紙芝居DSLファイル作成マニュアル

目次

1. 紙芝居DSLとは 3
2. ファイルの基本構造 3
3. 最小サンプル 4
4. ヘッダ部を書く 5
    4.1 バージョン行 5
    4.2 アセット定義 5
    4.3 アクター定義 7
    4.4 表紙定義 8
    4.5 ランタイム変数 8
    4.6 条件分岐の登録 8
    4.7 ポーズ案内の文言 9
5. シーンを書く 9
    5.1 シーンの基本方針 10
    5.2 コメント行 10
    5.3 シーンラベル 11
    5.4 TMPoseURL 11
6. アクションを書く 11
    6.1 背景を変える 11
    6.2 待つ 12
    6.3 音を鳴らす 12
    6.4 画面をフェードさせる 12
    6.5 アクターを表示する 13
    6.6 アクターを移動する 13
    6.7 セリフと思考吹き出し 13
    6.8 スキンを変える 14
    6.9 画像や音を連続再生する 14
    6.10 テキストアセットを表示・更新する 14
    6.11 シーンを分岐させる 15
    6.12 ポーズ認識を入れる 15
7. `urashima.txt` の構成例 16

8. 紙芝居DSLの作成手順 17
8.1 物語をシーンに分ける 17
8.2 必要なアセットを洗い出す 18
8.3 アセット名を決める 18
8.4 ヘッダを書く 18
8.5 各シーンを書く 19
8.6 ポーズ認識を追加する 19
9. 書き方の注意 20
10. テスト方法 20
10.1 まず文法を確認する 20
10.2 次にアセットを確認する 20
10.3 最後に演出を確認する 21
11. 改善しやすい台本にするコツ 21
11.1 シーン番号をコメントで書く 21
11.2 演出のまとまりごとに空行を入れる 21
11.3 ポーズの意味が分かる名前を使う 21
11.4 長い物語は小さく作る 22
12. 作成チェックリスト 22
13. 関連ドキュメント 22

紙芝居DSLファイル作成マニュアル

対象: 台本作者、教材作成者、授業設計者、開発者
対象DSL: `kamishibai=3.1`

1. 紙芝居DSLとは

紙芝居DSLは、紙芝居アプリに読み込ませるためのテキスト形式の台本です。背景、登場人物、画面上のテキスト、セリフ、音、移動、アニメーション、ポーズ認識、シーン分岐などを、1行ずつ書きます。

DSLは「Domain Specific Language」の略です。ここでは「紙芝居を作るための専用の書き方」という意味で使います。

このDSLの大きな特徴は、紙芝居の演出だけでなく、TMPoseモデルを使ったポーズ認識を物語進行に組み込めることです。

2. ファイルの基本構造

紙芝居DSLファイルは、次のような構造です。

```
kamishibai=3.1
setRuntimeVariable=startSceneIndex:1
setLoadingBackdrop=読み込み背景
setLoadingCostume=読み込み画像1,読み込み画像2
setPoseRecognitionSound=ポーズ認識中の効果音,認識成立時の効果音
asset=背景名,backdrop
asset=アセット名,costume:スプライト名
asset=音名,sound
asset=テキスト名,text
actor=アクター名,初期スキン
cover=表紙背景アセット名,表紙音声アセット名
---
# scene 1
sceneLabel=シーン名
TMPoseURL=ポーズモデルURL
action=演出
---
# scene 2
sceneLabel=別のシーン名
action=演出
```

最初の `---` より前をヘッダ部、以降をシーン部と呼びます。

部分	役割
バージョン行	`kamishibai=3.1` を書きます。必須です。
ヘッダ部	画像、音声、テキスト、登場人物、表紙、初期変数、分岐を定義します。
シーン部	シーンラベル、背景、テキスト、移動、音、ポーズ認識、分岐などを書きます。
`---`	シーンの区切りです。
`#`	コメント行です。台本の説明やメモに使えます。

3. 最小サンプル

まずは、背景を出し、登場人物にセリフを言わせるだけの最小例です。

```
kamishibai=3.1
asset=Beach,backdrop
asset=Hero,costume
asset=OpeningSound,sound
actor=Hero,Hero
cover=Beach,OpeningSound
---
# scene 1
sceneLabel=opening
action=stage:Beach
action=Hero:show:Hero:0,-60,30
action=Hero:say:こんにちは！:2
action=wait:1
```

この台本では、次の順に動きます。

1. 背景 `Beach` を表示する
2. アクター `Hero` を中央やや下に表示する
3. `こんにちは！` と2秒間言う
4. 1秒待つ

4. ヘッダ部を書く

4.1 バージョン行

```
kamishibai=3.1
```

台本の先頭に書きます。このアプリは `kamishibai=3.1` の台本として読み込みます。2.0の台本を流用する場合も、追加仕様とアセット識別子を確認してから3.1へ更新してください。

4.2 アセット定義

```
asset=アセット名,URL
```

画像、音声、テキストを登録します。外部URLのほか、`.sb3` 内のコスチューム、背景、音も利用できます。

例:

```
asset=Beach1,https://example.com/stages/Beach1.png
asset=Urashima-walk-1,https://example.com/skins/Urashima-walk-1.png
asset=OceanWave,https://example.com/sounds/OceanWave.mp3
```

プロジェクト内アセットの代表的な書式:

```
asset=Hero,costume
asset=Hero-happy,costume:Hero
asset=Beach,backdrop
asset=OpeningSound,sound
asset=Narration,text
```

- `costume` は、アセット名と同名のスプライト／コスチュームを使います。
- `costume:スプライト名` は、そのスプライト内でアセット名と同名のコスチュームを使います。
- `backdrop` と `sound` は、アセット名と同名のステージ背景／ステージ音を使います。
- `text` は、画面に表示できるテキストアセットを作ります。

名前を明示したい場合は、`costume:スプライト名:コスチューム名`、`backdrop:背景名`、`sound:スプライト名:音名`、`text:テキスト名` も使えます。ステージ音のスプライト名は `@stage` です。

アセット名は台本内で何度も参照します。分かりやすく、重複しない名前にしてください。

Loading表示を変更する

```
asset=loading1,https://example.com/loading/loading1.png
asset=loading2,https://example.com/loading/loading2.png
asset=loading3,https://example.com/loading/loading3.png
asset=loadingBackground,https://example.com/loading/background.png
setLoadingBackdrop=loadingBackground
setLoadingCostume=loading1,loading2,loading3
```

`setLoadingBackdrop` にはLoading中に表示する背景アセット名を１件指定します。指定した背景は最初に読み込まれ、読み込み完了直後にステージへ表示されます。指定を省略した場合は、タイトル画像を残さず、組み込みの真っ黒な背景を表示します。

`setLoadingCostume` には、`asset` で定義した画像アセット名をカンマ区切りで指定します。指定した画像はLoading背景に続けて他のアセットより先に読み込まれ、その後の通常アセット読込中に、特別な組み込みスプライト `Loading` へ順番に表示されます。通常アセットの１件目では `loading1`、２件目では `loading2`、４件目では再び `loading1` というように循環します。

Loading用の背景と画像は、読込進捗の完了数と総数から除外されます。たとえばLoading背景が1件、Loading用画像が3件、通常アセットが10件なら、吹き出しは`0 / 10`から`10 / 10`まで進みます。Loading画像の指定を省略した場合は、組み込みの`Loading`コスチュームを表示します。

進捗の吹き出しは、Loading画像とは別の固定アンカーから表示されます。このため、指定画像の大きさや非透明部分の外形が異なっても、画像の切替によって吹き出し位置は変わりません。

指定名は画像アセットとして定義されている必要があります。未定義の名前を指定すると読込エラーになります。

ポーズ認識中の効果音を変更する

```
asset=Clock Ticking,https://example.com/sounds/clock-ticking.mp3
asset=Sewing Machine,https://example.com/sounds/sewing-machine.mp3
setPoseRecognitionSound=Clock Ticking,Sewing Machine
```

`setPoseRecognitionSound`には、`asset`で定義した音声アセット名を最大2件、カンマ区切りで指定します。第1音は各ポーズの認識開始時にAsset Manager経由で再生し、認識成功またはスキップでそのポーズを終えると停止します。第2音はポーズ条件が成立したとき、「ポーズ認識」の値を更新する直前に再生します。

第2音は省略でき、従来の`setPoseRecognitionSound=Clock Ticking`も同じ動作を保ちます。コマンド自体を省略するか第1音に空文字を指定した場合、認識中の音は鳴りません。第2音が空文字の場合、認識成立時の音は鳴りません。

4.3 アクター定義

```
actor=アクター名,初期スキン名
```

登場人物を登録します。

例:

```
actor=Urashima,Urashima-walk-1
actor=Turtle,Turtle
actor=Princess,Princess
```

`action=Urashima:say:...`のように、アクションの対象として使う名前がアクター名です。

4.4 表紙定義

```
cover=背景アセット名,音声アセット名
```

アプリ起動時や物語終了後に表示する表紙画面を指定します。

例:

```
cover=Beach1,OceanWave
```

`cover` の第２引数には音声アセットを指定します。表紙で音を鳴らさない場合は、区切りのカンマを残して第２引数を空にします。アプリは第２引数が空でなく、同名のアセットが読み込み済みの場合だけ音声を再生します。

```
cover=Beach1,
```

4.5 ランタイム変数

```
setRuntimeVariable=変数名:値
```

紙芝居の実行中に参照する変数へ初期値を設定します。3.1 では、開始シーンや条件分岐の状態を台本から用意できます。

```
setRuntimeVariable=startSceneIndex:1
setRuntimeVariable=takeSeaRoute:true
```

`startSceneIndex` は、最初に実行するシーン番号を指定するための予約された変数です。通常は `1` にします。

4.6 条件分岐の登録

```
registerBranch=分岐名:条件1,条件2,...:シーンラベル1,シーンラベル2,...
```

条件と移動先の組を登録します。条件はRuntime Expressionの式として上から順に評価され、最初に真になった条件と同じ位置のシーンラベルが選ばれます。

```
setRuntimeVariable=takeSeaRoute:true
registerBranch=chooseRoute:takeSeaRoute,true:ocean,home
```

最後に `true` を置くと、どの条件にも一致しなかった場合の移動先を表せます。条件とシーンラベルの個数はそろえてください。

条件式では `==`、`!=`、`===`、`!==` による等価比較を使用できます。コマンド行の最初の `=` だけがキーと値の区切りになり、条件式内の `=` は保持されます。単独の `=` は条件式の演算子ではありません。

```
setRuntimeVariable=score:1
registerBranch=chooseRoute:score == 1,true:ocean,home
```

4.7 ポーズ案内の文言

ポーズ案内は、scene 0（最初の `---` より前）で定義できます。`ui.prompt` はランタイムが自動登録する予約済みテキストアセットなので、`asset=` は不要です。

```
text=ui.prompt:ポーズをとろう！
```

アセット名	表示場所
`ui.prompt`	ポーズ認識中の案内

ランタイムは物語開始時に `Pose!` へ戻してから scene 0 を実行するため、別の台本を続けて読み込んでも前の文言は残りません。

`Open`、`Reload`、`About`、`Language`、台本エラー、タイトル画面は台本を読む前から必要なアプリ UI です。これらは台本ではなくアプリの言語定義から表示し、標準の `（言語）` ブロックまたはメニューで選んだ日本語／英語に従います。旧台本に `text=ui.open:`、`text=ui.reload:`、`text=ui.about:`、`text=ui.invalidScript:` が残っていても、アプリ UI の文言には適用されません。

5. シーンを書く

シーンは `---` で区切ります。

```
---
# scene 1
sceneLabel=opening
TMPoseURL=https://example.com/model/
action=stage:Beach1
action=wait:1
action=Urashima:show:Urashima-walk-1:172,-77,25
action=Urashima:say:今日は浜辺を散歩しよう。:2
```

5.1 シーンの基本方針

シーンごとに、必要な背景、登場人物、音、セリフを指定します。アプリはシーン終了時にアクターを隠しますが、背景、画面効果、再生中の`bgm`は次のシーンへ引き継ぎます。BGMを継続する場合は、次のシーンで同じ`bgm`を指定し直す必要はありません。背景や画面効果は、意図しない持ち越しを避けるため明示することを推奨します。

よい書き方:

```
---
# scene 2
action=stage:Ocean
action=Urashima:show:Urashima-ride-1:170,5,25
action=bgm:Odesong
```

避けたい書き方:

```
---
# scene 2
# 背景やアクターを出さず、前シーンから残っている前提で書く
```

5.2 コメント行

`#`で始まる行はコメントです。

```
# scene 1
# ここでカメを助けるポーズを入れる
```

コメントは、シーン名、演出意図、修正メモに使うと便利です。

5.3 シーンラベル

```
sceneLabel=opening
```

シーンへ一意の名前を付けます。`branch`、`keyInputToChangeScene`、`touchInputToChangeScene` の移動先として使います。分岐を使う台本では、すべてのシーンに重複しないラベルを付けることをおすすめします。

5.4 TMPoseURL

```
TMPoseURL=https://example.com/model/
```

ポーズ認識モデルのURLを指定します。ポーズ認識を使うシーンでは、原則としてそのシーン内に `TMPoseURL` を書きます。

`TMPoseURL` は、シーン内のアクション実行前に読み込まれます。そのため、次のように `action=stage` より後に書いても、実際のアクション実行前にはモデル読み込みが終わります。ただし、人間が読むときの分かりやすさを優先して、シーン冒頭に書くことをおすすめします。

```
---
# scene 6
TMPoseURL=https://example.com/open-box-model/
action=stage:Beach2
action=Urashima:pose:Urashima-open-1,Urashima-open-2:open1,open2:OpenSound,OpenSound
```

6. アクションを書く

アクションは `action=` で書きます。

```
action=対象:命令:値
action=対象:命令:値:追加値
```

大きく分けると、背景・音・待機のアクションと、アクターに対するアクションがあります。

6.1 背景を変える

```
action=stage:Beach1
```

`Beach1` は `asset=Beach1,...` で登録済みの背景画像です。

6.2 待つ

```
action=wait:1.5
```

指定秒数だけ待ちます。

6.3 音を鳴らす

```
action=bgm:GuitarChords2
action=sound:Gong
```

`bgm` は音を鳴らして次のアクションへ進みます。`sound` は音の再生が終わるまで待ちます。

名前は `BGM` ですが、実装上は「待たずに鳴らす音」と考えると分かりやすいです。ループ再生専用ではありません。

下矢印キーでシーンをスキップしても、すでに再生中の`bgm`は継続し、シーンの残りに書かれた`bgm`も再生を開始します。現在再生待ちの`sound`は停止します。

6.4 画面をフェードさせる

```
action=transition:fadeOut
action=stage:次の背景
action=transition:fadeUp
```

`fadeOut` はステージを暗くし、`fadeUp` は明るく戻します。`reset` は明るさ効果を標準値へ戻します。背景切り替えの前後に置くと場面転換を滑らかに見せられます。

白く飛ばしてから背景を切り替える場合は、次のように記述します。

```
action=transition:fadeToWhite
action=stage:次の背景
action=transition:fadeFromWhite
```

`fadeToWhite`はステージの明るさを`+100`まで上げ、その状態を保持します。白飛び中に背景を切り替えたあと、`fadeFromWhite`で明るさを`0`へ戻すと、切替後の背景が徐々に見えるようになります。

下矢印キーでシーンをスキップした場合、残りの`transition`はアニメーションを待たず、台本に書かれた順番で最終明るさだけを適用します。たとえば残りに`fadeOut`と`fadeUp`がある場合、最終状態は`fadeUp`の明るさ`0`です。

6.5 アクターを表示する

```
action=Urashima:show:Urashima-walk-1:172,-77,25
```

意味:

- 対象アクター: `Urashima`
- 命令: `show`
- スキン: `Urashima-walk-1`
- x座標: `172`
- y座標: `-77`
- サイズ: `25`

座標はScratch/TurboWarpのステージ座標です。中央が `(0, 0)`、右がプラス、左がマイナス、上がプラス、下がマイナスです。

初期スキンをそのまま使う場合は、スキン名を省略して位置とサイズだけを書けます。

```
action=Fish:show:-130,-27,70
```

6.6 アクターを移動する

```
action=Urashima:moveTo:40,-57,1.5
```

`x,y,秒数` を指定します。指定秒数でその位置へ滑らかに移動します。

すぐに移動したい場合は `setPosition` を使います。

```
action=Urashima:setPosition:40,-57
```

6.7 セリフと思考吹き出し

```
action=Princess:say:ようこそ竜宮城へ。:2.5
action=Urashima:think:あっという間におじいさんになってしまった…:3
```

秒数を省略すると、次に変更されるまで表示されます。

吹き出しを消したい場合は、空のセリフを使います。

```
action=Urashima:say:
action=Princess:think:
```

6.8 スキンを変える

```
action=Urashima:setSkin:Urashima-surprised
```

登録済みアセットの画像に切り替えます。表情やポーズ違いの画像を用意しておくと、紙芝居らしい演出ができます。

サイズも同時に変える場合は、４つ目の要素に指定します。

```
action=Urashima:setSkin:Urashima-surprised:45
```

6.9 画像や音を連続再生する

繰り返し再生:

```
action=Fish:loop:Fish1,Fish2:1,1
```

一回だけ再生:

```
action=Hero:sequence:Hero1,StepSound,Hero2:0,0.5
```

`loop` はアセット数と待ち時間の数を同じにします。最後の待ち時間の後は先頭へ戻ります。`sequence` は一回だけバックグラウンド再生し、待ち時間はアセット数より１つ少なくします。待ち時間 `0` を使うと、画像と音などを同時に開始できます。

6.10 テキストアセットを表示・更新する

ヘッダでテキストアセットとアクターを登録し、通常の `show` で表示します。

```
asset=Narration,text
actor=Narration,Narration
action=text:Narration:むかし
action=Narration:show:Narration:0,0,100
action=wait:2
action=text:Narration:むかし　むかし、あるところに...
```

`action=text:テキストアセット名:文字列` は、アクション列のその位置で表示内容を更新します。`wait` と組み合わせることで、同じテキストアセットの内容を時系列に沿って変更できます。空の文字列を指定すると内容を消せます。 テキストは、明るい背景と暗い背景のどちらでも読めるよう、既定では白文字に黒い縁取りで表示されます。

```
action=text:Narration:
```

シーン直下の `text=...` も互換性のため読み込めますが、アクション列より先に処理されます。新しい台本や順次更新には `action=text:...` を使用してください。ただし、予約済みの `ui.*` 文言は時系列の演出ではなく初期設定なので、scene 0 の `text=...` で定義します。

6.11 シーンを分岐させる

登録済みの条件分岐を評価する場合:

```
action=branch:chooseRoute
```

キー入力で移動先を選ぶ場合:

```
action=keyInputToChangeScene:ArrowLeft,ArrowRight:leftRoute,rightRoute
```

アクターへのタッチで移動先を選ぶ場合:

```
action=touchInputToChangeScene:LeftDoor,RightDoor:leftRoute,rightRoute
```

キー／アクターのリストとシーンラベルのリストは、同じ個数を同じ順番で書きます。入力待ちは登録後も他のアクションと並行して続き、入力されると指定ラベルのシーンへ移動します。

6.12 ポーズ認識を入れる

基本形:

```
action=アクター名:pose:スキン名リスト:ポーズ名リスト:効果音リスト
```

例:

```
action=Urashima:pose:Urashima-help-1:help:SquishPop
```

これは次の意味です。

1. `Urashima` のスキンを `Urashima-help-1` にする
2. TMPoseで `help` ポーズを待つ
3. 成功したら `SquishPop` を鳴らす
4. 次のアクションへ進む

複数ポーズを連続させることもできます。

```
action=Urashima:pose:Urashima-ride-2,Urashima-ride-1,Urashima-ride-2,Urashima-ride-1:ride2,ride1,ride2,ride1:Splash,Splash,Splash,Splash
```

この場合、スキン、ポーズ名、効果音のリストを左から順番に対応させます。

順番	スキン	ポーズ名	効果音
1	`Urashima-ride-2`	`ride2`	`Splash`
2	`Urashima-ride-1`	`ride1`	`Splash`
3	`Urashima-ride-2`	`ride2`	`Splash`
4	`Urashima-ride-1`	`ride1`	`Splash`

7. `urashima.txt` の構成例

公開中の `urashima.txt` は、次のような構成になっています。Web版と用途別SB3は、浦島太郎の公開ページから利用できます。

シーン	内容	主な機能
ヘッダ	バージョン、初期変数、アセット、アクター、表紙、ポーズ案内を定義	`kamishibai`, `setRuntimeVariable`, `asset`, `actor`, `cover`, `text=ui.prompt`
opening	導入のナレーション	テキストアセット、フェード
scene 1	浜辺で浦島がカメを助ける	背景、セリフ、移動、ポーズ認識
scene 2	カメに乗って海を進む	移動、連続ポーズ、効果音
scene 3	竜宮城で姫に迎えられる	姫の表示、セリフ、ループアニメーション、踊りポーズ
scene 4	玉手箱を受け取る	会話、受け取りポーズ、退場
scene 5	浜辺に戻り、村の変化に気づく	背景変更、驚きスキン、セリフ
scene 6	玉手箱を開ける	複数ポーズ、効果音
scene 7	煙の中で老人になる	同期効果音、思考、絶望ポーズ
scene 8	竜宮城の別れ	複数アクター表示、セリフ
scene 9	エンディング	背景、完了音

この例は、3.1のアセット形式、シーンラベル、scene 0のポーズ案内、時系列の`action=text:`、フェード、ループアニメーション、ポーズ認識を組み合わせた教材として使いやすいです。

8. 紙芝居DSLの作成手順

8.1 物語をシーンに分ける

最初に、物語を5〜10個程度のシーンに分けます。

例:

1. 出会い
2. 移動
3. 目的地に到着
4. 重要な選択
5. 結末

各シーンで、背景、登場人物、セリフ、音、参加者の動作を決めます。

8.2 必要なアセットを洗い出す

次の一覧を作ります。

種類	例
背景	浜辺、海、城、森、エンディング画面
キャラクター画像	通常、驚き、喜び、困り顔、動作中
効果音	決定音、ジャンプ音、拍手、波の音
BGM	場面の雰囲気を作る音
ポーズ用スキン	参加者に真似してほしい姿勢の画像

8.3 アセット名を決める

おすすめの命名規則:

```
背景: Beach1, Ocean, Castle
人物: Hero-walk-1, Hero-happy, Hero-open-1
音: OpenSound, SuccessSound, OceanWave
```

この例では、参照箇所を見分けやすくするため英数字、ハイフン、アンダースコアを使用しています。名前は文字列として照合されるため、定義箇所と参照箇所で同じ表記を使います。アクション要素として使う名前に半角 `:` があると別の要素に分割され、リスト要素として使う名前に半角 `,` があると別の項目に分割されます。

8.4 ヘッダを書く

すべてのアセットを `asset` で登録し、登場人物を `actor` で登録します。

```
kamishibai=3.1
setRuntimeVariable=startSceneIndex:1
asset=Beach,backdrop
asset=Hero,costume
asset=OceanWave,sound
actor=Hero,Hero
cover=Beach,OceanWave
```

8.5 各シーンを書く

シーンごとに、背景、表示、セリフ、移動、待機を順番に書きます。

```
---
# scene 1
sceneLabel=opening
action=stage:Beach
action=wait:1
action=Hero:show:Hero:0,-60,30
action=Hero:say:冒険に出発だ！:2
action=wait:1
```

8.6 ポーズ認識を追加する

ポーズを使うシーンには、まず `TMPoseURL` を書きます。

```
TMPoseURL=https://example.com/pose-model/
```

次に `pose` アクションを書きます。

```
action=Hero:pose:Hero-jump-1:jump:JumpSound
```

複数ポーズを連続させたい場合は、カンマ区切りで書きます。

```
action=Hero:pose:Hero-left,Hero-right:left,right:StepSound,StepSound
```

9. 書き方の注意

注意点	理由
先頭に `kamishibai=3.1` を書く	アプリが対応台本か確認するためです。
`---` でシーンを区切る	アプリがシーン単位で実行するためです。
1行に1つの命令を書く	パーサが行単位で処理するためです。
最初の `=` はコマンドの区切り	2個目以降の `=` は値として保持されます。
`:` はアクションの区切り	`say` と `think` の本文中では、コロンを全角 `：` で書きます。
`,` はリストの区切り	アセット名やポーズ名には半角カンマを入れないでください。
ポーズ名はTMPoseモデルと一致させる	モデルのラベルと違うと認識されません。
ラベル名は重複させない	条件や入力による移動先を一意に決めるためです。
分岐の条件数と移動先数をそろえる	同じ位置の条件とラベルを対応させるためです。
各シーンで表示状態を明示する	アクターは隠れますが、背景、画面効果、BGMは意図的に変更するまで持ち越されるためです。
音声ファイルは短めにする	`sound` は音が終わるまで待つためです。

10. テスト方法

10.1 まず文法を確認する

- `kamishibai=3.1` がある
- `asset`, `actor`, `cover` が正しく書かれている
- `---` がある
- すべての `action` が `action=...` で始まっている
- 分岐先の `sceneLabel` が存在する

10.2 次にアセットを確認する

- URLをブラウザで開ける
- 画像が表示される

- 音声が再生できる
- アセット名のスペルが一致している

10.3 最後に演出を確認する

- 背景が意図通りに切り替わる
- アクターの位置とサイズが適切
- セリフの表示時間が読みやすい
- 音が長すぎない
- ポーズ認識が成功する
- 条件分岐、キー入力、タッチ入力が正しい移動先を選ぶ

11. 改善しやすい台本にするコツ

11.1 シーン番号をコメントで書く

```
# scene 1: 浜辺でカメと出会う
```

11.2 演出のまとまりごとに空行を入れる

```
action=stage:Beach1
action=wait:1

action=Urashima:show:Urashima-walk-1:172,-77,25
action=Urashima:say:今日は浜辺を散歩しよう。:2
```

11.3 ポーズの意味が分かる名前を使う

よい例:

```
help
open1
open2
despair
```

分かりにくい例:

```
pose1
pose2
abc
```

11.4 長い物語は小さく作る

最初に1シーンだけ作って動かし、確認後に2シーン目を足すと、文法エラーや不足アセットをシーン単位で特定できます。紙芝居制作は、料理でいえば味見しながら作るタイプです。いきなり鍋いっぱいにしないのがコツです。

12. 作成チェックリスト

- [ ] 物語をシーンに分けた
- [ ] 背景画像を用意した
- [ ] 登場人物画像を用意した
- [ ] 音声アセットを用意した
- [ ] ポーズ認識モデルのURLを確認した
- [ ] `kamishibai=3.1` を書いた
- [ ] `asset` をすべて書いた
- [ ] `actor` をすべて書いた
- [ ] `cover` を書いた
- [ ] 各シーンを `---` で区切った
- [ ] 分岐に使うシーンへ重複しない `sceneLabel` を付けた
- [ ] 各シーンに必要な背景とアクター表示を書いた
- [ ] ポーズ名とモデルラベルが一致している
- [ ] 条件・入力と移動先ラベルの対応を確認した
- [ ] 最初から最後まで再生確認した

13. 関連ドキュメント

- `03-user-guide.md`: 紙芝居アプリの操作方法
- `05-command-reference.md`: コマンドとアクションの詳細仕様
- `01-executive-summary-adult.md`: 大人向け概要説明
- `02-executive-summary-kids.md`: 子供向け概要説明
- `06-developer-guide.md`: 成果物とビルダーの利用、開発、検証、公開の手順
- `07-internal-specification.md`: 汎用アプリSB3の内部構造、呼出し関係、状態遷移
- `history.md`: 紙芝居DSL 2.0から3.1への変更履歴