

Ronin Chellakudam, Lauri Tahvanainen, Md Rezuanul Haque and Martin Frick

Department of Informatics IfI University of Zurich UZH

Binzmühlestrasse 14, CH---8050 Zürich, Switzerland

{ronin.chellakudam, lauri.tahvanainen, rezuanul.haque, martin.frick}@uzh.ch

**Abstract.** The abstract should summarize the contents of the paper and should contain at least 70 and at most 150 words. It should be written using the abstract format. This example is for illustrative purposes only; for example, paragraphis in the outline below are much shorter than we expect you to have! Follow good writing style. We strongly encourage you to list up to 5 relevant keywords below; the five given below are simply examples. Use your own! If you are having trouble thinking of keywords, see the IEEE Computer Society keyword taxonomy [4].

1. Introduction

Our goal was to create a platform (DApp) where players can bet against each other without a centralized provider demanding a commission in between. The type of bets we offer are “back”- and “lay”-bets (livetipsportal, 2021). The players have the possibility to back or to lay a bet which allows them to either bet for or against a certain outcome and to offer a bet to other players or to accept a bet another player has offered.

The bet can be almost on any type of event. A champions league football match, a beer league ice hockey match, a political event or even a funny bet with a friend about being able to throw a ping pong ball in to a glass while blind. The way we can achieve having bets of so many types is through having the default situation be that the players agree on the outcome of the event. After a bet has been laid, the players stake is frozen until the bet is over or removed. A user who lays a bet also must decide on some options. These include: the outcomes, the odds, the stake on the outcome and a description of the bet. By deciding the odds of the outcomes and betting on one of these outcomes the layer also fixes the possible stake on the other outcome.

1. Solution/DApp Design

What was the project, what were the specifications. If you are doing a game, you would put the characteristics of the game, what guided your work (eg the rules? the graphics? the language? the platform(s)?) If this was a learning project, what did you hope to learn? What was significant about this problem to you? What criteria did you use to determine your "success"?

A bet pool is created by an interaction with a smart contract. In a simple 1v1 pool the creator configures the pool and adds stake and waits for someone to agree on the odds and provide their stake. The stakes would then be locked in the pool for a certain time or until both parties vote on the result. When the outcome is known, both parties interact with the contract to broadcast their opinion on the result. If these opinions match the rewards are delegated to the winner accordingly. If there is a dispute on the result, the player can send the dispute to Kleros court for arbitration. The jury will make a decision and the smart contract will react to the decision awarding the winner and punishing the other. For the pool to use Kleros there needs to be certain amount of stake locked for a possible court case. In addition, a percentage of 5% of the stake of the user who is wrong will be charged.

In our project, we were guided by some of the biggest centralized betting platforms and their type of bets they offer. Or goal was to create a similar platform but decentralized and with any kinds of bets. As this was a learning project, we hoped to learn how to create our own DApp and to get familiar with solidity as well as with the tools needed like Ganache, MetaMask and many more.

For our DApp, it was crucial to implement a functioning smart contract, as it forms the basis of the project and allows the players to interact. It was also highly relevant to integrate the Kleros mechanism to have a decision if there is a disagreement about the outcome of the bet. Furthermore, we wanted to have a user interface that appeals to the users and is easy to use. To determine success, we ran tests on both the backend and frontend to make sure the DApp was working properly.

* 1. Use cases (If any)

What specific functions does your project accomplish for its users? A use case is an objective user(s) wants to achieve with a system. “Use cases are named with verb or verb + noun phrase. It is usually short yet descriptive enough to describe a user objective.” See Visual Paradigm’s brief tutorial for writing effective use cases [8].

A user can either create a new bet on the platform or accept a bet created by another user (e.g. place a bet). Thus, the user can view the list of all bets created and filter for his own bets or for specific bets he wants to back. All bets are placed in Ether. When the deadline is reached, the user can vote on the outcome of the bet. If there is agreement on the outcome, the winner of the bet can claim their winnings. In the event of a disagreement, the user has the option of triggering the Kleros mechanism. In the rare case that there is no result, e.g. if a match is cancelled, the user can claim a refund.

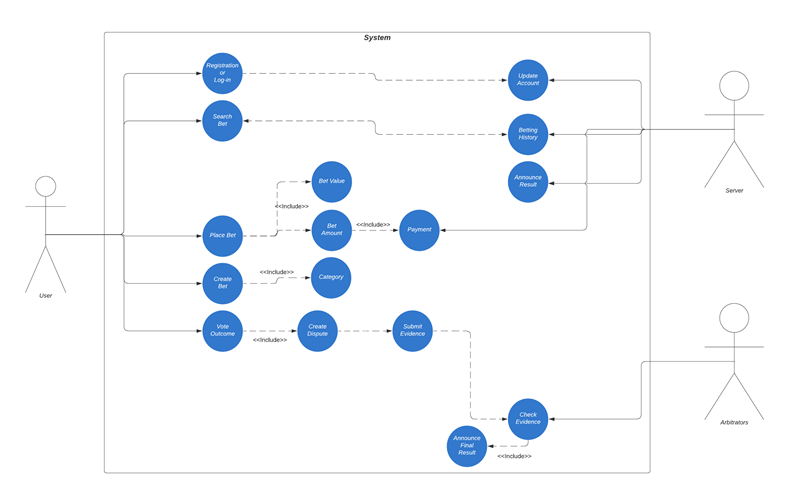


Figure 1: Use Case Diagram

* 1. Implementation choices

Details about your hardware and software choices (if you used pre-existing packages, for instance.) How you went about testing your project or what criteria you used in order to determine your outcomes.

If your project has an interesting or novel hardware configuration, you should include a diagram here, or in a separate subsection “Hardware Architecture”.

To interact with our smart contract, we used web3 which is an JavaScript library allowing us to send and receive transactions from a smart contract which is deployed on the local Ethereum platform Ganache. We further use MetaMask which can be installed as plugin for web browsers to connect to Ganache for Ether transactions. Also, we used MetaMask for testing purposes as it maintains the user’s wallet. By using it we can simulate transactions performed by a user. We further decided to use Graph as a decentralized data base to store all our bets.

We further used the ReactJS framework to develop our Webapp. In particular we used Next.JS to design an UI which is appealing to the users and easy to handle as well as scalable on all sorts of devices.

* 1. Software Architecture

How the code itself is organized into classes and/or functions. It's good to have an example that puts your project in context. Examples will vary depending on your project.

We also strongly suggestion you include a software architecture diagram. If your project has a database, we suggest you include a conceptual database model diagram. Remember, it is not enough to simply include the figure in your paper; you must also give it a caption and explain it in the text (referring to the figure in that text).

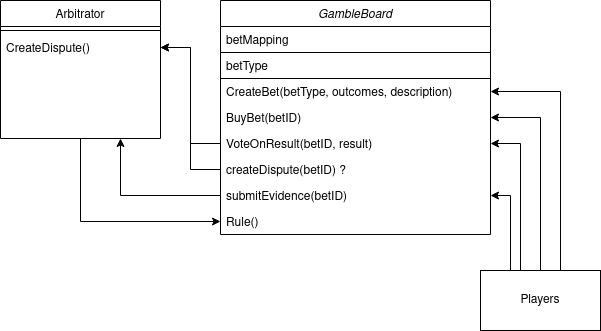


Figure 2: GambleBoard Software Architecture

* 1. Program Code

If relevant you could include a short piece of program code that you use to demonstrate some feature of your work. The LNCS template provides a way of listing your code in the proper format; in Word, this is the Prog.Code button. Use it!

Below is an example of some simple C++ graphics code that draws a robot arm. Note how the current ViewModel is pushed onto the stack prior to calling the translate and rotate functions; thus World Coordinates are preserved across those functions.

void drawArm(mat4 &mv)

{

mvMatrixStack.pushMatrix(mv);

mv = mv\*Translate(0,armLength/2.,0);

mv = mv\*Scale(.2,armLength,.2);

glUniformMatrix4fv( model\_view, 1, GL\_TRUE, mv );

myCube.draw();

mv = mvMatrixStack.popMatrix();

}

Back-end

A Bet is saved as struct as shown below:

Struct Bet {

uint256 stakingDeadline;

uint256 votingDeadline;

uint256 backerStake; // Arbitration fee is added to the fee payers stake.

uint256 creatorStake;

RulingOption outcome;

State state;

bytes1 voteEvidenceBools;

address payable creator;

address payable backer;

string description;

string creatorBetDescription;

}

Players can interact with the smart contract using the following interfaces:

**CreateBet(description, creatorBetDescription, league, country, categroy, stakingDeadline, timeToVote, creatorOdd, metaEvidence)**

-A player creates a bet, giving all the necessary information about the bet. The bet is then mapped to an id when created with all the necessary information. The data structure implementation should be discussed and tested, storage costs gas.

-Calculates the amount a backer has to stake from the creator’s stake and odd

-Sets state of the bet to open

-Emits a message

-Returns if successful, betID

**PlaceBet(betID)**

-A player backs a bet, giving the betID as parameter.

-Check that the state is open

-Emits a message

**VoteOnOutcome(betID, outcome)**

-A player interacts with the contract to give their suggestion on the outcome of the bet

-If all have given their suggestion the following happens:

a: All players agree and the winnings can be claimed by the winner

b: Players do not agree and d to Kleros: For case there needs to be a fee deposited for Kleros.

-Returns if successful

**refund(betID)**

-If no players voted in time or if the votes were on NO\_OUTCOME, the stakes can be refunded by triggering this function.

- State will be set to refunded

-Emits a message

**claimWinnings(betID)**

-If only one player voted within the time to vote, the winner of the bet will be choosen based on the one voting.

-Player who won the bet can claim winning, but can also be called by loser

-Function can only be called after voting Deadline

-State will be set to resolved

-Emits a message

**CreateDispute(betID, fee)**

-A player creates a dispute about a bet.

-Returns if successful

**provideEvidence(betID, evidence)**

-A player submits evidence on the dispute of a bet. Is done by emitting.

**executeruling(uint256 \_disputeID, uint256 \_ruling) external**

-Is called by the arbitrator to give a ruling on the bet

-Distribute value to players of the bet according to the ruling.

* 1. Front-End / Graphical User Interface

tbd

1. Conclusions/Critique

Identify both what succeeded (your positive results) and what didn't. If your work had a major shift in orientation you should have provided an indication earlier but you could discuss the consequences of that change here. Specify what design or programming issues you ran into and how you addressed them.

To what extent did the final version of your project resemble your original design for the project (as described in your project proposal and project plan.)

* Which parts of your project work went the way you expected them to? Which ones did not?
* How did keeping minutes affect your group's work? How did it shape your experience of your project work.
* If you were to do this project again, what would you do differently next time?

1. Future Work

What did you learn and what future plans do you have for this work. Or if you have no specific plans to continue the project, how the learning you accomplished here will inform your future studies/work.

1. References

Just list your references here (don’t include this paragraph). Use standard format (e.g., look at what follows). Be sure to include texts or web sites you used as references, as well as scholarly journals. Remember you are aiming to provide material that would help someone recreate your project. Alphabetize your references.

1. Czajkowski, K., Fitzgerald, S., Foster, I., Kesselman, C.: Grid Information Services for Distributed Resource Sharing. In: 10th IEEE International Symposium on High Performance Distributed Computing, pp.181--184. IEEE Press, New York (2001)
2. Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, San Francisco (1999)
3. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The Physiology of the Grid: an Open Grid Services Architecture for Distributed Systems Integration. Technical report, Global Grid Forum (2002)
4. IEEE Computer Society Keywords, http://www.computer.org/  
   portal/web/publications/acmtaxonomy
5. May, P., Ehrlich, H.C., Steinke, T.: ZIB Structure Prediction Pipeline: Composing a Complex Biological Workflow through Web Services. In: Nagel, W.E., Walter, W.V., Lehner, W. (eds.) Euro-Par 2006. LNCS, vol. 4128, pp. 1148--1158. Springer, Heidelberg (2006)
6. National Center for Biotechnology Information,   
   <http://www.ncbi.nlm.nih.gov>
7. Smith, T.F., Waterman, M.S.: Identification of Common Molecular Subsequences. J. Mol. Biol. 147, 195--197 (1981)

Visual Paradigm, [http://www.visual-paradigm.com/  
tutorials/writingeffectiveusecase.jsp](http://www.visual-paradigm.com/tutorials/writingeffectiveusecase.jsp)