ETロボコン2018
デベロッパー部門 アドバンストクラス

地区：沖縄　　地域：沖縄県西原町

チームNo. 200　　チーム名：ie-ryukyu

所属：琉球大学工学部情報工学科

チーム紹介、目標、意気込み

私たちはチーム「ie-ryukyu」です。琉球大学工学部情報工学科3年次7人のメンバーで構成されています。このメンバーでは初の参加となるので、完走目指して頑張ります。
今回のETロボコンでは、開発する際にモデルの作成が重要であると考えられるので、そちらに関しても力を入れて頑張っていきたいと思います。

モデルの概要

- 要求モデルでは、コースを完走させるために必要な機能について、SysML要求図でまとめた。（走行、ゲーム(ブロック並べ、AIアンサー) など）
- 中でも特に、「ゲーム(ブロック並べ)」（以下、ブロック並べ）を詳細に分析した。
- 分析～制御モデル(１つ目) までは、ブロック並べをどのように攻略したかについて まとめたものである。
- 制御モデル（２つ目）は、 AIアンサーを攻略する上で必要とされる技術について、検討したものをまとめたものである。

モデルの構成

1. 要求分析

まず、目標を定義した。
続いて、その目標をクリアする為に必要な要件を、SysMLの要求図を使って洗い出した。
なお今回は、「審査対象をRコースのゲームに関する部分に限定する」という指定があるため、ゲーム(ブロック並べ)を中心に要求分析を行った。

2. 分析モデル

まず、ブロック並べのゲームの要素定義をまとめた。
続いて、上記の定義から得られた課題を、どのようにして解決するかを検討し、フローチャート等にまとめた。

3. 設計モデル

まず、プログラム全体の構造（抽象度３層構造）を紹介した。
次に、ブロック並べのクラス図を示し、ブロック並べの攻略をどのように設計・実装したかを提示した。

4. 制御モデル

① ブロック並べの要素技術について。
ブロック並べを攻略する際に必要となる制御技術についてまとめた。

② AI アンサーの「数字の読み取り制御」について。
デジタル数字を読み取るための「8の字走行」と、アナログ数字を読み取るための「3return走行」を説明する。また、それらを成功させるための制御技術を紹介する。

1.要求モデル

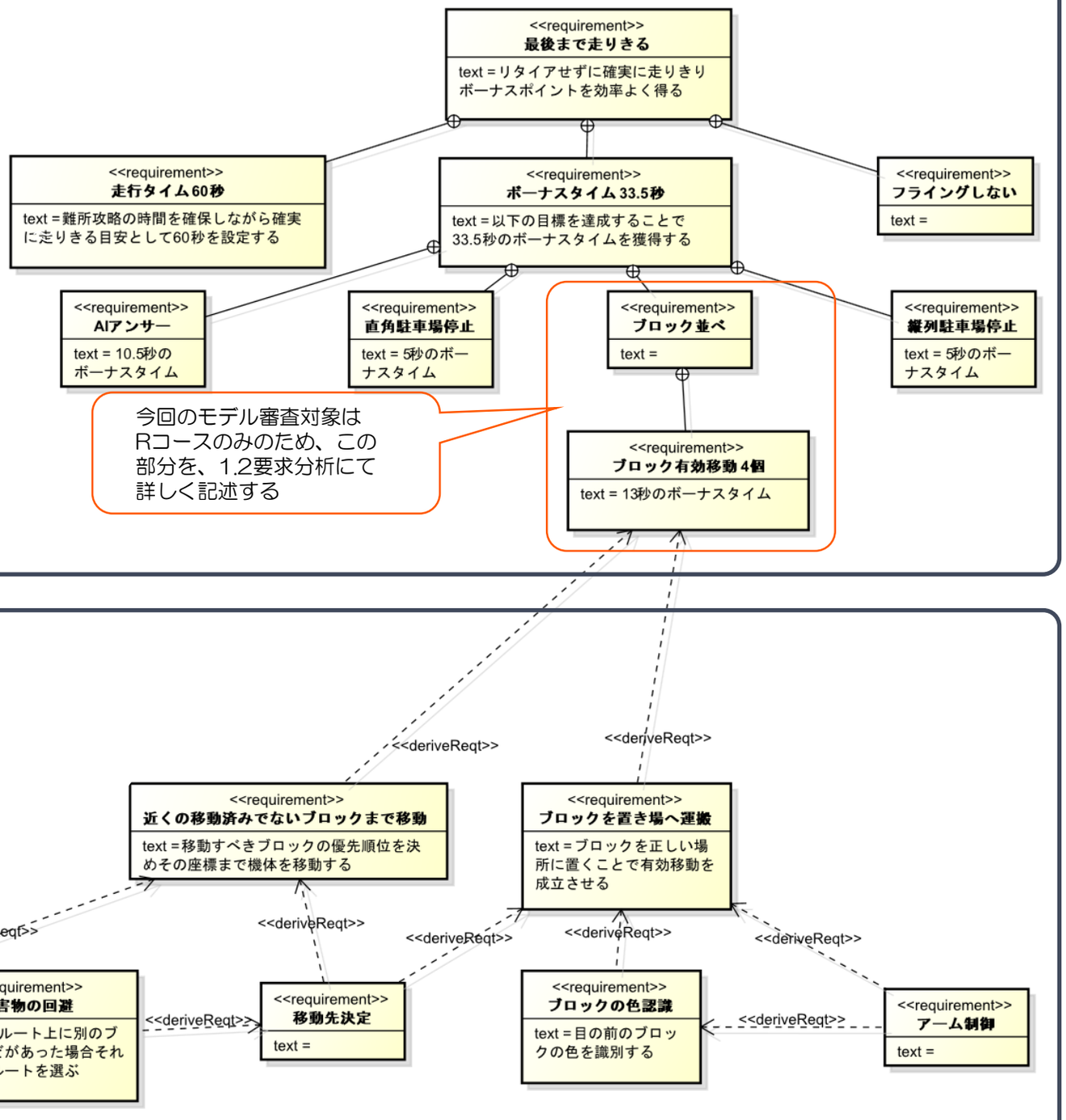
ie-ryukyu

1.1 目標

試走会の結果等を考慮し、以下のように目標を設定する。

・走行タイム45秒*2コース
・フライングしない
・AIアンサー：ビット回答3個
・AIアンサー：読み取り成功1個
・ブロック並べ：ブロック有効移動4個
・縦列駐車場停止
・直角駐車場停止

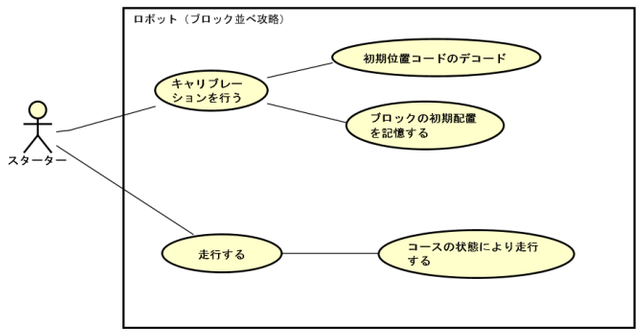
　これらを達成した場合、ボーナスタイムが33.5秒得られる。
　走行タイム45*2秒と合わせると、リザルトタイムは56.5秒となる。
　右に、上記の目標をまとめた要求図を示す。



1.2 要求分析

1.1で設定した目標を達成するために、システムが満たすべき要件を洗い出した。
特に、今回の審査対象である、「Rコースのブロック並べ」を中心に分析した。

1.3 ユースケース図



競技規約を分析し、ユースケース図にまとめた。

2.分析モデル

ie-ryukyu

2.1 課題：ブロック並べのゲーム内容の説明

・ブロック並べの構成要素

ブロック並べは、ブロック並べエリア上のブロック置き場にあるブロックを移動し、ブロックの移動先の結果によりボーナスタイムを獲得するゲームである。ブロック並べを行う際に必要な構成要素を図2.1に示す。

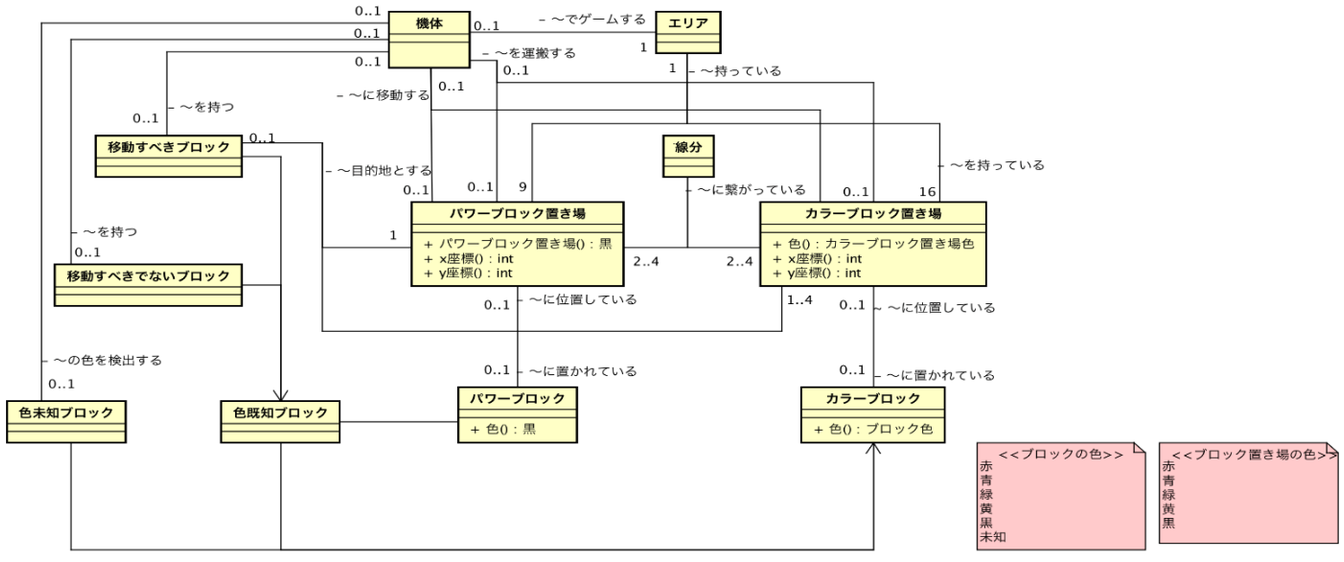


図2.1 ブロック並べの構成要素

2.2 解決：初期位置コードのデコード

初期位置コードのデコードを行なうにあたって初期位置コードを84722を例としてデコードを行なう。
ブロックのそれぞれの座標をa, b, c, dとし、パワーブロックの座標をeとする。 初期位置コードをそれぞれの座標で表すと、以下の式になる。

$$84772 = a \times 16^4 + b \times 16^3 + c \times 16^2 + d \times 16 + e.$$

上式よりa, b, c, d, eを求める。それぞれa, b, c, d, eは

$$\begin{aligned} a &= 84772 \div 16^4, \\ b &= (84772 - a \times 16^4) \div 16^3, \\ c &= \{84772 - (a \times 16^4 + b \times 16^3)\} \div 16^2, \\ d &= \{84772 - (a \times 16^4 + b \times 16^3 + c \times 16^2)\} \div 16, \\ e &= 84772 - (a \times 16^4 + b \times 16^3 + c \times 16^2 + d \times 16). \end{aligned}$$

と表すことができ、以下のようになる。

$$\begin{cases} a = 1, \\ b = 4, \\ c = 10, \\ d = 15, \\ e = 2. \end{cases}$$

よってブロックの位置は図2.2のようになっていることがわかり、パワーブロックの位置が正しいことがわかる。

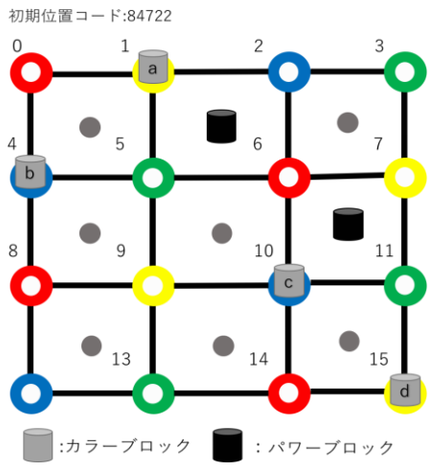


図2.2 初期座標コードにより配置されたブロック

2.3 解決：ブロック並べの攻略

・ブロック並べエリアの座標化

ブロック並べを行うあたり図2.3 のように座標を定めた。

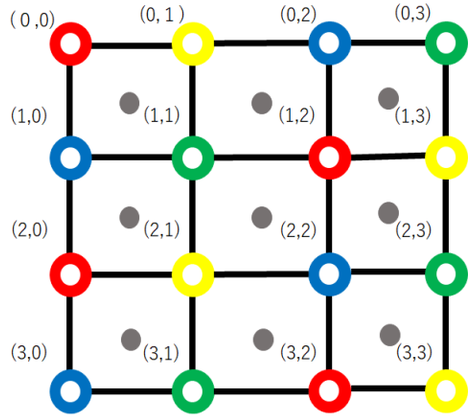


図2.3 ブロック並べの座標化

・運ぶべきブロックの優先順位決定

前述のブロック並べエリアの座標より、現在の機体のいる位置より最も近いブロックを求める。現在の機体の座標を(a_1, b_1)とし、移動したい場所の座標を(a_1, b_1)とした時に移動距離は以下のようになる。

横の移動距離： $|a_1 - a_0|$,
縦の移動距離： $|b_1 - b_0|$.

上記の式より、現在の機体の位置から最も近いブロックを探索し、次に運ぶブロックを決定する。

・パワーブロックの移動

ボーナスタイムをより獲得するためにパワーブロックを適切な座標に移動させる。適切な座標を求める仕組みを以下の図2.4に示す。

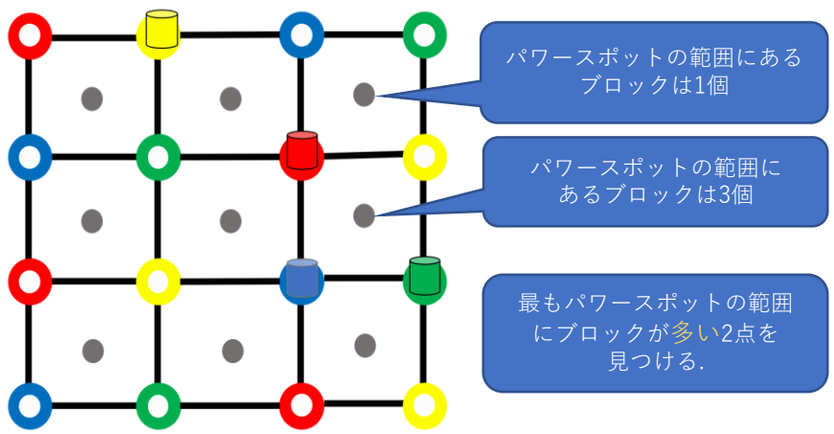


図2.4 パワーブロックの移動

2.分析モデル

ie-ryukyu

2.3 解決：ブロック並べの攻略

・ブロック並べでの動作定義

ブロック並べでの動作定義を図2.5と図2.6に示す。
図2.5ではカラーブロックを適切な場所に移動させる際に使用する処理をモデル図を用いて示した。

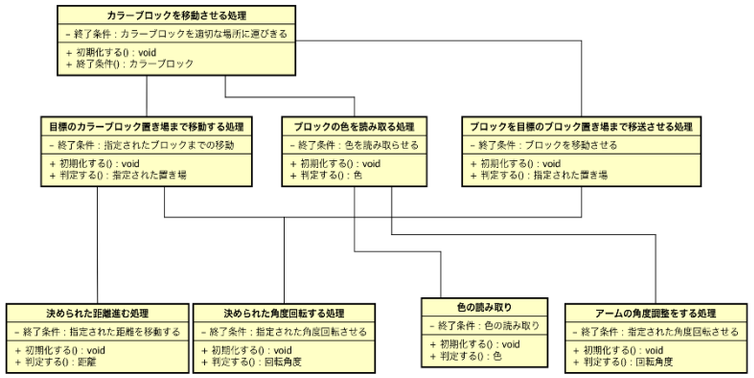


図2.5 カラーブロック並べの動作定義

図2.6ではパワーブロックを移動させる際に使用する処理をモデル図を用いて示した。

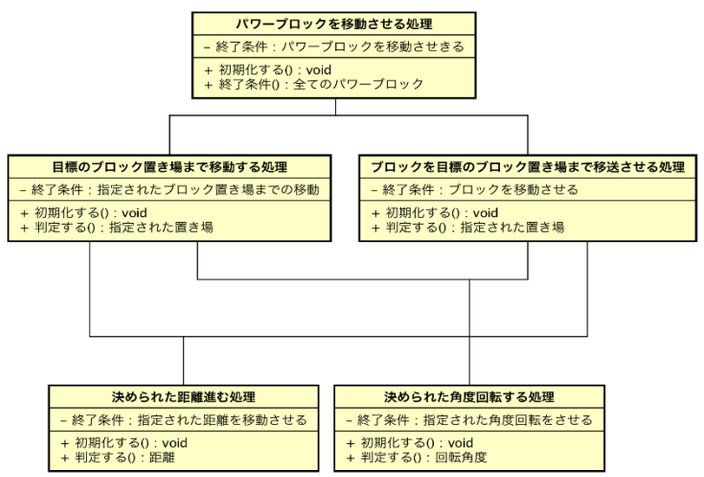


図2.6 パワーブロック並べの動作定義

・フローチャート

ブロック並べゲームの攻略の流れを図2.7に示す。

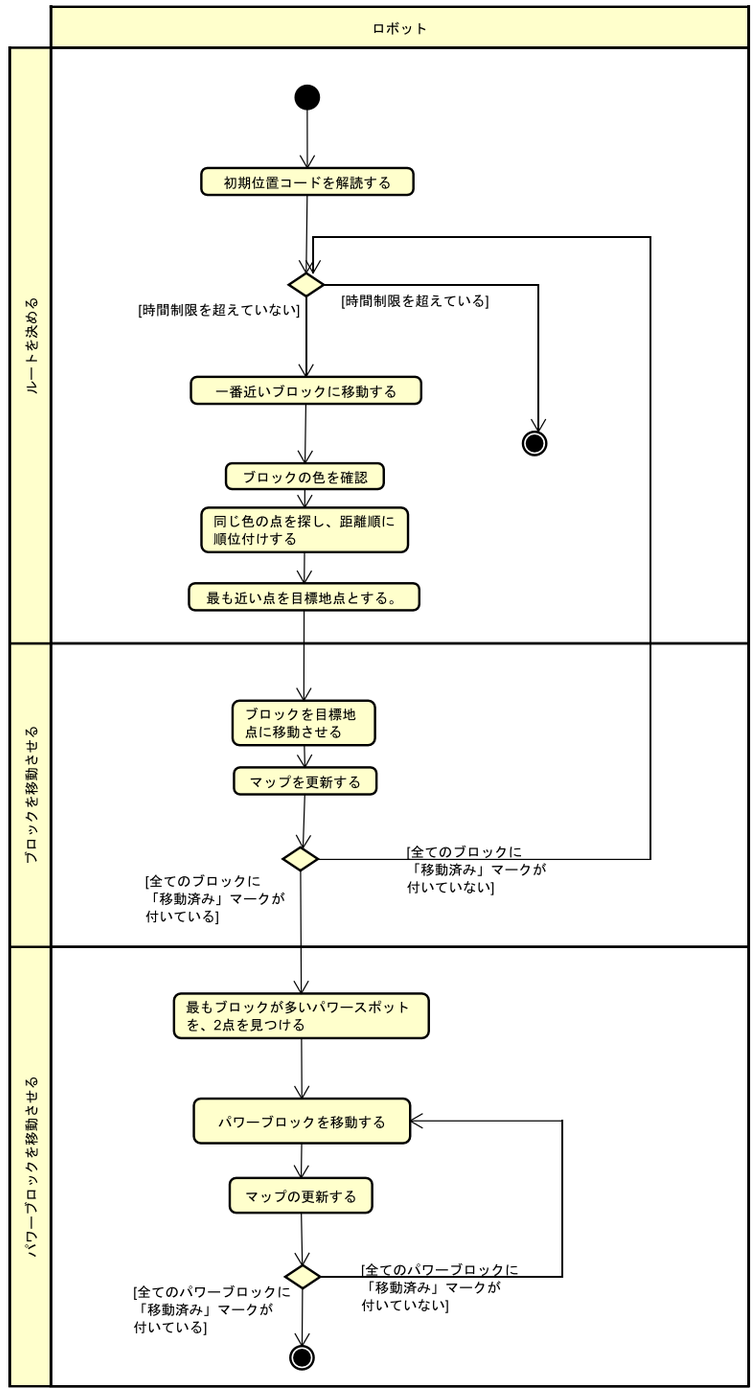


図2.7 フローチャート

・ブロックを目的地点に移動させる

図2.7 の中央、「ブロックを目的地点に移動させる」について、さらに具体的な方法を検討し、図2.8 のフローチャートにまとめた。

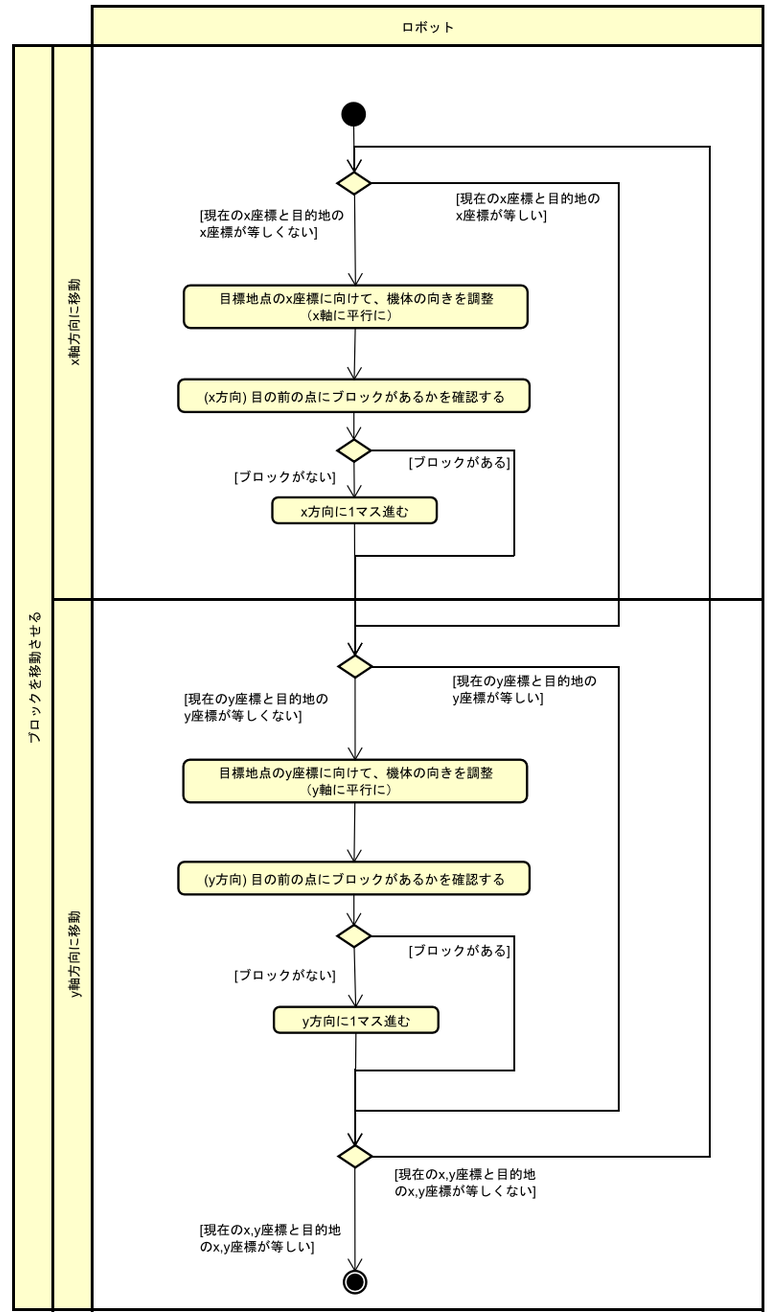


図2.8 目的地点の座標までの移動手順

3.設計モデル

ie-ryukyu

3.1 システム／ソフトウェア構造

・パッケージ構造

プログラムは、まず抽象度ごとに分け、さらに、役割ごとに分割した。
パッケージの構成を、図3.1 に示す。

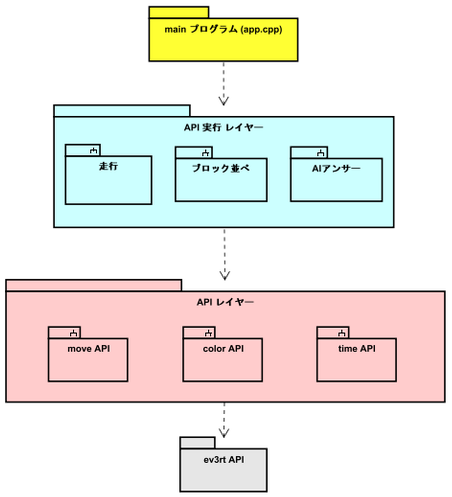


図3.1 パッケージ構造

「main プログラム」は、app.cpp 本体と、app が呼び出す高抽象度なプログラム郡である。1つ下層の「API 実行レイヤー」のプログラムを呼び出して実行する。

「API 実行レイヤー」は、比較的複雑な機能のAPI郡である。ゲーム攻略のための「演算」や「複雑な制御」が該当する。 1つ下層の、「API レイヤー」のプログラムを呼び出して実行する。

「API レイヤー」は、比較的単純な機能のAPI群である。「動作定義の動き」や「util系」の単純な処理が集められている。
「API レイヤー」は、チームメンバー全員が使うため、使用頻度が高い。よって、初期には最低限のものだけ実装し、実装ルールを簡単に決めた上で、各々で自由に拡張する方針をとった。また開発の中盤には、よく使うAPIを適宜追加・拡張していった。これにより、規格化の労力を軽減しつつ、より有用な API のみを抽出し、共有ことができた。

・クラス図

ブロック並べで使用する
クラスの関係を図3.2に示す。

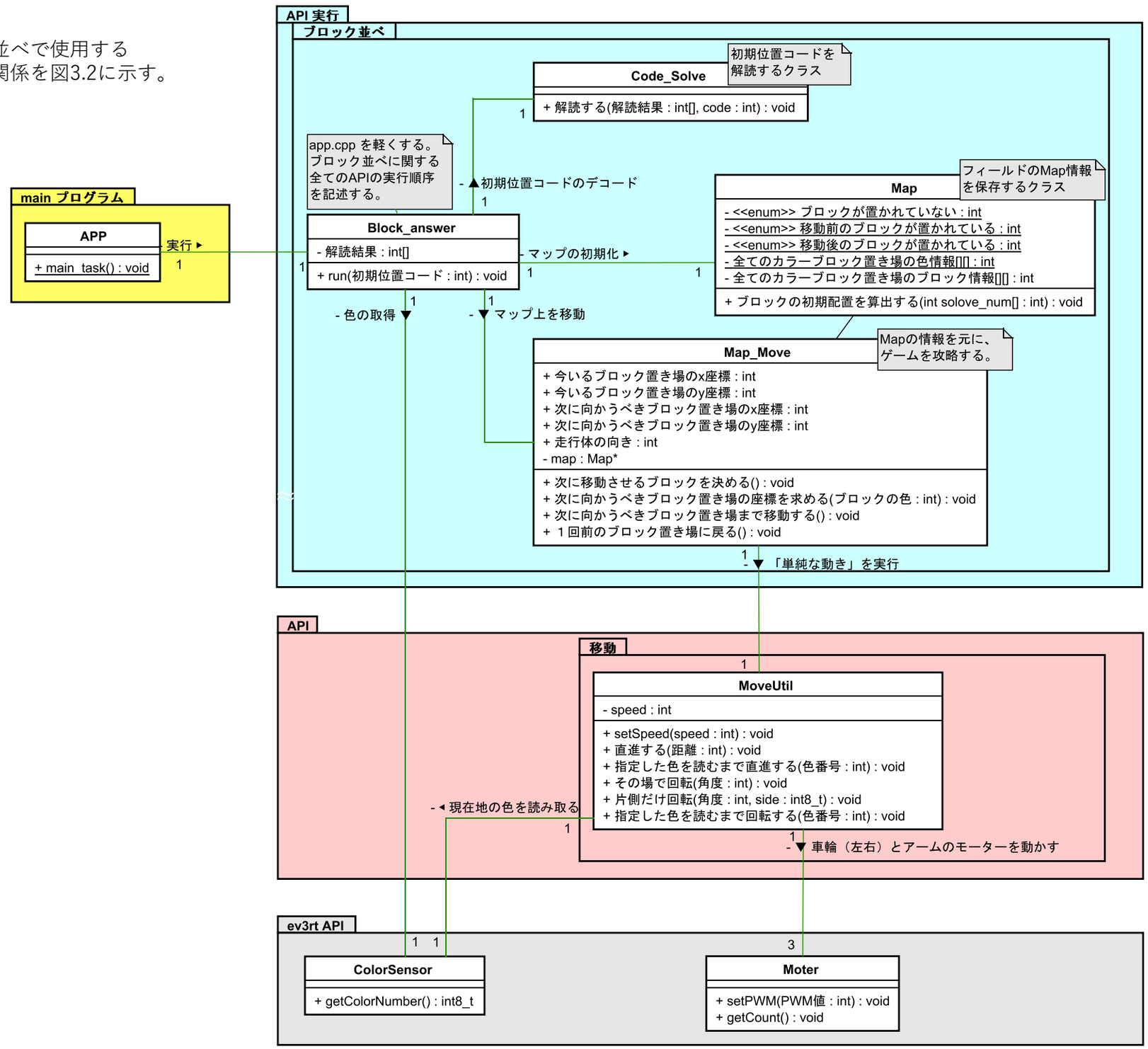


図3.2 クラス図

4.制御モデル

ie-ryukyu

4.1 ブロック並べの制御技術

・ブロックの色認識

初期位置コードから「ブロックの位置」は特定できる。しかし、「ブロックの色」についての情報は得られないため、ブロックの色を認識する機能を実装する必要がある。
ブロックの色の認識は、カラーセンサを用いて行った。具体的には、図4.1.1 のようにセンサをブロックに近づけることでブロックの色を認識する。

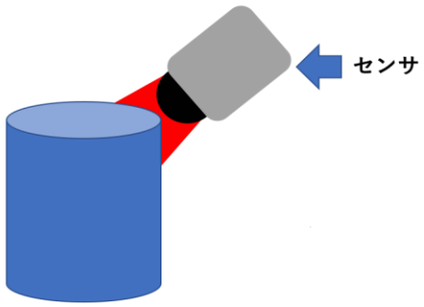


図4.1.1 ブロックの色の認識

・課題

ブロックの色を読み取るためには、センサーをある程度高く上げる必要がある。しかし、そうすると、少しのズレでも色を誤検知してしまうようになる。

・対策

課題となるズレを少なくするには、正確な制御が必要となることが考えられる。それには指定した距離、角度の処理をする必要がある。またできるだけ正確な制御をしたとしても色を読みとれない場合があると考えられる。その際にはブロック置き場の色を読み取り距離や角度の調整を行うことで上記の場合を解決することが出来ると考えられる。

・位置ズレ補正技術（自己位置修正）

フィールド上で回転や直進を何度も行うと、どうしてもズレが生じ、コースアウトや異常動作の原因になってしまう。その対策として、一般的に以下の２通りが考えられる。

1. 動作の精度を限界まで上げる。
2. 動作の前後でズレ補正動作を行う。

1.　2. の例を、それぞれ 図4.1.2 に示す。

・開発時の制約

私達チームには、1. を行う時間的余裕がなかったため、2. の自己位置修正に力を入れた。

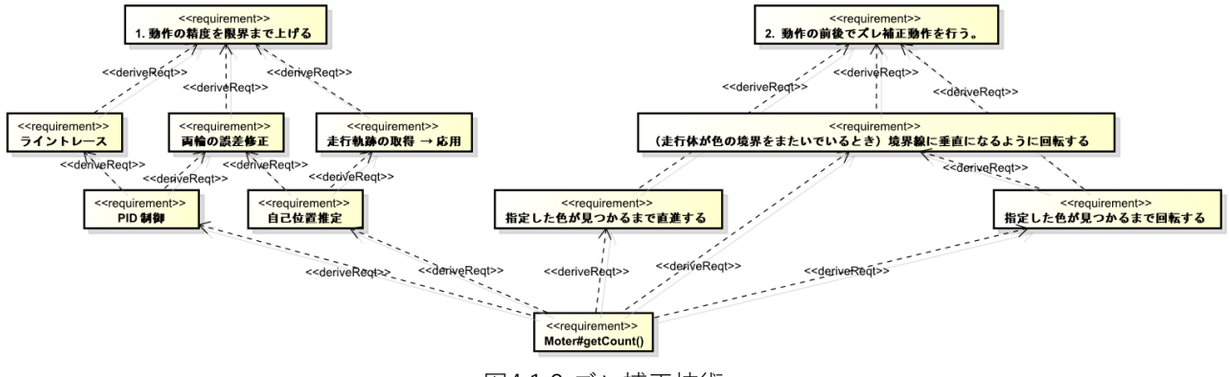


図4.1.2 ズレ補正技術

1. 動作の精度を限界まで上げる

Moter#getCount() による「走行距離の推定」と「機体の角度の推定」は行った。

2. 動作の前後でズレ補正動作を行う。

・指定した色を読み取るまで直進
目標のブロック置き場に着く直前に使用した。

・指定した色を読み取るまで回転
走行体の向きを変えるとき使用した。（目標のブロック置き場が右側にあるときなど）
（回転の際には、「カラーセンサーから走行体の中心までの距離 」≒「センサーから黒線までの距離」である必要がある。）

4.2 AIアンサー：数字画像の読み取り制御

・ゲーム攻略の流れ

ゲーム攻略のおおまかな流れを図4.2.1に示す。
今回は、図中の「8の字読み取り」と「3 return読み取り」（数字カード読み取り動作）のアルゴリズムを説明する。

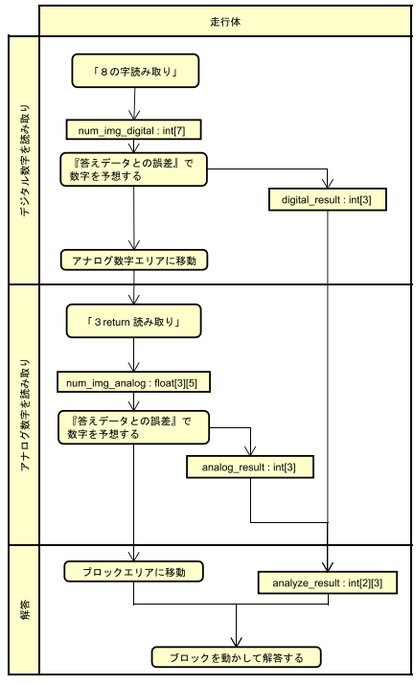


図4.2.1 アクティビティ図

・ゲーム戦略

「画素」の考え方に基づき、以下のように読み取る。
1）数字画像を分割し、各マスに配列の要素を割り当てる。
2）マス内に黒があれば、そのマスに対応する配列の要素の値を1(黒)とし、なければ0(白)とする。
アナログとデジタルでは「数字の予測が可能な”最低限の”データ」が異なるため、各々で読み取り動作を実装した。

・「8の字読み取り」（デジタル）

デジタル数字は７本の線分で構成されている。よって、「線分1本を１マス」とする。（図4.2.2の青文字）

【機能要件】
・「正確に」読み取るために、「線を垂直に横切る」
・「 ”最低限の”データ」を得るために、「同じ線を通らない最短のルート」を選ぶ。

以上を満たす動きとして、図4.2.2 の赤い一筆書きの動き(通称「８の字」) を採用する。

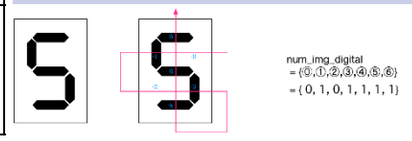


図4.2.2 「8の字読み取り」

・「3return読み取り」（アナログ）

アナログ数字は曲線で構成されているため、デジタルのような「線による分割」は難しい。
よって、図4.2.3 の青い格子で、「面による分割」を行う。

【機能要件】
・「正確に」読み取るために、黒:1、白:0 の2値ではなく、「読み取った色のうち何％が黒か」を記録する。
・「 ”最低限の”データ」を得るために3×5 で分割する。（図4.2-3 下側の 5×7の圧縮画像で、同じパターンがないことを確認済み）

以上を考慮し、図4.2.3 の赤い一筆書きの動き(通称「3return」)を採用する。

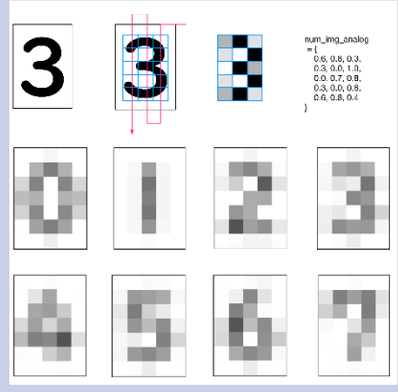


図4.2.3 「 3return読み取り 」

・数字画像の読み取り制御

理論上は、上述の「8の字」と「3return」によって、数字の判別に必要十分なデータを得られるはずである。しかし実際には、回転や直進時にズレが生じるため、正確に読み取ることは難しい。
そこで、今回も図4.1.2 に示した「位置ズレ補正技術」（自己位置修正）を挟むことで、成功率の向上を図った。

・位置ズレ補正　の使用箇所

・指定した色を読み取るまで直進
「数字カードからマットに出るとき」「数字カード間の移動時」
「その他、マーカーに着地するとき」に使用した。（なお、2回連続で使用する場合は、誤動作を防ぐため、数cm 進んで前の色を読み取らないようにする必要がある）

・指定した色を読み取るまで回転
数字カード読み取りの際の「回転」のほぼ全てで使用した。

・境界線に垂直になるように回転する
数字カード読み取りの「回転」の直後で特にズレやすいところを調べ、そこに追加した。