Space Wandering

宇宙に浮かぶミニゲームを巡りメダルを集めるゲーム

つまみ 淵野アタリ ちくわぶ

分担作業

つみ

Model担当
使用する部品の親クラスの作成
ミニゲームの制作

ちく

View担当
ロケットや星などの部品の制作
ミニゲームの制作

アタ

Controller担当
操作のチューニング
操作に使用するデータ構造の整備

分担作業

つみ

Model担当
使用する部品の親クラスの作成
ミニゲームの制作

ちく

View担当
ロケットや星などの部品の制作
ミニゲームの制作

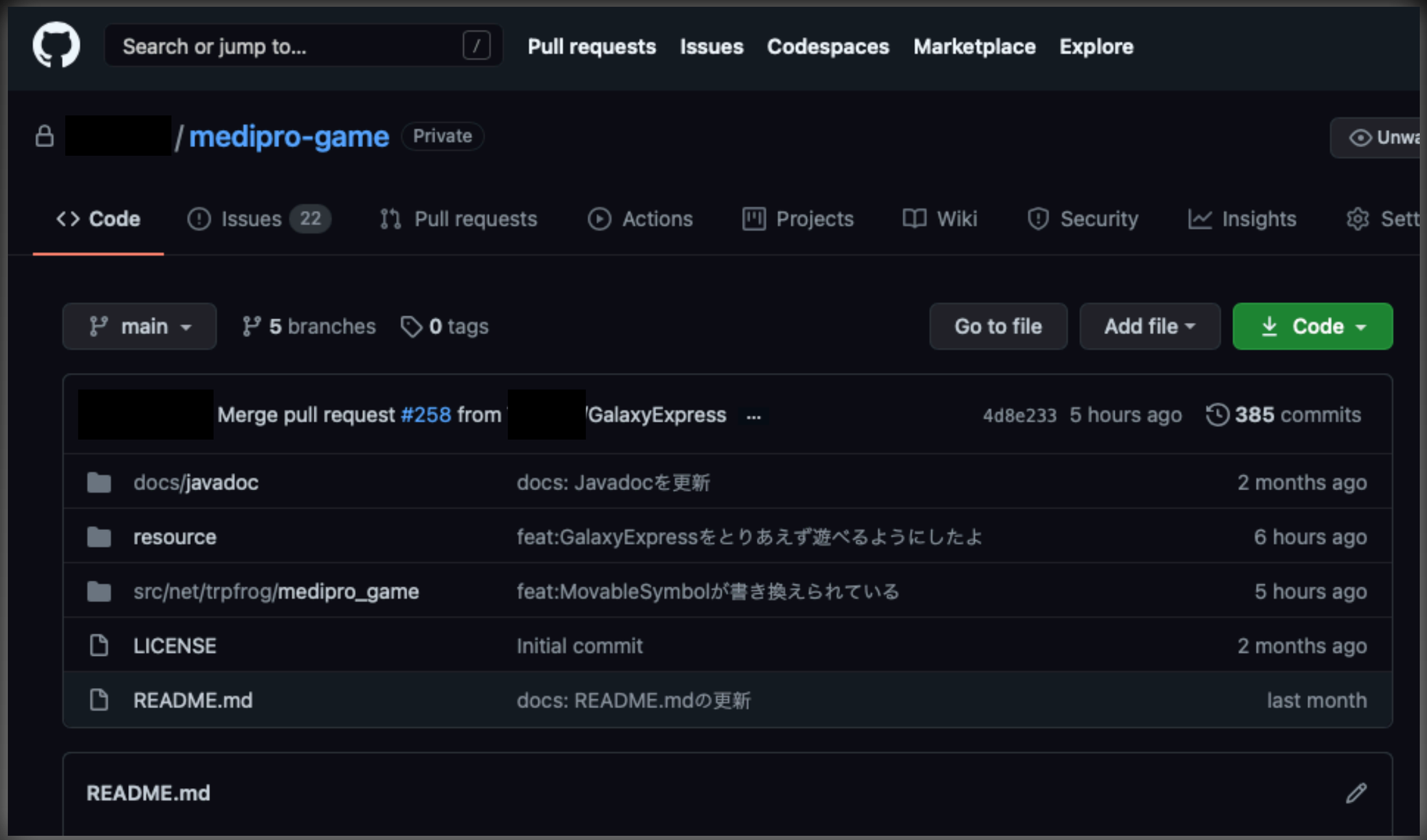
アタ

Controller担当
操作のチューニング
操作に使用するデータ構造の整備

基本はこのように分かれていたが、GitHubのIssues機能で
手が空いた人にどんどん仕事を割り振った

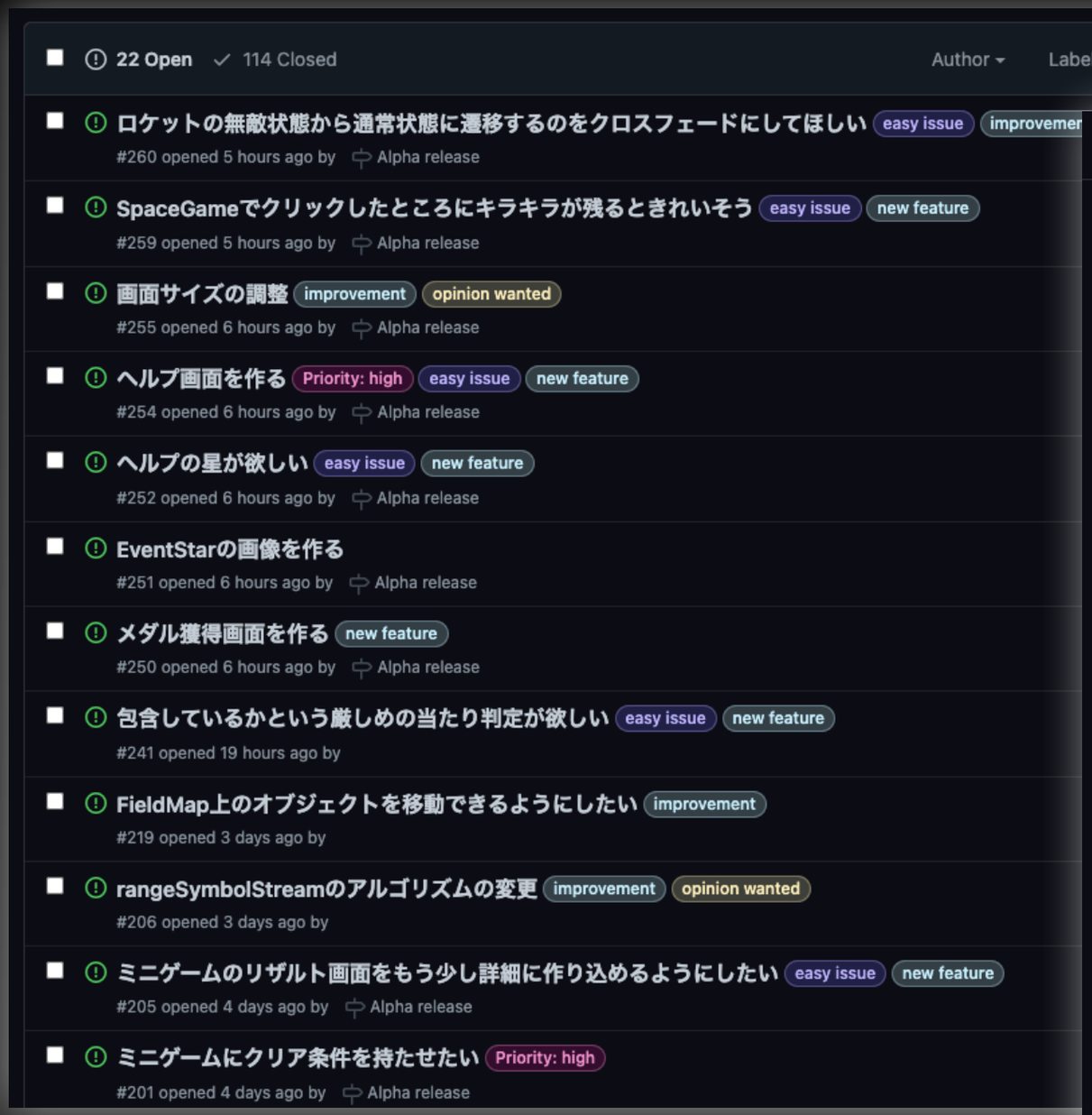
グループ制作での工夫点

GitHubの使用



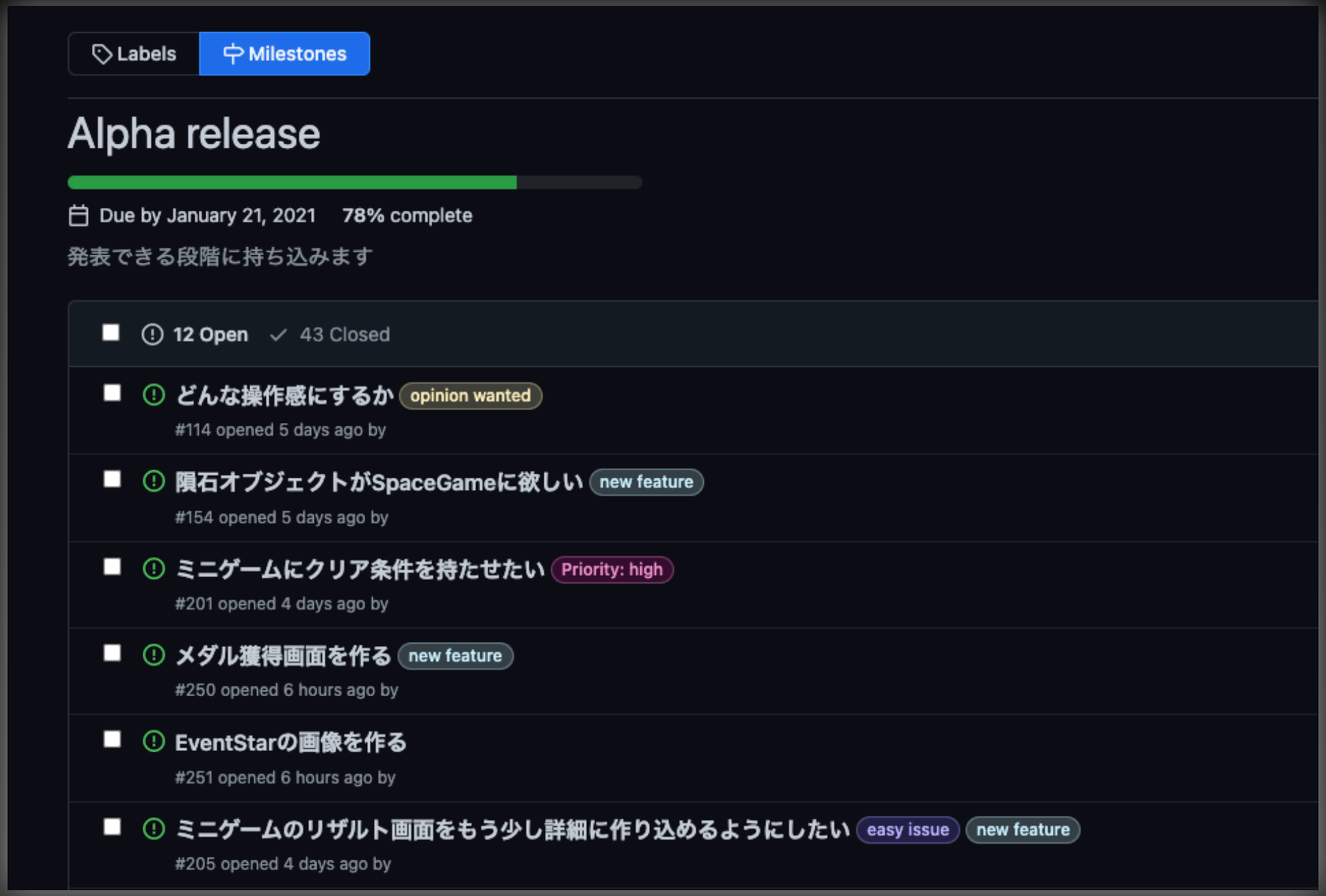
コードをオンラインで共有できる

GitHubの使用



Issues機能でタスク管理と意見交換ができる

GitHubの使用



Milestones機能で締切の管理ができる

GitHubの使用

PullRequest機能で事前にコードをチェックできる

→ 重大なミスでコードが消し飛ぶことがない！

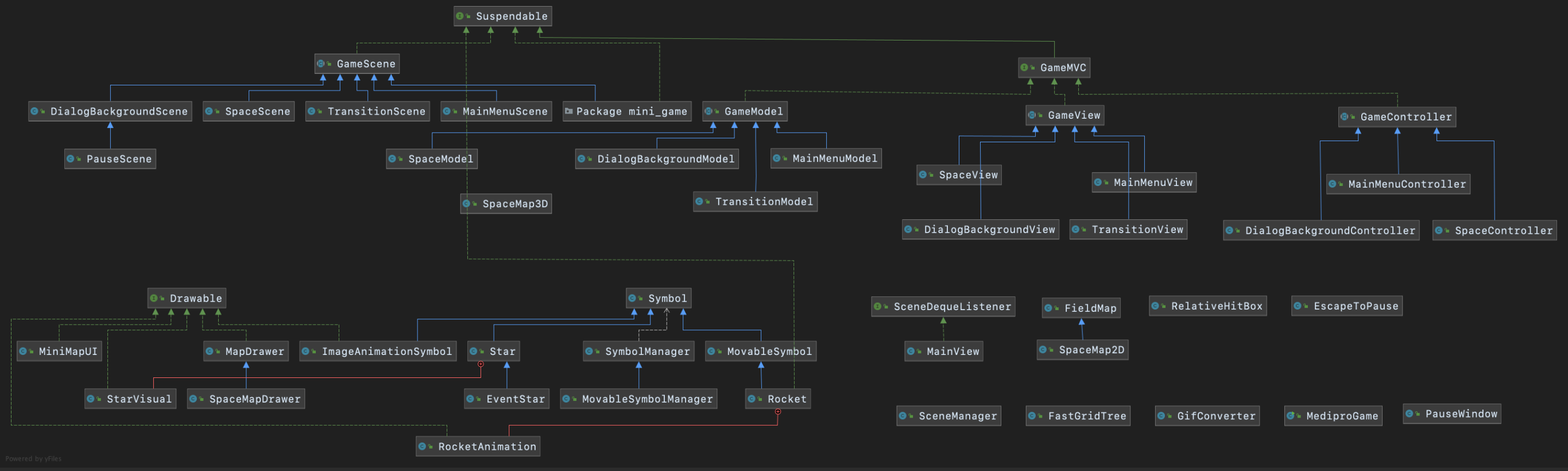
Discordの使用

会話にコードを埋め込めたり、通話ができたり、
画面共有ができたり、いろいろ揃っていて便利

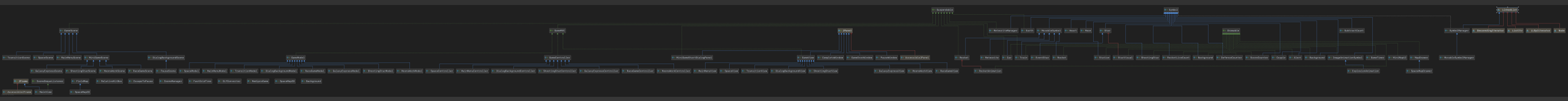
プログラム構成

プログラム構成

ミニゲームを除いたクラス図



全体



MainView
JFrame
RelativeHitBox
EventStar
Star
Symbol
Drawable
SceneManager
FieldMap
Rocket
MovableSymbol
MiniMapUI
IndicatorUI
SpaceMap2D
KeyState
MouseState
Suspendable
GameScene
SpaceScene
GameMVC
GameModel
SpaceModel
JPanel
GameView
SpaceView
Player
GameController
SpaceController
MouseWheel Listener
Key Listener
Mouse Listener

プログラム構成 (メイン部分)

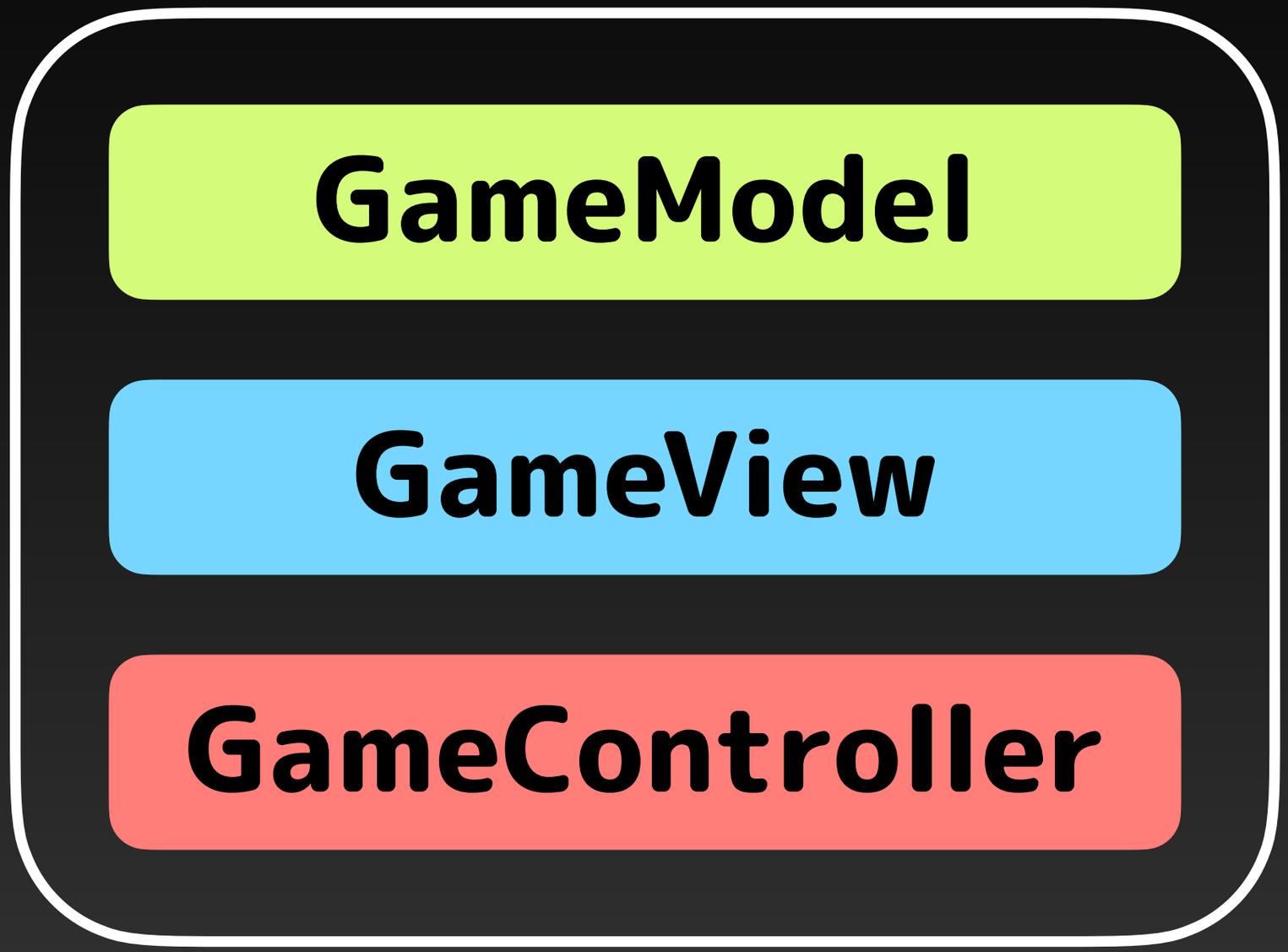
つ ち ア

作業分担

Model

シーン追加の仕組み

GameScene



MVCをまとめて管理するGameSceneクラスを用意

ゲーム本編

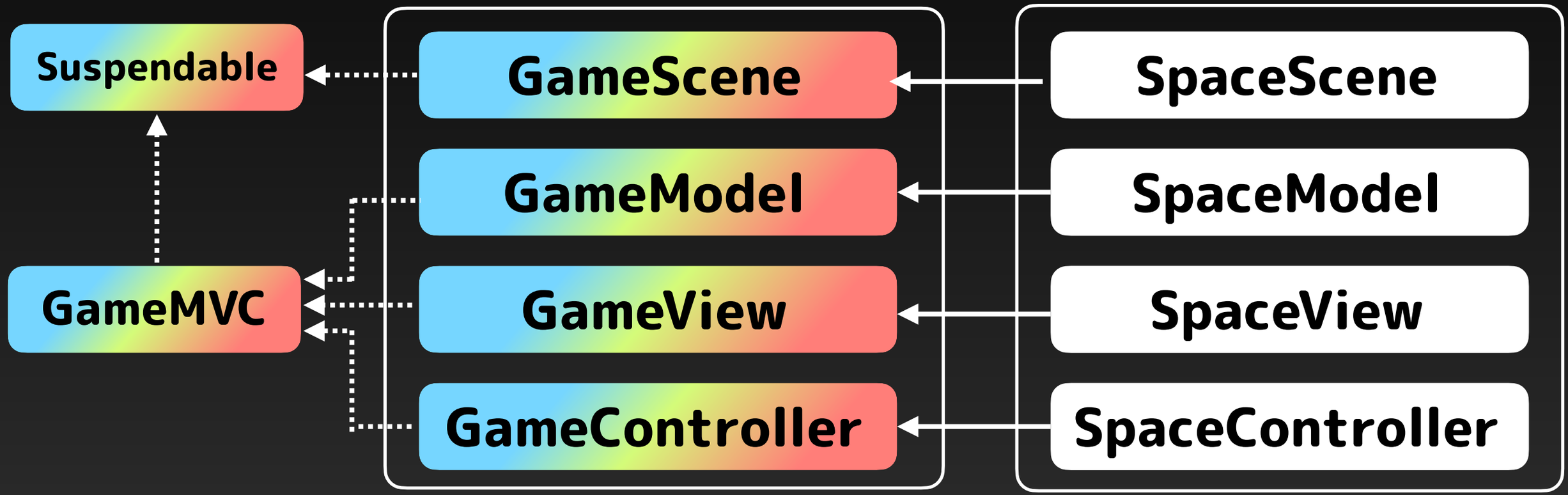
メインメニュー

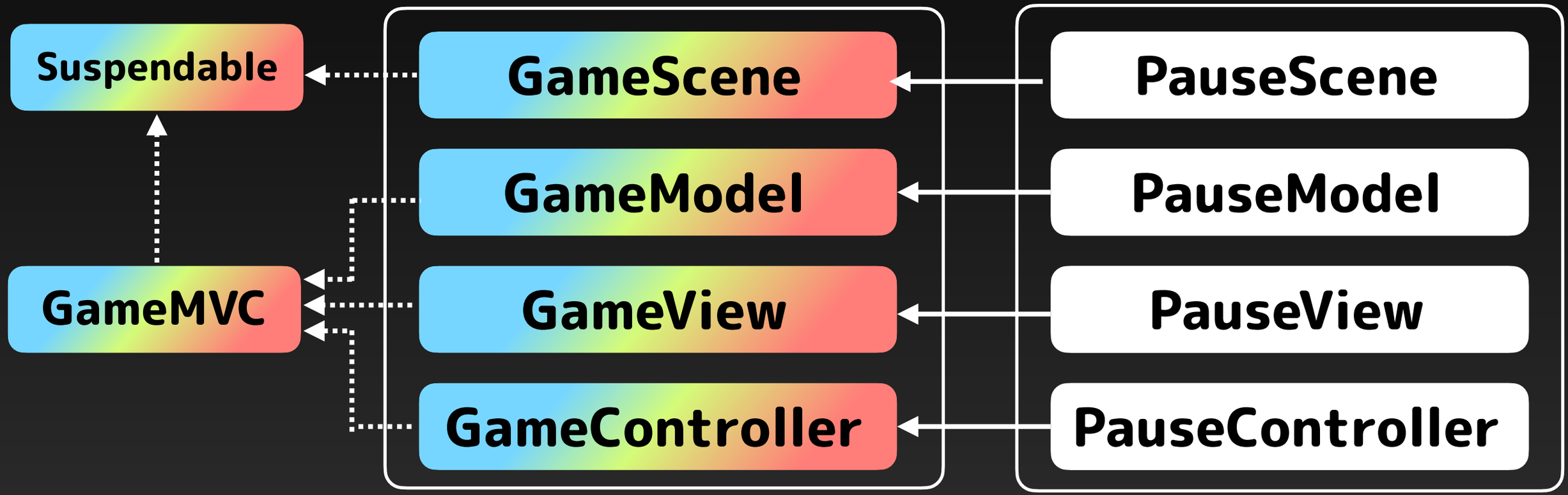
ポーズ画面

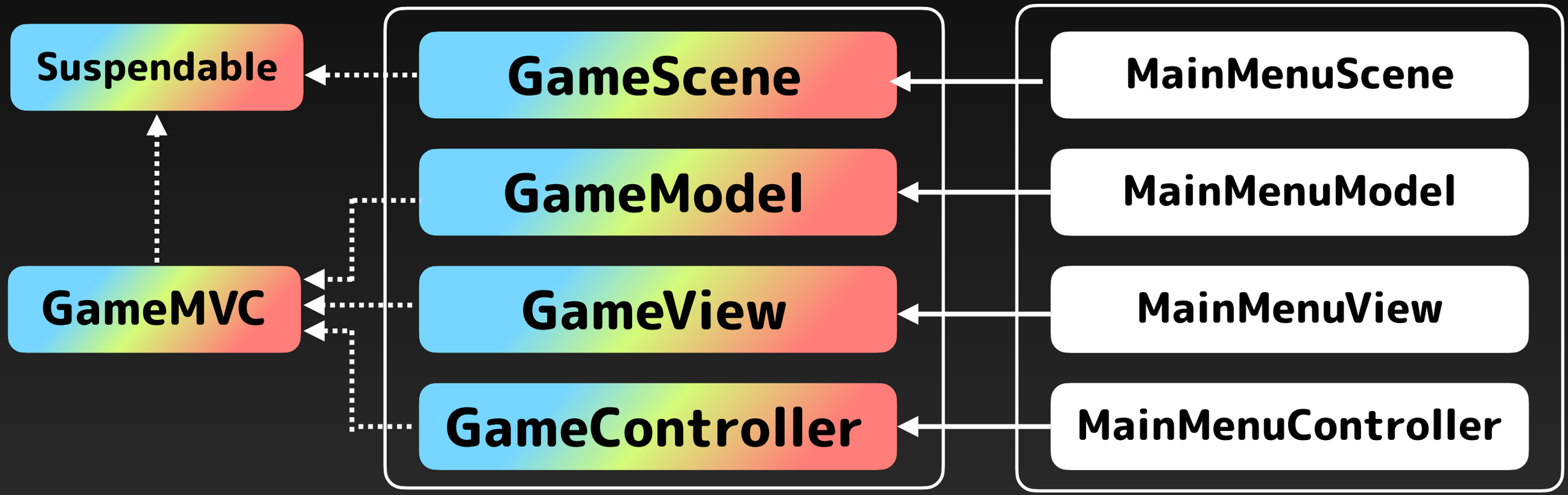
シーン遷移
アニメーション

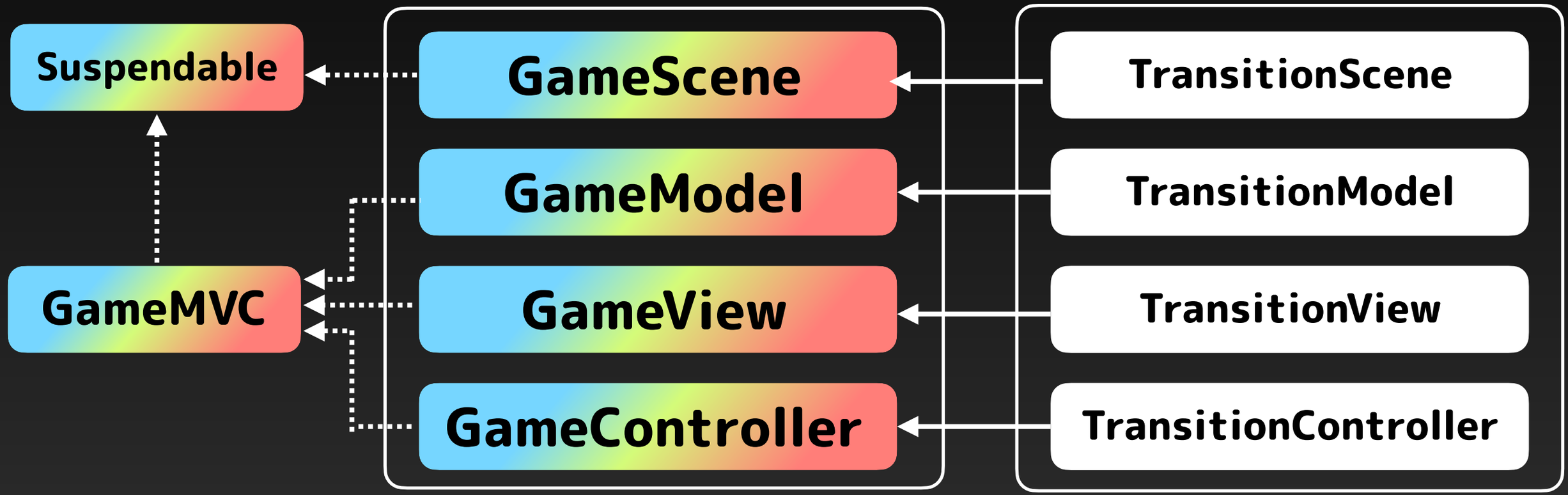
ミニゲーム

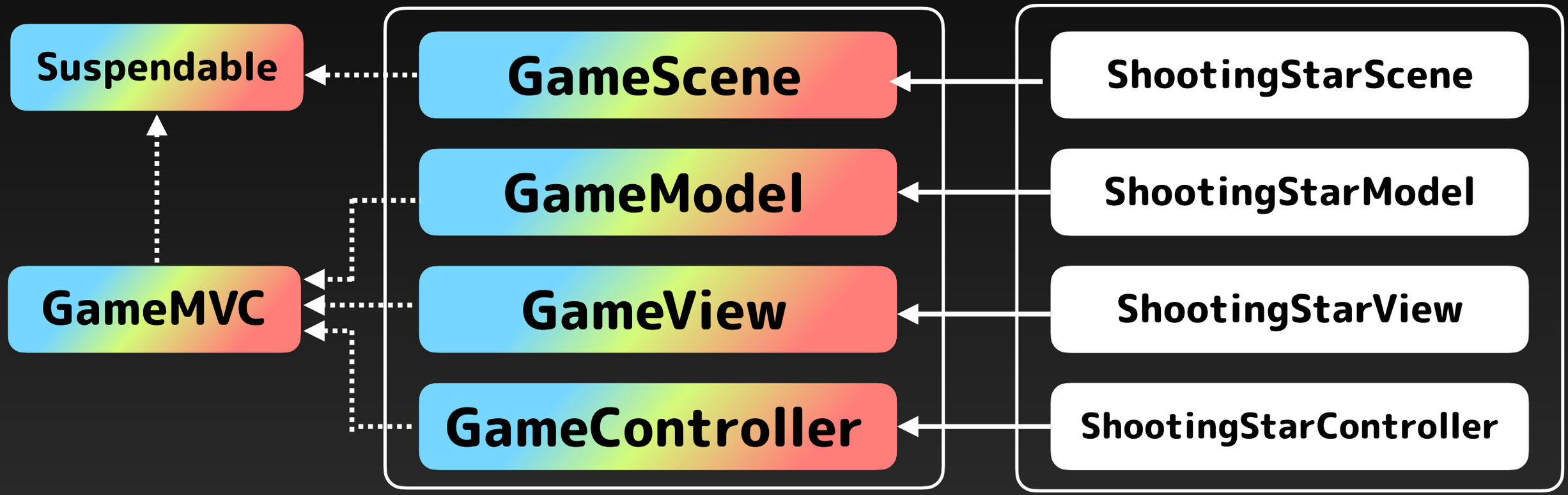
リザルト画面

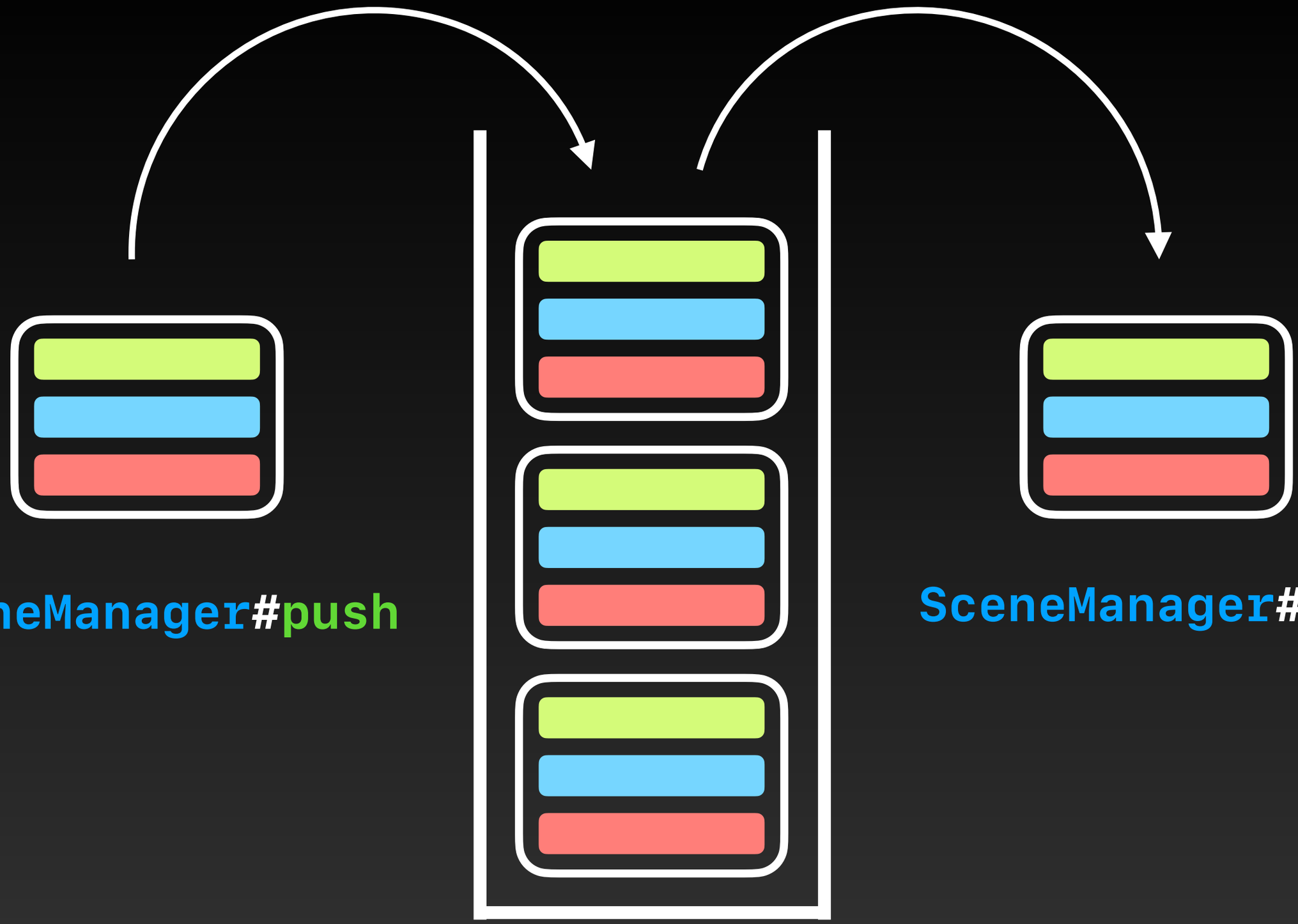

Suspendable
GameScene
SpaceScene
GameModel
SpaceModel
GameMVC
GameView
SpaceView
GameController
SpaceController


Suspendable
GameScene
PauseScene
GameModel
PauseModel
GameMVC
GameView
PauseView
GameController
PauseController


Suspendable
GameScene
MainMenuScene
GameModel
MainMenuModel
GameMVC
GameView
MainMenuView
GameController
MainMenuController


Suspendable
GameScene
TransitionScene
GameModel
TransitionModel
GameMVC
GameView
TransitionView
GameController
TransitionController

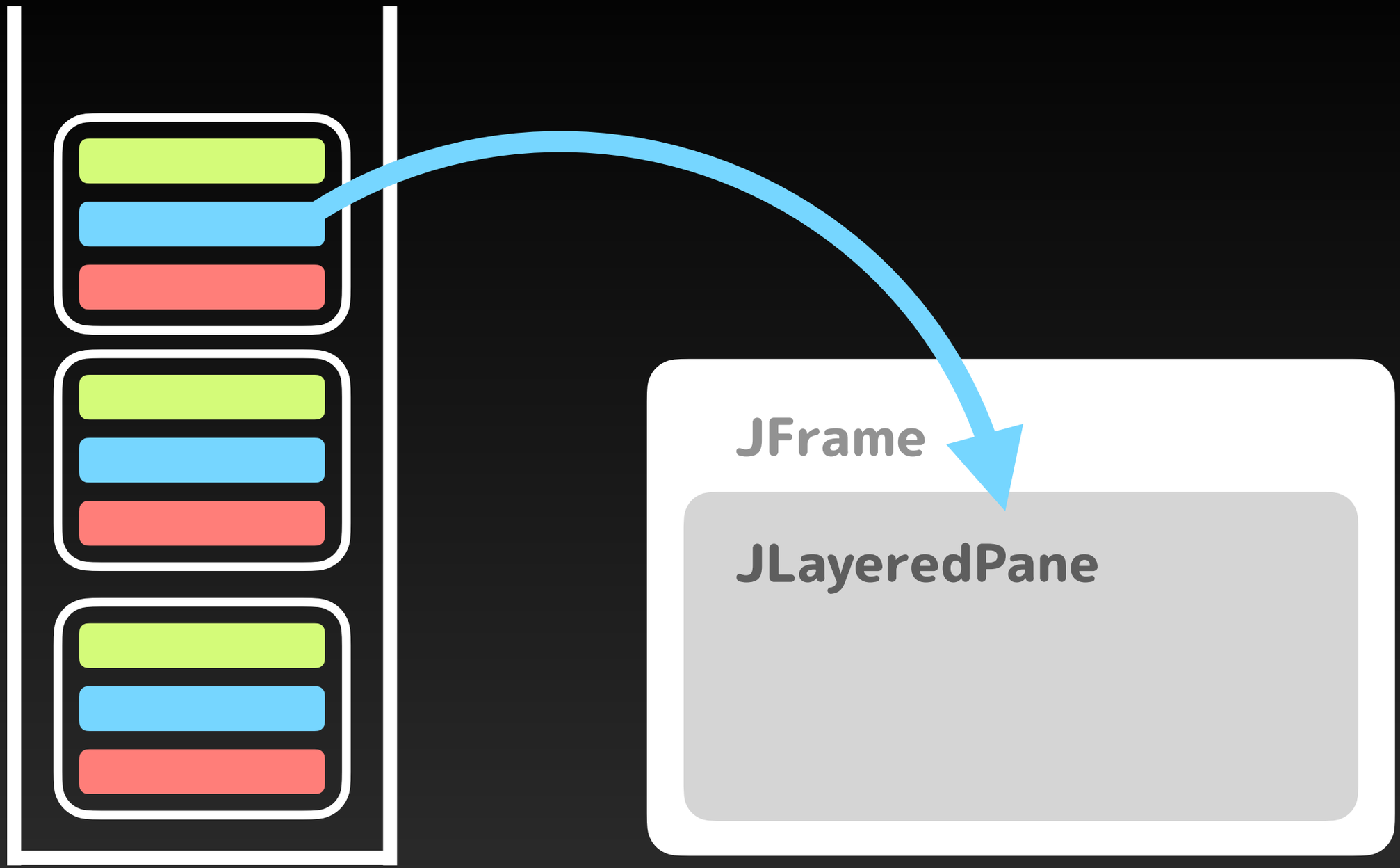

Suspendable
GameScene
ShootingStarScene
GameModel
ShootingStarModel
GameMVC
GameView
ShootingStarView
GameController
ShootingStarController



SceneManager#push

SceneManager#pop

さらにGameSceneをスタックで管理する
SceneManagerクラスを用意



SceneManager

MainView

MainViewはSceneManagerの上部から得られる
GameSceneのViewを最上部に追加

画面に現れる
オブジェクトのクラス

画面に現れるオブジェクト

```
public class Symbol {

    // オブジェクトの描画クラス
    private Drawable drawer;

    // 座標
    private Point2D point;

    // 角度
    private double angleDegrees;

    // 当たり判定
    private RelativeHitBox hitBox;
```

画面に現れるオブジェクト

```
public class Symbol {

    // オブジェクトの描画クラス
    private Drawable drawer;

    // 座標
    private Point2D point;

    // 角度
    private double angleDegrees;

    // 当たり判定
    private RelativeHitBox hitBox;
```

描画用クラス

```
public interface Drawable {
    public void draw(Graphics2D g);
}
```

描画用メソッドを1つ持つインタフェース

描画用クラス(使用例)

```
private Symbol rocket;

@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2 = (Graphics2D) g;
    rocket.getDrawer().draw(g2);
}
```

drawを呼び出すことでオブジェクトを描画できる

画面に現れるオブジェクト

```
public class Symbol {

    // オブジェクトの描画クラス
    private Drawable drawer;

    // 座標
    private Point2D point;

    // 角度
    private double angleDegrees;

    // 当たり判定
    private RelativeHitBox hitBox;
```

画面に現れるオブジェクト

```
public class Symbol {

    // オブジェクトの描画クラス
    private Drawable drawer;

    // 座標
    private Point2D point;

    // 角度
    private double angleDegrees;

    // 当たり判定
    private RelativeHitBox hitBox;
```

画面に現れるオブジェクト

```
public class Symbol {

    // オブジェクトの描画クラス
    private Drawable drawer;

    // 座標
    private Point2D point;

    // 角度
    private double angleDegrees;

    // 当たり判定
    private RelativeHitBox hitBox;
```

当たり判定クラス

```
public class RelativeHitBox extends Area {
    public Area createAbsoluteHitBoxArea(Symbol symbol) {
        double x = symbol.getX();
        double y = symbol.getY();
        double r = symbol.getAngleRadian();
        var af = new AffineTransform();
        af.translate(x, y);
        af.rotate(r);
        return this.createTransformedArea(af);
    }
```

Area に平行移動, 回転を同時に行うメソッドを
追加したクラス

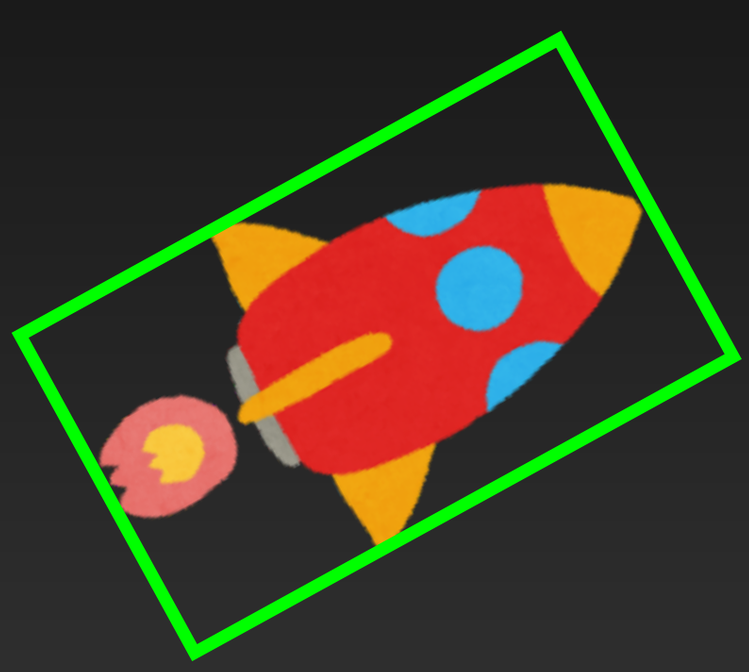
当たり判定クラス

createAbsoluteHitBoxArea(Symbol)

図形は相対的な位置で保存しているので
実際の位置にずらしてあげる

当たり判定クラス

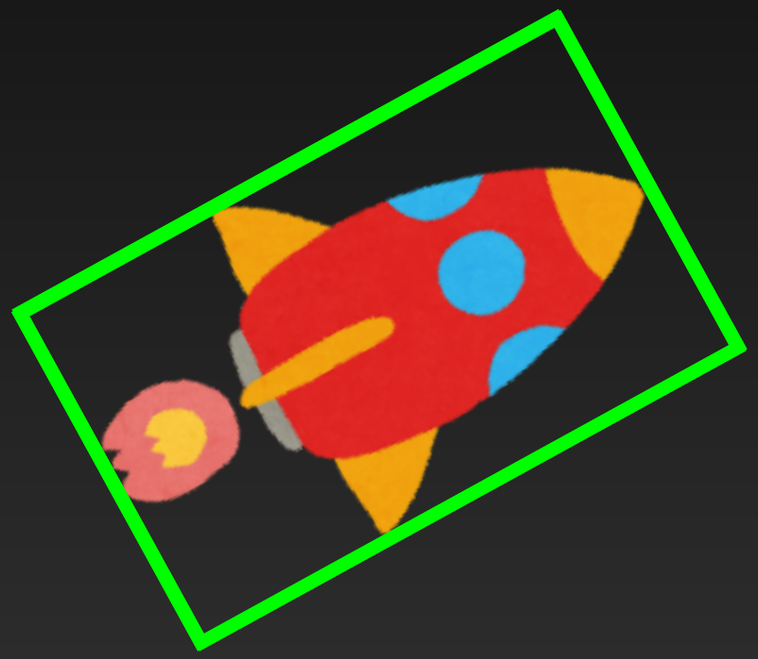
`createAbsoluteHitBoxArea(Symbol)`



図形は相対的な位置で保存しているので
実際の位置にずらしてあげる

当たり判定クラス

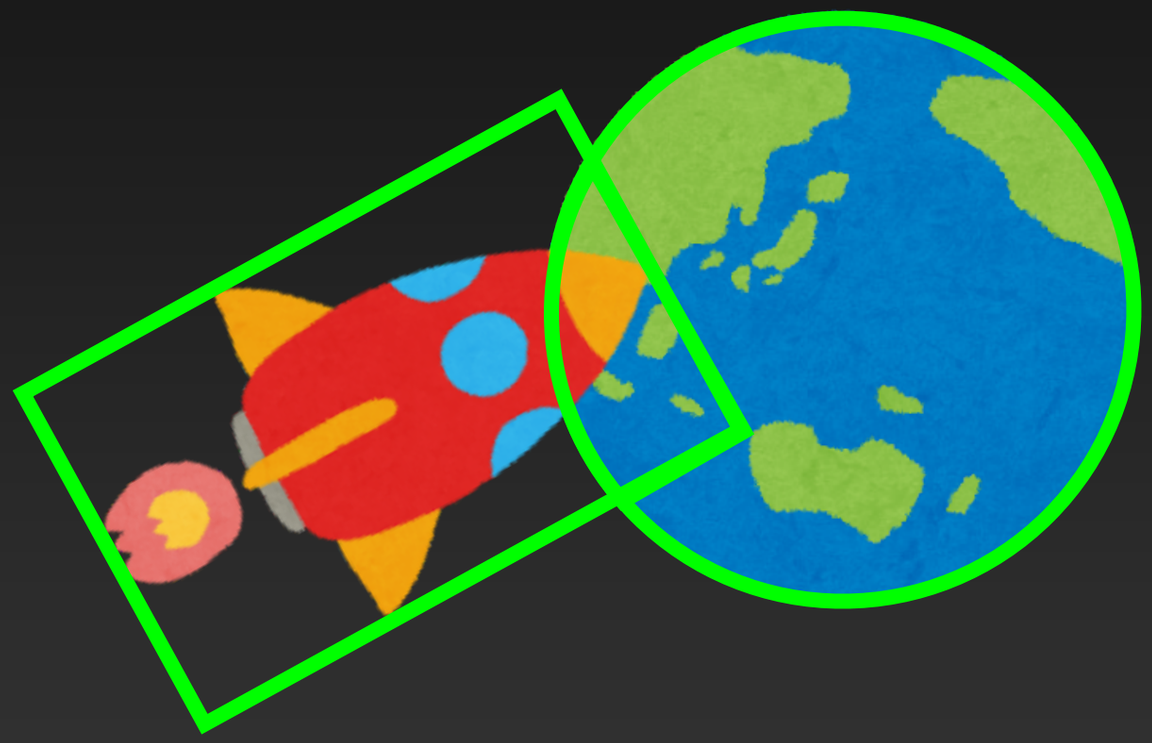
createAbsoluteHitBoxArea(Symbol)



図形は相対的な位置で保存しているので
実際の位置にずらしてあげる

当たり判定クラス

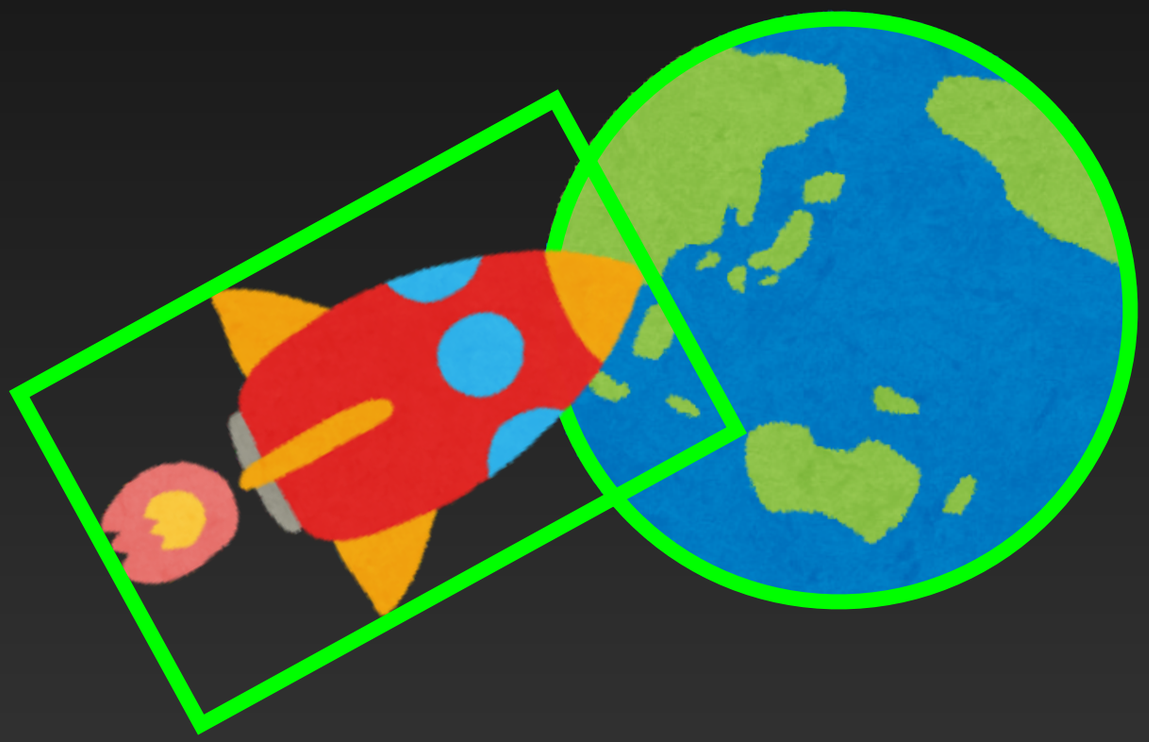
Area#intersect(Area)



Areaには共通部分を抜き出すメソッドが存在

当たり判定クラス

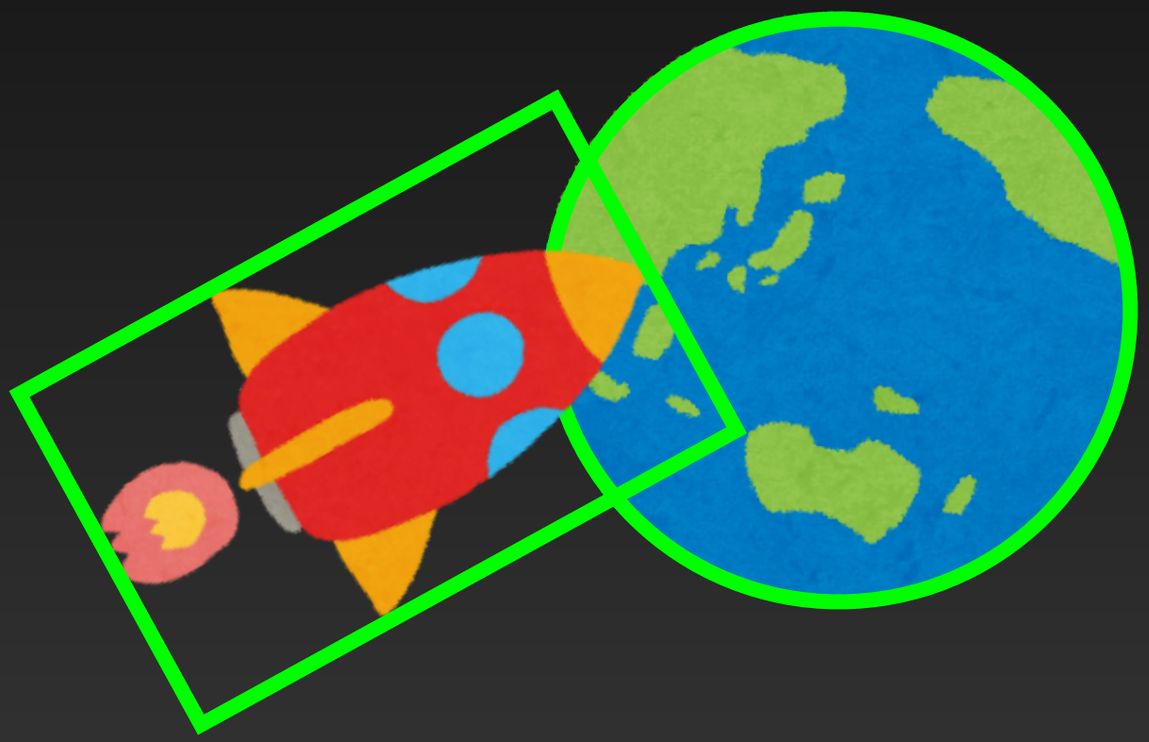
Area#intersect(Area)



Areaには共通部分を抜き出すメソッドが存在

当たり判定クラス

Symbol#touches(Symbol)



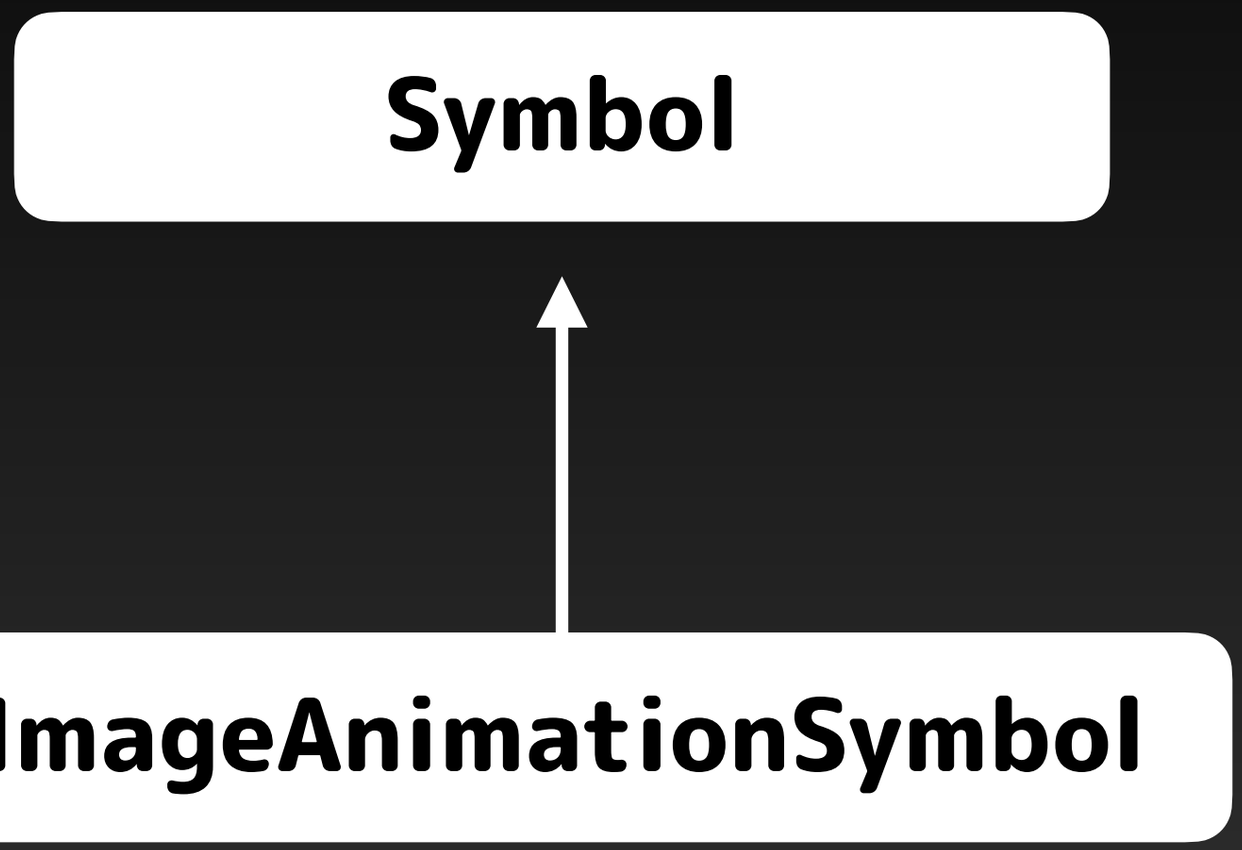
!= null

抜き出した図形がnullでなければ当たったと判定

Symbolクラスの使用例

```
public class Earth extends Symbol implements Drawable {
    private int r = 10;

    public Earth() {
        // 座標の設定
        this.setLocation(10, 10);
        // 当たり判定の範囲設定
        this.setRelativeHitBox(RelativeHitBox.makeCircle(r));
        // 描画内容の設定
        this.setDrawer(this);
    }

    @Override
    public void draw(Graphics2D g) {
        g.setColor(Color.BLUE);
        g.fillOval(getX() - r, getY() - r, 2 * r, 2 * r);
    }
}
```

Symbol関連のクラス



List<Image> を使いアニメーションを
再生・停止する機能をつけたSymbol

Symbol関連のクラス

Symbol

MovableSymbol

速度の概念を足したSymbol
向きからX軸, Y軸方向の速度も計算する

Symbol関連のクラス

LinkedList

SymbolManager<T extends Symbol>

Symbolを継承した型を管理するクラス
条件に合うものを一括削除したりできる

Symbol関連のクラス

LinkedList

↑

SymbolManager<T extends Symbol>

↑

MovableSymbolManager

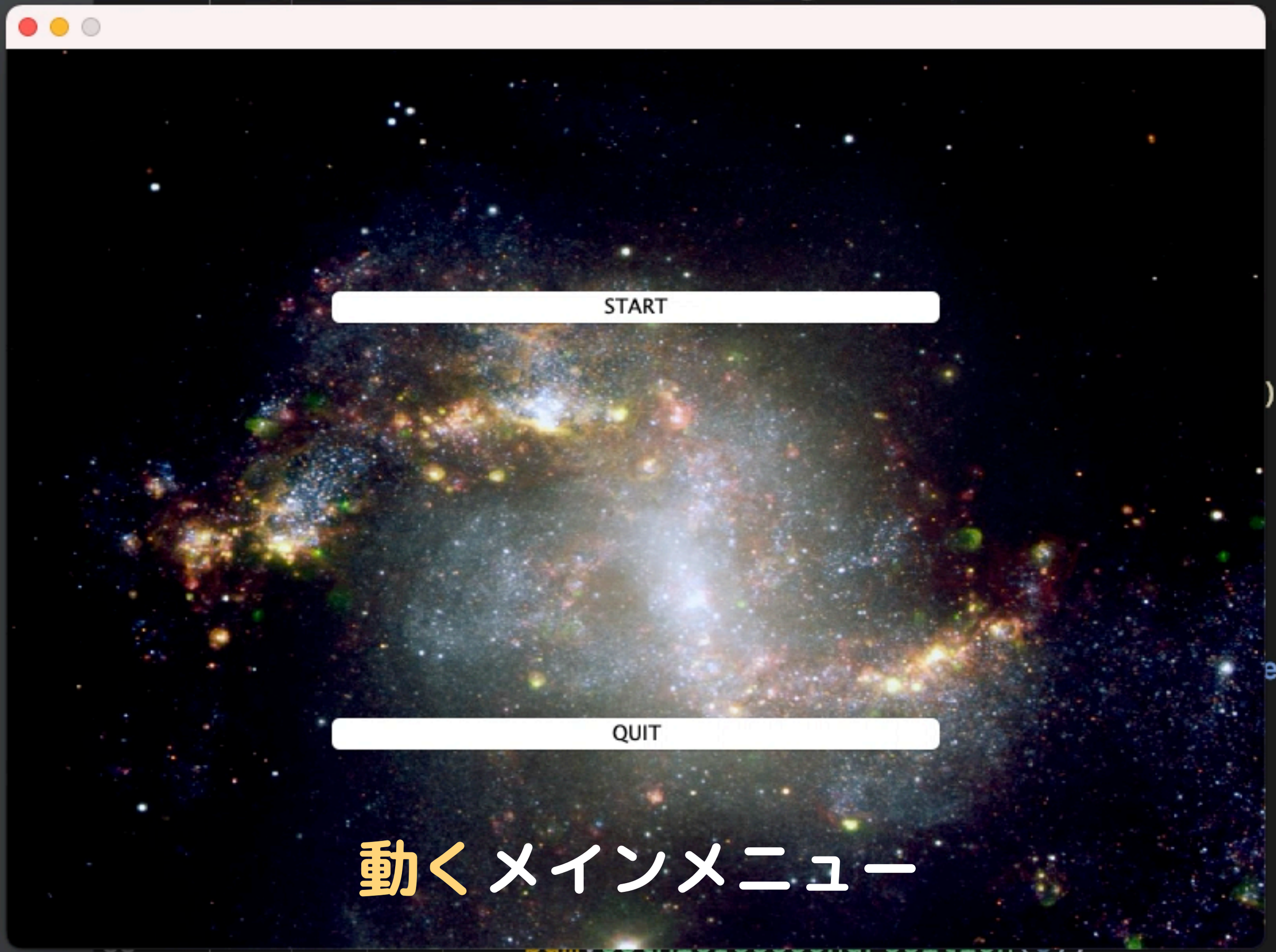
MovableSymbolを管理するクラス
Timerを用いて登録されたSymbolを同時に動かせる

[redacted]のAPI後世の為に公開すべき
メディプロの公式資料にするべき

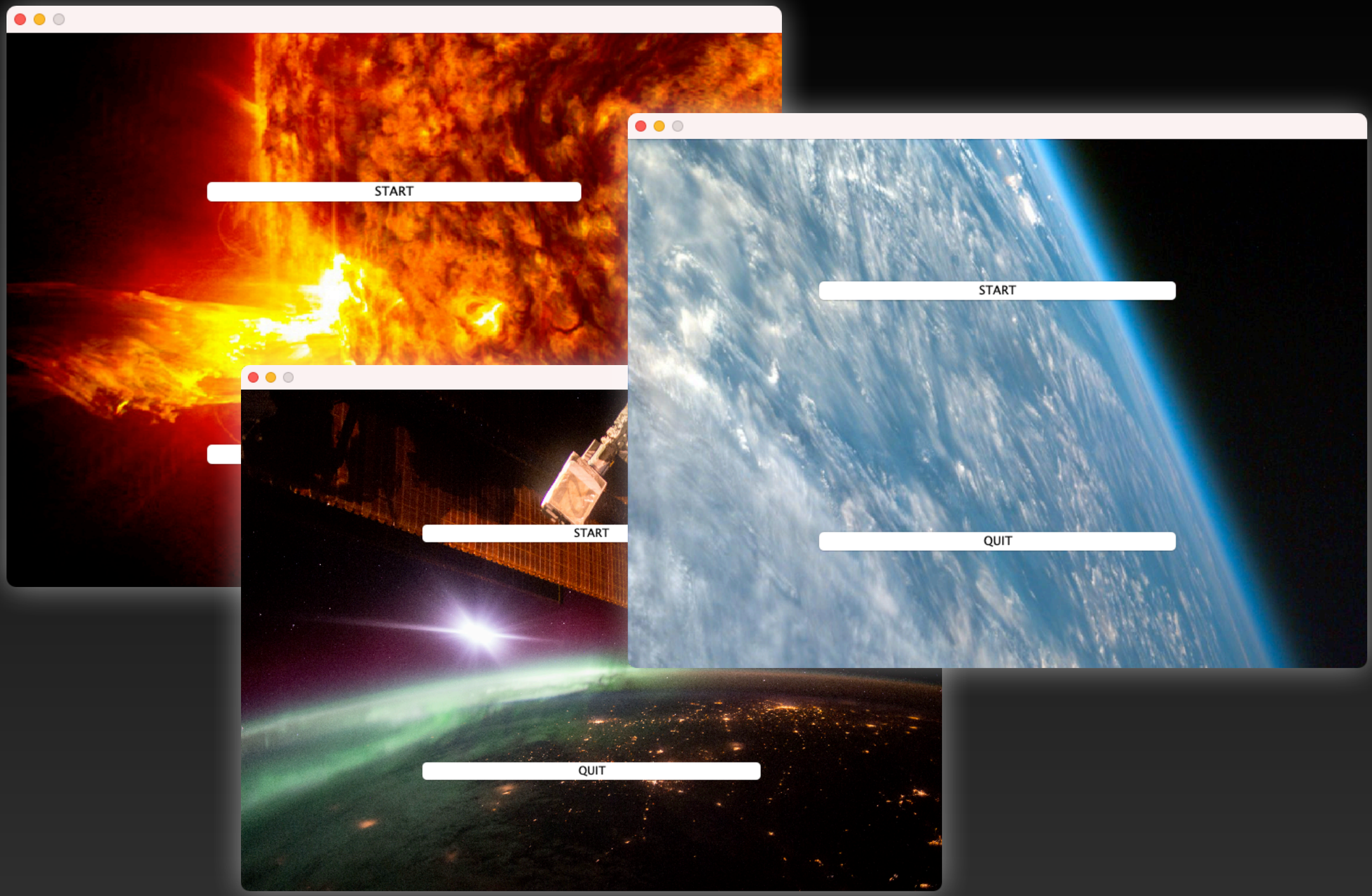
[redacted]のAPIが優秀すぎてjava何も分からんくてもコード書ける、これもうペタペタプログラミングだろ

😄

View



動くメインメニュー



ランダムな背景画像

ロケットのアニメーション・UIの作成

Controller

Controllerの責務

1. プレイヤーが**思った通りに**
   操作できること

2. 環境によって**差を生まない**こと

KeyListener#keyPressed だけを使った実装の問題点

1. 一つのキーのみしか受け付けない
   → **同時押し不可能**

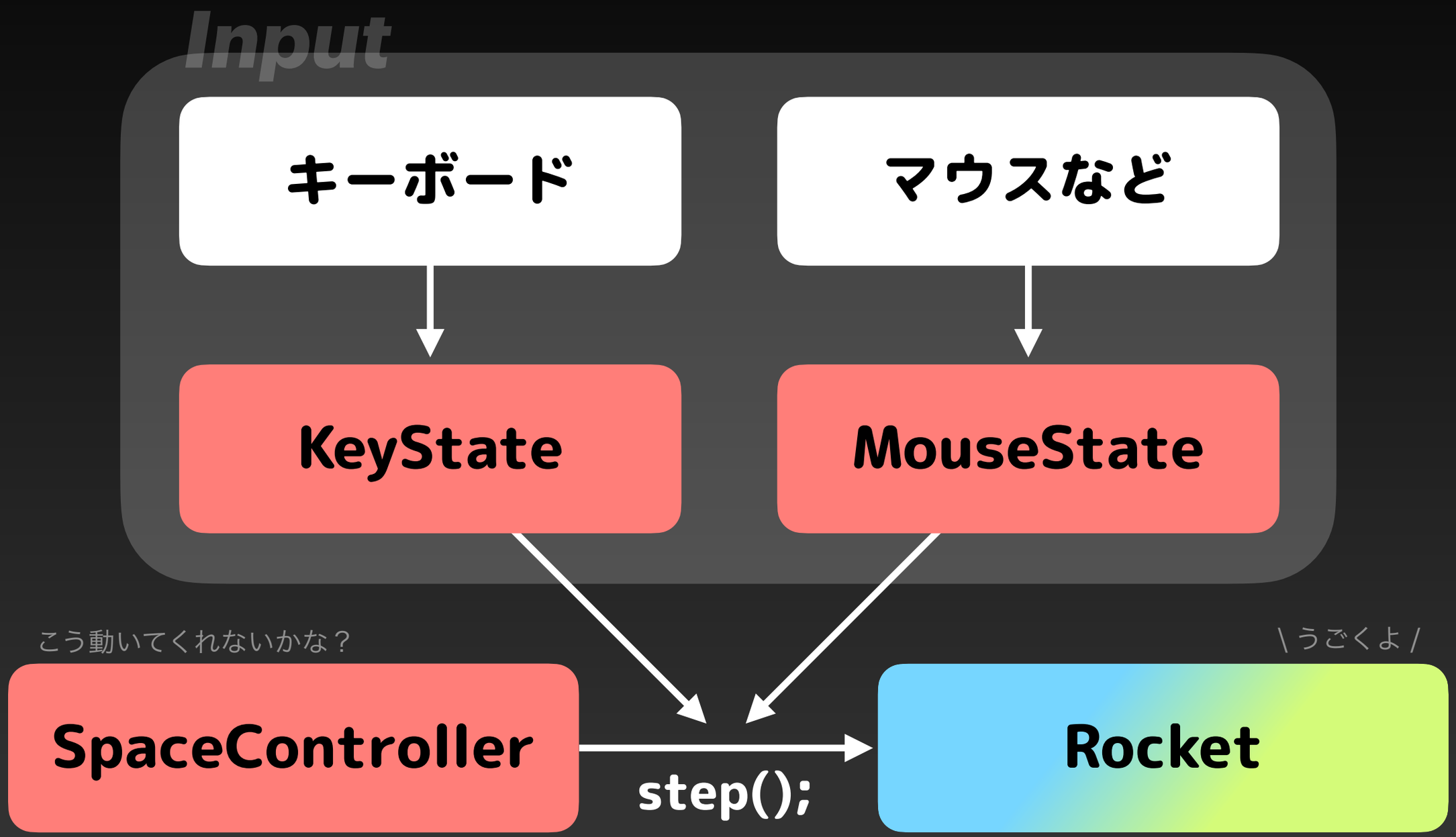
2. PCによって**発火間隔が異なる**

3. 1回目と2回目以降の発火の間に**ラグ**
   → **操作感の低下**

操作の入力と処理の分離

```
public class SpaceController
                    extends GameController {

    // キー入力情報の保持
    private KeyState keyState;

    // マウス入力情報の保持
    private MouseState mouseState;

    // 1フレームで行う処理
    private void step() { …… }

    // stepを一定間隔で実行するタイマー
    private Timer stepTimer;
```

操作の入力と処理の分離

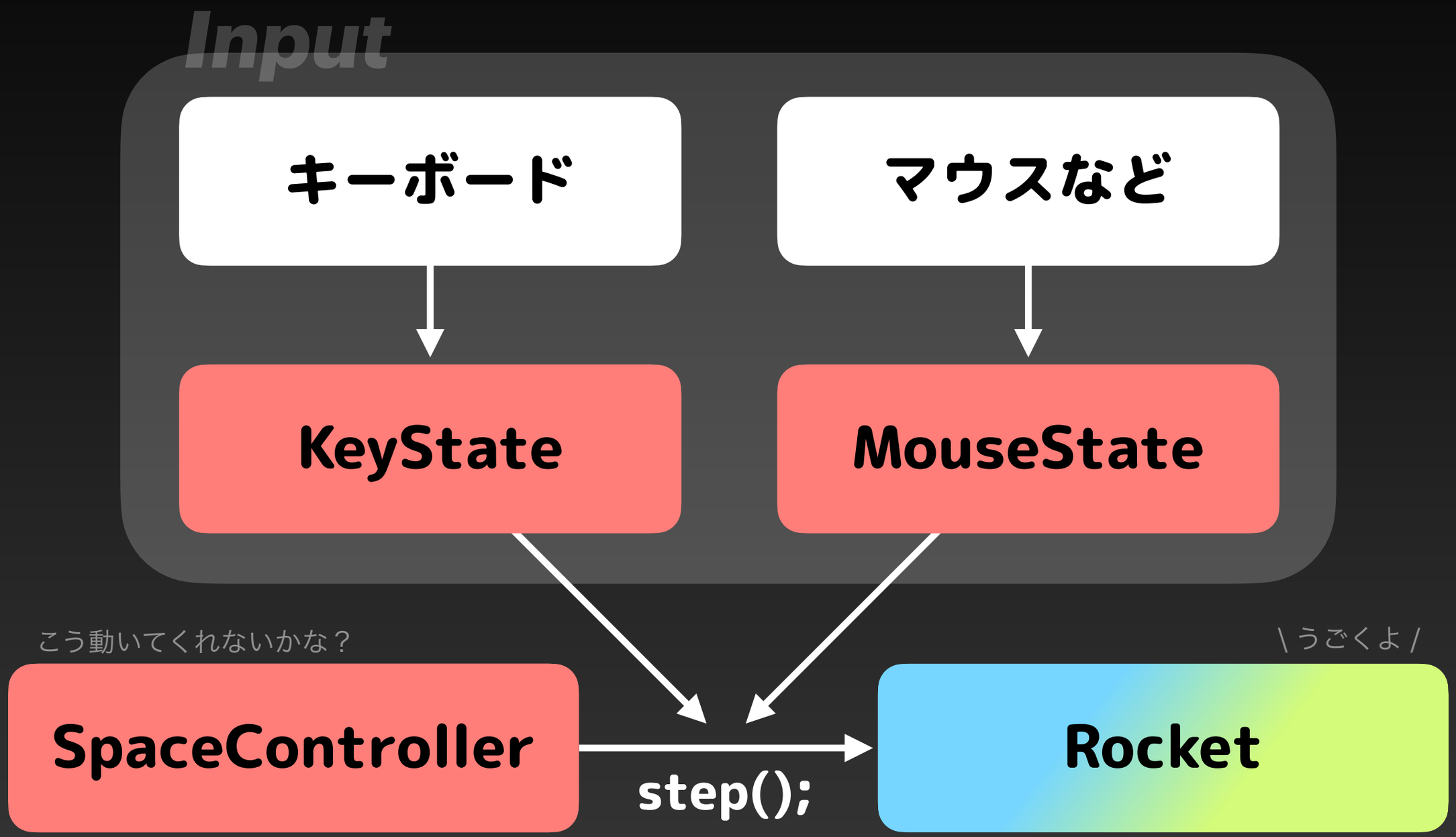


定期的に入力されたキーの情報を受け取り、Rocketに指示

```
class KeyState implements KeyListener {

    // 押されているキーの集合
    private Set<Integer> states;

    // 押したら追加
    @Override
    public void keyPressed(KeyEvent e) {
        states.add(e.getKeyCode());
    }

    // 離したら削除
    @Override
    public void keyReleased(KeyEvent e) {
        states.remove(e.getKeyCode());
    }
```

```
class MouseState implements MouseListener {

    // 押されているボタンの集合
    private boolean clicked, wheeled, wheelUp;

    // 押したら追加
    @Override
    public void mousePressed(MouseEvent e) {
        // メンバ変数に変更を加える
    }

    // 離したら削除
    @Override
    public void mouseReleased(MouseEvent e) {
        // メンバ変数に変更を加える
    }
```

操作の入力と処理の分離



入力されたキーの情報を受け取り、定期的にRocketに指示

```
private Timer t = new Timer(200, e -> step());

private void step() {

    boolean Wが押されている =
        keyState.isPressed(KeyEvent.VK_W);
    boolean Sが押されている =
        keyState.isPressed(KeyEvent.VK_S);

    if(Wが押されている || Sが押されている) {
        if(Wが押されている) rocket.加速();
        if(Sが押されている) rocket.減速();
    } else if (mouseState.isClicked()) {
        rocket.set速度(カーソルとの距離);
    }
}
```

KeyListener#keyPressed だけを使った実装の問題点

1. 一つのキーのみしか受け付けない
   → 同時押し不可能

2. PCによって挙動が異なる

3. 1回目と2回目以降の発火の間にラグ
   → 操作感の低下

解決

解決

解決

デモ・質疑応答