プログラミング演習 レポート

ミニゲーム

提出期限　2021年1月18日 17:00

組番号　408
学籍番号　17406

氏名　金澤雄大

1　目的

後期のプログラミング演習で学習した内容の理解度を高めるために, ミニゲームを作成することを目的とする.

2　ミニゲームの説明

本章では, ゲームの概要, 用語, 設定, 仕様の 4 つについて述べる.

2.1　ゲームの概要

ミニゲームとして,「桃太郎電鉄」[1](以下, 桃鉄) をイメージした「ちゃま鉄」を作成した.「ちゃま鉄」は鉄道会社の運営をイメージしたすごろく形式のゲームである. 本ゲームは,3 年決戦で 3 人でのプレイを想定しており,CPU キャラは存在しない. また桃鉄における「貧乏神」,「すりの銀次」,「臨時収入」を代表とする要素は開発時間の都合上実装していない.

2.2　プレイヤーと物件の設定

プレイヤーおよび物件の設定について説明する. 先述した通り, プレイヤー (社長と呼ぶ) は 3 人おり, ゲーム内ではターン順に「プレイヤー 1 社長」,「プレイヤー 2 社長」,「プレイヤー 3 社長」と呼ばれる仕様になっている. さらに各社長には色の設定が行われている. 社長名と色の対応を次に示す.

- プレイヤー 1 社長 ··· 青
- プレイヤー 2 社長 ··· ピンク
- プレイヤー 3 社長 ··· 黄色

各社長には「所持金」,「総資産」という 2 つのパラメータが割り振られている. ゲームスタート時の所持金は 1 億円, 総資産は 0 円である. 所持金は社長が手元に持っているお金のことである. 停車する駅には「物件」を購入できる「物件駅」というものがあり, この物件を購入することで総資産を増やすことができる. 図 1 に長野駅の物件の例を示す. 図 1 には 6 つの物件がある. 物件には「価格」,「収益率」という 2 つのパラメータがある. 例えば「りんごえん」の場合, 価格が「600 万円」, 収益率が「120%」である. 価格はその物件を購入するために必要な所持金であり, 収益率は決算 (後述) で手に入るお金の割合を示している. また, 同じ駅の物件を 1 人の社長がすべて購入すると「独占」という状態になる. 独占状態になった駅の収益率は 2 倍になり, 決算で 2 倍の収益が得られる仕様になっている.

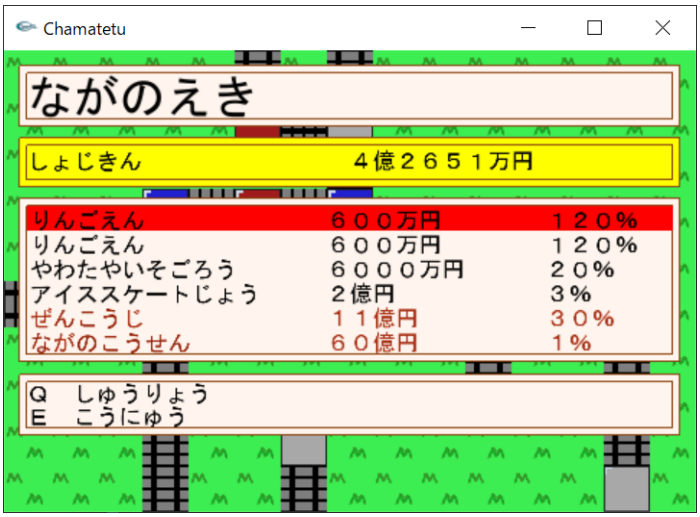

図 1: 物件の例 (長野駅)

2.3　ゲームの進行方法

ゲームの進行について説明する. ここではゲームの進行の概要について説明し, 実際の画面表示については実装部分と共に述べる. 図 2 にゲーム進行のフローチャートを示す. ゲームを開始すると, 初期設定が行われ, タイトル画面が表示される. 初期設定としてはゲームスタート時の駅の設定および年月の設定が行われる. ゲームスタート時の駅は長野駅, 年月は「1 年目 4 月」に設定が行われる仕様にした. 次に目的地の設定が行われる. 目的地の処理の詳細については!で述べる. 目的地の設定が完了するとゲームのメイン部分である社長の行動が始まる. 各社長はターン中にサイコロを 1 つふって出た目の数だけ進む, もしくはカードを使う, のどちらかの行動を行うことができる. 各社長が 1 回行動すると, 年月の経過処理として 1ヵ月経過する処理が行われる. 年月の経過処理後の処理は月によって変化する.3 月でない場合は再び社長の行動の処理が行われる.3 月の場合は社長の行動の前に「決算」という処理が行われる. 決算の処理については!で述べる. さらに 3 年目の場合は決算として最終成績が表示されゲームの終了処理が行われる.

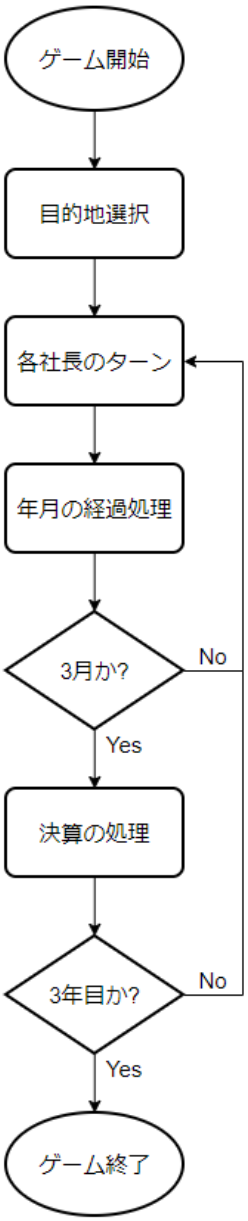

図 2: ゲームの進行

2.4 マップの設定

マップの設定について説明する. 図 3 に本ゲームのマップを示す. 図 3 の地名からも読み取れるように, 本ゲームは長野県を舞台にしている. ただし, 実際のゲームでは駅名は表示されない仕様になっている. マップは 32×32 の画像を敷き詰める形で描画しており, サイズは 960×960 である. なお, ウィンドウサイズは幅 480, 高さ 320 のため実際のゲームでは, 行動中の社長を中心として画面におさまる部分だけを描画している.

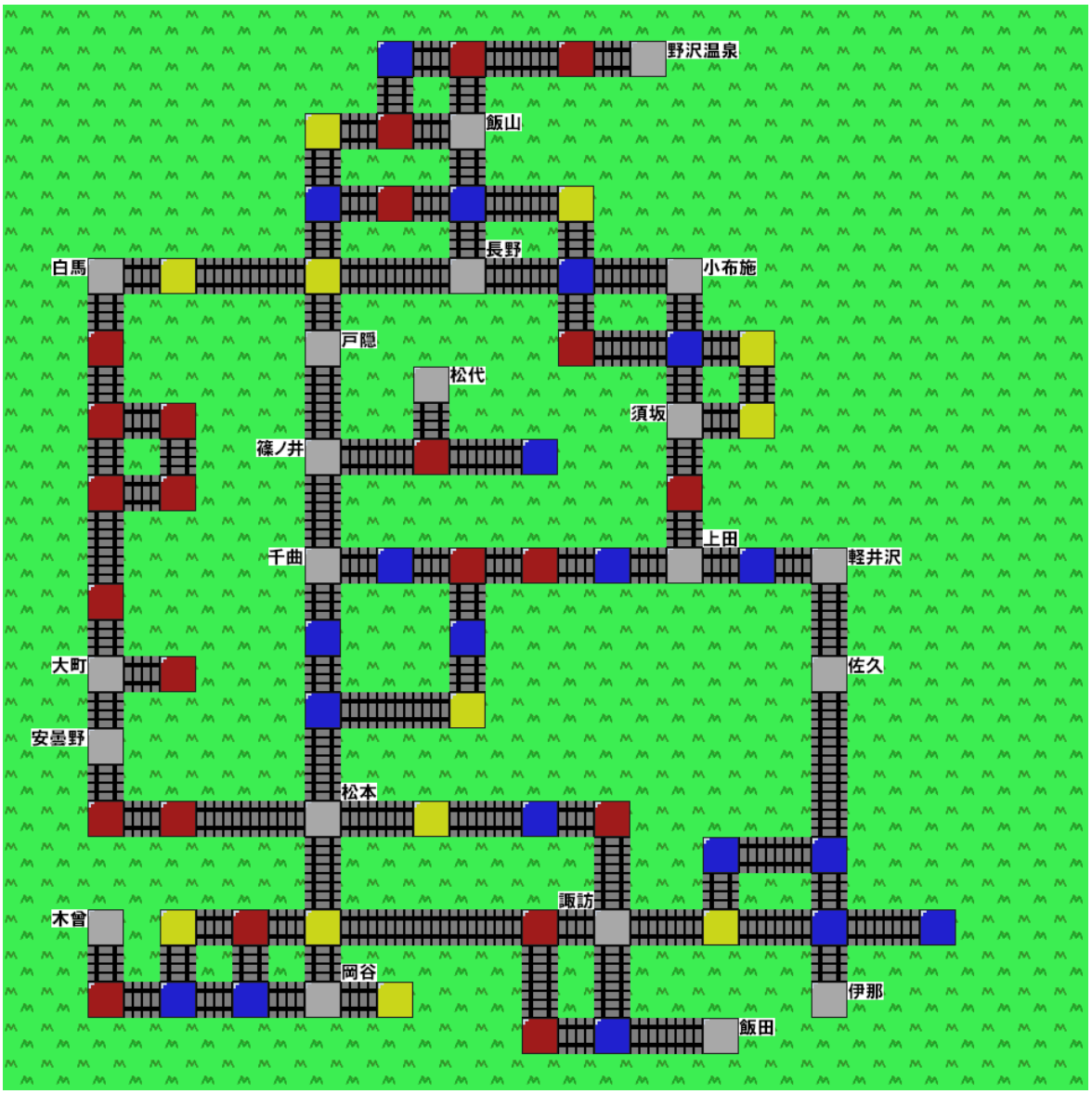

図 3: ゲームのマップ

マップを描画する画像には表 1 に示す種類のものがある. これらの画像は「/mapparts」に保存されている. 背景は季節によって変化する. 月と季節は次に示すようになっている. 図 3 のマップは背景が春の場合である. 季節ごとに背景が変化する仕様は桃鉄を参考にした.

- 春：3 月〜5 月
- 夏：6 月〜8 月
- 秋：9 月〜11 月
- 冬：12 月〜2 月

表 1: マップとして描画される画像の種類

画像の意味	画像のファイル名	実際の色や模様
背景 (春)	season1.png	明るい緑
背景 (夏)	season2.png	濃い緑
背景 (秋)	season3.png	茶色
背景 (冬)	season4.png	白
プラス駅	map1.png	青
マイナス駅	map2.png	赤
カード駅	map3.png	黄色
物件駅	map4.png	灰色
線路 (縦)	map5.png	灰色背景に黒の線路
線路 (横)	map6.png	灰色背景に黒の線路
目的地駅	map7.png	灰色背景に駅のマーク

2.5 駅の設定

駅の設定について説明する. サイコロをふって移動した社長が停車できる駅の種類には, 表 1 に示したように,「プラス駅」,「マイナス駅」,「カード駅」,「物件駅」,「目的地駅」の 5 つがある. プラス駅は停車するとお金がもらえる駅である. もらえるお金は夏が最も多く, 冬が最も少ない仕様になっている. マイナス駅は停車すると所持金が減少する駅である. 減少する金額は夏が最も少なく, 冬が最も多い仕様になっている. 減少する額によっては所持金が負になる, いわゆる借金という状態になることがある. この場合, 物件を売却することで借金を返済する処理が行われる. 本ゲームでの借金の返済は売却する物件を選択する方式ではなく, 自動で売却する物件を選ぶ方式を採用した. 売却する物件の優先順位は次に示す通りである. なお, 所持している全ての物件を売却しても借金が返済できない場合は所持金が負となった状態でターンが終了する.

1. 独占している駅の物件でなく, 借金額よりも価格が高い物件
2. 独占している駅の物件でなく, 借金額よりも価格が低い物件
3. 独占している物件で, 借金額よりも価格が高い物件
4. 独占している物件で, 借金額よりも価格が低い物件

カード駅は停車するとカードがもらえる駅である. カードは 5 枚まで所持することができ, カード駅に停車したときに既に 5 枚カードを持っている場合, この処理はスキップされる. カードは表 2 に示す 8 種類がある. カード名は桃鉄を参考にした. 表 2 のカードのうち, 急行カード, 特急カード, 新幹線カードの 3 種類はカードを仕様したあとにサイコロをふって移動することができる. 他のカードについは, 成功, 失敗にかかわらずターンが終了する.

表 2: カード名と効果

カード名	カードの効果
急行カード	サイコロが 2 個に増える.
特急カード	サイコロが 3 個に増える.
新幹線カード	サイコロが 4 個に増える.
サミットカード	すべての社長を自分のマスに集める.3 分の 2 の確率で成功する.
ぶっとびカード	ランダムな物件駅に移動する.
10 億円カード	10 億円が手に入る.
徳政令カード	借金を負っている社長の所持金が 0 円になる.
剛速球カード	他の社長のカードをすべて破棄する.2 分の 1 の確率で成功する.

2.6 決算の処理

決算の処理について説明する. 決算は 3 月が終了すると行われる処理である. 決算は, 所持している物件に応じて各社長の所持金が増加する処理である. ある社長が決算で得られる金額 S を計算する方法について説明する. 物件を n 個持っており, 所持している i 番目の物件の価格 p_i, 収益率 r_i, その物件が所属する駅が自分の独占のとき $d_i = 2$, 独占でないとき $d_i = 1$ とする. このとき, 決算で得られる金額 S は式 (1) で表せる.

$$S = \sum_{i=1}^{n} \frac{p_i r_i d_i}{100} \tag{1}$$

例えば, ある社長が, 図 1 の「やわたやいそごろう」と「アイススケートじょう」を所持している場合に決算でもらえる金額を計算してみる. 式 (1) に値を代入して計算を行うと, 式 (4) に示すように 1800 万円になる. この例では独占はしていないから d_i は常に 1 である.

$$\begin{aligned} S &= \sum_{i=1}^{n} \frac{p_i r_i d_i}{100} && (2)\\ &= \frac{1}{100}\left(6000 \times 10^4 \cdot 20 + 20000 \times 10^4 \cdot 3\right) && (3)\\ &= 1800 \times 10^4 && (4) \end{aligned}$$

本ゲームは 3 年決戦であるため,3 年目の決算は「最終成績」という形で表示される. 最終成績を表示した後はゲームを終了するように促す画面を表示する.

3 実行環境とビルド方法

本章では, 実行環境, ビルド方法, ディレクトリ構造の 3 つについて述べる.

3.1 実行環境

実行環境を 3 に示す.gcc とは「GNU Compiler Collection」の略称で,GNU プロジェクトが公開しているコンパイラのことである. make は Makefile にプログラムのコンパイルやリンクの方法を指示することで, コンパイルを簡単に行うことができるツールのことである. make を用いることは,gcc コンパイル時に, 長いオプションを入力しなくてよい, ファイルの更新を取得して必要なものだけをコンパイルしてくれるという利点がある.

表 3: 実行環境

CPU	Intel(R) Core(TM) i7-6500U 2.50GHz
メモリ	16.0GB DDR4
OS	Microsoft Windows 10 Home
gcc	version 9.3.0
make	version 4.3

3.2 ビルド方法

ビルド方法について説明する. まず,「j17406.tar.gz」を保存したディレクトリに移動する. 次にリスト 1 に示すコマンドを実行する. リスト 1 のコマンドを実行することで,j17406.tar.gz が解凍される.

リスト 1: j17406.tar.gz の解凍

```
gzip -dv j17406.tar.gz
tar xvf j17406.tar
```

解凍を行えたから, リスト 2 の 1 行目のコマンドを実行してビルドを行う. リスト 2 のコマンドを実行して「j17406.exe」が生成されていればビルド成功である.「j17406.exe」の実行はリスト 2 の 2 行目のコマンドで行う. リスト 2 の 2 行目のコマンドを実行して図 4 に示す画面が表示されれば, ゲームの起動が成功している.

リスト 2: make コマンド

```
make
.\j17406.exe
```

図 4: ゲームのスタート画面

3.3 ディレクトリ構造

リスト 3 に「/j17406」のディレクトリ構造を示す. リスト 3 は tree コマンドを用いてディレクトリ構造を表示したものである. ここでは, 深さ 1 のファイル, ディレクトリのみを表示している. なぜなら, 日本語画像や物件情報を保存しているディレクトリがあるため全てのファイルを表示すると見にくくなってしまうからである. ゲームの実装のためのコードは「game.c」および「j17406.c」に記述している. また, 関数, 定数の定義は「game.h」に記述している. 画像は, 画像の種類ごとにディレクトリを分けて保存している. ディレクトリ名と保存している画像の種類は表 4 の通りである.property.txt および「/property」に保存されている txt ファイルは駅の情報および物件の情報を保存している.

リスト 3: ディレクトリ構造

```
j17406
├── Makefile
├── charimg
├── description.html
├── descriptionimg
├── dice
├── eventparts
├── game.c
├── game.h
├── icon.o
├── j17406.c
├── mapparts
├── property
├── property.txt
└── readme.txt
```

表 4: 画像を保存するディレクトリ

ディレクトリ名	保存している画像の種類
charimg	日本語を画面に表示するための画像
dice	サイコロの画像
eventparts	社長の画像, スタート画面, 決算, ゲーム終了画面
mapparts	マップ描画のための画像
descriptionimg	description.html のための画像

4 プログラムの説明と実行結果

本章では次に示すプログラムの説明および実行結果について述べる. なお, プログラム中に登場する定数の値は付録の「game.h」(リスト!) を参照してほしい.

1. playerstatus 構造体
2. propertystatus 構造体
3. stationstatus 構造体
4. Map 配列の定義
5. 日本語プロトコルの定義
6. 画像の読み込み
7. 画像および日本語の表示
8. 駅および物件情報の読み込み
9. メイン関数 (j17406.c)
10. 画面サイズ変更への対応 (Reshape 関数)
11. ゲームの進行状況管理
12. ゲームの初期化とタイトル画面の表示
13. 目的地の設定処理

14. プレイヤーおよびマップの描画処理
15. ターンのはじめの処理
16. サイコロをふる処理
17. マス移動および停車駅の判定処理
18. 物件駅の処理
19. プラス駅の処理
20. マイナス駅および借金の処理
21. カード駅の処理
22. ターン終了時の処理
23. 決算および最終成績の処理
24. カードを使用したときの処理

4.1 playerstatus 構造体

playerstatus 構造体の定義と初期化について説明する. まず,playerstatus 構造体の定義について説明する. playerstatus 構造体は一人の社長の情報を保持するための構造体である. リスト 4 に「game.h」における playerstatus 構造体の定義を示す. playerstatus 構造体は「社長名」,「所持金」,「総資産」,「現在の座標 (x,y)」,「カード枚数」,「カードの通し番号」の 7 つをメンバとして持っている. 所持金および総資産は万円単位で扱うものとする. 例えば所持金が「2200 万円」場合, メンバ money には 2200 が代入される. これ以降にも金額を扱うための変数が登場するが, そのすべての変数は万円単位で扱うものとする. また,11 行目のように playerstatus 構造体の配列を定義することで, プレイ人数 3 人分の情報を保持する構造体の配列を作成している.

リスト 4: playerstatus 構造体の定義と初期化

```
1  // プレイヤーの情報構造体
2  struct playerstatus{
3      char name[NAMEMAX]; // プレイヤー名
4      int money; // 所持金
5      int assets; // 総資産
6      int x; // x座標(実描画座標)
7      int y; // y座標(実描画座標)
8      int cardnum; // 持っているカード枚数
9      int card[CARDMAX]; // カードの番号記憶
10 };
11
12     typedef struct playerstatus player;
13     player players[PLAYERNUM]; // 人数分の配列を確保
```

次に playerstatus 構造体を初期化する関数について説明する. リスト 5 に,playerstatus 構造体を初期化する関数である InitPlayer 関数のコードを示す.InitPlayer 関数の内部では,for 文を用いて playerstatus 構造体の配列を初期化している. リスト 5 中の定数の値は INITX が 224(7×32),INITY が 160(5×32),INITMONEY は 10000 である. メンバ (x,y) に代入している値は,!

リスト 5: InitPlayer 関数

```
1  // プレイヤー構造体を初期化
2  void InitPlayer(void){
3      int i,j;
4      for(i=0;i<PLAYERNUM;i++){
5          //プレイヤーhoge
```

```
        sprintf(players[i].name,"llpureiiyallms%d",i+1);
        players[i].x=INITX;
        players[i].y=INITY;
        players[i].money=INITMONEY;
        players[i].assets=0;
        players[i].cardnum=0;
        for(j=0;j<CARDMAX;j++){
            players[i].card[j]=0;
        }
    }
}
```

4.2 propertystatus 構造体

propertystatus 構造体は一つの物件の情報を保持するための構造体である.「game.h」における propertystatus 構造体の定義をリスト 6 に示す. propertystatus 構造体は「物件名」,「物件保持者」,「価格」,「収益率」の 4 つをメンバとして持っている. 物件保持者は表 5 のルールで扱うものとする.

リスト 6: propertystatus 構造体の定義

```
// 物件情報構造体
struct propertystatus{
char name[STRMAX]; // 物件名
int holder; // 物件所持者
int price; // 価格
int earnings; // 収益率
};

typedef struct propertystatus property;
```

表 5: 物件保持者メンバの意味

値	保持者
0	保持者なし
1	社長 1
2	社長 2
3	社長 3

4.3 stationstatus 構造体

stationstatus 構造体は一つの駅の情報を保持するための構造体である. リスト 7 に stationstatus 構造体の定義を示す. stationstatus 構造体は,「駅名」,「駅の座標 (x,y)」,「独占フラグ」,「物件の数」,「propertystatus 構造体の配列」の 6 つをメンバとして持つ. 駅の座標 (x,y) は playerstatus 構造体のような実座標ではなく, マップを描画するための配列のインデックスである. 独占フラグはその駅を誰が独占しているかを判別するために用いる. 独占フラグの値とその意味は表 5 と同じである.

リスト 7: stationstatus 構造体の定義と初期化

```
// 駅情報構造体
struct stationstatus{
char name[STRMAX]; // 駅名
int x; // x座標
int y; // y座標
int ismonopoly; // 独占フラグ
int propertynum; // 物件数
property plist[PROPERTMAX]; // 物件情報構造体の配列
};
```

```
10
11 typedef struct stationsstatus station;
12 station stations[STATIONNUM]; // 駅の数分の配列を確保
13 station distination; // 目的地配列
```

4.4 Map 配列の定義

マップの情報は Map 配列が保持している.Map 配列の定義をリスト 8 に示す. マップのサイズは 30×30 で, 配列のサイズは 30×31 である. x 方向の配列のサイズが 1 大きいのは,null 文字を格納するためである.Map 配列の文字列は「A」,「B」,「C」,「P」,「M」,「-」,「|」のいずれかの文字から構成されている. 表 6 に文字と実際に表示される画像の関係を示す.

リスト 8: Map 配列の定義

```
1  // マップ配列
2  char Map[YMAX][XMAX+1] = { //NULL文字に気を付ける
3          //01234567890123456789012345678 9
4           "AAAAAAAAAAAAAAAAAAAAAAAAAAAAA", // 0
5           "AAAAAAAAAAP-M--M-BAAAAAAAAAAA", // 1
6           "AAAAAAAAAA|A|AAAAAAAAAAAAAAA", // 2
7           "AAAAAAAAAC-M-BAAAAAAAAAAAAAAA", // 3
8           "AAAAAAAAA|AAA|AAAAAAAAAAAAAAA", // 4
9           "AAAAAAAAAP-M-P--CAAAAAAAAAAAAA", // 5
10          "AAAAAAAAA|AAA|AA|AAAAAAAAAAAA", // 6
11          "AAAB-C---C---B--P--BAAAAAAAAAA", // 7
12          "AAA|AAAAA|AAAAAA|AA|AAAAAAAAA", // 8
13          "AAAMAAAAABAAAAAAM--P-CAAAAAAAA", // 9
14          "AAA|AAAAA|AABAAAAAA|A|AAAAAAAA", // 0
15          "AAAM-MAAA|AA|AAAAAAB-CAAAAAAAA", // 1
16          "AAA|A|AAAB--M--PAAA|AAAAAAAAA", // 2
17          "AAAM-MAAA|AAAAAAAAAMAAAAAAAAA", // 3
18          "AAA|AAAAA|AAAAAAAAA|AAAAAAAAA", // 4
19          "AAA|AAAAAB-P-M-M-P-B-P-BAAAAAA", // 5
20          "AAAMAAAAA|AAA|AAAAAAAAA|AAAAAA", // 6
21          "AAA|AAAAAPAAAPAAAAAAAAA|AAAAAA", // 7
22          "AAAB-MAAA|AAA|AAAAAAAAABAAAAAA", // 8
23          "AAA|AAAAAP---CAAAAAAAAA|AAAAAA", // 9
24          "AAABAAAAA|AAAAAAAAAAAA|AAAAAA", // 0
25          "AAA|AAAAA|AAAAAAAAAAAA|AAAAAA", // 1
26          "AAAM-M---B--C--P-MAAAAA|AAAAAA", // 2
27          "AAAAAAAAA|AAAAAAA|AAP--PAAAAAA", // 3
28          "AAAAAAAAA|AAAAAAA|AA|AA|AAAAAA", // 4
29          "AAABAC-M-C-----M-B--C--P--PAAA", // 5
30          "AAA|A|A|A|AAAAA|A|AAAAA|AAAAAA", // 6
31          "AAAM-P-P-B-CAAA|A|AAAAABAAAAAA", // 7
32          "AAAAAAAAAAAAAAM-P--BAAAAAAAAA", // 8
33          "AAAAAAAAAAAAAAAAAAAAAAAAAAAAA"  // 9
34 };
```

表 6: 文字と画像の対応

文字	表示される画像
A	背景
B	物件駅
C	カード駅
P	プラス駅
M	マイナス駅
-	線路 (横)
|	線路 (縦)

4.5 日本語プロトコルの定義

桃鉄をイメージしたゲームを実装するうえで, 日本語を画面に表示できなければ, ローマ字が並んでわかりにくい. しかし GLUT は日本語に対応していない. そこで画像を用いて日本語をゲーム画面に描画する機能を作成した. これを日本語プロトコルと呼ぶことにする. 文字色は黒と赤のどちらかで描画することが可能である. 作成した日本語プロトコルでは次に示す文字を画面に表示することができる. 一部のアルファベット, 記号, 漢字をまとめて特殊文字と呼ぶことにする.

- 50 音 (ひらがな, カタカナ)
- 濁音 (ひらがな, カタカナ)
- 半濁音 (ひらがな, カタカナ)
- 小文字 (っ,ゃ,ゅ,ょ)(ひらがな, カタカナ)
- 数字
- 一部のアルファベット (W,A,S,D,E,Q)
- 一部の記号 (読点, 句点,%, マイナスの記号 (-), プラスの記号 (+))
- ゲームに頻出する漢字 (億, 万, 円)

これらの画像は「/charimg」に保存されている. 画像のサイズはすべて 32×32px である. 画像名の意味は図 5 の通りである. どの文字か判別する 2 文字はコード中で日本語を表示するための文字列に対応している. 実際の文字と日本語プロトコルにおける文字の表現方法は表 7 の通りである. 改行およびスペースは画像にはないが, 画像を表示する位置を調整することで表現できる. 表 7 の情報は game.c でリスト 9 に示すように定義されている.

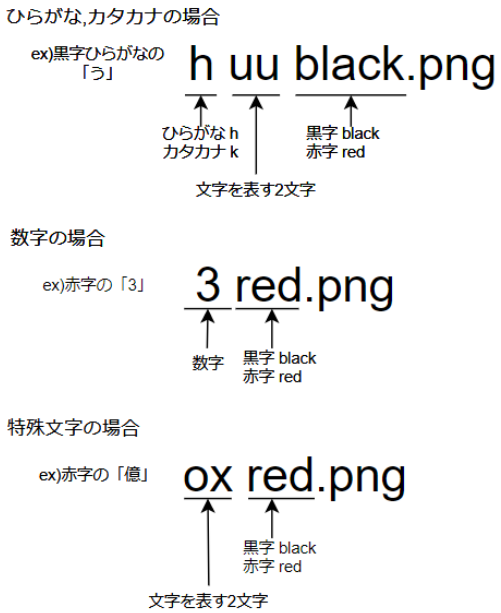

図 5: 画像の名前の意味

表 7: 日本語プロトコルの文字表現

あ aa	い ii	う uu	え ee	お oo
か ka	き ki	く ku	け ke	こ ko
さ sa	し si	す su	せ se	そ so
た ta	ち ti	つ tu	て te	と to
な na	に ni	ぬ nu	ね ne	の no
は ha	ひ hi	ふ hu	へ he	ほ ho
ま ma	み mi	む mu	め me	も mo
や ya	-	ゆ yu	-	よ yo
ら ra	り ri	る ru	れ re	ろ ro
わ wa	-	を wo	-	ん nn
ゃ la	-	ゅ lu	っ lt	ょ lo
が ga	ぎ gi	ぐ gu	げ ge	ご go
ざ za	じ zi	ず zu	ぜ ze	ぞ zo
だ da	ぢ di	づ du	で de	ど do
ば ba	び bi	ぶ bu	べ be	ぼ bo
ぱ pa	ぴ pi	ぷ pu	ぺ pe	ぽ po
0 0	1 1	2 2	3 3	4 4
5 5	6 6	7 7	8 8	9 9
円 ex	万 mx	億 ox	% px	- ms
+ ps	句点 mr	読点 tn	Q xq	W xw
E xe	A xa	S xs	D xd	-
改行 xx	スペース ss	-	-	-

リスト 9: jpProtcol 配列

```
// 日本語プロトコル
char jpProtcol[JPMAX+SPMAX][3] = {"aa","ii","uu","ee","oo",
                          "ka","ki","ku","ke","ko",
                          "sa","si","su","se","so",
                          "ta","ti","tu","te","to",
                          "na","ni","nu","ne","no",
                          "ha","hi","hu","he","ho",
                          "ma","mi","mu","me","mo",
                          "ya","yu","yo",
                          "ra","ri","ru","re","ro",
                          "wa","wo","nn",
                          "lt","la","lu","lo",
                          "ga","gi","gu","ge","go",
                          "za","zi","zu","ze","zo",
                          "da","di","du","de","do",
                          "ba","bi","bu","be","bo",
                          "pa","pi","pu","pe","po",
                          "0","1","2","3","4","5"
                          ,"6","7","8","9",
                          "ex","mx","ox","px","ms","ps",
                          "mr","tn","xq","xw","xe","xa","xs","xd"
                          };
```

4.6 画像の読み込み

画像を読み込む方法について説明する. リスト 10 に画像を読み込むための readImg 関数のコードを示す. 処理の内容はファイル名や読み込み先を変えて画像を読み込んでいるだけだから, リスト 10 の 7 行目から 11

行目のイベントマップの読み込みを例に説明する. 画像を読み込むためには 2 つの変数が必要である. 今の例では spimg と spinfo である. 変数 spimg には読み込んだ画像を識別するための値が代入され, 変数 spinfo には読み込んだ画像のと横幅, 縦幅を代表とする情報が格納される. これらの変数に pngBind 関数を用いて読み込んだ画像の情報を与えることで画像の読み込みを行っている.

リスト 10: readImg 関数

```
// 画像読み込み
void readImg(void){
    int i;
    char fname[100];

    // イベントマップ読み込み
    for(i=0;i<SP_NUM;i++){
        sprintf(fname,".\\eventparts\\sp%d.png",i+1);
        spimg[i] = pngBind(fname, PNG_NOMIPMAP, PNG_ALPHA,
       &spinfo[i], GL_CLAMP, GL_NEAREST, GL_NEAREST);
    }

   // 季節マップ読み込み
    for(i=0;i<SEASON_NUM;i++){
        sprintf(fname,".\\mapparts\\season%d.png",i+1);
        seasonimg[i] = pngBind(fname, PNG_NOMIPMAP, PNG_ALPHA,
       &seasoninfo[i], GL_CLAMP, GL_NEAREST, GL_NEAREST);
    }

    // マップイメージ読み込み
    for(i=0;i<=MAP_NUM;i++){
        sprintf(fname,".\\mapparts\\map%d.png",i+1);
        mapimg[i] = pngBind(fname, PNG_NOMIPMAP, PNG_ALPHA,
       &mapinfo[i], GL_CLAMP, GL_NEAREST, GL_NEAREST);
    }
    // プレイヤー画像を読み込み
    for(i=0;i<PLAYERNUM;i++){
        sprintf(fname,".\\eventparts\\player%d.png",i+1);
        playerimg[i] = pngBind(fname, PNG_NOMIPMAP, PNG_ALPHA,
       &playerinfo[i], GL_CLAMP, GL_NEAREST, GL_NEAREST);
    }

    // サイコロの画像を読み込み
    for(i=0;i<DICEMAX;i++){
        sprintf(fname,".\\dice\\dice%d.png",i+1);
        diceimg[i] = pngBind(fname, PNG_NOMIPMAP, PNG_ALPHA,
       &diceinfo[i], GL_CLAMP, GL_NEAREST, GL_NEAREST);
    }
    // read Hiragana black
    for(i=0;i<JPMAX;i++){
        sprintf(fname,".\\charimg\\h%sblack.png",jpProtcol[i]);
        hblackimg[i] = pngBind(fname, PNG_NOMIPMAP, PNG_ALPHA,
        &hblackinfo[i], GL_CLAMP, GL_NEAREST, GL_NEAREST);
    }

    // read Hiragana red
    for(i=0;i<JPMAX;i++){
        sprintf(fname,".\\charimg\\h%sred.png",jpProtcol[i]);
        hredimg[i] = pngBind(fname, PNG_NOMIPMAP, PNG_ALPHA,
        &hredinfo[i], GL_CLAMP, GL_NEAREST, GL_NEAREST);
    }
    // read Katakana black
    for(i=0;i<JPMAX;i++){
        sprintf(fname,".\\charimg\\k%sblack.png",jpProtcol[i]);
        kblackimg[i] = pngBind(fname, PNG_NOMIPMAP, PNG_ALPHA,
        &kblackinfo[i], GL_CLAMP, GL_NEAREST, GL_NEAREST);
    }
    // read Katakana red
    for(i=0;i<JPMAX;i++){
        sprintf(fname,".\\charimg\\k%sred.png",jpProtcol[i]);
        kredimg[i] = pngBind(fname, PNG_NOMIPMAP, PNG_ALPHA,
        &kredinfo[i], GL_CLAMP, GL_NEAREST, GL_NEAREST);
    }
    // read Special Str red
    for(i=JPMAX;i<JPMAX+SPMAX;i++){
```

```
66          sprintf(fname,".\\charimg\\%sred.png",jpProtcol[i]);
67          hredimg[i] = pngBind(fname, PNG_NOMIPMAP, PNG_ALPHA,
68          &hredinfo[i], GL_CLAMP, GL_NEAREST, GL_NEAREST);
69      }
70      // read Special Str black
71      for(i=JPMAX;i<JPMAX+SPMAX;i++){
72          sprintf(fname,".\\charimg\\%sblack.png",jpProtcol[i]);
73          hblackimg[i] = pngBind(fname, PNG_NOMIPMAP, PNG_ALPHA,
74          &hblackinfo[i], GL_CLAMP, GL_NEAREST, GL_NEAREST);
75      }
76  }
```

4.7 画像および日本語の表示

画像および日本語をゲーム画面に表示する方法について説明する. まず, 画像を表示する方法について説明する. 画像の表示はリスト 11 に示す PutSprite 関数を用いて行っている.PutSprite 関数は「Springs of C」! から引用した関数である. リスト 11 のコードでは, 引用した関数に, 引数として画像の表示倍率 scale を加え, 画像の縮小を行えるようにした. 画像の表示倍率はリスト 11 の 6 行目および 7 行目でテクスチャの幅と高さを scale 倍することで行っている.

リスト 11: PutSprite 関数

```
1   // (x,y)に大きさscaleの画像を表示
2   void PutSprite(int num, int x, int y, pngInfo *info,double scale)
3   {
4       int w, h;  //  テクスチャの幅と高さ
5
6       w = info->Width*scale;   //  テクスチャの幅と高さを取得する
7       h = info->Height*scale;
8
9       glPushMatrix();
10      glEnable(GL_TEXTURE_2D);
11      glBindTexture(GL_TEXTURE_2D, num);
12      glColor4ub(255, 255, 255, 255);
13
14      glBegin(GL_QUADS);  //  幅w, 高さhの四角形
15
16      glTexCoord2i(0, 0);
17      glVertex2i(x, y);
18
19      glTexCoord2i(0, 1);
20      glVertex2i(x, y + h);
21
22      glTexCoord2i(1, 1);
23      glVertex2i(x + w, y + h);
24
25      glTexCoord2i(1, 0);
26      glVertex2i(x + w, y);
27
28      glEnd();
29
30      glDisable(GL_TEXTURE_2D);
31      glPopMatrix();
32  }
```

次に日本語の表示方法について説明する. リスト 12 に日本語を表示するための関数である drawChar 関数および drawString 関数のコードを示す.drawChar 関数は「1 文字」の日本語を表示する関数であり,drawString 関数は drawChar 関数を連続して呼び出して文字列を表示する関数である.drawChar 関数は引数として jpProtcol 配列 (リスト 9) のインデックス num, ひらがな/カタカナのどちらで表示するかを示す kh, 黒/赤のどちらで描画するかを示す color, 描画する実座標 (x,y), 表示倍率 scale の 6 つを受け取る. 引数 kh は 0 のときひらがな,1 のときカタカナである. 引数 color は 0 のとき黒,1 のとき赤である. drawChar 関数の内部では, 受け取った引数から表示する文字の種類を判断し,PutSprite 関数で描画する処理を行っている.

drawString 関数は引数として描画する文字列 string, 文字色 color, 実座標 (x,y), 表示倍率 scale の 5 つを

受け取る.drawString 関数の内部では, まず引数として受け取った文字列 string を 1 文字または 2 文字ずつ取り出して, 文字の種類を判断している. リスト 12 では, 29 行目および 30 行目が数字かどうかの判別を行っている部分である. 日本語の判別は 35 行目, 特殊文字は 41 行目で判定している. 例外としてひらがな/カタカナ切り替えは 47 行目, 改行は 51 行目で判定している. 空白は判定していないが, これは該当する文字がない場合に何も描画せずに文字を表示する位置がずれることを利用している. このため, 空白は「ss」以外の文字列でも表現できるがコードのわかりやすさという観点から「ss」という文字に統一している. 文字の種類の判断が行えたら,jpProtcol 配列におけるその文字のインデックスおよび描画のための情報を drawChar 関数に渡して描画を行っている. 最後に 58 行目から 63 行目で次に表示する文字の位置を計算する処理を行っている. これらの処理によって画面に日本語を描画している.

リスト 12: 日本語表示のための関数

```
1  // 1文字の日本語を表示
2  // int kh : 0,Hiragana 1,Katakana
3  // int color 0,black 1,red
4  void drawChar(int num,int kh,int color,int x,int y,double scale){
5      if(kh==0){
6          if(color==0){ // hiragana black
7              PutSprite(hblackimg[num], x, y, &hblackinfo[num],scale);
8          }else{ //hiragana red
9              PutSprite(hredimg[num], x, y, &hredinfo[num],scale);
10             }
11         }else{
12         if(color==0){ // katakana black
13             PutSprite(kblackimg[num], x, y, &kblackinfo[num],scale);
14         }else{ // katakana red
15             PutSprite(kredimg[num], x, y, &kredinfo[num],scale);
16         }
17     }
18 }
19
20 // 引数stringの文字列を表示
21 void drawString(char *string,int color,int xo,int yo,double scale){
22     int i,j;
23     int len = strlen(string);
24     int x=xo;
25     int y=yo;
26     int flg;
27     int kh=0;
28     for(i=0;i<len;i++){
29         flg=string[i]-'0'; // インデクス計算
30         if((flg>=0)&&(flg<=9)){ // 数字描画
31             drawChar(JPMAX+flg,0,color,x,y,scale);
32             flg=1;
33         }else{
34             for(j=0;j<JPMAX;j++){ //日本語描画
35                 if((jpProtcol[j][0]==string[i])&&(jpProtcol[j][1]==string[i+1])){
36                     drawChar(j,kh,color,x,y,scale);
37                     break;
38                 }
39             }
40             for(j=JPMAX+10;j<JPMAX+SPMAX;j++){ //特殊文字描画
41                 if((jpProtcol[j][0]==string[i])&&(jpProtcol[j][1]==string[i+1])){
42                     drawChar(j,kh,color,x,y,scale);
43                     break;
44                 }
45             }
46             flg=1;
47             if((string[i]=='l')&&(string[i+1]=='l')){ //ひらがな/カタカナ切り替え
48                 kh=1-kh;
49                 flg=0;
50             }
51             if((string[i]=='x')&&(string[i+1]=='x')){ // 改行
52                 x=xo;
53                 flg=0;
54                 y+=IMGSIZE*scale;
55             }
56             i++;
57         }
```

```
58          if(flg==1){ // 次の座標に移動
59              x+=IMGSIZE*scale;
60              if(x>InitWidth-22){
61                  x=xo;
62                  y+=IMGSIZE*scale;
63              }
64          }
65      }
66  }
```

4.8 駅および物件情報の読み込み

駅および物件の情報を読み込む方法について説明する. 動作確認は次節のメイン関数で行う. まず, 駅の情報を読み込む方法について説明する. 駅の情報は property.txt に保存されている. リスト 13 に property.txt の内容を示す. property.txt は「駅名 x 座標,y 座標」という形式ですべての駅の情報が保存されている. 駅名は日本語プロトコルにおける駅名の表示である. 座標は実座標ではなく, Map 配列 (リスト 8) のインデックスである. 例えば飯山駅の場合, 駅名が「iiiiyama」, Map 配列における座標が (13,3) になっている. リスト 8 の (13,3) を確認すると「B」つまり物件駅になっている. また, 図 3 からこの位置にある駅は飯山駅であることがわかる. これらより property.txt で定義した駅名および座標が,Map 配列や画面表示と合致していることが確認できた. 同様にして全ての駅について駅名と座標が間違っていないことを筆者は確認した.

リスト 13: property.txt

```
1  nozawaoonnsenn 18,1
2  iiiiyama 13,3
3  togakusi 9,9
4  nagano 13,7
5  oobuse 19,7
6  suzaka 19,11
7  matusiro 12,10
8  sinonoii 9,12
9  hakuba 3,7
10 oooomati 3,18
11 tikuma 9,15
12 uueeda 19,15
13 karuiizawa 23,15
14 aadumino 3,20
15 saku 23,18
16 matumoto 9,22
17 suwa 17,25
18 kiso 3,25
19 ookaya 9,27
20 iiiida 20,28
21 iina 23,27
```

property.txt の形式が確認できたから, これを読み込む関数について説明する. リスト 14 に駅の情報を読み込むための関数である readStation 関数のコードを示す. リスト 14 においてファイルから読み取った情報を stationstatus 構造体に代入しているのは 11 行目から 14 行目である.11 行目で fscanf 関数を用いて「駅名 x 座標,y 座標」という情報を構造体に代入している.

リスト 14: readStation 関数

```
1  // ファイルから駅情報を取得
2  // stations構造体を初期化
3  void readStation(void){
4      FILE *fp;
5      int i=0;
6      fp=fopen("property.txt","r");
7      if(fp==NULL){ // 開けなかったとき
8          printf("file not found");
9          exit(0);
10     }else{ // 駅名と座標を取得
11         while(fscanf(fp,"%s %d,%d",stations[i].name,&stations[i].x,&stations[i].y)!=EOF){
12             stations[i].ismonopoly=0; // 独占フラグ初期化
```

```
            i++;
        }
        fclose(fp);
    }
}
```

次に物件の情報を読み込む方法について説明する. 物件の情報は「/property」に「駅名.txt」という形式で保存している. 駅名は日本語プロトコルにおける駅名である. リスト 15 およびリスト 16 に物件情報を保存しているファイルの例を示す. リスト 15 は長野駅, リスト 16 は松本駅である. 物件の情報は「物件名 価格, 収益率」という形式で保存している.

リスト 15: /property/nagano.txt

```
rinngoeenn 600,120
rinngoeenn 600,120
yawatayaiisogorouu 6000,20
llaaiisusukellmslltollzilouu 20000,3
zennkouuzi 110000,30
naganokouusenn 600000,1
```

リスト 16: /property/matumoto.txt

```
giluuuniluuullpannll 1200,130
kamikouuti 12000,80
sinnsiluuudaiigaku 60000,30
aasamaoonnsenn 80000,4
kiluuukaiitigaltkouu 90000,5
matumotozilouu 150000,5
```

物件情報の保存形式が確認できたから, これを読み込む関数について説明する. リスト 17 に物件の情報を読み込む関数である readProperty 関数のコードを示す. リスト 17 においてファイルから読み取った情報を propertystatus 構造体に代入しているのは,15 行目および 16 行目である. readStation 関数と同様に fscanf 関数を用いて「物件名 価格, 収益率」という情報を構造体に代入している.

リスト 17: readProperty 関数

```
// ファイルから物件情報を取得
void readProperty(void){
    FILE *fp;
    int i,j;
    char fname[100];
    for(i=0;i<STATIONNUM;i++){
        sprintf(fname,".\\property\\%s.txt",stations[i].name);
        fp=fopen(fname,"r");
        j=0;
        if(fp==NULL){ // 開けなかったとき
            printf("file not found in %s",stations[i].name);
            exit(0);
        }else{
            // 物件名,値段,収益率を取得
            while(fscanf(fp,"%s %d,%d",stations[i].plist[j].name,
            &stations[i].plist[j].price,&stations[i].plist[j].earnings)!=EOF){
            stations[i].plist[j].holder=0; // 購入フラグ初期化
            j++;
        }
        stations[i].propertynum=j; // 物件数を保存
        fclose(fp);
    }

    }
}
```

[1] 桃太郎電鉄,https://www.konami.com/games/momotetsu/teiban/ , 閲覧日 2021 年 1 月 5 日