COMP4901W Homework 4/5
Name: Hangyul Son
SID: 20537267

Exercise 1- Resubmit version

- The change is made so that either Bob or Alice can retrieve their 1 ether if the other participant does not send one's choice within a certain period of time.
- The transfer() has been modified to address.call{value:, gas2300}(""). The gas limit has been set in order to ensure that malicious contract cannot create out of gas error to limit the flow of execution of the function. Also there have been measures to prevent reentrancy attack which it can be vulnerable of.
- The number of keccak() hash function is reduced in order to save up gas.

The order of the game

1. Alice and Bob sends their hashed choice through SendChoice() function.
    a. They can only send their choice only if they haven't revealed yet. Because one might resend the choice after both choices have been revealed.
    b. The change is made so that either Bob or Alice can retrieve their 1 ether if the other participant does not send one's choice within a certain period of time.
2. Alice and Bob reveals their choice.
    a. They can only reveal their choice after they have sent their hash of the choice.
    b. The contract makes sure the sent choice and password is valid by matching their hashes using keccak() function.
    c. Alice or Bob might refuse to reveal their choice to mess up the game. Therefore set a deadline using the block.number once either Alice or Bob declares the choice.
3. Alice and Bob compare their choice. Decide the winner.
    a. Anybody can compare the choice once the deadline passes or if Alice and Bob both have declared their choice.
    b. If Alice or Bob have not declared the choice, then the smart contract considers this as a forfeit and sends 2 ether to whom have made the choice.
    c. Set gameOver variable to true.

- One might perform a front running attack. In other words Alice or Bob can see each other's transaction by having lots of nodes spread out. Or can be a miner oneself. And the attacker can send his choice after checking the opponent player's choice which would be considered cheating.
- Even though Alice or Bob misses the transaction, it will be mined and become available to everyone in accessbile to the blockchain. Then the same attack can be performed.
- Therefore, Alice and Bob must hash the choice before sending the choice. It is important to hash with a nonce as there are only 3 possible choices which the opponent might brute force to cheat in the game.
- After Alice or Bob reveals their choice if one recognizes that the game will be lost, one might not reveal their choice. This won't let the game continue and therefore must be punished.
- No third-party can tamper the game because in order to send the choice and reveal it, the smart contract requires the player to be either Alice or Bob. The transfer function is only related to Alice and Bob which makes the malicious entity impossible to steal ether. The malicious cannot deposit any unit of currency to the contract as it does not have a fallback function and the only payable function is sendChoice which requires the sender to be Alice or Bob. However, even if they could send currency to the contract, the game result and its reward amount won't change.

Exercise2 – Resubmit Version

The order of Auction

1. Seller creates a smart contract for auction.
    a. The seller defines the deposit for the item.
    b. The deadline of making the bid is for 24 hours which approximately translates to 6429 blocks, is initialized. The entire auction ends within approximately 48 hours, 24 hours for making the bid and 24 hours for checking if they have had the highest bid.
2. The participants can enter the bid for 24 hours before the auction ends.
    a. The bid amount should not be revealed until the deadline. Therefore, the participants hash their actual amount of bid and the password.
    b. The participants must make a deposit 'Larger' than the original bid amount. For example if one hopes to bid 100ETH, one must deposit more than 100ETH.
    c. The hashed bid and deposit amount will be mapped to the each and every sender's address.
    d. The others participants of this auction would be able to look at each other's deposit amount, and would be able to estimate the maximum amount of bid the participant have made. But since there could be huge disparity in between the two values, for example deposits 100ETH but only bid 1ETH, it effectively hides one's bid amount until actual revelation.
3. The participants can reveal bid amount and become the king or just get a refund of their deposit
    a. All participants return back their deposit as long as they reveal bid amount. In other words, if a participant does not reveal within the deadline, their deposit is locked up in the smart contract. Therefore, it is impossible for a participant to join with multiple identities and reveal only few bids.
    b. If actual bid amount that participant has made is less than the deposit, one's deposit is locked up in the smart contract. Therefore, it is irrational to enter a false bid amount as it would guarantee the bidder to lock up the deposit at the moment they make the bid.

Gas estimate:

makeBid(bytes32 bid)

To make the first bid of any participant

Explanation:

1. Comparison and validity checking operation require(makeBidDeadline > block.number).
2. The initialization of value to variable 'hashedBidAmount[msg.address]'.
3. The initialization of value to variable 'depositAmount[msg.address]'.

Since the parameter for the function has the type bytes32, the gas required for handling the variable 'bid' in this function has a cap. Therefore for any possible operation, as long as the function caller calls with the required parameter type, the maximum gas that would used will be at most 50000 gas. There is no other operation other than initializing a variable and require() operation which both not are operations that depend on how or by whom the function gets called.

If the same identity makes a bid again, makeBid(bytes32 bid) function would require less gas, approximately 29000 gas, because 'bidAmount[msg.address]' and 'depositAmount[msg.address]' only changes the already initialized value instead.

It is valid to argue that for any possible operation this method would take up at most 50000 gas.

checkBid(uint256 amount, uint256 password)

To check the result of the bid:

1. require(block.number > makeBidDeadline);
   require(checkBidDeadline > block.number);
   require(depositAmount[msg.sender] >= amount);
   require(!paid[msg.sender]);

There require operation itself requires neglectable amount of gas. Also the arithmetic comparison between uint256 and an another uint256 also takes up neglectable amount of gas. Since their type is bound, which means that unlike type such as 'string' or 'bytes[]', the maximum size of each variable will have a constant cap. And therefore, for any comparison operation between uint256, the max gas fees are estimable.

2. require(keccak256(abi.encodePacked(amount, password)) == hashedBidAmount[msg.sender]);

However on the other hand, hashing operation 'keccak256(abi.encodePacked(amount, password)' requires a lot of gas. Nevertheless, the variable 'amount' and 'password' both are bounded type, uint256, and therefore, there would be a max cap in which this hashing operation would demand. abi.encodePacked basically converts the two number into concatenated bytes array which would also have a max bound depending as two bound type uint256 are willing to become converted. Based on the calculation,

3. If one is the highest bidder..

```
if(amount > kingBid){
        address payable previousKing = king;
        previousKing.call{value:kingBid, gas: 2300}("");
```

```
            king = payable(msg.sender);
            uint256 returnDeposit = depositAmount[msg.sender]-amount;
            king.call{value:returnDeposit, gas: 2300}("");
            kingBid = amount;
    }
```

The operation would make two calls in which would send ether to the designated address. The maximum amount of gas that delegated receiving contract owns is only 2300 gas. So the fallback() or receive() methods should not perform complex tasks. The rest of the operations are basically assignment operation which has a fixed amount of gas usage. Therefore, it any possible operation, all lines of code within this part of code have a fixed amount of gas usage.

<u>It is valid to argue that for any possible operation, if one has become the highest bidder checkBid() method would take up at most 88000 gas</u>

4. Failed to be the highest bidder to receive deposit and bid amount:

```
else {
            payable(msg.sender).call{value:depositAmount[msg.sender], gas: 2300}("");
    }
```

As mentioned in part 3, the max gas of the method call would only take up 2300 gas, therefore this part of the function has a max gas usage cap.

<u>It is valid to argue that for any possible operation, if one failed to become the highest bidder checkBid() method would take up at most 66000 gas.</u>

Overall, the amount of approximate gas used for every participants of the auction given that the participant is rational and the bid is made only once with the same address,

<u>50000 gas + 90000 gas = 140000 gas for the highest bidders</u>
<u>50000 gas + 70000 gas = 120000 gas who failed to become highest bidders</u>