CakeCoin: A GoLang Port

CS 168: Cryptocurrencies and Security on the Blockchain
Ian SooHoo
Stan He
San Jośe State University
ian@iansoohoo.me
stan.he@sjsu.edu

December 14, 2020

1 Introduction

For this project, we implemented a blockchain based on Spartan Gold. Spartan Gold is a Blockchain library written in JavaScript. The goal of this project was to gain intimate knowledge of all the functions of Spartan Gold, and to translate the library into GoLang.

GoLang is a much more efficient language and it is a lot like C. GoLang is popular among those who write BlockChain and Cryptocurrencies. It was written by Google and it was specifically designed for networked servers that utilize multiple CPU cores.

1.1 Project Specifics

The specifics of our project are the following:

- There are just two people working on this project port, rather than the maximum three
- We chose GoLang over Rust due to its supposed smaller learning curve
- A proposal was submitted to have the coin be based around bakeries
- We ended up calling our project CakeCoin with the smallest bits being slices of cake

1.1 Existing Spartan Gold

To begin with, Spartan Gold has a total of eleven JavaScript classes that need to be converted into GoLang. However, we decided to start with just the basics. The first of these files is block. Block is a collection of transactions with connections to previous blocks. The second class we ported is the Blockchain class which tracks the configuration of the blockchain and contains some main utility methods. Client and miner are the next two classes. A miner is a client and a client is a public/private keypair with an address. Meanwhile a miner extends client and they mine blocks looking for proofs. We translated the utils which contains methods such as hashing (Hash), generating keypairs (GenerateKeypair), and signature verification (VerifySignature). These three methods are under the utils folder. We also translated the driver file, which creates a lot of the miners and the clients. FakeNet was important as well since it does the simulation of network events. Finally, transaction also needed to be ported since all the clients and miners rely on it.

1.2 Choosing a language

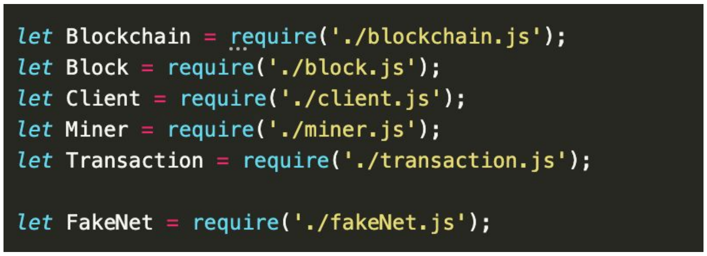
Since neither of us has any experience in these languages (GoLang or Rust), we studied GoLang and Rust. We decided to go with Go (I'm sure that's been said before). Our goal was then to learn as much about Go as we can. We accomplished this by spending time finding tutorials and other resources in order to understand how Go works and the complexities and challenges it might possess. Little did we know how complicated it was going to be. We decided we were gonna start converting SpartanGold to our chosen language after being approved. Unlike what we stated in our proposal where we said the following: "the first couple of classes may be done with peer programming or some collaboration so that both of us are able to participate in creating the foundation of the project, which can ensure quality and understanding" we ended up not doing this. We simply divided up the classes for implementation, taking time to integrate our work together to ensure our classes work. We ended up using GitHub for our project for version control and easy file sharing. Our goal was to achieve an A so we can maintain our grades in the class and explore Blockchain as much as we can to learn as much as we can.

2 Implementation

A lot of work went into this project, especially with only two people working on it. We were at a disadvantage since neither of us enjoy working with pointers or C, and GoLang was a lot like this. We met some interesting challenges along the way.

2.1 GoLang Language Basics

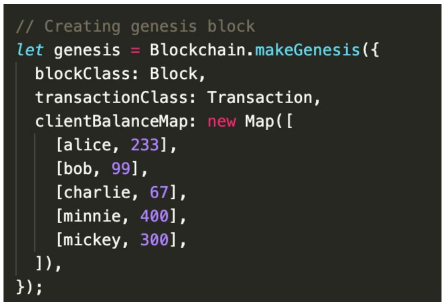
GoLang is structured in a way that all of the files in the packages are public. This means that you do not have to require and import/export files like in JavaScript.



```
let Blockchain = require('./blockchain.js');
let Block = require('./block.js');
let Client = require('./client.js');
let Miner = require('./miner.js');
let Transaction = require('./transaction.js');

let FakeNet = require('./fakeNet.js');
```

Fig. 1 driver.js, these imports are not needed in Go

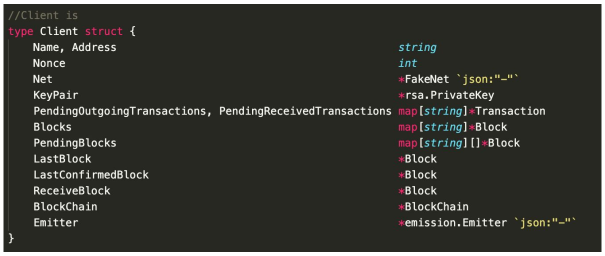
GoLang also does not support JSON objects as method arguments. In place of this, we built constructors (rather "new…" methods) for each.



```
// Creating genesis block
let genesis = Blockchain.makeGenesis({
  blockClass: Block,
  transactionClass: Transaction,
  clientBalanceMap: new Map([
    [alice, 233],
    [bob, 99],
    [charlie, 67],
    [minnie, 400],
    [mickey, 300],
  ]),
});
```

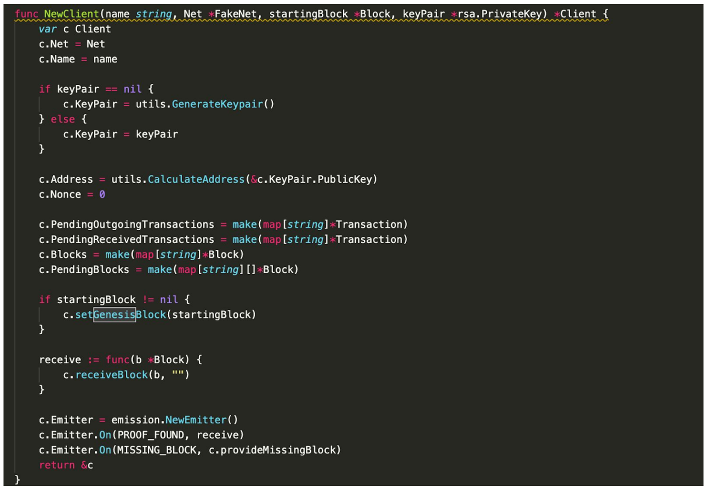
Fig 2. The genesis block creation is a JSON object in Driver.

Because there are no objects in Go, we had to work with structs. This made it incredibly difficult since there are no "this." calls, nor is there any way to create objects via a constructor. Rather objects are created using a call to a "new…" method. An example can be seen in Fig. 3. We had to collect all declared variables that were done throughout the JavaScript methods and place them in the struct, otherwise the variable would not be a part of the struct.



```
//Client is
type Client struct {
    Name, Address                                        string
    Nonce                                                int
    Net                                                  *FakeNet `json:"-"`
    KeyPair                                              *rsa.PrivateKey
    PendingOutgoingTransactions, PendingReceivedTransactions map[string]*Transaction
    Blocks                                               map[string]*Block
    PendingBlocks                                        map[string][]*Block
    LastBlock                                            *Block
    LastConfirmedBlock                                   *Block
    ReceiveBlock                                         *Block
    BlockChain                                           *BlockChain
    Emitter                                              *emission.Emitter `json:"-"`
}
```

Fig 3. All object variables MUST be declared within the struct. Client is pictured.



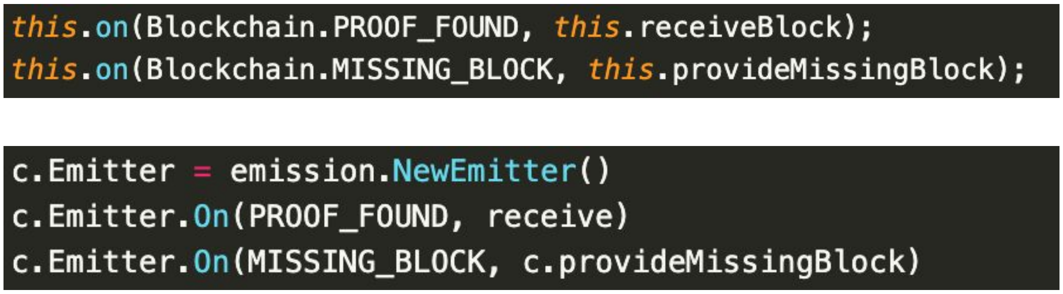
```go
func NewClient(name string, Net *FakeNet, startingBlock *Block, keyPair *rsa.PrivateKey) *Client {
    var c Client
    c.Net = Net
    c.Name = name

    if keyPair == nil {
        c.KeyPair = utils.GenerateKeypair()
    } else {
        c.KeyPair = keyPair
    }

    c.Address = utils.CalculateAddress(&c.KeyPair.PublicKey)
    c.Nonce = 0

    c.PendingOutgoingTransactions = make(map[string]*Transaction)
    c.PendingReceivedTransactions = make(map[string]*Transaction)
    c.Blocks = make(map[string]*Block)
    c.PendingBlocks = make(map[string][]*Block)

    if startingBlock != nil {
        c.setGenesisBlock(startingBlock)
    }

    receive := func(b *Block) {
        c.receiveBlock(b, "")
    }

    c.Emitter = emission.NewEmitter()
    c.Emitter.On(PROOF_FOUND, receive)
    c.Emitter.On(MISSING_BLOCK, c.provideMissingBlock)
    return &c
}
```

Fig 4. Since we cannot create a Client object, using a constructor, all initializations must be done within a “NewClient” method.

2.2 Event Emitter and AfterFunc

Client, and by extension Miner in JavaScript both extend the EventEmitter class. This means that any method with this.on (setting up listeners) would listen for this.emit. We went looking for a package that would help us do this. We found Chuck Preslar’s emission package.

(https://godoc.org/github.com/chuckpreslar/emission)



```javascript
this.on(Blockchain.PROOF_FOUND, this.receiveBlock);
this.on(Blockchain.MISSING_BLOCK, this.provideMissingBlock);
```

```go
c.Emitter = emission.NewEmitter()
c.Emitter.On(PROOF_FOUND, receive)
c.Emitter.On(MISSING_BLOCK, c.provideMissingBlock)
```

Fig 5.. Emission in JavaScript (top) vs Emission in GoLang with Chuck Preslar’s package.

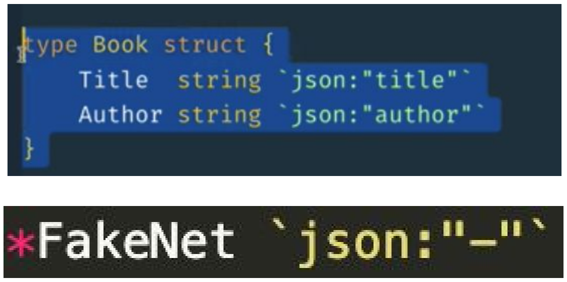
Here you see that we’re storing a NewEmitter into the Emitter variable. We have already declared the emitter variable in the struct since it needs to be part of the instance of the struct. On a different note, to replace setTimeout we used an AfterFunc timer and a time package.

2.2 Annoyances with Go (or “Go Woes”)

We have a lot of annoyances with Go. A lot of this comes down actually to JavaScript and the implementation of the port. First of all all the methods are public with constant type declarations. There are no generic object structs. Instead, an interface {} must be used. We ended up not going this route and instead being stricter with our function headers. Go also does not allow slices on Maps and there are no object constructors. Go has very little error tolerance as well. Go also does not do method overloading. Some other annoyances is a lack of “syntactic sugar”. A lot of the syntax is weird. For example variables can be declared either with var varname vartype = value OR shortened with varname := value. This is weird and makes no sense as to why the compiler can’t tell if a variable is declared or not. Go also does not have any default parameters, so we had to create a parameter, check if it is invalid like -1 and then check to see if a default value needs to be set.

2.3 JSON Marshalling and Unmarshalling

Using JSON is possible via the “encoding/json” package. There are two main functions that are used: marshal and unmarshal. When the marshal function is run, any labelled keys will receive the struct’s values. Labeled fields with `json:“=”` skips the field.

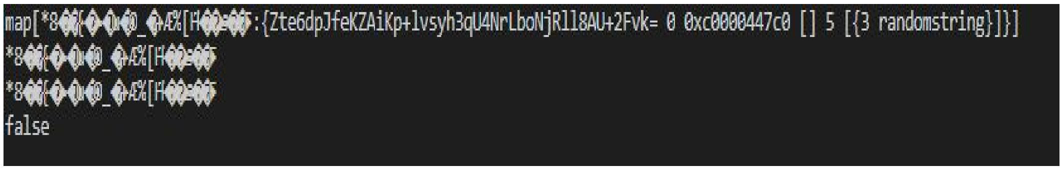


```
type Book struct {
    Title  string `json:"title"`
    Author string `json:"author"`
}
```

```
*FakeNet `json:"-"`
```

Fig 6. Assigning JSON keys to a struct field (top) and telling the marshal function to skip (bottom)

We use marshalling and unmarshalling in deserializeBlock in Blockchain and under toJson in block. FakeNet and Transaction also use it, but to a lesser extent.We ran into some issues with JSON Marshalling where it would spit out gibberish. The reason is the Marshalling function seems to do some type of processing. This then will affect finding a proof.

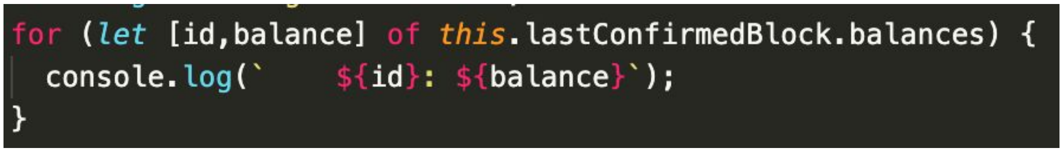


```
map[*8��{�-�)�@_�+Æ%[H��2=��:{Zte6dpJfeKZAiKp+lvsyh3qU4NrLboNjRll8AU+2Fvk= 0 0xc0000447c0 [] 5 [{3 randomstring}]}]
*8��{�-�)�@_�+Æ%[H��2=��
*8��{�-�)�@_�+Æ%[H��2=��
false
```

Fig 7. Encoding issues with JSON Marshalling

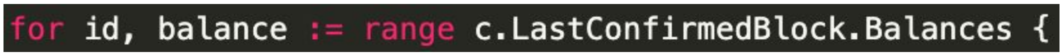
2.4 For of and ForEach, Maps, Sets

One of the annoyances of Go is there is no ForEach or "for of" methods. Rather, a for…range loop is used. It takes a range such as a Map and iterates over key,value.



```
for (let [id,balance] of this.lastConfirmedBlock.balances) {
  console.log(`    ${id}: ${balance}`);
}
```

Fig 8. "For of" in JavaScript



```
for id, balance := range c.LastConfirmedBlock.Balances {
```

Fig 9. "for…range" loop in Go

Sets are not a thing in Go, so we decided to create our on Sets struct. This is basically a Map using an interface. We decided not to go with Sets in the end, however, and so we removed it from the project file. It is still viewable in the Github history.

2.5 Default Parameters

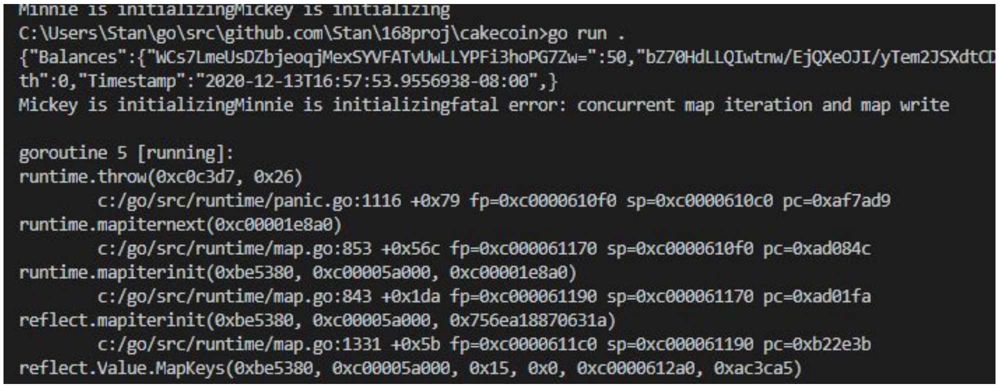
Go does not support default parameters, rather we implemented a check for nil. If it is nil then we set a default parameter. Another way we were looking to implement a default parameter was to check if the value is a nonsense number like a negative, then we set the default value. In other places like it expecting a boolean and setting a default parameter, we just decided to rewrite the code around it and not include a fix for the default parameter there.

2.6 Pointer Problems

We had lots of these. Enough said.

2.7 Concurrency Issues

We are running into some collision issues. We are working on implementing mutex locks to fix this. It can be seen as multiple access to a Map is occurring at the same time:

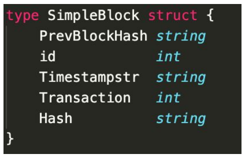


```
Minnie is initializingMickey is initializing
C:\Users\Stan\go\src\github.com\Stan\168proj\cakecoin>go run .
{"Balances":{"WCs7LmeUsDZbjeoqjMexSYVFATvUwLLYPFi3hoPG7Zw=":50,"bZ70HdLLQIwtnw/EjQXeOJI/yTem2JSXdtCD
th":0,"Timestamp":"2020-12-13T16:57:53.9556938-08:00",}
Mickey is initializingMinnie is initializingfatal error: concurrent map iteration and map write

goroutine 5 [running]:
runtime.throw(0xc0c3d7, 0x26)
        c:/go/src/runtime/panic.go:1116 +0x79 fp=0xc0000610f0 sp=0xc0000610c0 pc=0xaf7ad9
runtime.mapiternext(0xc00001e8a0)
        c:/go/src/runtime/map.go:853 +0x56c fp=0xc000061170 sp=0xc0000610f0 pc=0xad084c
runtime.mapiterinit(0xbe5380, 0xc00005a000, 0xc00001e8a0)
        c:/go/src/runtime/map.go:843 +0x1da fp=0xc000061190 sp=0xc000061170 pc=0xad01fa
reflect.mapiterinit(0xbe5380, 0xc00005a000, 0x756ea18870631a)
        c:/go/src/runtime/map.go:1331 +0x5b fp=0xc0000611c0 sp=0xc000061190 pc=0xb22e3b
reflect.Value.MapKeys(0xbe5380, 0xc00005a000, 0x15, 0x0, 0xc0000612a0, 0xac3ca5)
```

Fig 10. Concurrency issue errors

2.8 TCP/IP

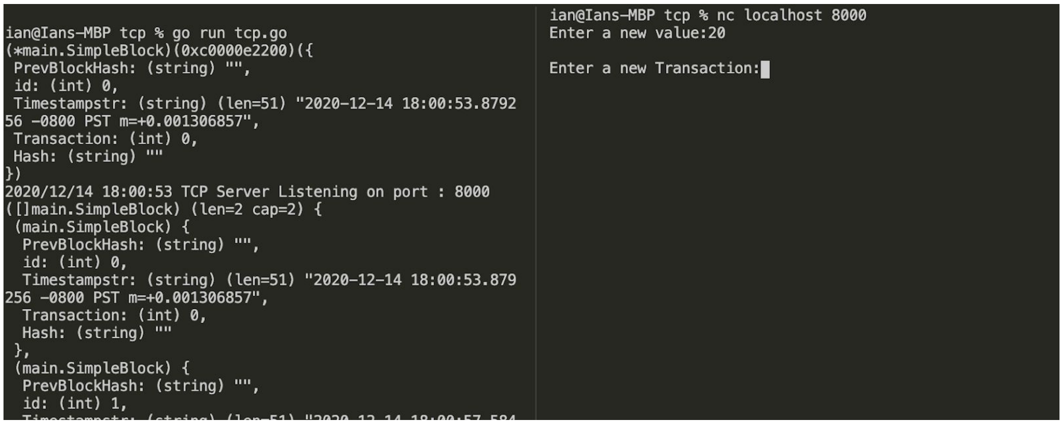
Following a few tutorials on creating an TCP/IP network in GoLang and starting with a TCP Port we are able to create a small network, it is separate from Spartan Gold, however. Methods such as hashVal, hasValidProof, and NewBlock were copied over from our Block, but I was not able to get it to work with a transaction. Rather, I used a SimpleBlock with just ints and strings. We import our utils to handle the block hashing, using the same utils port we created.



```go
type SimpleBlock struct {
    PrevBlockHash string
    id            int
    Timestampstr  string
    Transaction   int
    Hash          string
}
```

Fig 11. Simple Block retains some ofBlocks's fields, but modified to be strings and ints

First, a defer closing connection is run to prevent the connection from closing due to inactivity. Then the user is prompted to enter a new value. The new value is taken in via a Scanner. Using go functions, if it's not a number or we can't add a new block, we throw an error. Otherwise we append it to the end of the blockchain. If the chain isn't the longest, we replace it using replaceChain.



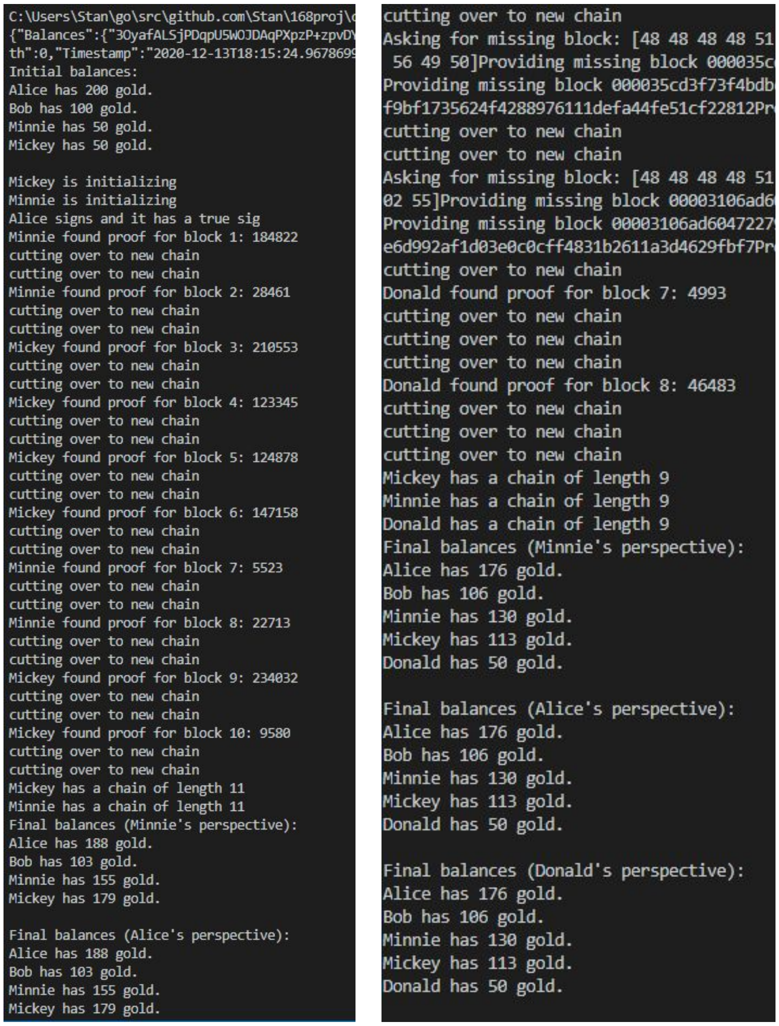
```
ian@Ians-MBP tcp % go run tcp.go
(*main.SimpleBlock)(0xc0000e2200)({
 PrevBlockHash: (string) "",
 id: (int) 0,
 Timestampstr: (string) (len=51) "2020-12-14 18:00:53.8792
56 -0800 PST m=+0.001306857",
 Transaction: (int) 0,
 Hash: (string) ""
})
2020/12/14 18:00:53 TCP Server Listening on port : 8000
([]main.SimpleBlock) (len=2 cap=2) {
 (main.SimpleBlock) {
  PrevBlockHash: (string) "",
  id: (int) 0,
  Timestampstr: (string) (len=51) "2020-12-14 18:00:53.879
256 -0800 PST m=+0.001306857",
  Transaction: (int) 0,
  Hash: (string) ""
 },
 (main.SimpleBlock) {
  PrevBlockHash: (string) "",
  id: (int) 1,
  Timestampstr: (string) (len=51) "2020-12-14 18:00:57.584
```

```
ian@Ians-MBP tcp % nc localhost 8000
Enter a new value:20

Enter a new Transaction:
```

Fig 12. A simple network running and receiving a transaction

4 Conclusion

To conclude, this project was extremely frustrating due to the source language and the destination language. We found lots of minor annoyances with GoLang that is a result of Spartan Gold using so many of the convenient features of JavaScript and other languages that just isn't available in Go. We are still working on the concurrency issues, but our Blockchain is working well.



```
C:\Users\Stan\go\src\github.com\Stan\168proj\c
{"Balances":{"3OyafALSjPDqpU5WOJDAqPXpzP+zpvDY
th":0,"Timestamp":"2020-12-13T18:15:24.9678699
Initial balances:
Alice has 200 gold.
Bob has 100 gold.
Minnie has 50 gold.
Mickey has 50 gold.

Mickey is initializing
Minnie is initializing
Alice signs and it has a true sig
Minnie found proof for block 1: 184822
cutting over to new chain
cutting over to new chain
Minnie found proof for block 2: 28461
cutting over to new chain
cutting over to new chain
Mickey found proof for block 3: 210553
cutting over to new chain
cutting over to new chain
Mickey found proof for block 4: 123345
cutting over to new chain
cutting over to new chain
Mickey found proof for block 5: 124878
cutting over to new chain
cutting over to new chain
Mickey found proof for block 6: 147158
cutting over to new chain
cutting over to new chain
Minnie found proof for block 7: 5523
cutting over to new chain
cutting over to new chain
Minnie found proof for block 8: 22713
cutting over to new chain
cutting over to new chain
Mickey found proof for block 9: 234032
cutting over to new chain
cutting over to new chain
Mickey found proof for block 10: 9580
cutting over to new chain
cutting over to new chain
Mickey has a chain of length 11
Minnie has a chain of length 11
Final balances (Minnie's perspective):
Alice has 188 gold.
Bob has 103 gold.
Minnie has 155 gold.
Mickey has 179 gold.

Final balances (Alice's perspective):
Alice has 188 gold.
Bob has 103 gold.
Minnie has 155 gold.
Mickey has 179 gold.
```

```
cutting over to new chain
Asking for missing block: [48 48 48 48 51
 56 49 50]Providing missing block 000035c
Providing missing block 000035cd3f73f4bdb
f9bf1735624f4288976111defa44fe51cf22812Pr
cutting over to new chain
cutting over to new chain
Asking for missing block: [48 48 48 48 51
02 55]Providing missing block 00003106ad6
Providing missing block 00003106ad6047227
e6d992af1d03e0c0cff4831b2611a3d4629fbf7Pr
cutting over to new chain
Donald found proof for block 7: 4993
cutting over to new chain
cutting over to new chain
cutting over to new chain
Donald found proof for block 8: 46483
cutting over to new chain
cutting over to new chain
cutting over to new chain
Mickey has a chain of length 9
Minnie has a chain of length 9
Donald has a chain of length 9
Final balances (Minnie's perspective):
Alice has 176 gold.
Bob has 106 gold.
Minnie has 130 gold.
Mickey has 113 gold.
Donald has 50 gold.

Final balances (Alice's perspective):
Alice has 176 gold.
Bob has 106 gold.
Minnie has 130 gold.
Mickey has 113 gold.
Donald has 50 gold.

Final balances (Donald's perspective):
Alice has 176 gold.
Bob has 106 gold.
Minnie has 130 gold.
Mickey has 113 gold.
Donald has 50 gold.
```

Fig 13. Blockchain working correctly (left) and a late miner (right)

5 Works Cited

- **Slice and keys**
  - **https://stackoverflow.com/questions/20297503/slice-as-a-key-in-map**
  - **https://www.reddit.com/r/golang/comments/7zikw5/why_slice_of_raw_type_eg_byte_is_not_supported_as/**
- **Iterating over Maps**
  - **https://www.golangprograms.com/how-to-iterate-over-a-map-using-for-loop-in-go.html**
- **Inheritance**
  - **https://www.geeksforgeeks.org/inheritance-in-golang/**
- **Emitters**
  - **https://godoc.org/github.com/chuckpreslar/emission**
- **Marshalling**
  - **https://www.youtube.com/watch?v=Osm5SCw6gPU&t=628s**
- **Variadic Functions**
  - **https://gobyexample.com/variadic-functions**
- **Time**
  - **https://golang.org/pkg/time/#AfterFunc**
- **Sets**
  - **https://www.davidkaya.com/sets-in-golang/**
- **GoLang does NOT support optional parameters, default parameter values, or method overloading.**
  - **https://yourbasic.org/golang/overload-overwrite-optional-parameter/**