



Sheep Walk II

ステージ 1 ★

ステージ 2 ★★

ステージ 3 ★★★

ゲーム終了

概要

基本情報

タイトル	: Sheep Walk						
ジャンル	: 3Dパズルゲーム						
制作期間	: 1ヶ月						
制作人数	: <table border="1"><tr><td>プランナー</td><td>1人</td></tr><tr><td>デザイナー</td><td>3人</td></tr><tr><td>プログラマー</td><td>1人</td></tr></table>	プランナー	1人	デザイナー	3人	プログラマー	1人
プランナー	1人						
デザイナー	3人						
プログラマー	1人						
担当箇所	: プログラムすべて						
使用言語	: C#						
使用ツール	: Unity2022.1.1f1						
対応プラットフォーム	: Windows						

制作意図

授業の課題で「3Dパズルゲーム」というテーマで作成しました。

プログラマーは私1人だったため、限られた期間で確実に完成できる内容を意識しました。

ゲーム内容

クリックで地形を回転させて羊を女の子のところまで導くパズルゲーム。

時間切れもしくは、羊が障害物に当たってしまうとゲームオーバー。

アピールポイント

1. カメラ制御

1. 選択したオブジェクトのほうへ向く
2. カメラの向きに合わせてUIを回転させる

2. クリック操作

3. オブジェクトの回転

4. オブジェクトのアウトラインの表示



1. カメラ制御

1. 選択したオブジェクトのほうへ向く

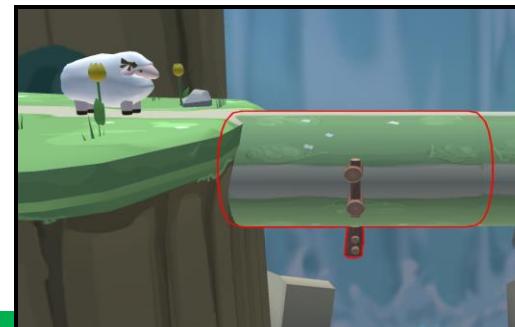
選択したオブジェクトのほうへ向く処理は以下の方法で実装しました。

- ①カメラからマウスクリックされた場所へレイを飛ばしてレイが当たったオブジェクトを取得する。
- ②取得したオブジェクトのタグ名が” Baum” だった場合は、取得したゲームオブジェクトを別の変数に格納する。
- ③カメラの位置と向きたいオブジェクトの位置からベクトルを求める。
- ④その値をLookRotation関数で回転情報に変換する。
- ⑤Slerp関数で現在地から向きたいオブジェクトまで滑らかに回転する。

該当スクリプト

①～② CameraController.cs 83行目 ～ 125行目
③～⑤ CameraController.cs 149行目 ～ 157行目

```
108 // ヒットしたオブジェクトのタグが Baum だったら
109 else if (hit.collider.gameObject.CompareTag("Baum"))
110 {
111     // クリックされたゲームオブジェクトをセットする
112     clickedGameObject = hit.collider.gameObject;
113
114     // 前と違う Baum がクリックされたら入れ替える
115     if (clickedGameObject != clickedGameObject_old)
116     {
117         clickedGameObject_old.GetComponent<Outline>().enabled = false;
118         clickedGameObject_old = clickedGameObject;
119     }
149     // バームクーヘンの方を向く
150     // ターゲット方向のベクトルを取得
151     relativePos = clickedGameObject.transform.position - this.transform.position;
152
153     // 方向を、回転情報を変換
154     rotation = Quaternion.LookRotation(relativePos);
155
156     // 現在の回転情報と、ターゲット方向の回転情報を補完する
157     transform.rotation = Quaternion.Slerp(this.transform.rotation, rotation, speed);
```



1. カメラ制御

2. カメラの向きに合わせてUIを回転させる

カメラの向きに合わせて矢印ボタンUIを回転させる処理は以下の方法で実装しました。

- ①カメラのY軸回転を90度で丸めて、矢印ボタンのZ軸の回転に代入する。
- ②横向きのオブジェクトがない場合はそこで処理を終える。
- ③横向きのオブジェクトがある場合は、矢印ボタンをさらに90度回転させる。
- ④羊の進行方向に合わせて矢印ボタンの回転を調整して、クリックされたときに正しい向きで回転できるようにする。

該当スクリプト

CameraController.cs 159行目 ~ 190行目

```
159 // カメラと一緒に矢印ボタンも回転する  
160 ArrowButton.transform.rotation = Quaternion.Euler(0, 0, Mathf.Round(transform.eulerAngles.y / 90) * 90 - 90);  
161  
162 // 横向きバームクーヘンがなければこのあと処理はしない  
163 if (landscapeBaum.Length == 0)  
164 {  
165     return;  
166 }  
167  
168 // 横向きバームクーヘンだったらさらに-90度回転させる  
169 for (int i = 0; i < landscapeBaum.Length; i++)  
170 {  
171     GameObject obj = landscapeBaum[i];  
172  
173     // クリックされたオブジェクトが横向きバームクーヘンとして登録されていたら  
174     if (clickedGameObject == obj)  
175     {  
176         float adjustedRotation = Mathf.Round(transform.eulerAngles.y / 90) * 90;  
177  
178         if (i == 3 || i == 4 || i == 5)  
179         {  
180             // 横向きで進行方向が右から左のバームクーヘン  
181             ArrowButton.transform.rotation = Quaternion.Euler(0, 0, adjustedRotation);  
182         }  
183         else  
184         {  
185             // 横向きで進行方向が左から右のバームクーヘン  
186             ArrowButton.transform.rotation = Quaternion.Euler(0, 0, adjustedRotation - 180);  
187         }  
188     }  
189 }
```



2. クリック操作

クリック操作ではレイがUIを貫通することと
クリック判定が厳しいことの2つの課題があり、それぞれ
以下の方法で解決しました。

レイがUIを貫通する問題

①EventSystem.current.currentSelectedGameObject
でUIオブジェクトの有無を調べることで解決しました。

該当スクリプト

CameraController.cs 78行目 ~ 81行目

```
78     | if (EventSystem.current.currentSelectedGameObject != null)
79     | {
80     |     |
81     | }
```



クリック判定が厳しい問題

①レイが取得したオブジェクトのタグ名が指定の名前だった場合は、取得したオブジェクトの親オブジェクトを格納する
ようにして解決しました。

該当スクリプト

CameraController.cs 88行目 ~ 95行目

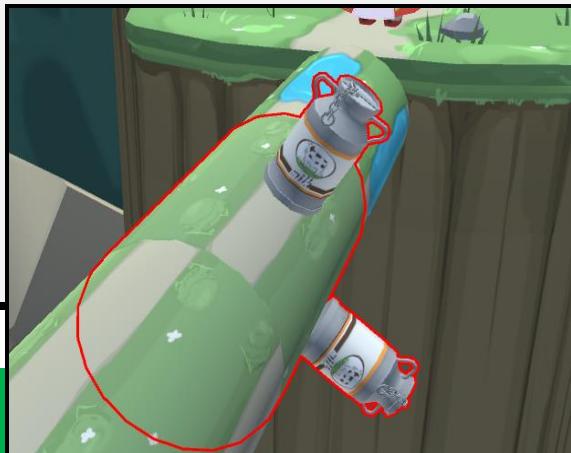
```
88     | // ヒットしたコライダーのタグがpuddleだったら
89     | if (hit.collider.gameObject.CompareTag("puddle") ||
90     |     hit.collider.gameObject.CompareTag("fence") ||
91     |     hit.collider.gameObject.CompareTag("right") ||
92     |     hit.collider.gameObject.CompareTag("left"))
93     |     |
94     |     // ヒットしたコライダーの親オブジェクトをセット
95     |     clickedGameObject = hit.collider.gameObject.transform.parent.gameObject;
```

3. オブジェクトの回転

オブジェクトが回転していることを分かりやすくするために回転をなめらかにする処理は以下の方法で実装しました。

- ① 矢印ボタンがクリックされたらオブジェクトに方向を指定して回転指示を出す。
- ② オブジェクトをLerp関数で徐々に指定された位置まで回転をする。

- ① 該当スクリプト RotateArrowButton.cs 48行目 ~ 62行目
- ② 該当スクリプト RotateBaum.cs 21行目 ~ 35行目



```
48 public void OnArrowButton()
49 {
50     // ゲームオブジェクトを代入
51     ClickBaum = cameraController.GetClickedGameObject();
52
53     // クリックされたゲームオブジェクトが空でなければ
54     if (ClickBaum != null || playerController.GetMoveStart() == false)
55     {
56         // クリックされたオブジェクトのスクリプトを取得する
57         rotateBaum = ClickBaum.GetComponent<RotateBaum>();
58
59         // バームクーヘンに回転指示を出す
60         rotateBaum.SetTargetRotation((int)dir);
61     }
62 }
```

```
21 public void Update()
22 {
23     // TimeFact秒で今いる場所から1/10まで間を詰めるための値
24     var t = 1 - Mathf.Pow(0.1f, Time.deltaTime / TimeFact);
25     transform.localPosition = Vector3.Lerp(transform.localPosition, TargetPosition, t);
26     transform.localScale = Vector3.Lerp(transform.localScale, TargetScale, t);
27     transform.localRotation = Quaternion.Lerp(transform.localRotation, TargetRotation, t);
28 }
29
30 public void SetTargetRotation(int dir)
31 {
32     // 回転方向を設定
33     TargetRotation *= Quaternion.Euler(0, 0, 90 * dir);
34 }
35 }
```

4. オブジェクトのアウトラインの表示

選択されているオブジェクトを分かりやすくするために Outline toolkit (URP) を使用してアウトラインを表示しました。

- ①アウトラインを表示したいオブジェクトのInspectorに Outlineスクリプトをアタッチ、非アクティブ化する。
- ②クリックで選択されたときにアクティブ化する。
- ③選択されたオブジェクトが変更された場合は非アクティブ化する。

該当スクリプト
CameraController.cs 88行目 ~ 123行目



Outline (Script)

Script

Outline Mode

Outline Color

Outline Width

Optional

Precompute Outline

Outline

Outline All

3

```
97 // 前と違う Baum がクリックされたら入れ替える
98 if (clickedGameObject != clickedGameObject_old)
99 {
100     clickedGameObject_old.GetComponent<Outline>().enabled = false;
101     clickedGameObject_old = clickedGameObject;
102 }
103
104 // クリックされたゲームオブジェクトのアウトラインを表示
105 clickedGameObject.GetComponent<Outline>().enabled = true;
```

反省点

反省点

- ①カメラの制御の実装に想定より多くの時間がかかったこと。
- ②デザイナーの用意した素材を期間に間に合わないという理由でゲームに挿入できなかつたこと。
- ③仕様の疑問点をすぐにプランナーに相談しなかつたため、作業が出来ない時間が生まれたこと。
- ④子どもを対象にしたゲームなのにチュートリアルがないことと、カメラの操作が複雑になつたこと。

まとめ：

積極的な行動をしなかつたため様々な弊害が出たこと

改善点・学び

- ①間に合いそうになかった場合は、早くに周囲に相談をすることが大切だと学んだ。
- ②①と同じように間に合わないと思われる場合は、チームメンバーと相談すること。
- ③積極的に行動をすることで問題を後回しにせず、やることが明確になると学んだ。
- ④ターゲットを意識したゲームを作るということを念頭に置いて開発をすることが重要だと学んだ。

まとめ：

チーム制作では周囲と協力して制作をすることが大事