FIAP ON

BLOCKCHAIN ADVANCED

SMART CONTRACTS

HENRIQUE POYATOS

8

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE CÓDIGOS-FONTE

SUMÁRIO

8 SMART CONTRACTS

A tecnologia do *blockchain* tem trazido consigo uma variedade de outras propostas de aplicação, evoluindo nas mais diferentes direções, e a mais promissora dentre elas é chamada de *smart contract*.

8.1. Mas o que são Smart Contracts?

Os *Smart Contracts* (ou contratos inteligentes) podem ser definidos como o próximo passo evolucionário na prevenção de fraudes (GULKER, 2017). De acordo com Raskin (2017), o contrato inteligente é um acordo entre as partes cuja execução é automática, sendo essa automação realizada por um código de computador que traduz um discurso legal em um programa executável.

Assim sendo, o algoritmo verifica com suas estruturas condicionais se determinadas condições foram satisfeitas e, caso tenham sido, ele automaticamente executa os termos estipulados no contrato. Os *Smart Contracts* permitem que as partes interessadas se comprometam previamente com os termos a serem executados sem, no entanto, determinar uma autoridade central (um sistema judicial) para fazer valer sua execução (GULKER, 2017).

Figura 8.1 – *Smart Contracts*
Fonte: FIAP (2019)

Portanto, podemos criar contratos, que outrora eram feitos no papel, de forma digital e, melhor do que isso, com ações automatizadas. Para tal, é necessário traduzir em linguagem de programação as condições e ações a serem realizadas, tornando

os contratos autoexecutáveis. Existem várias propostas de como se implementar essa tecnologia, a mais promissora e que está sendo desenvolvida há anos é a dos contratos inteligentes na plataforma Ethereum.

Uma vez confeccionado o contrato em uma linguagem de programação utilizando uma linguagem conhecida como solidity, o contrato é publicado em um *blockchain* da plataforma Ethereum. Esse procedimento traz uma propriedade importante: uma vez publicado, ele não pode ser alterado ou modificado (herdando tal característica dos *blockchain*), ou seja, para que as condições sejam modificadas, o contrato atual precisa ser destruído (rescindido) ou abandonado, e um novo deve tomar o seu lugar. Dessa maneira, há garantias de que o contrato será executado exatamente com as condições que foram previamente estabelecidas, de forma compulsória.

8.2. Aplicações em *Smart Contracts*

Graças a essa importante tecnologia, novas aplicações se tornaram possíveis. Veremos a seguir alguns dos exemplos mais proeminentes de contratos inteligentes.

8.2.1. OpenBazaar, o ML sem o ML

Uma das aplicações em funcionamento desde 2014 que eliminam intermediários é o OpenBazaar, uma plataforma de comércio eletrônico ponto a ponto (P2P), ligando compradores e vendedores diretamente. Por não haver intermediários, não há taxas ou restrições para seu uso.

Figura 8.2 – Logo do OpenBazaar
Fonte: OpenBazaar (2018)

Por funcionar em uma rede descentralizada, as informações dos anúncios e das transações realizadas não ficam em servidores centrais ou sites, funcionando em uma rede similar à rede Onion, do navegador Tor. Para acessar o *marketplace*, existe um navegador web capaz de acessar os anúncios e, para comodidade, possui uma carteira de criptomoedas embutida, a fim de realizar os pagamentos em Bitcoin, Bitcoin Cash ou Zcash (o vendedor escolhe em qual criptomoeda quer receber).

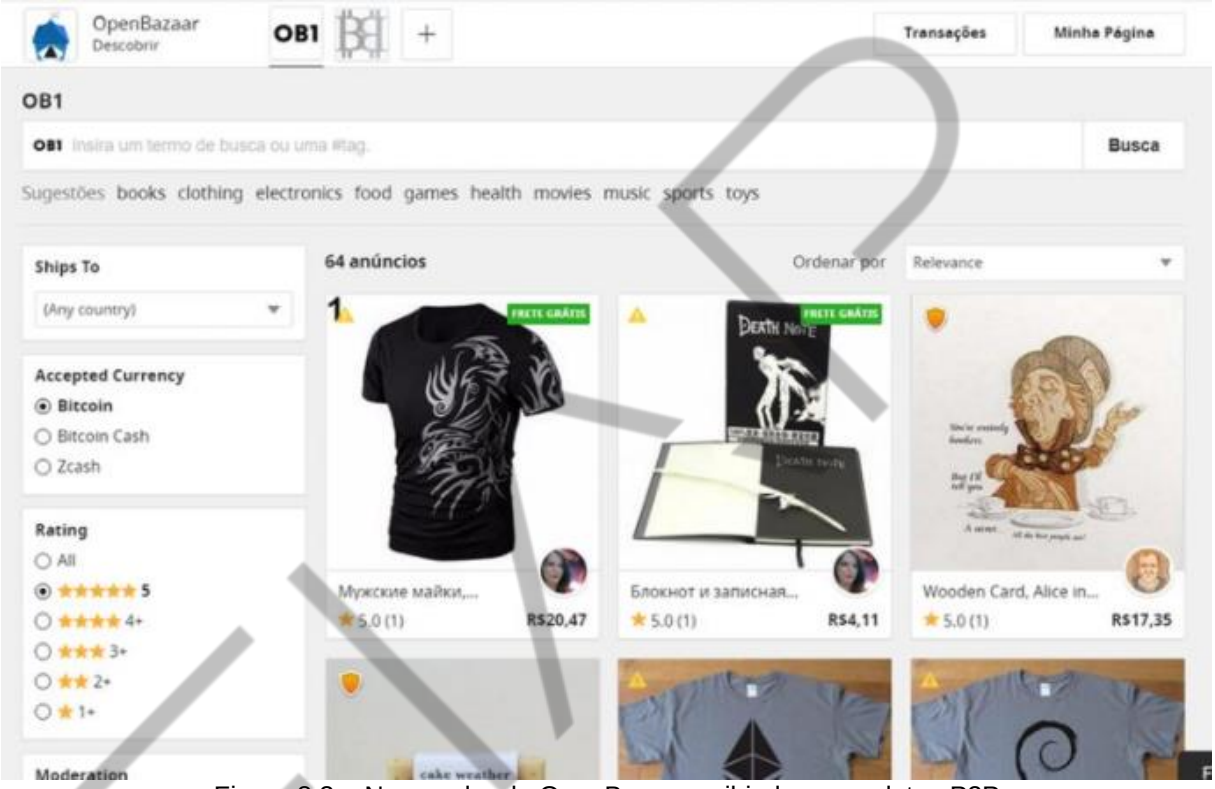


Figura 8.3 – Navegador do OpenBazaar exibindo os produtos P2P
Fonte: Elaborado pelo autor (2018)

Quando acontece uma venda, é estabelecido um contrato inteligente entre comprador e vendedor para a garantia das partes envolvidas. O sistema permite o ranqueamento de seus membros e a qualificação de vendedores após as compras, como em qualquer *e-commerce* em que estejamos acostumados a comprar. O OpenBazaar não é uma empresa, é uma comunidade de *software* livre promovendo a inovação na forma como contratamos produtos pela internet.

8.2.2. Arcade.city, o uber sem o uber

O Uber é um grande exemplo de uma nova economia baseada em *Sharing Economy*. Além disso, é uma startup *Data-driven*, ou “guiada por dados”, afinal, seu maior valor é se utilizar de tecnologias como Big Data e Inteligência Artificial para gerar valor. Se há alguma dúvida, saiba que é uma empresa que não tem bens físicos significativos ou uma frota de carros, no entanto, segundo Melo (2016), com apenas sete anos de existência, seu valor de mercado superava o das gigantes e centenárias Ford e GM.

Entretanto, a parte da “empresa sem uma frota de carros” está prestes a mudar. Segundo Gibbs (2017), para o jornal *The Guardian*, o Uber planejava comprar 24 mil veículos autônomos da Volvo ao final de 2017, e, em meados de 2018, a empresa já possuía mais de uma centena de carros em testes nas ruas de cidades nos Estados Unidos e Canadá.

Figura 8.4 – Ford Fusion autônomo da Uber em testes nas ruas de Pittsburgh
Fonte: Johnson & Fitzsimmons (2018)

Sendo assim, podemos concluir que, cedo ou tarde, a empresa vai retirar o motorista da equação, embora esse seja, hoje, uma parte importante de seu modelo de negócio. Pode levar algumas décadas, especialmente em países como o Brasil, mas certamente vai acontecer.

Contratos inteligentes permitem a eliminação de intermediários, e um grupo de motoristas em Austin, no Texas, resolveu fazer o contrário: tirar o Uber da equação.

O Arcade.City é a primeira iniciativa bem-sucedida de uma rede de viagens P2P, ou seja, motoristas e passageiros são conectados ponto a ponto. A iniciativa em Austin já está em funcionamento há alguns anos, prestando o serviço de deslocamento seguro sem incidentes (DAVID, 2017). Em 2017, a cidade contava com mais de 150 motoristas ligados à rede de viagens (WISTROM, 2017).

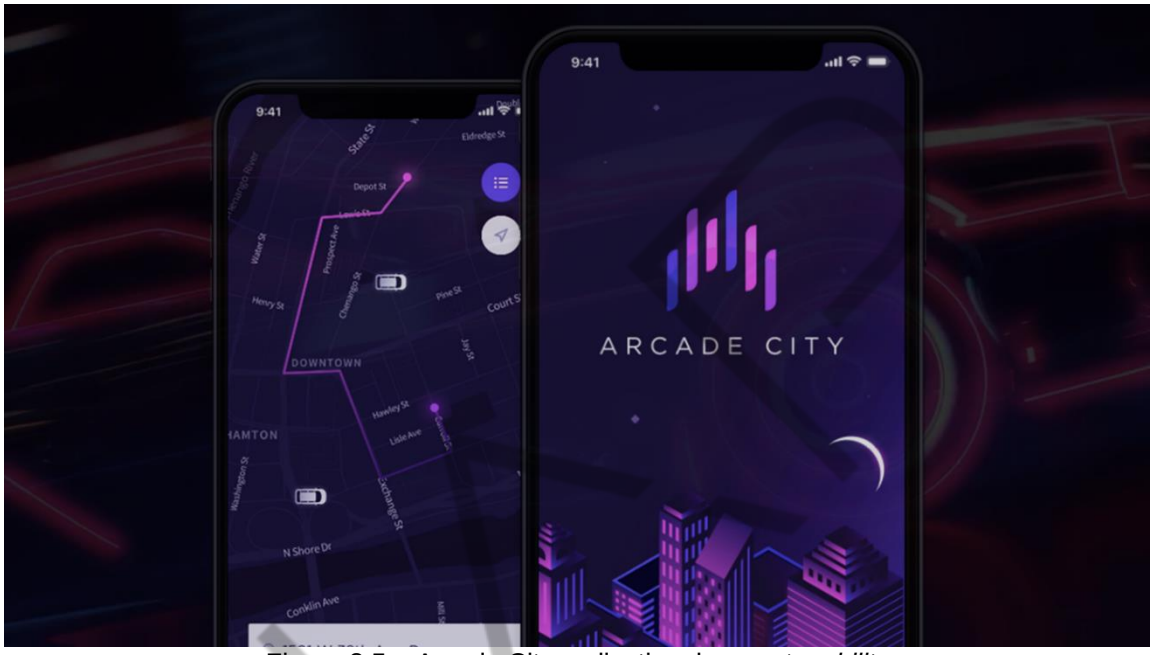


Figura 8.5 – Arcade City, aplicativo de *smart mobility*
Fonte: Arcade.City (2018)

Por consequência desse modelo, os motoristas são livres no serviço para definirem suas próprias tarifas, criar sua própria rede de clientes frequentes ou oferecer serviços adicionais, como entregas ou viagens. Assim como os concorrentes, o Arcade.City permite a avaliação de motoristas e passageiros, formando um sistema de reputação que garante mais segurança aos envolvidos.

Algumas experiências em outras cidades e países, como é o caso do Rio de Janeiro, já foram realizadas. O serviço realizou um ICO de seu Arcade Token com o intuito de custear o desenvolvimento de um aplicativo mobile que possa ser utilizado em qualquer país e cumpra as regras que foram estabelecidas no *whitepaper*.

8.2.3. Musicoin, direitos autorais fonográficos

A indústria musical sofreu várias revoluções ao longo das décadas: desde o fonógrafo de Thomas Edson, que gravou a primeira voz humana, passando pelo

walkman da Sony, os CDs, o iTunes e, finalmente, no modelo vigente, os *streamings*, dos quais temos como opções o Spotify, YouTube Music, Amazon Music, Deezer e vários outros. Entretanto, essas novas gigantes, por vezes, acabam ditando “suas regras para o mercado” e, assim, vários artistas acabaram declarando guerra ao Spotify e à Apple Music (como a Taylor Swift, no passado), queixando-se do baixo valor de repasse desses grandes serviços.

Outro fator a ser levantado são os números apresentados pelos serviços, afinal, quando o Spotify diz ao artista que sua música foi executada X milhões de vezes e, por consequência irá receber Y, pergunta-se até onde esses números correspondem à realidade, afinal, são esses mesmos serviços que realizam as apurações. Não resta ao artista outra alternativa senão confiar no intermediário.

Se o *blockchain* e os *smart contracts* têm o poder de eliminar intermediários, seria possível pensar em um modelo de *streaming* de áudio sem o Spotify? Ah, sim, e é aí que entra o Musicoin.

Figura 8.6 – Logo do Musicoin
Fonte: Musicoin (2018)

O **Musicoin** propõe um modelo de remuneração ao artista com base em um contrato PPP (*pay-per-play*, ou pagamento por execução), ou seja, toda vez que uma música for executada, um valor será enviado para a remuneração dos artistas, o contrato autogerenciável já faz a divisão financeira automaticamente. No exemplo da Figura “Funcionamento do Musicoin”, temos o seguinte: o guitarrista recebe 1/4 do valor, o baterista, o outro 1/4 e o cantor e compositor da música recebem os 2/4 restantes.

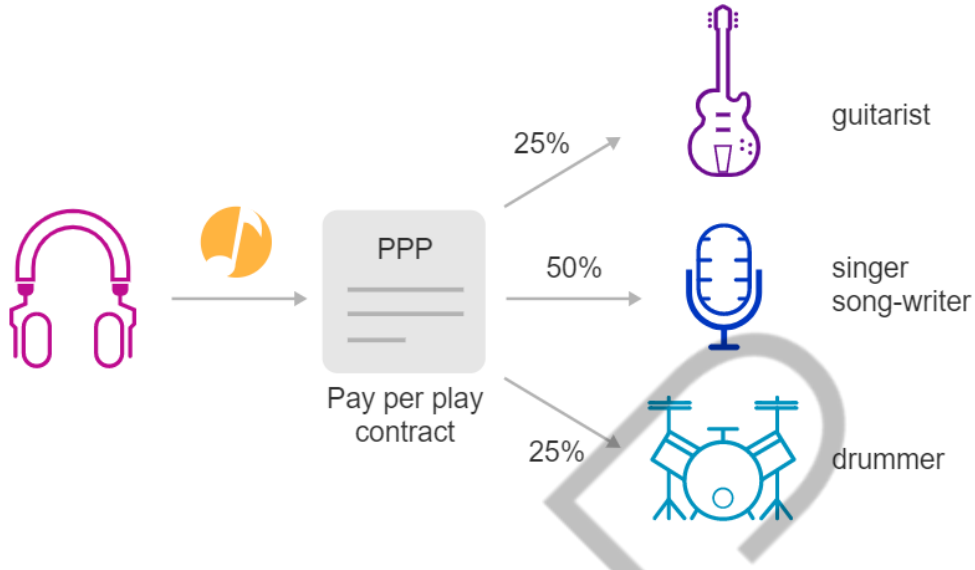


Figura 8.7 – Funcionamento do Musicoin
Fonte: Musicoin, adaptado por FIAP (2019)

Cada música teria o seu próprio contrato com uma distribuição diferente, ou seja, em outra música do mesmo álbum, podemos ter outra pessoa como compositor da letra, recebendo 10%, um produtor recebendo 15% e assim por diante.

Os ouvintes não precisam necessariamente pagar por execução, embora os artistas sejam remunerados dessa maneira. Em seu modelo de negócio existe uma criptomoeda que é minerada, conhecida como Musicoin (MUSIC), e 15% da recompensa pela mineração vai para um fundo chamado *Universal Basic Income* (UBI). Quando ouvimos uma música gratuitamente no serviço de *streaming* do Musicoin (e por meio de um eficiente aplicativo para smartphone), é a partir desse fundo que o artista é remunerado. Além disso, ouvintes podem comprar os Musicoins (ou $MUSIC) e dar *tips* (gorjetas) para seus artistas favoritos, encorajando-os a continuar seu bom trabalho.

8.3. *Smart Governance*, votação 2.0

> “Em essência, um blockchain é compartilhado, programável, criptograficamente seguro, tornando-se, portanto, um livro-razão confiável que nenhum usuário controla e que pode ser inspecionado por qualquer pessoa.” (SCHWAB apud KO, 2018, tradução livre)

Há sempre suspeitas de fraude em qualquer eleição realizada em qualquer nação do mundo; há muita coisa em jogo, e os vencedores têm acesso a altos valores e muito, muito poder. Assim sendo, sistemas de votação sempre levantarão suspeitas, especialmente dos perdedores. “Sempre”, será?

Existem várias propostas para o uso de *smart contracts* em sistemas de votação inteligentes, afinal, suas propriedades de imutabilidade sem que haja controle por nenhuma das partes tornam as eleições uma aplicação perfeita para a tecnologia.

Figura 8.8 – Votações inteligentes usando *smart contracts*
Fonte: FIAP (2019)

McCorry e colaboradores (2017) relatam ter implementado o conceito na rede Ethereum, chamando sua implementação de *Open Vote Network*:

> Apresentamos a primeira implementação de uma abordagem descentralizada e protocolo de voto pela internet com máxima privacidade dos eleitores usando o blockchain. A *Open Vote Network* (Rede de Votação Aberta) é adequada para as eleições de diretoria e está escrita como um contrato inteligente para a Ethereum. Ao contrário de protocolos de e-votação blockchain propostos anteriormente, esta é a primeira implementação que não depende de qualquer autoridade confiável para calcular o registro ou proteger a privacidade do eleitor. Em vez disso, a *Open Vote Network* (Rede de Votação Aberta é um protocolo auto-organizado, e cada eleitor está no

> controle da privacidade de seus próprios votos, de tal forma que só podem ser violados por uma colusão completa envolvendo todos os outros eleitores. A execução do protocolo é aplicada usando o consenso mecanismo que também protege o blockchain da rede Ethereum. Nós avaliamos a implementação na rede oficial de testes da Ethereum para demonstrar sua viabilidade. (MCCORRY et al., 2017, tradução livre).

Além de garantir a privacidade, o contrato traz transparência ao processo de apuração, pois, ao verificar as condições em que os votos foram apurados, é possível garantir que 1 pessoa = 1 voto. Ao utilizar um *blockchain* como infraestrutura para que o sistema de votação seja realizado, reduz-se drasticamente quaisquer fraudes que possam hoje ser realizadas em sistemas eleitorais tradicionais.

A tecnologia pode garantir uma transparência e audibilidade sem precedentes em processos democráticos, eliminando suspeitas em sistemas eleitorais como os que aconteceram na Bolívia em 2019, ou mesmo em aplicações de menor porte, como sistemas inteligentes de votação em parlamentos.

8.4. O *Smart contract* mais promissor é da plataforma Ethereum

A **Ethereum Foundation** vem agitando o mundo da *blockchain* desde o início do projeto no final de 2013 e início de 2014. Existem várias propostas do que seria a evolução do Bitcoin, e o projeto apresenta uma das mais interessantes possibilidades de “Bitcoin 2.0”. O Ethereum é um projeto de *blockchain* com uma criptomoeda, Ether, semelhante ao Bitcoin, mas, diferentemente do Bitcoin, cujas operações se restringem a adições (na carteira destino) e subtrações (na carteira de origem), a plataforma Ethereum tem o recurso adicional de uma linguagem de máquina virtual (quase) completa de Turing e capacidade de processamento incorporada à implementação do nó (MOLECKE, 2017). O Ethereum é a resposta para a pergunta: “e se o dinheiro fosse programável? ”.

Responder a essa pergunta tão ousada exige uma plataforma que suporte uma tecnologia conhecida como contratos inteligentes ou *Smart Contracts*, propostos por Nick Szabo ainda em 1997 (SZABO, 1997). Para tal, além da plataforma Ethereum, foi necessário criar uma linguagem específica para esse fim, a linguagem Solidity.

8.4.1. Solidity, a linguagem usada na plataforma Ethereum

Para possibilitar a implementação de *Smart Contract*, a plataforma Ethereum optou pela criação de uma linguagem de programação orientada a objetos e em alto nível para esse fim. Influenciada por C++, Python e JavaScript, **surgiu a linguagem Solidity**, atualmente na versão 0.7.0 (SOLIDITY, 2020).

Trata-se de uma linguagem fortemente tipada (são aquelas em que declaramos obrigatoriamente os tipos de variáveis), permite tipos complexos customizados e suporta bibliotecas e herança (SOLIDITY, 2020).

Assim como a linguagem Java, o Solidity é uma linguagem compilada e interpretada ao mesmo tempo: o código-fonte escrito na linguagem é armazenado em arquivos .sol e são submetidos a um compilador chamado ***Solidity Compiler* (solc)** e se transformam em bytecode; esse contrato compilado é publicado na plataforma Ethereum, que possui um interpretador conhecido como ***Ethereum Virtual Machine* (EVM)**, que é responsável por interpretar este bytecode. A diferença em relação à linguagem tradicional como o Java é que essa EVM é descentralizada, e o processamento do contrato acontece em milhares de nós que compõem uma rede Ethereum atualmente.

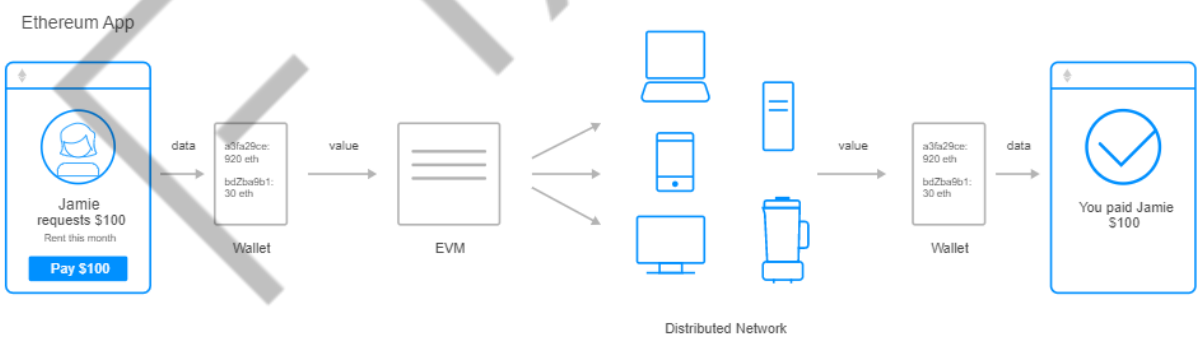


Figura 8.9 – Transferência de valores em uma plataforma Ethereum
Fonte: Coindesk (2019)

Com o Solidity, você pode criar contratos para usos como votação, financiamento coletivo, leilões cegos e carteiras com várias assinaturas.

8.5. Exemplo prático

Vamos a alguns exemplos práticos, para consolidar a ideia dos contratos inteligentes de maneira mais tácita. Para começarmos nossas primeiras implementações em contratos inteligentes na plataforma Ethereum, não será necessário computador potente ou complicadas instalações de *software*, bastará um modesto *desktop/laptop* com um navegador Google Chrome.

8.5.1. Instalação da carteira digital Metamask

Vamos começar com a abordagem mais simples, para a qual é necessária a instalação de um *plugin* no Google Chrome, e esse, em questão, é o Metamask, que embute uma carteira virtual de Ethers (e tokens ERC-20 e ERC-721, dois tipos de criptos que podem ser criados na rede Ethereum), permitindo transferir valores e interagir com as *dApps*, os *Decentralized applications*, que nada mais são do que contratos inteligentes com uma interface amigável. Com o Google Chrome, acesse metamask.io e clique no *link* "*Get chrome extension*":

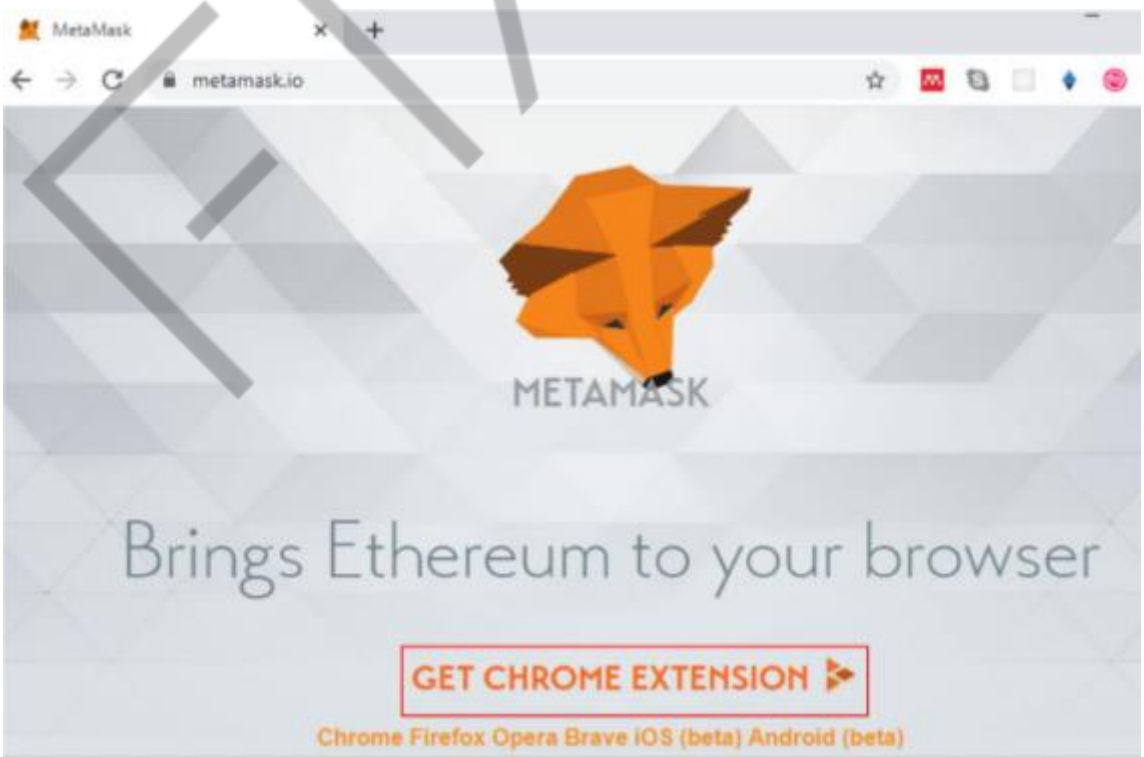


Figura 8.10 – *Website* do Metmask.io
Fonte: Elaborado pelo autor (2020)

Ao ser direcionado para a ***Chrome Web Store***, clique em “Usar no Chrome” e posteriormente “Adicionar extensão”.

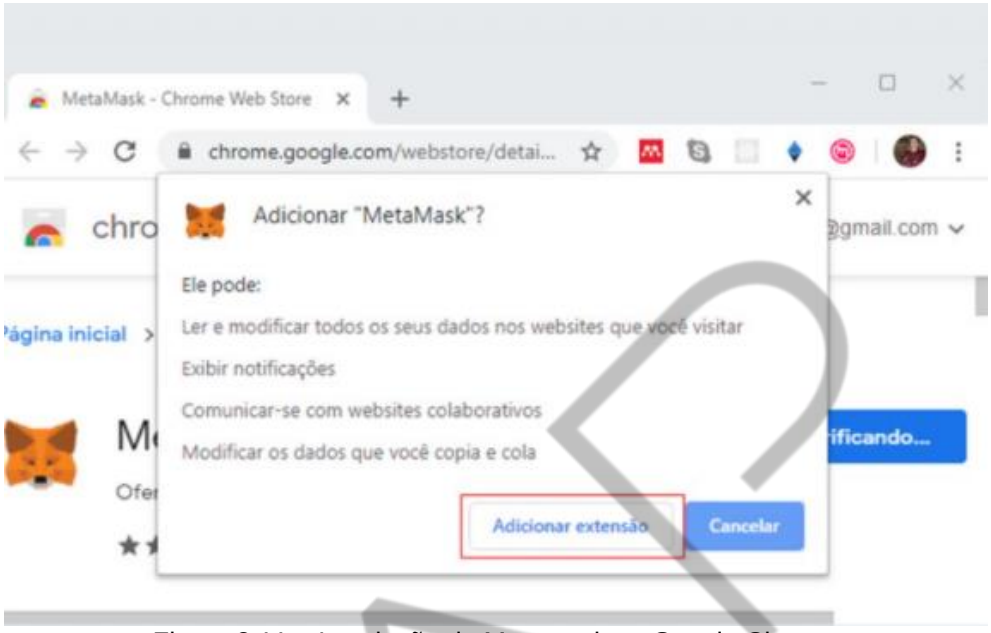


Figura 8.11 – Instalação do Metamask no Google Chrome
Fonte: Elaborado pelo autor (2020)

Metamask instalado, clique no botão “Primeiros passos”.

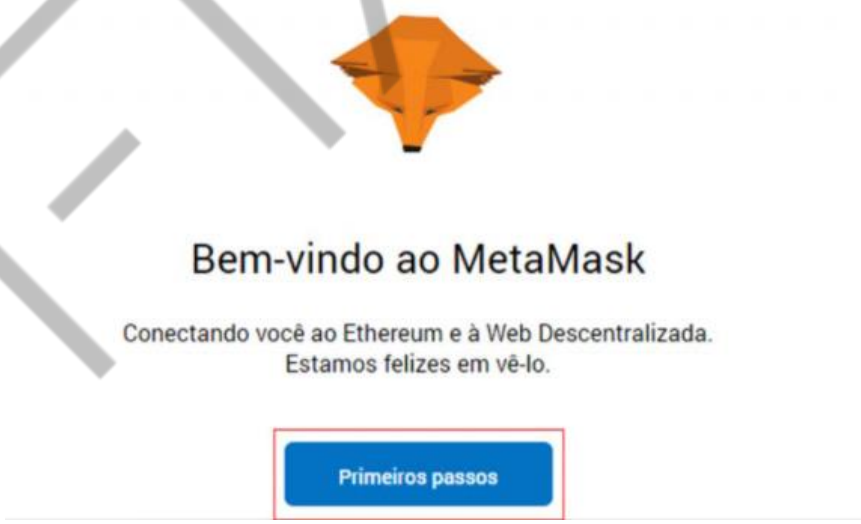


Figura 8.12 – Primeiros passos na criação da carteira digital Metamask
Fonte: Elaborado pelo autor (2020)

Como estamos criando uma carteira pela primeira vez, clique no botão “Crie uma Carteira”. Posteriormente, a carteira criada fornece doze palavras no idioma inglês que, na sequência, podem restaurar a carteira em qualquer desktop (ou mesmo apps no smartphone). Daí a opção “Importar Carteira”.

Figura 8.13 – Começo da criação da carteira no Metamask
Fonte: Elaborado pelo autor (2020)

O *plugin* solicita o envio de dados para melhoria das próximas versões do Metamask. Clique no botão que julgar mais adequado e seguimos adiante.

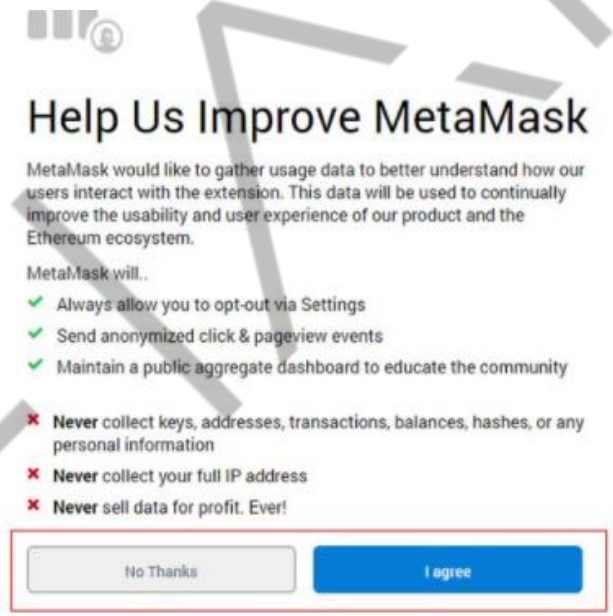


Figura 8.14 – Permissão de envio de dados no Metamask
Fonte: Elaborado pelo autor (2020)

Na sequência, informe a senha que será utilizada para assinar as transações, aceite o termo de uso e clique no botão “Criar”.

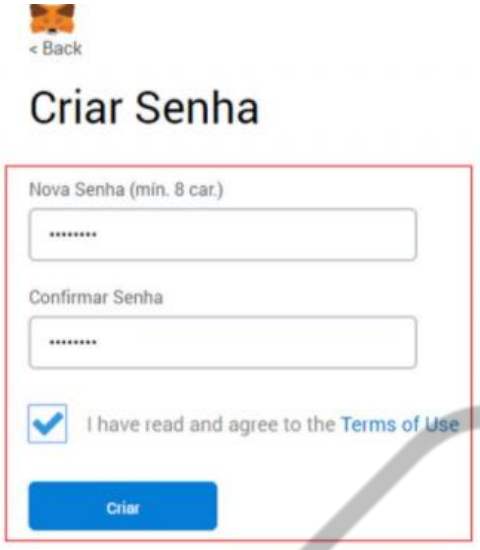


Figura 8.15 – Criação da senha de transações do Metamask
Fonte: Elaborado pelo autor (2020)

A frase secreta de *backup* composta por doze palavras no idioma inglês (conhecido como mnemônico). Clique para revelar e anote as doze palavras, ou clique em “Lembrar mais tarde”.

Figura 8.16 – Frase secreta de backup no Metamask
Fonte: Elaborado pelo autor (2020)

Pronto! A carteira no Metamask foi finalmente criada.

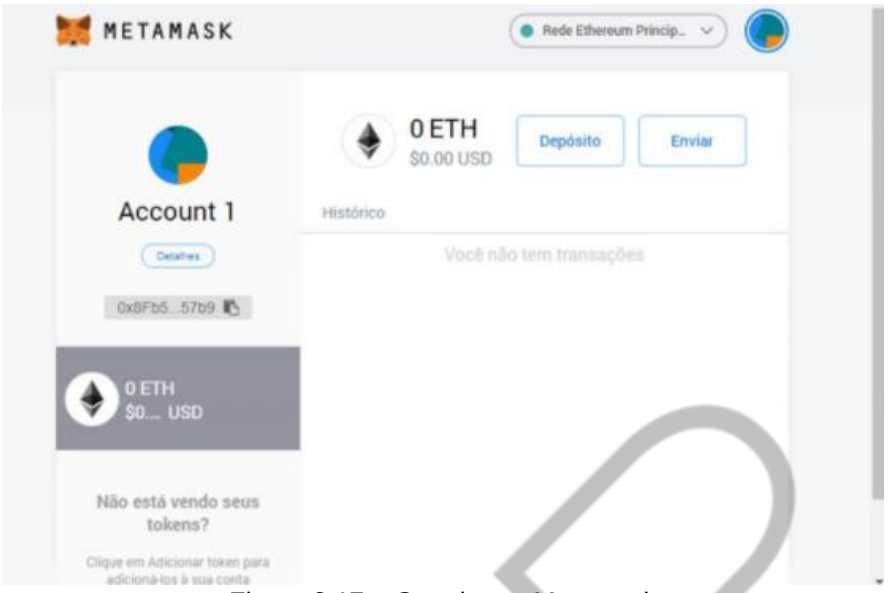


Figura 8.17 – Carteira no Metamask
Fonte: Elaborado pelo autor (2020)

8.5.2. Rede de testes e faucet no Metamask

Para implementar um contrato inteligente em uma rede de testes da plataforma Ethereum (e não gastar nenhum centavo para isso), é preciso apontar a carteira digital para uma rede de testes, como a rede Ropsten. Selecione a rede de testes na caixa de seleção na parte superior da carteira.

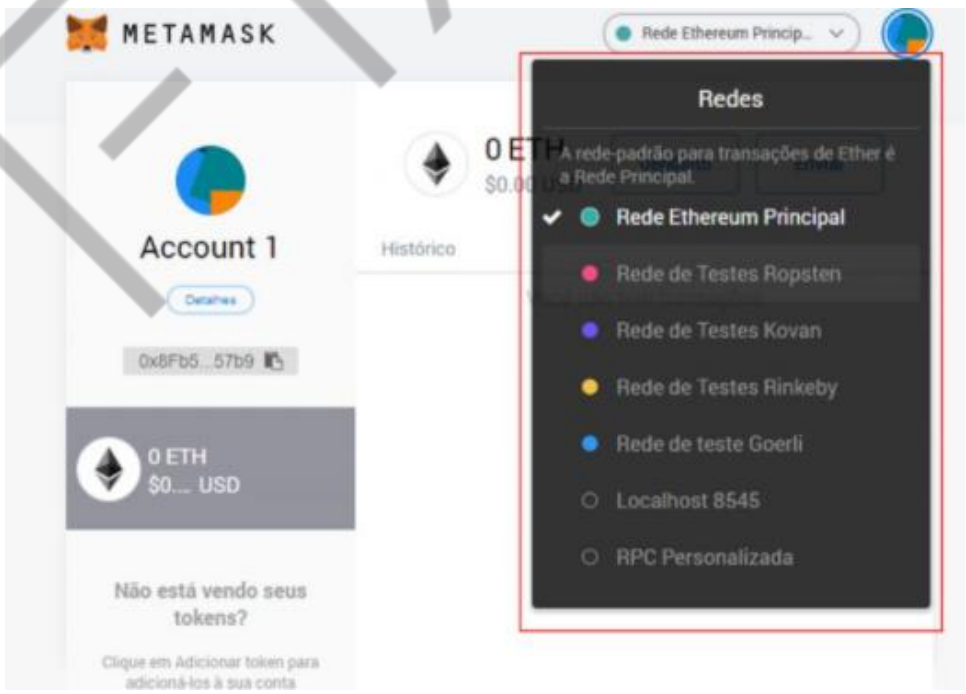


Figura 8.18 – Mudança da rede Ethereum no Metamask
Fonte: Elaborado pelo autor (2020)

Será necessária uma quantia em *ethers* para pagar os custos da publicação do contrato na rede além de processamentos e armazenamentos de dados no contrato. Felizmente, é possível pedir ethers de teste para a rede (chamados de faucets). Para isso, clique no botão “Depósito”, presente na carteira.

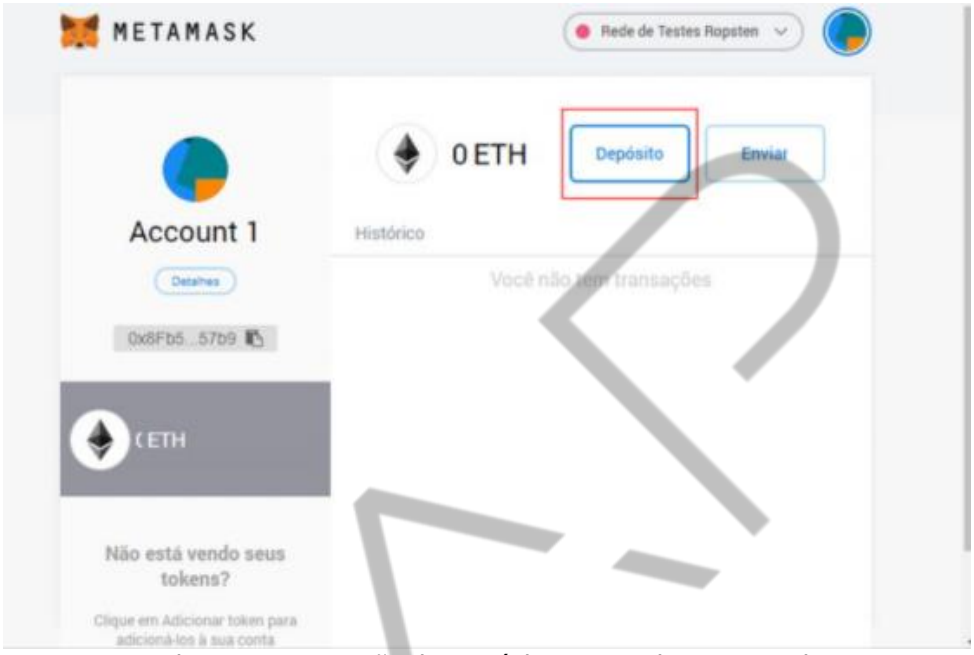


Figura 8.19 – Opção de Depósito na carteira Metamask
Fonte: Elaborado pelo autor (2020)

Na sequência, clique em “Obter Ether” na “Torneira de Testes”.

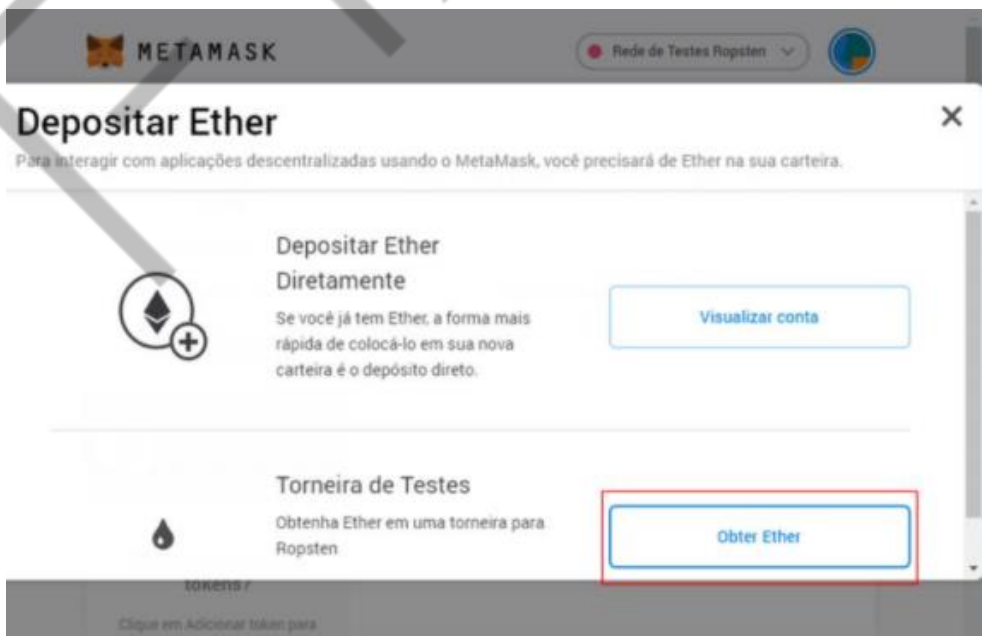


Figura 8.20 – Opção do Torneira de Testes no Metamask
Fonte: Elaborado pelo autor (2020)

Clique no botão “*request 1 ether from faucet*”. O dApp solicita permissão para a interação com a carteira, portanto, na tela que surge clique no botão “Conectar-se”.

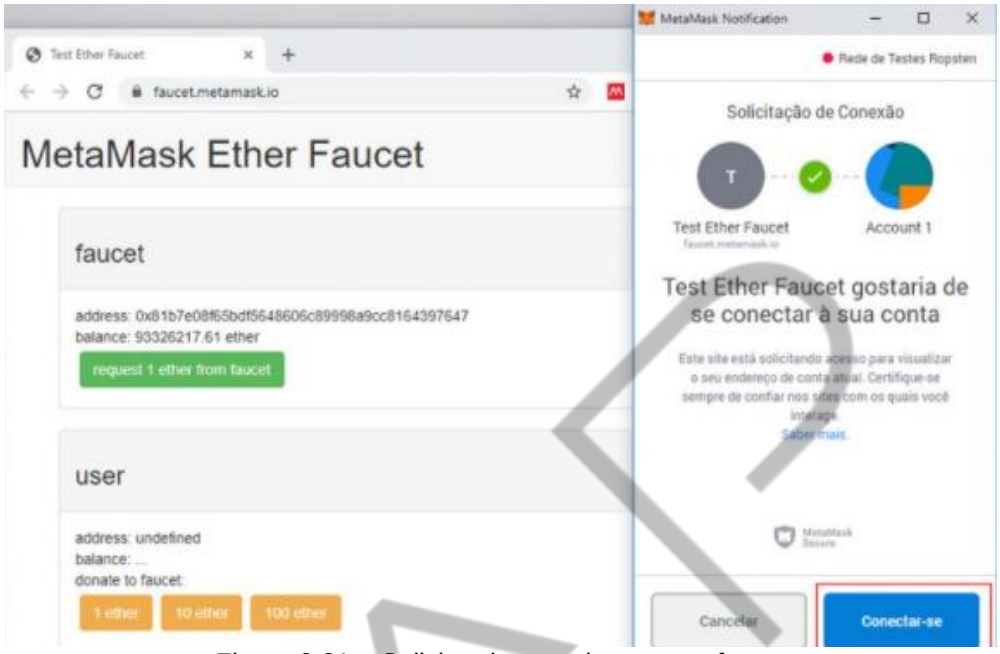


Figura 8.21 – Solicitando um ether para o *faucet*
Fonte: Elaborado pelo autor (2020)

Deve aparece um hash identificando unicamente a transação em “*transactions*”. Caso apareça um erro e, em poucos minutos, não aparecer 1 ether na carteira, recarregue a página e repita a operação clicando em “*request 1 ether from faucet*”.

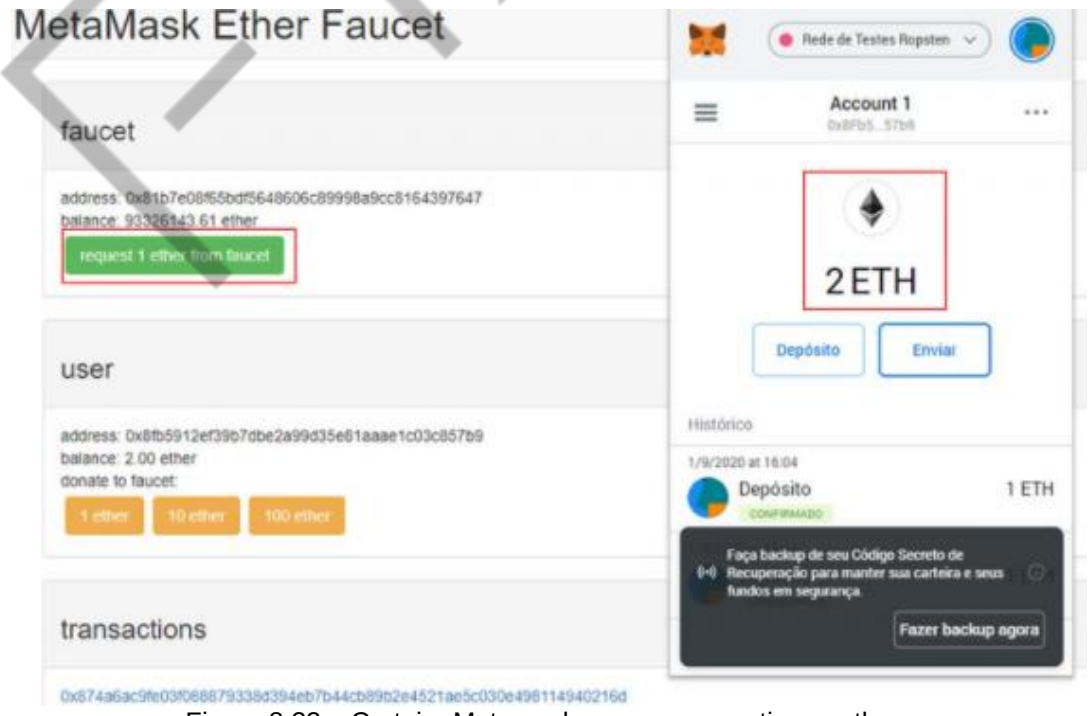


Figura 8.22 – Carteira Metamask com uma quantia em ethers
Fonte: Elaborado pelo autor (2020)

Contudo, talvez a faucet ou “torneira” do metamask dê erro dizendo que há muitas requisições para ele – infelizmente, há algum tempo ele sofre de congestionamento. Se for o caso, tente este outro endereço: <https://faucet.ropsten.be/>.

(Vide Figura “Outro faucet para solicitar ethers”).

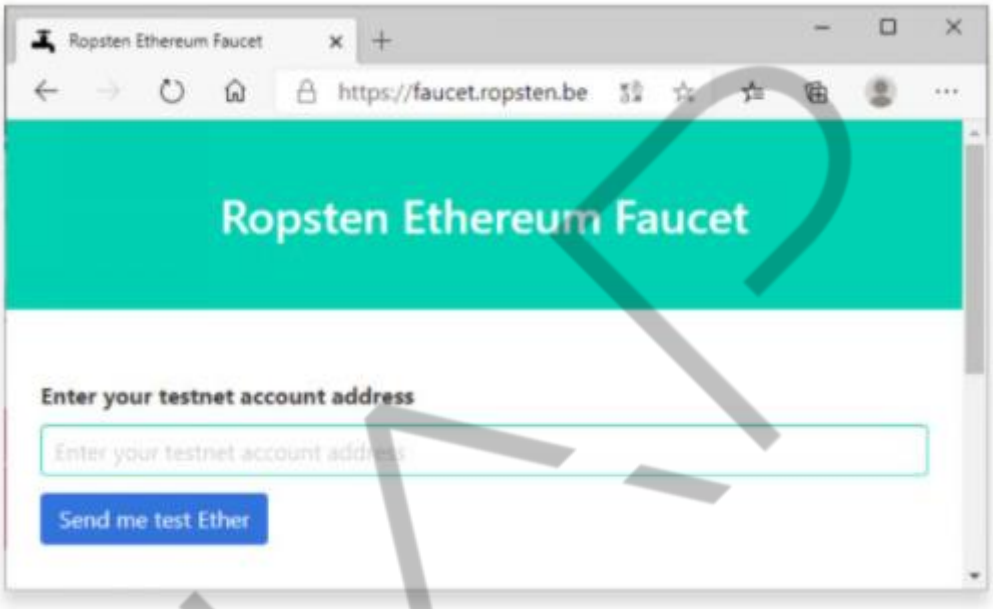


Figura 8.23 – Outro faucet para solicitar ethers
Fonte: Elaborado pelo autor (2020)

8.5.3. Publicação de um *Smart Contract* no Remx

Para implementar um contrato inteligente, a maneira mais fácil é utilizar um ambiente de desenvolvimento totalmente *web-based* disponibilizado pelo projeto Ethereum, o Remix. Para tal, no mesmo Google Chrome por meio do qual a carteira Metamask foi instalada, acesse <https://remix.ethereum.org/>.

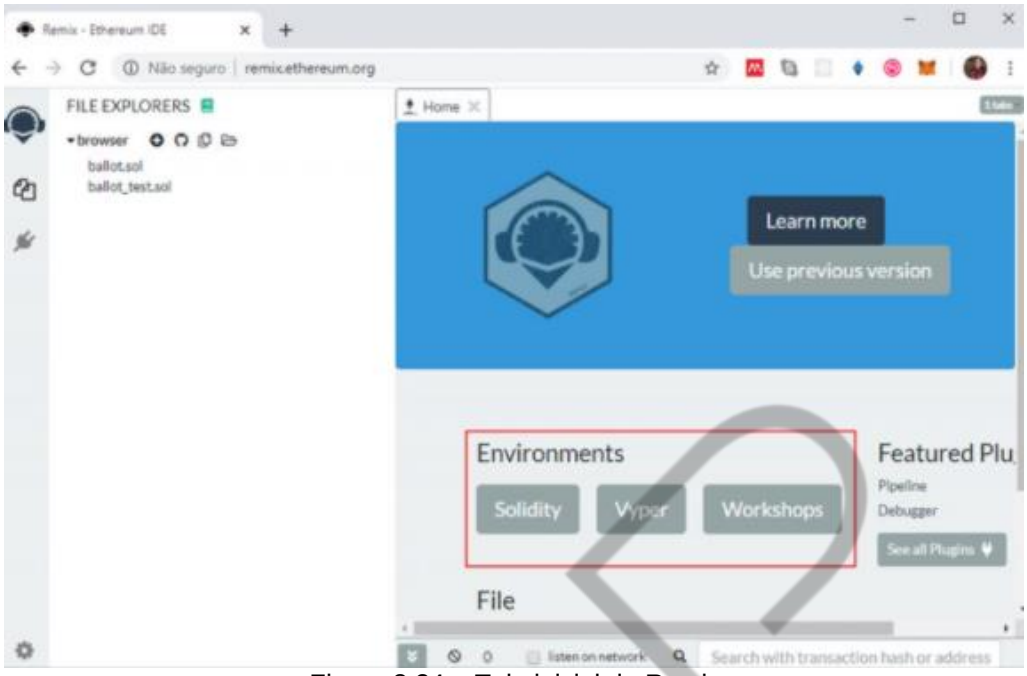


Figura 8.24 – Tela inicial do Remix
Fonte: Elaborado pelo autor (2020)

Repare que, em sua barra lateral esquerda, apenas dois botões de abas aparecem. Para habilitar os demais botões do ambiente, clique no botão “Solitidy” presente na seção “*Environments*”.

Em File Explorers, temos uma estrutura de arquivos com códigos-fonte disponíveis. Por padrão, os arquivos ballot.sol e ballot_test.sol já estão disponíveis e se referem a um exemplo de contrato de votação inteligente, portanto, dê uma olhada posteriormente, quando o Solidity se tornar um pouco mais familiar para você.

Clique no botão “+”, presente ao lado da palavra “browser”, e peça para criar um arquivo chamado “SimpleStorage.sol”. Este exemplo, presente na documentação oficial do Solidity, é uma espécie de exemplo do tipo “Alô mundo”, que pode ser um excelente primeiro contato com a tecnologia.

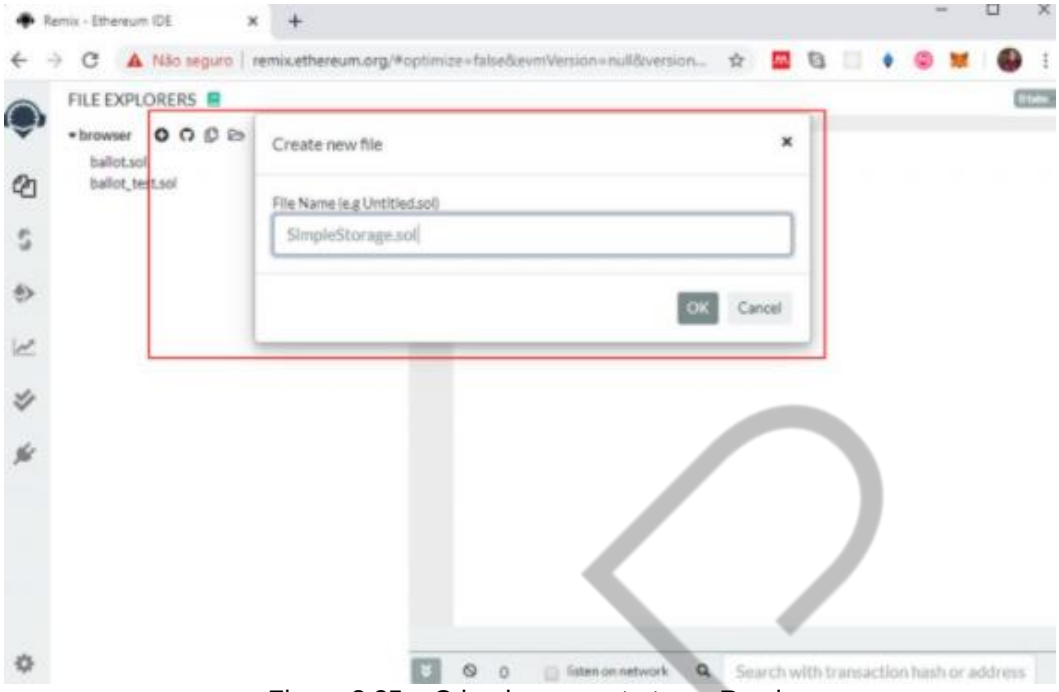


Figura 8.25 – Criando um contrato no Remix
Fonte: Elaborado pelo autor (2020)

Eis o código-fonte em Solidity conforme a documentação oficial. Vamos explicar as nuances desta linguagem, linha a linha.

```
pragma solidity >=0.4.0 <0.7.0;

contract SimpleStorage {
     uint storedData;

     function set(uint x) public {
          storedData = x;
     }

     function get() public view returns (uint) {
          return storedData;
     }
}
```

Código-fonte 8.1 – Primeiro contrato em Solidity
Fonte: Solidity (2020)

A instrução **`pragma solidity`** é obrigatória e serve para informar com qual(is) versão(ões) do compilador solidity o código-fonte é compatível. Repare que, por estar na versão 0.7.0 (abaixo de 1), a linguagem é considerada extremamente instável, com instruções novas surgindo e outras sumindo o tempo todo. Neste caso específico, o código seria compatível com versões maiores ou iguais a 0.4.0 e menores do que

0.7.0. É possível estabelecer uma versão exata, logo, `pragma solidity 0.5.0` significa que o código-fonte exige ser compilado em um compilador versão 0.5.0.

A instrução `contract` seguida pelo nome do contrato (**SimpleStorage**) não é muito diferente de qualquer linguagem orientada a objeto, ou seja, `contract` é equivalente ao `class`, e o objeto é a instância desse contrato, que será gerada no momento da publicação na plataforma Ethereum. As instruções presentes dentro do bloco iniciado pelo símbolo `{` e encerrado pelo `}` fazem parte do contrato em questão.

Na quarta linha, temos `uint storedData` , uma declaração de atributo, em um padrão bem conhecido das linguagens orientadas a objeto: o tipo de dado seguido pelo identificador do atributo. É nesses atributos que as informações do contrato são armazenadas (por isso, o nome sugestivo de *storedData*) e ficam visíveis por todas as operações do contrato. O tipo de dado **uint** é uma abreviação para ***unsigned integer*** ou número inteiro sem sinal, assim sendo, ***storedData*** aceita armazenar um número do conjunto dos números naturais. No Quadro “Tipos de dados disponíveis no Solidity”, são apresentados os tipos de dados disponíveis na linguagem.

Tipo de dado	Descrição
bool	O clássico tipo de dado booleano, aceitando apenas dois estados, `true` ou `false`.
int / uint	A linguagem conta com diversos tipos de dados para guardar números inteiros com ou sem sinal, dos mais diferentes tamanhos. Os tipos de dados int8, int16, int32, int64, int128, int256 guardam números inteiros com a quantidade de bits especificada no tipo de dado, logo, int64 usa 64 bits para guardar números no intervalo de -9.223.372.036.854.775.808 até 9.223.372.036.854.775.807. Com a letra “u” na frente, a variável guarda apenas números positivos e seu intervalor dobra, sendo assim, enquanto int16 trabalha no intervalo -32.768 até 32.767, uint16 guarda números de 0 até 65.535 com a mesma quantidade de bits (16, nestes casos). A ausência do número de bits remete à maior versão disponível, logo, int e uint são abreviações de int256 e uint256, respectivamente.
fixed / ufixed	Destinado para números fracionados (ou com ponto flutuante). Devem ser usadas as palavras-chave **ufixedMxN** ou **fixedMxN**, cujo M representa quantos bits são usados para a variável e o N, quantos casas decimais estão disponíveis para parte fracionada. Assim como no tipo de dado inteiro, M deve ser um número divisível por 8 dentro do intervalo 8 e 256 (bits). N, por sua vez, pode ser um número entre 0 a 80, incluindo ambos.

	Como no caso inteiro, os tipos de dados precedidos pela letra “u” armazenam apenas números positivos, dobrando o intervalo de número positivos armazenados pelas versões sem o “u”. **ufixed** and **fixed** são abreviações dos tipos **ufixed128x18** and **fixed128x18**, respectivamente. Atenção: estes tipos de dados ainda não estão implementados, embora apareçam na documentação oficial.
address	Este tipo de dado foi criado especificamente para guardar endereços de carteiras ou contratos na plataforma Ethereum, logo, **address**, guarda o valor de 20 bits, tamanho destes endereços. Existe uma variação deste tipo de variável, o **address payable**, que habilita as operações transfer e send do endereço ali armazenado. A distinção é a seguinte: **address payable** é um endereço por meio do qual é possível enviar Ethers, enquanto ao armazenado em **address** isso não é possível.
bytes	Utilizados para uma sequência de bytes de tamanho fixo, ideal para alfanuméricos. Estão disponíveis **bytes1** até **bytes32** e bytes é uma abreviação de bytes1.
String	Armazena alfanuméricos de tamanho variável no padrão UTF-8.

Enum	A forma mais simples de criar uma lista de valores pré-definidos no Solidity, segue exemplo de declaração:
	
	contract Produto {
	  enum EstadoOpcoes { Novo, Seminovo, Usado }
	  EstadoOpcoes estado = EstadoOpcoes.Novo;
	}

Quadro 8.1 – Tipos de dados disponíveis no Solidity
Fonte: Elaborado pelo autor (2020)

Voltando ao nosso código-fonte do contrato **SimpleStorage**, este possui dois métodos (ou operações) chamados de **set** e **get**, sendo **set** responsável por armazenar um número em **storedData** e **get** em exibí-lo. Repare que ambos possuem a palavra **public**, que define a visibilidade das funções, que podem ser chamadas livremente. O quadro “Tipos de visibilidade disponíveis no Solidity” apresenta este e outros tipos de visibilidade.

Tipo de visibilidade	Descrição
public	Um atributo ou método marcado como public pode ser acessado livremente, seja pelo próprio contrato, seus herdeiros (interno) ou outros contratos e dApps (externo).
private	Para atributos e métodos que serão acessados exclusivamente pelo contrato que as possui.
internal	Para atributos e métodos que serão acessados apenas internamente (seja pelo contrato que as possui ou por seus filhos, que as herdam).
external	Este tipo de visibilidade é exclusivo para métodos e, por essa razão, não está disponível para atributos. As funções definidas como **external** são

	consideradas parte da interface do contrato e serão chamadas externamente em transações e não podem ser chamadas internamente (ou seja, em um função external f, a chamada f() não funciona, mas this.f() sim).

Quadro 8.2 – Tipos de visibilidade disponíveis no Solidity
Fonte: Elaborado pelo autor (2020)

Enquanto o método **set**() recebe um parâmetro x do tipo **uint** e faz uma persistência de dados armazenando o número em storedData, **get**() é um método apenas de visualização e, por essa razão, recebe a palavra reservada **view** indicando isso. O comando **returns** seguido pelo tipo de dado indica o que a função está retornando.

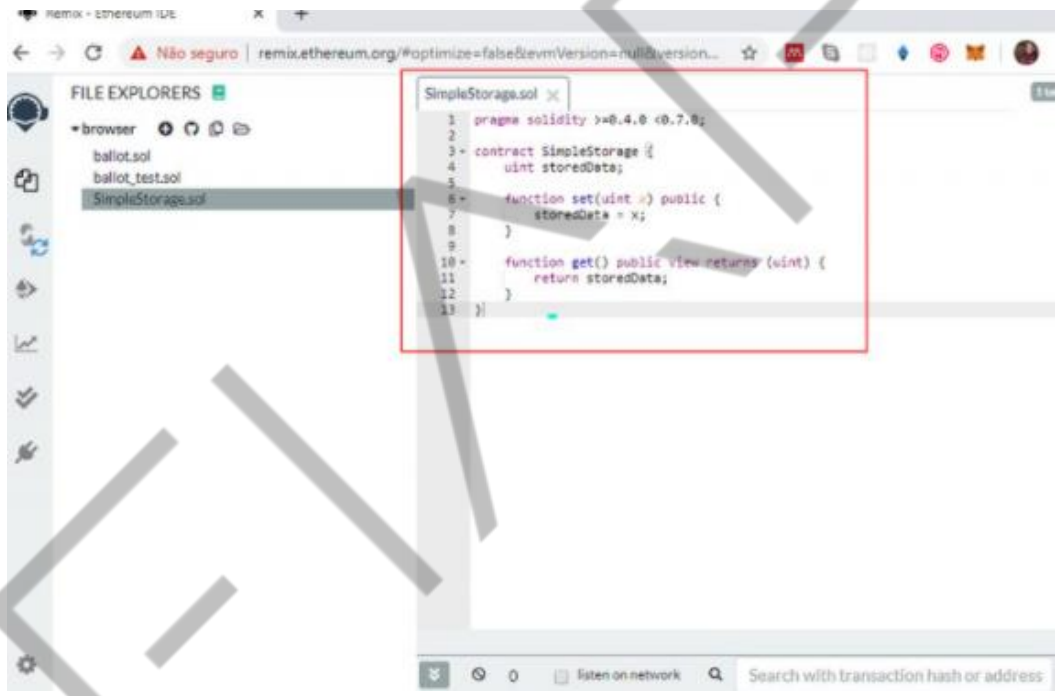


Figura 8.26 – Escrevendo o contrato no Remix em Solidity
Fonte: Elaborado pelo autor (2020)

Antes de mais nada, o contrato precisa ser compilado no compilador Solidity solc. No Remix, clique no segundo botão no menu esquerdo (com o logo do Solidity, um “S”) e selecione a versão do compilador (optamos por utilizar a versão 0.5.16, embora existam versões mais novas!).

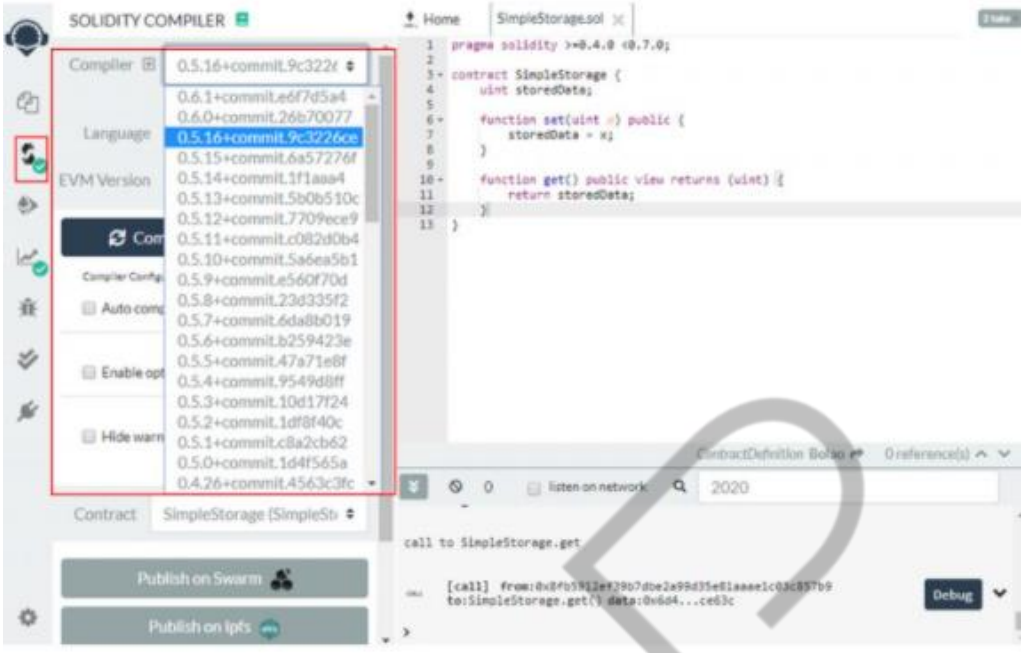


Figura 8.27 – Mudando a versão do compilador Solidity no Remix
Fonte: Elaborado pelo autor (2020)

Clique no botão “Compile SimpleStorage.sol” e o *bytecode* estará pronto.

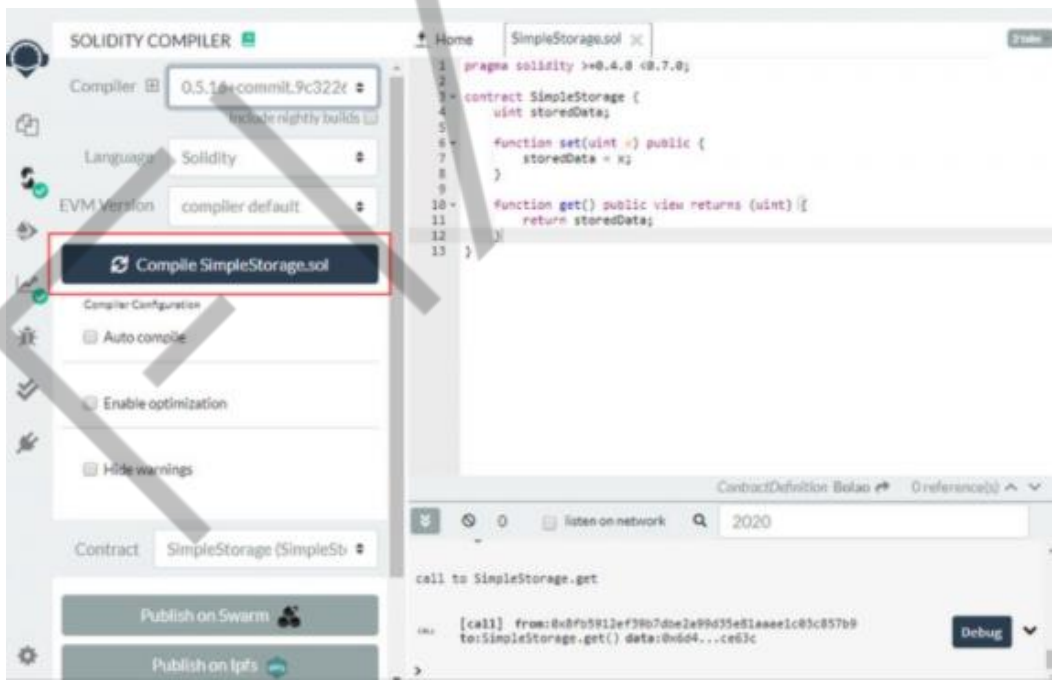


Figura 8.28 – Contrato compilado no Remix
Fonte: Elaborado pelo autor (2020)

No terceiro botão do menu esquerdo (com o logo da plataforma Ethereum), realize a publicação do contrato e, para tal, selecione a opção “Injected Web3” no campo “Environment”.

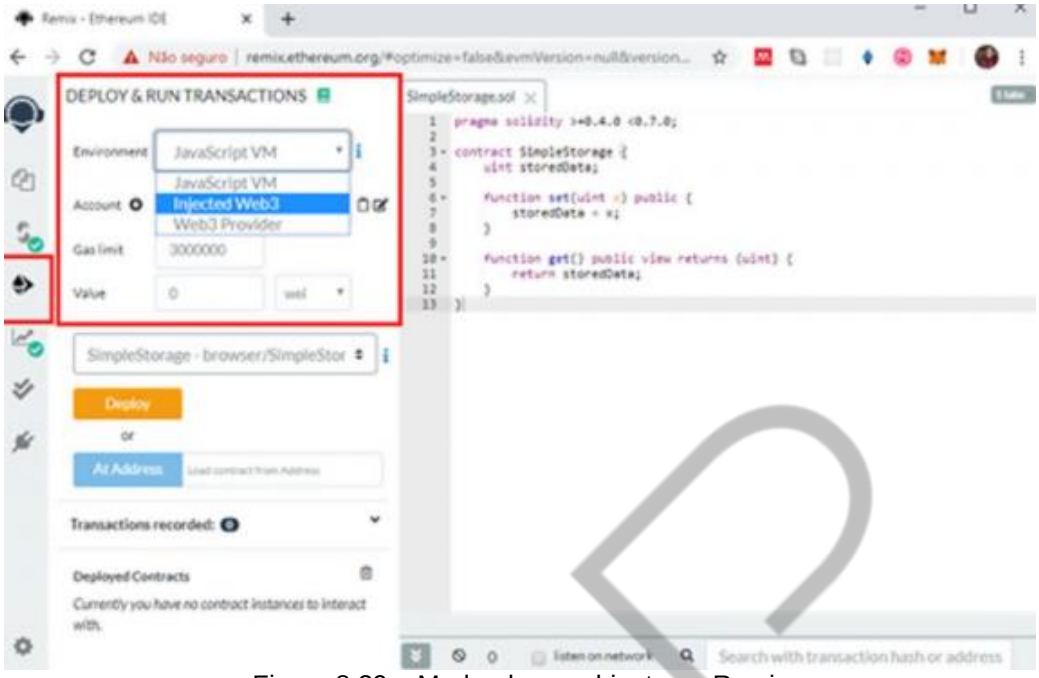


Figura 8.29 – Mudando o ambiente no Remix
Fonte: Elaborado pelo autor (2020)

Ao fazer isso, o Remix solicitará permissão para interagir com sua carteira mantida pelo Metamask. Clique em “Conectar-se” para conceder essa permissão.

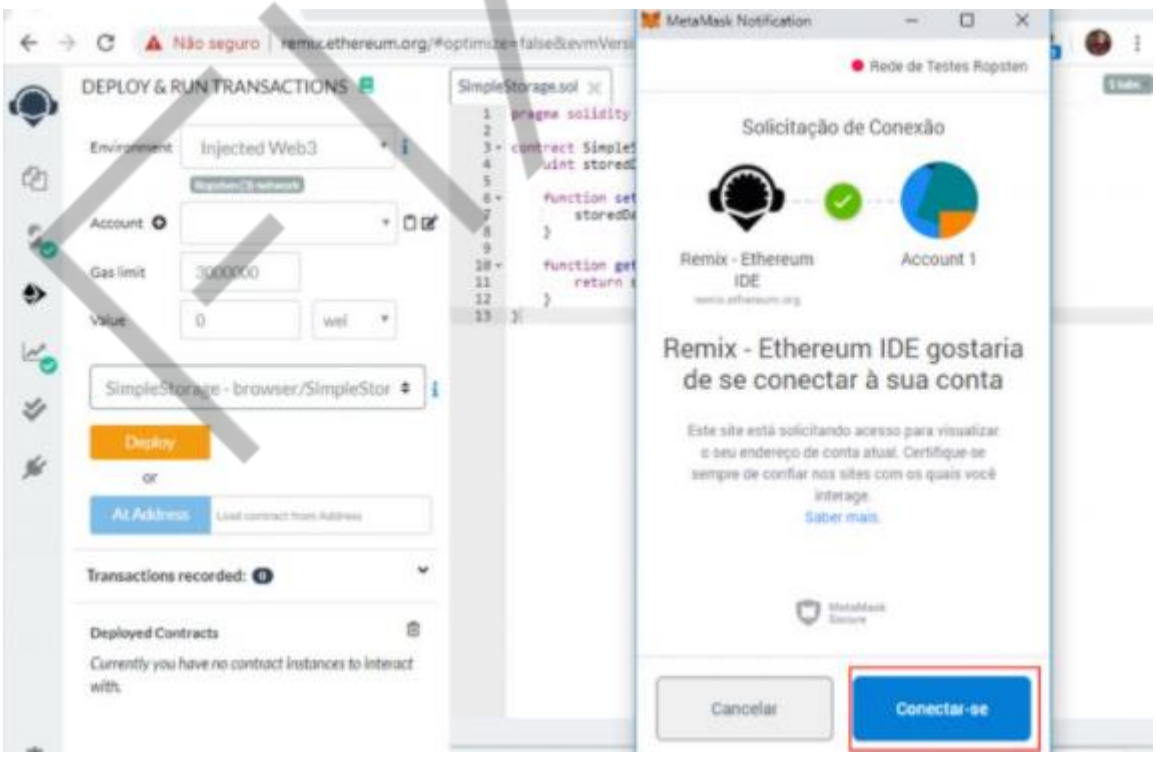


Figura 8.30 – Integração da carteira do Metamask com o Remix
Fonte: Elaborado pelo autor (2020)

Repare que a carteira do Metamask agora aparece no campo “Account”, com seu saldo (por aqui, 2 ethers). Clique no botão “Deploy”, e novamente a carteira do Metamask será acionada, agora em uma transação de “Implantação de Contrato”. Note que uma pequena taxa de rede é cobrada. No exemplo, por aqui, 0,000289 ETH (também conhecido como 289 Szabos ou 289000 GWei). Essa taxa de rede é calculada dinamicamente pelo Metamask, logo, provavelmente você notará uma diferença em sua execução. Note também que a taxa pode ser definida pelo usuário por meio do link “EDIT”. Clique no botão “Confirmar”.

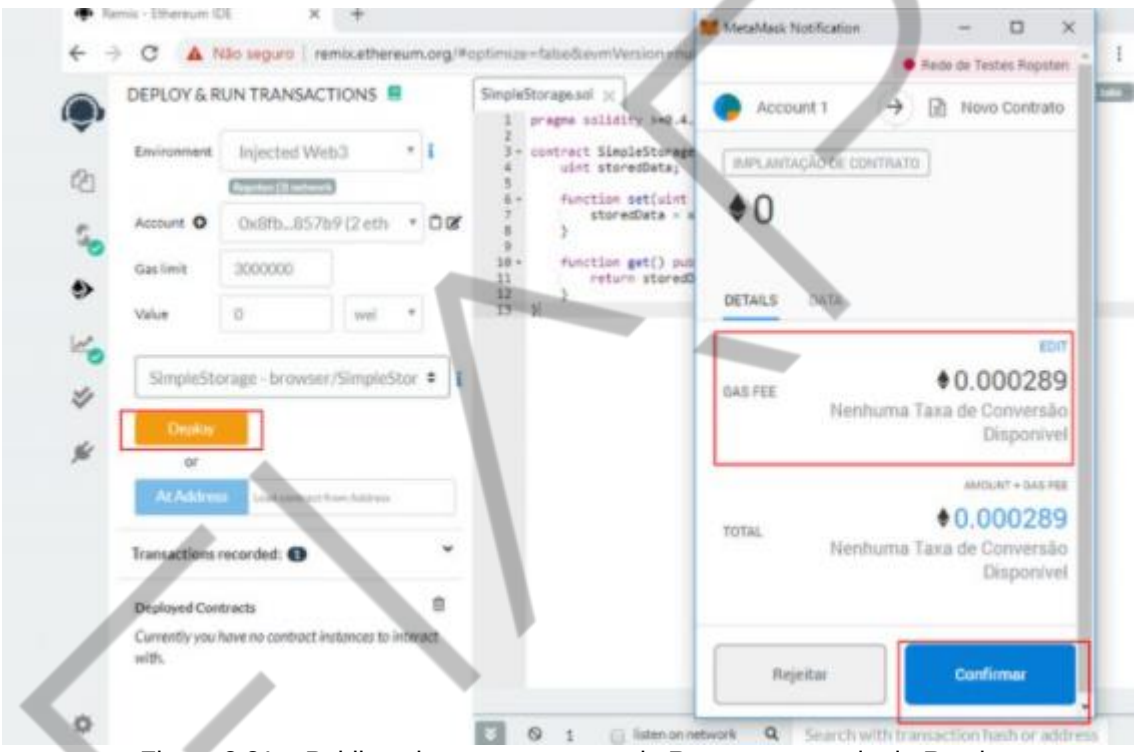


Figura 8.31 – Publicando o contrato na rede Ropsten por meio do Remix
Fonte: Elaborado pelo autor (2020)

O contrato foi publicado na rede Ropsten. Por aqui, o saldo da carteira do Metamask se tornou 1.9997 ETH, assim, a taxa para a publicação do contrato foi realmente paga. Ao ser publicado, uma instância do contrato SimpleStorage se torna ativa e um endereço o identifica unicamente nesta rede (em nosso caso, 0xA67348e15B18CB6155075eF248A8BB0fA1502Dda).

Experimente utilizar o contrato agora, primeiro, passando um número positivo qualquer por meio do método set() e, posteriormente, visualize-o usando o método get(). Os botões tornam-se disponíveis no Remix, conforme a Figura “Testando o contrato publicado na rede Ropsten no Remix”.

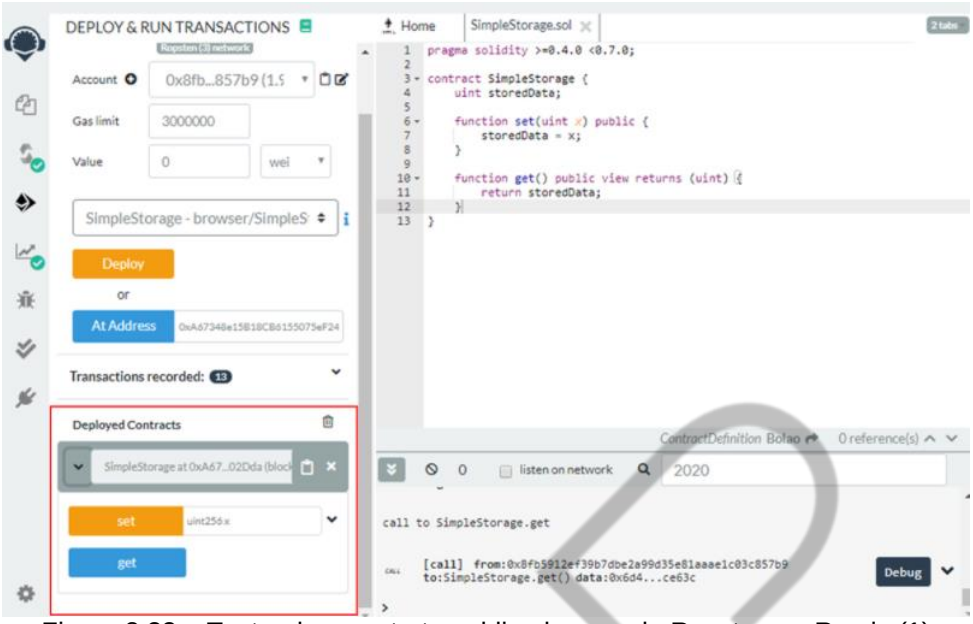


Figura 8.32 – Testando o contrato publicado na rede Ropsten no Remix (1)
Fonte: Elaborado pelo autor (2020)

Note que, ao utilizar o método set() informando um número (em nosso caso, o “150”), a carteira Metamask é novamente chamada e uma taxa será cobrada para rodar o método. Isso acontece porque o método set() faz uma mudança de estado no contrato, mudando um valor (no caso, storedData) que está armazenado nele. Sendo assim, sempre que houver uma mudança de valor ou um grande processamento envolvido, quem chama este método terá que pagar um *gas*, ou seja, um “combustível” para fazer rodar esse contrato. A analogia pode ser feita com um carro mesmo: para fazê-lo rodar, é necessário gastar parte do combustível que está em seu tanque.

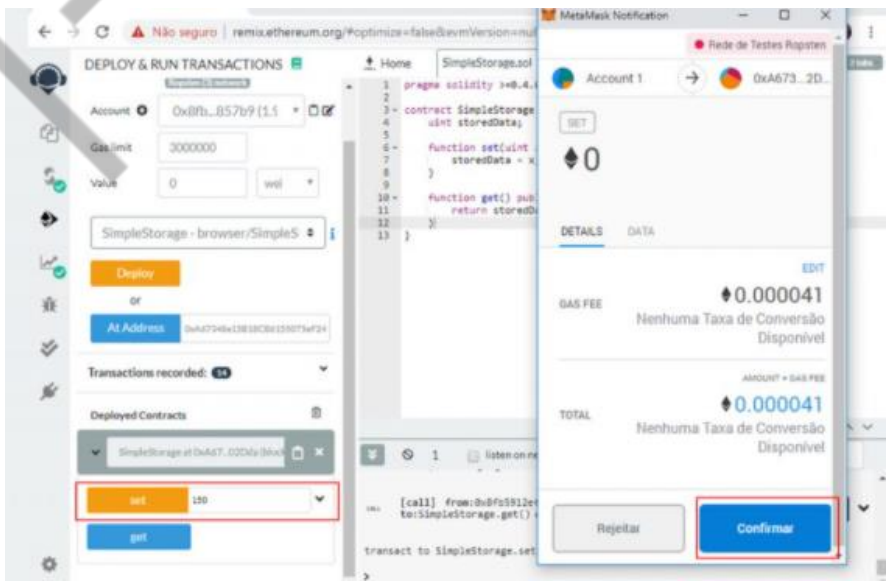
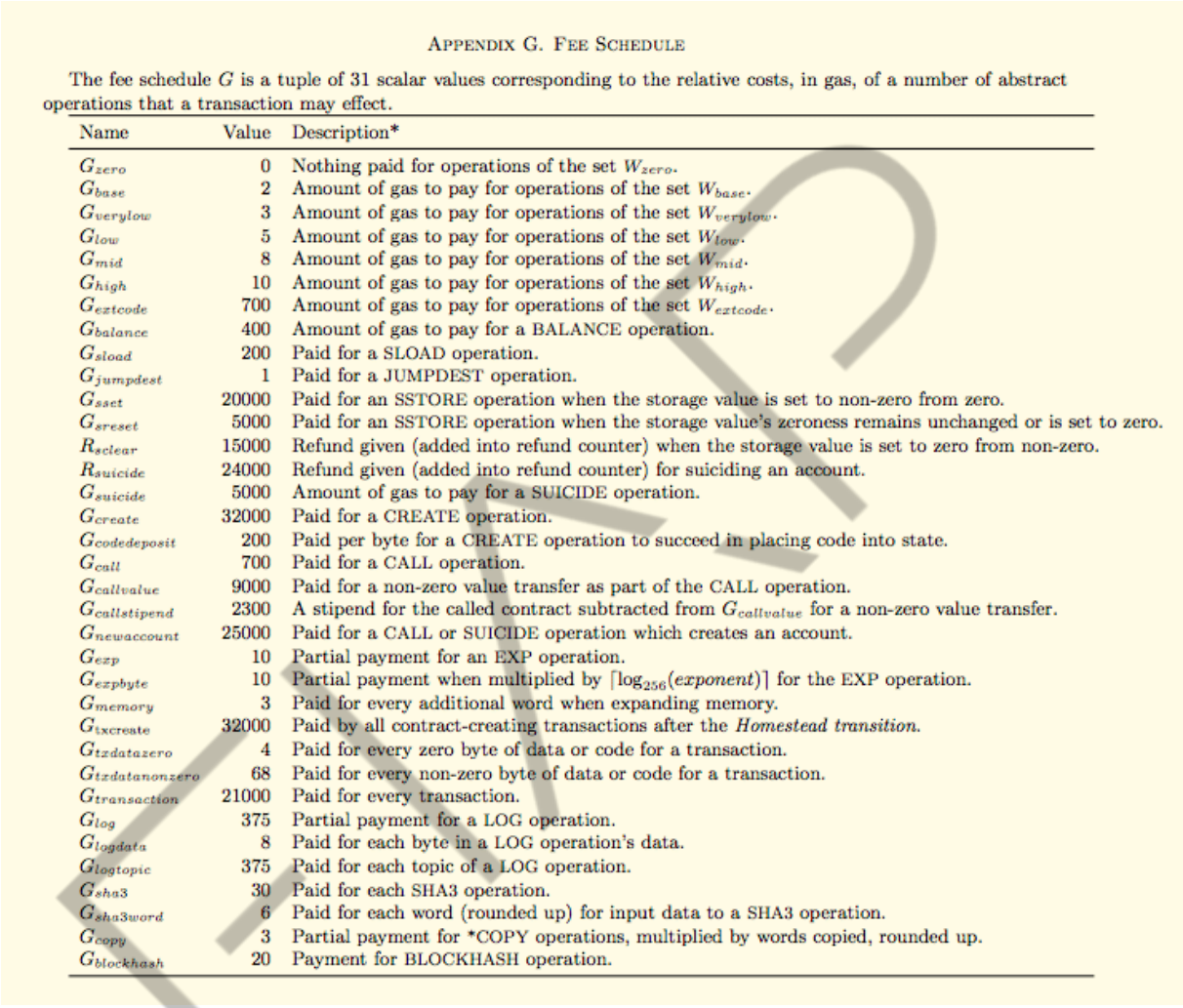


Figura 8.33 – Testando o contrato publicado na rede Ropsten no Remix(2)
Fonte: Elaborado pelo autor (2020)

É possível calcular este "*gas*"? Sim, cada instrução, tamanho de variável a ser armazenada, tudo acaba influindo; para ilustrar, na figura "Tabela para cálculo de *Gas* no Ethereum" Gavin Wood relaciona em sua EIP-150 o custo em gás de cada tipo de instrução.



APPENDIX G. FEE SCHEDULE

The fee schedule G is a tuple of 31 scalar values corresponding to the relative costs, in gas, of a number of abstract operations that a transaction may effect.

Name	Value	Description*
G_{zero}	0	Nothing paid for operations of the set W_{zero}.
G_{base}	2	Amount of gas to pay for operations of the set W_{base}.
$G_{verylow}$	3	Amount of gas to pay for operations of the set $W_{verylow}$.
G_{low}	5	Amount of gas to pay for operations of the set W_{low}.
G_{mid}	8	Amount of gas to pay for operations of the set W_{mid}.
G_{high}	10	Amount of gas to pay for operations of the set W_{high}.
$G_{extcode}$	700	Amount of gas to pay for operations of the set $W_{extcode}$.
$G_{balance}$	400	Amount of gas to pay for a BALANCE operation.
G_{sload}	200	Paid for a SLOAD operation.
$G_{jumpdest}$	1	Paid for a JUMPDEST operation.
G_{sset}	20000	Paid for an SSTORE operation when the storage value is set to non-zero from zero.
G_{sreset}	5000	Paid for an SSTORE operation when the storage value's zeroness remains unchanged or is set to zero.
R_{sclear}	15000	Refund given (added into refund counter) when the storage value is set to zero from non-zero.
$R_{suicide}$	24000	Refund given (added into refund counter) for suiciding an account.
$G_{suicide}$	5000	Amount of gas to pay for a SUICIDE operation.
G_{create}	32000	Paid for a CREATE operation.
$G_{codedeposit}$	200	Paid per byte for a CREATE operation to succeed in placing code into state.
G_{call}	700	Paid for a CALL operation.
$G_{callvalue}$	9000	Paid for a non-zero value transfer as part of the CALL operation.
$G_{callstipend}$	2300	A stipend for the called contract subtracted from $G_{callvalue}$ for a non-zero value transfer.
$G_{newaccount}$	25000	Paid for a CALL or SUICIDE operation which creates an account.
G_{exp}	10	Partial payment for an EXP operation.
$G_{expbyte}$	10	Partial payment when multiplied by $\lceil \log_{256}(exponent) \rceil$ for the EXP operation.
G_{memory}	3	Paid for every additional word when expanding memory.
$G_{txcreate}$	32000	Paid by all contract-creating transactions after the *Homestead transition*.
$G_{txdatazero}$	4	Paid for every zero byte of data or code for a transaction.
$G_{txdatanonzero}$	68	Paid for every non-zero byte of data or code for a transaction.
$G_{transaction}$	21000	Paid for every transaction.
G_{log}	375	Partial payment for a LOG operation.
$G_{logdata}$	8	Paid for each byte in a LOG operation's data.
$G_{logtopic}$	375	Paid for each topic of a LOG operation.
G_{sha3}	30	Paid for each SHA3 operation.
$G_{sha3word}$	6	Paid for each word (rounded up) for input data to a SHA3 operation.
G_{copy}	3	Partial payment for *COPY operations, multiplied by words copied, rounded up.
$G_{blockhash}$	20	Payment for BLOCKHASH operation.

Figura 8.34 – Tabela para cálculo de *Gas* no Ethereum
Fonte: Wood (s.d.)

A chamada de um método que envolva cálculos ou alteração de variáveis, como é o caso do set(), vai custar, no mínimo, 21000 *gas* que, segundo Wood (s.d.) é o preço pago para cada transação. O número em *gas* que representa o custo de rodar um determinado método do contrato deve ser multiplicado pelo preço do *gas*, o chamado ***gas price***. Segundo Rosic (2018), o preço médio do *gas* gira em torno de 20 Gwei (ou 0,00000002 ETH), mas este método pode subir ou descer: se a rede está muito congestionada, o valor médio do *gas* pode subir para 30, 40 Gwei. Note que é possível determinar o *gas price* que você está disposto a pagar: se você tem pressa

e quer ter certeza de que uma determinada transação do contrato vai rodar com prioridade, basta comprar o combustível pelo preço mais caro. Se o *gas price* médio naquele momento for 20 Gwei, pagar 25 ou 30 já será mais do que suficiente para que os mineradores do Ethereum deem preferência para suas transações. Para finalizar, se o método de um *Smart Contract* precisa de 30.000 *gas* para rodar, e o *gas price* médio é 0,00000002 ETH, basta multiplicar um pelo outro e chegamos ao custo de rodar aquele método (neste exemplo, 30.000 gas x 0,00000002 ETH = 0,0006 ETH).

Na demonstração, é possível observar que o *gas price* da rede Ropsten é sensivelmente menor; Mesmo a rede principal, após algumas restruturações oriundas da atualização do software de mineração da plataforma Ethereum (atualmente na versão Istanbul) resultou em um impacto no *gas price* médio que, segundo Etherscan (2020) está em torno de 10 Gwei.

Por outro lado, por ser um método apenas de visualização (está até mesmo marcado com a palavra reservada *view*) o método get() não gera custo, olhar para o contrato é “de graça”. O Remix destaca os métodos de forma diferente em sua interface, ao pintar os botões de chamada em cores diferentes (o set() é laranja enquanto o get() é azul). Bem, clique no botão de chamada do método get() e experimente acessar o dado armazenado no contrato.

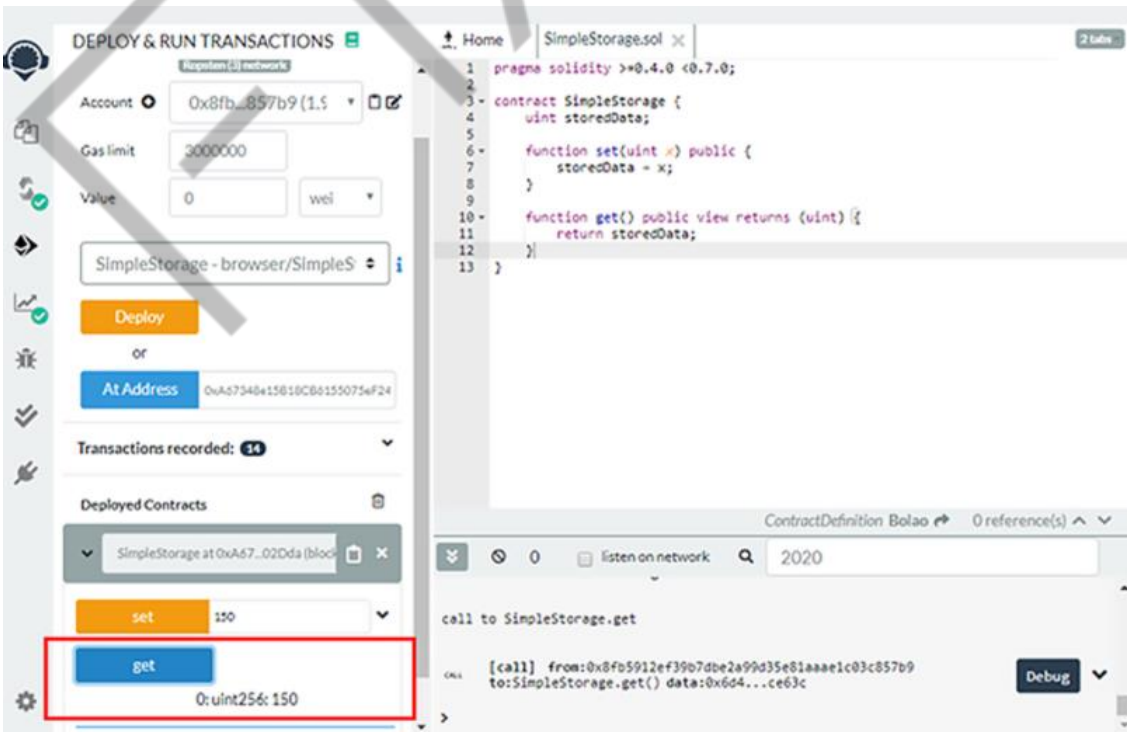


Figura 8.35 – Testando o contrato publicado na rede Ropsten no Remix (3)
Fonte: Elaborado pelo autor (2020)

8.6. Exemplo de contrato mais elaborado

Vamos, agora, escrever um contrato um pouco mais elaborado, de forma a apresentar outros conceitos importantes do Solidity. O exemplo a seguir é um clássico jogo de azar do tipo “Bolão”: criaremos um método para entrar no bolão (pagando um valor, é claro) e, depois de juntar uma quantia interessante, uma figura que chamaremos aqui de gerente “fecha as apostas”, solicitando ao contrato sortear um dos apostadores aleatoriamente, entregando toda a bolada. Eis o código.

```
pragma solidity 0.5.16;


contract Bolao {
    address private gerente;
    address[] private jogadores;
    address payable private vencedor;

    constructor() public {
        gerente = msg.sender;
    }

    function entrar() public payable {
        require(msg.value == .1 ether);
        jogadores.push(msg.sender);
    }

    function descobrirGanhador() public restricted {
        uint index = randomico() % jogadores.length;
        vencedor = address(uint160(jogadores[index]));
        vencedor.transfer(address(this).balance);
        jogadores = new address[](0);
        vencedor = address(uint160(0x0));
    }

    modifier restricted() {
        require(msg.sender == gerente);
        _;
    }

    function getSaldo() public view returns (uint) {
        return uint(address(this).balance);
    }

    function randomico() private view returns (uint) {
        return
uint(keccak256(abi.encodePacked(block.difficulty,        now,
jogadores)));
```

```
        }
}
```

Código-fonte 8.2 – Contrato de Bolão no Solidity
Fonte: Elaborado pelo autor (2020)

Neste contrato, são declarados três atributos: um para guardar o endereço da carteira do gerente, um para guardar endereços dos jogadores que fazem apostas e um para guardar o endereço da carteira do vencedor (e realizar a transferência). Repare que o tipo de dado para jogadores é `address[]`, com colchetes, simbolizando uma estrutura conhecida como array, que permite o armazenamento de múltiplos endereços de carteira (por isso jogador**es**, no plural). O atributo vencedor é do tipo novo, `address payable`, para que a transferência do saldo do bolão seja possível.

O método **construct()**, como o nome sugere, é utilizado no momento da construção e instanciação do contrato, ou seja, é executado uma única vez no momento do seu *deploy*. O termo **msg.sender** sempre trará o endereço da carteira que chamou o método, ou seja, o remetente da transação. Nesse caso, fica estabelecido que quem publicar o contrato, é seu gerente (nada mais justo).

O "coração" do contrato é o método **entrar()**; observe que ele é do tipo **payable**, ou seja, além dos gastos com o ***gas***, um valor deve ser mandado nessa transação (o valor em **wei, a décima-oitava casa do ether**, pode ser observado em **msg.value**). Para estabelecer certa justiça nas apostas, uma função require() é utilizada para verificar se o valor enviado é de 0,1 ETHER, nem mais, nem menos. Dessa forma, entrar possui um ticket de valor fixo. A remoção dessa linha permitiria enviar absolutamente qualquer valor para o bolão.

O método `descobrirGanhador()` é exclusivo do gerente (observe que ao declarar o método temos a palavra `restricted`, e a regra de restrição é implementada a seguir, por uma estrutura chamada `modifier`, que verifica se `msg.sender == gerente`). O procedimento é usar um método `randomico()`, implementado mais ao final do contrato, que sorteia o número baseado no número de jogadores, chamando uma função de transferência (`transfer()`), transferindo todo o saldo do contrato (`address(this).balance`). Os demais procedimentos "zeram" os atributos de jogadores e vencedor para a "brincadeira" começar novamente.

Para testar o contrato, alguns cuidados devem ser tomados: em primeiro lugar, é necessário criar mais carteiras no Metamask, para que tenhamos múltiplos papéis neste contrato. Para tal, basta clicar no ícone redondo na raposa presente no canto superior direito do Google Chrome, e selecionar “Criar Conta”.

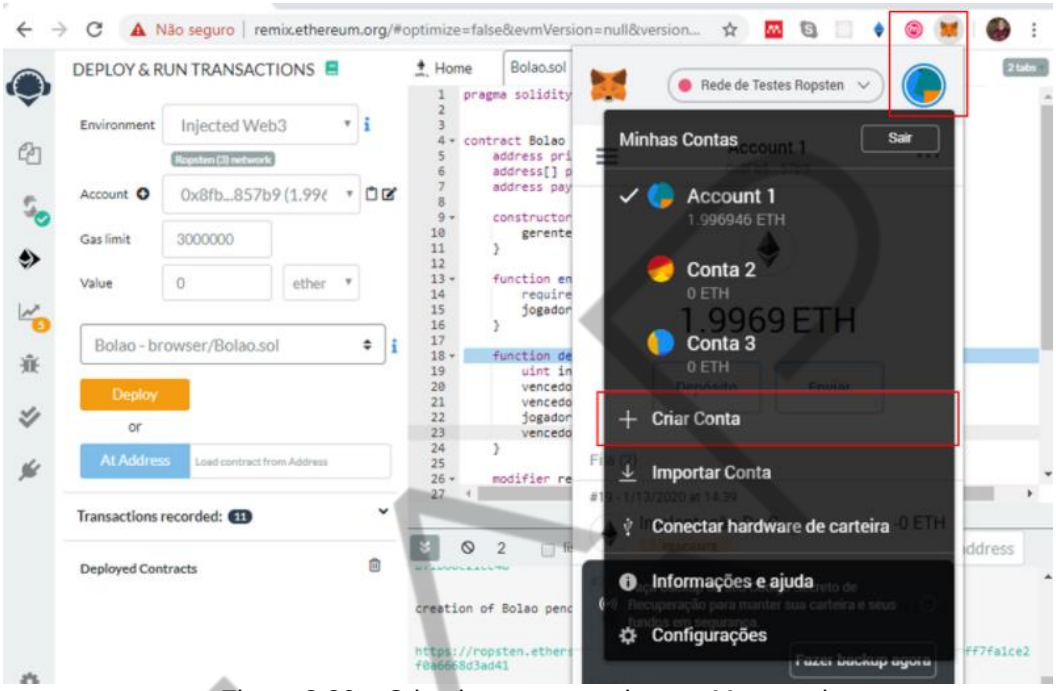


Figura 8.36 – Criando novas carteiras no Metamask
Fonte: Elaborado pelo autor (2020)

É aconselhável fazer o procedimento duas vezes, assim, é possível separar bem os papéis: a primeira carteira publica o contrato e fica como gerente, e as contas dois e três são os jogadores. É necessário transferir um valor para essas novas carteiras. Para ter acesso ao endereço da carteira nova, selecione a nova carteira no ícone redondo e clique em cima do endereço que aparece de forma reduzida. Pronto, o endereço da carteira foi copiado para a área de transferência.

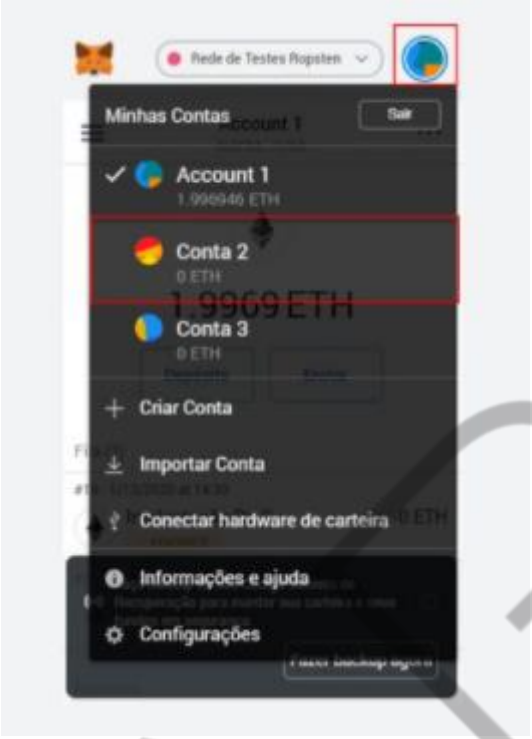


Figura 8.37 – Criando novas carteiras no Metamask (1)
Fonte: Elaborado pelo autor (2020)

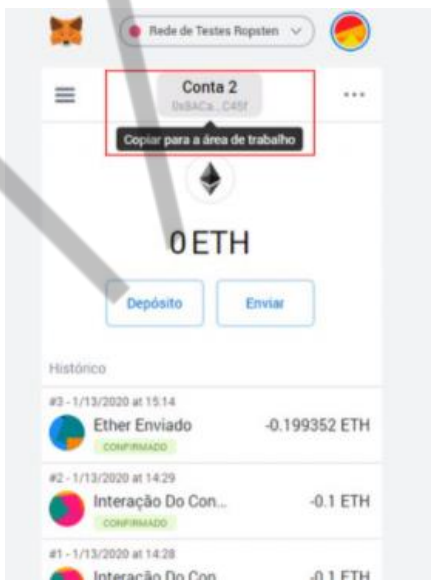


Figura 8.38 – Criando novas carteiras no Metamask (2)
Fonte: Elaborado pelo autor (2020)

Depois de publicado o contrato pela primeira carteira, selecione a segunda carteira no Metamask e, antes de clicar no botão “entrar” (em vermelho, já que é um método *payable*), coloque o valor a ser enviado pela transação, 0.1, e a grandeza em ethers (o padrão é wei). Somente aí o botão de entrar pode ser clicado e, após a devida confirmação, este se torna um apostador.

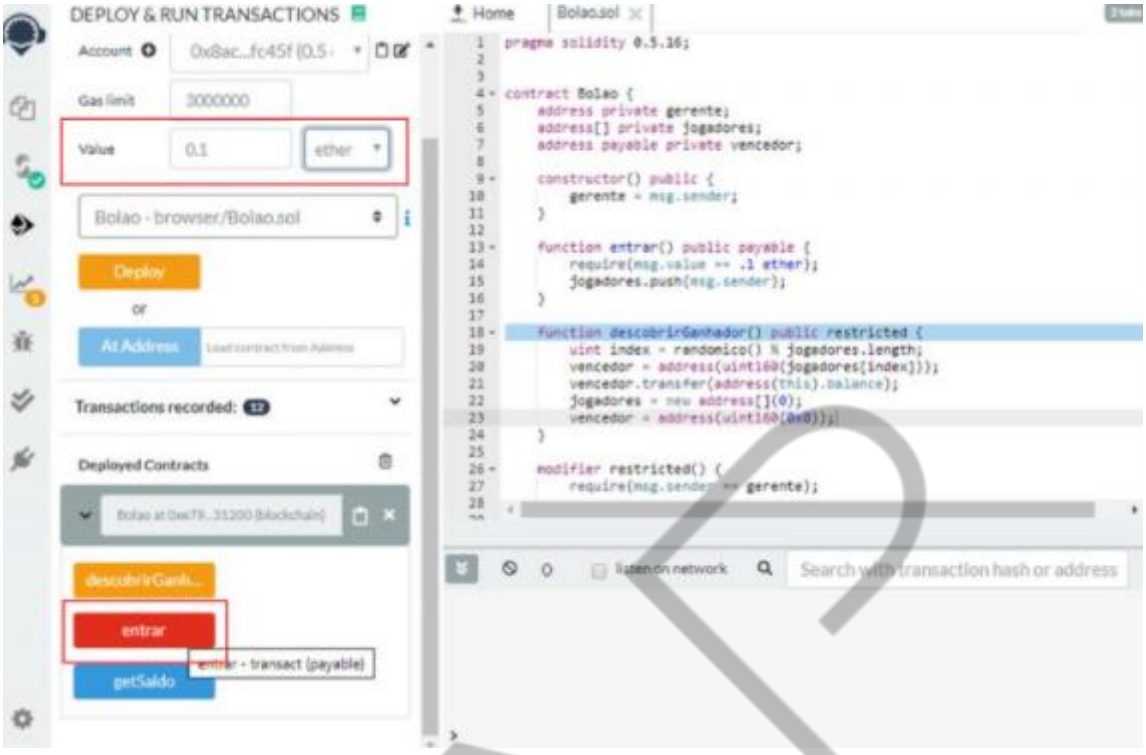


Figura 8.39 – Entrando no bolão
Fonte: Elaborado pelo autor (2020)

O botão getSaldo exibe a quantia que o contrato está custodiando (não estranhe o valor, ele será informado em wei!). Note que, clicar no botão de entrar informando um valor diferente de 0,1 ETH ou clicar no botão “descobrirGanhador” com uma carteira que não seja do gerente resultará em erros, o que significa que as regras de restrição do contrato estão sendo devidamente seguidas.

8.7. Outras opções de ambiente de desenvolvimento solidity

O desenvolvimento de *Smart Contracts* em Remix, embora viável, não é a solução mais robusta para grandes projetos. Trata-se de um contato inicial sem a necessidade de grandes instalações e estruturas, mas pode deixar a desejar conforme os projetos se tornarem mais ambiciosos.

Uma das possibilidades é um *framework* conhecido como **Truffle**. Além do compilador ***Solidity Compiler***, vários scripts de automação, bibliotecas e outras facilidades são disponibilizadas por ele.

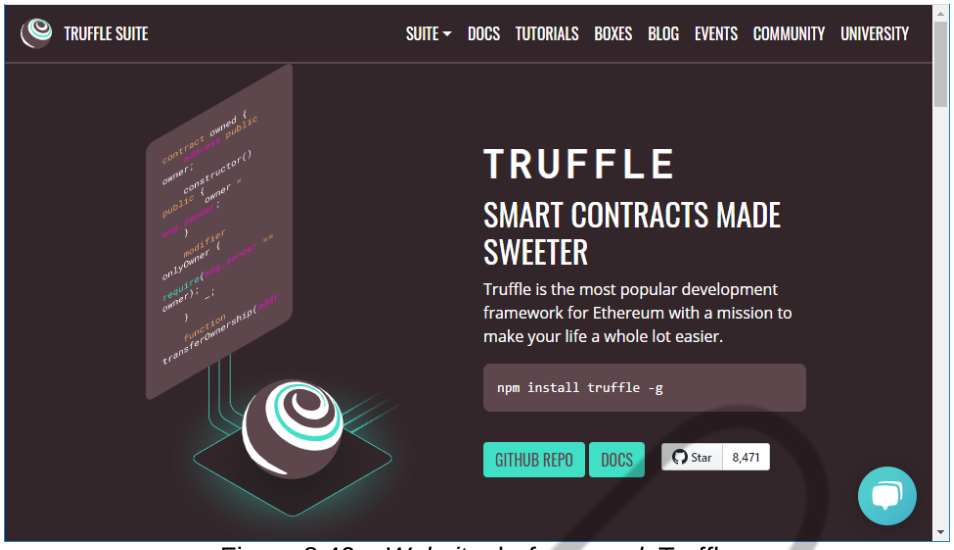


Figura 8.40 – *Website* do *framework* Truffle
Fonte: Elaborado pelo autor (2020)

Para instalá-lo, é necessária a instalação prévia do Nodejs (www.nodejs.org). Somente aí, o npm (*node package manager*) se torna disponível na linha de comando do Windows (em algumas distribuições Linux, contudo, o npm é uma instalação à parte) e o comando mostrado na Figura "*Website* do *framework* Truffle" pode ser usado para instalar o *framework*.

Realizar todos os testes em redes como a Ropsten, Rinkeby e Kovan podem se tornar um processo lento e sem o controle necessário. Uma boa opção é instalar o Ganache, o chamado "*blockchain* em um clique". Fácil de instalar e rodar, permite subir uma plataforma Ethereum local, mais rápida e com fartos recursos em Ether (nada de ficar pedindo na torneira de fauctes).

Figura 8.41 – Criando novas carteiras no Metamask (3)
Fonte: Elaborado pelo autor (2020)

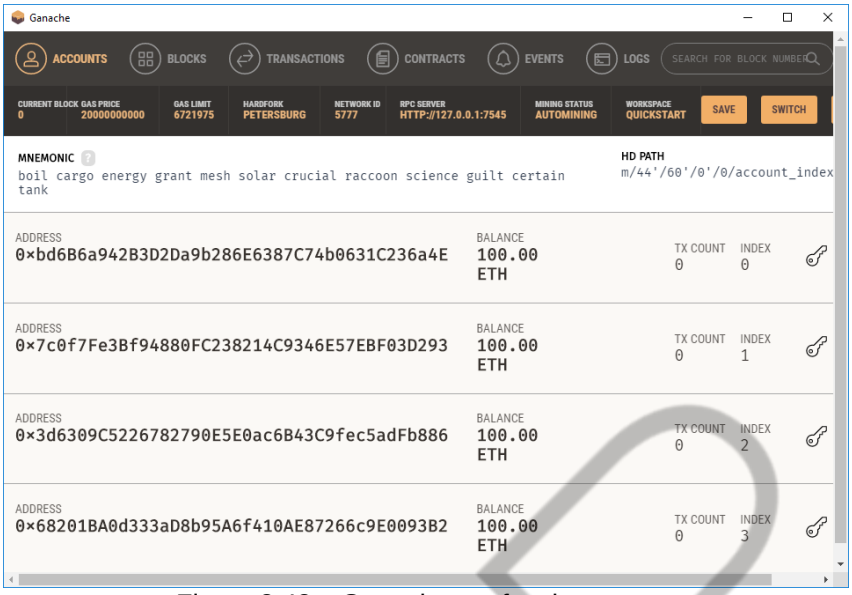


Figura 8.42 – Ganache em funcionamento
Fonte: Elaborado pelo autor (2020)

Como pode ser visto nas abas da Figura “Ganache em funcionamento” é possível visualizar informações dos contratos publicados, verificar cada transação da plataforma, entre outros.

O Metamask possui uma opção para configurar uma rede personalizada, conforme pode ser visto na figura “Configurando o Metamask para uma rede personalizada”.

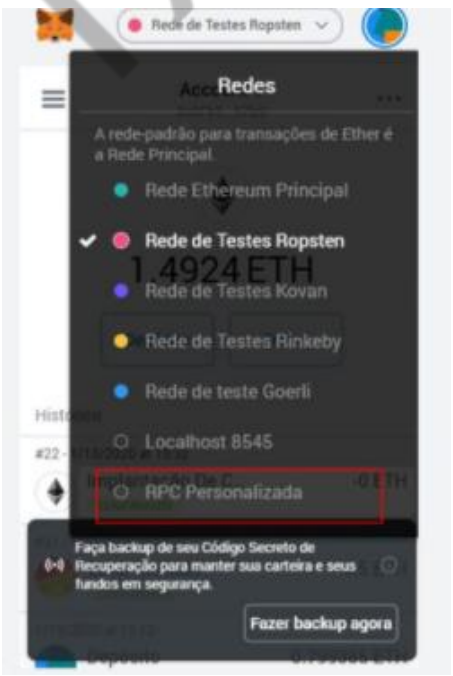


Figura 8.43 – Configurando o Metamask para uma rede personalizada
Fonte: Elaborado pelo autor (2020)

CONCLUSÃO

A tecnologia dos ***Smart contracts*** abre uma gama gigantesca de possibilidades de automação e negócio, e este capítulo não tinha a pretensão de formar desenvolvedores de contratos inteligentes em **Solidity**, mas de ser um pontapé inicial, respondendo às perguntas mais básicas desse procedimento técnico. Esperamos que este capítulo tenha aguçado seu interesse pelo assunto e que seja uma das alternativas de especialização no futuro.

REFERÊNCIAS

ARCADE.CITY. **Arcade City WebSite.** 2018. Disponível em: <http://arcade.city/>. Acesso em: 11 jun. 2018.

BLOCKCHAIN “Smart Contracts” Will Revolutionize Voting in Elections. 2018. Disponível em: <https://medium.com/@spireProtocol/blockchain-smart-contracts-will-revolutionize-voting-in-elections-acb5abc7beb6>. Acesso em: 17 set. 2020.

BLOCKCHAIN: Honduras 1-France 0? 2015. Disponível em: <https://blockchainfrance.net/2015/09/16/le-honduras-adopte-la-blockchain/>. Acesso em: 11 dez. 2019.

CHANDRAN, R. **Modernizing land records in Honduras can help stem violence, says analyst.** 2017. Disponível em: <https://www.reuters.com/article/us-honduras-landrights-tech/modernizing-land-records-in-honduras-can-help-stem-violence-says-analyst-idUSKBN1AR151>. Acesso em: 11 dez. 2019.

COINDESK. **Ethereum 101.** 2019. Disponível em: <https://www.coindesk.com/learn/ethereum-101/what-is-ethereum>. Acesso em: 9 jan. 2020.

DAVID, C. **Começou agora a venda e a distribuição para a comunidade do Arcade Token ($ARCD).** 2017. Disponível em: <https://blog.arcade.city/come%C3%A7ou-agora-a-venda-e-a-distribui%C3%A7%C3%A3o-para-a-comunidade-do-arcade-token-arcd-eadd9b8f2430>. Acesso em: 17 set. 2020.

EICHMANN, K. **Machine** Economy – a decentralized future that is enabled by autonomous machine-to-Machine transactions! 2018. Disponível em: <https://medium.com/innogy-innovation-hub/machine-economy-a-decentralized-future-that-is-enabled-by-autonomous-machine-to-machine-e497b90f13c1>. Acesso em: 10 jun. 2018.

EIGHT, E**. Spotify CEO Daniel Ek Talks Convincing Taylor Swift to Re-Join Service.** 2018. Disponível em: <https://www.rollingstone.com/music/news/spotify-ceo-talks-convincing-taylor-swift-to-re-join-service-w518672>. Acesso em: 17 set. 2020.

GIBBS, S. **Uber plans to buy 24,000 autonomous Volvo SUVs in race for driverless future.** 2017. Disponível em: <https://www.theguardian.com/technology/2017/nov/20/uber-volvo-suv-self-driving-future-business-ride-hailing-lyft-waymo>. Acesso em: 11 jun. 2018.

GULKER, M. **Are Smart Contracts the Future of Fraud Prevention?** 2017. Disponível em: <https://www.aier.org/article/are-smart-contracts-future-fraud-prevention>. Acesso em: 17 set. 2020.

JOHNSON, L.; FITZSIMMONS, M. **Uber self-driving cars**: **everything you need to know.** 2018. Disponível em: <https://www.techradar.com/news/uber-self-driving-cars>. Acesso em: 17 set. 2020.

KO, T. **A guide to developing an Ethereum decentralized voting application.** 2018. Disponível em: <https://medium.freecodecamp.org/developing-an-ethereum-decentralized-voting-application-a99de24992d9>. Acesso em: 17 set. 2020.

LEFF, A. **Costa Rica: "For Sale," or just a scam?** 2011. Disponível em: <https://www.pri.org/stories/2010-11-23/costa-rica-sale-or-just-scam>. Acesso em: 11 dez. 2019.

MCCORRY, P.; et al. **A Smart Contract for Boardroom Voting with Maximum Voter Privacy.** 2017. Disponível em: < https://eprint.iacr.org/2017/110.pdf>. Acesso em: 17 set. 2020.

MCINTYRE, H. Why Did Taylor Swift Really Rejoin Spotify? **Forbes**, 2017. Disponível em: <https://www.forbes.com/sites/hughmcintyre/2017/06/27/why-did-taylor-swift-really-rejoin-spotify/#7fc83189373d>. Acesso em: 17 set. 2020.

MELO, L. **Após aporte de US$ 2,1 bi, Uber já vale mais que Ford ou GM.** 2016. Disponível em: < https://exame.com/negocios/com-aporte-de-us-2-1-bi-uber-ja-vale-mais-que-ford-ou-gm/#:~:text=Assim%2C%20o%20Uber%20valeria%20mais,%2C2%20bilh%C3%B5es%20de%20d%C3%B3lares).>. Acesso em: 17 set. 2020.

MOLECKE, R. **How To Learn Solidity: The Ultimate Ethereum Coding Tutorial.** 2017. Disponível em: <https://blockgeeks.com/guides/solidity/>. Acesso em: 9 jan. 2020.

OPENBAZAAR. **OpenBazaar Website.** 2018. Disponível em: <https://www.openbazaar.org>. Acesso em: 11 jun. 2018.

RASKIN, M. **The Law and Legality of Smart Contracts.** 2017. Disponível em: <https://www.georgetownlawtechreview.org/the-law-and-legality-of-smart-contracts/GLTR-04-2017/>. Acesso em: 17 set. 2020.

ROSIC, A. **What is Ethereum Gas? [The Most Comprehensive Step-By-Step Guide Ever!** 2018. Disponível em: < https://blockgeeks.com/guides/ethereum-gas/>. Acesso em: 13 jan. 2020.

SOLIDITY. **Solidity v0.7.0.** 2020. Disponível em: <https://solidity.readthedocs.io/en/v0.7.0/>. Acesso em: 17 ago. 2020.

SZABO, N. **The Idea of Smart Contracts.** 1997. Disponível em: < https://nakamotoinstitute.org/the-idea-of-smart-contracts/ >. Acesso em: 13 jan. 2020.

TAYEB, H. **'A big step in the right direction': Settlemint's founder and CEO on blockchain e-voting.** 2016. Disponível em: <https://www.tearsheet.co/funding/a-big-step-in-the-right-direction-settlemints-founder-and-ceo-on-blockchain-e-voting>. Acesso em: 17 set. 2020.

WISTROM, B. **Scrutinized by Governments, Austin's Arcade City Expands Uber Alternative Ride-Hailing Model.** 2017. Disponível em: <https://www.americaninno.com/austin/scrutinized-by-governments-austins-arcade-city-expands-uber-alternative-ride-hailing-model/>. Acesso em: 17 set. 2020.

WOOD, G. **Ethereum: a secure decentralised generalised transaction ledger eip-150 revision**. Disponível em: <http://paper.gavwood.com/>. Acesso em: 13 jan. 2020.