Københavns Universitet

Department of Computer Science

WWF: GoBisonGo!

A blockchain-based charity betting platform

Authors:
Alexander Steen Andersen (ldp372)
Baider Diwan (gbc132)
Dan Graakjær Kristensen (nbs151)
David Aamand Møller (npt476)
Lasse Halberg Haarbye (lpt113)
Sille Lie Barrett (klf846)

Course Coordinators:
Boris Düdder
Fritz Henglein
Omry Ross

Subjects in Blockchain Technology
August 27, 2018
No. pages: 24

Abstract

This report is about a blockchain-based charity betting platform to fund WWF's mission to preserve wildlife.

GoBisonGo! is a dapp (decentralized application), where users can place bets on bison. The winning bison is the one that moves the longest distance each day. The winning betters split the pot and WWF gets 5 percent. This way the users can enjoy gambling, support bison wild life and WWF can get funding at the same time.

Technical aspects: The bison movement is tracked trough IoT devices. The money transactions is handled through blockchain, more specifically the Ethereum platform. Blockchain is chosed beacause of transparency, security and demonstrating the funding impacts through transactions.

Contributions

Alexander: Front-backend integration and unit testing.
Baider: Business case.
Dan: Front-backend integration and deployment.
David: Angular and solidity coding.
Lasse: Solidity coding and architecture.
Sille: Frontend and business case.

Table of contents

1 Introduction

One of the major concerns globally is wildlife preservation, since biodiversity has never been more threatened than it is today. With growing exploitation of nature, only a few places in Europe have remained with its nature preserved in its wildest state. These places are not only habitats for endangered species, but they also play a crucial role in the wellbeing of the natural environment and therefore provides a vital service to our society.

World Wildlife Fund (WWF) is the worlds leader in wildlife conservation. Their mission is to conserve nature and accommodate the most pressing threats to the diversity of life on earth[1]. One of their focus areas in Europe is the Carpathian Mountains, which has one third of Europe's plant species and mineral waters. This also makes it one of Europe's most important biodiversity reservoirs. The Carpathian Mountains is home for the largest concentration of brown bears, grey wolves, Eurasian lynx and, since recently, the European bison. The named species have been chosen as primary indicators in regard to the health of the ecosystem.

The European bison plays a vital role in the ecosystem. If a reservoir has a sufficient number of bison, then the animals are able to maintain the landscape and cause other natural processes in the food chain. Their nomadic nature makes them travel huge distances, which can create natural passages for other species, favor genetic flows, help seed dispersal, or act as prey for the large carnivores. Therefore one of WWF's major concerns is the health and preservation of the European bison in Eastern Europe.[2]

WWF Romania is looking for new ways to carry out their mission of environment preservation through the European bison. While they do so they face multiple challenges both specifically to their cause, but also problems concerning non-governmental organizations (NGO's) in general, where technology and innovative thinking might be a solution.

Technologies such as blockchain might hold multiple solutions for WWF. Despite of its young age, the cryptocurrencies have on multiple occasions made consumers care about charities or virtual animals such as CryptoKitties, which is a game to trade and breed kitties.

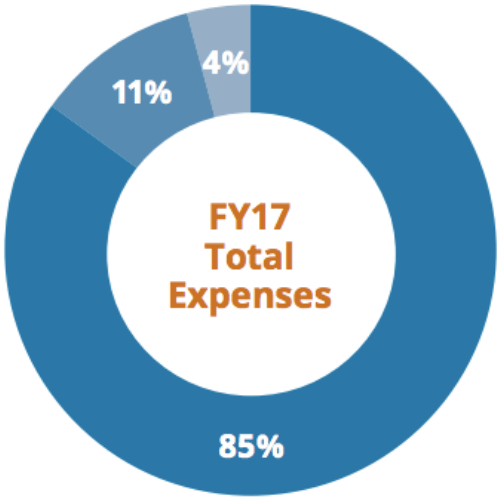
1.1 WWF Romania goals, challenges and report scope

As an NGO, WWF Romania has multiple goals they need to fulfill in order to serve their mission properly. While there are multiple problem statements that might be relevant to achieve these goals, this report will focus on attracting funding. The funding in itself is a costly operation, as we can see in figure 1 which shows the spend distribution for WWF in 2017.

From figure 1 it is seen that WWF spent $320 million in 2017, where $35 million or 15% of WWF's total expenses was solely on fund-raising, finance and administration [3]. The latest data from WWF Romania is 2016, where they spent $1.5 million [3], while there's no data available on the fund-raising.

Charity Navigator, an NGO dedicated to assess the validity of NGO's, estimates that 19,9% of WWF's expenses came from fund-raising expenses[4][1]

Sites such as Charity Navigator strive to differentiate the "good" charities from the "bad" charities. Whether a charity is one or the other is determined by two parameters: their financial health, measured through their expenses and fundraising efficiency, and their transparency and accountability, measured through their governance structure, policies and information available about their operations. WWF has received 2 out of 4 stars in their ability to raise money efficiently.



Programs	$270,757,409	85%
Fundraising	35,115,104	11%
Finance and administration	14,400,291	4%

Figure 1: WWF spend, WWF Annual Report 2017

Therefore the challenges WWF faces as an organization in order to receive charity have been identified as the following: raising funds, show donation transparency and demonstrate impact of donation.[5]

- **Public interest and engagement** In order for people to donate money to charity, the organization has to clearly communicate their goals as well as document the impact of the money. Apart from this, we aim to create a public interest in the project by rewarding the users. In this way the users will be able to win money when they donate. The idea is to attract more donors and thereby more money for the project.
- **Transparency** When people donate money to an environmental charity it can be difficult to track where the money goes or how it is spent. Bureaucracy, corruption and inefficiency are common in the charity space.[5]
- **Documentation of impact** In a world of complexity it can be difficult for individuals to see the direct effects of their actions on a cause. The incentives for acting in an environmental sustainable way aren't always clear, especially in the short run.

By identifying the goals and the challenges this makes the scope of a possible solution easier. This has led us to the following problem statement:

[1]According to Generally Accepted Accounting Principles (GAAP), NGO's are allowed to report some of their specific joint costs from combined educational campaigns and fund-raising solicitations as program costs[6]

How can WWF Romania raise more money and awareness for their cause, while also handling their main challenges through blockchain?

This report proposes a blockchain-based charity betting platform to fund WWF's mission to preserve wildlife, while tackling some of the issues NGO's and charities face today. This is in order to support the highly impactful project of preserving wildlife.

We aim to reach the goal by enabling funding in terms of rewarding the donor with the opportunity to win money. Success of the solution for WWF can be measured through the amount of donations and the number of active users. If both criteria are met, WWF should in the long term see an improvement in the European bison status.

1.2 Proposed solution

We propose a decentralized betting platform, GoBisonGo!, where users can bet on accomplishments of the bison such as distance traveled and max altitude. We intend to fulfill WWF Romania's mission by implementing a blockchain-based betting game with the use of bison data provided by WWF.

Each bison will be wearing an IoT tracking device, which enables the possibility to track their exact position in real-time, giving users the opportunity to get to know their bison's habits a bit more, whereafter they can place a bet on a particular bison.

GoBisonGo! - A blockchain charity betting platform GoBisonGo! is the worlds first socially conscious charitable betting platform, allowing users to make money while helping the Romanian biodiversity. The platform will be implemented both as an app and a site, while this report focuses on the site implementation.
The game is as follows; users can bet fixed amount of money on a specific bison, before 10 AM each day. Once the bets are closed, the money is locked in a smart contract. At 8 PM, the same day, the smart-contract is executed and the winner is announced. Users will receive a notification if they have won.
The users that have bet on the winning bison, get the pot of money except of 5%, which goes to WWF to support the bison project. If several users have chosen the same bison, they will split the prize. If no user has bet on the winning bison, all the money goes to WWF. It will be possible for the user to constantly track the bison on a map and see how far it has moved.

The solution seeks to help raise money for the project as well as to inform the users on the impact of their charity on the European bison. The solution can be split into two parts; the real world implementation of tracking and the building of the platform.

The value for the end-user offered by the proposed platform is an opportunity to win money in a conscious way. Betting and gambling is sometimes viewed as an irresponsible activity, but GoBisonGo! offers an alternative where you help nature by betting.
Betting on animals typically involves bad conditions for the animals, e.g. bull fighting or horse racing. People for the Ethical Treatment of Animals (PETA) consider both as animal cruelty[7][8].

GoBisonGo! is the exact opposite, the more you play, the more you help the animals you bet on.

Furthermore, the user will never be in a position where they get no value at all. If they win they both help a good cause and win money. If they lose they will help the cause. The underlying assumption here is that giving to charity is in itself rewarding [9].
Betting on the European bison will presumably cause the user to be more engaged to the animals and their habitats, and through that the cause. It will also most likely help the cause to have its own platform, to communicate news and create awareness.

The obvious value for WWF is of course the funding, but there might be more than that. It was established that WWF Globally spends 11-19% of their budget annually on fund-raising, estimation varying on accounting method [1] [4]. It is also established that WWF Romania spends around $1.5 million annually, based on 2016 report.[3]. If the solution is implemented, the fund-raising expenses will only be the cost of maintaining the platform. We estimate that the cost of doing so would be lower than 11-19% of $1.5 million($165-$285 K), thus being a cheaper fund-raising method than the WWF Global standard.

One of the goals for WWF Romania is measuring and tracking the health of the bison and the surrounding nature. Implementing the platform would help greatly on this goal, since they could track the animals. Tracking the bison through IoT data, while also knowing their habits could give WWF a good indicator of how the animals and the habitats in general are doing.

Most of the betting platform will be built on the distributed blockchain system, which gives WWF new possibilities in terms of facing their general challenges, but also those related more specifically to betting platforms. This report will focus on building an early prototype for the platform.

Why blockchain?

A blockchain is a distributed system of ledgers(blocks) linked together through cryptography(chain). The technology holds multiple interesting properties, such as decentralization, authentication and immutability [10]. Bitcoin was the first system built using a blockchain, using the technology to build a decentralized electronic currency. Miners in the network validated new transactions by trying to solve a difficult cryptographic puzzle.
Multiple platforms have been built on top of the blockchain, in some cases allowing smart contracts. These have been characterized by some as blockchain 2.0 [11]. A smart contract is a contract implemented, verified and enforced by code. Smart contracts allow agreements to be executed efficient without the need of centralized agents or any third parties.

This particular solution will be implemented in Ethereum. Implementing the solution in blockchain can help a great deal in handling the aforementioned challenges that WWF faces in terms of getting the public to donate money, more specifically:

- **Transparency** Blockchain technology can ensure that money intended to be a reward for conservation or a specific cause does not disappear into unintended pocket through bureaucratic labyrinths. Blockchain money could even be released automatically to the right parties for meeting specific environmental targets. The autonomy of banks makes it easier to trans-

fer money directly to the charity target without going through centralized institutions or a complex web of middlemen.

- **Demonstrating impact** Blockchain technology can help both individuals and companies to achieve information about the the real impact of their actions by tracking transaction within the company. This might encourage them to take actions that benefit the environment. In our case it could be used to track the wellbeing of the Bison as well as how the donations have helped achieve the goals from WWF. More specifically WWF can clearly demonstrate the flow of money in by looking at the public transactions going in and out of the smart contract.

Since the betting is digital, but the creatures that are bet upon are located in the physical world, moving about in a remote location in the Carpathian mountains, it is very important to demonstrate transparency. Otherwise users will be more easily suspicious if they feel that they have no idea of what is going on. If all the data or at least some of it was stored in the blockchain it would help to ease potential suspicion of fraud. The above benefit help government, partners and beneficiaries to gain insight on each transaction. This can build a bigger trust between WWF and their stakeholders.

2 GoBisonGo! - Implemetation

2.1 System architecture

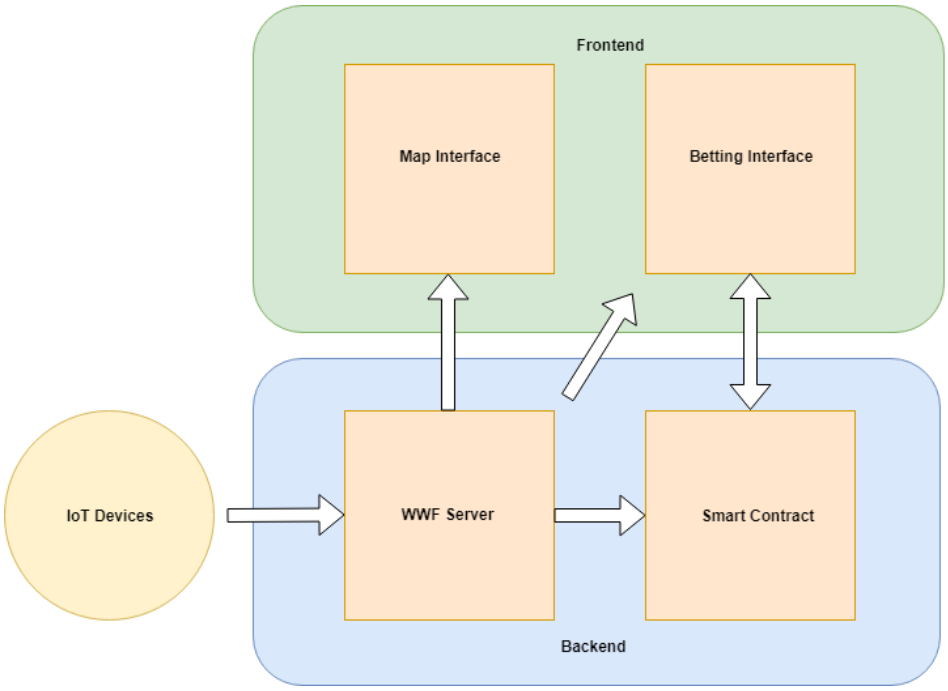


Figure 2: Illustration of the platform's data flow

The GoBisonGo! platform will be composed of multiple parts since the combination of IoT-data, blockchain and betting requires a special setup. Figure 2 illustrates the different components of the system architecture.
The figure above shows the flow of data that starts in the Bison IoT data, where it will feed into the map interface and the WWF processing which will adjust the data to be ready for deployment to the blockchain. The blockchain will store the smart contracts that connect with the betting interface while also keeping track of the data that comes from the WWF processing step. More specifically the process will look like this:

1. **Bison IoT data** Each of the bison will be wearing a tracker, allowing WWF to track its location, altitude and basic health parameters such as heartbeat. This device will be streaming live.

2. (a) **Map Interface** The map interface consists of a live view of the bison location for the users to see. This gives the users a better overview of the possible routes and terrains for the bison. Furthermore, the users are able to constantly see if their bison is in the lead.

   (b) **WWF Processing** Since the IoT devices will stream for 100+ bison many times a minute, the magnitude of the data is not fit for the blockchain. Storing it directly into the blockchain under the current circumstances would make it unnecessary costly and perhaps even impossible. Therefore we suggest a server of some sorts to process the data,

so only the most relevant information is stored in the blockchain. See 3.1.5 for further discussion on this topic.

3. **Blockchain** The blockchain will get pushed the processed IoT data from the bison, while the smart contracts will be stored on the blockchain and read the processed data. When the results are in from the daily race the contract will transfer money either to both WWF and the winner(s) or WWF alone.

4. **Betting interface** In the betting interface the users can place their bets. The available bison to bet on are listed in the interface. When the user places the bet, he or she sends money to the smart contract. Users will receive a notification regarding whether they have won or not.

2.1.1 Front-end

The front-end of the solution will, as mentioned earlier, consist of two parts: the live map and the betting interface. The front-end is implemented in Angular 6 which is a framework for building web applications. Figure 3 illustrates a screenshot from the map interface.

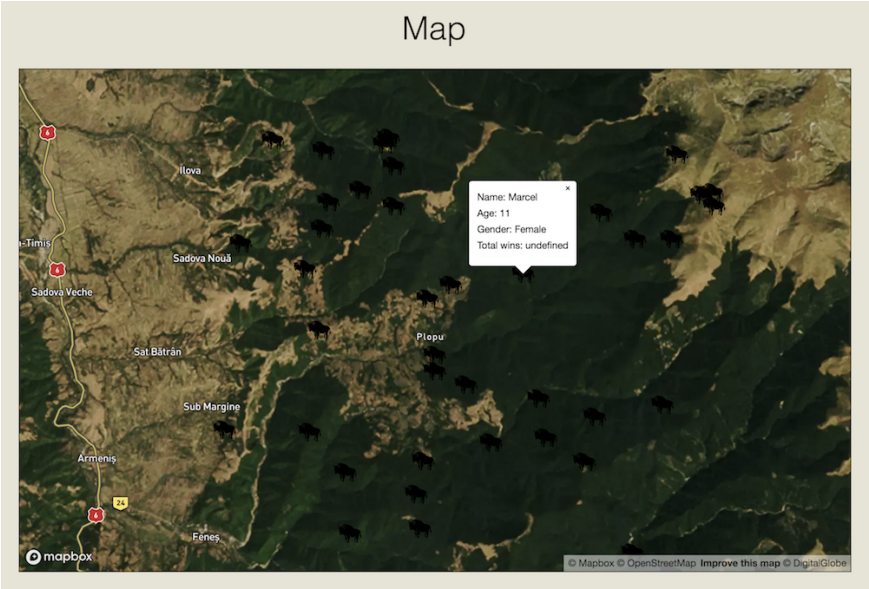


Figure 3: Map interface

The map interface gets the data after WWF has processed the IoT data. The users are able to constantly track the position of the bison on the map which is created with the use of a map API, namely Mapbox GL JS. For the prototype we have created a test data set of 100 bison with randomly generated coordinates within the area. Each bison has the following attributes: name, gender, age, distance traveled, amount of wins and longitude and latitude coordinates. Using the API and the data, each bison is placed on the map represented with an image of a bison. When clicking on a bison, a pop-up appears with the data of the specific bison. When the system is implemented with the real-time data of the bison, the bison on the map will move responsively according to the data. The betting interface will work as the main page of the website. The page will display a

list of the bison, each including the bison's information, the amount of users who has bet on it, an image of it as well as a button to bet on it. The user can search for a bison in a search field. When a user clicks on the bet-button, they are redirected to their wallet where they have to accept that a fixed amount of money will be transfered, and the bet will be placed. The user is therefore communicating directly with the smart contract, and the money is placed on the blockchain. The betting interface is illustrated in Figure 4.

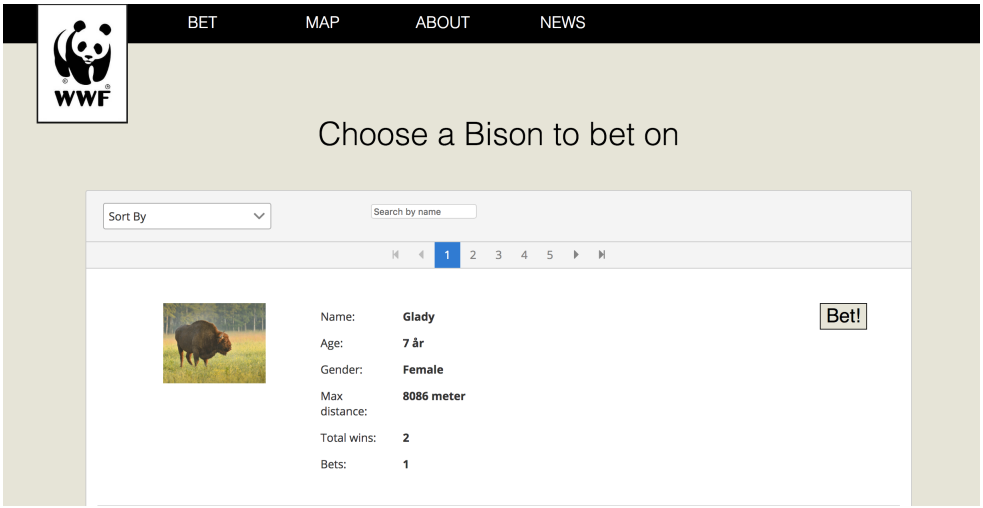


Figure 4: Betting interface

In addition to the live map and the betting interface, the website includes two other pages, namely *about* and *news*. Due to the scope of this project, we have not focused on creating these pages. The *about* page has the purpose to inform the users of the project, and will include information on what the money will be used for, and how the game works. The *news* page has the purpose to update the users with news on the project, including how their money has been spent, and what difference their support has made. This is in order to engage the users to keep playing.

The website can be accessed on the following URL: `http://bison.travedevelopment.dk/`

2.1.2 Back-end - Smart-contract

The smart contract is written in Solidity using the newest version (0.4.24 as of this writing) and it is listed in its entirety in Appendix A.

Data structures & storage variables

`adress owner`
This is the storage variable that stores the address that deployed the contract. This is used to enforce permissions (e.g. ensure that only the owner can withdraw funds from the contract).

`uint BET_AMOUNT`
This is the amount of Wei (10^-18 ether) that a bet costs. It can be changed by the owner and is

publicly visible to anyone using `getPrice`.

`bytes32[] bisonNames`
This `bytes32` list contains the names of all the bison. All the elements are used as keys in `bisons`. The reason for choosing the `bytes32` type are that it fits inside a single word in the EVM [14] and is therefore much cheaper in gas [15]. The downsides to this is that it only allows bison names of up to 32 ASCII characters and that the bytes have to be converted to human readable strings in the front-end. This compromise is worth it in our case as we have some for loops that can also be expensive.

`mapping(bytes32 => uint) bisons`
This is a mapping from bison names to their travel distance.

`address[] betters`
A list of people that have placed bets. All the elements are used as keys in `bets`.

`mapping(address => bytes32) bets`
This is a mapping from a better to the bison that they placed a bet on.

The smart contract also consists of several functions that interact with these data structures and the API is described below:

Constructor and fallback function
`constructor (bytes32[] _initialBisonNames, uint[] _initialMaxDistances) public`
The constructor initialises the datastructes by calling `updateData` on the two initial lists provided as parameters.

`function () public payable`
This is a special function that handles ether that is sent to the contract. It does nothing other than accept the ether. It can be considered a donation to the contract.

Read-write functions
`updateData(bytes32[] _bisonNameList, uint[] _maxDistances) public payable`
This function takes a list of bison names and a list of their max distances traveled (in corresponding order) and will insert the data into the data structures mentioned above. It requires that the lengths of the two lists are equal and was also supposed to require that the caller address is equal to `owner` (WWF). See 3.1.2 for more info about this technical challenge.

`placeBet(bytes32 _bisonName) public payable`
When placing a bet on a bison, a person will invoke `placeBet` as well as transfer a predetermined amount of ether along with the call. The function requires that

- the amount is exactly equal to `BET_AMOUNT`,

- the bison exists, and
- that the caller hasn't already placed a bet.

If any of these don't apply, the function exits early and returns any unspent gas to the caller.

`electWinner() public`
This is the most comprehensive function in the contract but it can be broken down into three basic steps:

1. Finding the winner bison
2. Calculating and transferring prizes and profit
3. Clearing the bets

`clearBisons() public`
Clear the `bisonNames` list. This, in turn, also means that we no longer keep track of keys for the `bisons` mapping, giving the sense of having deleted them without actually making costly write operations on the blockchain.

`clearBetters() public`
Clear the `betters` list. This, in turn, also means that we no longer keep track of keys for the `bets` mapping, giving the sense of having deleted them without actually making costly write operations on the blockchain.

`clearAll() public`
Calls `clearBisons()` and `clearBetters()`.

`function setPrice(uint _price) public`
This function is used to change the price of a bet. The caller is required to be the owner. It is assumed that it is not called while betting is open, preventing different users from paying different amounts.

Read-only functions
`getBisonNames() public view returns (bytes32[])`
Returns a list of bison names (`bisonNames`) in the order that they were inserted.

`getBisonData() public view returns (uint[])`
Returns a list of distances traveled by each bison. The order and length corresponds to that of `bisonNames`.

`totalBetsFor(bytes32 _bisonName) public view returns (uint)`
Returns the total number of people that have placed a bet on a specific bison. Requires the bison to exist.

`totalBetsForAllBisons() public view returns (uint[])`
Returns a list of unsigned integers representing the number of bets placed on each bison. The ordering of the list matches the one returned by `getBisonNames`
`totalPot() public view returns (uint)`
Returns the total sum of all bets. This will resemble the actual balance of the contract.

`bisonExists (bytes32 _bisonName) private view returns (bool)`
Returns true if `_bisonName` is present in the `bisonNames` list and false otherwise.

`betterExists (address _better) private view returns (bool)`
Returns true if `_better` is present in the `betters` list and false otherwise.

`getPrice() public view returns (uint)`
Returns the current price of a bet.

Visibility
In Solidity each variable and function has a visibility setting which can alter the access rights for different parties. They are: external, public, internal, and private [17].

We have declared most functions `public` as they are supposed to be called by our front-end; only a few helper functions are declared `private`. Our state variables default to `internal` because our contract is not interacting with other contracts.

Pure and view functions
Functions marked as `view` promise not to write to the blockchain state variables but will be able to read from it. Functions that are marked as `pure` will neither read from or write to the blockchain state variables [18]. About half of our functions are marked `view` and we have no `pure` functions.

Payable and non-payable
When a function is marked `payable` it means that it can accept ether. We have marked our `placeBet` as `payable` so that people can send the amount required to make a bet.

2.2 Deployment

The smart contract was written and deployed using the Remix IDE [19]. In the early stages of development, the contract was deployed on the JavaScript Virtual Machine for efficiency and ease, as the VM has no dependencies and transactions happen instantly.
To test how the contract interacted with the front-end, it was deployed on a local ganache server, using MetaMask [20] web3 injection to connect Remix with the ganache client. After the contract was deployed to the ganache server, the front-end was connected to the contract using the Web3 JavaScript API for Ethereum [21].
When the contract was fully developed and the front-end was successfully interacting with the contract on Ganache, it was deployed onto the Ropsten testnet [22]. This makes it possible for anyone to interact with the contract, as long as they are connected to the Ropsten testnet. The deployed

contract can be viewed on Etherscan on the following address:

`https://ropsten.etherscan.io/address/0xfd39b75ea171d25efeed020a1b1e5270f510161c`

To make interactions with the contract more user friendly, the contract was registered with Infura [23], hereby eliminating the need for the user to be connected to Ropsten locally through services like Geth [24].

To interact with and see the full functionality of the deployed contract take following steps:

1. Go to `http://remix.ethereum.org`
2. Go to the 'Run' tab
3. Load the contract address (0xfD39B75ea171D25efEeD020A1B1E5270f510161c)
4. Expand the deployed contract and all functions should be visible.

2.3 Unit testing

Unit testing is about testing the small individual parts of an application commonly referred to as units. For this project we have done unit testing by writing functions in Solidity code that test different features for our code. The test file is called test.sol and can be run in Remix.

Function	Purpose
test_constructor_success()	Tests that the constructor updates the owner as the message sender.
test_updateData_differentSize()	Tests that updateData() throws "Data lists must be of same size" when the parameter lists are not the same length.
test_updateData_success()	Tests that updateData() returns when the parameter lists are the same length.
test_placeBet_noSuchBison()	Tests that placeBet() throws "Bison does not exist", when the parameter is a non-existent bison.
test_placeBet_alreadyPlaced()	Tests that placeBet() throws "This address already placed a bet", when the parameter is a non-existent bison.
test_placeBet_wrongAmount()	Tests that placeBet() throws "You must bet exactly BET_AMOUNT", when the parameter is a non existing bison.
test_placeBet_success()	Tests that placeBet() can return without throwing exceptions and update the bets map.
test_totalBetsFor_success()	Tests that totalBetsFor() returns 1 when 1 bet is placed on a given bison.
test_totalPot_success()	Tests that totalPot() returns 1 times the bet amount when 1 bet is placed.
test_getBisonNames_success()	Tests that the length of getBisonNames() is 3 after pushing 3 bisons to the BisonNames lists.

3 Discussion

3.1 Challenges

3.1.1 Scheduling payout

Since smart contract calls cannot be scheduled, we have to rely on WWF to schedule the calls to the `electWinner` function. A good compromise would be to additionally allow users to call `electWinner` if it hasn't already been called by WWF, thus preventing the funds from getting stuck in the contract if WWF for some reason no longer wants to keep maintaining the system. Naturally, there would need to be validation in place to make sure that the function will only succeed if a certain amount of time has passed since the last winner.

3.1.2 Permissions

We have run into a bug where requiring the caller of certain privileged functions (e.g. updateData) breaks the function. It seems to be a bug in Solidity as others have reported the same thing. The specific thing that happens is that the `require(msg.sender == owner)` (currently commented out in `updateData`) does not succeed, even when the owner calls the function. We have tried several solutions such as creating a Solidity modifier which only led other weird errors. We have chosen to leave it as it is, as we're certain that it can be implemented in some other way, either with some other control flow or when the issue is resolved by the Solidity developers.

3.1.3 Rare special case

If everyone bets on the winning bison and WWF takes 5%, the resulting pot divided by the number of winners is less than the amount of their initial bet. This would mean that in rare cases, winners would actually lose money. The more users, the less likely it is to happen. An easy solution could be to have WWF renounce their 5% and let the winners share the total pot, resulting in the return of their initial value. This solution is not currently implemented but would not take more than a few lines of Solidity code.

3.1.4 A third party has to be responsible for uploading bison data

We haven't managed to find a solution to this as WWF will be the ones who gathers and owns the data. It would be an improvement if the IoT devices were made so that they automatically uploaded the data, but there is no way to prove that the devices haven't been tampered with in the physical world.

3.1.5 Solution does not scale well

Due to the limited amount of gas (determined by the block gas limit [16]) and use of for-loops in our code, we cannot handle large amounts of data. Currently only 70 bison can be inserted into the blockchain without resulting in an "out of gas" or "Transaction exceeds block gas limit" exception. Since we use a word (32 bytes) to store the name of each bison and another word to store its max distance traveled, it gives us $70 * 32\text{B} * 2 = 4480\text{B} \approx 4,5\text{kB}$.
4,5 kB of data in today's world doesn't get you very far and it becomes clear that a system capable of storing larger amounts of data is necessary.

A solution to this could be to store the data on IPFS, a content-addressable peer-to-peer distributed file system [13]. References to this data could then be used on the blockchain as these are much cheaper to store, requiring only 32 bytes and thus fitting into a single word in the EVM [14].

3.2 Future possibilities

There are numerous ways that WWF could improve and expand on the proposed solution. While not all solutions are equally easy to implement, this is what WWF could do:

- **Interactive map** Figure 3 displays what a simple version of the map looks like. Making a more interactive map containing various information could improve the overall user experience. E.g. a mouse-over could reveal what kind of terrain the bison currently is in and what animals lives there. This will give users a better idea of how long the bison would walk from a given point. A side effect of that is that it would engage users more in the nature of the Carpathian mountains, which is one of WWF's goals.

- **Custom amount** When the platform was built all users could bet a fixed amount. Changing this setting could help WWF raise more money and improve customer experience, since different users have different betting budgets.

- **Betting odds** In this beta version of the platform each winning user gets paid back the total betting pool, minus the WWF cut, divided by the number of winning users. An interesting take on the platform would be calculating odds for the bison winning. In horse betting the odds are calculated on data from various experts, conditions surrounding the horse, and historical performance. Similar data could be used to calculate odds for the bison.

- **Develop an app** The discussed solution is currently implemented on a website. Developing an app for the platform could improve user engagement, since users could bet and interact with the platform on the go. Most betting-sites for normal sites have frequently used apps.

- **Expand to other animals** The concept of charity betting can be easily extended to other animals. A fun variation of the same game could be betting on migratory birds, traveling from one end of the world to another. Tracking of the birds is something scientists do already and if the platform is built it would not be much work to adjust for another animal [12].

4 Conclusion

The purpose of this report was to propose a solution to WWF on how to raise more money and interest around their cause of saving and preserving the European bison, while also face some of their challenges as an NGO. WWF currently uses between 35-49 million USD yearly on fund-raising expenses, depending on sources, making up 11-19% of the budget. [3] [4].

We propose a blockchain-based environment betting platform, where users bet on how far a bison walks a day. When the winner(s) is announced WWF takes a fee of the total winning pot. If no one wins WWF will receive all of the money. Our proposed solution will lower the fund-raising expenses, optimize the fund-raising efficiency and is estimated to run at lower fund-raising expense than current operations.

Furthermore, implementing the platform on the Ethereum blockchain will help WWF Romania tackle some of the problems NGO's face in general. More specifically, transparency and measure of performance would be more apparent, since the transaction flow would be visible in the blockchain, while the well-being of the Carpathians as a bison-habitat could be assessed through the geographical position of the bison.

The platform will require IoT devices to be mounted on the bison, to track the location of the bison in real-time. The IoT devices will feed over in the front-end map interface, allowing users to track their bets, while also going through WWF for processing, allowing the data to be inserted into the blockchain. The solution will have some challenges, such as scheduled pay-outs, permissions, third-part handling of data etc., but nothing that can't be solved.

Furthermore, the platform as a prototype has big potential if developers were to implement a few changes, such as an interactive map, betting odds and an app. Some ideas are more demanding, such as implementing concept on migratory birds, but would expand the idea without requiring an extensive amount of work.

References

[1] *WWF - Endagered Species Conservation.*
WWF Foundation, 2018
https://www.worldwildlife.org/

[2] *Blockchain Summer School WWF Use Case: Sustainable Landscapes – Bison in the Carpathians and preserving Europe's largest wilderness stronghold*
WWF Foundation, 2018

[3] *WWF Romania - Raport Anual 2016.*
WWF Romania, 2017
http://d2ouvy59p0dg6k.cloudfront.net/downloads/raport˙anual˙2016.pdf

[4] *Charityn Navigator - WWF.*
Charity Navigator, 2018
https://www.charitynavigator.org/index.cfm?bay=search.summary&orgid=4770

[5] *Arguments against charity.*
BBC, 2014
http://www.bbc.co.uk/ethics/charity/against_1.shtml

[6] *Charityn Navigator - Glossary - Joint Cost Allocation Adjustment.*
Charity Navigator, 2018
https://www.charitynavigator.org/index.cfm?bay=glossary.word&gid=75&print=1

[7] *Bullfighting.*
PETA, 2018
https://www.peta.org/issues/animals-in-entertainment/cruel-sports/bullfighting/

[8] *Horse-racing.*
PETA, 2018
https://www.peta.org/issues/animals-in-entertainment/horse-racing/

[9] *The Secret to Happiness Is Helping Others.*
Jenny Santi
Time, 04-08-2017
http://time.com/collection-post/4070299/secret-to-happiness/

[10] *Bitcoin: A Peer-to-Peer Electronic Cash System.*
Satoshi Nakamoto.
Bitcoin, 2015.
https://bitcoin.org/bitcoin.pdf

[11] *Blockchain: Blueprint for a New Economy.*
Melanie Swan.
O'reilly, 2015.

[12] *With these maps, you can track migratory birds in near real time.*
Tom Oder
Mother Nature Network, 22-08-2018
https://www.mnn.com/earth-matters/animals/stories/cornell-updated-birdcast-maps-track-bird-migrations

[13] *IPFS - Content Addressed, Versioned, P2P File System (DRAFT 3).*
Juan Benet
IPFS Project, 01-04-2015
https://github.com/ipfs/papers/blob/35a5684c6092035eff257fb5fd6045768a7ae830/ipfs-cap2pfs/ipfs-p2p-file-system.pdf

[14] *Ethereum Design Rationale.*
Ethereum Foundation
Ethereum Foundation, 21-08-2018
https://github.com/ethereum/wiki/wiki/Design-Rationale/6e97c9cea49605264c6f4d1dc9e1939b1f89a5a3

[15] *Bytes and strings in Solidity.*
Cryptopusco
Medium, 09-01-2018
https://medium.com/@cryptopusco/bytes-and-strings-in-solidity-f2cd4e53f388

[16] *Accounts, Transactions, Gas, and Block Gas Limits in Ethereum.*
Hudson Jameson
Hudson Jameson, 27-06-2017
https://hudsonjameson.com/2017-06-27-accounts-transactions-gas-ethereum/

[17] *Visibility and getters.*
Ethereum Foundation
Read the Docs, 2018
https://solidity.readthedocs.io/en/v0.4.24/contracts.html#visibility-and-getters

[18] *Functions.*
Ethereum Foundation
Read the Docs, 2018
https://solidity.readthedocs.io/en/v0.4.24/contracts.html#functions

[19] *Remix.*
Solidity IDE
remix.ethereum.org

[20] *MetaMask.*
MetaMask browser extension
https://metamask.io/

[21] *web3js.*
Ethereum JavaScript API
https://web3js.readthedocs.io/en/1.0/

[22] *Ropsten*.
Ropsten testnet
https://github.com/ethereum/ropsten

[23] *Infura*.
Scalable Blockchain Infrastructure
https://infura.io/

[24] *Geth*.
Go-Ethereum Wiki
https://github.com/ethereum/go-ethereum/wiki/geth

Appendices

A Smart contract code

Listing 1: The BisonCasino smart contract code

```
pragma solidity ^0.4.24;

contract BisonCasino {

    // WWF's address which will receive %5 of winners' earnings
    address owner;

    uint BET_AMOUNT = 1000;

    /* Data structures */
    bytes32[] bisonNames;
    mapping(bytes32 => uint) bisons;

    address[] betters;
    mapping(address => bytes32) bets;


    /* Constructor and fallback function */
    constructor (bytes32[] _initialBisonNames, uint[] _initialMaxDistances) public
        ↪ {
        updateData(_initialBisonNames, _initialMaxDistances);

        owner = msg.sender;
    }

    function () public payable {
    }

    /* Read-write functions */
    function updateData(bytes32[] _bisonNameList, uint[] _maxDistances) public {
        //require(msg.sender == owner, "Only the owner can update data");
        require(_bisonNameList.length == _maxDistances.length, "Data lists must be
            ↪ of same size");

        for(uint i = 0; i != _bisonNameList.length; i++) {
            bytes32 bisonName = _bisonNameList[i];

            if(!bisonExists(bisonName))
                bisonNames.push(bisonName);
```

```
        bisons[bisonName] = _maxDistances[i];
    }
}

function setPrice(uint _price) public {
    //require owner
    BET_AMOUNT = _price;
}

function placeBet(bytes32 _bisonName) public payable {
    require(bisonExists(_bisonName), "Bison does not exist");
    require(!betterExists(msg.sender), "This address already placed a bet");
    require(msg.value == BET_AMOUNT, "You must bet exactly BET_AMOUNT");

    betters.push(msg.sender);
    bets[msg.sender] = _bisonName;
}

function electWinner() public {
    uint i;

    // find best bison
    bytes32 bestBison = "";
    for(i = 0; i < bisonNames.length; i++) {
        bytes32 newBison = bisonNames[i];

        if(i == 0) {
            bestBison = newBison;
        } else {
            if(bisons[newBison] > bisons[bestBison]) {
                bestBison = newBison;
            }
        }
    }

    // calculate winnings and profit
    uint wwfcut = totalPot() / 20;
    uint payout = totalPot() - wwfcut;

    uint bestBisonBets = totalBetsFor(bestBison);
    if(bestBisonBets > 0) {
        uint split = payout / bestBisonBets;

        // transfer winnings and profit
        for(i = 0; i < betters.length; i++) {
```

```
            address better = betters[i];

            if(bets[better] == bestBison) {
                better.transfer(split);
            }
        }
    }

    owner.transfer(address(this).balance);

    betters.length = 0;
}

function clearBisons() public {
    bisonNames.length = 0;
}

function clearBetters() public {
    betters.length = 0;
}

function clearAll() public {
    clearBisons();
    clearBetters();
}

/* Read-only functions */
function getPrice() public view returns (uint) {
    return BET_AMOUNT;
}

function getBisonNames() public view returns (bytes32[]) {
    return bisonNames;
}

function getBisonData() public view returns (uint[]) {
    uint[] memory bisonData = new uint[](bisonNames.length);
    for(uint i = 0; i < bisonNames.length; i++) {
        bytes32 bison = bisonNames[i];
        bisonData[i] = bisons[bison];
    }
    return bisonData;
}

function totalBetsFor(bytes32 _bisonName) public view returns (uint) {
```

```
    require(bisonExists(_bisonName), "Bison does not exist");

    // count total bets
    uint totalBets = 0;
    for(uint i = 0; i < betters.length; i++) {
        address better = betters[i];

        if(bets[better] == _bisonName) {
            totalBets += 1;
        }
    }
    return totalBets;
}

function totalBetsForAllBisons() public view returns (uint[]) {
    uint[] memory betArray = new uint[](bisonNames.length);
    for(uint i = 0; i < bisonNames.length; i++) {
        bytes32 bison = bisonNames[i];

        betArray[i] = totalBetsFor(bison);
    }
    return betArray;
}

function totalPot() public view returns (uint) {
    return betters.length * BET_AMOUNT;
}

function bisonExists (bytes32 _bisonName) private view returns (bool) {
    for(uint i = 0; i < bisonNames.length; i++) {
        if(bisonNames[i] == _bisonName) {
            return true;
        }
    }
    return false;
}

function betterExists (address _better) private view returns (bool) {
    for(uint i = 0; i < betters.length; i++) {
        if(betters[i] == _better) {
            return true;
        }
    }
    return false;
}
```

```
}
```