2024-04-22

バージョン: 2.0

デザインパターン

ステート

Design Pattern | State

Shozo Tanaka

内容

[図目次 2](#_Toc164629371)

[はじめに 3](#_Toc164629372)

[1. レガシーコード 4](#_Toc164629373)

[1.1 考察 7](#_Toc164629374)

[2. ステートパターン 9](#_Toc164629375)

[3. 状態遷移 10](#_Toc164629376)

[4. Stateパターンの作り方 11](#_Toc164629377)

[5. ステートインタフェースを定義する 11](#_Toc164629378)

[6. 「立つ」状態を作る 12](#_Toc164629379)

[6.1 Standingクラスを宣言する 12](#_Toc164629380)

[6.2 Standingクラスを実装する 13](#_Toc164629381)

[7. 「屈む」状態を作る 15](#_Toc164629382)

[7.1 Downingクラスを定義する 15](#_Toc164629383)

[7.2 Downingクラスを実装する 16](#_Toc164629384)

[8. マリオクラスを作る 18](#_Toc164629385)

[8.1 Marioクラスを定義する 18](#_Toc164629386)

[8.2 Marioクラスを実装する 20](#_Toc164629387)

[練習問題 21](#_Toc164629388)

[9. 「ジャンプ」状態を作る 22](#_Toc164629389)

[9.1 Jumpingクラスを定義する 22](#_Toc164629390)

[9.2 Jumpingクラスを実装する 23](#_Toc164629391)

[9.2.1 ジャンプ中の移動処理 24](#_Toc164629392)

[9.2.2 空中に存在している間の処理 24](#_Toc164629393)

[9.2.3 地面に着地した場合の処理 25](#_Toc164629394)

[9.2.4 「←」「→」キーを押下げる場合「走る」状態に遷移する 26](#_Toc164629395)

[9.2.5 「←」「→」キーを解放した場合「立ち」状態に遷移する 26](#_Toc164629396)

[9.3 マリオのジャンプを描画する 27](#_Toc164629397)

[練習問題 ランニングを作成する 27](#_Toc164629398)

# 図目次

[図1 Marioの動作 4](#_Toc164610653)

[図2 ステートのクラス 9](#_Toc164610654)

[図3 状態遷移図 10](#_Toc164610655)

# 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。

# 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。

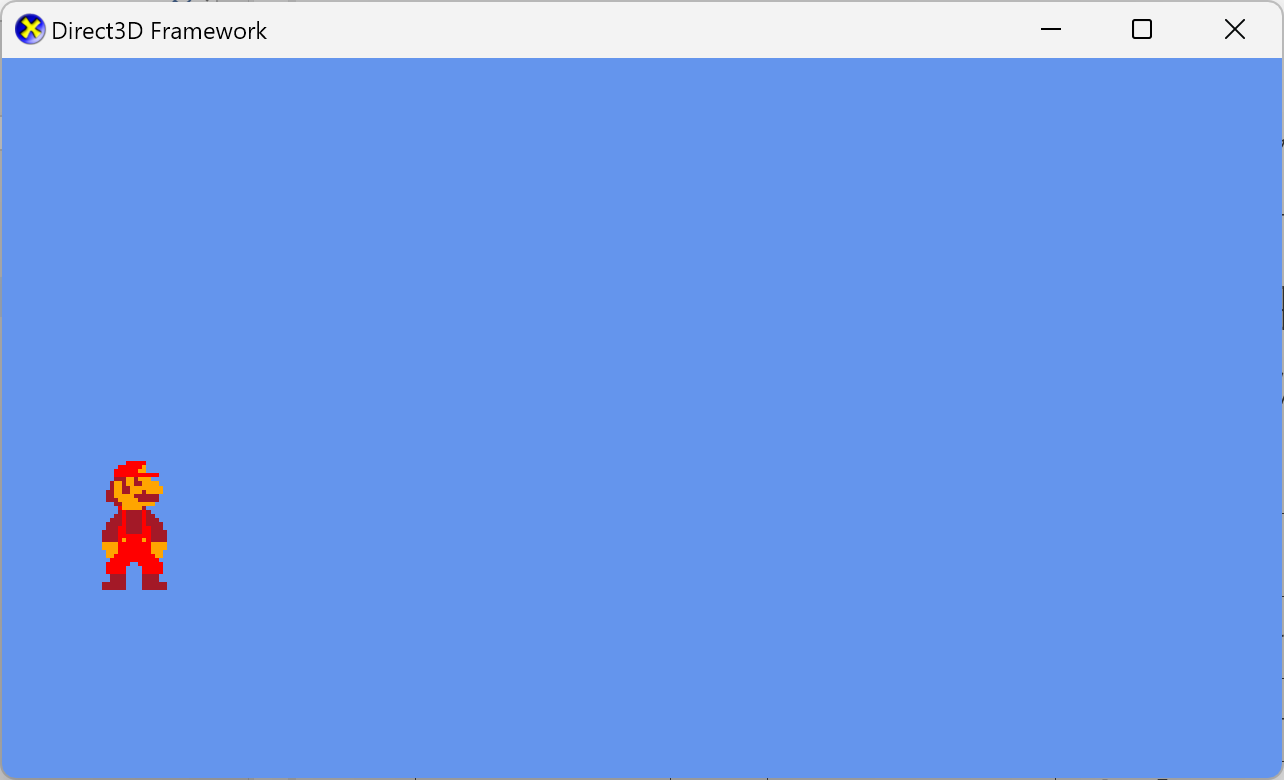


図1 Marioの動作

Marioクラスのヘッダーファイルを示します。

1. #pragma once
2. #ifndef MARIO\_DEFINED
3. #define MARIO\_DEFINED
4. #include "SpriteBatch.h"
5. #include "WICTextureLoader.h"
6. #include "Graphics.h"
7. class Mario
8. {
9. // プレイヤーの状態
10. enum State
11. {
12. STANDING\_STATE,
13. JUMPING\_STATE,
14. DUCKING\_STATE,
15. };
16. public:
17. // コンストラクタ
18. Mario(float x, float y);
19. // デストラクタ
20. ~Mario();
21. // 初期化する
22. void Initialize();
23. // 更新する
24. void Update(const DirectX::Keyboard::KeyboardStateTracker& keyboardStateTracker);
25. // 描画する
26. void Render();
27. private:
28. // グラフィックス
29. Graphics\* m\_graphics = Graphics::GetInstance();
30. // スタンディング
31. Microsoft::WRL::ComPtr<ID3D11ShaderResourceView> m\_standing;
32. // ジャンプ
33. Microsoft::WRL::ComPtr<ID3D11ShaderResourceView> m\_jumping;
34. // ダウニング
35. Microsoft::WRL::ComPtr<ID3D11ShaderResourceView> m\_downing;
36. // 現在のテクスチャ
37. Microsoft::WRL::ComPtr<ID3D11ShaderResourceView> m\_currentTexture;
38. // 表示位置
39. float m\_x, m\_y;
40. // 速度
41. float m\_vx, m\_vy;
42. // 加速度
43. float m\_ax, m\_ay;
44. // プレイヤーの状態
45. State m\_currentState;
46. };
47. #endif // MARIO\_DEFINED

Marioクラスの実装ファイルは次のようになります。

1. #include "pch.h"
2. #include "Mario.h"
3. #include "DeviceResources.h"
4. // コンストラクタ
5. Mario::Mario(float x, float y)
6. :
7. m\_currentState(State::STANDING\_STATE),
8. m\_currentTexture(nullptr),
9. m\_standing(nullptr),
10. m\_jumping(nullptr),
11. m\_downing(nullptr),
12. m\_x(x),
13. m\_y(y),
14. m\_vx(0.0f),
15. m\_vy(0.0f),
16. m\_ax(0.0f),
17. m\_ay(0.2f)
18. {
19. }
20. // デストラクタ
21. Mario::~Mario()
22. {
23. }
24. // 初期化する
25. void Mario::Initialize()
26. {
27. // デバイスを取得する
28. auto\* device = m\_graphics->GetInstance()->GetDeviceResources()->GetD3DDevice();
29. // テクスチャをロードする
30. DirectX::CreateWICTextureFromFile(device, L"Resources\\Image\\standing.png", nullptr,
31. m\_standing.GetAddressOf());
32. DirectX::CreateWICTextureFromFile(device, L"Resources\\Image\\jumping.png", nullptr,
33. m\_jumping.GetAddressOf());
34. DirectX::CreateWICTextureFromFile(device, L"Resources\\Image\\downing.png", nullptr,
35. m\_downing.GetAddressOf());
36. // プレイヤの初期状態を「立ち」に設定する
37. m\_currentState = State::STANDING\_STATE;
38. }
39. // 更新する
40. void Mario::Update(const DirectX::Keyboard::KeyboardStateTracker& keyboardStateTracker)
41. {
42. // プレイヤの状態に合わせた行動を行う
43. switch (m\_currentState)
44. {
45. case State::STANDING\_STATE: // 立っている場合
46. if (keyboardStateTracker.IsKeyPressed(DirectX::Keyboard::Keys::Space))
47. {
48. // 速度を設定する
49. m\_vy = -10.0f;
50. // プレイヤの状態を「ジャンプ」に設定する
51. m\_currentState = State::JUMPING\_STATE;
52. }
53. else if (keyboardStateTracker.IsKeyPressed(DirectX::Keyboard::Keys::Down))
54. {
55. // プレイヤの状態を「屈む」に設定する
56. m\_currentState = DUCKING\_STATE;
57. }
58. // テクスチャを「立ち」に設定する
59. m\_currentTexture = m\_standing;
60. break;
61. case State::JUMPING\_STATE: // ジャンプする状態
62. if (m\_y == 400.0f)
63. {
64. // プレイヤの状態を「立ち」に設定する
65. m\_currentState = State::STANDING\_STATE;
66. }
67. // テクスチャを「ジャンプ」に設定する
68. m\_currentTexture = m\_jumping;
69. break;
70. case State::DUCKING\_STATE: // 屈む場合
71. if (keyboardStateTracker.IsKeyReleased(DirectX::Keyboard::Keys::Down))
72. {
73. // プレイヤの状態を「立ち」に設定する
74. m\_currentState = State::STANDING\_STATE;
75. }
76. // テクスチャを「屈む」に設定する
77. m\_currentTexture = m\_downing;
78. break;
79. }
80. // 速度に加速度を加算する
81. m\_vx += m\_ax;
82. m\_vy += m\_ay;
83. // 位置に速度を加算する
84. m\_x += m\_vx;
85. m\_y += m\_vy;
86. // 地面を判定する
87. if (m\_y > 400.0f)
88. {
89. m\_y = 400.0f;
90. m\_vy = 0.0f;
91. }
92. }
93. // 描画する
94. void Mario::Render()
95. {
96. // スプライトバッチを開始する
97. m\_graphics->GetSpriteBatch()->Begin();
98. // スプライトを描画する
99. m\_graphics->GetSpriteBatch()->Draw(m\_currentTexture.Get(), DirectX::SimpleMath::Vector2(m\_x, m\_y));
100. // スプライトバッチを終了する
101. m\_graphics->GetSpriteBatch()->End();
102. }

## 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。

IState

+handle

ConcreateState

+handle

ConcreateState

+handle

Context

-state : State

+Request

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。

# 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 | 1. [↓]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。

⑧ プレイヤが地面に着地し「←」「→」キーを押し下げている

⑦ [SPACE]キーを押し下げる

④ [←] [→]キーを解放

した

② [↓]キーを解放した

① [↓]キーを押し下げる

⑤ [SPACE]キーを押し下げる

③ [←] [→]キーを押し下げる

立つ(Standing)

走る(Running)

屈む(Downing)

ジャンプ(Jumping)

⑥ プレイヤが地面に着地し「←」「→」キーを解放

している

図3 状態遷移図

# 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。

# 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。

1. #pragma once
2. #ifndef STATE\_DEFINED
3. #define STATE\_DEFINED
4. #include "StepTimer.h"
5. // IStateインターフェイスを定義する
6. class IState
7. {
8. public:
9. // 初期化する
10. virtual void Initialize() = 0;
11. // 処理を実行する
12. virtual void Update(const DirectX::Keyboard::KeyboardStateTracker& keyboardStateTracker) = 0;
13. // テクスチャを描画する
14. virtual void Render() = 0;
15. // 後始末をする
16. virtual void Finalize() = 0;
17. // 仮想デストラクタ
18. virtual ~IState() = default;
19. };
20. #endif // STATE\_DEFINED

IStateインタフェース(IState.h)

# 「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。

1. #pragma once
2. #ifndef STANDING\_DEFINED
3. #define STANDING\_DEFINED
4. #include "IState.h"
5. #include "PlayScene.h"
6. #include "Mario.h"
7. class Mario;
8. // Standingクラスを宣言する
9. class Standing : public IState
10. {
11. public:
12. // コンストラクタ
13. Standing(Mario\* mario);
14. // デストラクタ
15. ~Standing();
16. // Standingオブジェクトを初期化する
17. void Initialize() override;
18. // 立ちを実行する
19. void Update(const DirectX::Keyboard::KeyboardStateTracker& keyboardStateTracker) override;
20. // 立ちを描画する
21. void Render() override;
22. // 後始末をする
23. void Finalize() override;
24. private:
25. // グラフィックス
26. Graphics\* m\_graphics = Graphics::GetInstance();
27. // マリオ
28. Mario\* m\_mario;
29. // 立ちテクスチャ
30. ID3D11ShaderResourceView\* m\_standingTexture;
31. };
32. #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。

1. #include "pch.h"
2. #include "Standing.h"
3. class PlayScene;
4. // コンストラクタ
5. Standing::Standing(Mario\* mario)
6. :
7. m\_mario(mario),
8. m\_standingTexture(m\_mario->GetStandingTexture())
9. {
10. }
11. // デストラクタ
12. Standing::~Standing()
13. {
14. }
15. // 「立ち」状態を初期化する
16. void Standing::Initialize()
17. {
18. }
19. // 「立ち」状態を更新する
20. void Standing::Update(const DirectX::Keyboard::KeyboardStateTracker& keyboardStateTracker)
21. {
22. // ②「↓」キーを解放した場合「屈む」状態に遷移する
23. if (keyboardStateTracker.IsKeyPressed(DirectX::Keyboard::Keys::Down))
24. {
25. // 「屈む」状態状態に変更する
26. m\_mario->ChangeState(m\_mario->GetDowning());
27. }
28. }
29. // 「立ち」状態を描画する
30. void Standing::Render()
31. {
32. // 回転を設定する
33. float rotation = 0.0f;
34. // スケールを設定する
35. float scale = 1.0f;
36. // 原点を設定する
37. const DirectX::SimpleMath::Vector2 origin = DirectX::SimpleMath::Vector2(0.0f, 0.0f);
38. // スプライト効果を設定しない
39. DirectX::SpriteEffects effects = DirectX::SpriteEffects\_None;
40. // 水平方向に反転させる
41. //effects = DirectX::SpriteEffects\_FlipHorizontally;
42. // スプライトバッチを取得する
43. m\_graphics->GetSpriteBatch()->Begin();
44. // スプライトを描画する
45. m\_graphics->GetSpriteBatch()->Draw(
46. m\_standingTexture,
47. m\_mario->GetPosition(),
48. nullptr,
49. DirectX::Colors::White,
50. rotation,
51. origin,
52. scale,
53. effects
54. );
55. m\_graphics->GetSpriteBatch()->End();
56. }
57. // 後処理をする
58. void Standing::Finalize()
59. {
60. }

Standing.cpp

# 「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。

1. #pragma once
2. #ifndef DOWNING\_DEFINED
3. #define DOWNING\_DEFINED
4. #include "IState.h"
5. #include "PlayScene.h"
6. #include "Mario.h"
7. class Mario;
8. // Downクラスを宣言する
9. class Downing : public IState
10. {
11. public:
12. // コンストラクタ
13. Downing(Mario\* mario);
14. // デストラクタ
15. ~Downing();
16. // Downingオブジェクトを初期化する
17. void Initialize() override;
18. // 屈むを更新する
19. void Update(const DirectX::Keyboard::KeyboardStateTracker& keyboardStateTracker) override;
20. // 屈むを描画する
21. void Render() override;
22. // 後始末をする
23. void Finalize() override;
24. private:
25. // グラフィックス
26. Graphics\* m\_graphics = Graphics::GetInstance();
27. // マリオ
28. Mario\* m\_mario;
29. // 立ちテクスチャ
30. ID3D11ShaderResourceView\* m\_downingTexture;
31. };
32. #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。

1. #include "pch.h"
2. #include "Downing.h"
3. class Mario;
4. // コンストラクタ
5. Downing::Downing(Mario\* mario)
6. :
7. m\_mario(mario),
8. m\_downingTexture(m\_mario->GetDowningTexture())
9. {
10. }
11. // デストラクタ
12. Downing::~Downing()
13. {
14. }
15. // 「屈む」を初期化する
16. void Downing::Initialize()
17. {
18. }
19. // 「屈む」を更新する
20. void Downing::Update(const DirectX::Keyboard::KeyboardStateTracker& keyboardStateTracker)
21. {
22. if (keyboardStateTracker.IsKeyReleased(DirectX::Keyboard::Keys::Down))
23. {
24. // 「立ち」状態に状態遷移する
25. m\_mario->ChangeState(m\_mario->GetStanding());
26. }
27. }
28. // 「屈む」を描画する
29. void Downing::Render()
30. {
31. // 回転を設定する
32. float rotation = 0.0f;
33. // スケールを設定する
34. float scale = 1.0f;
35. // 原点を設定する
36. const DirectX::SimpleMath::Vector2 origin = DirectX::SimpleMath::Vector2(0.0f, 0.0f);
37. // スプライト効果を設定しない
38. DirectX::SpriteEffects effects = DirectX::SpriteEffects\_None;
39. // 水平方向に反転させる
40. //effects = DirectX::SpriteEffects\_FlipHorizontally;
41. m\_graphics->GetSpriteBatch()->Begin();
42. // スプライトを描画する
43. m\_graphics->GetSpriteBatch()->Draw(
44. m\_downingTexture,
45. m\_mario->GetPosition(),
46. nullptr,
47. DirectX::Colors::White,
48. rotation,
49. origin,
50. scale,
51. effects
52. );
53. m\_graphics->GetSpriteBatch()->End();
54. }
55. // 後処理をする
56. void Downing::Finalize()
57. {
58. }

Ducking.cpp

# 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。

1. #pragma once
2. #ifndef MARIO\_DEFINED
3. #define MARIO\_DEFINED
4. #include "SpriteBatch.h"
5. #include "WICTextureLoader.h"
6. #include "Graphics.h"
7. #include "IState.h"
8. #include "Standing.h"
9. #include "Downing.h"
10. class Standing;
11. class Downing;
12. class PlayScene;
13. class Mario
14. {
15. public:
16. // 立ち状態を取得する
17. Standing\* GetStanding() { return m\_standing.get(); }
18. // 屈む状態を取得する
19. Downing\* GetDowning() { return m\_downing.get(); }
20. // スタンディングテクスチャを取得する
21. ID3D11ShaderResourceView\* GetStandingTexture() const { return m\_standingTexture.Get(); }
22. // ダウンイングテクスチャを取得する
23. ID3D11ShaderResourceView\* GetDowningTexture() const { return m\_downingTexture.Get(); }
24. public:
25. // 位置を取得する
26. DirectX::SimpleMath::Vector2 GetPosition() const { return m\_position; }
27. // 位置を設定する
28. void SetPosition(const DirectX::SimpleMath::Vector2& position) { m\_position = position; }
29. // 速度を取得する
30. DirectX::SimpleMath::Vector2 GetVelocity() const { return m\_velocity; }
31. // 速度を設定する
32. void SetVelocity(const DirectX::SimpleMath::Vector2& velocity) { m\_velocity = velocity; }
33. // 加速度を取得する
34. DirectX::SimpleMath::Vector2 GetAcceleration() const { return m\_acceleration; }
35. // 加速度を設定する
36. void SetAcceleration(const DirectX::SimpleMath::Vector2& acceleration) { m\_acceleration = acceleration; }
37. // 立ち状態に遷移する
38. // void ChangeStandingState() { m\_currentState = m\_standing.get(); }
39. // 屈む状態に遷移する
40. // void ChangeDowningState() { m\_currentState = m\_downing.get(); }
41. // 状態を遷移する
42. void ChangeState(IState\* currentState) { m\_currentState = currentState; }
43. public:
44. // コンストラクタ
45. Mario(PlayScene\* playScene);
46. // デストラクタ
47. ~Mario();
48. // 初期化する
49. void Initialize();
50. // 更新する
51. void Update(const DirectX::Keyboard::KeyboardStateTracker& keyboardStateTracker);
52. // 描画する
53. void Render();
54. // 後処理を行う
55. void Finalize();
56. private:
57. // プレイシーン
58. PlayScene\* m\_playScene;
59. // グラフィックス
60. Graphics\* m\_graphics = Graphics::GetInstance();
61. // キーボードステートステートトラッカー
62. DirectX::Keyboard::KeyboardStateTracker\* m\_keyboardStateTracker;
63. // スタンディング
64. std::unique\_ptr<Standing> m\_standing;
65. // ダウニイング
66. std::unique\_ptr<Downing> m\_downing;
67. // スタンディング
68. Microsoft::WRL::ComPtr<ID3D11ShaderResourceView> m\_standingTexture;
69. // ダウニング
70. Microsoft::WRL::ComPtr<ID3D11ShaderResourceView> m\_downingTexture;
71. // 位置
72. DirectX::SimpleMath::Vector2 m\_position;
73. // 速度
74. DirectX::SimpleMath::Vector2 m\_velocity;
75. // 加速度
76. DirectX::SimpleMath::Vector2 m\_acceleration;
77. // 現在の状態
78. IState\* m\_currentState;
79. };
80. #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。

1. #include "pch.h"
2. #include "Mario.h"
3. #include "DeviceResources.h"
4. class Standing;
5. class Downing;
6. // コンストラクタ
7. Mario::Mario(PlayScene\* playScene)
8. :
9. m\_playScene(playScene), // プレイシーン
10. m\_keyboardStateTracker{}, // キーボードステートトラッカー
11. m\_standing{}, // スタンディング
12. m\_downing{}, // ダウニイング
13. m\_position(DirectX::SimpleMath::Vector2(100.0f, 400.0f)), // 位置
14. m\_velocity{}, // 速度
15. m\_acceleration{}, // 加速度
16. m\_currentState{} // 現在の状態
17. {
18. }
19. // デストラクタ
20. Mario::~Mario()
21. {
22. }
23. // 初期化する
24. void Mario::Initialize()
25. {
26. // デバイスを取得する
27. auto device = m\_graphics->GetInstance()->GetDeviceResources()->GetD3DDevice();
29. // 「立ち」テクスチャ
30. DirectX::CreateWICTextureFromFile(device, L"Resources\\Image\\standing.png", nullptr,
31. m\_standingTexture.GetAddressOf());
32. // 「屈む」テクスチャ
33. DirectX::CreateWICTextureFromFile(device, L"Resources\\Image\\downing.png", nullptr,
34. m\_downingTexture.GetAddressOf());
35. // 「立ち」状態オブジェクトを生成する
36. m\_standing = std::make\_unique<Standing>(this);
37. // 「屈む」状態オブジェクトを生成する
38. m\_downing = std::make\_unique<Downing>(this);
39. // 初期状態を「立ち」に設定する
40. m\_currentState = m\_standing.get();
41. }
42. // 更新する
43. void Mario::Update(const DirectX::Keyboard::KeyboardStateTracker& keyboardStateTracker)
44. {
45. // 現在の状態を更新する
46. m\_currentState->Update(keyboardStateTracker);
47. }
48. // 描画する
49. void Mario::Render()
50. {
51. // 現在の状態を描画する
52. m\_currentState->Render();
53. }
54. // 後処理を行う
55. void Mario::Finalize()
56. {
57. }

Mario.cpp

## 練習問題

マリオのジャンプ状態を作成して下さい。

# 「ジャンプ」状態を作る

IStateインタフェースを実装するJumpクラスを作成します。

## 9.1 Jumpingクラスを定義する

「ジャンプ」状態クラスである「Jumping」クラスを定義します。IState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「グラフィックスへのポインタ」「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。

1. #pragma once
2. #ifndef JUMPING\_DEFINED
3. #define JUMPING\_DEFINED
4. #include "IState.h"
5. #include "PlayScene.h"
6. #include "Mario.h"
7. class Mario;
8. // Jumpingクラスを定義する
9. class Jumping : public IState
10. {
11. public:
12. // コンストラクタ
13. Jumping(Mario\* mario);
14. // デストラクタ
15. ~Jumping() override;
16. // 初期化する
17. void Initialize() override;
18. // 更新する
19. void Update(const DX::StepTimer& timer, const DirectX::Keyboard::KeyboardStateTracker& keyboardStateTracker) override;
20. // 描画する
21. void Render() override;
22. // 後処理をする
23. void Finalize() override;
24. private:
25. // グラフィックス
26. Graphics\* m\_graphics = Graphics::GetInstance();
27. // マリオ
28. Mario\* m\_mario;
29. // 立ちテクスチャ
30. ID3D11ShaderResourceView\* m\_jumpingTexture;
31. };
32. #endif // JUMPING\_DEFINED

Jumping.h

## 9.2 Jumpingクラスを実装する

マリオがジャンプするJumpingクラスの実装をします。Jumping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、マリオのテクスチャはMarioクラスでロードしているため、Marioクラスのインスタンスへのポインタ(m\_jumpingTexture(m\_mario->GetJumpingTexture()))を通じて「ジャンプ」テクスチャを取得します。

Jumpking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。状態遷移図を参照すると「ジャンプ」状態から「ジャンプ」「立つ」状態に遷移します。次に「ジャンプ」状態の更新処理で実装する必要があるものを示します。

「ジャンプ」状態の更新で実装する必要がある処理

・ジャンプ中の移動処理

・空中に存在している間の処理

・地面に着地した場合の処理

・⑧「←」「→」キーを押し下げている場合の「走る」状態に遷移する処理

・⑥「←」「→」キーを解放した場合の「立ち」状態に処理に遷移する処理

・「ジャンプ」状態から「立ち」状態に遷移する処理

⑧ プレイヤが地面に着地し「←」「→」キーを押し下げている

⑦ [SPACE]キーを押し下げる

④ [←] [→]キーを解放

した

② [↓]キーを解放した

① [↓]キーを押し下げる

⑤ [SPACE]キーを押し下げる

③ [←] [→]キーを押し下げる

立つ(Standing)

走る(Running)

屈む(Downing)

ジャンプ(Jumping)

⑥ プレイヤが地面に着地し「←」「→」キーを解放

している

図5 「ジャンプ」の状態遷移図

「ジャンプ」状態の更新処理には、不完全なコードが存在しますので、プログラムを完成させて下さい。

// 更新する

void Jumping::Update(const DX::StepTimer& timer, const DirectX::Keyboard::KeyboardStateTracker& keyboardStateTracker)

{

…

}

### 9.2.1 ジャンプ中の移動処理

マリオのジャンプ中の移動処理を実装します。手順としては「速度に加速度を加算する」、次に「位置に速度を加算する」処理を追加します。

1. // 更新する
2. void Jumping::Update(const DX::StepTimer& timer, const DirectX::Keyboard::KeyboardStateTracker& keyboardStateTracker)
3. {
4. // 速度に加速度を加算する
5. ;
6. // 位置に速度を加算する
7. ;
8. …
9. }

Update関数(Jumping.h)

### 9.2.2 空中に存在している間の処理

マリオが空中に存在しているかどうかはマリオの位置のＹ軸の位置とプレイシーンに定義されている地面(PlayScene::GROUND)を比較します。コードは次のようになります。

// 空中にいる場合

if (m\_mario->GetPosition().y > PlayScene::GROUND)

{

…

}

マリオが空中に存在している場合の処理を記述します。まず、マリオの位置を取得した後、マリオのＹ座標の位置に地面の高さを再設定します。次に速度を取得した後にＹ軸方向の速度に0.0fを設定します。

1. // 空中にいる場合
2. if (m\_mario->GetPosition().y > PlayScene::GROUND)
3. {
4. // 位置を取得する
5. DirectX::SimpleMath::Vector2 position = m\_mario->GetPosition();
6. // 地面の高さを設定する
7. ;
8. // 位置を設定する
9. ;
10. // 速度を設定する
11. DirectX::SimpleMath::Vector2 velocity = m\_mario->GetVelocity();
12. // 速度(VY)を設定する
13. ;
14. // 速度を設定する
15. ;
16. }

Update関数(Jumping.h)

### 9.2.3 地面に着地した場合の処理

地面に着地した場合、「←」「→」キーが押し下げているかどうかを調べてます。「←」「→」キーが押し下げられている場合にはマリオの向きに対して矢印の方向を設定し、「走る」状態にします。

// 地面に着地した場合

if (floor(m\_mario->GetPosition().y) == PlayScene::GROUND)

{

…

}

### 9.2.4 「←」「→」キーを押下げる場合「走る」状態に遷移する

予め直前のキーボードステートを取得します。状態遷移図の⑧「→」キーを押下げている場合、マリオを右方向に設定した後、「走る」状態に設定します。

1. // 直前のキーボードステートを取得する
2. DirectX::Keyboard::State keyboardState;
3. // ⑧「→」キーを押し下げている場合右方向に走る
4. if(keyboardState.IsKeyDown(DirectX::Keyboard::Right))
5. {
6. // 右向きを設定する
7. m\_mario->SetDirection(Direction::RIGHT);
8. // 現在の状態を「走る」状態に遷移する
9. ;
10. }

Update関数(Jumping.h)

状態遷移図の⑧「←」キーを押し下げている場合、マリオを左方向に設定した後、「走る」状態に設定します。

1. // ⑧「←」キーを押し下げている場合左方向に走る
2. else if(keyboardState.IsKeyDown(DirectX::Keyboard::Left))
3. {
4. // 左向きを設定する
5. ;
6. // 現在の状態を「走る」状態に遷移する
7. ;
8. }

Update関数(Jumping.h)

### 9.2.5 「←」「→」キーを解放した場合「立ち」状態に遷移する

「←」「→」キーを押し下げていない、つまり「←」「→」キーを解放した場合「ジャンプ」状態から「立ち」状態に遷移します。

1. else
2. {
3. // ⑥「←」「→」キーを解放した場合に現在の状態を「立ち」状態に遷移する
4. ;
5. }

Update関数(Jumping.h)

## 9.3 マリオのジャンプを描画する

Render関数では「ジャンプ」状態の描画を行います。マリオの向きを取得して水平方向に反転させるかどうかを調べ、マリオが左を向いている場合はテクスチャを反転させます。プログラム中に不完全なプログラムが3カ所存在します。

1. // 描画する
2. void Jumping::Render()
3. {
4. // 回転を設定する
5. float rotation = 0.0f;
6. // スケールを設定する
7. float scale = 1.0f;
8. // 原点を設定する
9. const DirectX::SimpleMath::Vector2 origin = DirectX::SimpleMath::Vector2(0.0f, 0.0f);
10. // スプライト効果を設定しない
11. DirectX::SpriteEffects effects = DirectX::SpriteEffects\_None;
12. // 水平方向に反転させる
13. if (m\_mario->GetDirection() == Direction::LEFT)
14. {
15. // 水平方向に反転させる
16. effects ;
17. }
18. // スプライトバッチを開始する
19. ;
20. // スプライトを描画する
21. m\_graphics->GetSpriteBatch()->Draw(
22. m\_jumpingTexture,
23. m\_mario->GetPosition(),
24. nullptr,
25. DirectX::Colors::White,
26. rotation,
27. origin,
28. scale,
29. effects
30. );
31. // スプライトバッチを終了する
32. ;
33. }

Render関数(Jumping.h)

## 練習問題 ランニングを作成する

マリオのランニング状態を作成して下さい。