

วิธีการเล่นเบื้องต้น

- ใช้ 2 account ในการเล่น โดยให้ `account1` และ `account2` เรียกใช้ฟังก์ชัน `addPlayer()` โดยวางเงินคนละ 1 ether เพื่อเข้าร่วมเกม
- ให้ แต่ละ `account` เรียกใช้ฟังก์ชัน `input()` เพื่อทำการเป่ายิ้งฉุบ โดยต้องใส่ parameter 2 อย่างคือ
 - `choice` ค่าที่ต้องการจะออกโดยที่สามารถเลือกค่าได้ตาม [นี้](#)
 - `salt` ค่า random เพื่อให้ hash เปลี่ยนไป
- เมื่อข้อ 2 เสร็จสิ้นให้เรียกใช้ฟังก์ชัน `revealChoice()` เพื่อเป็นการประกาศ ค่า `choice` ที่ตนเองเลือก โดยใส่ parameter 2 ค่า ดังนี้
 - `choice`
 - `salt`
- โดยเมื่อ ทำการ `revealChoice` เสร็จสิ้นทั้งสองผู้เล่น ระบบจะตรวจสอบว่าผู้เล่นใดเป็นผู้ชนะ และมอบ `reward` ให้ผู้เล่นที่เป็นฝ่ายชนะ
- กรณีผู้เล่นต้องการถอนเงินคืน สามารถเรียกใช้ฟังก์ชัน `withdraw()` เพื่อถอนเงินคืนได้ โดยต้องเป็นไปตาม [เงื่อนไข](#)

1. ปัญหา: Front Running

แก้ไขโดยใช้ commit reveal โดยเมื่อผู้เล่นเรียกใช้ฟังก์ชัน `input()` จะทำการ commit `choice` ที่ตนเลือก จากนั้นเมื่อ commit ครบทั้งสองผู้เล่นแล้ว ให้แต่ละผู้เล่นทำการ reveal ผ่านฟังก์ชัน `revealChoice()` เพื่อแจ้งค่า `choice` ที่ตนเลือก

2. ปัญหา: ล็อคเงิน

แก้ไขผ่านฟังก์ชัน `withdraw()` ซึ่งจะสามารถคืนเงินได้ ภายใน 3 เงื่อนไข ดังนี้

- มีคนลงขันแค่คนเดียว
- มีคนลงขัน 2 คน แต่มีคน commit แค่คนเดียว
- มีคนลงขัน 2 คน มีคน commit 2 คน แต่ยอม reveal แค่คนเดียว

โดยจะทำการคืน `reward` ทั้งหมดให้ผู้ที่ไม่ผิดกติกาเพียงคนเดียว

3. ปัญหา: ยากต่อการจะรู้ว่า account ใดเป็น `idx` เป็น 0 หรือ 1

ทำการสร้างตัวแปร `addressToPlayer` ซึ่งเป็นตัวแปรประเภท mapping (`address => uint`) ทำให้เราสามารถเข้าถึง `idx` ของ `player` ได้โดยตรง ส่งผลให้เราไม่จำเป็นต้องใส่ค่า `idx` ในการเรียกฟังก์ชัน `input()`

4. ปัญหา: Contract เล่นได้แค่รอบเดียว

แก้ไขโดยใช้ฟังก์ชัน `_restartGame()` ซึ่งจะทำการรีเซ็ตค่าทุกอย่างเพื่อให้สามารถเริ่มเกมใหม่อีกครั้งโดยไม่ต้อง Deploy Contract ใหม่

5. ปัญหา: เกมซับซ้อนมากยิ่งขึ้น

แก้ไขโดยให้ **choice** มีค่าได้ถึง 0 - 7 โดยที่

- 0 - Rock 🪨
- 1 - Water 💧
- 2 - Air 🌬️
- 3 - Paper 📄
- 4 - Sponge 🧽
- 5 - Scissors ✂️
- 6 - Fire 🔥
- 7 - Undefined ❌

การแสดงผลการทดสอบ 🧪

1. กรณีมีผู้แพ้ชนะ 🏆

1. เริ่มต้น ทั้งสอง **Account** มียอดเงินเท่ากับ 100 Ether

```
✓ 0x583...40225 (100 ether)
  0xD8...92148 (100 ether)
```

2. **Account** ทั้งสอง ทำการลงขันด้วย 1 Ether -> เหลือประมาณ 98.999 Ether

```
(98.9999999999999862004 ether)
(98.9999999999999874304 ether)
```

3. **Account** ทั้งสองทำการเข้ายังขั้ว ดังนี้

- **Account1** ทำการออก **choice** = 1 (Water) 💧 พร้อมกับใส่ **salt** = "salt_for_account1"

input

choice:

1

salt:

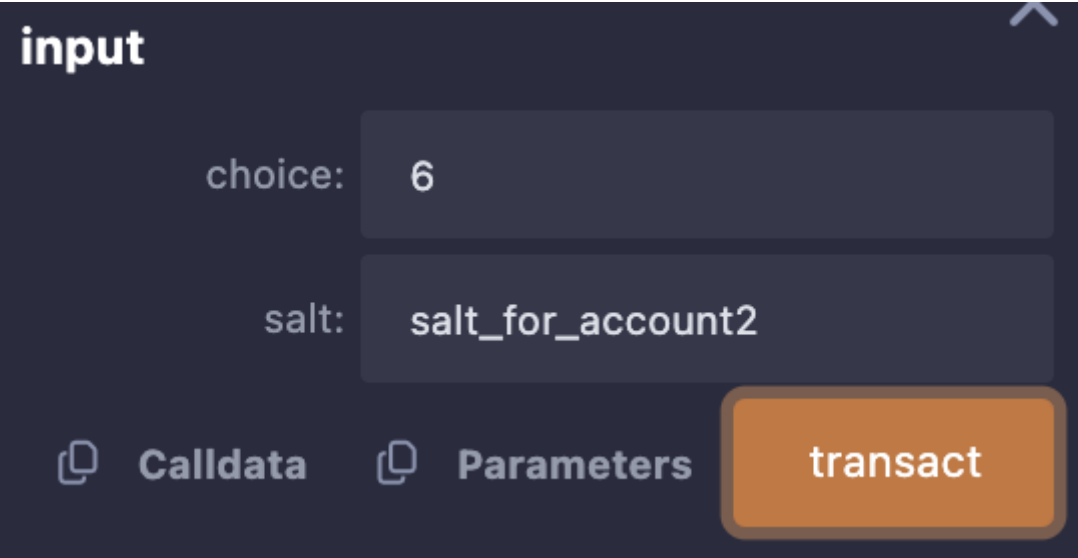
salt_for_account1

Calldata

Parameters

transact

- Account2 ทำการออก Fire (choice=6) พร้อมกับใส่ salt = "salt_for_account2" 🔥



input

choice: 6

salt: salt_for_account2

Calldata Parameters transact

4. Account1 ชนะ Account2 เนื่องจาก น้ำ💧 ชนะ ไฟ🔥 ดังนั้น Account1 จะได้ reward

✓ 0x583...40225 (100.999999999999964239 ether)
0xD8...92148 (98.9999999999999695966 ether)

2. กรณีเสมอ =

1. เริ่มต้น ทั้งสอง Account มียอดเงินเท่ากับ 100 Ether

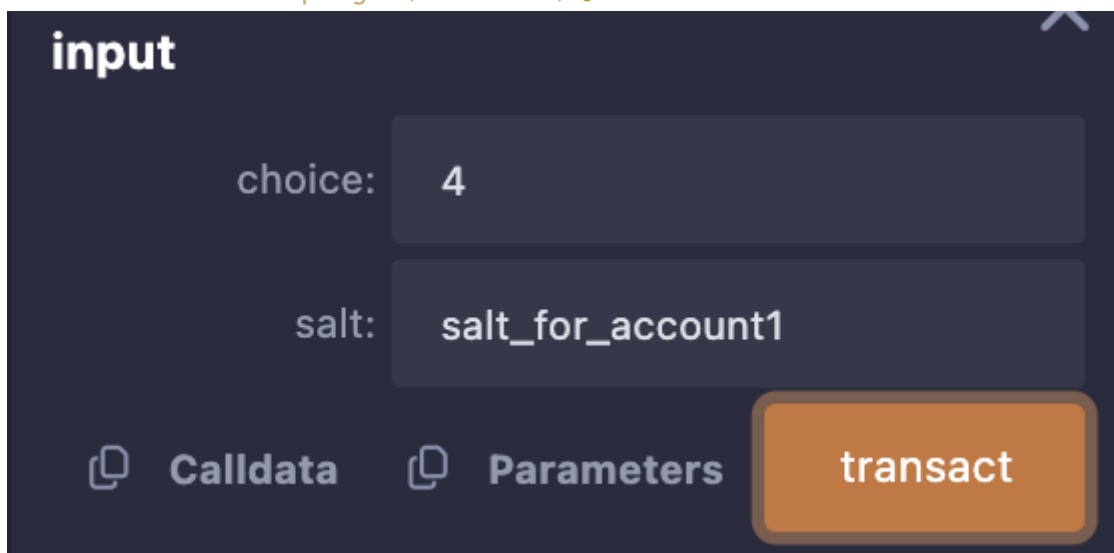
✓ 0x147...C160C (100 ether)
0x4B0...4D2dB (100 ether)

2. Account ทั้งสอง ทำการลงขันด้วย 1 Ether -> เหลือประมาณ 99 Ether

0x147...C160C (98.9999999999999862004 ether)
✓ 0x4B0...4D2dB (98.9999999999999874304 ether)

3. Account ทั้งสองทำการเข้ายังขัน ดังนี้

- Account1 ทำการออก Sponge (choice=4) 🟡



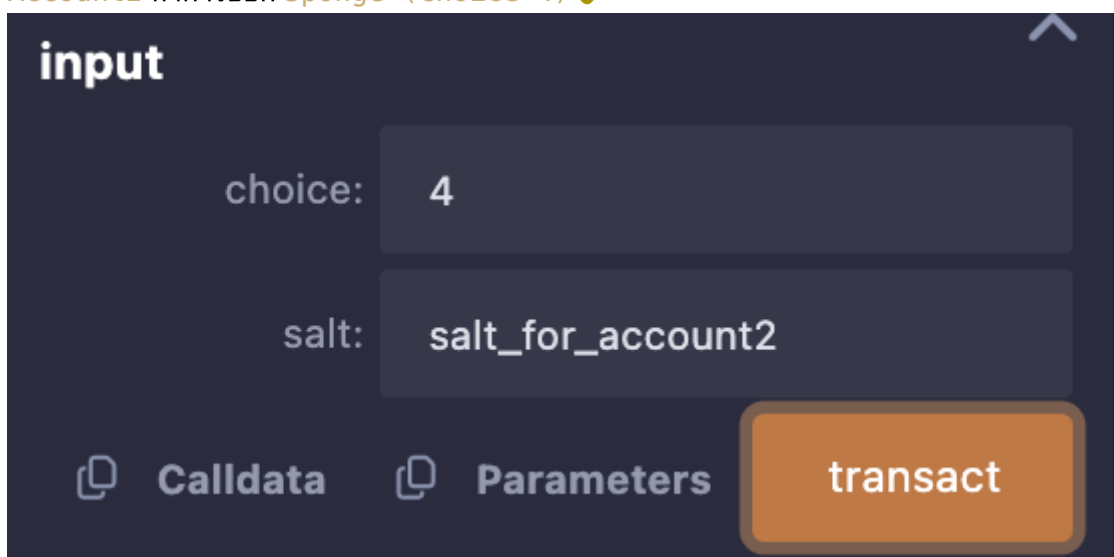
input

choice: 4

salt: salt_for_account1

Calldata Parameters transact

- Account2 ทำการออก Sponge (choice=4) 🟡



input

choice: 4

salt: salt_for_account2

Calldata Parameters transact

4. Account1 เสมอกับ Account2 เนื่องจาก ออก Sponge เหมือนกัน ดังนั้น reward จะแบ่งกลับไปหาสองคนนี้

✓ 0x147...C160C (99.99999999999999634212 ether)
0x4B0...4D2dB (99.99999999999999695966 ether)