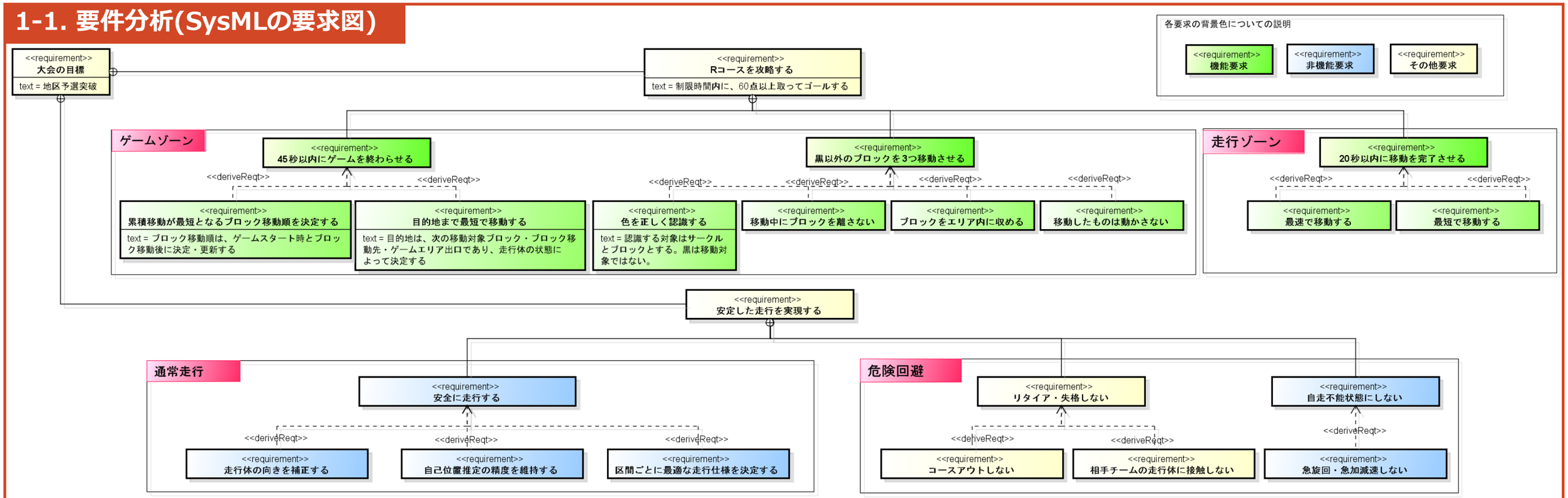
Rコースで60点以上取ることを目標とし、要求事項を整理した

しんよこ
(°Д°)

1-1. 要件分析(SysMLの要求図)



1-2. 要求に対応する要素

要求	要求に対応する要素と参照先										
	移動対象ブロック決定(ゲームの攻略手順)	ブロックの色の管理	自己位置推定機能	経路の選択基準	座標指定移動	最適速度設定	色認識	アーム制御	向き補正走行	自己位置補正	ゲームエリアと走行体の抽象化
	2-6	2-5	3-1	2-4	3-3	3-6	3-5	3-5	3-4	3-2	2-1
累積移動が最短となるブロック移動順を決定する	●	●									
目的地まで最短で移動する			●	●	●	●					●
色を正しく認識する							●	●			
移動中にブロックを離さない					●	●					
ブロックをエリア内に収める					●	●					
移動したものは動かさない		●									
最速で移動する						●					
最短で移動する				●	●	●					●
走行体の向きを補正する									●		
自己位置推定の精度を維持する									●	●	●
区間ごとに最適な走行仕様を決定する											●
コースアウトしない											
相手チームの走行体に接触しない											
急旋回・急加減速しない											

※要求に対応する要素の詳細は後述する
(記述先は各番号を参照)

地区予選突破に、Rコースで必要となる点数を60点以上と想定し、チームの目標とした。

目標達成に必要な要求をSysMLの要求図を用いて段階的に詳細化した。(1-1. 要件分析)

また、要件分析で抽出した派生要求を実現するため、各要求に対して、必要となる要素技術を洗い出した。(1-2. 要求に対応する要素技術)

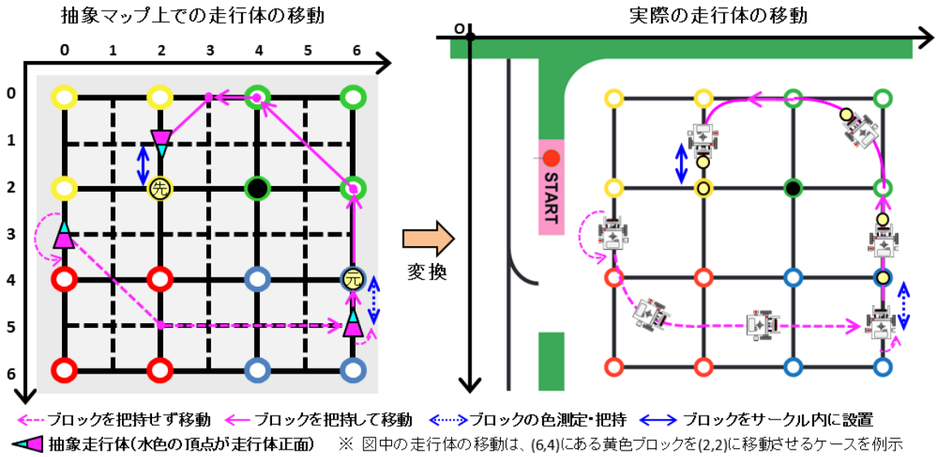
1. 要求 | **2. 分析** | 3. 制御 | 4. 設計(1) | 4. 設計(2)

抽象化したゲームエリア上での走行体の振る舞いと実際の操作を分離

しんよこ
(°Д°)

2-1. ゲームエリアと走行体の抽象化

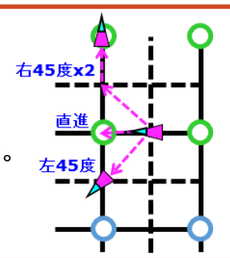
４ｘ４マスの格子であるゲームエリアを、格子点間の中点で分割し、下図のような座標系で表現した**抽象マップ**を設ける。抽象マップ上では走行体は位置、向き、把持ブロックの有無を持つ**抽象走行体**として表現する。走行体の振る舞いを抽象マップ上で検討し、実際の動作へ変換する。



2-2. 抽象走行体の移動規則

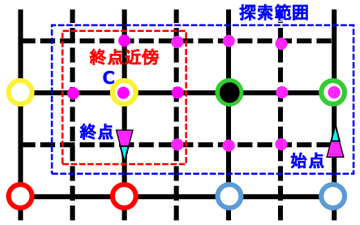
抽象走行体の移動は以下の規則に従う。

1. 停止中は、常にサークルとサークルの中間(以降、**中間点**)に存在する。
2. 停止中の向きは、必ずXまたはY軸に平行である。
3. ブロックを**把持しておらず、かつ停止中のみ**左右**90**または**180度**向きを変えることができる。
4. **移動中かつ格子点上でのみ**その向きを左右**45度**変えることができる(左図に例示)。
5. ブロックを把持する際を除き、ブロックが設置されている点を通過しない。



2-3. 抽象マップ上での経路探索

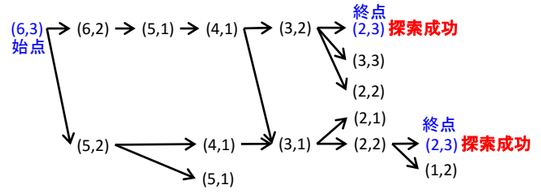
1. 探索範囲および経由点の設定 始点から終点への経路の探索範囲は、『始点』と『終点での走行体の向きと反対の隣接点(右図の点C)を中心とする**最大2x2の矩形**(以降、**終点近傍**：右図の赤破線)』を含む矩形(右図の青破線)とする。走行体は探索範囲中の格子点を経由するが(以降、**経由点**：右図のピンク色の点)、始点と終点を除く矩形の頂点は除外する。



2. 経路の探索 始点を出発し**2-2**の規則に従って移動しながら、通過した経由点を記録する**探索子**を設ける。探索子は右下図のように、次の移動先が複数になるたびに複製・分岐し、それぞれの方向に移動する。探索子の停止条件は以下の通り。

(探索成功)
・終点に到達し、終点での走行体の向きとの差が±45度以内である。

(探索失敗)
・終点に到達し、終点での走行体の向きとの違いが±90度以上である。
・点Cを除く**終点近傍**の点を2回通過する。
・同じ経由点を2回通過する。
・移動可能な経由点が存在しない。



ブロックを把持した場合の探索子の分岐パターンの例

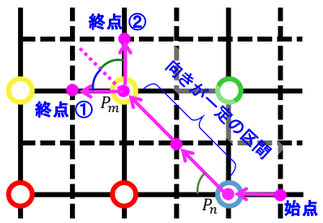
2-4. 経路の選択基準

2-3で探索した経路の、抽象マップ上での移動距離と抽象走行体の向きの変更から**移動コスト**を算出し、移動コストが最も低い経路を選択する。

1. 移動距離由来のコスト 経路上の隣接する経由点を結ぶ直線の長さの総和。

2. 向きの変更に由来するコスト

抽象走行体の移動	現実の走行体への影響
向きの急激な変化	移動速度低下, 自己位置推定の精度低下
向きの変化に要した移動距離	向きの急激な変化による現実の走行体への影響を低減



経路上で向きの変化する点：P_n
点P_nでの向きの変化：A_n(度)
点P_mの次に向きが変化する点：P_m
点P_mでの向きの変化：A_m(度) ・・・(向きの変化は右周りを正とする)

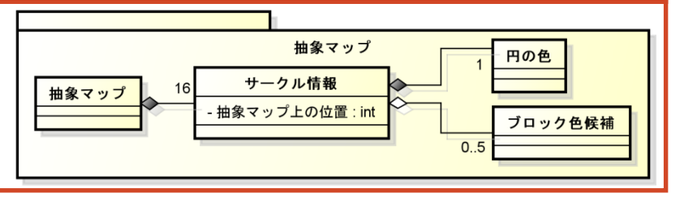
$$\text{向き変更に由来するコスト} = \sum \frac{2(1+|A_n - A_m|/90)}{|P_n P_m|}$$

移動コスト = 移動距離由来のコスト + 向きの変更に由来するコスト

(例) 上図での①の移動コスト
移動距離由来のコスト $= 1+\sqrt{2}\times 2+1 \cong 4.82$
向きの変更に由来するコスト $= 2\times(1+\frac{|45+45|}{90})/(2\sqrt{2}) = 1.41$
移動コスト $= 4.82+1.41 = 6.23$

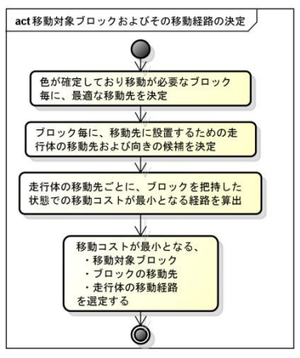
2-5. ブロックの色の管理

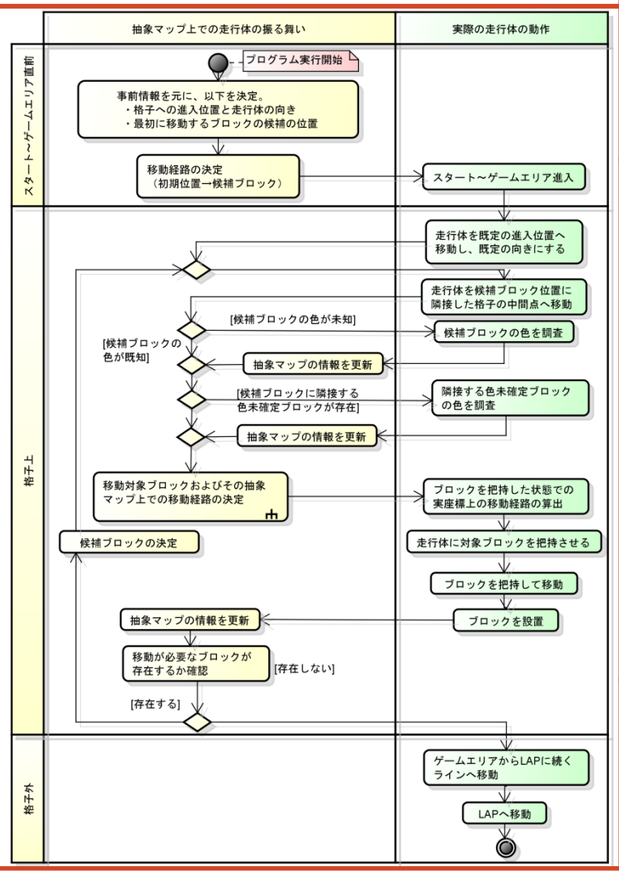
抽象マップ上では、各サークルは自身の色情報と設置されているブロックの色の候補を0~5個持つ(色候補が0個のサークルはブロックが設置されていない)。ブロックの色が確定、またはブロックが移動される度に抽象マップの情報は更新される。



2-6. ゲームの攻略手順

2-1～2-5で定義したモデルを用いてゲームを攻略するための手順を示す。抽象マップから実際の座標系への変換および走行体の各種動作については、**3. 制御** で述べる。

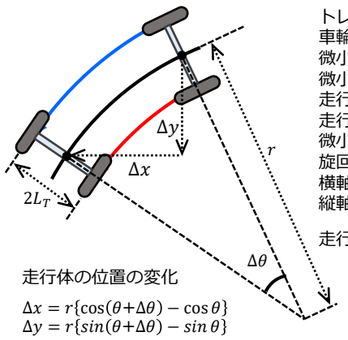




Rコースをクリアするための要素技術を検討し、制御戦略を立てた

しんよこ
(°Д°)

3-1. 自己位置推定機能 要素技術



トレッド長(mm)： $2L_T$
車輪半径(mm): R_W
微小時間での右サーボモーターの回転角の変化量$(deg.)$：$\Delta\varphi_R$
微小時間での左サーボモーターの回転角の変化量$(deg.)$：$\Delta\varphi_L$
走行体の座標(mm)：x, y
走行体の方向$(rad.)$：θ
微小時間での機体の進行方向の変化量$(rad.)$：$\Delta\theta$
旋回半径：r
横軸方向の位置変化(mm)：Δx
縦軸方向の位置変化(mm)：Δy

走行体の位置の変化

$\Delta x = r\{\cos(\theta+\Delta\theta) - \cos\theta\}$
$\Delta y = r\{sin(\theta+\Delta\theta) - sin\,\theta\}$

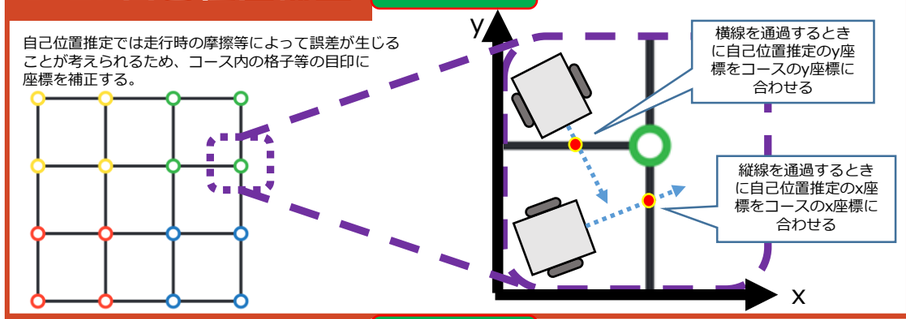
走行体の旋回半径と方向変化

$$r = \frac{L_T(\Delta\varphi_R + \Delta\varphi_L)}{\Delta\varphi_R - \Delta\varphi_L} \qquad \Delta\theta = \frac{\pi R_w(\Delta\varphi_R - \Delta\varphi_L)}{360L_T}$$

(直進時は無限大となる)

3-2. 自己位置補正 要素技術

自己位置推定では走行時の摩擦等によって誤差が生じることが考えられるため、コース内の格子等の目印に座標を補正する。



3-3. 座標指定移動 要素技術

走行体の現在位置$\overrightarrow{P_0} = \begin{pmatrix} P_{0x} \\ P_{0y} \end{pmatrix}$での速度および向き $\overrightarrow{v_0} = \begin{pmatrix} v_{0x} \\ v_{0y} \end{pmatrix}$

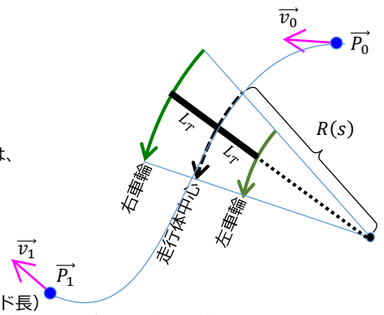
目標点$\overrightarrow{P_1} = \begin{pmatrix} P_{1x} \\ P_{2y} \end{pmatrix}$での速度および向き $\overrightarrow{v_1} = \begin{pmatrix} v_{1x} \\ v_{1y} \end{pmatrix}$

3次スプライン曲線 $\overrightarrow{P(s)} = \overrightarrow{a_0} + \overrightarrow{a_1}s + \overrightarrow{a_2}s^2 + \overrightarrow{a_3}s^3 \ (0 \le s \le 1)$ の係数は、

$\overrightarrow{a_0} = \overrightarrow{P_0}$
$\overrightarrow{a_1} = \overrightarrow{P_1}$
$\overrightarrow{a_2} = 3\overrightarrow{P_1} - 3\overrightarrow{P_0} - 2\overrightarrow{v_0} - \overrightarrow{v_1}$
$\overrightarrow{a_3} = -2\overrightarrow{P_1} + 2\overrightarrow{P_0} + \overrightarrow{v_0} + \overrightarrow{v_1}$

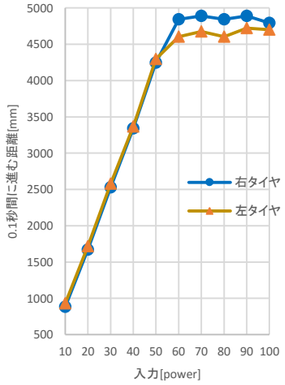
曲率半径 $R(s) = \left|\overrightarrow{P(s)}'\right|^3 / \left(\overrightarrow{P(s)}' \times \overrightarrow{P(s)}''\right)$

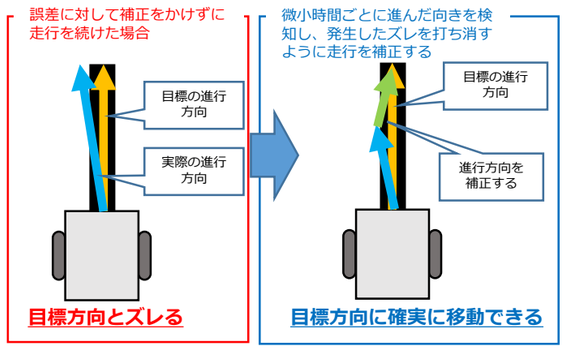
左右モーターの回転速度の比 $= (R(s) - L_T) : (R(s) + L_T)$ $\quad (L_T$：トレッド長)
現在の右車輪の回転速度から、左車輪の回転速度の目標値を決定し、負のフィードバック制御を行う。



3-4. 向き補正走行 要素技術

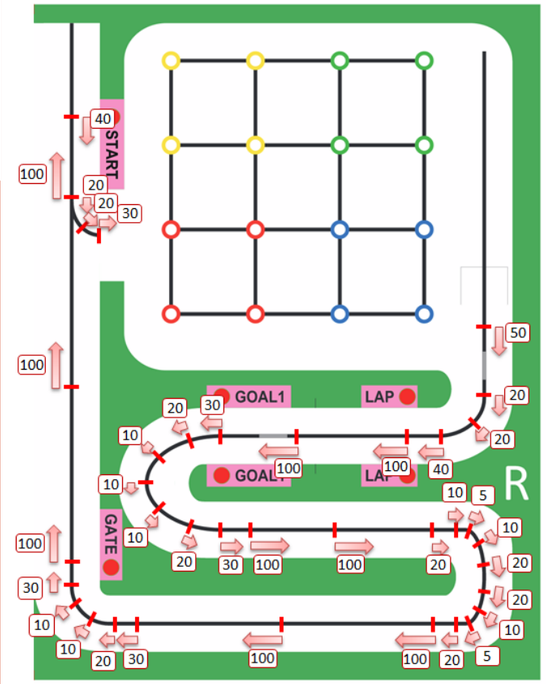
入力[power]値が高い場合、左右モーターへの入力[power]値が同じであっても、左右タイヤの進む距離には下グラフのように差が生じる。





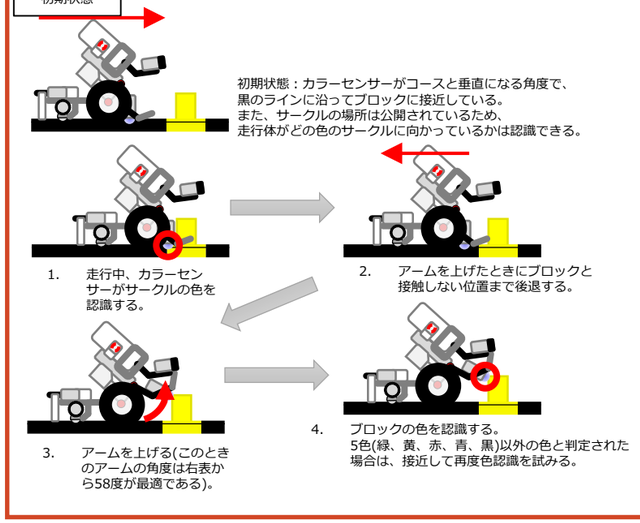
3-6. 走行 制御戦略

要素技術3-1 ～ 3-4によって、安定した座標指定移動が可能となるので、Rコース上での走行は全て座標指定移動で行う。
さらに、走行速度をコースの形状に従って設定する(下図参照)ことで、走行の安定性を向上させる。



3-5.色認識・アーム制御 要素技術

色認識、アーム制御、車輪制御を用いて、次の手順によってブロックをアームに収めることを実現する。

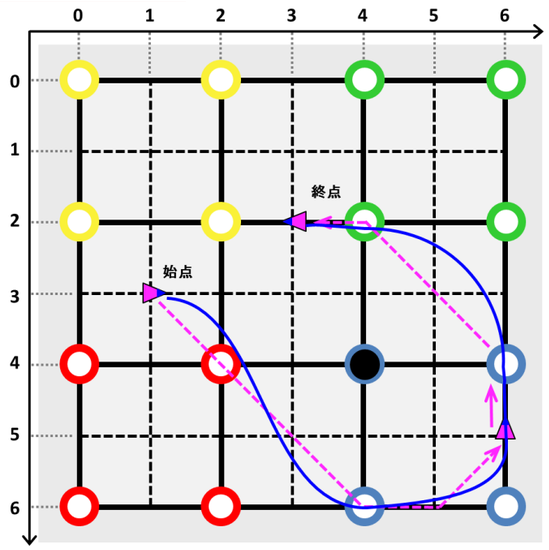


アーム角度ごとのブロックの色判定結果

アーム角度[度]	対象ブロックの色				
	緑	黄	青	赤	黒
40	BLACK	BLACK	BLACK	BLACK	BLACK
41	BLACK	BLACK	BLACK	BLACK	BLACK
42	BLACK	BLACK	BLACK	BLACK	NONE
43	BLACK	BROWN	BLACK	RED	NONE
44	BLACK	BROWN	BLACK	RED	NONE
45	BLACK	BROWN	BLACK	RED	NONE
46	BLACK	BROWN	BLACK	RED	NONE
47	BLACK	BROWN	BLACK	RED	NONE
48	BLACK	BROWN	BLUE	RED	NONE
49	BLACK	BROWN	BLUE	RED	NONE
50	BLACK	BROWN	BLUE	RED	NONE
51	BLACK	BROWN	BLUE	RED	NONE
52	GREEN	BROWN	BLUE	RED	NONE
53	GREEN	BROWN	BLUE	RED	NONE
54	GREEN	BROWN	BLUE	RED	NONE
55	GREEN	YELLOW	BLUE	RED	NONE
56	GREEN	YELLOW	BLUE	RED	NONE
57	GREEN	YELLOW	BLUE	RED	NONE
58	GREEN	YELLOW	BLUE	RED	NONE
59	GREEN	YELLOW	BLUE	RED	NONE
60	GREEN	YELLOW	BLUE	RED	NONE
61	GREEN	YELLOW	BLUE	RED	NONE
62	GREEN	YELLOW	BLUE	RED	NONE
63	BLACK	YELLOW	BLUE	RED	NONE
64	BLACK	YELLOW	BLUE	RED	NONE
65	BLACK	YELLOW	BLUE	RED	BLACK
66	BLACK	YELLOW	BLUE	RED	BLACK
67	BLACK	YELLOW	BLUE	RED	BLACK
68	BLACK	YELLOW	BLUE	RED	NONE
69	BLACK	YELLOW	BLUE	RED	NONE
70	BLACK	BROWN	BLUE	BLACK	NONE

アーム角度58度周辺で色が確実に区別できることがわかった。
(このとき、NONEはBLACKとみなす)

3-7. ゲーム 制御戦略



抽象マップ上で決定された移動経路は直線的な軌跡だが、これを現実の走行体の動きに変換する際には、要素技術3-3.を用いて、経由点を滑らかに結ぶ3次スプライン曲線に変換する。

これによって、滑らかで迅速に移動する。

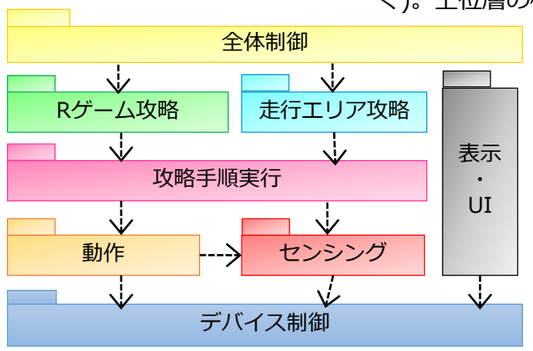
抽象マップ上の走行体の移動経路
現実での走行体の移動経路

ブロック並べのドメインモデルと走行体の操作を分離する

しんよこ
(°Д°)

4-1-1. パッケージ構成

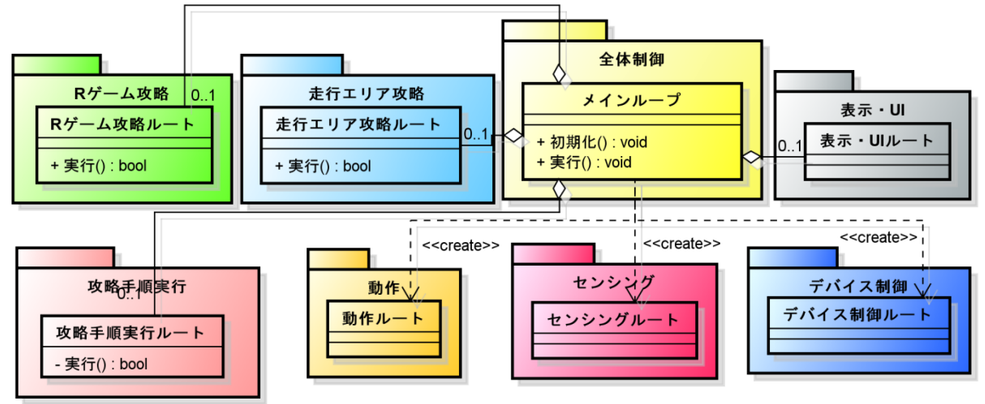
システムをその機能の抽象度に着目して階層化し、パッケージとした(表示・UIを除く)。上位層の機能はその直下の層によって実現される。



パッケージ名	概要
全体制御	各パッケージのインスタンスの管理、戦略の実行。
Rゲーム攻略	ブロック並べ攻略のモデルを管理し、戦術を作成する。
走行エリア攻略	ブロック並べ以外のコース情報から走行戦術を作成する。
攻略手順実行	Rゲーム攻略、走行エリア攻略が作成した一連の指示を実行する。
動作	ブロック色認識等のなどの複雑な動作を行う。
センシング	走行体の推定位置等の二次的な情報の生成・管理を行う。
デバイス制御	動作、センシングに、デバイス操作の低レベルIFを提供する。
表示・UI	ボタンの押下の検知、およびLCD、スピーカーの制御を行う。

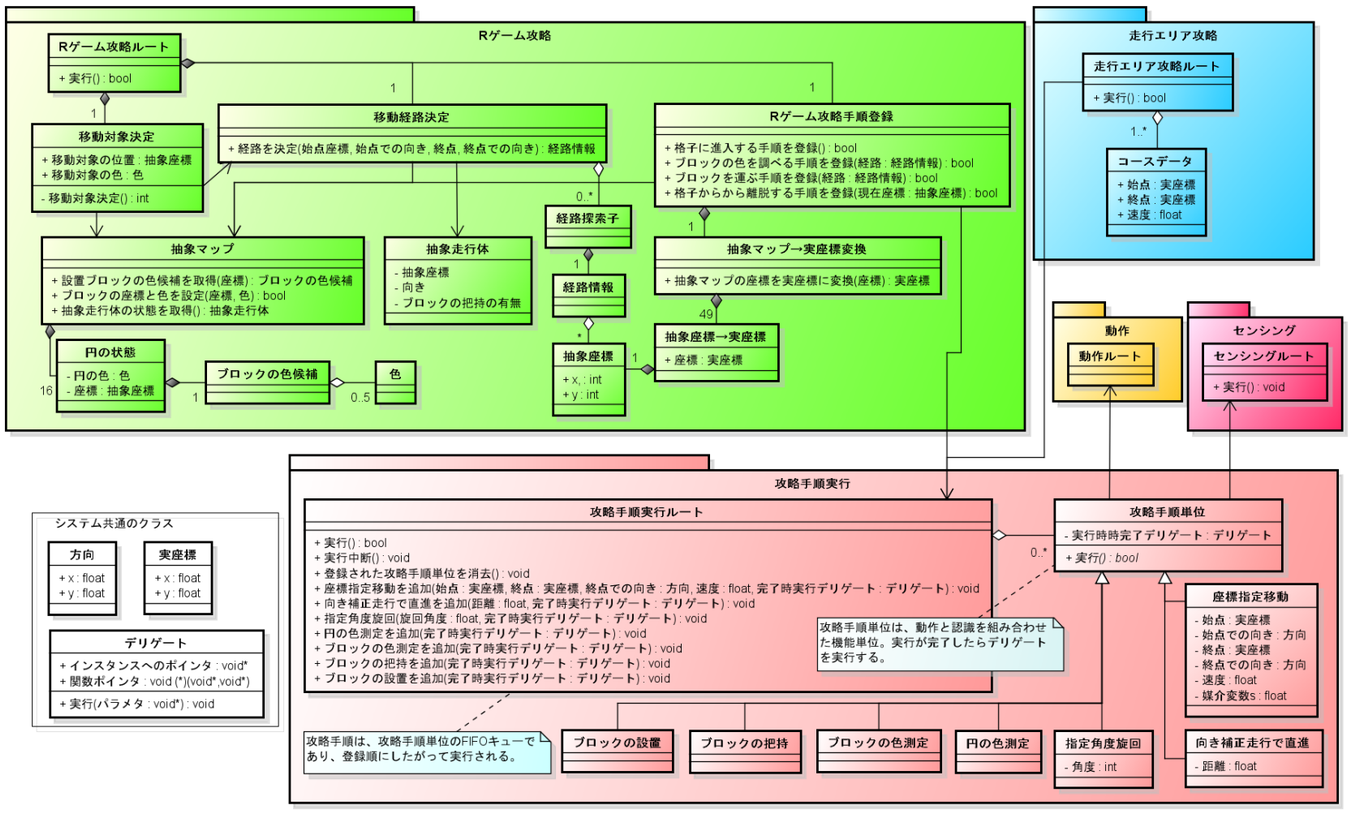
4-1-2. 全体制御

全体制御は各パッケージのルート（外部からパッケージを利用する際の窓口となるオブジェクト）を作成する。また、Rゲーム攻略、走行エリア攻略、表示・UIのルートのインスタンスを保持する。メインループ内では、Rゲーム攻略または走行エリア攻略および攻略手順実行パッケージのルートの実行関数が呼び出される。



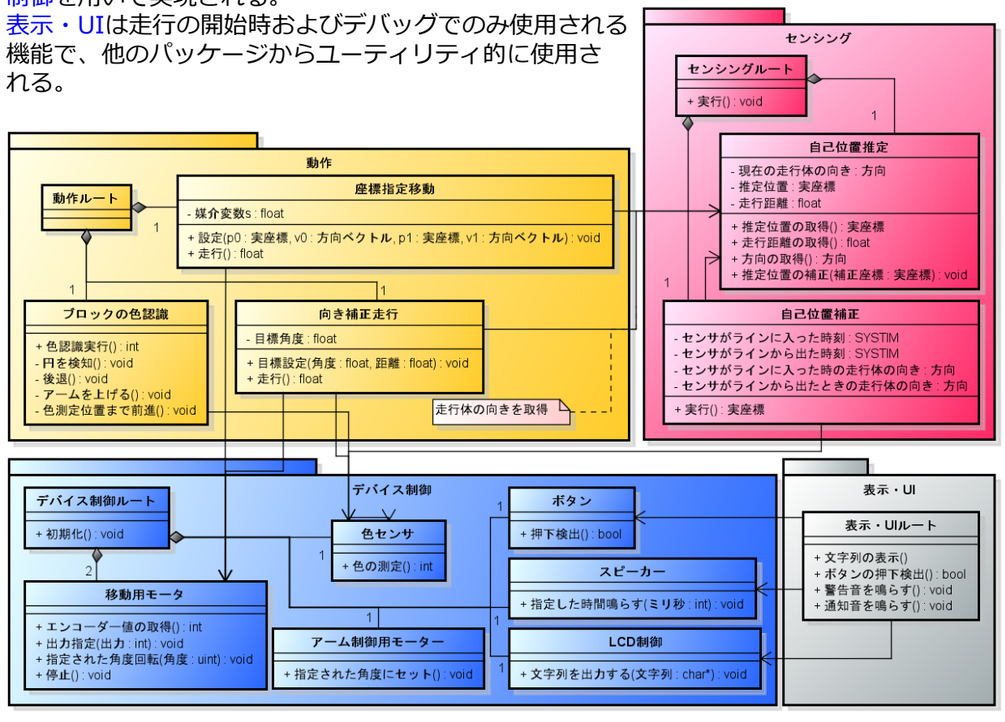
4-1-3. Rゲーム攻略・走行エリア攻略・攻略手順実行

Rゲーム攻略および走行エリア攻略は直接走行体を操作せず、攻略の手順を攻略手順実行に登録する。攻略手順実行は登録された順に攻略手順を実行する。攻略手順実行の結果(ブロックの色測定等)は、Rゲーム攻略の状態更新メンバ関数のデリゲートを介して反映される。



4-1-4. 動作・センシング・デバイス制御・UI

動作およびセンシングは、攻略手順実行が必要とする機能(**3.制御**)を提供する（例：ブロック色認識、自己位置推定等）。これらの機能はEV3 APIの簡易的なラッパーであるデバイス制御を用いて実現される。
表示・UIは走行の開始時およびデバッグでのみ使用される機能で、他のパッケージからユーティリティ的に使用される。

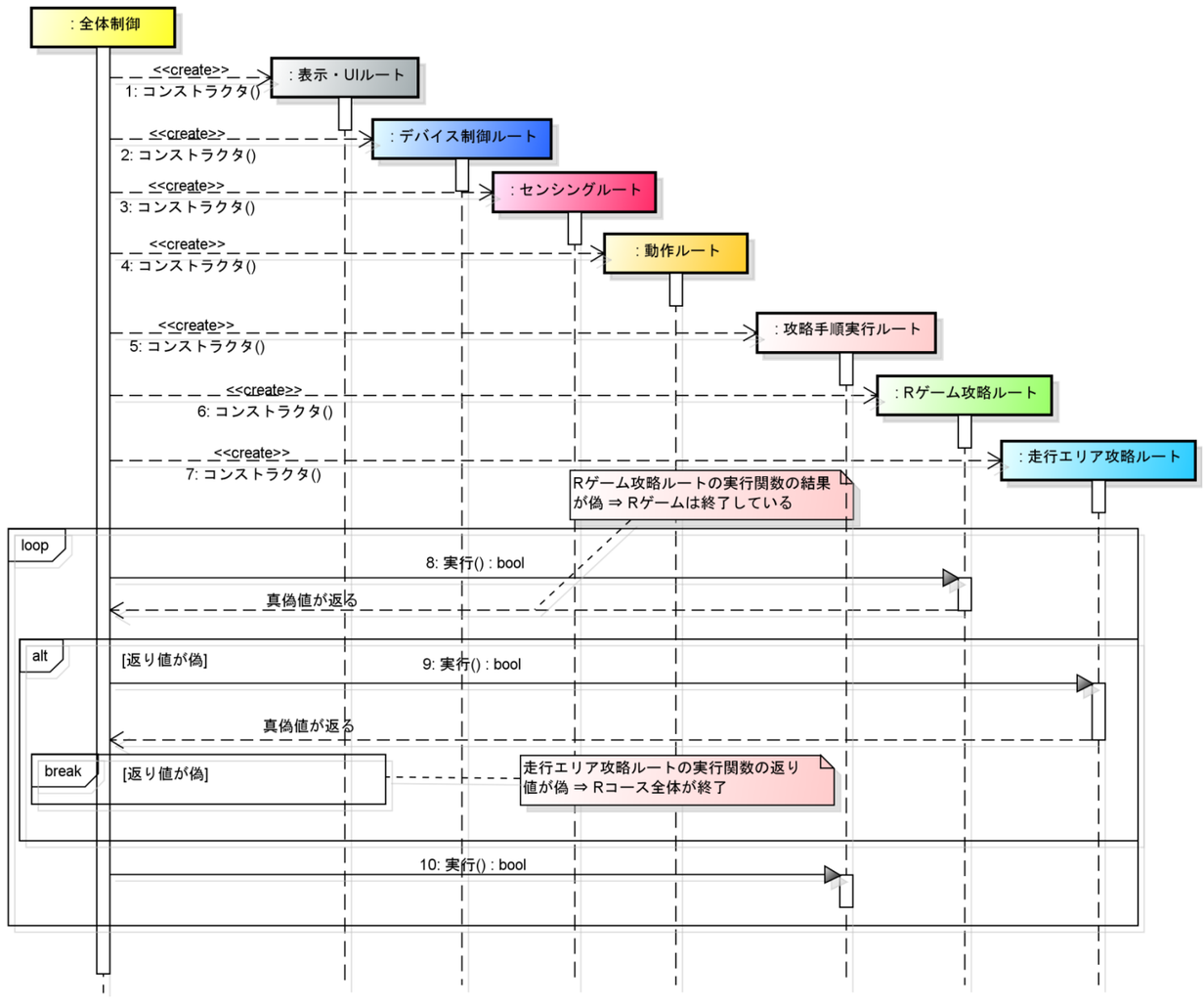


システムは各パッケージのメイン関数の呼び出しで実行される

しんよこ
(°Д°)

4-2-1. システム全体のシーケンス

プログラム開始時に、**4.1** で示した階層の下から上の順に全パッケージのルートが作成される。全パッケージのルートを作成後、全体制御のメインループで、Rゲーム攻略ルート、走行エリア攻略ルート、攻略手順実行ルートの実行関数が呼びだされる。



4-2-2. Rゲーム攻略のシーケンス

Rゲームの攻略をシーケンスを、攻略手順の作成と実行に注目して例示する（各パッケージのルートは既に作成済み）。