログラムでゲームを試す
- ▶ 改良したい点を考える
- ▶ 実装方法を考える
- ▶ ゲームで試す

※ サンプルプログラム3は少し難しいプログラムですが、内容を理解し、自分のプログラムに活用することにチャレンジしてみてください。

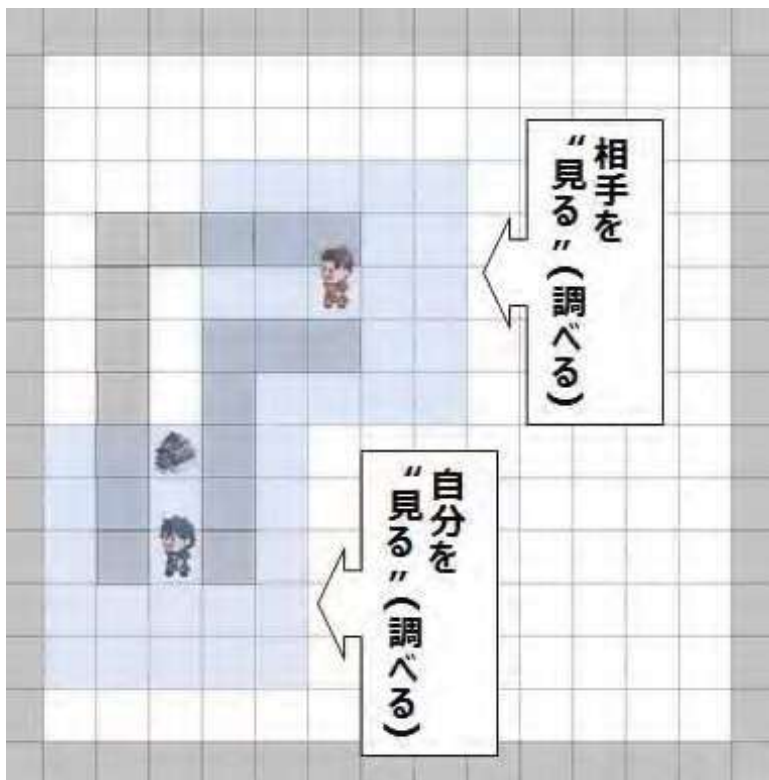
(3) 優勝者からのアドバイス (第1回大会優勝者「高田.exe」さん)

※ 第1回大会からはルールが変わっていますが、全体の考え方を参考にしてください。

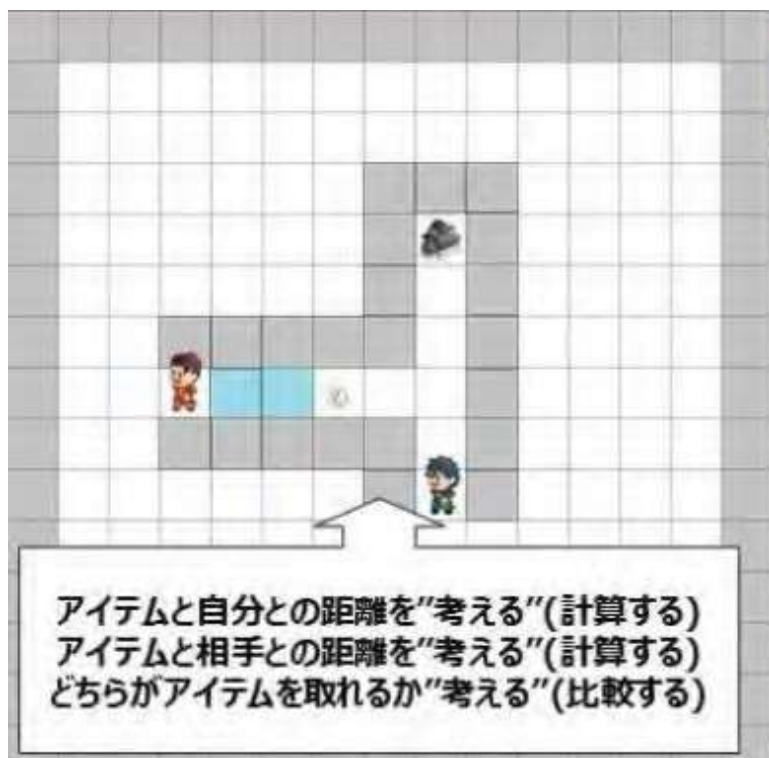
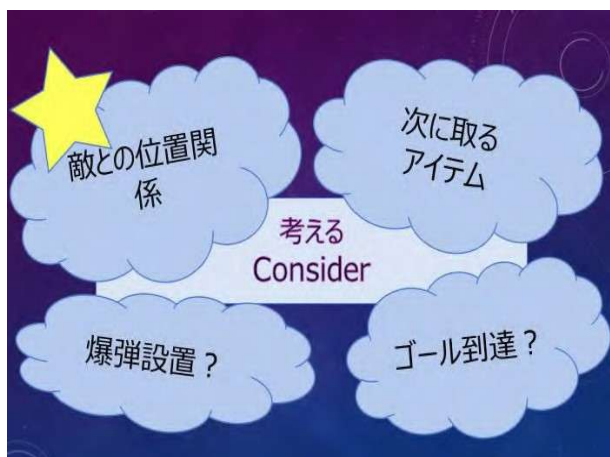
- 「見る」「考える」「判断する」が重要



•「見る」=「調べる」、自分だけでなく相手も見る



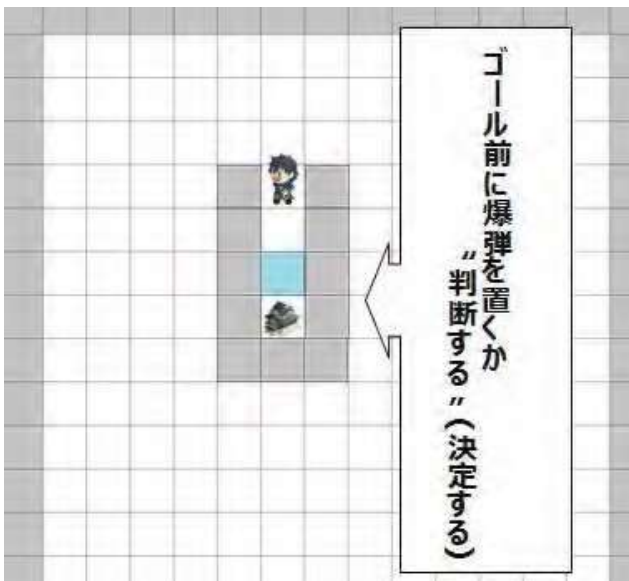
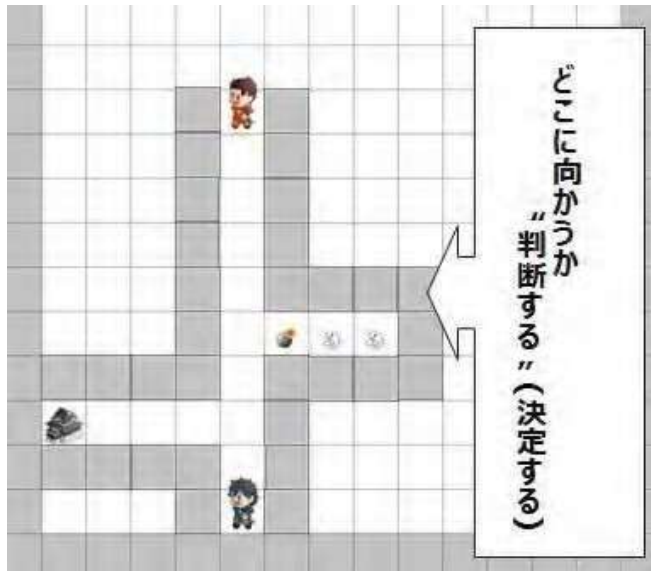
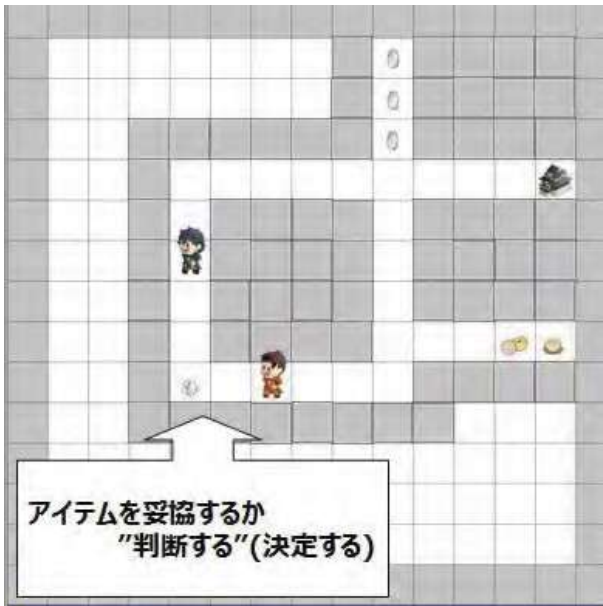
•「考える」=「計算する・比較する」



- 「自分と加点アイテムとの距離」
- 「相手と加点アイテムとの距離」
- ⇒相手の方が近い場合は、その加点アイテムは相手
がとるだろう!
- ⇒だったらゴールに向かった方が有利?
- ⇒ついでにゴール前にトラップアイテムを置いて相手
の減点を狙おう!



- 「判断する」=「見る」、「考える」の結果に応じて
 - ⇒アイテムを妥協
 - ⇒目的地を判断
 - ⇒爆弾をどこに置くか判断



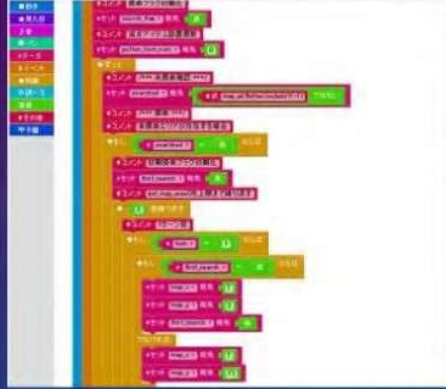
ここからはじめよう

```
【スモウルビー戦略】
・相手のルートに導線を書く。
  (ゴール直前に導線を置くのはやめたほうが良い気がする)
・シロイルカは対取るマン(減点取ってでも)
・相手の距離を見ながら、余裕があれば加点をとりつつシロイルカに向かう)
・自分の取れそうにないアイテムは単々に取る)
・ルートのみおとし
・距離とシロイルカがなくなった時点でゴールに直行するのは悪い)
(ざりざりまで移動、歩きながら計算しやがれ) (カゲクンの取捨)
・ゴールするよりもアイテムをとることで結果的にこっちが高くなるのならば)
・無駄な行き来を避けたい(高田vsカゲクンでの高田)
・サンプルマップでのバグ
  ・x高田200→カゲクン312 一次問題)
  ・x高田300→x審査
  ・カゲクン200→x審査 →ラウンド1が34点の確率)
・ルート上の加点/減点計算(演)
・xasp2/roundでカゲクンと戦うときに落ちるバグ(演)
・xasp2/round2では一歩と戦うときに最後まで結らないバグ(演)
・ゴール必要計算のバグ直し(演)
  ・環 上53, 下51
  ・デバッグ方法
    ・変換 上52
    ・相手まカゲクン以外にしてみる
    ・asp2のround1で、お茶を取った後にゴールに直行しなければ失敗)
・その他メール来てたのでカゲクンにも確認してもらうこと!)
・またたまり考慮の距離計算)
(EOF)
```

テキストエディタ(メモ帳等)に自分のやりたいことを書き出してみる

ここからはじめよう

スモウルビーに慣れる



ここからはじめよう

できればRubyにチャレンジ

```
end
# 各位置情報取得
# マップ全域の加点アイテムの場所を取得
treasu = locate_objects(sq.size: 15, cent: [7, 7], objects: ["a", "b", "c", "d"])
# マップ全域の減点アイテムの場所を取得
treasures = locate_objects(sq.size: 15, cent: [7, 7], objects: ["a", "b", "c", "d"])
# 各1個に並び替え
treasures.sort_by!{|treasure| calc_route(dst: treasure).size }
# マップ全域のシロイルカの場所を取得
beluga = locate_objects(sq.size: 15, cent: [7, 7], objects: ["d"])
# 各1個に並び替え
beluga.sort_by!{|beluga| calc_route(dst: beluga).size }

# 目的地点移動
# シロイルカルートを引く
if beluga.route == true
  # 目的地点 シロイルカの
  dst = beluga.first
  # シロイルカが存在しない場合
  if beluga.first == nil
    beluga.route = false
  end
  # 相手がいゴールしていない場合
  if (other_player.tmp_x == goal_x && other_player.tmp_y == goal_y)
    # 全量記録開始
    dst = compromise(dst, other_player.tmp_x, other_player.tmp_y, beluga)
    # 次のシロイルカがない/取れない場合 シロイルカルート終了
    beluga.route = false if dst == nil
  end
  # 並び道
  if calc_route(dst: treasures.first).length <= 3 # 2つ分以内
    if calc_route(dst: treasures.first).length % 2 == 1 # 奇数
      dst = treasures.first
    end
  end
end
```



あなただけのAIプログラムを

8. その他の説明

(1) スモウルビー甲子園の構成

- ・スモウルビー甲子園は、Cドライブ上の「koshien20xx」フォルダに保存されています。

名前	更新日時	種類	サイズ
game_server	20xx/05/21 18:00	ファイル フォルダ	
game_viewer	20xx/07/09 9:23	ファイル フォルダ	
log	
map_editor	
Ruby218_32	20xx/05/21 18:00	ファイル フォルダ	
sample_maps	
shared	20xx/05/21 18:00	ファイル フォルダ	
smalruby	
src	
LICENSE	20xx/01/16 13:55	ファイル	2 KB
main	20xx/01/16 13:55	VBScript Script ファイル	1 KB
main_debug	1 KB
map_edit	1 KB

(2) マップエディタの使い方

- ・マップエディタを使って、オリジナルのマップを作ることができます。

名前	更新日時	種類	サイズ
game_server	20xx/07/04 17:47	ファイル フォルダ	
game_viewer	20xx/07/08 20:43	ファイル フォルダ	
log	20xx/07/04 17:47	ファイル フォルダ	
map_editor	
Ruby218_32	20xx/07/04 17:46	ファイル フォルダ	
sample_maps	20xx/07/04 17:46	ファイル フォルダ	
shared	20xx/07/04 17:46	ファイル フォルダ	
smalruby	20xx/07/04 17:46	ファイル フォルダ	
src	20xx/07/04 17:46	ファイル フォルダ	
LICENSE	20xx/07/04 11:00	ファイル	2 KB
main.vbs	20xx/07/04 11:00	VBScript Script ファ...	1 KB
main debug.bat	20xx/07/04 11:00	Windows バッチ ファ...	1 KB
map_edit.bat	KB

ダブルクリックでマップエディタが起動

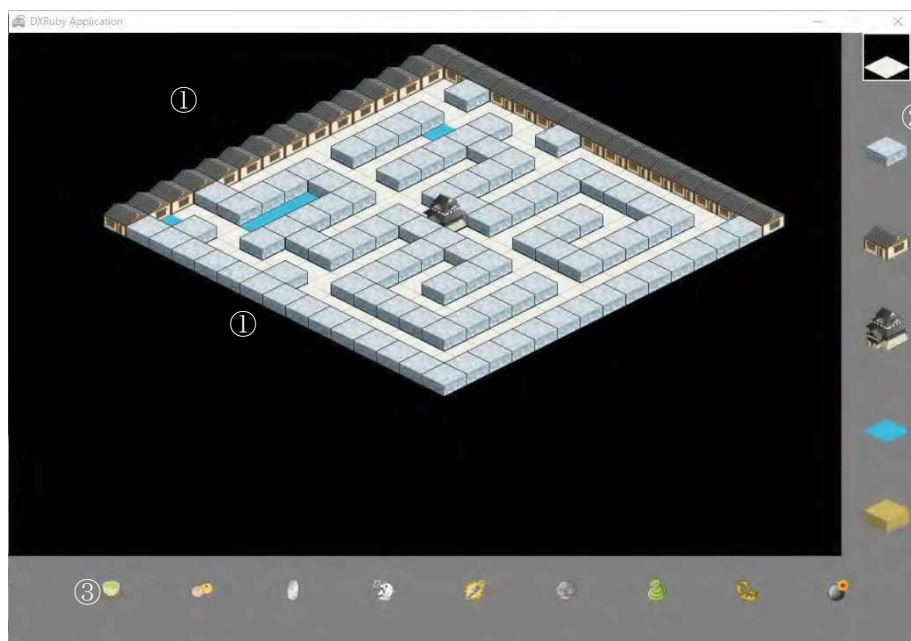
・マップエディタで使用するファイルは以下に格納されています。

このファイルを game_server 上のファイルと置き換えることで作成したマップでゲームを試せます。



※編集したいマップファイルをここに貼り付けます。

・マップエディタの使用方法は次のとおりです。



- ① マップ描画エリア
- ② マップチップ選択エリア
- ③ アイテム選択エリア

[操作方法]

- ・マップチップ選択エリアから配置したいマップチップを選択(右クリック)し、マップ描画上の配置したい場所に右クリックすると、新しいマップチップが配置されます。
- ・同様にアイテム選択エリアから配置したいアイテムを選択(右クリック)し、マップ描画上の配置したい場所に右クリックすると、新しいアイテムが配置されます。

[編集したマップの保存方法等]

- ・Escキーを押すことにより、データが保存され、マップエディタが終了します。
- ・ファイルは koshien20xx > map_editor > data に保存されています。
- ・間違って編集しないよう保存したファイルはどこか別の場所に移して置きましょう。(koshien20xx > map_editor > data に新しくフォルダを作って保存しておくとも便利です。)

(3) ログの確認方法

- ログファイルを確認することができます。



- また、pメソッドを組み込んでおくことにより、変数の中身を確認することもできます。
(ログファイルに変数の中身も記録されます。)



- main_debug からゲームを起動することで、コマンドプロンプトを見ながらゲームを試すこともできます。



(4) エラーの調べ方

(4-1) ログファイルの見方

プレイヤー1の場合は player1.log、プレイヤー2の場合は player2.log を見てください。その中に「エラーが発生しました」という行があります。その次の行からエラー内容が記録されています。

プログラムのどの部分でエラーが発生したか調べるには、ログの中の「自分が作成した AI プログラムのファイル名」を探し、その右隣の数字を見てください。それが Ruby 画面における行番号を表しています。以下の例では「sample.rb」という AI プログラムの Ruby 画面の 13行目で発生していることが記録されています。Ruby 画面の 13 行目を調べて、ブロック画面のどの位置か確認してください。

(例)

```
E, [20xx-11-15T19:42:45.036778 #5204] ERROR -- : エラーが発生しました (NameError)
undefined local variable or method 'route_target' for
#<Smalruby::Console:0x41973c0> E, [20xx-11-15T19:42:45.037762 #5204] ERROR -- :
C:/koshien20xx/smalruby/programs/ sample.rb:13:in 'block (2 levels) in <main>'
```

(4-2) タイムアウトする(その1)

【症状】タイムアウトする

【ログファイル】

「(NameError) undefined local variable or method ○○」と記録されている

```
E, [20xx-11-15T19:42:45.036778 #5204] ERROR -- : エラーが発生しました (NameError)
undefined local variable or method 'route_target' for
#<Smalruby::Console:0x41973c0> E, [20xx-11-15T19:42:45.037762 #5204] ERROR -- :
C:/koshien20xx/smalruby/programs/ sample.rb:13:in 'block (2 levels) in <main>'
```

【原因】変数名の入力ミスの可能性があります。

【対応策】変数名の入力ミスがあれば修正してください。

(4-3) タイムアウトする(その2)

【症状】タイムアウトする

【ログファイル】

「(ArgumentError) wrong number of arguments (0 for 2)」 「○○ in 'move_to'」と記録されている

```
E, [20xx-11-15T20:41:44.862595 #7988] ERROR -- : エラーが発生しました
(ArgumentError) wrong number of arguments (0 for 2)
E, [20xx-11-15T20:41:44.862595 #7988] ERROR -- :
C:/koshien20xx/src/lib/ai_lib.rb:197:in 'move_to' E, [20xx-11-15T20:41:44.862595
#7988] ERROR -- : C:/koshien20xx/src/lib/ai_lib.rb:37:in 'move_to' E, [20xx-11-
15T20:41:44.862595 #7988] ERROR -- : C:/koshien20xx/smalruby/programs/
sample.rb:74:in 'block (2 levels) in <main>'
```

【原因】「移動する」の座標が正しくない可能性があります。

【対応策】加点アイテムを取ったあとに発生することが多いです。「マップ情報を取得」がきちんとできているか確認してください。

(4-4)キャラクターが動かない

【症状】ターンは進行するが、キャラクターが動かない

【ログファイル】

「移動に失敗しました message=移動できない座標です」と記録されている

```
D, [20xx-11-15T20:36:33.630178 #11400] DEBUG -- : moveTo を実行します 引数=[10, 5]
E, [20xx-11-15T20:36:33.643144 #11400] ERROR -- : 移動に失敗しました message=移動できない座標です
```

【原因】移動することができないマス(壁など)に移動しようとしています。

【対応策】「マップ情報を取得」を実行している範囲とそうでない範囲があると、最短経路の検索の結果に壁を通る経路が含まれてしまって発生することがあります。「マップ情報を取得」がきちんとできているか確認してください。

(4-5)AIが停止しました

【症状】「AIが停止しました」と表示され START ボタンが表示されない

【ログファイル】

「AIライブラリを読み込みました」の直後に「エラーが発生しました」が記録されている

```
I, [20xx-11-15T19:34:49.821730 #19652] INFO -- : --- AIライブラリを読み込みました ---
E, [20xx-11-15T19:34:51.422715 #19652] ERROR -- : エラーが発生しました
(NoMethodError) undefined method 'set_name' for
#<Smalruby::Console:0x3fe7720>
```

【原因】テンプレートに含まれるブロックのどれかが無くなっている可能性があります。

【対応策】無くなっているブロックを追加してください。

9. 参考

(例題1)

1. 最初にマップ全体の「マップ情報の取得」を実施
2. マップ探索が終わったら、全ての加点アイテムの座標を調べる
3. 一番近い加点アイテムから順番に取りに行く。(加点アイテムを取ったら次の加点アイテムに向かう)
4. 30ターンになったらゴールに向かう

※注意:加点アイテムを取った後に移動先を見失わないように!

(例題2)

1. 最初にマップ全体の「マップ情報の取得」を実施
2. マップ探索が終わったら、
 - 草薙剣の座標を調べる
 - シロイルカの座標を調べる
 - 爆弾とトラバサミの座標を調べる
3. 爆弾とトラバサミを避けながら草薙剣を取りに行く。
条件を満たすルートがない場合は避けずに取りに行く。
4. 草薙剣がなくなったら、爆弾とトラバサミを避けながらシロイルカを取りに行く。
条件を満たすルートがない場合は避けずに取りに行く。
5. シロイルカもなくなったら、ゴールに向かう

※注意:加点アイテムを取った後に移動先を見失わないように!