2024-04-18

バージョン: 1.0

デザインパターン

ステート

Design Pattern | State

Shozo Tanaka

目次

図目次

はじめに

ゲームに登場するキャラクタには「走ったり」「歩いたり」「立ち止まったり」するなど様々な「状態(State)」があります。この状態をクラスにしてキャラクタを作ることで効率のよいキャラクタを生成できます。この状態をクラスにするパターンを「ステートパターン」と言います。

1. レガシーコード

「State Legacy」というプロジェクトは、キャラクタ(マリオ)がキー入力によりジャンプしたり、屈んだりするプログラムです。プロジェクトをダウンロードして、ビルドしてみましょう。実行すると、下記のような画面になります。このプログラムはデザインパターンを使用しない、ごく一般的なプログラムです。

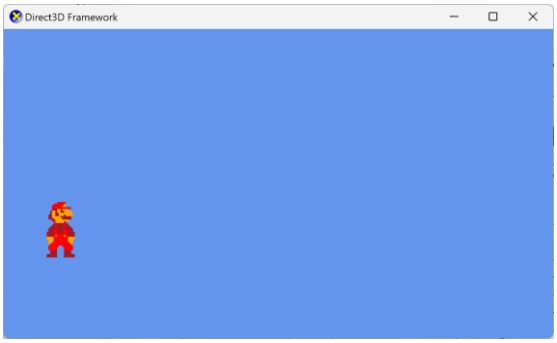

図 1 Mario の動作

Mario クラスのヘッダーファイルを示します。

```
001  #pragma once
002  #ifndef MARIO_DEFINED
003  #define MARIO_DEFINED
004
005  #include "SpriteBatch.h"
006  #include "WICTextureLoader.h"
007  #include "Graphics.h"
008
009  class Mario
010  {
011          // プレイヤーの状態
012          enum State
013          {
014                  STANDING_STATE,
015                  JUMPING_STATE,
016                  DUCKING_STATE,
017          };
018
019  public:
020          // コンストラクタ
021          Mario(float x, float y);
022          // デストラクタ
023          ~Mario();
024          // 初期化する
025          void Initialize();
026          // 更新する
027          void Update(const DirectX::Keyboard::KeyboardStateTracker& keyboardStateTracker);
028          // 描画する
029          void Render();
030
```

```
031 private:
032         // グラフィックス
033         Graphics* m_graphics = Graphics::GetInstance();
034 
035         // スタンディング
036         Microsoft::WRL::ComPtr<ID3D11ShaderResourceView> m_standing;
037         // ジャンプ
038         Microsoft::WRL::ComPtr<ID3D11ShaderResourceView> m_jumping;
039         // ダウニング
040         Microsoft::WRL::ComPtr<ID3D11ShaderResourceView> m_downing;
041         // 現在のテクスチャ
042         Microsoft::WRL::ComPtr<ID3D11ShaderResourceView> m_currentTexture;
043 
044         // 表示位置
045         float m_x, m_y;
046         // 速度
047         float m_vx, m_vy;
048         // 加速度
049         float m_ax, m_ay;
050         // プレイヤーの状態
051         State m_currentState;
052 };
053 #endif              // MARIO_DEFINED
```

Mario クラスの実装ファイルは次のようになります。

```
001 #include "pch.h"
002 #include "Mario.h"
003 #include "DeviceResources.h"
004 
005 // コンストラクタ
006 Mario::Mario(float x, float y)
007 :
008 m_currentState(State::STANDING_STATE),
009 m_currentTexture(nullptr),
010 m_standing(nullptr),
011 m_jumping(nullptr),
012 m_downing(nullptr),
013 m_x(x),
014 m_y(y),
015 m_vx(0.0f),
016 m_vy(0.0f),
017 m_ax(0.0f),
018 m_ay(0.2f)
019 {
020 }
021 
022 // デストラクタ
023 Mario::~Mario()
024 {
025 }
026 
027 // 初期化する
028 void Mario::Initialize()
029 {
030         // デバイスを取得する
```

```
031         auto* device = m_graphics->GetInstance()->GetDeviceResources()->GetD3DDevice();
032
033         // テクスチャをロードする
034         DirectX::CreateWICTextureFromFile(device, L"Resources¥¥Image¥¥standing.png", nullptr,
035           m_standing.GetAddressOf());
036         DirectX::CreateWICTextureFromFile(device, L"Resources¥¥Image¥¥jumping.png", nullptr,
037           m_jumping.GetAddressOf());
038         DirectX::CreateWICTextureFromFile(device, L"Resources¥¥Image¥¥downing.png", nullptr,
039           m_downing.GetAddressOf());
040
041         // プレイヤの初期状態を「立ち」に設定する
042         m_currentState = State::STANDING_STATE;
043 }
044
045 // 更新する
046 void Mario::Update(const DirectX::Keyboard::KeyboardStateTracker& keyboardStateTracker)
047 {
048         // プレイヤの状態に合わせた行動を行う
049         switch (m_currentState)
050         {
051                 case State::STANDING_STATE: // 立っている場合
052                 if (keyboardStateTracker.IsKeyPressed(DirectX::Keyboard::Keys::Space))
053                 {
054                         // 速度を設定する
055                         m_vy = -10.0f;
056                         // プレイヤの状態を「ジャンプ」に設定する
057                         m_currentState = State::JUMPING_STATE;
058                 }
059                 else if (keyboardStateTracker.IsKeyPressed(DirectX::Keyboard::Keys::Down))
060                 {
061                         // プレイヤの状態を「屈む」に設定する
062                         m_currentState = DUCKING_STATE;
063                 }
064                 // テクスチャを「立ち」に設定する
065                 m_currentTexture = m_standing;
066                 break;
067
068         case State::JUMPING_STATE: // ジャンプする状態
069                 if (m_y == 400.0f)
070                 {
071                         // プレイヤの状態を「立ち」に設定する
072                         m_currentState = State::STANDING_STATE;
073                 }
074                 // テクスチャを「ジャンプ」に設定する
075                 m_currentTexture = m_jumping;
076                 break;
077
078         case State::DUCKING_STATE: // 屈む場合
079                 if (keyboardStateTracker.IsKeyReleased(DirectX::Keyboard::Keys::Down))
080                 {
081                         // プレイヤの状態を「立ち」に設定する
082                         m_currentState = State::STANDING_STATE;
083                 }
084                 // テクスチャを「屈む」に設定する
085                 m_currentTexture = m_downing;
086                 break;
087 }
088
```

```
089         // 速度に加速度を加算する
090         m_vx += m_ax;
091         m_vy += m_ay;
092         // 位置に速度を加算する
093         m_x += m_vx;
094         m_y += m_vy;
095         // 地面を判定する
096         if (m_y > 400.0f)
097         {
098                 m_y = 400.0f;
099                 m_vy = 0.0f;
100         }
101 }
102
103 // 描画する
104 void Mario::Render()
105 {
106         // スプライトバッチを開始する
107         m_graphics->GetSpriteBatch()->Begin();
108         // スプライトを描画する
109         m_graphics->GetSpriteBatch()->Draw(m_currentTexture.Get(), DirectX::SimpleMath::Vector2(m_x, m_y));
110         // スプライトバッチを終了する
111         m_graphics->GetSpriteBatch()->End();
112 }
```

1.1 考察

Mario クラスにさらにマリオの動作を加えていくと、どのような問題が起きそうですか？

このような問題を解決するためのデザインパターンとしてステートパターンを活用することができます。

2. ステートパターン

ステートパターンは、「状態をクラスとして表現し、状態ごとに振る舞いを切り替えられる」ようにするデザインパターンです。
ステートは、「オブジェクトの内部状態が変化したときに、オブジェクトが振る舞いを変える」ようにします。クラス内では振る舞いの変化を記述せず、「状態を表すオブジェクトを導入する」ことで実現します。ゲーム開発では必修のデザインパターンの一つです。

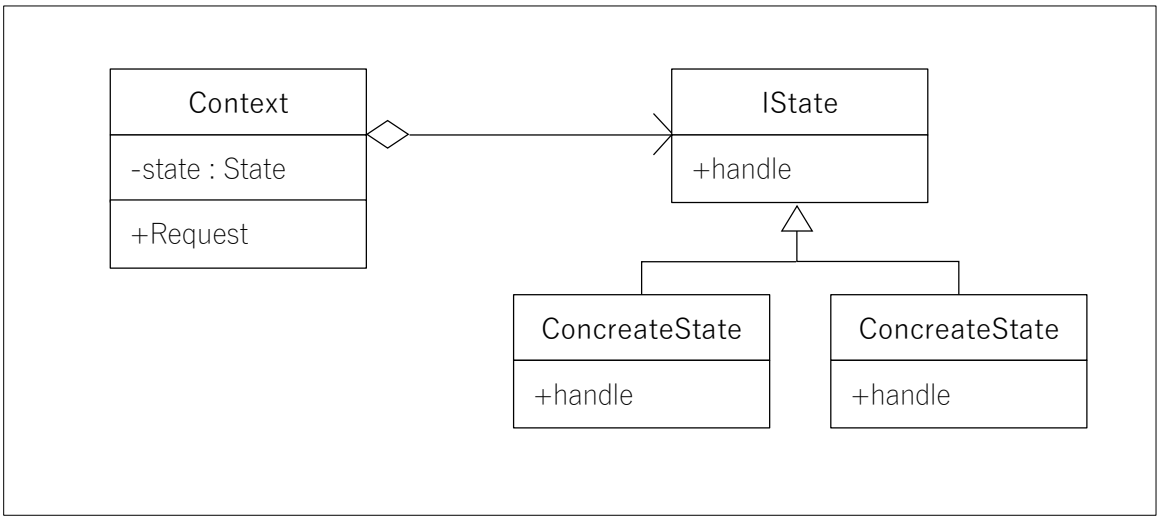

図 2 ステートのクラス

IState クラス

状態を表す「IState インタフェースを宣言し、handle は状態固有の振る舞いをおこなう」メソッドを宣言します。Istate インターフェースには状態に必要となる純粋仮想関数を宣言します。

ConcreteState クラス

ConcreteState クラスは「IState インタフェースの派生クラスでインターフェースを実装するクラス」です。ConcreteState クラスには状態ごとの具体的な処理内容を記述します。マリオの場合、Running や Jumping が実装クラスになります。

Context クラス

State クラスのオブジェクトを Context オブジェクト内部に保持し、具体的な処理をそのオブジェクトに委譲します。これにより、ConcreteState クラスに依存することなく、ConcreteState オブジェクトを素早く切り替え実行することができます。Context クラスは Player クラスなどキャラクタクラスとなります。

3. 状態遷移

これから作成するプレイヤには、４つの状態(「立つ(Standing)」「屈む(Downing)」「走る(Running)」「ジャンプ(Jumping)」)を所有します。次にマリオがどのように状態を遷移するかについて表にまとめます。

状態遷移

・「立つ」状態から「屈む」「走る」「ジャンプ」状態に遷移する
・「屈む」状態から「立つ」状態に遷移する
・「走る」状態から「立つ」「ジャンプ」状態に遷移する
・「ジャンプ」状態「走る」「立つ」状態に遷移する

状態		状態	条 件
立つ	➡	屈む	① [↓]キーを押し下げている
		走る	③ [←] [→]キーを押し下げている
		ジャンプ	⑤ [SPACE]キーを押し下げている
屈む		立つ	② [↓]キーを解放した
走る		立つ	④ [←] [→]キーを解放した
		ジャンプ	⑦ [SPACE]キーを押し下げている
ジャンプ		走る	⑧ プレイヤが地面に着地し「←」「→」キーを押し下げている
		立つ	⑥ プレイヤが地面に着地し「←」「→」キーを解放している

状態遷移表

状態遷移を図で表現すると次のようになります。

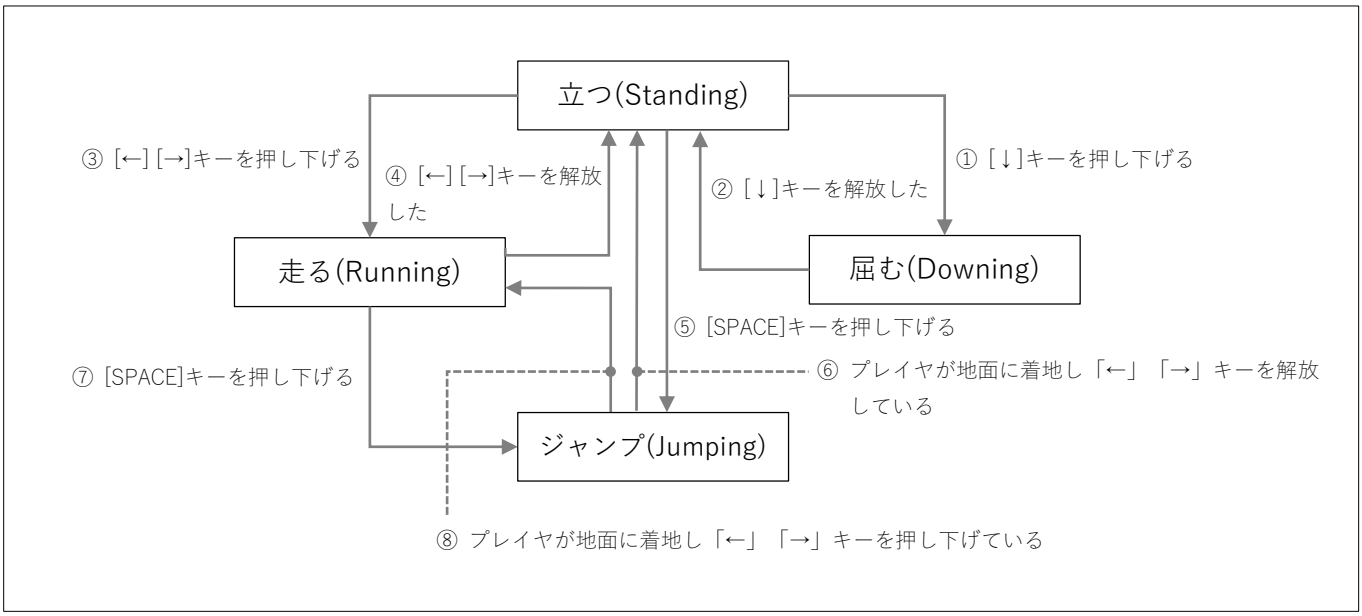

図 3 状態遷移図

4. State パターンの作り方

ステートパターンの作り方の手順にまとめます。

State パターンの作成手順

① IState インタフェースを定義する。
② 状態を表す実装クラスを定義し、クラスを実装する。
③ Player クラスで状態オブジェクトを生成する。
④ ChangeState 関数で遷移する状態へのポインタを IState ポインタに設定する。
⑤ IState ポインタに格納されたオブジェクトの更新と描画をおこなう。

5. ステートインタフェースを定義する

IState インタフェースを定義し、キャラクタの状態を処理するときに必要となる純粋関数を宣言します。

```
001  #pragma once
002  #ifndef STATE_DEFINED
003  #define STATE_DEFINED
004  #include "StepTimer.h"
005
006  // IState インターフェイスを定義する
007  class IState
008  {
009  public:
010          // 初期化する
011          virtual void Initialize() = 0;
012          // 処理を実行する
013          virtual void Update(const DirectX::Keyboard::KeyboardStateTracker& keyboardStateTracker) = 0;
014          // テクスチャを描画する
015          virtual void Render() = 0;
016          // 後始末をする
017          virtual void Finalize() = 0;
018          // 仮想デストラクタ
019          virtual ~IState() = default;
020  };
021
022  #endif // STATE_DEFINED
```

IState インタフェース(IState.h)

6. 「立つ」状態を作る

IState インタフェースを宣言後は、マリオの「立つ」状態クラスを作成します。

6.1 Standing クラスを宣言する

「立つ」状態クラスである「Standing」クラスを定義します。インタフェースを実装する関数を宣言します。メンバー変数としては Mario クラスのインスタンスへのポインタ、テクスチャ変数の参照用の変数を宣言します。

```
001  #pragma once
002  #ifndef STANDING_DEFINED
003  #define STANDING_DEFINED
004  #include "IState.h"
005  #include "PlayScene.h"
006  #include "Mario.h"
007
008  class Mario;
009
010  // Standing クラスを定義する
011  class Standing : public IState
012  {
013  public:
014          // コンストラクタ
015          Standing(Mario* mario);
016          // デストラクタ
017          ~Standing();
018          // Standing オブジェクトを初期化する
019          void Initialize() override;
020          // 立ちを実行する
021          void Update(const DirectX::Keyboard::KeyboardStateTracker& keyboardStateTracker) override;
022          // 立ちを描画する
023          void Render() override;
024          // 後始末をする
025          void Finalize() override;
026
027  private:
028          // グラフィックス
029          Graphics* m_graphics = Graphics::GetInstance();
030          // マリオ
031          Mario* m_mario;
032          // 立ちテクスチャ
033          ID3D11ShaderResourceView* m_standingTexture;
034  };
035
036  #endif // STANDING_DEFINED
```

Standing.h

6.2 Standing クラスを実装する

Standing クラスの実装をします。Standing クラスでは Mario クラスのインスタンスへのポインタを取得し、マリオの情報を取得します。同様に、マリオが所有する様々なテクスチャは Mario クラスで宣言しているため、Mario クラスのインスタンスへのポインタを通じて取得できます。
Standing クラスの Update 関数ではプレイヤの「立つ」状態の処理を実装します。状態遷移図の「立つ」状態から遷移する Downing クラスと Jumping クラスは Standing クラスの実装後に追加します。
Render 関数ではマリオの「立つ」状態を描画します。

```
001  #include "pch.h"
002  #include "Standing.h"
003
004  class PlayScene;
005
006  // コンストラクタ
007  Standing::Standing(Mario* mario)
008  :
009  m_mario(mario),
010  m_standingTexture(m_mario->GetStandingTexture())
011  {
012  }
013
014  // デストラクタ
015  Standing::~Standing()
016  {
017  }
018
019  // 「立ち」状態を初期化する
020  void Standing::Initialize()
021  {
022  }
023
024  // 「立ち」状態を更新する
025  void Standing::Update(const DirectX::Keyboard::KeyboardStateTracker& keyboardStateTracker)
026  {
027      // ②「↓」キーを解放した場合「屈む」状態に遷移する
028      if (keyboardStateTracker.IsKeyPressed(DirectX::Keyboard::Keys::Down))
029      {
030          // 「屈む」状態状態に変更する
031          m_mario->ChangeState(m_mario->GetDowning());
032      }
033  }
034
035  // 「立ち」状態を描画する
036  void Standing::Render()
037  {
038      // 回転を設定する
039      float rotation = 0.0f;
040      // スケールを設定する
041      float scale = 1.0f;
042      // 原点を設定する
043      const DirectX::SimpleMath::Vector2 origin = DirectX::SimpleMath::Vector2(0.0f, 0.0f);
044      // スプライト効果を設定しない
```

```
045         DirectX::SpriteEffects effects = DirectX::SpriteEffects_None;
046         // 水平方向に反転させる
047         //effects = DirectX::SpriteEffects_FlipHorizontally;
048         // スプライトバッチを取得する
049         m_graphics->GetSpriteBatch()->Begin();
050         // スプライトを描画する
051         m_graphics->GetSpriteBatch()->Draw(
052                 m_standingTexture,
053                 m_mario->GetPosition(),
054                 nullptr,
055                 DirectX::Colors::White,
056                 rotation,
057                 origin,
058                 scale,
059                 effects
060         );
061         m_graphics->GetSpriteBatch()->End();
062 }
063
064 // 後処理をする
065 void Standing::Finalize()
066 {
067 }
```

Standing.cpp

7. 「屈む」状態を作る

マリオの「屈む」状態を実現する Downing クラスを作成します。

7.1 Downing クラスを定義する

「屈む」状態を実現する Downing クラスを定義します。IState インタフェースで宣言された関数を実装する関数を宣言します。メンバー変数としては Mario クラスのインスタンスへのポインタ、テクスチャ変数の参照用の変数を宣言します。

```
001  #pragma once
002  #ifndef DOWNING_DEFINED
003  #define DOWNING_DEFINED
004  #include "IState.h"
005  #include "PlayScene.h"
006  #include "Mario.h"
007
008  class Mario;
009
010  // Down クラスを定義する
011  class Downing : public IState
012  {
013  public:
014          // コンストラクタ
015          Downing(Mario* mario);
016           // デストラクタ
017          ~Downing();
018          // Downing オブジェクトを初期化する
019          void Initialize() override;
020          // 屈むを更新する
021          void Update(const DirectX::Keyboard::KeyboardStateTracker& keyboardStateTracker) override;
022          // 屈むを描画する
023          void Render() override;
024          // 後始末をする
025          void Finalize() override;
026
027  private:
028          // グラフィックス
029          Graphics* m_graphics = Graphics::GetInstance();
030          // マリオ
031          Mario* m_mario;
032          // 立ちテクスチャ
033          ID3D11ShaderResourceView* m_downingTexture;
034  };
035
036  #endif // DOWNING_DEFINED
```

Ducking.h

7.2 Downing クラスを実装する

Downing クラスの実装をします。Downing クラスでは Mario クラスのインスタンスへのポインタを取得し、マリオの情報を取得します。同様に、マリオの「屈む」テクスチャは Mario クラスで宣言しているため、Mario クラスのインスタンスへのポインタを通じて取得します。
Ducking クラスの Update 関数ではプレイヤの「屈む」状態の処理を実装します。状態遷移図の「②「↓」キーを解放した場合「立つ」状態に遷移する」を実装します。Render 関数ではマリオの「屈む」状態を描画します。

```
001  #include "pch.h"
002  #include "Downing.h"
003
004  class Mario;
005
006  // コンストラクタ
007  Downing::Downing(Mario* mario)
008  :
009  m_mario(mario),
010  m_downingTexture(m_mario->GetDowningTexture())
011  {
012  }
013
014  // デストラクタ
015  Downing::~Downing()
016  {
017  }
018
019  // 「屈む」を初期化する
020  void Downing::Initialize()
021  {
022  }
023
024  // 「屈む」を更新する
025  void Downing::Update(const DirectX::Keyboard::KeyboardStateTracker& keyboardStateTracker)
026  {
027          if (keyboardStateTracker.IsKeyReleased(DirectX::Keyboard::Keys::Down))
028          {
029                  // 「立ち」状態に状態遷移する
030                  m_mario->ChangeState(m_mario->GetStanding());
031          }
032  }
033
034  // 「屈む」を描画する
035  void Downing::Render()
036  {
037          // 回転を設定する
038          float rotation = 0.0f;
039          // スケールを設定する
040          float scale = 1.0f;
041          // 原点を設定する
042          const DirectX::SimpleMath::Vector2 origin = DirectX::SimpleMath::Vector2(0.0f, 0.0f);
043          // スプライト効果を設定しない
044          DirectX::SpriteEffects effects = DirectX::SpriteEffects_None;
```

```
045         // 水平方向に反転させる
046         //effects = DirectX::SpriteEffects_FlipHorizontally;
047 
048         m_graphics->GetSpriteBatch()->Begin();
049         // スプライトを描画する
050         m_graphics->GetSpriteBatch()->Draw(
051                 m_downingTexture,
052                 m_mario->GetPosition(),
053                 nullptr,
054                 DirectX::Colors::White,
055                 rotation,
056                 origin,
057                 scale,
058                 effects
059         );
060         m_graphics->GetSpriteBatch()->End();
061 }
062 
063 // 後処理をする
064 void Downing::Finalize()
065 {
066 }
```

Ducking.cpp

8. マリオクラスを作る

Mario クラスを作成します。Mario クラスではマリオの状態を管理し、状態遷移をしながらマリオの動きを処理します。

8.1 Mario クラスを定義する

Mario クラスを定義します。Mario クラスではキーボードの入力情報を取得したり、マリオの位置や移動のための情報を宣言したり、画像リソースの読み込みをおこなったりします。また、マリオの状態を生成し、現在の状態の更新や描画したりするのに必要なメンバー変数を宣言し、クラス外部から Mario クラス内部の変数を取得したり設定したりするためのアクセッサを宣言します。

```
001  #pragma once
002  #ifndef MARIO_DEFINED
003  #define MARIO_DEFINED
004
005  #include "SpriteBatch.h"
006  #include "WICTextureLoader.h"
007  #include "Graphics.h"
008  #include "IState.h"
009  #include "Standing.h"
010  #include "Downing.h"
011
012  class Standing;
013  class Downing;
014  class PlayScene;
015
016  class Mario
017  {
018  public:
019          // 立ち状態を取得する
020          Standing* GetStanding() { return m_standing.get(); }
021          // 屈む状態を取得する
022          Downing* GetDowning() { return m_downing.get(); }
023          // スタンディングテクスチャを取得する
024          ID3D11ShaderResourceView* GetStandingTexture() const { return m_standingTexture.Get(); }
025          // ダウンイングテクスチャを取得する
026          ID3D11ShaderResourceView* GetDowningTexture() const { return m_downingTexture.Get(); }
027
028  public:
029          // 位置を取得する
030          DirectX::SimpleMath::Vector2 GetPosition() const { return m_position; }
031          // 位置を設定する
032          void SetPosition(const DirectX::SimpleMath::Vector2& position) { m_position = position; }
033          // 速度を取得する
034          DirectX::SimpleMath::Vector2 GetVelocity() const { return m_velocity; }
035          // 速度を設定する
036          void SetVelocity(const DirectX::SimpleMath::Vector2& velocity) { m_velocity = velocity; }
037          // 加速度を取得する
038          DirectX::SimpleMath::Vector2 GetAcceleration() const { return m_acceleration; }
```

```
039         // 加速度を設定する
040         void SetAcceleration(const DirectX::SimpleMath::Vector2& acceleration) { m_acceleration = acceleration; }
041
042         // 立ち状態に遷移する
043         // void ChangeStandingState() { m_currentState = m_standing.get(); }
044         // 屈む状態に遷移する
045         // void ChangeDowningState() { m_currentState = m_downing.get(); }
046         // 状態を遷移する
047         void ChangeState(IState* currentState) { m_currentState = currentState; }
048
049 public:
050         // コンストラクタ
051         Mario(PlayScene* playScene);
052         // デストラクタ
053         ~Mario();
054         // 初期化する
055         void Initialize();
056         // 更新する
057         void Update(const DirectX::Keyboard::KeyboardStateTracker& keyboardStateTracker);
058         // 描画する
059         void Render();
060         // 後処理を行う
061         void Finalize();
062
063 private:
064         // プレイシーン
065         PlayScene* m_playScene;
066         // グラフィックス
067         Graphics* m_graphics = Graphics::GetInstance();
068         // キーボードステートステートトラッカー
069         DirectX::Keyboard::KeyboardStateTracker* m_keyboardStateTracker;
070         // スタンディング
071         std::unique_ptr<Standing> m_standing;
072         // ダウニイング
073         std::unique_ptr<Downing> m_downing;
074         // スタンディング
075         Microsoft::WRL::ComPtr<ID3D11ShaderResourceView> m_standingTexture;
076         // ダウニング
077         Microsoft::WRL::ComPtr<ID3D11ShaderResourceView> m_downingTexture;
078
079         // 位置
080         DirectX::SimpleMath::Vector2 m_position;
081         // 速度
082         DirectX::SimpleMath::Vector2 m_velocity;
083         // 加速度
084         DirectX::SimpleMath::Vector2    m_acceleration;
085         // 現在の状態
086         IState* m_currentState;
087 };
088
089 #endif              // MARIO_DEFINED
```

Marior.h

8.2 Mario クラスを実装する

Mario クラスでは次の手続きをおこなうための実装を行います。

プレイヤの手続き

・キーボードからキー入力の情報を取得する手続きを行う
・テクスチャなどの画像リソースを生成する
・Standing クラスや Ducking クラスのインスタンスの生成や初期化を行う
・プレイヤの初期状態として「立つ」状態を設定する
・プレイヤの初期位置、初期の加速度を設定する
・現在の状態オブジェクトの更新を行う
  m_currentState->Update(keyboardStateTracker)
・マリオの現在の状態の描画を行う
  m_currentState->Render(spriteBatch)

次に Mario クラスの実装ファイルを示します。

```
001  #include "pch.h"
002  #include "Mario.h"
003  #include "DeviceResources.h"
004
005  class Standing;
006  class Downing;
007
008  // コンストラクタ
009  Mario::Mario(PlayScene* playScene)
010  :
011  m_playScene(playScene),                                    // プレイシーン
012  m_keyboardStateTracker{},                                  // キーボードステートトラッカー
013  m_standing{},                                              // スタンディング
014  m_downing{},                                               // ダウニイング
015  m_position(DirectX::SimpleMath::Vector2(100.0f, 400.0f)),  // 位置
016  m_velocity{},                                              // 速度
017  m_acceleration{},                                          // 加速度
018  m_currentState{}                                           // 現在の状態
019  {
020  }
021
022  // デストラクタ
023  Mario::~Mario()
024  {
025  }
026
027  // 初期化する
028  void Mario::Initialize()
029  {
030      // デバイスを取得する
031      auto device = m_graphics->GetInstance()->GetDeviceResources()->GetD3DDevice();
032
```

```
033         // 「立ち」テクスチャ
034         DirectX::CreateWICTextureFromFile(device, L"Resources¥¥Image¥¥standing.png", nullptr,
035           m_standingTexture.GetAddressOf());
036         // 「屈む」テクスチャ
037         DirectX::CreateWICTextureFromFile(device, L"Resources¥¥Image¥¥downing.png", nullptr,
038         m_downingTexture.GetAddressOf());
039
040         // 「立ち」状態オブジェクトを生成する
041         m_standing = std::make_unique<Standing>(this);
042         // 「屈む」状態オブジェクトを生成する
043         m_downing = std::make_unique<Downing>(this);
044         // 初期状態を「立ち」に設定する
045         m_currentState = m_standing.get();
046 }
047
048 // 更新する
049 void Mario::Update(const DirectX::Keyboard::KeyboardStateTracker& keyboardStateTracker)
050 {
051         // 現在の状態を更新する
052         m_currentState->Update(keyboardStateTracker);
053 }
054
055 // 描画する
056 void Mario::Render()
057 {
058         // 現在の状態を描画する
059         m_currentState->Render();
060 }
061
062 // 後処理を行う
063 void Mario::Finalize()
064 {
065 }
```

Mario.cpp

練習問題

マリオのジャンプ状態を作成して下さい。