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. 社長はゲームの内では, 図 1 の画像で表示される. 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. マップ上で表示される社長の画像のこの色の設定になっている.

図 1: 社長のアイコン

- 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 2 に長野駅の物件の例を示す. 図 2 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.

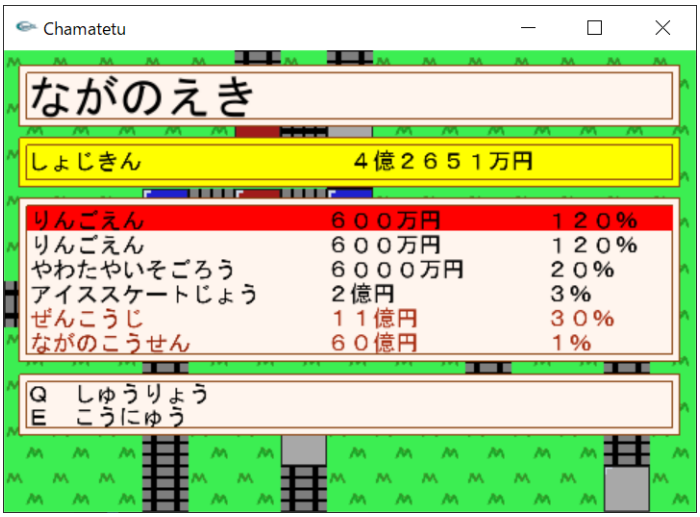

図 2: 物件の例 (長野駅)

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 3 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.

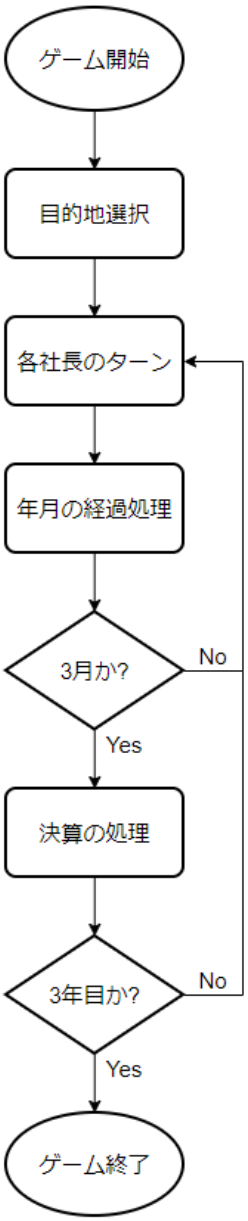

図 3: ゲームの進行

2.4 マップの設定

マップの設定について説明する. 図 4 に本ゲームのマップを示す. 図 4 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.

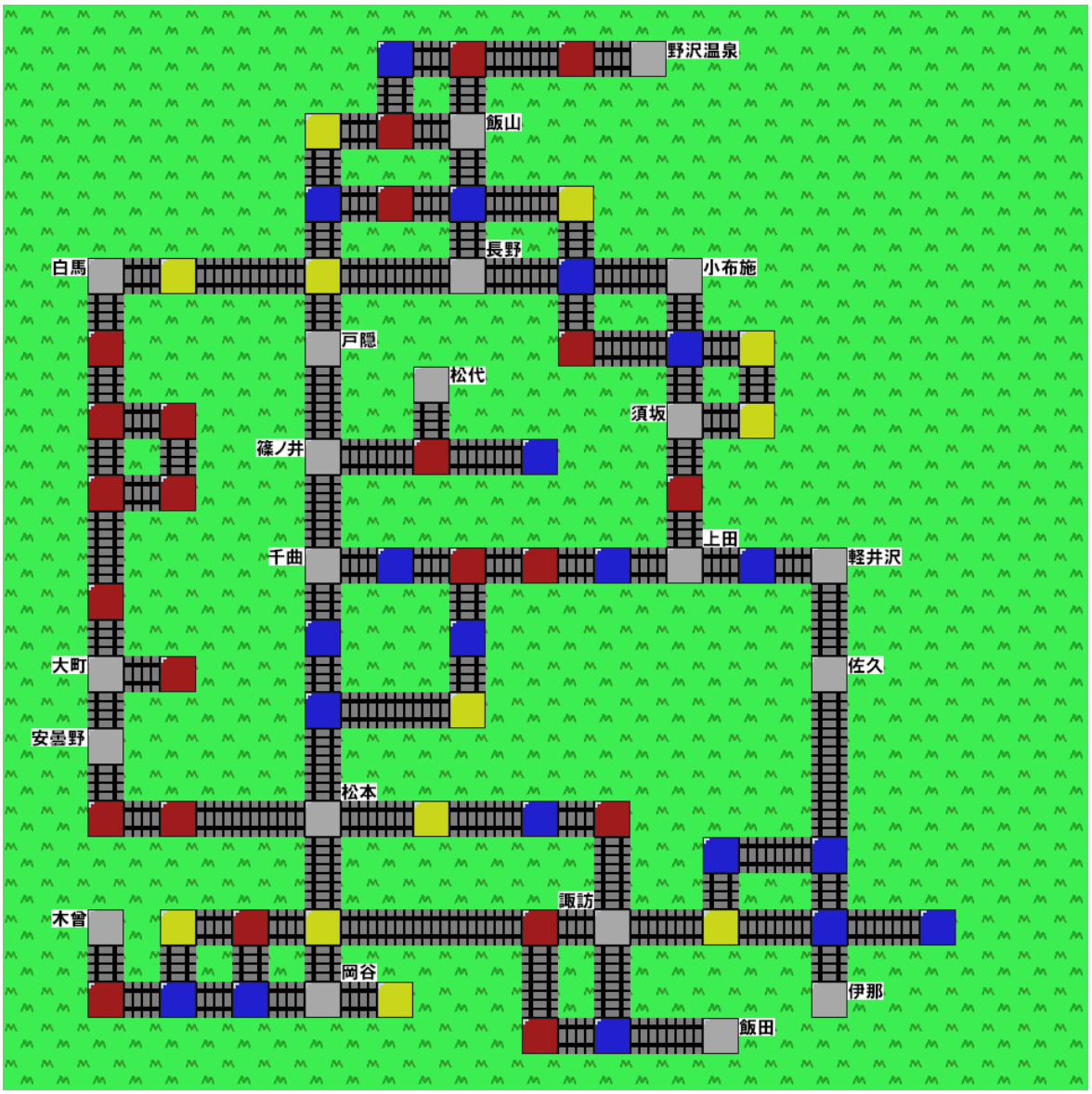

図 4: ゲームのマップ

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 4 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 2 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 5 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 5: 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. ウィンドウサイズ変更への対応 (Reshape 関数)
11. ゲームの進行状況管理
12. キーボード入力の処理
13. ゲームの初期化とタイトル画面の表示

14. 目的地の設定処理

15. プレイヤーおよびマップの描画処理

16. ターンのはじめの処理

17. サイコロをふる処理

18. マス移動および停車駅の判定処理

19. 物件駅の処理

20. プラス駅の処理

21. マイナス駅および借金の処理

22. カード駅の処理

23. ターン終了時の処理

24. 決算および最終成績の処理

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 416(13×32),INITY が 224(7×32),INITMONEY は 10000 である. 座標 (13,7) は Map 配列における長野駅の座標である.playerstatus 構造体の座標 (x,y) は画面に描画する実座標を保持する仕様にしているため画像サイズである 32 倍している.

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;
```

```
    for(i=0;i<PLAYERNUM;i++){
        //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
};

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

4.4 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

```
// マップ配列
char Map[YMAX][XMAX+1] = { //NULL文字に気を付ける
       //0123456789012345678901234567890
        "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA", // 0
        "AAAAAAAAAAP-M--M-BAAAAAAAAAAA", // 1
        "AAAAAAAAAA|A|AAAAAAAAAAAAAAA", // 2
        "AAAAAAAAAC-M-BAAAAAAAAAAAAAAA", // 3
        "AAAAAAAAA|AAA|AAAAAAAAAAAAAAA", // 4
        "AAAAAAAAAP-M-P--CAAAAAAAAAAAAA", // 5
        "AAAAAAAAA|AAA|AA|AAAAAAAAAAAA", // 6
        "AAAB-C---C---B--P--BAAAAAAAAAA", // 7
        "AAA|AAAAA|AAAAAA|AA|AAAAAAAAA", // 8
        "AAAMAAAAABAAAAAAM--P-CAAAAAAAA", // 9
        "AAA|AAAAA|AABAAAAAA|A|AAAAAAAA", // 0
        "AAAM-MAAA|AA|AAAAAAB-CAAAAAAAA", // 1
        "AAA|A|AAAB--M--PAAA|AAAAAAAAA", // 2
        "AAAM-MAAA|AAAAAAAAAMAAAAAAAAA", // 3
        "AAA|AAAAA|AAAAAAAAA|AAAAAAAAA", // 4
        "AAA|AAAAAB-P-M-M-P-B-P-BAAAAAA", // 5
        "AAAMAAAAA|AAA|AAAAAAAAA|AAAAAA", // 6
        "AAA|AAAAAPAAAPAAAAAAAAA|AAAAAA", // 7
        "AAAB-MAAA|AAA|AAAAAAAAABAAAAAA", // 8
        "AAA|AAAAAP---CAAAAAAAAA|AAAAAA", // 9
        "AAABAAAAA|AAAAAAAAAAAA|AAAAAA", // 0
        "AAA|AAAAA|AAAAAAAAAAAA|AAAAAA", // 1
        "AAAM-M---B--C--P-MAAAAA|AAAAAA", // 2
        "AAAAAAAAA|AAAAAAA|AAP--PAAAAAA", // 3
        "AAAAAAAAA|AAAAAAA|AA|AA|AAAAAA", // 4
        "AAABAC-M-C-----M-B--C--P--PAAA", // 5
        "AAA|A|A|A|AAAAA|A|AAAAA|AAAAAA", // 6
        "AAAM-P-P-B-CAAA|A|AAAAABAAAAAA", // 7
        "AAAAAAAAAAAAAAM-P--BAAAAAAAAA", // 8
        "AAAAAAAAAAAAAAAAAAAAAAAAAAAAA"  // 9
};
```

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 6 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.

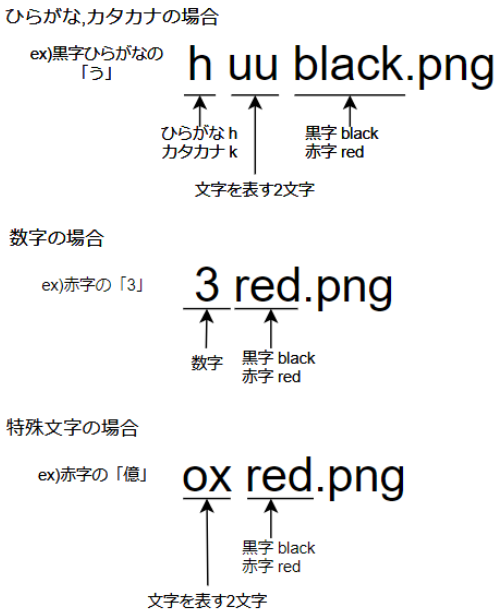

図 6: 画像の名前の意味

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  }
```

日本語の中でも, 金額だけを表示したい場合がある. このため, 引数として金額を与えると画面に表示するを作成した. リスト 13 に金額を画面に表示する関数である drawMoney 関数のコードを示す.drawMoney 関数は引数として受け取った金額 money を座標 (x,y) に表示する関数である. 引数 color で色,scale で大きさの指定を行うことも可能である. 金額の表示方法について説明する. まず, リスト 13 の 6 行目および 7 行目で, 引数として受け取った金額 money を億, 万の単位に分割する処理を行っている. 金額は負のときで符号を表示する必要があるから, 符号によって処理を変える必要がある. 8 行目から 17 行目が金額が 0 より大きいとき,18 行目から 31 行目が金額が負のときの処理である. 金額が正のとき, 文字列 fname に日本語プロトコルにおける「hoge 億 huga 万」と表示されるように文字列を代入している. 金額が負のときは「ms」を文字列の先頭につけることでマイナスを表す符号「-」が表示されるように処理している.

リスト 13: drawMoney 関数

```
1   // 数字を描画
2   void drawMoney(int money,int x,int y,int color,double scale){
3       char fname[50];
4       int oku,man;
5       // 億の桁,万の桁を計算
6       oku = money/10000;
7       man = money%10000;
8       if(money>=0){ // お金がプラスの時
9           if(oku!=0){
10              if(man!=0){
11                  sprintf(fname,"%dox%dmxex",oku,man);
12              }else{
13                  sprintf(fname,"%doxex",oku);
14              }
15          }else{
16              sprintf(fname,"%dmxex",man);
17          }
18      }else{ // お金がマイナスの時
19      // 数字部分の符号を反転
20          oku*=-1;
21          man*=-1;
22          if(oku!=0){
23              if(man!=0){
24                  sprintf(fname,"ms%dox%dmxex",oku,man);
25              }else{
26                  sprintf(fname,"ms%doxex",oku);
27              }
28          }else{
29              sprintf(fname,"ms%dmxex",man);
30          }
31      }
32      // 画面出力
33      drawString(fname,color,x,y,scale);
34  }
```

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 14 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 4 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 14: property.txt

```
nozawaoonnsenn 18,1
iiiiyama 13,3
togakusi 9,9
nagano 13,7
oobuse 19,7
suzaka 19,11
matusiro 12,10
sinonoii 9,12
hakuba 3,7
oooomati 3,18
tikuma 9,15
uueeda 19,15
karuiizawa 23,15
aadumino 3,20
saku 23,18
matumoto 9,22
suwa 17,25
kiso 3,25
ookaya 9,27
iiiida 20,28
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 15 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 15 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 15: readStation 関数

```
// ファイルから駅情報を取得
// stations構造体を初期化
void readStation(void){
    FILE *fp;
    int i=0;
    fp=fopen("property.txt","r");
    if(fp==NULL){ // 開けなかったとき
        printf("file not found");
        exit(0);
    }else{ // 駅名と座標を取得
        while(fscanf(fp,"%s %d,%d",stations[i].name,&stations[i].x,&stations[i].y)!=EOF){
            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 16 およびリスト 17 に物件情報を保存しているファイルの例を示す. リスト 16 は長野駅, リスト 17 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 16: /property/nagano.txt

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

リスト 17: /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 18 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 18 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 18: 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);
    }

    }
}
```

4.9 メイン関数 (j17406.c)

メイン関数のコードについて説明する. リスト 19 にメイン関数のコードを示す. メイン関数ではウィンドウの生成, 画像の読み込み, 構造体の初期化を行う関数の実行, コールバック関数の登録, メインループ呼び出しの 5 つの処理を行っている. ウィンドウの生成は 12 行目および 13 行目で行っている.12 行目で生成するウィンドウのサイズを指定し,13 行目でウィンドウを生成する処理を行っている. 画像の読み込みの関数である readImg 関数は 23 行目で実行している. 構造体の初期化は 24 行目から 26 行目で行っている.

構造体の初期化が正確に行われていることを確認するために, デバッグ用の関数として dispPlayer 関数および dispStation 関数を作成した. リスト 20 に dispPlayer 関数および dispStation 関数の定義を示す. これらの関数はリスト 19 の 27 行目および 28 行目で実行している. 実際に提出したファイルではコメントアウトしているため実行はされていない.dispPlayer 関数は社長の情報を保持する playerstatus 構造体内容をすべてコンソール出力する関数である. dispPlayer 関数は引数として表示する範囲を指定できる変数 detail がある. 引数として 0 を与えるとすべての社長の情報を表示し, 1,2,3 のいずれかを与えると表 5 に対応する社長の情報のみが表示される. ここでは引数として 0 を与え, すべての社長の情報を表示する機能のみを用いる. 同様に,dispStation 関数は駅および物件の情報を保持する stationstatus 構造体および propertystatus 構造体の内容をすべて表示する関数である.dispStation 関数も引数として表示する範囲を指定できる変数 detail を持っている. 引数として 0 を与えた場合は全ての駅の情報を表示し,1 以上の整数を与えた場合は, 対応するインデックスの情報のみが表示される仕様になっている. インデックスと駅の対応は, リスト 14 の 1 行目

の野沢温泉駅を 0,2 行目の飯山駅を 1 ··· というふうになっている.
リスト 19 の 32 行目から 35 行目ではコールバック関数の登録を行っている. コールバック関数とはキーボード入力や画面のリサイズを代表とするイベントが発生したときに実行される関数のことである. 最後に 37 行目で glutMainLoop 関数を実行することで, メインループを呼び出す処理を行っている.

リスト 19: main 関数

```
#include <GL/glut.h>
#include <GL/glpng.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "game.h"

int main(int argc, char **argv)
{
    srand((unsigned) time(NULL));
    glutInit(&argc, argv);
    glutInitWindowSize(InitWidth,InitHeight);
    glutCreateWindow("Chamatetu");
    glutInitDisplayMode(GLUT_RGBA | GLUT_ALPHA);
    glClearColor(1.0, 1.0, 1.0, 0.0);

    //  テクスチャのアルファチャネルを有効にする設定
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

    // 画像読み込みと構造体の初期化
    readImg();
    InitPlayer();
    readStation();
    readProperty();
    dispPlayer(0); // デバッグ用
    dispStation(0); // デバッグ用
    turnstatus=0;
    inflg=0; // 進行状況を初期化
    //イベント登録
    glutReshapeFunc(Reshape);
    glutDisplayFunc(Display);
    glutKeyboardFunc(keyboard);
    glutTimerFunc(RESHAPETIME, Timer, 0);
    //  イベントループ突入
    glutMainLoop();

    return(0);
}
```

リスト 20: 構造体の初期化を確認するための関数

```
// デバッグ用関数
// プレイヤー構造体を表示
// detail : 0 全部表示 , else その番号の駅を表示
void dispPlayer(int detail){
    int i,j;
    if(detail==0){
        for(i=0;i<PLAYERNUM;i++){
            printf("--------------------\n");
            printf("%s社長 (%d,%d)\n",players[i].name,
            players[i].x,players[i].y);
            printf("\n");
            printf("所持金 : %d\n",players[i].money);
            printf("総資産 : %d\n",players[i].assets);
            printf("カード枚数 : %d\n",players[i].cardnum);
            for(j=0;j<CARDMAX;j++){
                printf("%d ",players[i].card[j]);
            }
            printf("\n--------------------\n\n");
        }
    }else{
        printf("--------------------\n");
```

```
22          printf("%s社長 (%d,%d)\n",players[detail-1].name,
23          players[detail-1].x,players[detail-1].y);
24          printf("\n");
25          printf("所持金 : %d\n",players[detail-1].money);
26          printf("総資産 : %d\n",players[detail-1].assets);
27          printf("カード枚数 : %d\n",players[detail-1].cardnum);
28          for(j=0;j<CARDMAX;j++){
29              printf("%d ",players[detail-1].card[j]);
30          }
31          printf("\n--------------------\n\n");
32      }
33  }
34
35  // デバッグ用関数
36  // 駅情報を表示
37  void dispStation(int detail){
38      int i,j;
39      if(detail==0){
40          for(i=0;i<STATIONNUM;i++){
41              printf("--------------------\n");
42              printf("%s駅 (%d,%d)\n",stations[i].name,
43              stations[i].x,stations[i].y);
44              printf("独占フラグ : %d   物件数 : %d\n",stations[i].ismonopoly,
45              stations[i].propertynum);
46              for(j=0;j<stations[i].propertynum;j++){
47                  printf("%s %d %d %d\n",stations[i].plist[j].name,
48                  stations[i].plist[j].price,stations[i].plist[j].earnings,
49                  stations[i].plist[j].holder);
50              }
51              printf("--------------------\n\n");
52          }
53          }else{
54          printf("--------------------\n");
55          printf("%s駅 (%d,%d)\n",stations[detail-1].name,stations[detail-1].x,
56          stations[detail-1].y);
57          printf("独占フラグ : %d   物件数 : %d\n",stations[detail-1].ismonopoly,
58          stations[detail-1].propertynum);
59          for(j=0;j<stations[detail-1].propertynum;j++){
60              printf("%s %d %d %d\n",stations[detail-1].plist[j].name,
61              stations[detail-1].plist[j].price,stations[detail-1].plist[j].earnings,
62              stations[detail-1].plist[j].holder);
63          }
64          printf("--------------------\n\n");
65      }
66  }
```

構造体の初期化が正確に行われていることを確認する. まず,playerstatus 構造体について確認する. 初期化が正しく行えていれば, 座標は (416,224), 所持金は 10000, それ以外のメンバには 0 が代入されているはずである. リスト 21 にリスト 19 を実行したときのコンソール画面を示す. リスト 21 から全ての社長についてメンバに代入されている値が正しいことが確認できる. これより InitPlayer 関数で playerstatus 構造体の初期化が正しく行えていることが確認できた.

リスト 21: dispPlayer 関数の実行結果

```
1   --------------------
2   llpureiiyallms1社長 (416,224)
3
4   所持金 : 10000
5   総資産 : 0
6   カード枚数 : 0
7   0 0 0 0 0
8   --------------------
9
10  --------------------
11  llpureiiyallms2社長 (416,224)
12
13  所持金 : 10000
14  総資産 : 0
15  カード枚数 : 0
16  0 0 0 0 0
17  --------------------
```

```
18
19 --------------------
20 llpureiiyallms3社長 (416,224)
21
22 所持金 : 10000
23 総資産 : 0
24 カード枚数 : 0
25 0 0 0 0 0
26 --------------------
```

次に stationstatus 構造体および propertystatus 構造体の初期化が正確に行えていることを確認する. リスト 22 にリスト 19 における dispStation 関数の実行結果を示す. 実行結果は行数があるためここでは野沢温泉駅, 長野駅, 伊那駅の 3 つ駅における結果を表示している. また, 実行結果が正しいかどうかは長野駅の場合について確認する. リスト 22 でにおける長野県の情報はリスト 14 およびリスト 16 の長野駅の情報と一致している. このことから readStation 関数および readProperty 関数を用いて長野駅の情報を構造体に正確に代入できていることが確認できる. 他の駅について筆者が読み込んだ内容と元ファイルの内容が一致していること確認した. これらより, 構造体の初期化が正しく行えていることが確認できた.

リスト 22: dispStation 関数の実行結果

```
1  --------------------
2  nozawaoonnsenn駅 (18,1)
3  独占フラグ : 0   物件数 : 6
4  nozawanaoyaki 3000 80 0
5  nozawanaoyaki 3000 80 0
6  nozawanaoyaki 3000 80 0
7  ooooyu 12000 5 0
8  llsukillmszilouu 40000 10 0
9  llsukillmszilouu 40000 10 0
10 --------------------
11
12 ～省略～
13
14 --------------------
15 nagano駅 (13,7)
16 独占フラグ : 0   物件数 : 6
17 rinngoeenn 600 120 0
18 rinngoeenn 600 120 0
19 yawatayaiisogorouu 6000 20 0
20 llaaiisusukellmslltollzilouu 20000 3 0
21 zennkouuzi 110000 30 0
22 naganokouusenn 600000 1 0
23 --------------------
24
25 ～省略～
26
27 --------------------
28 iina駅 (23,27)
29 独占フラグ : 0   物件数 : 3
30 rinngollpaiill 3000 75 0
31 bunnguiitouuge 30000 7 0
32 takatooozilouusi 130000 5 0
33 --------------------
```

4.10 ウィンドウサイズ変更への対応 (Reshape 関数)

ウィンドウサイズの変更に対する対応について説明する. 本ゲームではウィンドウサイズは固定する仕様にしている. この処理を行うのが Reshape 関数である. リスト 23 に Reshape 関数のコードを示す. ウィンドウサイズを固定する処理は 11 行目で行っている.

リスト 23: Reshape 関数

```
1 // ウィンドウサイズ変更時の処理
2 void Reshape(int w, int h)
3 {
```

```
4       glViewport(0, 0, w, h);
5       glMatrixMode(GL_MODELVIEW);
6       glLoadIdentity();
7       gluOrtho2D(0, w, 0, h);
8       glScaled(1, -1, 1);
9       glTranslated(0, -h, 0);
10      //windowサイズ固定
11      glutReshapeWindow(InitWidth,InitHeight);
12  }
```

4.11 ゲームの進行状況管理

ゲームの進行状況管理について説明する. ゲームの進行状況は turnstatus,inflg というグローバル変数を用いて行っている.turnstatus はサイコロをふる, 移動をするという進行状況の大きな変化を管理するための変数である.inflg はサイコロをふる, サイコロをとめる, 結果のダイアログを表示する, という細かい進行状況を管理する変数である. ここでは turnstatus 変数について説明する.

表 8 に turnstatus 変数の値と進行状況を示す. turnstatus 変数の値による処理の分岐は Display 関数で行っている. リスト 24 に Display 関数のコードを示す.Display 関数の処理について説明する.3 行目および 4 行目ではターン中の社長を画面の中心に描画するための計算を行っている. この計算の意味については!で述べる.6 行目から 34 行目は turnstatus 変数の値ごとに呼び出す関数を分け, 表 8 に示した進行状況別に関数を呼び出す処理を行っている.turnstatus=3 および 11〜14 は未使用である理由は, 追加機能実装のための余地を残しているためである. 呼び出している関数の内容については次節以降で述べる.

リスト 24: Display 関数

```
1   // ディスプレイ関数
2   void Display(void){
3       tx = players[turn].x/IMGSIZE-CX; // 中央座標計算
4       ty = players[turn].y/IMGSIZE-CY;
5       glClear(GL_COLOR_BUFFER_BIT); // 描画クリア
6       if(turnstatus==0){ // ゲーム初期化処理
7           startgame();
8       }else if(turnstatus==1){ // 目的地設定
9           desisionDist();
10      }else if(turnstatus==2){ // ターンのはじめ
11          startTurn();
12      }else if(turnstatus==3){ // 予備
13          turnstatus++;
14      }else if(turnstatus==4){ // サイコロをふる処理
15          rollDice();
16      }else if(turnstatus==5){ // マス移動
17          moveMass();
18      }else if(turnstatus==6){ // 条件分岐
19          checkMass();
20      }else if(turnstatus==7){ // 物件購入処理
21          purchaseProperty();
22      }else if(turnstatus==8){ // プラス駅の処理
23          plusMass();
24      }else if(turnstatus==9){ // マイナス駅の処理
25          minusMass();
26      }else if(turnstatus==10){ // カード駅の処理
27          cardMass();
28      }else if(turnstatus==15){ // 月別分岐
29          endTurn();
30      }else if(turnstatus==16){ // 決算月
31          processKessan();
32      }else if(turnstatus==17){ // 最終成績
33          endgame();
34      }
35      glFlush();
36  }
```

表 8: 変数 turnstatus の意味

値	進行状況
0	必要な変数の初期化とタイトルの表示
1	目的地の設定
2	ターンのはじめの処理
3	未使用
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
10	カード駅の処理
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

4.12 ゲームの初期化とタイトル画面の表示

ゲームの初期化とタイトル画面の表示について説明する. この処理は turnstatus が 0 のときの処理である. メイン関数 (リスト 19) の 29 行目および 30 行目で, グローバル変数 turnstatus および inflg に 0 が代入されている. このため, メインループが始めると,Display 関数 (リスト 24) から startgame 関数がはじめに呼び出される.

リスト 25 に startgame 関数のコードを示す. startgame 関数の処理について説明する. inflg=0 のとき, グローバル変数の初期化を行って,inflg を 1 にする処理を行っている. グローバル変数 month および year は年月管理を行うための変数であり, ゲームスタート時に「1 年目 4 月」となる仕様はここで実装している. グローバル変数 turn は現在誰のターンかを管理するための変数である. グローバル変数 turn の値とターンの関係は表 9 の通りである. 最初に行動する社長は「プレイヤー 1 社長」だから, 変数 turn には 0 を代入している. グローバル変数 goalflg は目的地設定の際に表示する文字を変化させるための変数である. ゲームスタート時は値が 0 で, 誰かが目的地に到着すると値が 1 になる.

リスト 25: ゲームスタート時の処理

```
1  // ゲーム開始時の処理
2  void startgame(void){
3      if(inflg==0){
4          Initvalue(); // 変数初期化
5          month=4; // 4月にセット
6          year=1; // 1年目にセット
7          calseason(); // 季節計算
8          turn=0; // プレイヤー1のターンにセット
9          goalflg=0; // ゴールフラグ初期化
10         inflg++;
11     }else if(inflg==1){
12         PutSprite(spimg[3],0,0,&spinfo[3],1);
13     }else if(inflg==2){
14         inflg=0;
15         turnstatus=1;
16     }
17 }
```

Initvalue 関数について説明する. リスト 26 に Initvalue 関数のコードを示す. keyboardflg はキーボード入力を受け付ける/受け付けないという管理を行うフラグである. キーを押し続けた場合に, 画面表示がスキップされてしまうことを防ぐため, このような変数を導入している. 5 行目から 7 行目で初期化を行っている dummyresult 配列は, サイコロやプラス駅を代表とする一定時間で変化するランダムな値を画面に表示するときに用いる配列である. キー入力で乱数の生成を停止するときに, タイマーと再描画のタイミングによっては画面の描画の内容と変数に代入される値が異なってしまう場合がある. このため,「ダミー」を用いて, 実際の乱数の計算と画面に表示する乱数を扱う変数を分けることで画面描画の内容と変数に代入される値の整合性をとっている.dummyresult 配列はこのダミーの結果を保持する配列である. 8 行目のグローバル変数 direction は行動中の社長の移動方向を保持する変数である. ここでは上下左右どの方向でもない-1 に設定している. 9 行目のグローバル変数 selectpos はプレイヤーがカードや物件を選択するときに, どれを選択したかを保持するための変数である. ここでは初期値として-1 に設定している.

リスト 26: Initvalue 関数

```
1  // 変数初期化
2  void Initvalue(void){
3      int i;
4      keyboardflg=0;
5      for(i=0;i<SAIKOROMAX;i++){
6          dummyresult[i]=0;
7      }
8      direction=-1;
9      selectpos=0;
10 }
```

表 9: グローバル変数 turn の意味

値	社長
0	社長 1
1	社長 2
2	社長 3

calseason 関数の処理について説明する. リスト 27 に calseason 関数のコードを示す. calseason 関数は月を管理する変数 month から, 季節を計算する関数である. calseason 関数の内部ではグローバル変数 season に季節を表す値を代入する処理を行っている. グローバル変数 season の値と季節の対応は表 10 の通りである.

リスト 27: calseason 関数

```
1  //季節番号を計算
2  // 0:春 3~5月
3  // 1:夏 6~8月
4  // 2:秋 9~11月
5  // 3:冬 12~2月
6  void calseason(void){
7      if((3<=month)&&(month<=5)){
8          season=0;
9      }else if((6<=month)&&(month<=8)){
10         season=1;
11     }else if((9<=month)&&(month<=11)){
12         season=2;
13     }else{
14         season=3;
15     }
16 }
```

表 10: グローバル変数 season の意味

値	季節
0	春
1	夏
2	秋
3	冬

inflg=1 のとき, 図 7 に示すタイトル画面を表示する処理を行っている.inflg=1 の状態から inflg=2 の状態に移るためにはキーボードの E キーの入力が必要である. 画面表示ではタイトル画面から目的地の設定画面に移る処理に該当する. リスト 28 にこれを実装している keyboard 関数の抜粋, keyboardTimer 関数,isE 関数の 3 つのコードを示す. keyboard 関数全体は付録を参照してほしい. keyboard 関数は turnstatus=0 のとき,E キーの入力を受け取ると inflg をインクリメントする処理を行う. 入力されたキーが E であるかどうかは isE 関数で判定している.isE 関数 (リスト 28 の 11 行目から 17 行目) は引数として受け取った文字が E のとき 1, それ以外では 0 を返す処理を行っている.E キーかどうか判定する関数を作成したのは, 本ゲームのほとんどの入力が E キーで完結するためである.

キーボードの入力の処理が完了すると一定時間キーボード入力を無視する処理を実装している. 画面切り替えのほとんどは E キーが担当しているため,E キーを長押しされると画面の描画が混沌としてしまう. これを防ぐために一定時間キーボード入力を無視する処理を実装している. この処理は keyboard 関数および keyboardTimer 関数で実装している. キー入力の処理が完了した後, リスト 28 の 31 行目で keyboardflg を 1 にして, 32 行目で keyboardTimer 関数をコールバック関数として呼び出している. この場合は 500ms 後に keyboardTimer 関数が呼び出される.keyboardTimer 関数の定義はリスト 28 の 2 行目から 6 行目である. keyboardTimer 関数は keyboardflg を 0 にする処理を行っている.keyboard 関数は,21 行目に示すように keyboardflg が 0 のときのみキー入力に対する処理を行うため,keyboardTimer 関数が呼び出されるまでの間にキー入力が行われても何もしない. これらによってキーボードの入力を一定時間無視する処理を実装している.

inflg=2 のときは,inflg を 0 にリセットして turnstatus を 2, つまり目的地の設定を行う処理を行うようにセットしている. この処理によって, 次に Display 関数が実行されるときに, 目的地の処理が行われる.

図 7: タイトル画面

リスト 28: キーボードの入力制御 (ゲームスタート)

```
1 //キーボード入力管理タイマー
2 void keyboardTimer(int t)
3 {
4     // キーボード入力ロックを解除
5     keyboardflg=0;
6 }
```

```
7
8  // eを押したか判定
9  // 1: キーがE
10 // 0 : キーがEでない
11 int isE(unsigned char key){
12     if(key=='e'){
13         return 1;
14     }else{
15         return 0;
16     }
17 }
18
19 void keyboard(unsigned char key,int x,int y){
20     int locktime =500;
21     if(keyboardflg==0){ // キーボード入力がロックされていないとき
22         if(turnstatus==0){ //タイトル
23             if(isE(key)){
24                 inflg++;
25             }
26         }
27
28         (中略)
29
30         if(turnstatus!=5){
31             keyboardflg=1; // キーボード入力ロック
32             glutTimerFunc(locktime, keyboardTimer, 0); // ロック解除タイマー
33     }
34     }
35 }
```

4.13 目的地の設定処理

目的地の設定について説明する.Display 関数では turnstatus が 1 のときの処理である. turnstatus が 1 のときのキーボードの処理についてはタイトル画面と同様に E キーを押すと inflg が 1 インクリメントされる仕組みになっている. このため,keyboard 関数のコードは省略する. リスト 29 に目的地の設定を行う関数である desicionDist 関数のコードを示す.

リスト 29: desicionDist 関数

```
1  // 目的地決定処理
2  void desicionDist(void){
3      char fname[150];
4      PutSprite(spimg[2],0,0,&spinfo[2],1); // 背景表示
5      if(inflg==0){
6          if(goalflg==0){ // 初めて目的地をセットするとき
7              // さいしょのもくてきちをきめます.
8              // Eでルーレットをまわしてください.
9              sprintf(fname,"saiisilonomokutekitiwokimemasumrxxxede
10             llrullmsllrelttollwomawasitekudasaiimr");
11         }else if(goalflg==1){
12             // つぎのもくてきちをきめます.
13             // Eでルーレットをまわしてください.
14             sprintf(fname,"tuginomokutekitiwokimemasumrxxxedellru
15             llmsllrelttollwomawasitekudasaiimr");
16             }
17         drawText(fname,11,225,InitWidth-22,42,0);
18     }else if(inflg==1){ // 乱数生成用の設定
19         dummynum=1;
20         dummyresult[0]=0;
21         range=STATIONNUM; // rangeを駅の数にセット
22         randflg=1; // ダミータイマーロック解除
23         //タイマー呼び出し
24         glutTimerFunc(RANDTIME, RandTimer, 0);
25         inflg++;
26     }else if(inflg==2){ // ダミーリザルトを表示
27         drawString(stations[dummyresult[0]].name,0,InitWidth/2-80,105,1);
28         // Eでとめます.
29         sprintf(fname,"xedetomemasumr");
30         drawText(fname,11,225,InitWidth-22,42,0);
```

```
31      }else if(inflg==3){
32          randflg=0; // タイマー停止
33          randresult=rand()%range; // 結果を計算
34          // 目的地の座標,名前を格納
35          distination.x=stations[randresult].x;
36          distination.y=stations[randresult].y;
37          sprintf(distination.name,"%s",stations[randresult].name);
38          inflg++;
39      }else if(inflg==4){
40          // 目的地を画面出力
41          // もくてきちはhogeです.
42          // Eをおしてください.
43          sprintf(fname,"mokutekitiha%sdesumrxxxewooositekudasaiimr",distination.name);
44          drawText(fname,11,225,InitWidth-22,42,0);
45          drawString(distination.name,0,InitWidth/2-80,105,1);
46      }else if(inflg==5){ // status更新
47          inflg=0;
48          if(goalflg==1){
49              turnstatus=7;
50          }else{
51              turnstatus=2;
52          }
53      }
54  }
```

desicionDist 関数の処理について説明する．まず,inflg の値にかかわらず行われる処理について説明する．これは背景を表示する処理のことである．背景を表示する処理は 4 行目で行っている, 表示される背景は図 8 に示すものである．

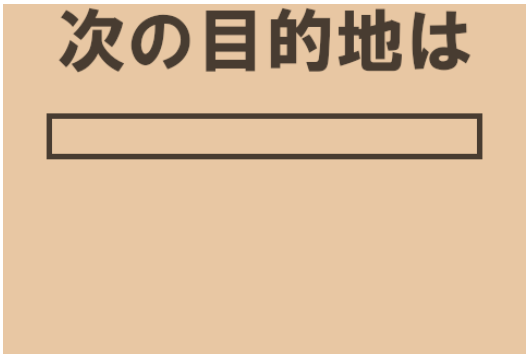

図 8: 目的地設定画面の背景

inflg=0 のとき, 目的地を決める趣旨のダイアログを表示する処理を行っている．表示する文字は goalflg によって異なる．goalflg=0, つまり初めて目的地を決めるときは図 9 に示すように「さいしょのもくてきちをきめます.E でルーレットをまわしてください.」というダイアログが表示される．inflg=1, つまり目的地の再設定が行われるときは図 10 に示すように「つぎのもくてきちをきめます.E でルーレットをまわしてください.」というダイアログが表示される．

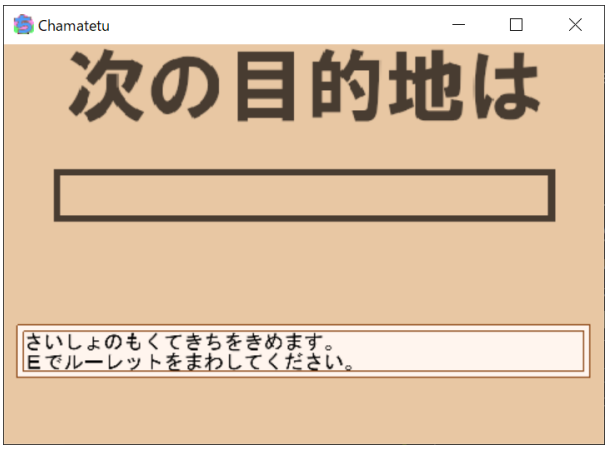

図 9: 初めて目的地を設定する場合の表示

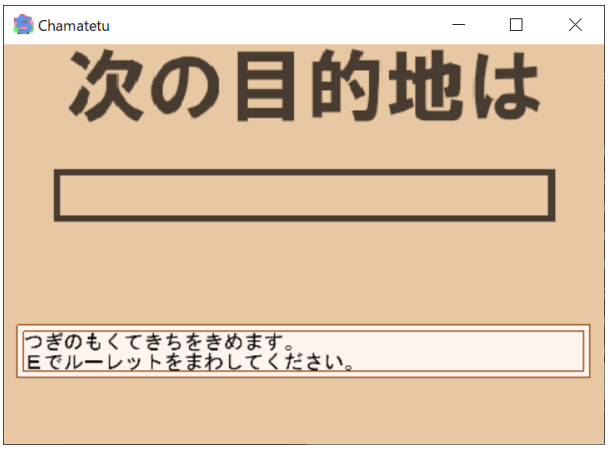

図 10: 目的地を再設定する場合の表示

ダイアログの表示方法について説明する. リスト 30 にダイアログを表示するための関数を示す. リスト 30 には drawDialog 関数と drawText の 2 つの関数の定義が記述されている.drawDialog 関数はダイアログの左上の座標 (x,y) から幅 width, 高さ height のダイアログを表示する関数である. ダイアログの描画は OpenGL のプリミティブ (図形を簡単な図形に分割して描くこと) を用いて行っている. 描画方法は,glBegin 関数と glEnd 関数の間に glVertex2i 関数を用いて座標を必要な数だけ並べることで行う. 並んだ図形の結び方は glBegin 関数の引数として指定する. ここでは中を塗りつぶす四角形を描画する「GL_QUADS」と中を塗りつぶさない線を描画する「GL_LINE_LOOP」を用いている. ダイアログは 3 つの図形を組み合わせて表示している. 1 つ目はダイアログの背景になるペールオレンジの四角形である. これはリスト 30 の 4 行目から 9 行目で描画している. 2 つめは外側の縁取り部分 (茶色) である. これはリスト 30 の 12 行目から 18 行目で描画している. 3 つめは内側の縁取り部分 (茶色) である. これはリスト 30 の 21 行目から 26 行目で描画している.

drawText 関数はダイアログの生成と文字の表示は同時に行うための関数である. リスト 30 では 29 行目から 34 行目である.drawText 関数は引数として受け取った文字列 string を drawDialog 関数と drawString 関数を用いて描画する処理を行っている. drawText 関数ではダイアログと文字列が被らないように drawString 関数の引数として与える座標 (x+5,y+5) のようにすることでダイアログと表示する文字列が被らないようにする処理も行っている.

リスト 30: ダイアログ表示の実装

```
1 // ダイアログを画面に描画
```

```
2  void drawDialog(int x,int y,int width,int height){
3      // ダイアログの背景を描画
4      glBegin(GL_QUADS);
5      glVertex2i(x,y);
6      glVertex2i(x,y+height);
7      glVertex2i(x+width,y+height);
8      glVertex2i(x+width,y);
9      glEnd();
10
11     // 外側の四角を縁取り
12     glColor3ub(139,69,19);
13     glBegin(GL_LINE_LOOP);
14     glVertex2i(x,y);
15     glVertex2i(x,y+height);
16     glVertex2i(x+width,y+height);
17     glVertex2i(x+width,y);
18     glEnd();
19
20     // 内側の四角を縁取り
21     glBegin(GL_LINE_LOOP);
22     glVertex2i(x+5,y+5);
23     glVertex2i(x+5,y+height-5);
24     glVertex2i(x+width-5,y+height-5);
25     glVertex2i(x+width-5,y+5);
26     glEnd();
27 }
28
29 // テキスト表示
30 void drawText(char *string,int x,int y,int width,int height,int color){
31     glColor3ub(255,245,238);
32     drawDialog(x,y,width,height);
33     drawString(string,color,x+5,y+5,0.5);
34 }
```

図 9 および図 10 の状態で E キーを押すと inflg がインクリメントされ inflg=1 の処理が行われる. inflg=1 の処理は乱数生成のための変数準備とダミーの乱数生成関数の起動である. inflg=1 のときの処理は図 29 の 18 行目から 26 行目である. 10 行目ではグローバル変数 dummynum に 1 を代入している. 変数 dummynum は dummyresult 配列からいくつ結果を取り出すかを示す変数である. ダミー乱数の生成は複数の乱数を同時に生成したい場合があるため,dummyresult 配列からいくつ結果を取り出すかで管理している. 21 行目ではグローバル変数 range に駅の数を代入している. 変数 range は乱数の生成範囲を指定する変数である. 変数 range に代入した値 M, 生成される乱数 x とすると, 乱数を生成する関数では $0 \leq x < M$ の範囲の整数乱数が生成される.22 行目ではグローバル変数 randflg を 1 にすることでダミー乱数を生成するタイマーを起動する準備をしている. 24 行目では, ダミー乱数を生成するタイマーである RandTimer 関数をコールバック関数として登録する処理を行っている.

リスト 31 に RandTimer 関数のコードを示す. RandTimer 関数の内部では, dummynum 個の乱数を生成し, その結果を dummyresult 配列に代入する処理を行っている. そして, randflg が 1 のときは再度 RandTimer 関数を呼び出す処理を行っている. 定数 RANDTIME の値は 100 であるから,100ms おきに RandTimer 関数が呼び出される.

リスト 31: RandTimer 関数

```
1  // ダミー乱数を一定時間ごとに生成するタイマー
2  void RandTimer(int t)
3  { // (0,range-1)の範囲の乱数を生成
4      int i;
5      for(i=0;i<dummynum;i++){
6          dummyresult[i] = rand()%range;
7      }
8      if(randflg==1){ // randflgがたっているときタイマー継続
9          glutTimerFunc(RANDTIME, RandTimer, 0);
10     }
11 }
```

最後に inflg をインクリメントする処理を行っている (リスト 29 の 25 行目). この処理によって inflg=1 の処理は一回のみ行われ,inflg=2 の処理が始まる.

次に inflg=2 のときの処理について説明する.inflg=2 のとき, 生成したダミー乱数の結果を画面に出力する処理を行っている. 図 11 および図 12 に inflg=2 のときの画面表示を示す. 図 11 および図 12 に示すように, 目的地駅のダミー表示は 100ms おきに変化する. また,「E でとめます.」というダイアログが表示される.

図 11: ランダムな駅の表示例 1

図 12: ランダムな駅の表示例 2

inflg=2 の処理は, リスト 29 では 26 行目から 30 行目である. inflg=2 のとき, drawString 関数および drawText 関数を用いて, ダミー乱数のインデックスに該当する駅名と「E でとめます.」というダイアログを表示している.

inflg=2 から inflg=3 への変化はキーボードの E キー入力によって行われる. inflg=3 のときの処理はタイマーの停止とダミーでない本当の結果の計算である. リスト 29 では 31 行目から 38 行目である. inflg=3 の時の処理内容について説明する. 31 行目で変数 randflg を 0 にすることで乱数を生成するタイマーである RandTimer が 100ms おきに呼び出される処理を停止している. これは RandTimer 関数 (リスト 31) の 8 行目で randflg が 1 のときにタイマーをコールバック関数として再登録しているためである. リスト 29 の 33 行目では, 結果として表示する乱数を生成しグローバル変数 randresult に代入している. そして 35 行目から 36 行目で目的地の情報を保持する stationstatus 型の変数 distination に, 目的地の座標および名前を代入する処理を行っている. 最後に変数 inflg をインクリメントすることで inflg=2 の処理が 1 度だけ実行され,inflg=3 の処理に移るようにしている.

inflg=4 のときの処理について説明する. inflg=4 のときの処理は, inflg=3 で計算して結果を画面に表示することである. これによって目指すべき目的地が画面に表示される. 図 13 が inflg=4 のときの画面表示の例である. 図 13 から読み取れるように inflg=4 のときは, 画面に目的地とダイアログが表示される. リスト

29 における inflg=4 の処理は 39 行目から 45 行目である．inflg=4 のとき, drawText 関数で「もくてきちは hoge です.E をおしてください.」というダイアログを表示し,drawString 関数で目的地名を表示する処理を行っている．

図 13: 目的地の決定

最後に inflg=5 のときの処理について説明する．inflg=4 から inflg=5 への変化は E キー入力の入力によって行われる．リスト 29 では 46 行目から 53 行目である．目的地を設定した後の処理は 2 通りある．初めての目的地設定の場合は, turnstatus=2, つまりターンのはじめの処理が行われる．2 回目以降の目的地の設定の場合,turnstatus=7, つまり物件の購入処理が行われる．この処理を行っているのが 48 行目から 52 行目である．目的地の設定が初めてかどうかはグローバル変数 goalflg が管理しているから, 変数 goalflg の値によって turnstatus を変更する処理を行っている．これらの処理によって目的地の設定を行っている．

4.14 プレイヤーおよびマップの描画処理

プレイヤーおよびマップの描画処理について説明する．これらの処理は turnstatus が 2,4,5,6,7,8,9,10 で共通する処理である．まず, マップの描画について説明する．マップの描画はリスト 32 に示す drawMap 関数で行っている．マップは図 4 に示したように 960×960 のサイズである．これに対して, ウィンドウのサイズは 480×320 であるため, マップ全体をウィンドウに表示することはできない．そこで, ターン中の社長を画面の中心に描画し, マップは社長を中心にウィンドウにおさまる範囲を描画する仕様にした．

リスト 32: drawMap 関数

```
1  // マップを描画
2  void drawMap(void){
3      int x,y;
4      int drawx,drawy;
5      int img_num;
6      for(y=0;y<InitHeight/IMGSIZE;y++){
7          for(x=0;x<InitWidth/IMGSIZE;x++){
8              drawx = x*IMGSIZE;
9              drawy = y*IMGSIZE;
10             img_num = getmapnum(x+tx,y+ty);
11             if((distination.x==x+tx)&&(distination.y==y+ty)){ // 目的地のとき
12                 // 目的地画像を描画
13                 PutSprite(mapimg[DIST], drawx, drawy, &mapinfo[DIST],1);
14             }else if(img_num==WALL){ // 草原マップのとき
15                 // 季節にあった草原を描画
16                 PutSprite(seasonimg[season], drawx, drawy, &seasoninfo[season],1);
17             }else{
18                 // マップ描画
19                 PutSprite(mapimg[img_num], drawx, drawy, &mapinfo[img_num],1);
20             }
```

```
21          }
22      }
23  }
```

実際の処理について説明する. まず, ターン中の社長の座標から, 図 14 に示すマップの左上となる Map 配列のインデックス (tx,ty) を計算する. この値は別の関数でも仕様するため, Display 関数のリスト 24 の 3 行目および 4 行目で行っている. 定数 CX は 7,CY は 5 になっている. このため, ターン中の社長のインデックス (x,y) から (CX,CY) を引くことで, ターン中の社長を中心としたときの画面左上の Map 配列におけるインデックスが計算できる.drawMap 関数では 2 重の for 文を用いてマップの描画を行っている. for 文内の処理について説明する. リスト 32 の 8 行目および 9 行目では実際に描画する座標 (drawx,drawy) を計算している. 画像のサイズは 32×32 に統一しているため, 画像サイズを表す定数 (IMGSIZE=32) をインデックス計算のための座標 (x,y) にかけることで実際の座標が計算できる.10 行目では,Map 配列から, 描画する画像の種類を取得している. リスト 33 に getmapnum 関数の定義と, getmapnum 関数内に登場する定数を示す. getmapnum 関数では, 引数として与えたインデックス (x,y) がどの種類のマスかを返す処理を行っている. 15 行目から 18 行目では配列の添え字をはみ出したときの処理を行っている. 配列の処理をはみ出した引数を指定した場合, 背景の番号を返すように設定している. リスト 32 では getmapnum 関数にインデックス (x+tx,y+ty) を与えることで, インデックスが (tx,ty) だけずれ, ターン中の社長を中心とするマップ描画を実現している.

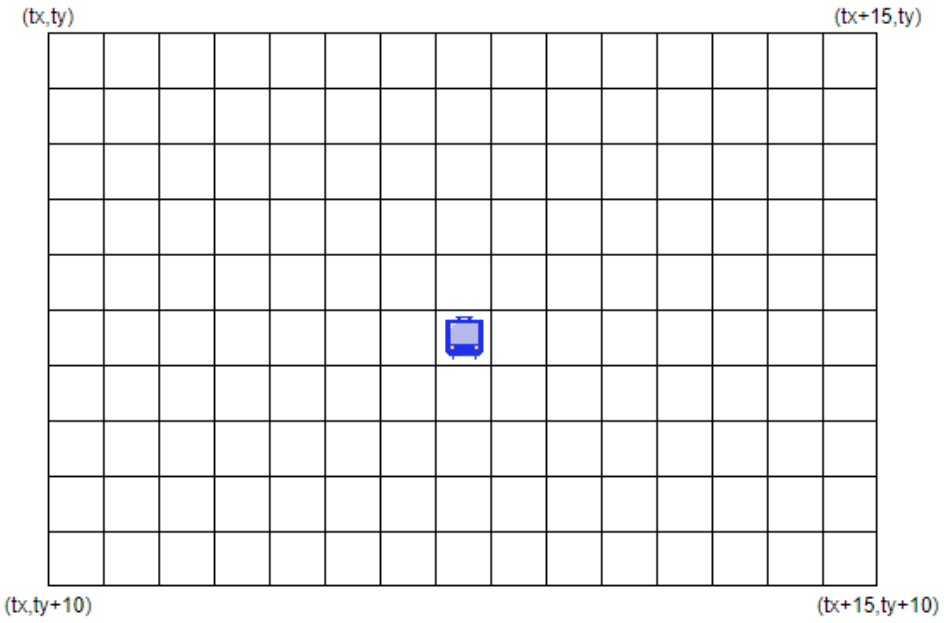

図 14: マップの描画方法

リスト 33: getmapnum 関数

```
1  // マス ID の定義
2  #define PLUSMASU 0 // プラス駅
3  #define MINUSMASU 1 // マイナス駅
4  #define CARDMASU 2 // カード駅
5  #define PROPERTYMASU 3 // 物件駅
6  #define SENRO1 4 // 線路(縦)
7  #define SENRO2 5 // 線路(横)
8  #define DIST 6 // 目的地駅
9  #define WALL 623 // 草原
10
11 // マップの画像番号を取得
12 int getmapnum(int x,int y){
13     int img_num;
```

```
14      // 配列番号をはみ出した場合
15      if((x<0)||(x>=XMAX)){
16          return 623; // 草原マップを返す
17      }else if((y<0)||(y>=YMAX)){
18          return 623; // 草原マップを返す
19      }
20
21      switch (Map[y][x]){
22      case 'A': // 草原
23          img_num=623;
24          break;
25      case 'B': // 物件
26          img_num=3;
27           break;
28      case '|': // 線路(縦)
29          img_num=4;
30          break;
31      case '-': // 線路(横)
32          img_num=5;
33          break;
34      case 'P': // プラス駅
35          img_num=0;
36          break;
37      case 'M': // マイナス駅
38          img_num=1;
39          break;
40      case 'C': // マイナス駅
41          img_num=2;
42          break;
43      }
44      return img_num;
45  }
```

表示する画像の種類を取得できたから, 画像の表示を行う. リスト 32 では 11 行目から 20 行目の処理である. 11 行目から 13 行目は, 例外処理として目的地駅のときに目的地駅の画像を表示する処理を行っている.14 行目から 16 行目では表示する画像が背景のときの処理を行っている. 背景は季節に合わせたものを表示する必要があるため, 変数 season を季節別の画像を格納している seasonimg 配列に渡して描画している.17 行目から 20 行目では目的地および背景以外のときの画像の描画を行っている. これらの処理によってマップを描画している.

次に社長の描画方法について説明する. リスト 34 に社長を画面に表示するための関数である drawPlayer 関数のコードを示す.drawPlayer 関数の内部処理について説明する.drawPlayer 関数はターン中ではない社長を先に描画してから, ターン中の社長を描画する処理を行っている. ターン中の社長を最前面に描画しないと, 行動中にマップのどこにいるか分からなくなってしまうため, このような仕様にしている. ターン中ではない社長を描画はリスト 34 の 6 行目から 14 行目の行っている. ターン中の社長かどうかは 7 行目の if 文で判定している. ターン中の社長出ない場合は, 表示する座標を計算し,PutSprite 関数で描画を行っている. ターン中の社長の描画はリスト 34 で行っている. 座標 (CENTX,CNETY) は画面中心のインデックス (CX,CY) を画像サイズ IMGSIZE 倍した値である. これによって, ターン中の社長が常に画面中心に描画される.

リスト 34: drawPlayer 関数

```
1   // プレイヤーを描画
2   // 最上面にターン中のプレイヤーを描画
3   void drawPlayer(void){
4       int transx,transy;
5       int i;
6       for(i=0;i<PLAYERNUM;i++){
7           if(i!=turn){ // ターン中のプレイヤー以外を描画
8               transx = players[i].x/IMGSIZE;
9               transy = players[i].y/IMGSIZE;
10              PutSprite(playerimg[i], (transx-tx)*IMGSIZE,
11              (transy-ty)*IMGSIZE, &playerinfo[i],1);
12          }
13      }
14      // ターン中のプレイヤーを最上レイヤーに表示
```

```
15      PutSprite(playerimg[turn], CENTX,CENTY, &playerinfo[turn],1);
16  }
```

実際の動作について確認する. ここでは, 次の 3 つの動作について確認する.

1. マップおよび社長が表示されるか
2. 社長が移動したときの動作
3. マップ端の動作

まず, マップおよび社長が表示されるかを確認する. ここでは, ダイアログを代表とするマップと社長以外の表示が少ない turnstatus=5 のマス移動を例に説明する. 図 15 に turnstatus=5 のときの画面表示の例を示す. 図 15 の状況は, 全社長が長野駅におり, 行動中の社長はプレイヤー 1 社長である. レポートでは白黒のため伝わらないが, プレイヤー 1 社長が最前面に描画されている. このことから, ターン中の社長が最前面に表示される処理が正確に行えていることが確認できる. さらに図 15 からターン中の社長が画面の中心に描画されており, マップは社長を中心に描画されていることがわかる. これより, 社長を画面中心, マップを社長を中心に画面範囲で描画できていることが確認できる.

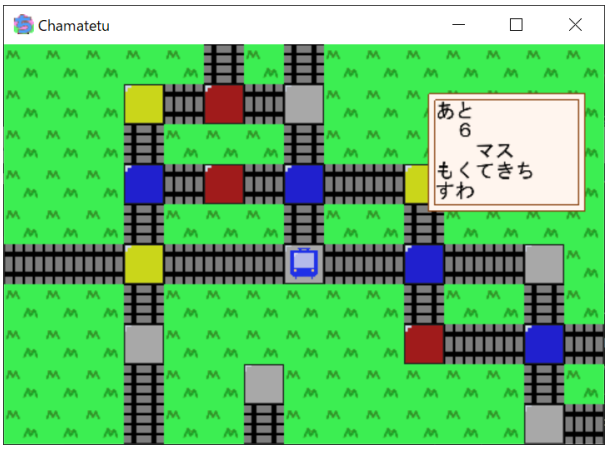

図 15: マップおよび社長の表示

次に社長が移動したときの動作について確認する. ターン中の社長が移動すると, ターン中の社長を中心に描画したまま, マップの表示が変わることを確認する. 図 16 に社長が移動したときの画面表示を示す. 図 16 は図 15 から左に 128px 移動した状態である. 図 16 から, 社長が移動した場合も, 社長が中心に表示され, マップは社長を中心に描画されることがわかる.

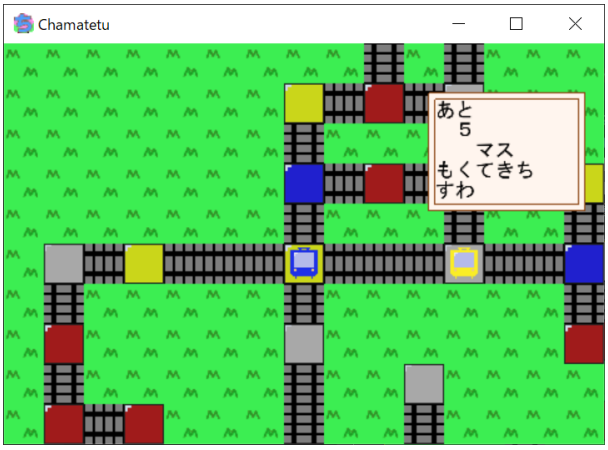

図 16: 社長が移動したときの画面表

最後にマップ端の動作を確認する. 社長の位置によっては, マップの描画位置は Map 配列の定義されている領域をはみ出す場合ある. この場合に背景が表示されることを確認する. 図 17 にプレイヤー 1 社長がマップ端の表示される位置に移動したときの画面表示を示す. 図 17 のプレイヤー 1 社長の Map 配列におけるインデックスは (3,7) である. 社長を中心に描画しようとすると画面表示の左上の Map 配列におけるインデックスは (-4,2) であるため添え字をはみ出している. 図 17 から, 添え字がはみ出している場合に背景が表示されていることが確認できる. これらから, ターン中の社長を中心に描画する処理と, マップをターン中の社長を中心に画面におさまる範囲で描画する処理について確認できた.

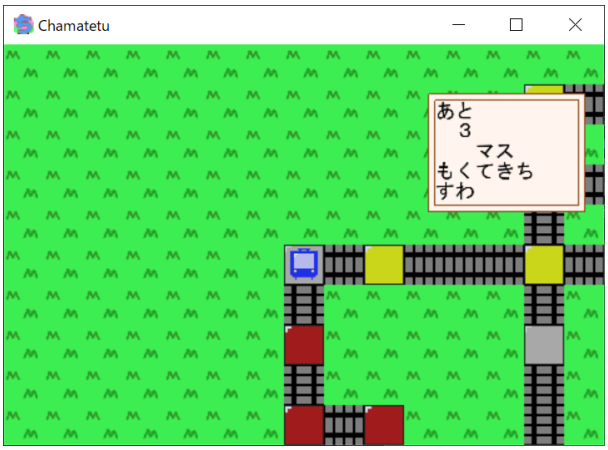

図 17: マップ端の表示

4.15 ターンのはじめの処理

ターンのはじめの処理について説明する. Display 関数 (リスト 24) では turnstatus=2 のときの処理である. Display 関数では turnstatus が 2 のときに,startTurn 関数を呼び出している. リスト 35 に startTurn 関数のコードを示す. startTurn 関数の処理を, 実際の動作を交えて説明する. startTurn 関数の処理において,inflg の状態に関係なく実行される処理は, drawMap 関数と drawPlayer 関数である. これらによってマップおよび社長の画像が画面に表示される. drawMap 関数と drawPlayer 関数は次節以降でも頻繁に登場するが, 使用方法は同じであるため説明は省略する.

次に変数 inflg=0 のときの処理について説明する. リスト 35 では 7 行目から 12 行目である. 変数 inflg=0 のときグローバル変数の初期化を行っている. グローバル変数 saikoro は振るサイコロの数を制御するための変数である. サイコロは通常 1 個のみ振るが, カードの効果によって最大 4 個まで振ることができる仕様になっている. ここでは, サイコロの数を 1 に設定している. 11 行目ではグローバル変数 selectpos に 0 を代入している. 変数 selectpos はユーザーが選択を行う場面で今どれを選択しているのかを示す変数である. 変数の初期化が終了すると,12 行目で変数 inflg をインクリメントして sinflg=0 の状態から inflg=1 の状態に変化させている.

リスト 35: startTurn 関数

```
1  // ターン開始時の処理
2  void startTurn(void){
3      int i;
4      char fname[150];
5      drawMap(); // マップ描画
6      drawPlayer(); // プレイヤー描画
7      if(inflg==0){
8          saikoro=1;
9          dummynum=1;
10         keyboardflg=0;
11         selectpos=0;
```

```
12          inflg++;
13      }else if(inflg==1){
14          // プレイヤーカラーでダイアログ生成
15          glColor3ub(playercolor[turn][0],playercolor[turn][1],playercolor[turn][2]);
16          drawDialog(11,11,InitWidth-22,34+16);
17          // hogeねんめ hugaがつ
18          sprintf(fname,"%dnennmess%dgatu",year,month);
19          drawString(fname,0,16,11+8,0.5);
20          // 所持金表示
21          drawMoney(players[turn].money,InitWidth/2,11+8+16,0,0.5);
22          // hogeしゃちょう
23          sprintf(fname,"%ssilatilouusssssss",players[turn].name);
24          drawString(fname,0,16,11+8+16,0.5);
25          // サイコロ
26          // カード
27          sprintf(fname,"llsaiikoroxxkallmslldoll");
28          drawDialog(11,175,74,42);
29          // セレクトポジション表示
30          if(selectpos == 0){
31              glColor3ub(255,0,0);
32              drawQUAD(16,180,64,16);
33          }else if(selectpos==1){
34              glColor3ub(255,0,0);
35              drawQUAD(16,196,64,16);
36          }
37          drawString(fname,0,16,180,0.5);
38          // hogeしゃちょうのばんです.
39          sprintf(fname,"%ssilatilouunobanndesumr",players[turn].name);
40          drawText(fname,11,225,InitWidth-22,42,0);
41      }else if(inflg==2){
42          if(selectpos==0){
43              inflg++;
44          }else{
45              if(players[turn].cardnum==0){
46                  inflg=4;
47              }else{
48                  selectpos=0;
49                  inflg=5;
50              }
51          }
52      }else if(inflg==3){ // status更新(サイコロをふる)
53          inflg=0;
54          turnstatus++;
55      }else if(inflg==4){ // カードがないとき
56          sprintf(fname,"llkallmslldollga1maiimoaarimasennmr");
57          drawText(fname,11,225,InitWidth-22,42,0);
58      }else if(inflg==5){
59          glColor3ub(255,245,238);
60          drawDialog(155,50,10+10*16,10+16*(players[turn].cardnum+1));
61          // セレクトポジション表示
62          glColor3ub(255,0,0);
63          drawQUAD(160,55+selectpos*16,10*16,16);
64
65          for(i=0;i<players[turn].cardnum;i++){
66              if(players[turn].card[i]!=0){
67                  drawString(cardname[players[turn].card[i]-1],0,160,55+i*16,0.5);
68              }
69          }
70          drawString("modoru",0,160,55+players[turn].cardnum*16,0.5);
71      }else if(inflg==6){
72          if(selectpos==players[turn].cardnum){ // もどるのとき
73              selectpos=0;
74              inflg=1;
75          }else{
76              rcard = cardprocess(players[turn].card[selectpos]);
77              inflg++;
78          }
79      }else if(inflg==7){
80          if(rcard==0){
81              // こうげきがかわされた.
82              sprintf(fname,"kouugekigakawasaretamr");
83              nextflg=1;
84          }else if(players[turn].card[selectpos]==KYUKO){
85              // サイコロが2つになった.
```

```
86              sprintf(fname,"llsaiikorollga2tuninalttamr");
87              nextflg=0;
88          }else if(players[turn].card[selectpos]==TOKKYU){
89              // サイコロが3つになった.
90              sprintf(fname,"llsaiikorollga3tuninalttamr");
91              nextflg=0;
92          }else if(players[turn].card[selectpos]==SINKANSEN){
93              // サイコロが4つになった.
94              sprintf(fname,"llsaiikorollga4tuninalttamr");
95              nextflg=0;
96          }else if(players[turn].card[selectpos]==SAMMIT){
97              // ぜんしゃちょうがhogeしゃちょうのもとにあつまった.
98              sprintf(fname,"zennsilatilouuga%ssilatilouunomotoniaatumalttamr",
99              players[turn].name);
100             nextflg=1;
101         }else if(players[turn].card[selectpos]==BUTTOBI){
102             // hogeしゃちょうはいったいどこへ.
103             sprintf(fname,"%ssilatilouuhaiilttaiidokohe",players[turn].name);
104             nextflg=1;
105         }else if(players[turn].card[selectpos]==JUOKU){
106             // hogeしゃちょうにプラス10億円.
107             sprintf(fname,"%ssilatilouunillpurasull10oxexmr",players[turn].name);
108             nextflg=1;
109         }else if(players[turn].card[selectpos]==TOKUSEIREI){
110             // ぜんしゃちょうのしゃっきんがちょうけしになった.
111             sprintf(fname,"zennsilatilouunosilaltkinngatilouukesininalttamr");
112             nextflg=1;
113         }else if(players[turn].card[selectpos]==GOUSOKKYU){
114             // ほかのちゃちょうのカードがなくなった.
115             sprintf(fname,"hokanosilatilouunollkallmslldollganakunalttamr");
116             nextflg=1;
117         }
118         drawText(fname,11,225,InitWidth-22,42,0);
119     }else if(inflg==8){ //
120         // 使ったカードの消去
121         for(i=selectpos;i<players[turn].cardnum-1;i++){
122             players[turn].card[i]=players[turn].card[i+1];
123         }
124         players[turn].cardnum--;
125         inflg=0;
126         if(nextflg==1){
127             turnstatus=15; // ターン終了
128         }else{
129             turnstatus++; // サイコロをふる処理
130         }
131     }
132 }
```

inflg=1 および inflg=2 のときの処理について説明する. 社長のターンが始まると図 18 および図 19 に示すような画面表示が行われる. 図 18 はプレイヤー 1 社長の場合, 図 19 はプレイヤー 2 社長の場合の画面である. 図 18 から, ターンのはじめには年月および所持金を表示するダイアログ, サイコロとカードのどちらを使用するか選択するダイアログ,「(社長名) しゃちょうのばんです.」という表示を行うダイアログの 3 つが表示されることがわかる. また, 図 18 および図 19 の画面では,「W」および「S」キーで, サイコロとカードの選択を示す赤い四角形の位置を変化させることができる. これをセレクトポジションとよぶことにする. セレクトポジションはデフォルトで「サイコロ」になっており,「S」キーを押すと, 図 20 に示すように「カード」が選択された状態になる. 「カード」を選択した状態で「W」キーを押すと「サイコロ」を選択した状態になる. どちらかを選択した状態で「E」を押した場合の処理は後で行う. これらを実装しているのが inflg=1 および inflg=2 のときの処理である.

図 18: ターンのはじめの画面 (プレイヤー 1 社長)

図 19: ターンのはじめの画面 (プレイヤー 2 社長)

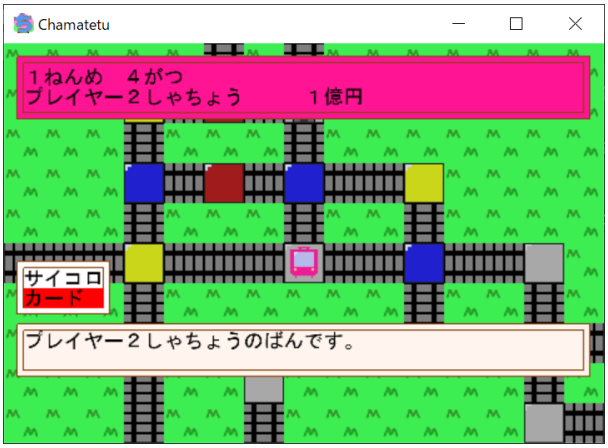

図 20: カードを選択した状態

ダイアログおよびセレクトポジションの処理を実装するコードについて説明する. リスト 35 では 13 行目から 51 行目である. inflg が 1 のとき, 画面に先述した 3 つのダイアログを表示する処理を行っている. 年月および所持金を表示するダイアログは 15 行目から 21 行目, サイコロとカードのどちらを使用するか選択するダイアログは 27 行目から 37 行目,「(社長名) しゃちょうのばんです.」という表示を行うダイアロ

グは 39 行目および 40 行目である. セレクトポジションはグローバル変数 selectpos で管理している. 変数 selectpos はグローバル変数であるから,「W」または「S」のキーの入力があったときに keyboard 関数で変数 selectpos の値を変更する処理を行っている. リスト 36 に keyboard 関数における turnstatus=2 のときの処理を示す. 変数 selectpos の値を操作しているのはリスト 36 の 9 行目から 20 行目である. 9 行目では inflg が 1 かどうか判定を行っている. これは,turnstatus1 のときの他のキーボード入力処理と区別するためである.10 行目から 17 行目は押されたキーが「W」または「S」のキーかどうかを判定し変数 selectpos を変更する処理を行っている. 変数 selectpos が 0 のとき「サイコロ」,1 のとき「カード」になる仕様になっている.

リスト 36: ターンのはじめのキーボード入力の処理

```
1  // キーボード入力管理
2  void keyboard(unsigned char key,int x,int y){
3      int locktime =500;
4      int transx = players[turn].x/IMGSIZE;
5      int transy = players[turn].y/IMGSIZE;
6  if(keyboardflg==0){ // キーボード入力がロックされていないとき
7      (省略)
8      }else if(turnstatus==2){ // ターンのはじめ
9      if(inflg==1){
10         if(key=='w'){
11             if(selectpos==1){
12                 selectpos=0;
13             }
14         }else if(key=='s'){
15             if(selectpos==0){
16                 selectpos=1;
17             }
18         }else if(isE(key)){
19             inflg++;
20         }
21     }else if(inflg==4){
22         if(isE(key)){
23             inflg=1;
24         }
25     }else if(inflg==5){
26         if(key=='w'){
27             if(selectpos>=1){
28                 selectpos--;
29             }
30         }else if(key=='s'){
31             if(selectpos<players[turn].cardnum){
32                 selectpos++;
33             }
34         }else if(isE(key)){
35             inflg++;
36         }
37     }else{
38         if(isE(key)){
39             inflg++;
40         }
41     }
42     }
43     (省略)
44
45     if(turnstatus!=5){
46         keyboardflg=1; // キーボード入力ロック
47         glutTimerFunc(locktime, keyboardTimer, 0); // ロック解除タイマー
48     }
49 }
50 }
```

keyboard 関数で selectpos の値を操作することで, リスト 35 の 30 行目の if 文の真偽が変化し, どちらを選択しているか示すための赤い四角形を描画している. 赤い四角形は drawQUAD 関数で描画している. リスト 37 に drawQUAD 関数の定義を示す. drawQUAD は引数として, 四角形を描画する左上の座標 (x,y) および四角形の幅 width, 高さ height を受け取る. 内部では, プリミティブを用いた描画を行う OpenGL の関数を用いて四角形の描画を行っている.

リスト 37: drawQUAD 関数

```
// 四角形を描画
void drawQUAD(int x,int y,int width,int height){
    glBegin(GL_QUADS);
    glVertex2i(x,y);
    glVertex2i(x,y+height);
    glVertex2i(x+width,y+height);
    glVertex2i(x+width,y);
    glEnd();
}
```

inflg=1 の状態から inflg2 の状態になる処理は, リスト 36 の 18 行目から 20 行目に示すように E キーの入力による inflg のインクリメントによって行っている.inflg が 2 のときの処理 (リスト 35 の 41 行目から 51 行目) は, E キーを押したときのセレクトポジションによって処理を分岐する処理である. セレクトポジションが「サイコロ」, つまり変数 selectpos=0 の場合は inflg を 3 にしている. セレクトポジションが「カード」, つまり変数 selectpos が 1 の場合はカード枚数を確認し, カード枚数が 0 のとき inflg を 4, そうでないときは inflg を 5 にしている.

inflg=3 のときの処理について説明する. リスト 35 では 52 行目から 54 行目である. inflg が 3 のときの処理はサイコロを振る処理に turnstatus の値を変更することである.turnstatus=3 は未使用であるが, ここでは turnstatus をインクリメントする処理を行っている.

inflg=4 のときの処理を説明する.inflg=4 の処理はカードが 1 枚もないときに,「カード」が選択された時の処理である. この場合, 図 21 に示すように「カードが 1 まいもありません.」と画面に表示される. リスト 35 では 55 行目から 57 行目である. 図 21 の状態で,E キーを押すとリスト 36 の 21 行目から 24 行目に示す処理によって inflg=1, つまり図 18 に示す画面に戻る.

図 21: カードが 1 枚もないときのカード表示

最後に,inflg が 5〜8 のときの処理について説明する. inflg が 5〜8 のときの処理は図 18 で「カード」を選択したときの処理である. カードを持っている場合に「カード」を選択すると図 22 に示すように, 手持ちのカードが表示される. 図 22 の例では「サミットカード」,「とっきゅうカード」,「ごうそっきゅうカード」の 3 種類を持っている. また, 一番下のセレクトポジションには「もどる」という項目がある. いずれかのカードを選択した状態で E キーを押すと, カードが使用される.「もどる」を選択して E キーを押すと図 18 の画面に戻る.

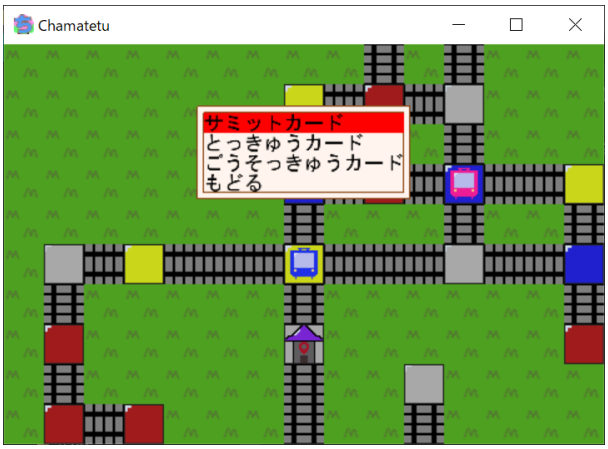

図 22: 手持ちのカードの表示

これらが inflg が 5〜8 のときの処理である. リスト 35 では 58 行目から 122 行目である. 「カード」が選択されるとまず inflg=5 の処理が行われる. リスト 35 では 58 行目から 70 行目である. inflg=5 の処理は, 手持ちのカード,「もどる」, セレクトポジションの 3 つ画面に表示する処理である. 手持ちのカードは playerstatus 型の配列 players のメンバ card が保持しているから,drawString 関数でこれを表示している. inflg=5 でのセレクトポジションの処理はリスト 36 の 25 行目から 41 行目に記述されている. セレクトポジションの移動処理は「カード」,「サイコロ」を選択するときの処理と同じであるため, 説明は省略する.

カードを選択して E キーを押すと inflg=6 の処理が行われる. リスト 35 では 71 行目から 78 行目である. ターン中の社長のカード枚数 c とすると, 変数 selectpos の値 s が $0 \leq s \leq c-1$ のときは, 所持しているカードのいずれかが選択されているため, カードの効果を発動する関数である cardprocess 関数を実行している. 変数 selectpos の値がターン中の社長のカード枚数と等しいとき,「もどる」が選択されているため inflg=1, つまり図 18 の画面に戻るようにしている.

カードの効果を発動する関数である cardprocess 関数について説明する. リスト 38 に cardprocess 関数のコードを示す.cardprocess 関数は引数として効果を発動するカードの番号を受け取る. また, 返り値としてカードの発動が成功した場合に 1, 失敗した場合に 0 を返す. カードの番号と発動するカードはリスト 38 の 2 行目から 10 行目のようになっている. 各カードの処理について説明する. 発動するカードが, 急行カード, 特急カード, 新幹線カードのいずれかのときサイコロの数を管理するグローバル変数 saikoro の値を更新する処理を行っている. サイコロの数は急行カードのときは 2 個, 特急カードのときは 3 個, 新幹線カードのときは 4 個にしている.

サミットカードの場合は, 成功または失敗の計算をしたのちに, カードの効果を発動している. 確率 $\frac{2}{3}$ で成功するために, 乱数を生成して 3 で割った値が 0 でないときにカードの効果を発動するようにしている. サミットカードの効果は, すべての社長をターン中の社長の駅に集める効果であるから, 全ての社長の座標をターン中の社長の座標で上書きする処理を行っている. ぶっとびカードの効果は社長をランダムな物件駅に飛ばす効果であるから, 物件駅の数を表す定数 STATIONNUM で乱数をわることで, ランダムな駅のインデックスを取得している. そして, 取得したインデックスに該当する物件駅の座標を stations 構造体から取得して画像サイズ IMGISIZE 倍することで, 新しい座標が計算される. この座標をターン中の社長の座標に代入することで, ランダムな駅へぶっとぶ効果を実装している. 10 億円カードは使用すると 10 億円が手に入るカードであった.10 億円カードの時の処理は, ターン中の社長の money をプラス 100000 する処理ことで 10 億円を所持金に追加している. 徳政令カードは借金を負っている社長の所持金を 0 にするカードであった. このため, 全ての社長について所持金を保持するメンバ money が 0 よりも小さいか判定し,0 よりも小さい場合はメンバ money を 0 にする処理を行っている. 剛速球カードは確率 $\frac{1}{2}$ で成功し, 他の社長のカードを全て破棄するカードであった. このため, 生成した乱数を 2 で割り 0 のときはターン中の社長以外のカード配列およびカード枚数を 0 で埋める処理を行っている. これらの処理によってカードの効果を実装している.

リスト 38: cardprocess 関数

```
// カード番号の定義
#define KYUKO 1 // 急行カード
#define TOKKYU 2 // 特急カード
#define SINKANSEN 3 // 新幹線カード
#define SAMMIT 4 // サミットカード
#define BUTTOBI 5 // ぶっとびカード
#define JUOKU 6 // 十億円カード
#define TOKUSEIREI 7 // 徳政令カード
#define GOUSOKKYU 8 // 剛速球カード

// カード処理
int cardprocess(int num){
    int r=1;
    int i,j,randst;
    if(num==KYUKO){ // 急行カード
        saikoro=2;
    }else if(num==TOKKYU){ // 特急カード
        saikoro=3;
    }else if(num==SINKANSEN){ // 新幹線カード
        saikoro=4;
    }else if(num==SAMMIT){ // サミットカード
        if(rand()%3!=0){
            for(i=0;i<PLAYERNUM;i++){
                players[i].x=players[turn].x;
                players[i].y=players[turn].y;
            }
        }else{
            r=0;
        }
    }else if(num==BUTTOBI){ // ぶっとびカード
        randst = rand()%STATIONNUM;
        players[turn].x = stations[randst].x*IMGSIZE;
        players[turn].y = stations[randst].y*IMGSIZE;
    }else if(num==JUOKU){ // 10億円カード
        players[turn].money+=100000;
    }else if(num==TOKUSEIREI){ // 徳政令カード
        for(i=0;i<PLAYERNUM;i++){
            if(players[i].money<0){
                players[i].money=0;
            }
        }
    }else if(num==GOUSOKKYU){ // 剛速球カード
        if(rand()%2){
        for(i=0;i<PLAYERNUM;i++){
            if(i!=turn){
                players[i].cardnum=0;
                for(j=0;j<CARDMAX;j++){
                    players[i].card[j]=0;
                }
            }
        }
        }else{
           r=0;
        }
    }
    dummynum=saikoro;
    return r;
}
```

カードの発動の成功/失敗はリスト 35 の 76 行目に示すように変数 rcard に保存される. カードの処理が終了すると, リスト 35 の 77 行目に示すように,inflg がインクリメントされ inflg=7 の処理が行われる. inflg=7 のときの処理について説明する. inflg=7 の処理はカードを使用した結果を画面に表示する処理である. 図 23〜図 31 に各カードを使用したときの画面表示を示す. 図 23〜図 30 はカードの使用に成功した場合, 図 31 はカードの使用に失敗した場合である. 使用したカードによって表示する内容が変化する処理はリスト 35 の 80 行目から 117 行目に記述されている. また, グローバル変数 nextflg の値を急行カード, 特急カード, 新幹線カードのいずれかのカードのとき 0, それ以外のカードのとき 1 になるようにする処理を行っている. 急行カード, 特急カード, 新幹線カードの 3 つのカードはカードを使用した後にサイコロを振るため, 変数 nextflg

を使用して, 使用したカードの判別を行っている.

図 23: 急行カードを使用したときの表示

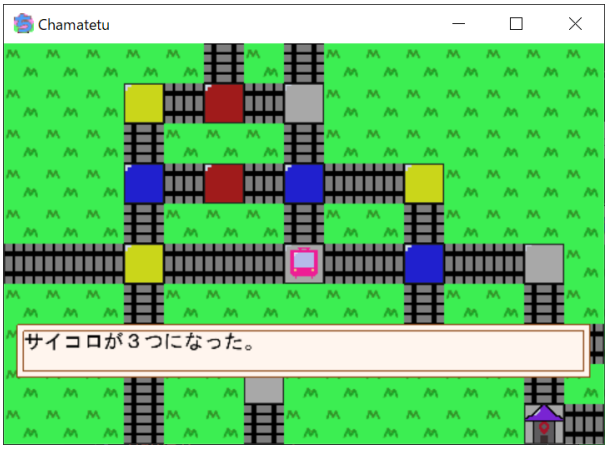

図 24: 特急カードを使用したときの表示

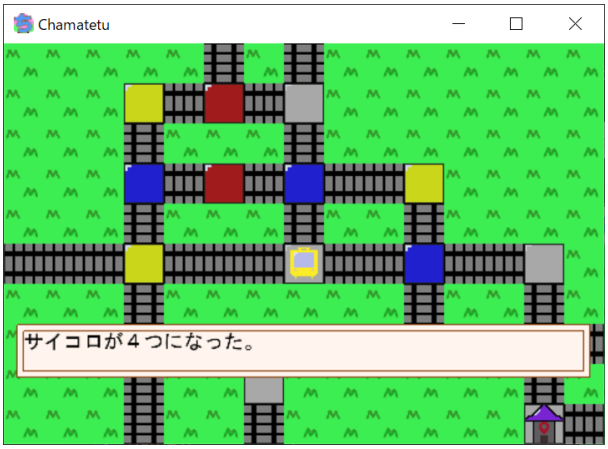

図 25: 新幹線カードを使用したときの表示

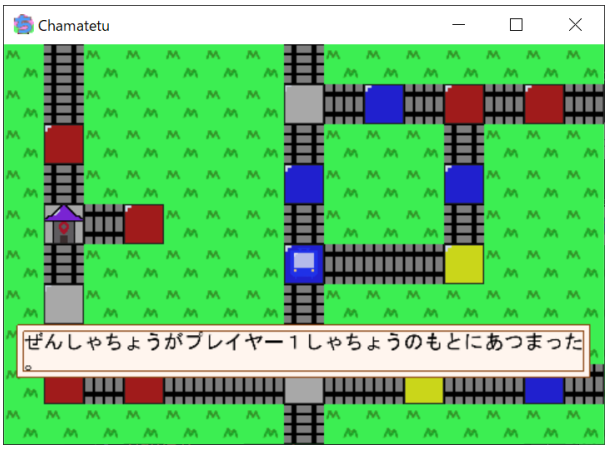

図 26: サミットカードを使用したときの表示

図 27: ぶっとびカードを使用したときの表示

図 28: 10 億円カードを使用したときの表示

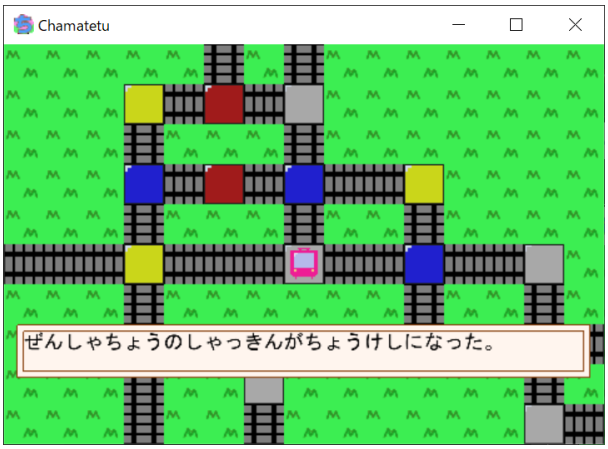

図 29: 徳政令カードを使用したときの表示

図 30: 剛速球カードを使用したときの表示

図 31: カード使用が失敗したときの表示

使用したカードの種類別の画面出力が行われた状態で E キーを押すと inflg=7 の状態から inflg=8 の状態に変化する．inflg=8 の処理は, 使用したカードの消去と turnstatus の変化の処理である．リスト 35 では 119 行目から 131 行目である．カードの消去は 121 行目から 124 行目で行っている．turnstatus の変化の処理は 126 行目から 130 行目で行っている．nextflg が 1 のとき, サイコロを増やすカードが仕様されたこ

とを示しているため turnstatus を 4, つまりサイコロを振る処理にセットしている．nextflg が 0 のときは turnstatus を 15, つまりターン終了処理にセットしている. これらの処理によってターンのはじめの処理を実装している.

4.16 サイコロをふる処理

サイコロを振る処理について説明する．これは turnstatus=4 のときの処理である．Display 関数 (リスト 24) では turnstatus=4 のとき,rollDice 関数を呼び出している．リスト 39 に rollDice 関数のコードを示す. また,turnstatus=4 のときの keyboard 関数のコードをリスト 40 に示す．turnstatus=4 のときのキーボードの処理は E キーを押したときに inflg をインクリメントするだけである.

リスト 39: rollDice 関数

```
// サイコロをふる処理
void rollDice(void){
    int i;
    char fname[150];
    drawMap();
    drawPlayer();
    if(inflg==0){
        // ダミーサイコロを起動
        for(i=0;i<SAIKOROMAX;i++){
            dummyresult[i]=0;
        }
        range=DICEMAX;
        randflg=1;
        glutTimerFunc(RANDTIME, RandTimer, 0);
        inflg=1;
    }else if(inflg==1){
        // サイコロ描画
        for(i=0;i<dummynum;i++){
            PutSprite(diceimg[dummyresult[i]], 416, 32+32*i, &diceinfo[dummyresult[i]],1);
        }
        // Eでサイコロをとめます.
        sprintf(fname,"xedellsaiikorollwotomemasumr");
        drawText(fname,11,225,InitWidth-22,42,0);
    }else if(inflg==2){
        // サイコロ結果処理
        randflg=0;
        recount=0;
        for(i=0;i<saikoro;i++){
            randresulttmp[i] = rand()%range;
            recount+=randresulttmp[i]+1;
        }
        randresult=recount;
        inflg++;
    }else if(inflg==3){
        for(i=0;i<saikoro;i++){
            PutSprite(diceimg[randresulttmp[i]], 416, 32+32*i,
            &diceinfo[randresulttmp[i]],1);
        }
        // Eをおしてください.
        sprintf(fname,"xewooositekudasaiimr");
        drawText(fname,11,225,InitWidth-22,42,0);
    }else if(inflg==4){
        inflg=0;
        turnstatus=5;
    }
}
```

リスト 40: サイコロを振るときのキーボードの処理

```
void keyboard(unsigned char key,int x,int y){
    int locktime =500;
    int transx = players[turn].x/IMGSIZE;
    int transy = players[turn].y/IMGSIZE;
if(keyboardflg==0){ // キーボード入力がロックされていないとき
```

```
6       (省略)
7       }else if(turnstatus==4){ // サイコロをふるとき
8           if(isE(key)){
9               inflg++;
10          }
11      }
12      (省略)
13      if(turnstatus!=5){
14          keyboardflg=1; // キーボード入力ロック
15          glutTimerFunc(locktime, keyboardTimer, 0); // ロック解除タイマー
16      }
17  }
18  }
```

rollDice 関数の処理について説明する．変数 inflg=0 のとき, ダミー乱数を生成するための変数の初期化およびコールバック関数の登録を行っている．この処理は目的地設定のときの乱数生成の準備の処理とほぼ同じである. 乱数の生成範囲は定数 DICEMAX=6 を与えている．inflg=1 のときは図 32 および図 33 に示すようにダミー乱数で生成した値をサイコロで表示する処理と,「E でサイコロをとめます.」というダイアログの表示を行っている．図 32 はサイコロ 1 つ, つまりターンのスタート時に「サイコロ」を選択した場合, 図 33 は新幹線カードを使用してサイコロを 4 つにしたときの処理である. 急行カード, 特急カードの場合については, サイコロの数が変化するだけであるから省略する．図 32 ではサイコロとして「5」が表示されいるが実際には 100ms おきにランダムなサイコロの画像が表示される．サイコロの画像データは diceimg 配列および diceinfo 配列に格納されているおり, 表示したいサイコロの目から 1 を引いた値をインデックスで指定することでサイコロの画像が表示できる．

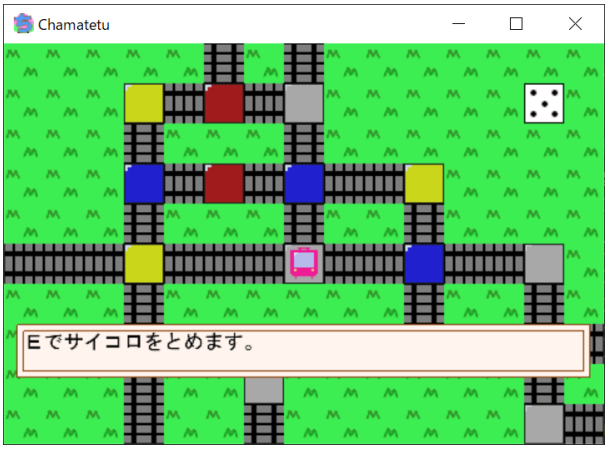

図 32: サイコロを振ったときの画面表示 1

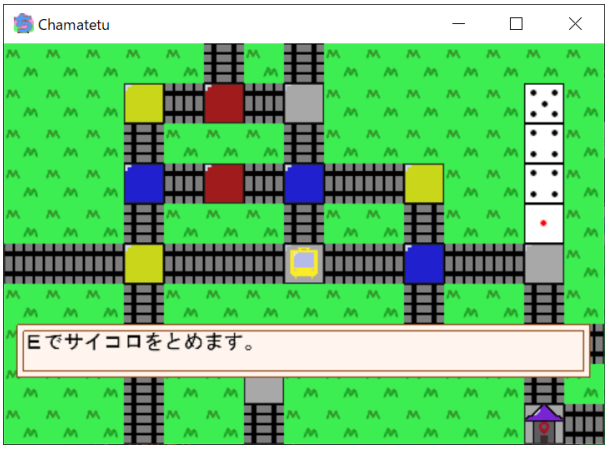

図 33: サイコロを振ったときの画面表示 2

inflg=1 の状態でキーボードから E キーの入力があると inflg がインクリメントされ,inflg=2 の処理が行われる. リスト 39 では 24 行目から 33 行目である. inflg=2 の処理はダミーの乱数生成を停止して, 本当の結果を計算することである.28 行目から 32 行目でサイコロの数 saikoro をループ回数として, 本当の結果を計算している.29 行目で生成した乱数をグローバル変数 (配列)randresulttmp に格納する. 格納した値はサイコロを画面に表示するためのインデックスの番号であるから, これに 1 を足してグローバル変数 recout に加算する. これによって, サイコロの結果の合計が変数 recount に代入される. 最後に画面に結果を出力するための変数 randresult に recount を代入して, inflg をインクリメントしている.
inflg=3 および inflg=4 のときの処理について説明する. inflg=2 でサイコロの出目の計算を行ったから,inflg=3 ではこれを画面に表示する処理を行う. inflg=3 では, 図 34 および図 35 に示すように, サイコロの出目と「E をおしてください.」というダイアログを表示する処理を行っている. inflg=3 の処理は, リスト 39 の 34 行目から 41 行目である. サイコロの出目は randresulttmp 配列に格納されているため, これを PutSprite 関数で表示している. inflg=3 の状態で E キーを押すと inflg がインクリメントされ,inflg=4 の処理が行われる.inflg=4 の処理は turnstatus を 5, つまり移動処理に更新する処理を行っている. これによって, サイコロの出目を表示する画面から, 移動を行う画面に切り替わる.

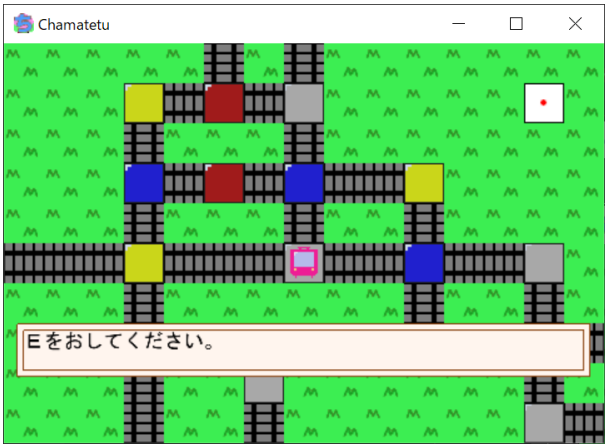

図 34: サイコロをとめたときの画面表示 1

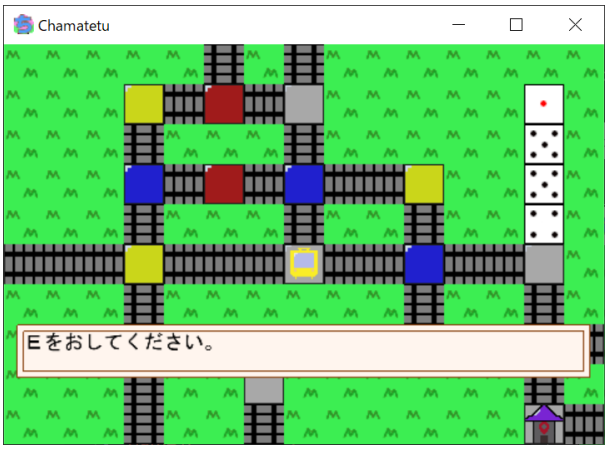

図 35: サイコロをとめたときの画面表示 2

4.17 マス移動および停車駅の判定処理

マスの移動および停車駅の判定の処理について説明する.Display 関数 (リスト 24) では turnstatus=5 および 6 のときの処理である. まず, マス移動の処理である turnstaus=5 の処理について説明する. turnstaus=5

の状態はサイコロの出目の数だけ駅を移動することができる．「マス」と「駅」の違いについて説明する．「マス」は線路を含む移動できる部分のことである．これに対して「駅」は停車できる部分のことである．例として，サイコロの出目で 2 が出たときの動作を説明する．図 36 がサイコロを振って 2 がでた状態である．図 36 から読み取れるように，移動可能なときは，右上のダイアログに残り移動可能な駅の数と目的地設定が表示される．

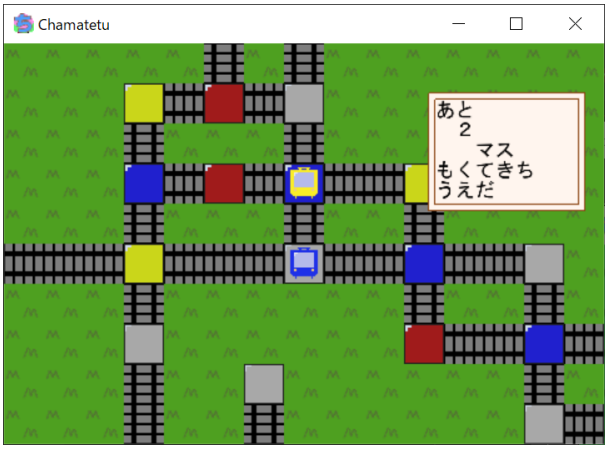

図 36: 出目が 2 の状態

移動は次に示すキーで可能である．図 37 に，図 36 の状態で D キーを押して右に 1 駅移動したときの画面表示を示す．図 37 から，右上のダイアログの残り移動可能な駅の数が 2 から 1 に減少したことが読み取れる．このように，移動に対応して右上のダイアログの値が変化する．ちなみに，図 37 の状態で A キーを押して戻ったときの仕様は図 36 に示す状態になる．ここでは 1 駅しか戻っていないが,2 駅以上戻った場合も同様に残り移動可能な駅の数が上昇する．図 37 の状態で左以外の方向に移動すると，残り移動可能な駅の数が 0 になる．残り移動可能な駅の数が 0 になると,turnstatus=5 から turnstatus=6 の状態になり，どの駅に停車したのか判定され，処理が分岐する．停車した駅ごとの処理は次節以降で述べる．

- 「W」 . . . 上
- 「A」 . . . 左
- 「S」 . . . 下
- 「D」 . . . 右

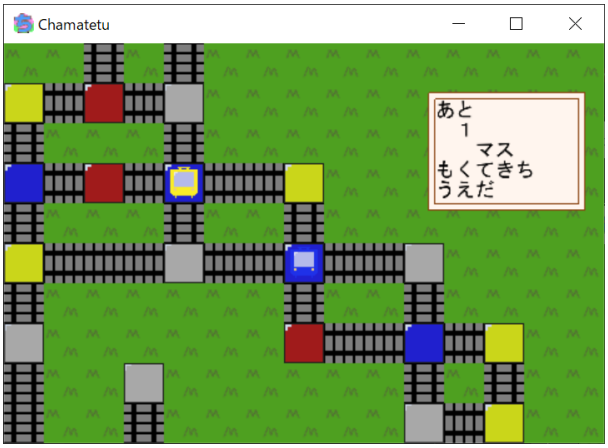

図 37: 右に 1 駅移動した状態

これらの動作の実装部分について説明する．まず, 画面にダイアログを表示する処理について説明する．Display 関数 (リスト 24) では turnstatus=5 のとき moveMass 関数を呼び出している．リスト 41 に moveMass 関数のコードを示す．moveMass 関数の処理は図 36 の右上に表示されるダイアログを生成する処理である．ダイアログの生成は,11 行目および 12 行目で行っている．13 行目から 16 行目では, 変数 recount が 0 になったときに turnstatus を 6 つまり駅の判定に更新する処理を行っている．

リスト 41: moveMass 関数

```
1  // 駅移動処理
2  void moveMass(void){
3      char fname[150];
4      drawMap();
5      drawPlayer();
6      // あと
7      //      hoge
8      //        マス
9      // もくてきち
10     // huga
11     sprintf(fname,"aatoxxss%dxxssssllmasullxxmokutekitixx%s",recount,distination.name);
12     drawText(fname,340,40,125,94,0);
13     if(recount==0){ // 移動マスを消費したらstatus更新
14         turnstatus=6;
15     }
16 }
```

次に移動の処理を行う部分について説明する．まず, キー入力を管理する keyboard 関数の処理について説明する．リスト 42 に turnstatus=5 のときの,keyboard 関数のコードを示す. リスト 42 において, キー入力から, 進む方向を制御しているのは 11 行目から 21 行目である. 進方向はグローバル変数 direction に格納している．変数 direction と値の対応は上が 0, 右が 1, 下が 2, 左が 3, どの方向でもないとき-623 という仕様にしている．

リスト 42: 移動のための入力処理

```
1  // キーボード入力管理
2  void keyboard(unsigned char key,int x,int y){
3      int locktime =500;
4      int transx = players[turn].x/IMGSIZE;
5      int transy = players[turn].y/IMGSIZE;
6  if(keyboardflg==0){ // キーボード入力がロックされていないとき
7
8      (省略)
9
10     }else if(turnstatus==5){ //移動
11         if(key=='w'){ // 上
12             direction=0;
13         }else if(key=='d'){ // 右
14             direction=1;
15         }else if(key=='s'){ // 下
16             direction=2;
17         }else if(key=='a'){ // 左
18             direction=3;
19         }else { // それ以外
20             direction=-623;
21         }
22         if(direction!=-623){
23             if(isMovable(transx,transy)){
24                 // 移動履歴を書き込み
25                 massRecord[randresult-recount][0]=transx;
26                 massRecord[randresult-recount][1]=transy;
27                 // 次の駅を計算
28                 nextStation(transx,transy);
29                 keyboardflg=1; // キーボード入力ロック
30                 glutTimerFunc(MOVETIME, MoveTimer, 0); //移動タイマー起動
31             }
32         }
33     }
34
35     (省略)
36
```

```
37      if(turnstatus!=5){
38          keyboardflg=1; // キーボード入力ロック
39          glutTimerFunc(locktime, keyboardTimer, 0); // ロック解除タイマー
40      }
41  }
42  }
```

キー入力から方向の情報を取得できたから, その方向に移動可能かどうかの判定を行う. この判定はリスト 42 の 23 行目の isMovable 関数で行っている. リスト 43 に isMovable 関数のコードを示す. isMovable 関数は引数として受け取った Map 配列のインデックス (x,y) とグローバル変数 direction の値からその方向に移動できるかどうかを判定する関数である. 探索は, 探索したい方向の Map 配列におけるインデックスを isWall 関数に渡すことで行っている.isWall 関数の真偽によって, 進める場合は 1, 進めない場合は 0 を返す. リスト 44 に isWall 関数のコードを示す. 進むことができるかどうかは, その方向が背景でないことを確認すればよい. このため, isWall 関数は引数として受けとった Map 配列におけるインデックスを getmapnum 関数 (リスト 33) に渡すことで, 進みたい方向が背景かどうかを判別している. 進みたい方向が背景のときは 1, 背景でないときは 0 を返している.

リスト 43: isMovable 関数

```
1   // 0 : 上
2   // 1 : 右
3   // 2 : 下
4   // 3 : 左
5   //
6   // 進めるとき 1
7   // 進めないとき 0
8   int isMovable(int x,int y){
9       if(direction==0){ // 上
10          if(isWall(x,y-1)){
11              return 0;
12          }
13      }else if(direction==1){ //右
14          if(isWall(x+1,y)){
15              return 0;
16          }
17      }else if(direction==2){ //下
18          if(isWall(x,y+1)){
19              return 0;
20          }
21      }else{ //左
22          if(isWall(x-1,y)){
23              return 0;
24          }
25      }
26      return 1;
27  }
```

リスト 44: isWall 関数

```
1   // 行先が壁かどうか判定
2   // 1 : 壁
3   // 0 : 壁でない
4   int isWall(int x,int y){
5       if(getmapnum(x,y)==WALL){
6           return 1;
7       }
8       return 0;
9   }
```

移動可能かどうかの判定の結果, 移動可能であれば次の停車駅の計算と移動履歴の書き込み処理を行う. 移動履歴の書き込みはリスト 42 の 25 行目および 26 行目で行っている. 移動履歴の書き込みはグローバル変数 (配列)massRecord に, 今いる座標を書き込むことで行っている. 書き込まれるインデックスは 1 回目の移動のとき 0,2 回目の移動のとき 1,... というふうになっている. 28 行目では, 次の停車駅を計算する関数である nextStation 関数を実行している. リスト 45 に nextStation 関数のコードを示す. nextStation 関数

では, 変数 direction の示す方向を,1 マスずつ getmapnum 関数を用いて探索することで, 次に止まる駅の座標を計算している. なぜこのような関数が必要かというと, 社長が移動する場所には線路と駅の 2 つがあり, 停車できるのは駅だけである. 次に停車する駅の座標を決めてから移動する処理を行わないと, 移動のたびに駅か線路かを判定する必要があるため処理が重くなってしまう. このため先に停車する座標を計算してから移動の処理を行うようにしている. 次の停車駅の座標はグローバル変数 nx,ny に代入される.

リスト 45: nextStation 関数

```
1  // 次の駅を取得
2  void nextStation(int x,int y){
3      int sx=x;
4      int sy=y;
5      if(direction==0){ // 上
6          while(1){
7              sy--;
8              if(getmapnum(sx,sy)!=SENRO1){ // 駅を発見したら
9                  break;
10             }
11         }
12     }else if(direction==1){ // 右
13         while(1){
14             sx++;
15             if(getmapnum(sx,sy)!=SENRO2){ // 駅を発見したら
16                 break;
17             }
18         }
19     }else if(direction==2){ // 下
20         while(1){
21             sy++;
22             if(getmapnum(sx,sy)!=SENRO1){ // 駅を発見したら
23                 break;
24             }
25         }
26     }else if(direction==3){ // 左
27         while(1){
28             sx--;
29             if(getmapnum(sx,sy)!=SENRO2){ // 駅を発見したら
30                 break;
31             }
32         }
33     }else{ // エラーチェック用
34         sx=-623;
35         sy=-623;
36     }
37     // 次の駅の座標をセット
38     nx=sx;
39     ny=sy;
40 }
```

次の停車駅の計算と移動履歴の書き込み処理が行えたから, 移動の処理を行う. リスト 42 では 30 行目の処理である. turnstatus=5 以外のときは, キーボード入力があると一定時間入力を受け付けない処理を行っていたが, turnstatus=5 のときは移動が完了するまでキーボードの入力をロックする仕様になっている. このため, リスト 42 の 37 行目に示すように, turnstatus=5 のときは例外処理として keyboardTimer 関数を実行しないようになっている.

30 行目でコールバック関数に登録している MoveTimer 関数が, 定数 MOVETIME(100ms) おきに移動処理を行う関数である. リスト 46 に MoveTimer 関数のコードを示す. リスト 46 において, 移動の処理を行っているのは 7 行目の move 関数である. リスト 47 に move 関数の処理を示す. move 関数の処理はターン中の社長の座標を, 移動する方向に定数 MOVESIZE(ここでは 16 に設定している) だけ変化させる処理である.

リスト 46 の 11 行目から 15 行目は,nextStation 関数で計算した次の駅の座標と, ターン中の社長の座標が一致しない, つまり停車する駅についていないときに, タイマーを再度呼び出す処理行っている. この処理によって nextStation 関数で計算した駅の座標に一致するまで移動し続け, 座標が一致すると移動が終了する. nextStation 関数で計算した駅に到着したときの処理はリスト 46 の 17 行目から 32 行目である. 17 行目から 32 行目の処理は残り移動可能な駅の数の再計算である. この計算は, まず完了した移動と移動履歴

を照合して進む, 戻るのどちらの移動が行われたのか判別する． i 回目に戻る移動が行われた場合, 移動履歴を管理している配列 massRecord の i-2 番目に格納されている座標と, いまいる座標が一致するはずである.19 行目および 20 行目ではこれを判定している. 座標が一致している場合は, 変数 recount をインクリメントすることで, 残り移動可能な駅の数を増やしている. 一致しない場合は進む移動が行われているから変数 recount をデクリメントすることで, 残り移動可能な駅の数を減らしている.

残り移動可能な駅の数が 0 のときの処理は 28 行目から 32 行目で行っている. 残り移動可能な駅の数が 0 のとき, キーボードの入力をロックしている. 0 でないときは, 次の移動の入力を受け付けるために, キーボード入力のロックを解除している. これらの処理によって, 駅の移動処理を実装している.

リスト 46: MoveTimer 関数

```
1  // 駅移動管理タイマー
2  void MoveTimer(int t)
3  {
4      int transx;
5      int transy;
6      // 移動処理
7      move();
8      // 座標変換
9      transx = players[turn].x/IMGSIZE;
10     transy = players[turn].y/IMGSIZE;
11     if((transx != nx)||(transy != ny)){ // 次の駅の座標と同じか
12         glutTimerFunc(MOVETIME, MoveTimer, 0); // 同じでないときタイマー継続
13     }else if((players[turn].x%IMGSIZE!=0)||(players[turn].y%IMGSIZE!=0)){
14         // 余りが0でないとき
15         glutTimerFunc(MOVETIME, MoveTimer, 0); // タイマー継続
16     }else{
17         if(randresult-recount>0){ // まだ移動可能かどうか
18         // 移動可能のとき,以前の移動履歴をチェック
19             if((massRecord[randresult-recount-1][0]==transx)&&
20             (massRecord[randresult-recount-1][1]==transy)){
21                 recount++; // 戻ったとき残りカウント数増加
22             }else{
23                 recount--; // 進んだとき残りカウント数減少
24             }
25         }else{
26             recount--; // カウント減少
27         }
28         if(recount==0){ // カウント終了のときキーボード入力をロック
29             keyboardflg=1;
30         }else{ // キーボード入力のロックを解除
31             keyboardflg=0;
32         }
33     }
34 }
```

リスト 47: move 関数

```
1  // 移動処理
2  void move(void){
3      if(direction==0){ // 上
4          players[turn].y-=MOVESIZE;
5      }
6      if(direction==1){ // 右
7          players[turn].x+=MOVESIZE;
8      }
9      if(direction==2){ // 下
10         players[turn].y+=MOVESIZE;
11     }
12     if(direction==3){ // 左
13         players[turn].x-=MOVESIZE;
14     }
15 }
```

次に turnstatus=6 のときの処理について説明する. turnstatus=6 のときの処理は停車した駅を判定して処理を分岐する処理である. Display 関数 (リスト 24) では turnstatus=6 のとき, checkMass 関数を実行している. リスト 48 に checkMass 関数のコードを示す. リスト 48 では inflg=0 のとき, 社長が停車した駅の

種類を getmapnum 関数で取得し, 次に行う処理のために inflg および turnstatus を更新する処理を行っている. 12 行目に示すように getmapnum 関数の返り値はローカル変数 st に代入している.13 行目から 27 行目では, この値を駅を表す定数と比較することで, 駅ごとに処理を分けている. 物件駅に停車したときの処理は 13 行目から 20 行目である. 物件駅に停車したときは, まずその駅が目的駅かどうかを判定している. 目的地のときは, 変数 goalflg を 1 にして, ターン中の社長の所持金をプラス 3 億円する処理を行っている. そして,inflg をインクリメントしている.inflg=1 のとき図 38 に示すように, 画面に「(社長名) しゃちょうが 1 ばんのり, おめでとうございます.(社長名) しゃちょうにプラス 3 億円.」と表示される. 図 38 の状態で E キーを押すと inflg インクリメントされ,inflg=2 の処理が行われる. inflg=2 の処理は turnstatus を 1 に更新する処理である. turnstatus=1 の処理は目的地の設定を行う処理である. これによって目的地に到着したときに目的地が再設定される.

停車した駅がプラス駅, マイナス駅, カード駅のときの turnstatus の更新はリスト 48 の 21 行目から 27 行目で行っている. これらの処理によって, 停車した駅を判定して, 処理を分岐させている.

リスト 48: checkMass 関数

```
1  // 停車駅の判定と処理の分岐
2  void checkMass(void){
3      int st; //止まった駅の番号を保持
4      int transx,transy; // プレイヤーの座標変換用
5      char fname[200];
6      drawMap();
7      drawPlayer();
8      if(inflg==0){
9          keyboardflg=0; // キーボードロック解除
10         transx = players[turn].x/IMGSIZE;
11         transy = players[turn].y/IMGSIZE;
12         st = getmapnum(transx,transy);
13         if(st==PROPERTYMASU){ // 物件に止まったとき
14             if((transx == distination.x)&&(transy == distination.y)){ // 目的地なら
15                 goalflg=1; // ゴールフラグをたてる
16                 players[turn].money+=30000; // プラス3億円
17                 inflg++;
18             }else{ //目的地でないなら
19                 turnstatus=7;
20             }
21         }else if(st==PLUSMASU){ // プラス駅に止まったとき
22             turnstatus=8;
23         }else if(st==MINUSMASU){ // マイナス駅に止まったとき
24             turnstatus=9;
25         }else if(st==CARDMASU){ // マイナス駅に止まったとき
26             turnstatus=10;
27         }
28     }else if(inflg==1){
29         // hogeしゃちょうがhugaに1ばんのり.おめでとうございます.
30         // hogeしゃちょうにプラス3億円.
31         sprintf(fname,"%ssilatilouuga%sni1bannnorimroomedetouugozaiimasumr%s
32         silatilouunillpurasull3oxexmr",players[turn].name,distination.name,
33         players[turn].name);
34         drawText(fname,11,225,InitWidth-22,42,0);
35     }else if(inflg==2){
36         inflg=0;
37         turnstatus=1; // 目的地再設定
38     }
39 }
```

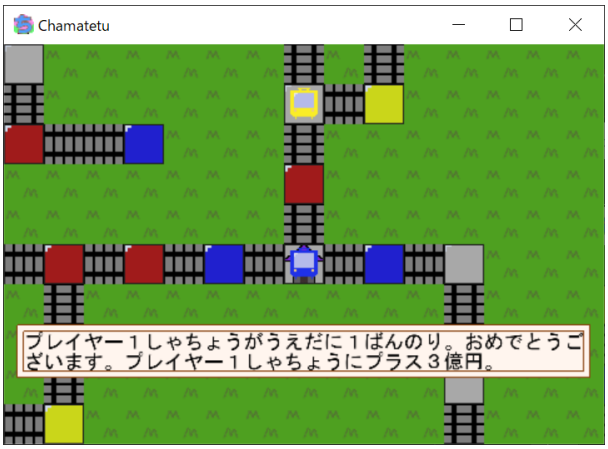

図 38: 目的地に到着したときの表示

4.18 物件駅の処理

物件駅に停車したときの処理について説明する．まず, 物件駅に停車したときの動作について説明する．物件駅に停車すると図 39 に示すように物件情報, セレクトポジション,「Q しゅうりょう E こうにゅう」というダイアログの 3 つが表示される．図 39 では「W」キーおよび「S」キーで購入する物件を選択することができ, 所持金が足りる場合は「E」キーを押すことで物件を購入することができる．また「Q」キーを押すことで物件の購入を終了できる．図 40 に「りんごえん」を購入した時の画面表示を示す．図 40 から「りんごえん」を購入すると, 所持金が購入した物件の価格分引かれることがわかる．またレポートでは伝わらないが購入した物件は背景がプレイヤーカラーで表示される．また, 誰かが購入した物件および所持金では購入できない物件は赤文字で表示される．レポートでは文字色の変化は伝わらないため画像は省略する．

図 39: 物件購入画面 (長野駅)

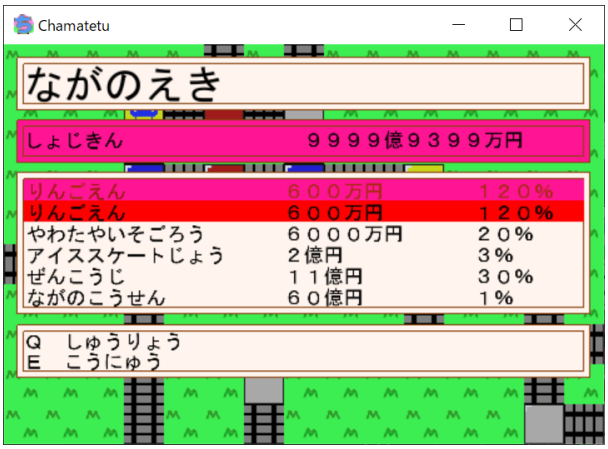

図 40: 「りんごえん」を購入した場合の画面表示

物件を独占したときの画面表示について確認する. 図 41 に長野駅を独占したときの画面表示を占めす. 図 41 から読み取れるように, 独占を行うと「(社長名) しゃちょうのどくせんです.」というダイアログが表示される.

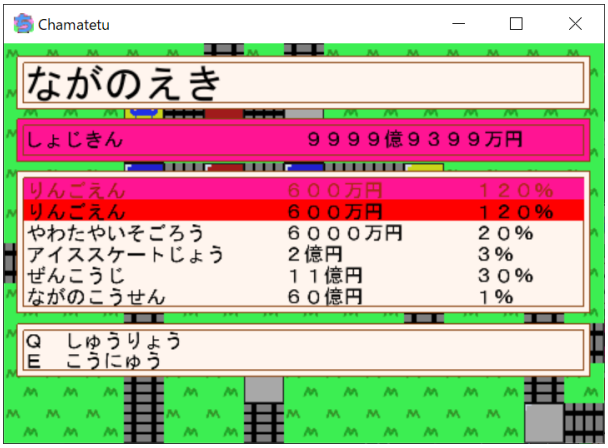

図 41: 独占したときの画面表示

これらを実装しているコードについて説明する. 物件の購入は,Display 関数 (リスト 24) では turnstatus=7 のときの処理である. Display 関数では turnstatus=7 のとき, purchaseProperty 関数を実行している. リスト 49 に purchaseProperty 関数のコードを示す. また, リスト 50 に turnstatus=7 のときの keyboard 関数のコードを示す.

リスト 49: purchaseProperty 関数

```
// 物件購入処理
void purchaseProperty(void){
    drawMap();
    drawPlayer();
    if(inflg==0){
        keyboardflg=0;
        selectpos=0;
        inflg++;
    }else if(inflg==1){
        drawStation(); // 物件情報描画
    }else if(inflg==2){
        inflg=0;
        turnstatus=15;
    }
```

```
}
```

リスト 50: 物件購入時のキーボード入力の処理

```
// キーボード入力管理
void keyboard(unsigned char key,int x,int y){
    int locktime =500;
    int transx = players[turn].x/IMGSIZE;
    int transy = players[turn].y/IMGSIZE;
if(keyboardflg==0){ // キーボード入力がロックされていないとき

    (省略)

    }else if(turnstatus==7){ // 物件購入
        locktime=200;
        if(key=='s'){ // ポジションを下へ
            if(selectpos<propertynum-1){
                selectpos+=1;
            }
        }else if(key=='w'){ // ポジションを上へ
            if(selectpos>=1){
                selectpos-=1;
            }
        }else if(isE(key)){ // 物件購入
            if(ispurchase(selectpos)){
                purchase(selectpos);
            }
        }else if(key=='q'){ // 購入終了
                inflg++;
        }
        }

        (省略)

        if(turnstatus!=5){
            keyboardflg=1; // キーボード入力ロック
            glutTimerFunc(locktime, keyboardTimer, 0); // ロック解除タイマー
        }
}
}
```

purchaseProperty 関数では,inflg=1 のときセレクトポジションを管理する変数 selectpos を 0 に設定している. これによって, 物件購入画面が表示されたときにセレクトポジションが最上部の物件を選択するようになっている. inflg=2 のときは,drawStation 関数を実行して物件の情報およびダイアログを表示している. リスト 51 に drawStation 関数のコードを示す. drawStation 関数では, まず停車した駅の座標から, どの物件駅に停車したのかを判別している. この処理はリスト 51 の 10 行目から 18 行目で行っている. 駅の判別が行えたから, 画面に物件の情報を描画する処理を行う. 21 行目から 24 行目で駅名,27 行目から 31 行目で所持金の描画を行っている. 物件の表示は 34 行目から 64 行目で行っている. 所持している物件をプレイヤーカラーの背景で描画しているのは 43 行目から 47 行目の処理である. セレクトポジションを描画しているのは 50 行目から 53 行目での処理である. ターンスタート時の処理と同様に keyboard 関数 (リスト 50) の 12 行目から 19 行目でキー入力に合わせて変数 selectpos を更新し,50 行目から 53 行目で変数 selectops を描画する処理を行うことでセレクトポジションを表示している.

リスト 51: drawStation 関数

```
 // 物件情報を描画
void drawStation(void){
    char fname[100];
    int i,j;
    int holder;
    int color;
    int transx = players[turn].x/IMGSIZE;
    int transy = players[turn].y/IMGSIZE;
    // どの駅か識別
    for(i=0;i<STATIONNUM;i++){
```

```
11          // 駅の座標が一致したら
12           if((stations[i].x==transx)&&(stations[i].y==transy)){
13              // 物件数を取得
14              propertynum = stations[i].propertynum;
15              // 配列番号を取得
16              stid = i;
17           }
18       }
19
20      // 駅名表示
21      glColor3ub(255,245,238);
22      drawDialog(11,11,InitWidth-22,42);
23      sprintf(fname,"%seeki",stations[stid].name);
24      drawString(fname,0,16,16,1);
25
26      // 所持金表示
27      glColor3ub(playercolor[turn][0],playercolor[turn][1],
28      playercolor[turn][2]);
29      drawDialog(11,61,InitWidth-22,34);
30      drawString("silozikinn",0,16,61+8,0.5);
31      drawMoney(players[turn].money,2*InitWidth/4,61+8,0,0.5);
32
33      // 物件表示
34      glColor3ub(255,245,238);
35      drawDialog(11,103,InitWidth-22,11+17*stations[stid].propertynum);
36      for(j=0;j<propertynum;j++){
37
38          // 収益率を文字列に変換
39          sprintf(fname,"%dpx",stations[stid].plist[j].earnings);
40
41          // 物件の所有者がいるとき,所有者カラーで物件を囲む
42          holder = stations[stid].plist[j].holder;
43          if(holder!=0){
44              glColor3ub(playercolor[holder-1][0],playercolor[holder-1][1],
45              playercolor[holder-1][2]);
46              drawQUAD(16,108+j*17,InitWidth-32,17);
47          }
48
49          // セレクトポジションを表示
50          if(selectpos == j){
51              glColor3ub(255,0,0);
52              drawQUAD(16,108+j*17,InitWidth-32,17);
53          }
54          // 物件の表示色設定
55          if(ispurchase(j)){
56              color=0;
57          }else{
58              color=1;
59          }
60          // 物件を表示
61          drawString(stations[stid].plist[j].name,color,18,42+11+50+7+17*j,0.5);
62          drawMoney(stations[stid].plist[j].price,InitWidth/2-16,42+11+50+7+17*j,color,0.5);
63          drawString(fname,color,16+3*InitWidth/4,42+11+50+7+17*j,0.5);
64      }
65      // 独占ダイアログ表示
66      if(stations[stid].ismonopoly!=0){
67          sprintf(fname,"%ssilatilouunodokusenndesumr",
68          players[stations[stid].ismonopoly-1].name);
69          drawText(fname,11,223,InitWidth-22,32,0);
70          sprintf(fname,"xqsssiluuurilouuxxxesskouuniluuu");
71          drawText(fname,11,273,InitWidth-22,42,0);
72      }else{ // 操作ダイアログ表示
73              sprintf(fname,"xqsssiluuurilouuxxxesskouuniluuu");
74              drawText(fname,11,225,InitWidth-22,42,0);
75      }
76  }
```

物件が購入できるとき黒字, 購入できないとき赤字で表示する処理はリスト 51 の 55 行目から 59 行目で行っている．55 行目では ispurchase 関数という関数を用いて物件が購入できるかどうかを判定している．リスト 52 に ispurchase 関数のコードを示す．ispurchase 関数では物件が購入できるかどうかを, 既に誰かが購入しているケースと値段が足りないケースの 2 つに分けて判定している．既に誰かの物件かどうかは

playerstatus 構造体の holder メンバに記述されているからこれを確認している. 値段が足りるかどうかは, ターン中の社長の所持金 money と購入しようとしている物件の値段を比較することで行っている.

リスト 52: ispurchase 関数

```
// 物件が購入できるか取得
// 1 : 取得可能
// 0 : 取得不可能
int ispurchase(int id){
    int flg=1;
    // 既に誰かの物件のとき
    if(stations[stid].plist[id].holder!=0){
        flg=0;
    }
    // 値段が足りないとき
    if(players[turn].money < stations[stid].plist[id].price){
        flg=0;
    }
    return flg;
}
```

独占ダイアログおよび画面下部の表示はリスト 51 の 66 行目から 75 行目の処理で行っている. 独占かどうかは stations 構造体のメンバ ismonopoly に記述されているから, これを判定して独占ダイアログを表示/非表示にする処理を行っている.

次に購入の処理について説明する. 購入は E キーが押されたときに行われる. リスト 50 の 20 行目から 23 行目が E キーを押されたときの処理である. E キーが押されたとき, 購入可能かどうかを ispurchase 関数で確認して, 購入可能であれば購入処理を行う purchase を実行している. リスト 53 に purchase 関数のコードを示す. purchase 関数の 6 行目から 10 行目では, 総資産, 所持金, 購入フラグの 3 つを更新している. 12 行目から 21 行目では, 独占したかどうかのチェックを行っている. 独占したかどうかは, 同一駅内の全ての物件の購入フラグがターン中の社長のものと一致するかどうかを確認すれば判定できる. このため,for 文を用いて物件の購入フラグをチェックしている. 独占のときは独占フラグを立てる処理を行っている. これらの処理によって物件を購入する処理を実装している.

リスト 53: purchase 関数

```
//物件購入処理
void purchase(int id){
    int i;
    int monopolyCheck=0; // 独占チェック用
    // 総資産を計算
    players[turn].assets+=stations[stid].plist[selectpos].price;
    // 所持金を計算
    players[turn].money-=stations[stid].plist[selectpos].price;
    // 購入済みフラグをたてる
    stations[stid].plist[selectpos].holder=turn+1;
    // 独占チェック
    for(i=0;i<propertynum;i++){
        if(stations[stid].plist[i].holder==turn+1){
            monopolyCheck++;
        }
    }
    // 独占のとき
    if(monopolyCheck==propertynum){
        // 独占フラグをたてる
        stations[stid].ismonopoly=turn+1;
    }
}
```

物件購入画面で Q キーを押してたときの処理について説明する. リスト 50 の 24 行目から 26 行目に示すように, Q キーを押すと inflg がインクリメントされ,inflg=2 の処理が行われる. リスト 49 の 13 行目に示すように, inflg=2 のとき turnstatus を 15, つまりターンの終了処理に更新している. これによって, Q キーを押すと物件購入処理が終了する.

4.19　プラス駅の処理

プラス駅に停車したときの処理について説明する. まず, プラス駅に停車したときの動作について説明する. 図 42 にプラス駅に停車したときの画面表示を示す. 図 42 では「2349 万円」と表示されている部分は, 実際には 100ms おきにランダムな金額が表示されている. 図 42 の状態で E キーを押すと図 43 のようにランダムな表示が停止して所持金が表示される. 図 43 の状態で E キーを押すとターンが終了する.

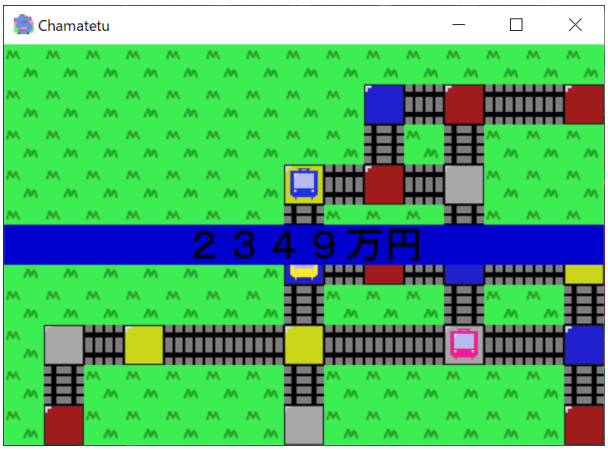

図 42: プラス駅に停車したときの画面表示

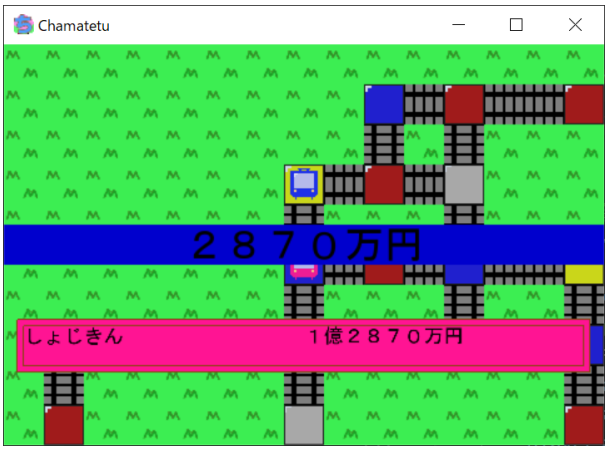

図 43: 図 42 の状態で E キーを押したときの画面表示

この動作を実行しているのが turnstatus=8 のときの処理である. Display 関数 (リスト 24) では turnstatus=8 のとき plusMass 関数を実行している. リスト 54 に plusMass 関数のコードを示す. リスト 54 では inflg=0 のときダミータイマーを起動する処理を行っている. 乱数の生成範囲は pulsarray に最小値, 最大値という形で記述されている. 例えば 4 月の場合 2000 万円から 4000 万円の範囲の乱数が生成される. 範囲は月ごとに乱数生成に変えることで, 季節ごとにもらえる金額が変化する仕様を実装している. inflg=1 のときはダミーの乱数の生成結果を画面に出力する処理を行っている. inflg=1 から inflg=2 の状態への更新は E キーの入力によって行われる. turnstatus=8 のときのキーボードの処理は E キーの入力で inflg をインクリメントするだけであるから省略する. inflg=2 では本当の結果を計算する処理を行って inflg をインクリメントする処理を行っている. inflg=3 のときは本当の結果と所持金を表示するダイアログを表示する処理を行っている. これによって図 43 に示す画面表示が行われている. inflg=3 の状態で E キーが押されると inflg=4 の処理が行われる. inflg=4 のとき,turnstatus を 15, つまりターンの終了処理を行うように更新している. これらの処理によってプラス駅の処理を実装している.

リスト 54: plusMass 関数

```
// プラス駅の色
int pluscolor[3] = {0,0,205};

// プラス駅の月別乱数表
int plusarray[MAXMONTH][2] = {{400,1200}, // 1月
                              {200,800}, // 2月
                              {400,1500}, // 3月
                              {2000,4000}, // 4月
                              {3000,7000}, // 5月
                              {4000,8000}, // 6月
                              {12000,30000}, // 7月
                              {35000,70000}, // 8月
                              {18000,32000}, // 9月
                              {6000,12000}, // 10月
                              {3000,7000}, // 11月
                              {2000,4000}}; // 12月

// プラス駅の処理
void plusMass(){
    char fname[150];
    drawMap();
    drawPlayer();
    if(inflg==0){
        dummynum=1;
        // ダミータイマー起動
        dummyresult[0]=0;
        keyboardflg=0;
        range=plusarray[month-1][1]-plusarray[month-1][0];
        randflg=1;
        glutTimerFunc(RANDTIME, RandTimer, 0);
        inflg++;
    }else if(inflg==1){
        // ダミー出力
        glColor3ub(pluscolor[0],pluscolor[1],pluscolor[2]);
        drawQUAD(0,InitHeight/2-16,InitWidth,IMGSIZE);
        drawMoney(plusarray[month-1][0]+dummyresult[0],InitWidth/2-IMGSIZE*3,
        InitHeight/2-16,0,1);
    }else if(inflg==2){
        // 結果を計算
        randflg=0;
        randresult = rand()%range;
        tmpmoney = plusarray[month-1][0]+randresult;
        players[turn].money+=tmpmoney;
        inflg++;
    }else if(inflg==3){
        // 所持金ダイアログ表示
        glColor3ub(pluscolor[0],pluscolor[1],pluscolor[2]);
        drawQUAD(0,InitHeight/2-16,InitWidth,IMGSIZE);
        drawMoney(tmpmoney,InitWidth/2-IMGSIZE*3,InitHeight/2-16,0,1);
        // 所持金表示
        glColor3ub(playercolor[turn][0],playercolor[turn][1],playercolor[turn][2]);
        drawDialog(11,220,InitWidth-22,42);
        drawMoney(players[turn].money,InitWidth/2,225,0,0.5);
        // しょじきん
        sprintf(fname,"silozikinn");
        drawString(fname,0,16,225,0.5);
    }else if(inflg==4){
        inflg=0;
        turnstatus=15;
    }
}
```

4.20 マイナス駅および借金の処理

マイナス駅および借金時の処理について説明する. まず, マイナス駅に停車したときの動作を確認する. 図 44 にマイナス駅に停車したときの画面表示を示す. 図 44 で「-1339 万円」と表示されている部分は, 実際には 100ms ときにランダムな金額が表示されている. 図 44 の状態で E キーを押すと図 45 のようにランダム

な表示が停止して所持金が表示される．所持金が負の値でない場合は, 図 45 の状態で E キーを押すとターンが終了する．

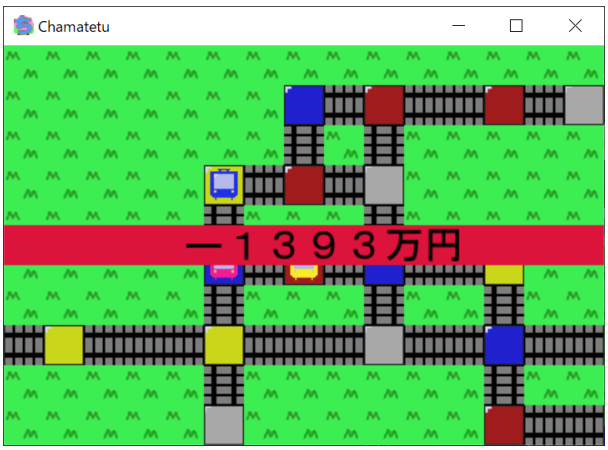

図 44: マイナス駅に停車したときの画面表示

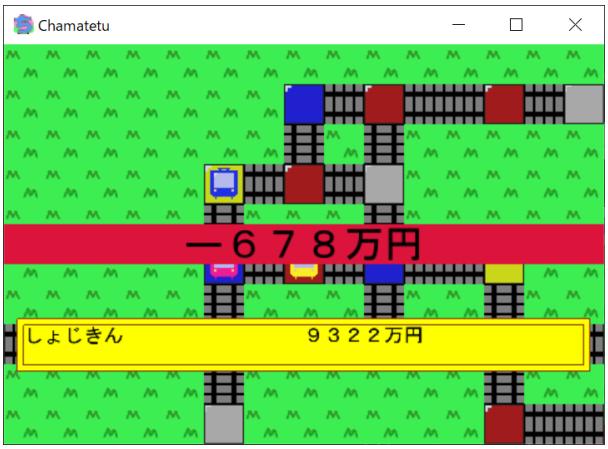

図 45: 図 44 の状態で E キーを押したときの画面表示

次に, 借金を背負ったときの動作について確認する. 借金を背負ったときの動作は物件を持っているかどうかで変化する. まず, 物件を一つも処理していないときの動作を確認する. なお, 確認のためにマイナス駅で生成される乱数の値をプログラムで事前に打ち込んでいる. 図 46 に物件を一つも持っていないときにマイナス駅に停車して,-19 億円の借金を負ったときの画面表示を示す. 図 46 の状態で E キーを押すと, 図 47 に示すように「うれるぶっけんがありません. しゃっきんをせおってしまいました.」というダイアログが表示される．図 47 の状態で E キーを押すとターンが終了する．

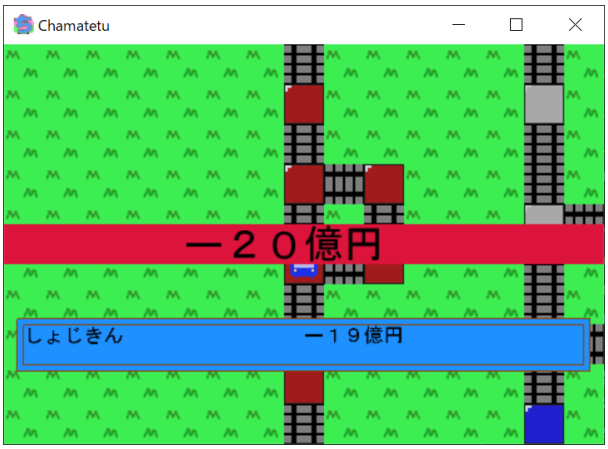

図 46: -19 億円の借金を背負った状態

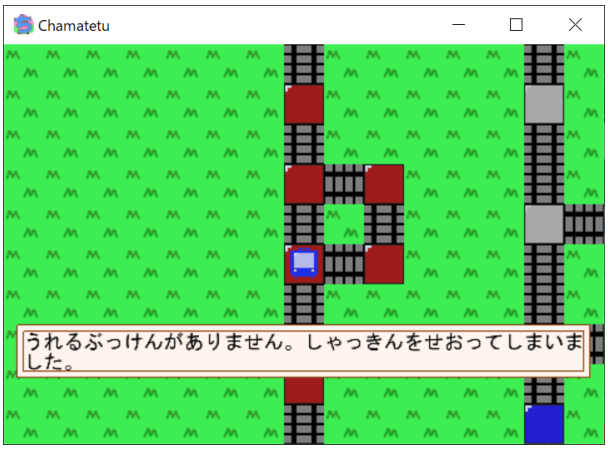

図 47: 売れる物件がないというダイアログの表示

次に, 物件を持っているときの動作を確認する. いま, ある社長が図??に示すように小布施駅の物件を 5 つ購入し, 所持金が 4400 万円である状態にあるとする. この社長の総資産は 5600 万円である. 図??で物件を購入した次のターンに, マイナス駅で所持金から 20 億円引かれるときの画面表示を図 49 に示す. 図 49 から,「しゃっきんをせおってしまいました.5 けんのぶっけんをばいきゃくします.」と表示されていることがわかる. この場合, 総資産 5600 万円に対して借金額が 19 億 4600 万円であるから, すべての物件を売却して, かつ 19 億円の借金が所持金に残る. このため, 小布施で購入した 5 件の物件がすべて売却されている. 図 49 の次のターンに所持金を確認すると図 50 に示すように-19 億円になっている.

図 48: 小布施の物件を購入した状態

図 49: 売却した物件数の表示

図 50: 所持金が負になっていることの確認

借金をしたが, 物件を売却することで借金が回復する場合についても確認する. 先の例と同様に小布施の物件を 5 つ購入し, マイナス駅で 600 万円の借金をした状態であるとする. 図 51 にこの状態で借金を返済する処理を行ったときの画面表示を示す. 図 51 から「しゃっきんをせおってしまいました.1 けんのぶっけんをばいきゃくします.」という表示が行われていることがわかる. 実際に小布施駅に戻って売却された物件を確

認すると, 図 52 に示すように借金額 600 万円に最も近い 600 万円の北斎館が売却されていることがわかる. 物件の売却の優先順位は!で説明した通りであるためこの場合北斎館が売却される.

図 51: 借金を返済したときの画面表示

図 52: 物件が購入済みでなくなっていることの確認

これらを実装しているのが turnstatus=9 のときの処理である. Display 関数 (リスト 24) で turnstatus=9 のとき minusMass 関数を実行している. リスト 55 に minusMass 関数のコードを示す. turnstatus=9 のときのキーボードの処理は E キーの入力で inflg をインクリメントするだけであるから省略する.

リスト 55: minusMass 関数

```
// マイナス駅の色
int minuscolor[3] = {220,20,60};

// マイナス駅の月別乱数表
int minusarray[MAXMONTH][2] = {{10000,25000}, // 1月
                               {20000,80000}, // 2月
                               {10000,25000}, // 3月
                               {500,1500}, // 4月
                               {400,1200}, // 5月
                               {300,1000}, // 6月
                               {200,600}, // 7月
                               {100,400}, // 8月
                               {500,1500}, // 9月
                               {2000,4000}, // 10月
                               {3000,8000}, // 11月
                               {4000,9000}}; // 12月

// マイナス駅の処理
```

```
19 void minusMass(void){
20     char fname[150];
21     drawMap();
22     drawPlayer();
23     if(inflg==0){
24         // ダミータイマー起動
25         dummynum=1;
26         dummyresult[0]=0;
27         keyboardflg=0;
28         range=minusarray[month-1][1]-minusarray[month-1][0];
29         randflg=1;
30         glutTimerFunc(RANDTIME, RandTimer, 0);
31         inflg=1;
32     }else if(inflg==1){
33         // ダミー出力
34         glColor3ub(minuscolor[0],minuscolor[1],minuscolor[2]);
35         drawQUAD(0,InitHeight/2-16,InitWidth,IMGSIZE);
36         drawMoney(-minusarray[month-1][0]-dummyresult[0],
37         InitWidth/2-IMGSIZE*3,InitHeight/2-16,0,1);
38     }else if(inflg==2){
39         // 結果を計算
40         randflg=0;
41         randresult = rand()%range;
42         tmpmoney = -minusarray[month-1][0]-randresult;
43         players[turn].money+=tmpmoney;
44         inflg++;
45     }else if(inflg==3){
46         // 所持金ダイアログ表示
47         glColor3ub(minuscolor[0],minuscolor[1],minuscolor[2]);
48         drawQUAD(0,InitHeight/2-16,InitWidth,IMGSIZE);
49         drawMoney(tmpmoney,InitWidth/2-IMGSIZE*3,InitHeight/2-16,0,1);
50         // 所持金表示
51         glColor3ub(playercolor[turn][0],playercolor[turn][1],playercolor[turn][2]);
52         drawDialog(11,220,InitWidth-22,42);
53         drawMoney(players[turn].money,InitWidth/2,225,0,0.5);
54         sprintf(fname,"silozikinn");
55         drawString(fname,0,16,225,0.5);
56     }else if(inflg==4){
57         if(players[turn].money<0){ //借金を背負ったとき
58             inflg++;
59         }else{ // それ以外
60             inflg=0;
61             turnstatus=15;
62         }
63     }else if(inflg==5){ // 借金返済処理
64         rdebet = debtprocess(); // 売却した物件数を取得
65         inflg++;
66     }else if(inflg==6){
67         if(rdebet==-1){
68             // うれるぶっけんがありません. しゃっきんをせおってしまいました.
69             sprintf(fname,"uurerubultkenngaaarimasennmr
70             silaltkinnwoseoolttesimaiimasitamr");
71         }else{
72             // しゃっきんをせおってしまいました. hogeけんのぶっけんをばいきゃくしました.
73             sprintf(fname,"silaltkinnwoseoolttesimaiimasitamr
74             %dkennnobultkennwobaiikilakusimasitamr",rdebet);
75         }
76         drawText(fname,11,225,InitWidth-22,42,0);
77     }else if(inflg==7){
78         inflg=0;
79         turnstatus=15;
80     }
81 }
```

inflg=1 から inflg=4 までの処理はプラス駅の場合とほぼ同じである. プラス駅との違いは, 符号の扱いである. RandTimer 関数は 0 以上の乱数しか生成できない. このためマイナス駅での月別の乱数の範囲は,minusarray でマイナスされる額の絶対値を保持している. そして, 生成した乱数を-1 倍することでマイナスする額を計算している.

プラス駅の処理と異なるのは inflg=4 以降の処理である. inflg=4 のときターン中の社長の所持金を確認して, 所持金が 0 未満, つまり借金を負ったときは,inflg=5 の処理を行うようにしている. 借金を負っていな

いときは turnstatus=15, つまりターンの終わりの処理を行うように turnstatus を更新している.
　inflg=5 および 6 のとき, 借金を返済する処理を行い, 結果を画面に表示する処理を行っている.

4.21　カード駅の処理

4.22　ターン終了時の処理

4.23　決算および最終成績の処理

5　付録 ソースコード

参考文献

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