```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import pygame
from pygame.locals import *
import codecs
import os
import random
import struct
import sys

SCR_RECT = Rect(0, 0, 640, 480)
GS = 32
DOWN, LEFT, RIGHT, UP = 0, 1, 2, 3
STOP, MOVE = 0, 1  # 移動タイプ
PROB_MOVE = 0.005  # 移動確率
TRANS_COLOR = (190, 179, 145)  # マップチップの透明色

sounds = {}  # サウンド

TITLE, FIELD, TALK, COMMAND = range(4)


def load_image(dir, file, colorkey=None):
    file = os.path.join(dir, file)
    try:
        image = pygame.image.load(file)
    except pygame.error, message:
        print "Cannot load image:", file
        raise SystemExit, message
    image = image.convert()
    if colorkey is not None:
        if colorkey is -1:
            colorkey = image.get_at((0, 0))
        image.set_colorkey(colorkey, RLEACCEL)
    return image


def split_image(image):
    """128x128のキャラクターイメージを32x32の16枚のイメージに分割
    分割したイメージを格納したリストを返す"""
    imageList = []
    for i in range(0, 128, GS):
        for j in range(0, 128, GS):
            surface = pygame.Surface((GS, GS))
            surface.blit(image, (0, 0), (j, i, GS, GS))
            surface.set_colorkey(surface.get_at((0, 0)), RLEACCEL)
            surface.convert()
            imageList.append(surface)
    return imageList


class PyRPG:
    def __init__(self):
        pygame.init()
        # フルスクリーン化 + Hardware Surface使用
        self.screen = pygame.display.set_mode(
            SCR_RECT.size, DOUBLEBUF | HWSURFACE | FULLSCREEN)
        pygame.display.set_caption(u"PyRPG 26 ゲーム状態の導入")
        # サウンドをロード
        self.load_sounds("data", "sound.dat")
        # キャラクターチップをロード
        self.load_charachips("data", "charachip.dat")
        # マップチップをロード
        self.load_mapchips("data", "mapchip.dat")
        # パーティの作成
        self.party = Party()
        player1 = Player("swordman_female", (3, 5), DOWN, True, self.party)
        player2 = Player("elf_female2", (3, 4), DOWN, False, self.party)
        player3 = Player("priestess", (3, 3), DOWN, False, self.party)
        player4 = Player("magician_female", (3, 2), DOWN, False, self.party)
        self.party.add(player1)
        self.party.add(player2)
```

```
        self.party.add(player3)
        self.party.add(player4)
        # マップの作成
        self.map = Map("field", self.party)
        # メッセージエンジン
        self.msg_engine = MessageEngine()
        # メッセージウィンドウ
        self.msgwnd = MessageWindow(Rect(140, 334, 360, 140), self.msg_engine)
        # コマンドウィンドウ
        self.cmdwnd = CommandWindow(Rect(16, 16, 216, 160), self.msg_engine)
        # タイトル画面
        self.title = Title(self.msg_engine)
        # メインループを起動
        self.game_state = TITLE
        self.mainloop()

    def mainloop(self):
        """メインループ"""
        clock = pygame.time.Clock()
        while True:
            clock.tick(60)
            self.update()             # ゲーム状態の更新
            self.render()             # ゲームオブジェクトのレンダリング
            pygame.display.update()  # 画面に描画
            self.check_event()        # イベントハンドラ

    def update(self):
        """ゲーム状態の更新"""
        if self.game_state == TITLE:
            self.title.update()
        elif self.game_state == FIELD:
            self.map.update()
            self.party.update(self.map)
        elif self.game_state == TALK:
            self.msgwnd.update()

    def render(self):
        """ゲームオブジェクトのレンダリング"""
        if self.game_state == TITLE:
            self.title.draw(self.screen)
        elif self.game_state == FIELD or self.game_state == TALK or self.game_state == COMMAND:
            offset = self.calc_offset(self.party.member[0])
            self.map.draw(self.screen, offset)
            self.party.draw(self.screen, offset)
            self.msgwnd.draw(self.screen)
            self.cmdwnd.draw(self.screen)
            self.show_info()  # デバッグ情報を画面に表示

    def check_event(self):
        """イベントハンドラ"""
        for event in pygame.event.get():
            if event.type == QUIT:
                pygame.quit()
                sys.exit()
            if event.type == KEYDOWN and event.key == K_ESCAPE:
                pygame.quit()
                sys.exit()
            # 表示されているウィンドウに応じてイベントハンドラを変更
            if self.game_state == TITLE:
                self.title_handler(event)
            elif self.game_state == FIELD:
                self.field_handler(event)
            elif self.game_state == COMMAND:
                self.cmd_handler(event)
            elif self.game_state == TALK:
                self.talk_handler(event)

    def title_handler(self, event):
        """タイトル画面のイベントハンドラ"""
        if event.type == KEYUP and event.key == K_UP:
            self.title.menu -= 1
            if self.title.menu < 0:
```

```
                self.title.menu = 0
            elif event.type == KEYDOWN and event.key == K_DOWN:
                self.title.menu += 1
                if self.title.menu > 2:
                    self.title.menu = 2
            if event.type == KEYDOWN and event.key == K_SPACE:
                sounds["pi"].play()
                if self.title.menu == Title.START:
                    self.game_state = FIELD
                    self.map.create("field")  # フィールドマップへ
                elif self.title.menu == Title.CONTINUE:
                    pass
                elif self.title.menu == Title.EXIT:
                    pygame.quit()
                    sys.exit()

        def field_handler(self, event):
            """フィールド画面のイベントハンドラ"""
            # スペースキーでコマンドウィンドウ表示
            if event.type == KEYDOWN and event.key == K_SPACE:
                sounds["pi"].play()
                self.cmdwnd.show()
                self.game_state = COMMAND

        def cmd_handler(self, event):
            """コマンドウィンドウが開いているときのイベントハンドラ"""
            player = self.party.member[0]  # 先頭プレイヤー
            # 矢印キーでコマンド選択
            if event.type == KEYDOWN and event.key == K_LEFT:
                if self.cmdwnd.command <= 3:
                    return
                self.cmdwnd.command -= 4
            elif event.type == KEYDOWN and event.key == K_RIGHT:
                if self.cmdwnd.command >= 4:
                    return
                self.cmdwnd.command += 4
            elif event.type == KEYUP and event.key == K_UP:
                if self.cmdwnd.command == 0 or self.cmdwnd.command == 4:
                    return
                self.cmdwnd.command -= 1
            elif event.type == KEYDOWN and event.key == K_DOWN:
                if self.cmdwnd.command == 3 or self.cmdwnd.command == 7:
                    return
                self.cmdwnd.command += 1
            # スペースキーでコマンド実行
            if event.type == KEYDOWN and event.key == K_SPACE:
                if self.cmdwnd.command == CommandWindow.TALK:  # はなす
                    sounds["pi"].play()
                    self.cmdwnd.hide()
                    chara = player.talk(self.map)
                    if chara != None:
                        self.msgwnd.set(chara.message)
                        self.game_state = TALK
                    else:
                        self.msgwnd.set(u"そのほうこうには　だれもいない。")
                        self.game_state = TALK
                elif self.cmdwnd.command == CommandWindow.STATUS:  # つよさ
                    # TODO: ステータスウィンドウ表示
                    sounds["pi"].play()
                    self.cmdwnd.hide()
                    self.msgwnd.set(u"つよさウィンドウが　ひらくよてい。")
                    self.game_state = TALK
                elif self.cmdwnd.command == CommandWindow.EQUIPMENT:  # そうび
                    # TODO: そうびウィンドウ表示
                    sounds["pi"].play()
                    self.cmdwnd.hide()
                    self.msgwnd.set(u"そうびウィンドウが　ひらくよてい。")
                    self.game_state = TALK
                elif self.cmdwnd.command == CommandWindow.DOOR:  # とびら
                    sounds["pi"].play()
                    self.cmdwnd.hide()
                    door = player.open(self.map)
```

```
                if door != None:
                    door.open()
                    self.map.remove_event(door)
                    self.game_state = FIELD
                else:
                    self.msgwnd.set(u"そのほうこうに　とびらはない。")
                    self.game_state = TALK
            elif self.cmdwnd.command == CommandWindow.SPELL:  # じゅもん
                # TODO: じゅもんウィンドウ表示
                sounds["pi"].play()
                self.cmdwnd.hide()
                self.msgwnd.set(u"じゅもんウィンドウが　ひらくよてい。")
                self.game_state = TALK
            elif self.cmdwnd.command == CommandWindow.ITEM:  # どうぐ
                # TODO: どうぐウィンドウ表示
                sounds["pi"].play()
                self.cmdwnd.hide()
                self.msgwnd.set(u"どうぐウィンドウが　ひらくよてい。")
                self.game_state = TALK
            elif self.cmdwnd.command == CommandWindow.TACTICS:  # さくせん
                # TODO: さくせんウィンドウ表示
                sounds["pi"].play()
                self.cmdwnd.hide()
                self.msgwnd.set(u"さくせんウィンドウが　ひらくよてい。")
                self.game_state = TALK
            elif self.cmdwnd.command == CommandWindow.SEARCH:  # しらべる
                sounds["pi"].play()
                self.cmdwnd.hide()
                treasure = player.search(self.map)
                if treasure != None:
                    treasure.open()
                    self.msgwnd.set(u"%s　をてにいれた。" % treasure.item)
                    self.game_state = TALK
                    self.map.remove_event(treasure)
                else:
                    self.msgwnd.set(u"しかし　なにもみつからなかった。")
                    self.game_state = TALK

    def talk_handler(self, event):
        """会話中のイベントハンドラ"""
        # スペースキーでメッセージウィンドウを次のページへ
        # なかった場合、フィールド状態へ戻る
        if event.type == KEYDOWN and event.key == K_SPACE:
            if not self.msgwnd.next():
                self.game_state = FIELD

    def calc_offset(self, player):
        """オフセットを計算する"""
        offsetx = player.rect.topleft[0] - SCR_RECT.width / 2
        offsety = player.rect.topleft[1] - SCR_RECT.height / 2
        return offsetx, offsety

    def show_info(self):
        """デバッグ情報を表示"""
        player = self.party.member[0]  # 先頭プレイヤー
        self.msg_engine.draw_string(
            self.screen, (300, 10), self.map.name.upper())  # マップ名
        self.msg_engine.draw_string(
            self.screen, (300, 40), player.name.upper())  # プレイヤー名
        self.msg_engine.draw_string(
            self.screen, (300, 70), "%d_%d" % (player.x, player.y))  # プレイヤー座標

    def load_sounds(self, dir, file):
        """サウンドをロードしてsoundsに格納"""
        file = os.path.join(dir, file)
        fp = open(file, "r")
        for line in fp:
            line = line.rstrip()
            data = line.split(",")
            se_name = data[0]
            se_file = os.path.join("se", data[1])
            sounds[se_name] = pygame.mixer.Sound(se_file)
```

```
        fp.close()

    def load_charachips(self, dir, file):
        """キャラクターチップをロードしてCharacter.imagesに格納"""
        file = os.path.join(dir, file)
        fp = open(file, "r")
        for line in fp:
            line = line.rstrip()
            data = line.split(",")
            chara_id = int(data[0])
            chara_name = data[1]
            Character.images[chara_name] = split_image(
                load_image("charachip", "%s.png" % chara_name))
        fp.close()

    def load_mapchips(self, dir, file):
        """マップチップをロードしてMap.imagesに格納"""
        file = os.path.join(dir, file)
        fp = open(file, "r")
        for line in fp:
            line = line.rstrip()
            data = line.split(",")
            mapchip_id = int(data[0])
            mapchip_name = data[1]
            movable = int(data[2])  # 移動可能か？
            transparent = int(data[3])  # 背景を透明にするか？
            if transparent == 0:
                Map.images.append(load_image(
                    "mapchip", "%s.png" % mapchip_name))
            else:
                Map.images.append(load_image(
                    "mapchip", "%s.png" % mapchip_name, TRANS_COLOR))
            Map.movable_type.append(movable)
        fp.close()


class Map:
    # main()のload_mapchips()でセットされる
    images = []  # マップチップ（ID->イメージ）
    movable_type = []  # マップチップが移動可能か？（0:移動不可, 1:移動可）

    def __init__(self, name, party):
        self.name = name
        self.row = -1  # 行数
        self.col = -1  # 列数
        self.map = []  # マップデータ（2次元リスト）
        self.charas = []  # マップにいるキャラクターリスト
        self.events = []  # マップにあるイベントリスト
        self.party = party  # Partyの登録（衝突判定用）
        self.load()  # マップをロード
        self.load_event()  # イベントをロード

    def create(self, dest_map):
        """dest_mapでマップを初期化"""
        self.name = dest_map
        self.charas = []
        self.events = []
        self.load()
        self.load_event()

    def add_chara(self, chara):
        """キャラクターをマップに追加する"""
        self.charas.append(chara)

    def update(self):
        """マップの更新"""
        # マップにいるキャラクターの更新
        for chara in self.charas:
            chara.update(self)  # mapを渡す

    def draw(self, screen, offset):
        """マップを描画する"""
```

```
        offsetx, offsety = offset
        # マップの描画範囲を計算
        startx = offsetx / GS
        endx = startx + SCR_RECT.width / GS + 1
        starty = offsety / GS
        endy = starty + SCR_RECT.height / GS + 1
        # マップの描画
        for y in range(starty, endy):
            for x in range(startx, endx):
                # マップの範囲外はデフォルトイメージで描画
                # この条件がないとマップの端に行くとエラー発生
                if x < 0 or y < 0 or x > self.col - 1 or y > self.row - 1:
                    screen.blit(self.images[self.default],
                                (x * GS - offsetx, y * GS - offsety))
                else:
                    screen.blit(self.images[self.map[y][x]],
                                (x * GS - offsetx, y * GS - offsety))
        # このマップにあるイベントを描画
        for event in self.events:
            event.draw(screen, offset)
        # このマップにいるキャラクターを描画
        for chara in self.charas:
            chara.draw(screen, offset)

    def is_movable(self, x, y):
        """(x,y)は移動可能か？"""
        # マップ範囲内か？
        if x < 0 or x > self.col - 1 or y < 0 or y > self.row - 1:
            return False
        # マップチップは移動可能か？
        if self.movable_type[self.map[y][x]] == 0:
            return False
        # キャラクターと衝突しないか？
        for chara in self.charas:
            if chara.x == x and chara.y == y:
                return False
        # イベントと衝突しないか？
        for event in self.events:
            if self.movable_type[event.mapchip] == 0:
                if event.x == x and event.y == y:
                    return False
        # 先頭プレイヤーと衝突しないか？
        # 先頭プレイヤー以外は無視
        player = self.party.member[0]
        if player.x == x and player.y == y:
            return False
        return True

    def get_chara(self, x, y):
        """(x,y)にいるキャラクターを返す。いなければNone"""
        for chara in self.charas:
            if chara.x == x and chara.y == y:
                return chara
        return None

    def get_event(self, x, y):
        """(x,y)にあるイベントを返す。なければNone"""
        for event in self.events:
            if event.x == x and event.y == y:
                return event
        return None

    def remove_event(self, event):
        """eventを削除する"""
        self.events.remove(event)

    def load(self):
        """バイナリファイルからマップをロード"""
        file = os.path.join("data", self.name + ".map")
        fp = open(file, "rb")
        # unpack()はタプルが返されるので[0]だけ抽出
        self.row = struct.unpack("i", fp.read(struct.calcsize("i")))[0]  # 行数
```

```
        self.col = struct.unpack("i", fp.read(struct.calcsize("i")))[0]  # 列数
        self.default = struct.unpack("B", fp.read(struct.calcsize("B")))[
            0]  # デフォルトマップチップ
        # マップ
        self.map = [[0 for c in range(self.col)] for r in range(self.row)]
        for r in range(self.row):
            for c in range(self.col):
                self.map[r][c] = struct.unpack(
                    "B", fp.read(struct.calcsize("B")))[0]
        fp.close()

    def load_event(self):
        """ファイルからイベントをロード"""
        file = os.path.join("data", self.name + ".evt")
        # テキスト形式のイベントを読み込む
        fp = codecs.open(file, "r", "utf-8")
        for line in fp:
            line = line.rstrip()  # 改行除去
            if line.startswith("#"):
                continue  # コメント行は無視
            if line == "":
                continue  # 空行は無視
            data = line.split(",")
            event_type = data[0]
            if event_type == "BGM":  # BGMイベント
                self.play_bgm(data)
            elif event_type == "CHARA":  # キャラクターイベント
                self.create_chara(data)
            elif event_type == "MOVE":  # 移動イベント
                self.create_move(data)
            elif event_type == "TREASURE":  # 宝箱
                self.create_treasure(data)
            elif event_type == "DOOR":  # とびら
                self.create_door(data)
            elif event_type == "OBJECT":  # 一般オブジェクト(玉座など)
                self.create_obj(data)
        fp.close()

    def play_bgm(self, data):
        """BGMを鳴らす"""
        bgm_file = "%s.mp3" % data[1]
        bgm_file = os.path.join("bgm", bgm_file)
        pygame.mixer.music.load(bgm_file)
        pygame.mixer.music.play(-1)

    def create_chara(self, data):
        """キャラクターを作成してcharasに追加する"""
        name = data[1]
        x, y = int(data[2]), int(data[3])
        direction = int(data[4])
        movetype = int(data[5])
        message = data[6]
        chara = Character(name, (x, y), direction, movetype, message)
        self.charas.append(chara)

    def create_move(self, data):
        """移動イベントを作成してeventsに追加する"""
        x, y = int(data[1]), int(data[2])
        mapchip = int(data[3])
        dest_map = data[4]
        dest_x, dest_y = int(data[5]), int(data[6])
        move = MoveEvent((x, y), mapchip, dest_map, (dest_x, dest_y))
        self.events.append(move)

    def create_treasure(self, data):
        """宝箱を作成してeventsに追加する"""
        x, y = int(data[1]), int(data[2])
        item = data[3]
        treasure = Treasure((x, y), item)
        self.events.append(treasure)

    def create_door(self, data):
```

```
        """とびらを作成してeventsに追加する"""
        x, y = int(data[1]), int(data[2])
        door = Door((x, y))
        self.events.append(door)

    def create_obj(self, data):
        """一般オブジェクトを作成してeventsに追加する"""
        x, y = int(data[1]), int(data[2])
        mapchip = int(data[3])
        obj = Object((x, y), mapchip)
        self.events.append(obj)


class Character:
    """一般キャラクタークラス"""
    speed = 4  # 1フレームの移動ピクセル数
    animcycle = 24  # アニメーション速度
    frame = 0
    # キャラクターイメージ（mainで初期化）
    # キャラクター名 -> 分割画像リストの辞書
    images = {}

    def __init__(self, name, pos, dir, movetype, message):
        self.name = name  # キャラクター名（ファイル名と同じ）
        self.image = self.images[name][0]  # 描画中のイメージ
        self.x, self.y = pos[0], pos[1]  # 座標（単位: マス）
        self.rect = self.image.get_rect(topleft=(self.x * GS, self.y * GS))
        self.vx, self.vy = 0, 0  # 移動速度
        self.moving = False  # 移動中か？
        self.direction = dir  # 向き
        self.movetype = movetype  # 移動タイプ
        self.message = message  # メッセージ

    def update(self, map):
        """キャラクター状態を更新する。
        mapは移動可能かの判定に必要。"""
        # プレイヤーの移動処理
        if self.moving == True:
            # ピクセル移動中ならマスにきっちり収まるまで移動を続ける
            self.rect.move_ip(self.vx, self.vy)
            if self.rect.left % GS == 0 and self.rect.top % GS == 0:  # マスにおさまったら移動完了
                self.moving = False
                self.x = self.rect.left / GS
                self.y = self.rect.top / GS
        elif self.movetype == MOVE and random.random() < PROB_MOVE:
            # 移動中でないならPROB_MOVEの確率でランダム移動開始
            self.direction = random.randint(0, 3)  # 0-3のいずれか
            if self.direction == DOWN:
                if map.is_movable(self.x, self.y + 1):
                    self.vx, self.vy = 0, self.speed
                    self.moving = True
            elif self.direction == LEFT:
                if map.is_movable(self.x - 1, self.y):
                    self.vx, self.vy = -self.speed, 0
                    self.moving = True
            elif self.direction == RIGHT:
                if map.is_movable(self.x + 1, self.y):
                    self.vx, self.vy = self.speed, 0
                    self.moving = True
            elif self.direction == UP:
                if map.is_movable(self.x, self.y - 1):
                    self.vx, self.vy = 0, -self.speed
                    self.moving = True
        # キャラクターアニメーション（frameに応じて描画イメージを切り替える）
        self.frame += 1
        self.image = self.images[self.name][self.direction *
                                            4 + self.frame / self.animcycle % 4]

    def draw(self, screen, offset):
        """オフセットを考慮してプレイヤーを描画"""
        offsetx, offsety = offset
        px = self.rect.topleft[0]
```

```
        py = self.rect.topleft[1]
        screen.blit(self.image, (px - offsetx, py - offsety))

    def set_pos(self, x, y, dir):
        """キャラクターの位置と向きをセット"""
        self.x, self.y = x, y
        self.rect = self.image.get_rect(topleft=(self.x * GS, self.y * GS))
        self.direction = dir

    def __str__(self):
        return "CHARA,%s,%d,%d,%d,%d,%s" % (self.name, self.x, self.y, self.direction, self.movetype,
self.message)


class Player(Character):
    """プレイヤークラス"""

    def __init__(self, name, pos, dir, leader, party):
        Character.__init__(self, name, pos, dir, False, None)
        self.leader = leader
        self.party = party

    def update(self, map):
        """プレイヤー状態を更新する。
        mapは移動可能かの判定に必要。"""
        # プレイヤーの移動処理
        if self.moving == True:
            # ピクセル移動中ならマスにきっちり収まるまで移動を続ける
            self.rect.move_ip(self.vx, self.vy)
            if self.rect.left % GS == 0 and self.rect.top % GS == 0:  # マスにおさまったら移動完了
                self.moving = False
                self.x = self.rect.left / GS
                self.y = self.rect.top / GS
                # TODO: ここに接触イベントのチェックを入れる
                if not self.leader:
                    return  # リーダーでなければイベントは無視
                event = map.get_event(self.x, self.y)
                if isinstance(event, MoveEvent):  # MoveEventなら
                    sounds["step"].play()
                    dest_map = event.dest_map
                    dest_x = event.dest_x
                    dest_y = event.dest_y
                    map.create(dest_map)
                    # パーティの全員を移動先マップへ
                    for player in self.party.member:
                        player.set_pos(dest_x, dest_y, DOWN)  # プレイヤーを移動先座標へ
                        player.moving = False
        # キャラクターアニメーション（frameに応じて描画イメージを切り替える）
        self.frame += 1
        self.image = self.images[self.name][self.direction *
                                            4 + self.frame / self.animcycle % 4]

    def move_to(self, destx, desty):
        """現在位置から(destx,desty)への移動を開始"""
        dx = destx - self.x
        dy = desty - self.y
        # 向きを変える
        if dx == 1:
            self.direction = RIGHT
        elif dx == -1:
            self.direction = LEFT
        elif dy == -1:
            self.direction = UP
        elif dy == 1:
            self.direction = DOWN
        # 速度をセット
        self.vx, self.vy = dx * self.speed, dy * self.speed
        # 移動開始
        self.moving = True

    def talk(self, map):
        """キャラクターが向いている方向のとなりにキャラクターがいるか調べる"""
```

```
        # 向いている方向のとなりの座標を求める
        nextx, nexty = self.x, self.y
        if self.direction == DOWN:
            nexty = self.y + 1
            event = map.get_event(nextx, nexty)
            if isinstance(event, Object) and event.mapchip == 41:
                nexty += 1  # テーブルがあったらさらに隣
        elif self.direction == LEFT:
            nextx = self.x - 1
            event = map.get_event(nextx, nexty)
            if isinstance(event, Object) and event.mapchip == 41:
                nextx -= 1
        elif self.direction == RIGHT:
            nextx = self.x + 1
            event = map.get_event(nextx, nexty)
            if isinstance(event, Object) and event.mapchip == 41:
                nextx += 1
        elif self.direction == UP:
            nexty = self.y - 1
            event = map.get_event(nextx, nexty)
            if isinstance(event, Object) and event.mapchip == 41:
                nexty -= 1
        # その方向にキャラクターがいるか？
        chara = map.get_chara(nextx, nexty)
        # キャラクターがいればプレイヤーの方向へ向ける
        if chara != None:
            if self.direction == DOWN:
                chara.direction = UP
            elif self.direction == LEFT:
                chara.direction = RIGHT
            elif self.direction == RIGHT:
                chara.direction = LEFT
            elif self.direction == UP:
                chara.direction = DOWN
            chara.update(map)  # 向きを変えたので更新
        return chara

    def search(self, map):
        """足もとに宝箱があるか調べる"""
        event = map.get_event(self.x, self.y)
        if isinstance(event, Treasure):
            return event
        return None

    def open(self, map):
        """目の前にとびらがあるか調べる"""
        # 向いている方向のとなりの座標を求める
        nextx, nexty = self.x, self.y
        if self.direction == DOWN:
            nexty = self.y + 1
        elif self.direction == LEFT:
            nextx = self.x - 1
        elif self.direction == RIGHT:
            nextx = self.x + 1
        elif self.direction == UP:
            nexty = self.y - 1
        # その場所にとびらがあるか？
        event = map.get_event(nextx, nexty)
        if isinstance(event, Door):
            return event
        return None


class Party:
    def __init__(self):
        # Partyのメンバーリスト
        self.member = []

    def add(self, player):
        """Partyにplayerを追加"""
        self.member.append(player)
```

```
    def update(self, map):
        # Party全員を更新
        for player in self.member:
            player.update(map)
        # 移動中でないときにキー入力があったらParty全員を移動開始
        if not self.member[0].moving:
            pressed_keys = pygame.key.get_pressed()
            if pressed_keys[K_DOWN]:
                # 先頭キャラは移動できなくても向きは変える
                self.member[0].direction = DOWN
                # 先頭キャラが移動できれば
                if map.is_movable(self.member[0].x, self.member[0].y + 1):
                    # 後ろにいる仲間から1つ前の仲間の位置へ移動開始
                    for i in range(len(self.member) - 1, 0, -1):
                        self.member[i].move_to(
                            self.member[i - 1].x, self.member[i - 1].y)
                    # 先頭キャラを最後に移動開始
                    self.member[0].move_to(
                        self.member[0].x, self.member[0].y + 1)
            elif pressed_keys[K_LEFT]:
                self.member[0].direction = LEFT
                if map.is_movable(self.member[0].x - 1, self.member[0].y):
                    for i in range(len(self.member) - 1, 0, -1):
                        self.member[i].move_to(
                            self.member[i - 1].x, self.member[i - 1].y)
                    self.member[0].move_to(
                        self.member[0].x - 1, self.member[0].y)
            elif pressed_keys[K_RIGHT]:
                self.member[0].direction = RIGHT
                if map.is_movable(self.member[0].x + 1, self.member[0].y):
                    for i in range(len(self.member) - 1, 0, -1):
                        self.member[i].move_to(
                            self.member[i - 1].x, self.member[i - 1].y)
                    self.member[0].move_to(
                        self.member[0].x + 1, self.member[0].y)
            elif pressed_keys[K_UP]:
                self.member[0].direction = UP
                if map.is_movable(self.member[0].x, self.member[0].y - 1):
                    for i in range(len(self.member) - 1, 0, -1):
                        self.member[i].move_to(
                            self.member[i - 1].x, self.member[i - 1].y)
                    self.member[0].move_to(
                        self.member[0].x, self.member[0].y - 1)

    def draw(self, screen, offset):
        # Partyの全員を描画
        # 重なったとき先頭キャラが表示されるように後ろの人から描画
        for player in self.member[::-1]:
            player.draw(screen, offset)


class MessageEngine:
    FONT_WIDTH = 16
    FONT_HEIGHT = 22
    WHITE, RED, GREEN, BLUE = 0, 160, 320, 480

    def __init__(self):
        self.image = load_image("data", "font.png", -1)
        self.color = self.WHITE
        self.kana2rect = {}
        self.create_hash()

    def set_color(self, color):
        """文字色をセット"""
        self.color = color
        # 変な値だったらWHITEにする
        if not self.color in [self.WHITE, self.RED, self.GREEN, self.BLUE]:
            self.color = self.WHITE

    def draw_character(self, screen, pos, ch):
        """1文字だけ描画する"""
        x, y = pos
```

```
        try:
            rect = self.kana2rect[ch]
            screen.blit(self.image, (x, y), (rect.x + self.color,
                                             rect.y, rect.width, rect.height))
        except KeyError:
            print "描画できない文字があります:%s" % ch
            return

    def draw_string(self, screen, pos, str):
        """文字列を描画"""
        x, y = pos
        for i, ch in enumerate(str):
            dx = x + self.FONT_WIDTH * i
            self.draw_character(screen, (dx, y), ch)

    def create_hash(self):
        """文字から座標への辞書を作成"""
        filepath = os.path.join("data", "kana2rect.dat")
        fp = codecs.open(filepath, "r", "utf-8")
        for line in fp.readlines():
            line = line.rstrip()
            d = line.split(" ")
            kana, x, y, w, h = d[0], int(d[1]), int(d[2]), int(d[3]), int(d[4])
            self.kana2rect[kana] = Rect(x, y, w, h)
        fp.close()


class Window:
    """ウィンドウの基本クラス"""
    EDGE_WIDTH = 4  # 白枠の幅

    def __init__(self, rect):
        self.rect = rect  # 一番外側の白い矩形
        # 内側の黒い矩形
        self.inner_rect = self.rect.inflate(-self.EDGE_WIDTH *
                                            2, -self.EDGE_WIDTH * 2)
        self.is_visible = False  # ウィンドウを表示中か？

    def draw(self, screen):
        """ウィンドウを描画"""
        if self.is_visible == False:
            return
        pygame.draw.rect(screen, (255, 255, 255), self.rect, 0)
        pygame.draw.rect(screen, (0, 0, 0), self.inner_rect, 0)

    def show(self):
        """ウィンドウを表示"""
        self.is_visible = True

    def hide(self):
        """ウィンドウを隠す"""
        self.is_visible = False


class MessageWindow(Window):
    """メッセージウィンドウ"""
    MAX_CHARS_PER_LINE = 20    # 1行の最大文字数
    MAX_LINES_PER_PAGE = 3     # 1行の最大行数（4行目は▼用）
    MAX_CHARS_PER_PAGE = 20 * 3  # 1ページの最大文字数
    MAX_LINES = 30             # メッセージを格納できる最大行数
    LINE_HEIGHT = 8            # 行間の大きさ
    animcycle = 24

    def __init__(self, rect, msg_engine):
        Window.__init__(self, rect)
        self.text_rect = self.inner_rect.inflate(-32, -32)  # テキストを表示する矩形
        self.text = []  # メッセージ
        self.cur_page = 0  # 現在表示しているページ
        self.cur_pos = 0  # 現在ページで表示した最大文字数
        self.next_flag = False  # 次ページがあるか？
        self.hide_flag = False  # 次のキー入力でウィンドウを消すか？
        self.msg_engine = msg_engine  # メッセージエンジン
```

```
        self.cursor = load_image("data", "cursor.png", -1)  # カーソル画像
        self.frame = 0

    def set(self, message):
        """メッセージをセットしてウィンドウを画面に表示する"""
        self.cur_pos = 0
        self.cur_page = 0
        self.next_flag = False
        self.hide_flag = False
        # 全角スペースで初期化
        self.text = [u'　'] * (self.MAX_LINES * self.MAX_CHARS_PER_LINE)
        # メッセージをセット
        p = 0
        for i in range(len(message)):
            ch = message[i]
            if ch == "/":  # /は改行文字
                self.text[p] = "/"
                p += self.MAX_CHARS_PER_LINE
                p = (p / self.MAX_CHARS_PER_LINE) * self.MAX_CHARS_PER_LINE
            elif ch == "%":  # \fは改ページ文字
                self.text[p] = "%"
                p += self.MAX_CHARS_PER_PAGE
                p = (p / self.MAX_CHARS_PER_PAGE) * self.MAX_CHARS_PER_PAGE
            else:
                self.text[p] = ch
                p += 1
        self.text[p] = "$"  # 終端文字
        self.show()

    def update(self):
        """メッセージウィンドウを更新する
        メッセージが流れるように表示する"""
        if self.is_visible:
            if self.next_flag == False:
                self.cur_pos += 1  # 1文字流す
                # テキスト全体から見た現在位置
                p = self.cur_page * self.MAX_CHARS_PER_PAGE + self.cur_pos
                if self.text[p] == "/":  # 改行文字
                    self.cur_pos += self.MAX_CHARS_PER_LINE
                    self.cur_pos = (
                        self.cur_pos / self.MAX_CHARS_PER_LINE) * self.MAX_CHARS_PER_LINE
                elif self.text[p] == "%":  # 改ページ文字
                    self.cur_pos += self.MAX_CHARS_PER_PAGE
                    self.cur_pos = (
                        self.cur_pos / self.MAX_CHARS_PER_PAGE) * self.MAX_CHARS_PER_PAGE
                elif self.text[p] == "$":  # 終端文字
                    self.hide_flag = True
                # 1ページの文字数に達したら▼を表示
                if self.cur_pos % self.MAX_CHARS_PER_PAGE == 0:
                    self.next_flag = True
        self.frame += 1

    def draw(self, screen):
        """メッセージを描画する
        メッセージウィンドウが表示されていないときは何もしない"""
        Window.draw(self, screen)
        if self.is_visible == False:
            return
        # 現在表示しているページのcur_posまでの文字を描画
        for i in range(self.cur_pos):
            ch = self.text[self.cur_page * self.MAX_CHARS_PER_PAGE + i]
            if ch == "/" or ch == "%" or ch == "$":
                continue  # 制御文字は表示しない
            dx = self.text_rect[0] + MessageEngine.FONT_WIDTH * \
                (i % self.MAX_CHARS_PER_LINE)
            dy = self.text_rect[1] + (self.LINE_HEIGHT +
                                      MessageEngine.FONT_HEIGHT) * (i / self.MAX_CHARS_PER_LINE)
            self.msg_engine.draw_character(screen, (dx, dy), ch)
        # 最後のページでない場合は▼を表示
        if (not self.hide_flag) and self.next_flag:
            if self.frame / self.animcycle % 2 == 0:
                dx = self.text_rect[0] + (self.MAX_CHARS_PER_LINE / 2) * \
```

```
                    MessageEngine.FONT_WIDTH - MessageEngine.FONT_WIDTH / 2
                dy = self.text_rect[1] + \
                    (self.LINE_HEIGHT + MessageEngine.FONT_HEIGHT) * 3
                screen.blit(self.cursor, (dx, dy))

    def next(self):
        """メッセージを先に進める"""
        # 現在のページが最後のページだったらウィンドウを閉じる
        if self.hide_flag:
            self.hide()
            return False
        # ▼が表示されてれば次のページへ
        if self.next_flag:
            self.cur_page += 1
            self.cur_pos = 0
            self.next_flag = False
            return True


class CommandWindow(Window):
    LINE_HEIGHT = 8  # 行間の大きさ
    TALK, STATUS, EQUIPMENT, DOOR, SPELL, ITEM, TACTICS, SEARCH = range(0, 8)
    COMMAND = [u"はなす", u"つよさ", u"そうび", u"とびら",
               u"じゅもん", u"どうぐ", u"さくせん", u"しらべる"]

    def __init__(self, rect, msg_engine):
        Window.__init__(self, rect)
        self.text_rect = self.inner_rect.inflate(-32, -32)
        self.command = self.TALK  # 選択中のコマンド
        self.msg_engine = msg_engine
        self.cursor = load_image("data", "cursor2.png", -1)
        self.frame = 0

    def draw(self, screen):
        Window.draw(self, screen)
        if self.is_visible == False:
            return
        # はなす、つよさ、そうび、とびらを描画
        for i in range(0, 4):
            dx = self.text_rect[0] + MessageEngine.FONT_WIDTH
            dy = self.text_rect[1] + (self.LINE_HEIGHT +
                                      MessageEngine.FONT_HEIGHT) * (i % 4)
            self.msg_engine.draw_string(screen, (dx, dy), self.COMMAND[i])
        # じゅもん、どうぐ、さくせん、しらべるを描画
        for i in range(4, 8):
            dx = self.text_rect[0] + MessageEngine.FONT_WIDTH * 6
            dy = self.text_rect[1] + (self.LINE_HEIGHT +
                                      MessageEngine.FONT_HEIGHT) * (i % 4)
            self.msg_engine.draw_string(screen, (dx, dy), self.COMMAND[i])
        # 選択中のコマンドの左側に▸を描画
        dx = self.text_rect[0] + \
            MessageEngine.FONT_WIDTH * 5 * (self.command / 4)
        dy = self.text_rect[1] + (self.LINE_HEIGHT +
                                  MessageEngine.FONT_HEIGHT) * (self.command % 4)
        screen.blit(self.cursor, (dx, dy))

    def show(self):
        """オーバーライド"""
        self.command = self.TALK  # 追加
        self.is_visible = True


class MoveEvent():
    """移動イベント"""

    def __init__(self, pos, mapchip, dest_map, dest_pos):
        self.x, self.y = pos[0], pos[1]  # イベント座標
        self.mapchip = mapchip  # マップチップ
        self.dest_map = dest_map  # 移動先マップ名
        self.dest_x, self.dest_y = dest_pos[0], dest_pos[1]  # 移動先座標
        self.image = Map.images[self.mapchip]
        self.rect = self.image.get_rect(topleft=(self.x * GS, self.y * GS))
```

```
    def draw(self, screen, offset):
        """オフセットを考慮してイベントを描画"""
        offsetx, offsety = offset
        px = self.rect.topleft[0]
        py = self.rect.topleft[1]
        screen.blit(self.image, (px - offsetx, py - offsety))

    def __str__(self):
        return "MOVE,%d,%d,%d,%s,%d,%d" % (self.x, self.y, self.mapchip, self.dest_map, self.dest_x,
self.dest_y)


class Treasure():
    """宝箱"""

    def __init__(self, pos, item):
        self.x, self.y = pos[0], pos[1]  # 宝箱座標
        self.mapchip = 46  # 宝箱は46
        self.image = Map.images[self.mapchip]
        self.rect = self.image.get_rect(topleft=(self.x * GS, self.y * GS))
        self.item = item  # アイテム名

    def open(self):
        """宝箱をあける"""
        sounds["treasure"].play()
        # TODO: アイテムを追加する処理

    def draw(self, screen, offset):
        """オフセットを考慮してイベントを描画"""
        offsetx, offsety = offset
        px = self.rect.topleft[0]
        py = self.rect.topleft[1]
        screen.blit(self.image, (px - offsetx, py - offsety))

    def __str__(self):
        return "TREASURE,%d,%d,%s" % (self.x, self.y, self.item)


class Door:
    """とびら"""

    def __init__(self, pos):
        self.x, self.y = pos[0], pos[1]
        self.mapchip = 45
        self.image = Map.images[self.mapchip]
        self.rect = self.image.get_rect(topleft=(self.x * GS, self.y * GS))

    def open(self):
        """とびらをあける"""
        sounds["door"].play()

    def draw(self, screen, offset):
        """オフセットを考慮してイベントを描画"""
        offsetx, offsety = offset
        px = self.rect.topleft[0]
        py = self.rect.topleft[1]
        screen.blit(self.image, (px - offsetx, py - offsety))

    def __str__(self):
        return "DOOR,%d,%d" % (self.x, self.y)


class Object:
    """一般オブジェクト"""

    def __init__(self, pos, mapchip):
        self.x, self.y = pos[0], pos[1]
        self.mapchip = mapchip
        self.image = Map.images[self.mapchip]
        self.rect = self.image.get_rect(topleft=(self.x * GS, self.y * GS))
```

```
    def draw(self, screen, offset):
        """オフセットを考慮してイベントを描画"""
        offsetx, offsety = offset
        px = self.rect.topleft[0]
        py = self.rect.topleft[1]
        screen.blit(self.image, (px - offsetx, py - offsety))

    def __str__(self):
        return "OBJECT,%d,%d,%d" % (self.x, self.y, mapchip)


class Title:
    """タイトル画面"""
    START, CONTINUE, EXIT = 0, 1, 2

    def __init__(self, msg_engine):
        self.msg_engine = msg_engine
        self.title_img = load_image("data", "python_quest.png", -1)
        self.cursor_img = load_image("data", "cursor2.png", -1)
        self.menu = self.START
        self.play_bgm()

    def update(self):
        pass

    def draw(self, screen):
        screen.fill((0, 0, 128))
        # タイトルの描画
        screen.blit(self.title_img, (20, 60))
        # メニューの描画
        self.msg_engine.draw_string(screen, (260, 240), u"ＳＴＡＲＴ")
        self.msg_engine.draw_string(screen, (260, 280), u"ＣＯＮＴＩＮＵＥ")
        self.msg_engine.draw_string(screen, (260, 320), u"ＥＸＩＴ")
        # クレジットの描画
        self.msg_engine.draw_string(
            screen, (130, 400), u"２００８　ＰＹＴＨＯＮでゲームつくりますがなにか？")
        # メニューカーソルの描画
        if self.menu == self.START:
            screen.blit(self.cursor_img, (240, 240))
        elif self.menu == self.CONTINUE:
            screen.blit(self.cursor_img, (240, 280))
        elif self.menu == self.EXIT:
            screen.blit(self.cursor_img, (240, 320))

    def play_bgm(self):
        bgm_file = "title.mp3"
        bgm_file = os.path.join("bgm", bgm_file)
        pygame.mixer.music.load(bgm_file)
        pygame.mixer.music.play(-1)


if __name__ == "__main__":
    PyRPG()
```