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　ゲームの説明

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いる. 舞台設定は長野県である. プレイヤー (社長と呼ぶ) の目的は,3 年経過時に自分の「総資産」を他の社長よりも多くすることである.

総資産を増やす方法を説明する前に, ゲームの進行について説明する. ゲームスタート時に目的地がランダムに設定され, すべての社長に「所持金」として 1 億円が配られる. 各社長は自分のターンにサイコロをふり, 出た目の数だけ好きな方向に進めことができる. 停車した駅には次に示す種類がある. マイナス駅で借金を背負った場合の処理はプログラムの説明の章で行う.

- プラス駅 ... 停車するとお金がもらえる駅
- マイナス駅 ... 停車するとお金が減る駅
- 物件駅 ... 停車すると物件を購入することができる駅
- 目的地駅 ... 特別な物件駅. 停車するとお金がもらえ, 物件を購入することができる駅

総資産は, 物件駅に停車して物件を購入することで増やすことができる. 物件駅とは図 1 に示すように, その地域の名産品や観光地を購入できる駅のことである. 図 1 の長野駅の場合,6 個の物件がある. 物件にはそれぞれ「物件名」,「価格」,「収益率」の 3 つが設定されている. 図 1 の「りんごえん」の場合は物件名が「りんごえん」, 価格が「600 万円」, 収益率が「120%」である. 所持金で購入できる物件は黒字, 購入できない物件は赤字で表示される. 図 1 では所持金で購入できる「りんごえん」, 「やわたやいそごろう」,「アイススケートじょう」が黒字,「ぜんこうじ」と「ながのこうせん」が赤字で表示されている. 同じ駅の全ての物件を購入すると「独占」と呼ばれる状態になる.

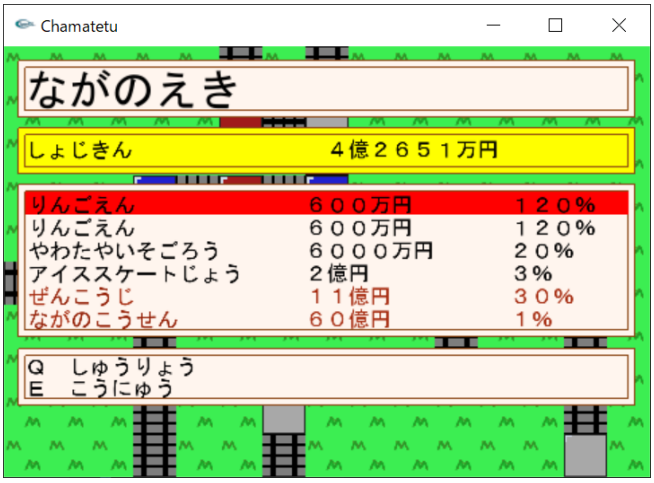

図 1: 物件の例 (長野駅)

次に年月の仕様について説明する. ゲームスタート時は「1 年目 4 月」に設定されている. プレイヤー 3 人のターンが 1 回終了すると 1ヵ月が経過し,「1 年目 3 月」が終了すると「2 年目 4 月」になる. さらに, 年月とは別に次に示すような季節の設定がある. ゲームの背景, プラス駅, マイナス駅の 3 つは季節によって変化

する. ゲームの背景は季節にあったもの (例えば冬は雪が積もる) が描画される. プラス駅は夏は億単位のお金がもらえ, 冬は百万円単位のお金しかもらえないという仕様になっている. マイナス駅は夏は百万円単位のお金しか減額されてないが, 冬は億単位で減額される仕様になっている. この仕様は本家である桃鉄を参考にした.

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

プラス駅と目的地駅でしかお金がもらえないと頻繁に金欠状態になってしまうため 3 月が終了すると「決算」が行われ, 総資産に応じたお金が手に入る. 決算でもらえるお金の計算方法について説明する. 物件を n 個持っているとき, 決算でもらえる金額 S は, 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 $d_i = 1$ とする変数を用いると式 (1) のように表せる.

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

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. 実際に計算すると式 (4) のように 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.

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

3　実行環境とビルド方法

本章では, 実行環境およびビルド方法について述べる.

3.1　実行環境

実行環境を 1 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 1: 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

!

4 プログラムと実行結果

本章では, ゲームの実装部分について説明する.

4.1 構造体の定義と初期化

本ゲームを実装するうえで, プレイヤーや物件の情報を格納する方法として構造体を用いた. 本節では次の 3 つの構造体の定義および初期化と値の格納について述べる.

1. playerstatus 構造体
2. propertystatus 構造体
3. stationstatus 構造体

4.1.1 playerstatus 構造体

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 1 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)」の 5 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. 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 1: 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]; // 社長名
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  };
9
10 typedef struct playerstatus player;
11 player players[PLAYERNUM]; // 人数分の配列を確保
```

4.1.2 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 2 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 2 のルールで扱うものとする.

リスト 2: propertystatus 構造体の定義

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

```
};

typedef struct propertystatus property;
```

表 2: 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

4.1.3 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 3 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番号である. 詳細については!で述べる. 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 2 と同じである.

リスト 3: 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
};

typedef struct stationstatus station;
station stations[STATIONNUM]; // 駅の数分の配列を確保
station distination; // 目的地配列
```

4.1.4 構造体への値の格納

構造体に値を格納するためのコードについて説明する. まず,playerstatus 構造体の初期化について説明する. リスト 4 に InitPlayer 関数および初期化を確認するための dispPlayer 関数のコードを示す. InitPlayer 関数では for 文を用いて playerstatus 構造体への値の格納を行っている.6 行目で不思議な文字列を社長名のメンバに格納しているが, これは日本語を画面出力するための独自プロトコルでの「プレイヤー 1」という表現である. 日本語プロトコルの説明については!で行う.dispPlayer 関数は playerstatus 構造体に値が格納されていることを確認するための関数である. 引数として 0 を与えるとすべての社長のメンバ情報を表示する. 引数として 1,2,3 のいずれかを与えると表 2 に対応した社長のメンバ情報だけを表示する.

リスト 4: playerstatus 構造体への値の格納

```
// 画像サイズ
#define IMGSIZE 32
// 初期プレイヤー座標
#define INITX 13*IMGSIZE
#define INITY 7*IMGSIZE
// 初期所持金
#define INITMONEY 10000
```

```
8
9  // プレイヤー構造体を初期化
10 void InitPlayer(void){
11     int i;
12     for(i=0;i<PLAYERNUM;i++){
13         //プレイヤーhoge
14         sprintf(players[i].name,"llpureiiyallms%d",i+1);
15         players[i].x=INITX;
16         players[i].y=INITY;
17         players[i].money=INITMONEY;
18         players[i].assets=0;
19     }
20 }
21
22 // デバッグ用関数
23 // プレイヤー構造体を表示
24 // detail : 0 全部表示 , else その番号の駅を表示
25 void dispPlayer(int detail){
26     int i;
27     if(detail==0){
28         for(i=0;i<PLAYERNUM;i++){
29             printf("--------------------\n");
30             printf("%s社長 (%d,%d)\n",players[i].name,players[i].x,players[i].y);
31             printf("\n");
32             printf("所持金 : %d\n",players[i].money);
33             printf("総資産 : %d\n",players[i].assets);
34             printf("--------------------\n\n");
35         }
36     }else{
37         printf("--------------------\n");
38         printf("%s社長 (%d,%d)\n",players[detail-1].name,
39         players[detail-1].x,players[detail-1].y);
40         printf("\n");
41         printf("所持金 : %d\n",players[detail-1].money);
42         printf("総資産 : %d\n",players[detail-1].assets);
43         printf("--------------------\n\n");
44     }
45 }
```

リスト 5 のコードを実行して, 構造体に値が代入されていることを確認する. 必要なヘッダファイルについては付録を参照してほしい. 正しく実行できていれば, 座標 (x,y) は (416,224), 所持金は 10000, 総資産は 0 になるはずである. リスト 6 にリスト 5 のコードの実行結果を示す. リスト 6 から, 座標, 所持金, 総資産がすべての社長について正しい値であることがわかる. これより playerstatus 構造体の定義および初期化の方法が確認できた.

リスト 5: InitPlayer 関数の動作確認

```
1  int main(int argc, char **argv){
2      InitPlayer();
3      dispPlayer(0);
4  }
```

リスト 6: リスト 5 の実行結果

```
1  --------------------
2  llpureiiyallms1社長 (416,224)
3
4  所持金 : 10000
5  総資産 : 0
6  --------------------
7
8  --------------------
9  llpureiiyallms2社長 (416,224)
10
11 所持金 : 10000
12 総資産 : 0
13 --------------------
14
15 --------------------
16 llpureiiyallms3社長 (416,224)
```

```
17
18 所持金 : 10000
19 総資産 : 0
20 --------------------
```

次に, 駅の情報および物件の情報を構造体に格納する方法について説明する. 駅の情報はリスト 7 に示す property.txt に記述されている. property.txt には各駅について「駅名」, 駅の座標 (x,y) が記述されている. 駅名は独自の日本語プロトコルで記述されている.

リスト 7: 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
```

駅の情報は property.txt をファイル読み込みによって行う. 駅の情報を読み込むための関数である readStation 関数のコードをリスト 8 に示す. リスト 8 において読み取った値を 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」という情報を読み取って構造体に代入している.

リスト 8: readStation 関数のコード

```
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
13             i++;
14         }
15         fclose(fp);
16     }
17 }
```

物件情報も同様にファイルから読み込む仕様にした. 物件情報は「/property」に「駅名.txt」という形で保存している. リスト 9 およびリスト 10 に物件情報を保存しているファイルの例を示す. リスト 9 は長野駅, リスト 10 は松本駅である. どちらの場合も, 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 9: /property/nagano.txt

```
1  rinngoeenn 600,120
2  rinngoeenn 600,120
```

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

リスト 10: /property/matumoto.txt

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

リスト 11 にファイルから物件情報を取得するための関数である readProperty 関数, および読み込めたことを確認するための dispStation 関数のコードを示す.readProperty 関数においてファイルから物件の情報を読み込んでいるのは 15 行目から 19 行目である.readStation 関数同様に,fscanf 関数を用いて一行ずつ物件情報を読み取って propertystatus 構造体に代入している.dispStation 関数の仕様も dispPlayer 関数と同様である. 引数として 0 を与えるとすべての駅の情報が標準出力され,1,2... という番号を与えると, 対応したインデックスの駅情報のみが出力される.

リスト 11: readProperty 関数と dispStation 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);
    }

    }
}

// デバッグ用関数
// 駅情報を表示
void dispStation(int detail){
    int i,j;
    if(detail==0){
        for(i=0;i<STATIONNUM;i++){
            printf("--------------------\n");
            printf("%s駅 (%d,%d)\n",stations[i].name,stations[i].x,stations[i].y);
            printf("独占フラグ : %d   物件数 : %d\n",stations[i].ismonopoly,
            stations[i].propertynum);
            for(j=0;j<stations[i].propertynum;j++){
                printf("%s %d %d %d\n",stations[i].plist[j].name,
                stations[i].plist[j].price,stations[i].plist[j].earnings,
                stations[i].plist[j].holder);
            }
            printf("--------------------\n\n");
        }
        }else{
        printf("--------------------\n");
        printf("%s駅 (%d,%d)\n",stations[detail-1].name,stations[detail-1].x,
```

```
        stations[detail-1].y);
        printf("独占フラグ : %d    物件数 : %d\n",stations[detail-1].ismonopoly,
        stations[detail-1].propertynum);
        for(j=0;j<stations[detail-1].propertynum;j++){
            printf("%s %d %d %d\n",stations[detail-1].plist[j].name,
            stations[detail-1].plist[j].price,
            stations[detail-1].plist[j].earnings,
            stations[detail-1].plist[j].holder);
        }
        printf("--------------------\n\n");
    }
}
```

リスト 12 に駅および物件の情報が正常に読み込まれ, 構造体に格納されていることを確認するコードを示す. 正しく実行できていれば, ファイルから読み取った駅および物件の情報が表示されるはずである. リスト 12 にリスト 12 のコードの実行結果を示す. 実行結果として 21 駅分の結果を載せると長くなるため, ここでは必要な駅のみを載せることにする. リスト 12 から, リスト 7 の一番目の野沢温泉駅から, 末尾の伊那駅まで全ての駅が読み込まれていることを確認した. また, 例として掲載した長野駅および松本駅について, ファイルに記述した情報と標準出力した情報に違いがないことが確認できる. これらより, ファイルから駅および物件の情報を読み込み, 構造体に格納できていることが確認できた.

リスト 12: 駅および物件読み込みの動作確認

```
int main(int argc, char **argv){
    readStation();
    readProperty();
    dispStation(0);
}
```

リスト 13: リスト 12 の実行結果

```
--------------------
nozawaoonnsenn駅 (18,1)
独占フラグ : 0    物件数 : 6
nozawanaoyaki 3000 80 0
nozawanaoyaki 3000 80 0
nozawanaoyaki 3000 80 0
ooooyu 12000 5 0
llsukillmszilouu 40000 10 0
llsukillmszilouu 40000 10 0
--------------------

(中略)

--------------------
nagano駅 (13,7)
独占フラグ : 0    物件数 : 6
rinngoeenn 600 120 0
rinngoeenn 600 120 0
yawatayaiisogorouu 6000 20 0
llaaiisusukellmslltollzilouu 20000 3 0
zennkouuzi 110000 30 0
naganokouusenn 600000 1 0
--------------------

(中略)

--------------------
matumoto駅 (9,22)
独占フラグ : 0    物件数 : 6
giluuuniluuullpannll 1200 130 0
kamikouuti 12000 80 0
sinnsiluuudaiigaku 60000 30 0
aasamaoonnsenn 80000 4 0
kiluuukaiitigaltkouu 90000 5 0
matumotozilouu 150000 5 0
--------------------

(中略)
```

```
39
40 --------------------
41 iina駅 (23,27)
42 独占フラグ : 0   物件数 : 3
43 rinngollpaiill 3000 75 0
44 bunnguiitouuge 30000 7 0
45 takatooozilouusi 130000 5 0
46 --------------------
```

4.2 メイン関数

メイン関数について説明する. リスト 14 にメイン関数のコードを示す. メイン関数の処理について説明する. リスト 14 の 12 行目および 13 行目ではウィンドウの生成を行っている. 本ゲームではウィンドウサイズは 480×320 の固定サイズとする.22 行目から 25 行目では, 画像の読み込みおよび構造体の初期化を行っている.28 行目から 31 行目ではコールバック関数の登録を行っている. これによって, キーボード入力やウィンドウのサイズ変更といったイベントが発生したときに, そのイベントに対応した関数が呼び出されて, 処理が行われる.

リスト 14: メイン関数

```
1  #include <GL/glut.h>
2  #include <GL/glpng.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <time.h>
6  #include "game.h"
7
8  int main(int argc, char **argv)
9  {
10     srand((unsigned) time(NULL));
11     glutInit(&argc, argv);
12     glutInitWindowSize(InitWidth,InitHeight);
13     glutCreateWindow("Chamatetu");
14     glutInitDisplayMode(GLUT_RGBA | GLUT_ALPHA);
15     glClearColor(1.0, 1.0, 1.0, 0.0);
16
17     //  テクスチャのアルファチャネルを有効にする設定
18     glEnable(GL_BLEND);
19     glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
20     glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
21
22     readImg();
23     InitPlayer();
24     readStation();
25     readProperty();
26     turnstatus=0;
27     //イベント登録
28     glutReshapeFunc(Reshape);
29     glutDisplayFunc(Display);
30     glutKeyboardFunc(keyboard);
31     glutTimerFunc(RESHAPETIME, Timer, 0);
32     //  イベントループ突入
33     glutMainLoop();
34
35     return(0);
36 }
```

4.3 日本語の表示方法

本節では日本語を表示する方法について, 次に示す内容を述べる. !

4.3.1 日本語プロトコルの説明

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. 文字色について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 2 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 3 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 3 の情報は game.c でリスト 15 に示すように定義されている.

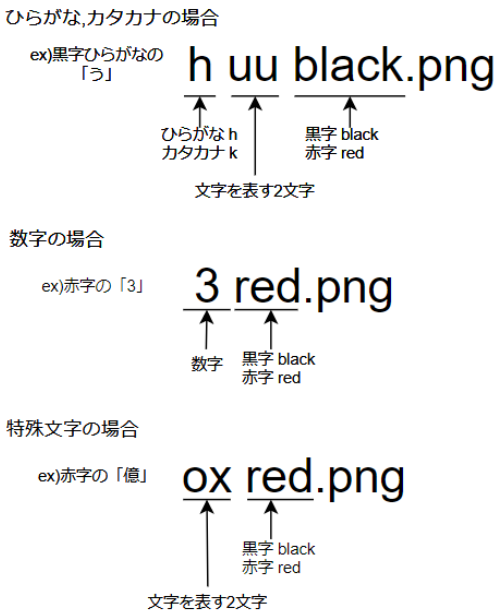

図 2: 画像の名前の意味

表 3: 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 15: 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"
                                  };
```

4.3.2 画像を読み込む方法

画像の読み込ム部分のコードをリスト 16 に示す. 画像の情報を格納する配列 (リスト 16 の 3 行目から 30 行目) は game.h に定義してある. readImg 関数はリスト 16 の 33 行目から 107 行目である. readImg 関数

では日本語プロトコルのため画像, イベント用の画像, 季節別背景の画像, マップの画像, プレイヤーの画像, サイコロの画像を読み込んでいる. 画像名は sprintf 関数を用いて指定し, 画像を pngBind 関数で読み込んでいる.

リスト 16: readImg 関数

```
// 画像用変数
// 季節画像
GLuint seasonimg[SEASON_NUM];
pngInfo seasoninfo[SEASON_NUM];
// マップ画像
GLuint mapimg[MAP_NUM];
pngInfo mapinfo[MAP_NUM];
// プレイヤー画像
GLuint playerimg[PLAYERNUM];
pngInfo playerinfo[PLAYERNUM];
// サイコロ画像
GLuint diceimg[DICEMAX];
pngInfo diceinfo[DICEMAX];
// イベント画像
GLuint spimg[SP_NUM];
pngInfo spinfo[SP_NUM];

// 日本語画像
// ひらがな黒
GLuint hblackimg[JPMAX+SPMAX];
pngInfo hblackinfo[JPMAX+SPMAX];
// ひらがな赤
GLuint hredimg[JPMAX+SPMAX];
pngInfo hredinfo[JPMAX+SPMAX];
// カタカナ黒
GLuint kblackimg[JPMAX];
pngInfo kblackinfo[JPMAX];
// カタカナ赤
GLuint kredimg[JPMAX];
pngInfo kredinfo[JPMAX];

// 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,
```

```
68          &diceinfo[i], GL_CLAMP, GL_NEAREST, GL_NEAREST);
69      }
70      // read Hiragana black
71      for(i=0;i<JPMAX;i++){
72          sprintf(fname,".\\charimg\\h%sblack.png",jpProtcol[i]);
73          hblackimg[i] = pngBind(fname, PNG_NOMIPMAP, PNG_ALPHA,
74          &hblackinfo[i], GL_CLAMP, GL_NEAREST, GL_NEAREST);
75      }
76
77      // read Hiragana red
78      for(i=0;i<JPMAX;i++){
79          sprintf(fname,".\\charimg\\h%sred.png",jpProtcol[i]);
80          hredimg[i] = pngBind(fname, PNG_NOMIPMAP, PNG_ALPHA,
81          &hredinfo[i], GL_CLAMP, GL_NEAREST, GL_NEAREST);
82      }
83      // read Katakana black
84      for(i=0;i<JPMAX;i++){
85          sprintf(fname,".\\charimg\\k%sblack.png",jpProtcol[i]);
86          kblackimg[i] = pngBind(fname, PNG_NOMIPMAP, PNG_ALPHA,
87          &kblackinfo[i], GL_CLAMP, GL_NEAREST, GL_NEAREST);
88      }
89      // read Katakana red
90      for(i=0;i<JPMAX;i++){
91          sprintf(fname,".\\charimg\\k%sred.png",jpProtcol[i]);
92          kredimg[i] = pngBind(fname, PNG_NOMIPMAP, PNG_ALPHA,
93          &kredinfo[i], GL_CLAMP, GL_NEAREST, GL_NEAREST);
94      }
95      // read Special Str red
96      for(i=JPMAX;i<JPMAX+SPMAX;i++){
97          sprintf(fname,".\\charimg\\%sred.png",jpProtcol[i]);
98          hredimg[i] = pngBind(fname, PNG_NOMIPMAP, PNG_ALPHA,
99          &hredinfo[i], GL_CLAMP, GL_NEAREST, GL_NEAREST);
100     }
101     // read Special Str black
102     for(i=JPMAX;i<JPMAX+SPMAX;i++){
103         sprintf(fname,".\\charimg\\%sblack.png",jpProtcol[i]);
104         hblackimg[i] = pngBind(fname, PNG_NOMIPMAP, PNG_ALPHA,
105         &hblackinfo[i], GL_CLAMP, GL_NEAREST, GL_NEAREST);
106     }
107 }
```

4.3.3 画像を表示する関数

画像を表示するための関数として PutSprite 関数がある. この関数は授業中に使用した関数であるが, 画像のサイズを変更できるように改造した. リスト 17 に PutSprite 関数のコードを示す.PutSprite 関数は引数として受け取った座標 (x,y) に倍率 scale で画像を描画する関数である.

リスト 17: 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  }
```

4.3.4 文字列による画像の表示制御

1 文字を描画する関数として,drawChar 関数を作成した. リスト 18 に drawChar 関数のコードを示す. drawChar 関数は引数として表示する文字の jpProtcol 配列のインデックス num, ひらがな/カタカナのどちらで描画するかを判別する変数 kh, 黒/赤のどちらで描画するかを判別する変数 color, 描画する座標 (x,y), 表示する大きさ scale を受け取る. 変数 kh はひらがなのとき 0, カタカナのとき 1 である. 変数 color は黒のとき 0, 赤のとき 1 である. 変数スケールは 1 のとき 32×32px で描画される.

リスト 18: drawChar 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  }
```

1 文字を描画するたびに drawChar 関数を呼び出すと大変だから, 文字列を引数として渡すと自動で描画が行われる関数を作成した. リスト 19 に drawString 関数のコードを示す. この関数を用いて実際に描画を行う例については,!節以降で説明する. drawString 関数は引数として受け取った文字列 string を描画する関数である. 文字列の他に引数として描画する色 color,1 文字目を表示する座標 (xo,yo), 表示する大きさ scale を受け取る. 座標 (xo,yo) は 1 文字目の左上の座標である. drawString 関数の内部では,for 文を用いて文字列 string の文字を 1 文字, または 2 文字ずつ取り出し, その文字が数字, 日本語, 特殊文字, スペース, 改行のどれにあたるかチェックしている. 該当する文字が発見された場合は drawChar 関数に配列の番号と描画位置の情報を渡して描画を行っている. 1 文字描画したら, 次の文字を描画するために, 描画する座標を計算する. これはリスト 19 の 40 行目から 43 行目で行っている.

リスト 19: drawString 関数

```
1   // 引数stringの文字列を表示
2   void drawString(char *string,int color,int xo,int yo,double scale){
3       int i,j;
4       int len = strlen(string);
5       int x=xo;
6       int y=yo;
7       int flg;
8       int kh=0;
9       for(i=0;i<len;i++){
```

```
10          flg=string[i]-'0'; // インデクス計算
11          if((flg>=0)&&(flg<=9)){ // 数字描画
12              drawChar(JPMAX+flg,0,color,x,y,scale);
13              flg=1;
14          }else{
15              for(j=0;j<JPMAX;j++){ //日本語描画
16                  if((jpProtcol[j][0]==string[i])&&(jpProtcol[j][1]==string[i+1])){
17                      drawChar(j,kh,color,x,y,scale);
18                      break;
19                  }
20              }
21              for(j=JPMAX+10;j<JPMAX+SPMAX;j++){ //特殊文字描画
22                  if((jpProtcol[j][0]==string[i])&&(jpProtcol[j][1]==string[i+1])){
23                      drawChar(j,kh,color,x,y,scale);
24                      break;
25                  }
26              }
27              flg=1;
28              if((string[i]=='l')&&(string[i+1]=='l')){ //ひらがな/カタカナ切り替え
29                  kh=1-kh;
30                  flg=0;
31              }
32              if((string[i]=='x')&&(string[i+1]=='x')){ // 改行
33                  x=xo;
34                  flg=0;
35                  y+=IMGSIZE*scale;
36              }
37              i++;
38          }
39          if(flg==1){ // 次の座標に移動
40              x+=IMGSIZE*scale;
41              if(x>InitWidth-22){
42                  x=xo;
43                  y+=IMGSIZE*scale;
44              }
45          }
46      }
47  }
```

4.4 Display 関数によるゲームの状況制御

ゲームの進行処理は一括して Display 関数で行っている.Display 関数のコードは行数があるため, 付録に掲載する. ゲームの進行状況には turnstatus と inflg という 2 つの変数を用いている. 変数 turnstatus は今どの処理を行っているのかを大きな区分で表すための変数である. 変数 inflg は各 turnstatus の中で, 詳細な処理 (例えばダイアログを表示する, タイマーを起動する) を行うための変数である. ゲームの進行状況と turnstatus の値の関係を表 4 に示す.turnstatus の 10〜14 が欠番になっているのは, 開発時間の都合上作成できなかったカード処理のために予約しておいた番号のためである. それぞれの turnstatus での処理は次節以降で説明する.turnstatus はメイン関数 (リスト 14) の 26 行目で 0 が代入されてメインループが開始される. このため, ゲーム開始時に turnstatus が 0, つまり必要な変数の初期化が行われ, ゲームが開始される.

表 4: 変数 turnstatus の意味

値	進行状況
0	必要な変数の初期化
1	タイトルおよびゲームルールの表示
2	目的地の設定
3	ターンのはじめの処理
4	サイコロをふる処理
5	マス移動の処理
6	停車した駅の判別と処理の分岐
7	物件購入処理
8	プラス駅の処理
9	マイナス駅の処理
15	ターン終了処理
16	決算処理
17	最終成績およびゲーム終了処理

Display 関数は, メイン関数 (リスト 14) の 29 行目でコールバック関数として登録している. さらに,31 行目で Display 関数を呼び出すためのタイマーもコールバック関数として登録している. リスト 20 に Timer 関数のコードを示す.Timer 関数は呼び出されると glutPostRedisplay 関数を実行する. これによって Display 関数が呼び出される. glutPostRedisplay 関数を実行して処理を終了すると一度きりの実行になってしまうため,8 行目でタイマーをセットしている. これによって 100[ms] おきに Timer 関数が呼び出され, 画面描画が更新される.

リスト 20: Timer 関数

```
1 // 再描画タイマー秒数
2 #define RESHAPETIME 100
3
4 // 画面更新用タイマー
5 void Timer(int t)
6 {
7     glutPostRedisplay();
8     glutTimerFunc(RESHAPETIME, Timer, 0);
9 }
```

4.5 ゲームの起動とルール説明

ゲームの起動とルール説明について説明する.Display 関数では turnstatus=0 および 1 のときの処理である. まず, 実際の動作を確認する. ゲームを起動すると図 3 に示す画面が表示される. 図 3 がタイトル画面である. 図 3 の状態で E キーを押すとルール説明の画面に進む.

図 3: タイトル画面

図 4〜図 7 にルール説明画面を示す. タイトル画面で E キーを押すとはじめに図 4 の画面が表示される. 図 4 の状態で E キーを押すと図 5 の画面が表示される. 図 6 の場合も同様である. 図 7 の場合で E キーを押すまたは,S キーを押した場合, 次節で示す目的地の設定が行われる.

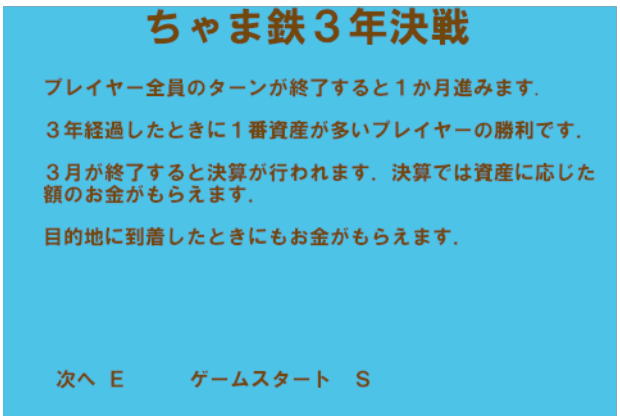

図 4: ルール説明 1

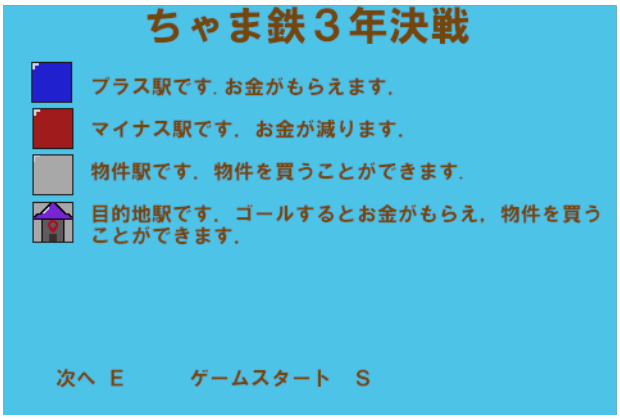

図 5: ルール説明 2

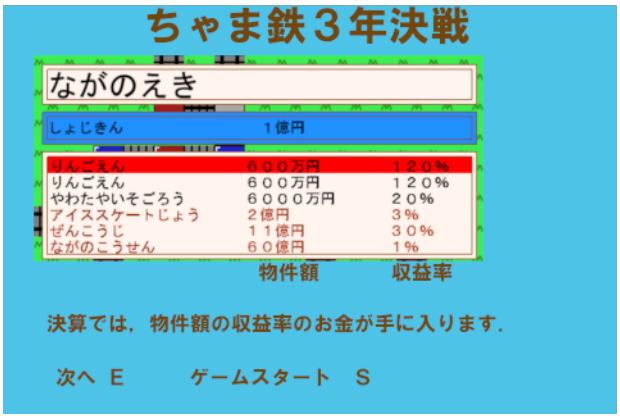

図 6: ルール説明 3

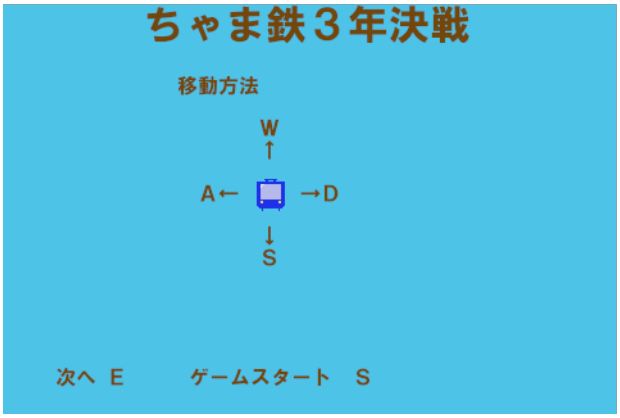

図 7: ルール説明 4

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