



WHITEPAPER

September 8 2017

# **SmartBillions. World's first multi-billion-dollar lottery.**

SmartBillions is the first fully decentralized and transparent lottery managed by an Ethereum smart contract. All bets and results are public and recorded on the Ethereum blockchain without 3<sup>rd</sup> party involvement. Only 0.25 % of the lottery funds are accessible to the smart contract operations.

SmartBillions tokens crowdsale ("ICO") goal is set at 200 000 ETH. 90% of the funds raised will be allocated to the Jackpot. The remaining funds will be designated to marketing.

SmartBillions Token ("PLAY") holders will receive a monthly dividend starting one month after the initial token offering closing date. The Estimated Annual Yield is over 30% .

Token Holders are given a revolutionary investment protection guarantee. They are allowed to redeem Tokens any time after the PLAY Tokens crowdsale closing date and reclaim most of the invested funds, even in the event of currently unpredictable issues.

# Welcome to SmartBillions

*As experienced decentralization enthusiasts and blockchain professionals, we created the first independent blockchain lottery based on the Ethereum smart contract architecture.*

*SmartBillions is not a company. It is a smart contract conducting a decentralized Jackpot lottery with a theoretically unlimited Jackpot potential. We believe that the quality and value of the SmartBillions contract technology will validate itself. To prove the security of the SmartBillions contract, the SmartBillions Team will host a hackathon 14 days prior to the start of the limited crowdfunding Token (PLAY) sales. SmartBillions will place 1500 ETH in the contract's Jackpot and invite anyone to hack the contract and withdraw the funds. The hackathon will confirm the contract's security and ensure that the Token holders' investments are protected.*

*We stand against greedy ICO's business models, fraudulent intentions and unfair fund and token distribution schemes without genuine products behind them. We stand for the fundamental blockchain values (decentralization, disintermediation, transparency, security and freedom), as well as comprehensive investor transparency. The Token and fund distribution models offered by SmartBillions are extremely favorable for the Investors. SmartBillions is the first ICO to offer downside protection for its crowdsale's backers.*

*Blockchain is not just about people - it's about the technology for the people. It is revolutionizing old industries which have long been based on community-unfavorable monopolies and lack of transparency. And just like blockchain and the idea of the smart contract, SmartBillions is not about the people behind it. It is for the people that use it and for their fully transparent freedom to win.*

## **The Team**

# Table of Contents

<b>1. PRODUCT INTRODUCTION</b>	5
<b>2. LOTTERY MARKET OVERVIEW</b>	6
<b>3. NEW LEVEL OF TRUST. SMARTBILLIONS MISSION AND VISION</b>	8
<b>4. KEY FEATURES AND COMPETITIVE ADVANTAGES</b>	9
a. Fully comprehensive transparency.	9
a1. Transparency of lottery draws.	10
b. Highly attractive expected wins.	12
c. Inviolable anonymity.	12
d. Convenient flexibility and user-friendly UX.	13
e. Instant results and payouts.	13
f. Low transaction costs.	13
g. Flexible Ticket value.	13
h. Elimination of cheating opportunities.	13
i. Favorable Ticket sales funds distribution.	13
j. Small house edge.	14
<b>5. MULTICHANNEL PROMOTION SCHEME</b>	15
<b>6. CROWDFUNDING MODEL AND PLAY TOKENS</b>	16
a. PLAY Token sales funds distribution.	17
b. PLAY Token distribution.	18
c. PLAY Token key features.	18
- Token redemption possibility.	18
- Token dividend payouts.	19
- Token exchangeability.	20
d. Uniqueness of the SmartBillions crowdsale investment.	20
<b>7. THE SMARTBILLIONS JACKPOT</b>	23
<b>8. SYSTEMIC SECURITY</b>	24
a. The Hackathon.	24
b. Security audit.	25
c. Lottery closing.	25
<b>9. PROJECT TIMELINE</b>	25
<b>10. APPENDIX: The SmartBillions smart contract.</b>	27

# 1. PRODUCT INTRODUCTION

A blockchain-based smart contract is a decentralized system architecture existing between a variety of permitted parties, where all intermediaries are eliminated. Banks and governments are now turning to blockchain systems as they are cheaper, faster and more secure than existing traditional system of data organization and exchange. While the blockchain mode of operations is designed as an expanding order (called blocks) register, it makes data fundamentally immune to alteration; the orders registered in blocks are distributed to make any future changes impossible.

The idea of smart contracts based on a decentralized ledger (digital contract, blockchain contracts, self-executing contracts) emerged from the work of cryptographer and legal academic Nick Szabo in the early 1990s. In essence, the idea can be summarized as a conversion of contracts to computer code, where the storing and replication takes place on the network. The process is automatically supervised by the network of computer devices participating in the blockchain circuit. As a result, the ledger feedback loop takes place and the transfer of assets or receiving of services/products is made possible.

They become an infrastructure that allows a zero conflict and fully transparent exchange of value (e.g. shares, money, property), bereft of “middleman” intermediaries. The smart contract infrastructure allows users to easily pay to the system (ledger), receive value and benefit from the security of automatically-enforced obligations based on rules and penalties specified in the contract.

Conceived and built by Vitalik Buterin, the Ethereum currency optimizes the smart contract approach by transferring currency (or assets) into the program-based platform. From that point on, “the program runs this code and at some point it automatically validates a condition and it automatically determines whether the asset should go to one person or back to the other person, or whether it should be immediately refunded to the person who sent it or some combination thereof,” explained Buterin at a recent *DC Blockchain Summit*. Meanwhile, this decentralized ledger system automatically stores and multiples the document – securing its immutability.

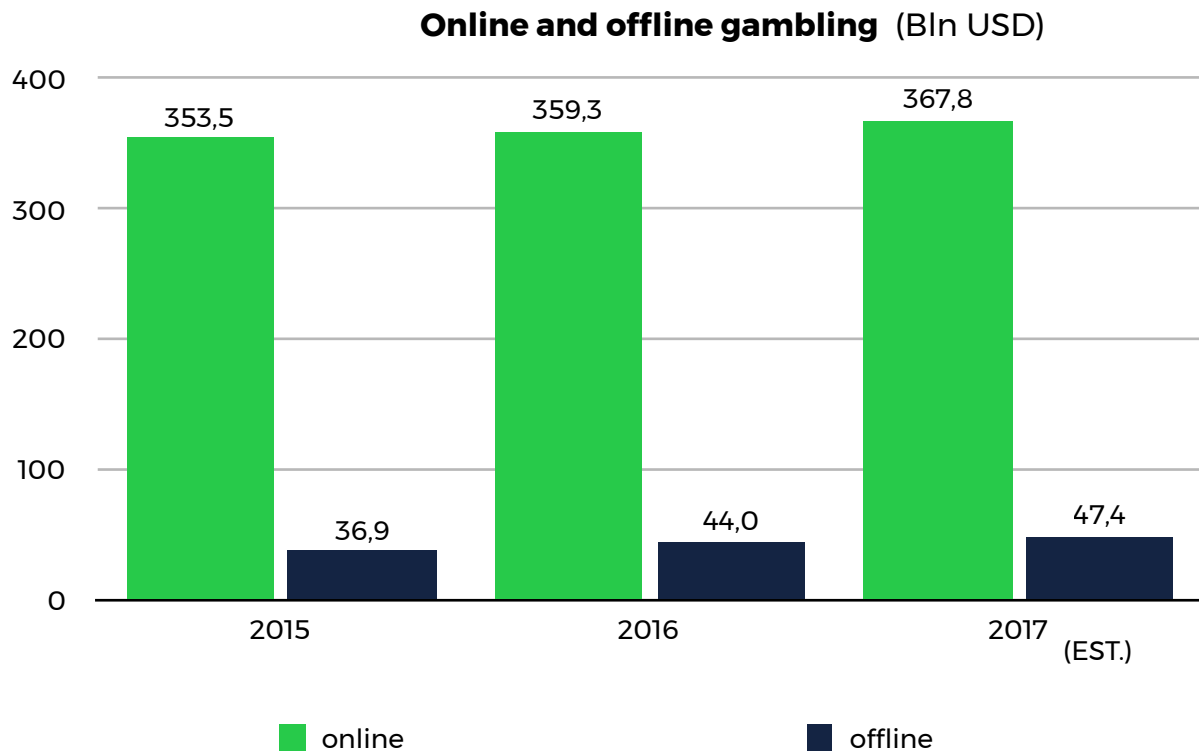
**The SmartBillions** is an Ethereum smart contract where all operations are recorded on the blockchain and public. SmartBillions is a lottery where players chose 6 numbers between 0 and 15 and set their ticket values. Payouts require at least 2 correct matches. The lottery results are taken from the hash of the third subsequent Ethereum block (its last 6 digits). This guarantees full transparency and fairness of the lottery.

The revolutionary character of the SmartBillions Ethereum smart contract lottery comes from:

1. The lottery is operated by a fully independent smart contract serving as a self-amending regulatory guarantor. The whole process is held on Ethereum blockchain.
2. The limited PLAY Tokens sale crowdfunding goal is set for the highest initial Jackpot of all online lotteries (180,000 ETH), with an unprecedented win structure not limited to the Jackpot value.
3. Full transparency and security thanks to the smart contract's elimination of any third-party involvement in the lottery process and funds management.
4. Lack of sign-up requirements providing full anonymity with instant Ticket purchase capabilities.
5. No prior deposit requirements - direct Ticket payment from players' digital wallets.
6. Immediate, secure and anonymous payouts directly after each lottery drawing.
7. Nearly instantaneous bet placement (under 1 minute), results (approximately 60 seconds) and win payout (instant).

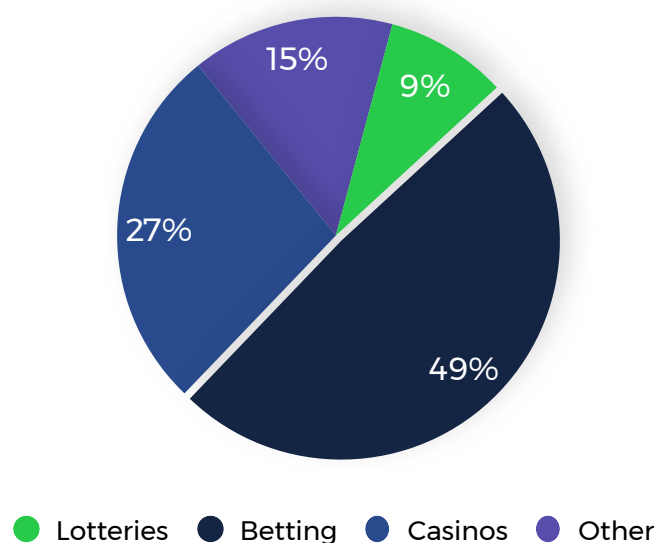
## 2. LOTTERY MARKET OVERVIEW

The global offline and online gambling market categories continue to grow year to year. While the first one reached out 359,3 billion USD in 2016, the online market was estimated at 44 billion USD. However, the online gambling category is experiencing faster growth (10%) than traditional offline gambling and is expected to reach 60 billion USD annually by 2020. The online lottery market subcategory currently accounts for approximately 4 billion USD annually and is expected to grow at a 10-12% per annum).



According to the World Lottery Association (WLA), in 2016, sales of offline draw-based lottery tickets accounted for 143.5 billion USD (approximately 30% of the global offline gambling market). On the other hand, data from H2 Gambling Capital, a betting and gaming consultancy, shows that online lottery ticket sales amounted to only 3.85 billion USD in the same period. Lotteries accounted for 9% of the online gambling market, with remainder made up of betting (49%) and online casinos (27%).

### Online gambling market



A lack of transparency is the main factor slowing down online lotteries' growth. None of the currently operating online lotteries are decentralized and transparent. Their processes depend on third party involvement which is not subject to any control mechanisms and creates a high risk of lottery operators influencing draw results and prize payouts.

SmartBillions will bring full transparency to online lotteries and marks the birth of a new kind of trust and quality in the market. With its innovative solutions, SmartBillions overcomes all of the existing online chance-gaming challenges. Furthermore, the global lottery market data does not limit SmartBillions' market potential, as it constitutes a new market category and is dedicated to both blockchain professionals as well as the general public.

SmartBillions' goal is to attain 10% of the online lottery market category turnover within its first year of operations.

### **3. NEW LEVEL OF TRUST. THE SMART-BILLIONS MISSION AND VISION**

The launch of SmartBillions marks the birth of a lottery which provides a new level of quality and trust. It is designed as a community-driven global initiative supporting Ethereum and gameplay communities. It is a part of the worldwide disruptive and democratizing blockchain movement. It is estimated that by 2025, as much as 70% of all possible markets in the world will rely on blockchain technology. The transparency and the trust attributed to the smart contracts operating on the blockchains, derive from the fact that all agreements are executed automatically and allow no third-party access or administrator intervention.

Blockchain allows a fundamental global shift in decentralized re-intermediated economy. It answers the social need for a global reinvention of lottery systems and their governance, which will lead to full transparency. The world's lotteries must either adapt to this new paradigm or get disrupted, as full transparency is certainly the industry's future. This shift is being dynamically introduced to all branches of the economy through blockchain technology.



**SmartBillions' mission** is to provide a new, fully transparent economy to the lottery world, bringing incontestable freedom and equal chances of winning to all players around the world.

Self-regulated smart contract solution gives SmartBillions an unprecedented chance to disrupt gaming category politics. The absence of internal and external governance, hidden fees, unclear rules and potential of deception will become the industry standard.

**SmartBillions' vision** is to become the first truly global, transparent and anonymous lottery with unlimited Jackpot potential. SmartBillions, with its internal and transparent management and no 3<sup>rd</sup> party involvement will revolutionize and create a new standard for all lotteries and will become a reference point for all future lottery projects. In short, we believe that SmartBillions will become the world's biggest lottery.

## 4. SMARTBILLIONS KEY FEATURES AND COMPETITIVE ADVANTAGES

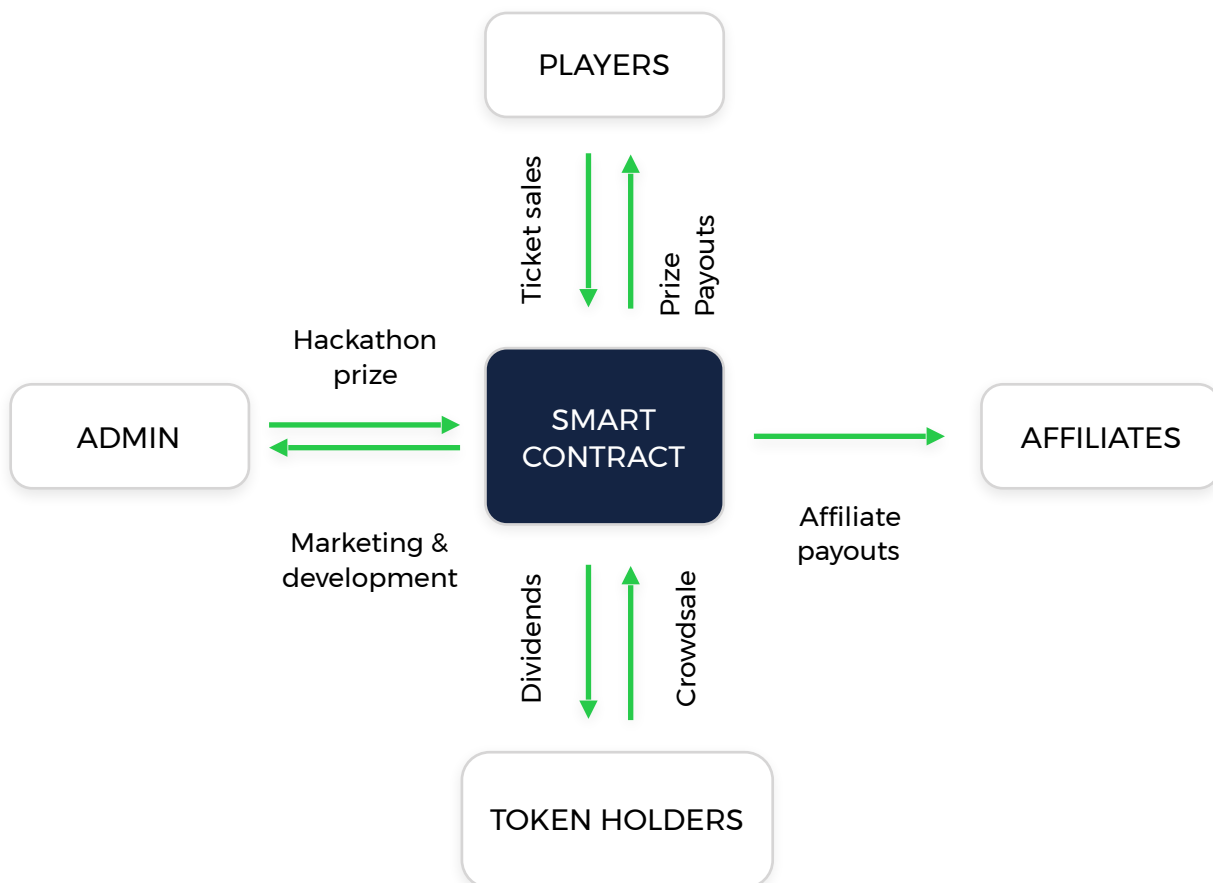
The key features of the SmartBillions Ethereum lottery sum up to both its uniqueness and fully transparent character. The most important defining features are:

### A. FULLY COMPREHENSIVE TRANSPARENCY

SmartBillions is a decentralized and self-managing smart contract lottery.

Contract balance and all transactions are public and transparent. Funds are under smart contract administration and fund distribution rules are unchangeable.

All lottery transactions are held on the Ethereum blockchain, all bets, results and payouts are public, independent from any third-party involvement and cannot be manipulated or influenced. The fund management scheme is presented below.



## A1. FULLY COMPREHENSIVE TRANSPARENCY

The rules of the SmartBillions Ethereum smart contract lottery are unchangeable. Players choose the predicted sequence of 6 numbers between 0 and 15 (hexadecimal Ethereum hash record) and set the lottery ticket value.

The draw result is the last 6 digits of the hash of the third ensuing Ethereum block.

Digits in Ethereum block hash are written in the Hexadecimal format. The table below presents simple conversion from hexadecimal to decimal numbers.

Hexadecimal	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The Lottery Drawing consist of the following steps:

- a. Player chooses 6 numbers (0-15) in specific order.
- b. For example: 4, 15, 9, 5, 0, 13.
- b. Player confirms lottery Ticket purchase transaction and waits for it to be included in an Ethereum block.
- c. The bet is accepted in block 4098562.
- d. The bet result will come in the third future block (+3): 4098565.
- e. Sample Block 4098565 hash: 0xa248b6b2504cdb483bd4aa392ab6c-cd4f4249638a371686e2ddd9117ac2f3b0d
- f. The draw result is the last 6 digits of block 4098565 hash, i.e. 2f3b0d.
- g. The draw results are converted into the decimal format:
  - 1st number: "2" = 2
  - 2nd number: "f" = 15
  - 3rd number: "3" = 3
  - 4th number: "b" = 11
  - 5th number: "0" = 0
  - 6th number: "d" = 13
- h. The numbers chosen by the player were: 4, 15, 9, 5, 13, 0.
  - a. The draw result is: 2, 15, 3, 11, 13, 0.
  - b. 3 numbers match: 15, 13 and 0.
  - c. Player receives payout.

Neither the administrator nor any other 3<sup>rd</sup> party can affect the lottery process. Once the bet is placed, the smart contract automatically executes the next steps.

## B. HIGHLY ATTRACTIVE EXPECTED WINS

SmartBillions payouts are instant, direct, secure and anonymous. They are many times higher than the ones offered by any previous lottery, as well as those offered by all current offline and online competitors. The odds and expected wins are presented in the table below:

## SmartBillions lottery win odds and multiplier structure:

HIT	WIN ODDS 1 TO:	WIN MULTIPLIER
2 out of 6	22	3
3 out of 6	249	25
4 out of 6	4,971	500
5 out of 6	184,414	20,000
6 out of 6	16,777,216	7,000,000

### C. INVIOABLE ANONYMITY

The systemic anonymity provided by SmartBillions protects the players and secures their interests. All potentially threatening and risk-generating factors have been eliminated from the lottery operating processes. Participating in and placing bets on the SmartBillions lottery does not require prior registration or deposits. The only thing the players need is a secure Ethereum wallet or Metamask wallet installed. Bet placing and lottery win payout processes do not produce any emails or confirmations and don't require any personal information to be provided.

### D. CONVENIENT FLEXIBILITY AND USER-FRIENDLY UX

Lottery participation by Ticket purchase is available to everyone.

To place a bet, a player may use the interface together with a Matamask wallet (a simple and user-friendly solution) or place a bet directly from their Ethereum wallet.

## **E. INSTANT RESULTS AND PAYOUTS**

Draw results appear approximately 60 seconds after the bet is placed (the third future Ethereum block). The process of placing a bet and awaiting the results does not take much more than a minute – an unprecedented achievement in both online and offline lottery history. Win withdrawal is possible right after the draw results are published, and the funds are immediately transferred to the same Ethereum wallet the bet was placed from.

## **F. LOW TRANSACTION COST**

Thanks to the unique smart contract architecture, the cost of each bet placement is reduced to the minimum. The cost of GAS (transaction cost) used to place the bet can be as low as 0.03 USD – significantly lower than in any other similar Ethereum contract transaction.

## **G. FLEXIBLE TICKET VALUE**

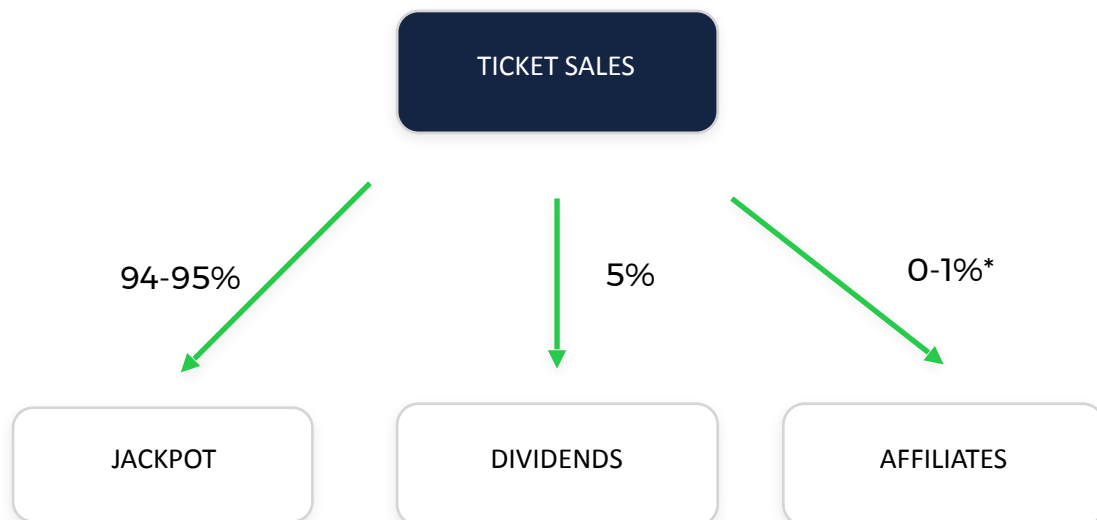
While the maximum bet value is set at 1 ETH, the Ticket value is not fixed and players are free to choose the value of their bet between 0,001 and 1 ETH. The maximum value of all tickets in one draw is set at 5 ETH for security reason.

## **H. ELIMINATION OF CHEATING OPPORTUNITIES**

The maximum cap (total bet value) for each draw is limited to 5 ETH for security reasons. This feature forecloses the possibility of potential cheating by miners.

## **I. FAVORABLE TICKET SALES FUNDS DISTRIBUTION**

94-95% of funds from Ticket sales are transferred directly to the Jackpot. When compared to the closest competitors, this is a far more favorable value for the product's growth and development. It guarantees the constant growth of the Jackpot and makes the SmartBillions increasingly more attractive for both players and PLAY Token holders.



\* Depending on the performance

## J. SMALL HOUSE EDGE

The house edge for the SmartBillions Ethereum-based lottery is set at 13.85%. Of that, 5% is allocated to the monthly dividend for limited PLAY Token holders and between 7,85% and 8,85% is transferred directly to the Jackpot. Up to 1% goes to affiliates depending on their performance. The house edge value is stated in the smart contract and it cannot be changed by the Administrator and any other 3<sup>rd</sup> party.

---

The competitive advantages of the SmartBillions Ethereum lottery compared to online and offline competitors:

	SMART BILLIONS	TRUE FLIP	POWERBALL
Max win	2 324 000 000 (2.34 billion) USD*	518 549 (518.5 thousand) USD**	52 800 000 (52.8 million) USD***
Jackpot payout	7 000 000 x	223 260 x	Volatile (26 400 000 x **)
Numbers range	Simply 0-15	Complex 1-49 and 1-26	Complex 1-69 and 1-26
Jackpot odds	1:16 777 216	1:49 578 984	1:292 201 338
Structure	Decentralised	Centralised	Centralised
Jackpot management	Smart Contract	Admin	Operator
House edge	13,85%	21,28%	>50 %
% of tickets sales to jackpot	95%	60%	50%
Transaction cost	0.03 \$	0.35 \$	Ticket purchase in person

\*1 ETH = 332 USD as for September 8 2017

\*\*As for September 08 2017 (<https://trueflip.io>)

\*\*\* As for September 04 2017 (<http://www.powerball.com/>)

## 5. MULTICHANNEL PROMOTION SCHEME

The SmartBillions marketing strategy and promotional plan is based on four pillars:

- a. An actively growing affiliate program, where 1% of the value of the Tickets sold will be transferred to affiliates driving the lottery's traffic. The constant increase in lottery popularity will have a positive effect on the growth of the value of the Tickets played. As a result, the referral program budget will grow dynamically in both value and attractiveness.
- b. Influencer engagement network initially built during the Token sales period. The SmartBillions influencer cooperation network will be further developed and used at later stages of the lottery's growth.
- c. Performance marketing tactics developed and continually evaluated for the highest conversion ratios.
- d. Informative and engaging publications in both blockchain and non-blockchain media worldwide – highlighting SmartBillions' role as the start of a new era in lottery history and potential for becoming the largest lottery in history.

The effectiveness of the marketing efforts will be driven by the ongoing growth of Jackpots offered. The lottery will be marketed and popularized not only within the blockchain community but throughout the general public.

## 6. CROWDFUNDING MODEL AND TOKENS

The SmartBillions limited PLAY Token sales model will use the "Initial Coin Offering" (ICO) process. PLAY Token sales will last for approximately two weeks. The crowdfund goal is set at 200 000 ETH. There is no minimal cap set.

Its uniqueness and revolutionary character is based on the following characteristics:

1. SmartBillions smart contract is finished and tested. The product is ready to launch unlike other ICO where the product development hasn't started yet.
2. 90% of the raised funds will be transferred directly to the lottery Jackpot.
3. Crowdfunding backers will receive almost 80% of the Tokens.



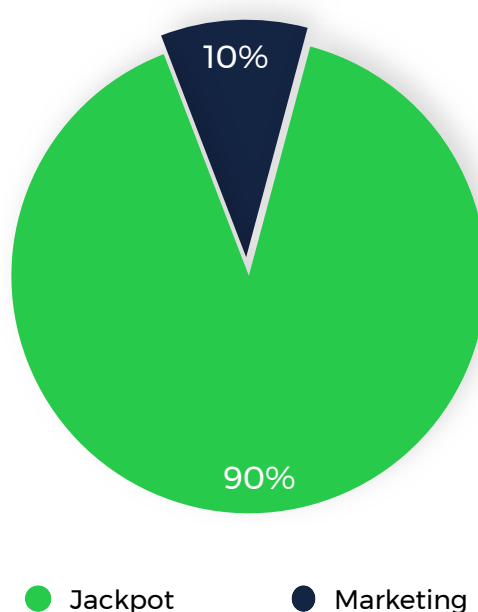
4. The first token sales with guaranteed downside protection for Investors and Token redemption possibility. The latter means that backers will have the unprecedented ability to redeem PLAY Tokens anytime – with most of their funds returned. This feature has never been offered in by any previous “ICOs” (initial coin offerings).
5. Monthly dividend payout directly from lottery Ticket sale revenues.
6. A ready to market product with full-feature operating ability, limiting project risk and need for long-term project participation.
7. Risk-free smart contract product with full security proven in the hackathon process (hackathon prize: 1500 ETH) provided by SmartBillions’ creators to validate the system’s safety).

## A.PLAY TOKEN SALES FUNDS DISTRIBUTION

The crowdfunding goal (cap) for the limited Token (PLAY Token) sales is set at 200,000 ETH. 90% of the raised sum will become the initial value of the Jackpot. The remaining 10% will be allocated for platform development and the lottery’s marketing budget to fuel the unceasing growth of the Jackpot value.

Unlike other “ICOs”, most of the funds – 90 % will be kept within the contract as the Jackpot funds, making the SmartBillions crowdfund highly attractive for the investors.

**Funds distribution from  
crowdfund (PLAY Token sales)**



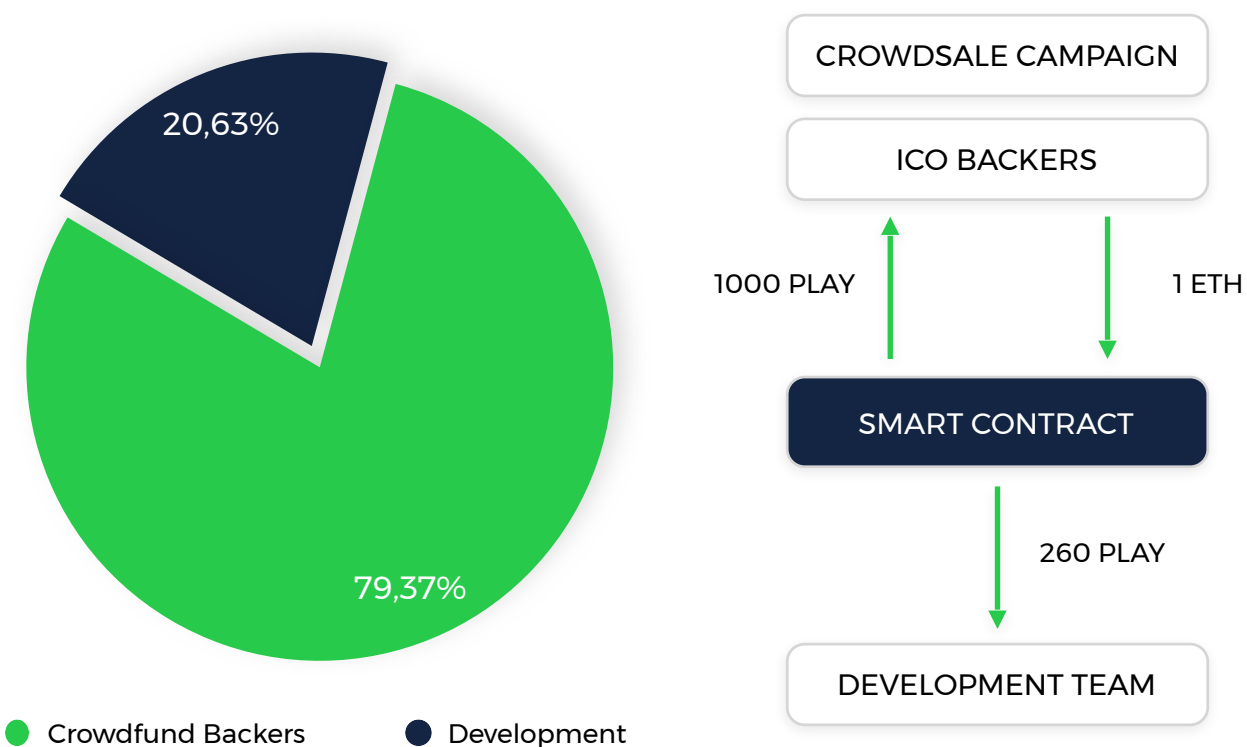
## B. TOKEN DISTRIBUTION

Crowdfund backers will receive 79.37 % and development team will receive 20.63 % off all PLAY Tokens created.

1 ETH = 1000 PLAY Tokens.

For every 100 PLAY Token created for crowdfunding backers, another 26 will be created for the development team (79.37% vs 20.63%). The goal is to create 252 000 000 PLAY Tokens. 200 000 000 of them allocated to the ICO Backers, and 52 000 000 to the development team.

**PLAY Token Distribution**



## C. PLAY TOKEN KEY FEATURES

SmartBillions Token (PLAY Token) is a standard ERC20 Token.

### a.Token redemption possibility.

Token redemption possibility is a truly revolutionary feature which has never been offered before in the history of ICOs.

Any time following the end of the limited PLAY Token sale, holders may redeem their Tokens. The funds raised for the Jackpot during the PLAY Tokens sale will be used to cover the cost of their redemption.

This feature reduces investors' risk to a minimum in the event that the lottery's performance falls below expectations or any other unexpected events affect the lottery. Token holders will be able to withdraw their funds and their potential loss will never exceed 28,57% of their initial investment. Holders redeeming their PLAY Tokens, will receive exactly 71,43% of their value in ETH based on the following calculation:

$$\frac{\text{Number of PLAY Tokens Redeemed}}{\text{Number of all PLAY Tokens}} \times \text{Initial Jackpot Value (ETH)}$$

Token redemption is facilitated through the *Disinvest function* (please see Appendix)

## **b. Token dividend payouts.**

Token holders will receive a dividend every 16 384 Ethereum blocks (approx. 1 month apart). A fixed 5% of all lottery income is allocated to the dividend payout. The payouts will be made directly from the smart contract balance which is inaccessible to all parties. Ticket sale revenues as well as the smart contract balance are public and cannot be affected by any third party. This makes SmartBillions different from other ICOs with Dividend payout promises where the payout value and the payout itself depends on the third-party actions and can be easily manipulated. The following calculation shows the value of the dividend for 1 PLAY Token each month:

$$\frac{\sum \text{ticket sales revenue in the given period} \times 5\%}{\text{Number of all PLAY Tokens}}$$

Dividend Payouts are facilitated through the *payDividends function* (please see Appendix).

PLAY Token Holders will be able to place an order for the dividend payout directly from their Ethereum wallet.

SmartBillions estimated annual dividend Value:

ANNUAL TICKET SALES USD	ANNUAL TICKET SALES ETH*	ANNUAL DIVIDEND USD	ANNUAL DIVI- DEND ETH	YIELD FOR TOKEN HOLDERS
400,000,000	1,333,333	20,000,000	66,667	26.46%
500,000,000	1,666,667	25,000,000	83,333	33.07%
750,000,000	2,500,000	37,500,000	125,000	49.60%

\*1 ETH = 300 USD as for Aug. 16 2017

### c. Token exchangeability

PLAY Tokens will be fully exchangeable and tradable on multiple markets. The SmartBillions development team plans to list the PLAY Tokens on multiple online exchange markets within 14 days from the conclusion of the initial token sales period.

## D. UNIQUENESS OF THE SMARTBILLIONS CROWDFUNDING INVESTMENT

The table below compares SmartBillions Crowdfunding with other recent lottery and gambling related ICOs:

	SMART BILLIONS	BITDICE	EDGELESS	TRUE FLIP
Raised funds storage	Smart contract - guaranteed security proven by hackathon	Internal Bitdice addresses - no guaranteed security	Internal Edgeless address - no guaranteed security	Internal True Flip address - no guaranteed security
Token Allocation	Immediately after receiving funds	7 days after crowd sales ends	Immediately after receiving funds	After the crowd-sale ends
Dividend Value	Transparent and public, calculated by smart contract with no 3rd party involvement	Depending on the Admin, Funds from internal address will be allocated to the Dividend payout address	Depending on the Admin, Funds from internal address will be allocated to the Dividend payout address	Depending on the Admin, Funds from internal address will be allocated to the Dividend payout address
Dividend payout	Managed by smart contract	Initiated by Admin	Initiated by Admin	Initiated by Admin
Funds allocated to Jackpot/ bankroll	90% lottery Jackpot	40% house bankroll	20% house bankroll	32% lottery Jackpot
Funds allocated to development team	10%	60%	80%	60%
ICO Backers' downside protection	Token redemption possibility	No protection	No protection	No protection

In comparison with its competitors, SmartBillions lottery is exhaustively transparent and fair. Its key uniqueness qualities are:

**Funds Storage:** All funds raised during the crowdsale will be only stored within the smart contract only - the contract's balance is public and transparent. Funds will be managed by a smart contract according to the rules set out prior to the ICO start and which cannot be changed afterward.

In the examined ICO models, funds are stored on multiple private addresses managed by Admin with no transparency. Moreover, only SmartBillions contract will be proven safe during the hackathon event which will take place before ICO starts, there's no guarantee that other projects funds are safe while stored on private addresses.

**Token Allocation:** All the PLAY Tokens will be automatically allocated by smart contract to the crowdsale backers immediately after the reception of the funds.

**Dividend value and payout:** All the SmartBillions lottery transactions are public and held on Ethereum blockchain. The dividend value allocated to Token holders represents 5% of all Ticket sales revenue in the last dividend period. The Token sales revenue is public and the dividend value is automatically calculated by the smart contract without any third-party involvement. The potential competitors' dividend value depends on their product performance and represents a share in the product profits. Those profits are not public, as they operate on the off-ledger solutions.

Admin may easily manipulate the dividend value and set its value at his own choice. Dividend payout also depends on the Admin only and Admin may decide not to pay Dividend at all.

**The use of funds:** The lion's share of the funds (90%) raised during the SmartBillions crowdfunding stage will be used for the lottery's Jackpot. Competitive ICOs assign just 20 to 40% of the raised funds to the house roll. Furthermore, only a maximum of 10% of the funds raised will be available to the development team for marketing and development processes, compared to a 60 to 80% figure in the other benchmarked gambling ICOs .

**Downside protection:** Unlike all other token offerings, the SmartBillions initial PLAY Tokens crowdfunding sale provides the investors with an unprecedented downside protection thanks to the one-of-a kind PLAY Tokens redemption possibility. If unforeseen circumstances cause the lottery's performance to fall below expectations or any other unexpected incident occurs, PLAY Token holders may redeem their Tokens and receive most of their funds back.

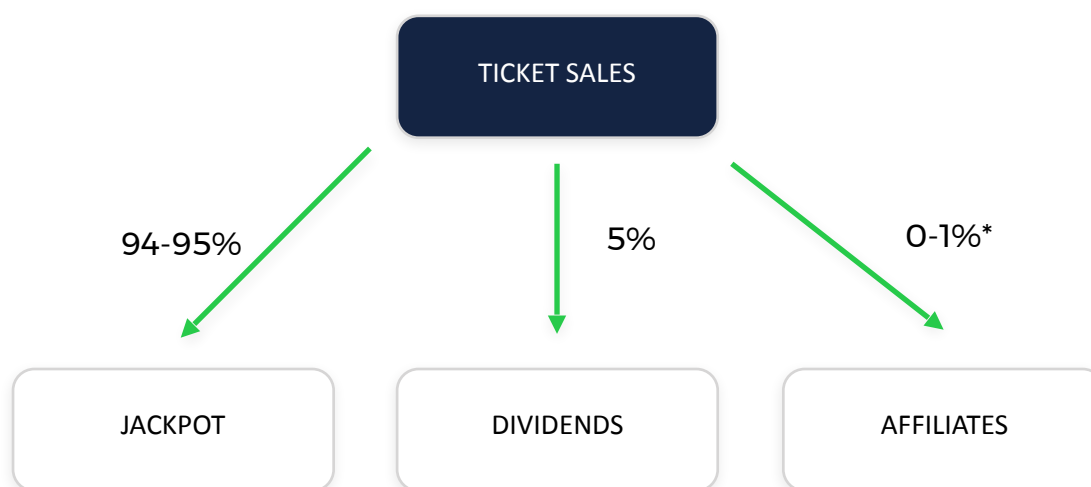
Other comparable projects may be shut down at any time and all funds are under 3<sup>rd</sup> party administration with no Investor control. Under that arrangement, worst case scenarios leave Investors with only worthless Tokens.

## 7. THE SMARTBILLIONS JACKPOT

The first SmartBillions Jackpot is derived from the proceeds of the PLAY token sales. 90% of the funds raised will constitute the initial SmartBillions Jackpot.

### a. Jackpot value

Between 94% and 95% of Ticket sales revenue will go toward the Jackpot. Another 5% is designated for PLAY Token holders' dividend payments and the remaining amount (up to 1%) will go toward compensating the lottery's affiliates.



\* Depending on the performance

This distribution of Ticket Sales revenues guarantees the Jackpot's continued growth and ensures that the SmartBillions lottery will constantly and dynamically grow in popularity.

## **b. Jackpot Management**

The Jackpot is managed by a smart contract with publicly known and unchangeable rules.

Admin will be allowed to withdraw only 0.25 % of the total Jackpot Value per week, for marketing expenses used to drive increased awareness, interest, participation and Jackpot value.

However, this is possible only if the overall Jackpot value is larger than the combined liabilities of potential redemptions of all PLAY Tokens in circulation as well as the current unpaid lottery winnings. Admin's withdraws of funds cannot harm the interests of the Investors and players.

## **c. Wins payouts**

If a particular player's win makes up more than 50% of the Jackpot value, that player can instantly withdraw only 50% of the Jackpot. After that, the player will be able to cash out up to 50% of the Jackpot value every period (approx. 1 month) until the withdrawal of the total value of the win. This provision ensures continued high Jackpot values, driving the lottery's appeal to the gaming clients.

The *walletBalanceOf* Function will inform about the lottery balance for the specific address (please see Appendix).

The *walletBlockOff* Function will inform about the soonest possible withdrawal date from the Jackpot for the specific address (please see Appendix).

# **8. SYSTEMIC SECURITY**

The SmartBillions lottery smart contract construction secures both the players' interests and the system's stability, bringing a high degree of trust to the online lottery world.

## **a. Hackathon**

Exactly 14 days before the initial PLAY Token sales offering, the SmartBillions development team will allocate 1500 ETH to the Jackpot as a reward to anyone capable of hacking the smart contract and withdrawing the funds. This own-risk solution will validate the contract's security.



## **b. Security audit**

The SmartBillions Ethereum contract was subjected to a third-party security audit. The contract was published online before the initial PLAY Token sales offering for testing. Additionally, the 1500 ETH prize available for hacking the smart contract is the best guarantee of the contract architecture's systemic security.

## **c. Lottery Closing**

In the event of unexpected circumstances (such as Ethereum protocol changes) which could affect the lottery and endanger the funds stored in the Jackpot, the contract allows Admin to withdraw surplus funds from the Jackpot remaining after settling all outstanding smart contract obligations, such as:

- Redemption of all remaining PLAY Tokens, and
- Payout of all outstanding lottery prizes.

Admin will be allowed to withdraw the surplus from the Jackpot only if more than 50% of all PLAY Tokens are redeemed. In this situation, it will be clear that lottery security might be vulnerable and that its funds must be secured. As a consequence, all the surplus funds will be allocated to the Jackpot of a new, comprehensively secured and updated lottery and the players and all relevant parties will be notified immediately and continuously updated during the process.

**There are no other circumstances under which funds from the Jackpot can be withdrawn by any party.**

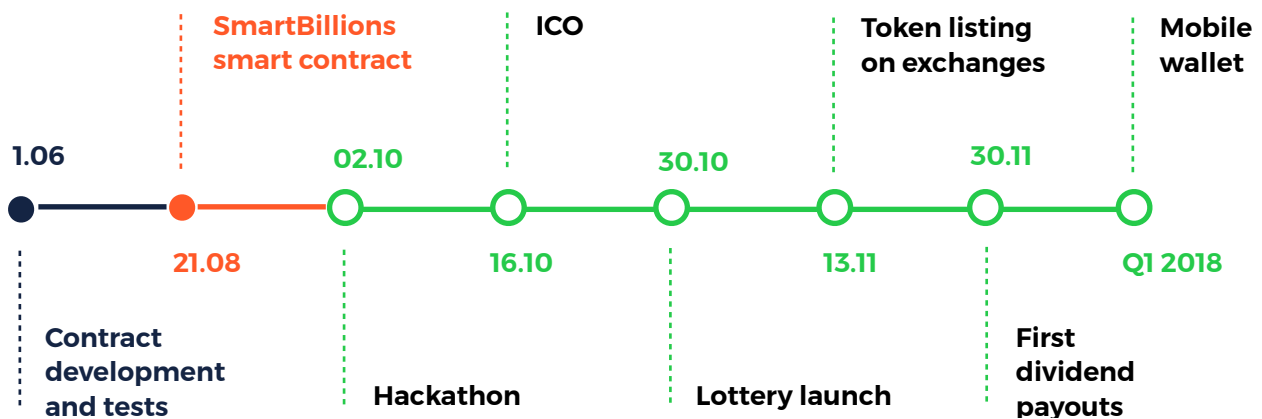
# **9. PROJECT TIMELINE**

The development phase of the SmartBillions Ethereum-based smart contract lottery has now ended. The contract was tested and the result of the third-party security audit was positive. The development team's efforts will now be directed to marketing, based on a strategy of identifying SmartBillions as the world's largest and best-known lottery with unlimited growth potential.

The team estimates that PLAY Tokens will be listed on multiple exchanges within 14 days after the initial PLAY Token sales offering ends.

In Q1 2018 The development team will launch a smart contract based exchange and PLAY Tokens will be first to be listed.

The SmartBillions team is working on a mobile Ethereum wallet designed to simplify lottery participation for mobile users. The release of the mobile wallet is planned for Q1 2018.



The development team will continue working on smart contract based products and will introduce more functionalities for PLAY tokens which will drive the price up.

In the near future the SmartBillions development team plans for the PLAY Token to become a universal global online gaming currency that will run across various gaming platforms.

## 10. APPENDIX:

### The SmartBillions smart contract.

```
1. pragma solidity ^0.4.13;
2.
3. library SafeMath {
4.   function sub(uint a, uint b) internal returns (uint) {
5.     assert(b <= a);
6.     return a - b;
7.   }
8.   function add(uint a, uint b) internal returns (uint) {
9.     uint c = a + b;
10.    assert(c >= a);
11.    return c;
12.  }
13.}
14.
15.contract ERC20Basic {
16.  uint public totalSupply;
17.  address public owner; //owner
18.  address public animator; //animator
19.  function balanceOf(address who) constant returns (uint);
20.  function transfer(address to, uint value);
21.  event Transfer(address indexed from, address indexed to, uint value);
22.  function commitDividend(address who) internal; // pays remaining dividend
23.}
24.
25.contract ERC20 is ERC20Basic {
26.  function allowance(address owner, address spender) constant returns (uint);
27.  function transferFrom(address from, address to, uint value);
28.  function approve(address spender, uint value);
29.  event Approval(address indexed owner, address indexed spender, uint value);
30.}
31.
32.contract BasicToken is ERC20Basic {
33.  using SafeMath for uint;
34.  mapping(address => uint) balances;
35.
36.  modifier onlyPayloadSize(uint size) {
37.    assert(msg.data.length >= size + 4);
38.    _;
39.  }
40.  /**
41.   * @dev transfer token for a specified address
42.   * @param _to The address to transfer to.
43.   * @param _value The amount to be transferred.
44.   */
45.  function transfer(address _to, uint _value) onlyPayloadSize(2 * 32) {
46.    commitDividend(msg.sender);
```

```

47. balances[msg.sender] = balances[msg.sender].sub(_value);
48. if(_to == address(this)) {
49.     commitDividend(owner);
50.     balances[owner] = balances[owner].add(_value);
51.     Transfer(msg.sender, owner, _value);
52. }
53. else {
54.     commitDividend(_to);
55.     balances[_to] = balances[_to].add(_value);
56.     Transfer(msg.sender, _to, _value);
57. }
58. }
59. /**
60. * @dev Gets the balance of the specified address.
61. * @param _owner The address to query the the balance of.
62. * @return An uint representing the amount owned by the passed address.
63. */
64. function balanceOf(address _owner) constant returns (uint balance) {
65.     return balances[_owner];
66. }
67.}
68.
69.contract StandardToken is BasicToken, ERC20 {
70. mapping (address => mapping (address => uint)) allowed;
71.
72. /**
73. * @dev Transfer tokens from one address to another
74. * @param _from address The address which you want to send tokens from
75. * @param _to address The address which you want to transfer to
76. * @param _value uint the amout of tokens to be transfered
77. */
78. function transferFrom(address _from, address _to, uint _value) onlyPayloadSize(3 * 32) {
79.     var _allowance = allowed[_from][msg.sender];
80.     commitDividend(_from);
81.     commitDividend(_to);
82.     balances[_to] = balances[_to].add(_value);
83.     balances[_from] = balances[_from].sub(_value);
84.     allowed[_from][msg.sender] = _allowance.sub(_value);
85.     Transfer(_from, _to, _value);
86. }
87. /**
88. * @dev Approve the passed address to spend the specified amount of tokens on behalf of msg-
    g.sender.
89. * @param _spender The address which will spend the funds.
90. * @param _value The amount of tokens to be spent.
91. */
92. function approve(address _spender, uint _value) {
93.     // https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
94.     assert(!((_value != 0) && (allowed[msg.sender][_spender] != 0)));
95.     allowed[msg.sender][_spender] = _value;
96.     Approval(msg.sender, _spender, _value);
97. }

```

```

98. /**
99.  * @dev Function to check the amount of tokens than an owner allowed to a spender.
100.  * @param _owner address The address which owns the funds.
101.  * @param _spender address The address which will spend the funds.
102.  * @return A uint specifying the amount of tokens still available for the spender.
103.  */
104. function allowance(address _owner, address _spender) constant returns (uint remaining) {
105.     return allowed[_owner][_spender];
106. }
107.}
108.
109./**
110. * @title SmartBillions contract
111. */
112.contract SmartBillions is StandardToken {
113.
114.    // metadata
115.    string public constant name = "SmartBillions Token";
116.    string public constant symbol = "PLAY";
117.    uint public constant decimals = 0;
118.
119.    // contract state
120.    struct Wallet {
121.        uint208 balance; // current balance of user
122.        uint16 lastDividendPeriod; // last processed dividend period of user's tokens
123.        uint32 nextWithdrawBlock; // next withdrawal possible after this block number
124.    }
125.    mapping (address => Wallet) wallets;
126.    struct Bet {
127.        uint192 value; // bet size
128.        uint32 betHash; // selected numbers
129.        uint32 blockNum; // blocknumber when lottery runs
130.    }
131.    mapping (address => Bet) bets;
132.
133.    uint public walletBalance = 0; // sum of funds in wallets
134.
135.    // investment parameters
136.    uint public investStart = 1; // investment start block, 0: closed, 1: preparation
137.    uint public investBalance = 0; // funding from investors
138.    uint public investBalanceMax = 200000 ether; // maximum funding
139.    uint public dividendPeriod = 1;
140.    uint[] public dividends; // dividends collected per period, growing array
141.
142.    // betting parameters
143.    uint public maxWin = 0; // maximum prize won
144.    uint public hashFirst = 0; // start time of building hashes database
145.    uint public hashLast = 0; // last saved block of hashes
146.    uint public hashNext = 0; // next available bet block.number
147.    uint public hashBetSum = 0; // used bet volume of next block
148.    uint public hashBetMax = 5 ether; // maximum bet size per block
149.    uint[] public hashes; // space for storing lottery results

```

```

150.
151. // constants
152. //uint public constant hashesSize = 1024 ; // DEBUG ONLY !!!
153. uint public constant hashesSize = 16384 ; // 30 days of blocks
154. uint public coldStoreLast = 0 ; // block of last cold store transfer
155.
156. // events
157. event LogBet(address indexed player, uint bethash, uint blocknumber, uint betsize);
158. event LogLoss(address indexed player, uint bethash, uint hash);
159. event LogWin(address indexed player, uint bethash, uint hash, uint prize);
160. event LogInvestment(address indexed investor, address indexed partner, uint amount);
161. event LogRecordWin(address indexed player, uint amount);
162. event LogLate(address indexed player,uint playerBlockNumber,uint currentBlockNumber);
163. event LogDividend(address indexed investor, uint amount, uint period);
164.
165. modifier onlyOwner() {
166.     assert(msg.sender == owner);
167.     _;
168. }
169.
170. modifier onlyAnimator() {
171.     assert(msg.sender == animator);
172.     _;
173. }
174.
175. // constructor
176. function SmartBillions() {
177.     owner = msg.sender;
178.     animator = msg.sender;
179.     wallets[owner].lastDividendPeriod = uint16(dividendPeriod);
180.     dividends.push(0); // not used
181.     dividends.push(0); // current dividend
182. }
183.
184. /* getters */
185.
186. /**
187.  * @dev Show length of allocated swap space
188.  */
189. function hashesLength() constant external returns (uint) {
190.     return uint(hashes.length);
191. }
192.
193. /**
194.  * @dev Show balance of wallet
195.  * @param _owner The address of the account.
196.  */
197. function walletBalanceOf(address _owner) constant external returns (uint) {
198.     return uint(wallets[_owner].balance);
199. }
200.
201. /**

```

```

202.  * @dev Show last dividend period processed
203.  * @param _owner The address of the account.
204.  */
205.  function walletPeriodOf(address _owner) constant external returns (uint) {
206.      return uint(wallets[_owner].lastDividendPeriod);
207.  }
208.
209.  /**
210.   * @dev Show block number when withdraw can continue
211.   * @param _owner The address of the account.
212.   */
213.  function walletBlockOf(address _owner) constant external returns (uint) {
214.      return uint(wallets[_owner].nextWithdrawBlock);
215.  }
216.
217.  /**
218.   * @dev Show bet size.
219.   * @param _owner The address of the player.
220.   */
221.  function betValueOf(address _owner) constant external returns (uint) {
222.      return uint(bets[_owner].value);
223.  }
224.
225.  /**
226.   * @dev Show block number of lottery run for the bet.
227.   * @param _owner The address of the player.
228.   */
229.  function betHashOf(address _owner) constant external returns (uint) {
230.      return uint(bets[_owner].betHash);
231.  }
232.
233.  /**
234.   * @dev Show block number of lottery run for the bet.
235.   * @param _owner The address of the player.
236.   */
237.  function betBlockNumberOf(address _owner) constant external returns (uint) {
238.      return uint(bets[_owner].blockNum);
239.  }
240.
241.  /**
242.   * @dev Print number of block till next expected dividend payment
243.   */
244.  function dividendsBlocks() constant external returns (uint) {
245.      if(investStart > 0) {
246.          return(0);
247.      }
248.      uint period = (block.number - hashFirst) / (10 * hashesSize);
249.      if(period > dividendPeriod) {
250.          return(0);
251.      }
252.      return((10 * hashesSize) - ((block.number - hashFirst) % (10 * hashesSize)));
253.  }

```

```

254.
255./* administrative functions */
256.
257. /**
258.  * @dev Change owner.
259.  * @param _who The address of new owner.
260.  */
261. function changeOwner(address _who) external onlyOwner {
262.     assert(_who != address(0));
263.     commitDividend(msg.sender);
264.     commitDividend(_who);
265.     owner = _who;
266. }
267.
268. /**
269.  * @dev Change animator.
270.  * @param _who The address of new animator.
271.  */
272. function changeAnimator(address _who) external onlyAnimator {
273.     assert(_who != address(0));
274.     commitDividend(msg.sender);
275.     commitDividend(_who);
276.     animator = _who;
277. }
278.
279. /**
280.  * @dev Set ICO Start block.
281.  * @param _when The block number of the ICO.
282.  */
283. function setInvestStart(uint _when) external onlyOwner {
284.     require(investStart == 1 && hashFirst > 0 && block.number < _when);
285.     investStart = _when;
286. }
287.
288. /**
289.  * @dev Set maximum bet size per block
290.  * @param _maxsum The maximum bet size in wei.
291.  */
292. function setBetMax(uint _maxsum) external onlyOwner {
293.     hashBetMax = _maxsum;
294. }
295.
296. /**
297.  * @dev Reset bet size accounting, to increase bet volume above safe limits
298.  */
299. function resetBet() external onlyOwner {
300.     hashNext = block.number + 3;
301.     hashBetSum = 0;
302. }
303.
304. /**
305.  * @dev Move funds to cold storage

```



```

306.  * @dev investBalance and walletBalance is protected from withdraw by owner
307.  * @dev if funding is > 50% admin can withdraw only 0.25% of balance weakly
308.  * @param _amount The amount of wei to move to cold storage
309.  */
310.  function coldStore(uint _amount) external onlyOwner {
311.      houseKeeping();
312.      require(_amount > 0 && this.balance >= (investBalance * 9 / 10) + walletBalance + _amount);
313.      if(investBalance >= investBalanceMax / 2){ // additional jackpot protection
314.          require((_amount <= this.balance / 400) && coldStoreLast + 4 * 60 * 24 * 7 <= block.number);
315.      }
316.      msg.sender.transfer(_amount);
317.      coldStoreLast = block.number;
318.  }
319.
320.  /**
321.   * @dev Move funds to contract jackpot
322.   */
323.   function hotStore() payable external {
324.       houseKeeping();
325.   }
326.
327.  /* housekeeping functions */
328.
329.  /**
330.   * @dev Update accounting
331.   */
332.   function houseKeeping() public {
333.       if(investStart > 1 && block.number >= investStart + (hashesSize * 5)){ // ca. 14 days
334.           investStart = 0; // start dividend payments
335.       }
336.       else {
337.           if(hashFirst > 0){
338.               uint period = (block.number - hashFirst) / (10 * hashesSize );
339.               if(period > dividends.length - 2) {
340.                   dividends.push(0);
341.               }
342.               if(period > dividendPeriod && investStart == 0 && dividendPeriod < dividends.length - 1) {
343.                   dividendPeriod++;
344.               }
345.           }
346.       }
347.   }
348.
349.  /* payments */
350.
351.  /**
352.   * @dev Pay balance from wallet
353.   */
354.   function payWallet() public {
355.       if(wallets[msg.sender].balance > 0 && wallets[msg.sender].nextWithdrawBlock <= block.num-
           ber){
356.           uint balance = wallets[msg.sender].balance;

```

```

357.     wallets[msg.sender].balance = 0;
358.     walletBalance -= balance;
359.     pay(balance);
360. }
361. }
362.
363. function pay(uint _amount) private {
364.     uint maxpay = this.balance / 2;
365.     if(maxpay >= _amount) {
366.         msg.sender.transfer(_amount);
367.         if(_amount > 1 finney) {
368.             houseKeeping();
369.         }
370.     }
371.     else {
372.         uint keepbalance = _amount - maxpay;
373.         walletBalance += keepbalance;
374.         wallets[msg.sender].balance += uint208(keepbalance);
375.         wallets[msg.sender].nextWithdrawBlock = uint32(block.number + 4 * 60 * 24 * 30); // wait 1
            month for more funds
376.         msg.sender.transfer(maxpay);
377.     }
378. }
379.
380. /* investment functions */
381.
382. /**
383.  * @dev Buy tokens
384.  */
385. function investDirect() payable external {
386.     invest(owner);
387. }
388.
389. /**
390.  * @dev Buy tokens with affiliate partner
391.  * @param _partner Affiliate partner
392.  */
393. function invest(address _partner) payable public {
394.     //require(fromUSA()==false); // fromUSA() not yet implemented :(
395.     require(investStart > 1 && block.number < investStart + (hashesSize * 5) && investBalance < in-
        vestBalanceMax);
396.     uint investing = msg.value;
397.     if(investing > investBalanceMax - investBalance) {
398.         investing = investBalanceMax - investBalance;
399.         investBalance = investBalanceMax;
400.         investStart = 0; // close investment round
401.         msg.sender.transfer(msg.value.sub(investing)); // send back funds immediately
402.     }
403.     else{
404.         investBalance += investing;
405.     }
406.     if(_partner == address(0) || _partner == owner){

```

```

407.     walletBalance += investing / 10;
408.     wallets[owner].balance += uint208(investing / 10); // 10% for marketing if no affiliates
409.     else{
410.         walletBalance += (investing * 5 / 100) * 2;
411.         wallets[owner].balance += uint208(investing * 5 / 100); // 5% initial marketing funds
412.         wallets[_partner].balance += uint208(investing * 5 / 100); // 5% for affiliates
413.         wallets[msg.sender].lastDividendPeriod = uint16(dividendPeriod); // assert(dividendPeriod == 1);
414.         uint senderBalance = investing / 10**15;
415.         uint ownerBalance = investing * 16 / 10**17 ;
416.         uint animatorBalance = investing * 10 / 10**17 ;
417.         balances[msg.sender] += senderBalance;
418.         balances[owner] += ownerBalance ; // 13% of shares go to developers
419.         balances[animator] += animatorBalance ; // 8% of shares go to animator
420.         totalSupply += senderBalance + ownerBalance + animatorBalance;
421.         Transfer(address(0),msg.sender,senderBalance); // for etherscan
422.         Transfer(address(0),owner,ownerBalance); // for etherscan
423.         Transfer(address(0),animator,animatorBalance); // for etherscan
424.         LogInvestment(msg.sender,_partner,investing);
425.     }
426.
427. /**
428.  * @dev Delete all tokens owned by sender and return unpaid dividends and 90% of initial in-
vestmment
429.  */
430. function disinvest() external {
431.     require(investStart == 0);
432.     commitDividend(msg.sender);
433.     uint initialInvestment = balances[msg.sender] * 10**15;
434.     Transfer(msg.sender,address(0),balances[msg.sender]); // for etherscan
435.     delete balances[msg.sender]; // totalSupply stays the same, investBalance is reduced
436.     investBalance -= initialInvestment;
437.     wallets[msg.sender].balance += uint208(initialInvestment * 9 / 10);
438.     payWallet();
439. }
440.
441. /**
442.  * @dev Pay unpaid dividends
443.  */
444. function payDividends() external {
445.     require(investStart == 0);
446.     commitDividend(msg.sender);
447.     payWallet();
448. }
449.
450. /**
451.  * @dev Commit remaining dividends before transfer of tokens
452.  */
453. function commitDividend(address _who) internal {
454.     uint last = wallets[_who].lastDividendPeriod;
455.     if((balances[_who]==0) || (last==0)){
456.         wallets[_who].lastDividendPeriod=uint16(dividendPeriod);
457.         return;

```

```

458.     }
459.     if(last==dividendPeriod) {
460.         return;
461.     }
462.     uint share = balances[_who] * 0xffffffff / totalSupply;
463.     uint balance = 0;
464.     for(;last<dividendPeriod;last++) {
465.         balance += share * dividends[last];
466.     }
467.     balance = (balance / 0xffffffff);
468.     walletBalance += balance;
469.     wallets[_who].balance += uint208(balance);
470.     wallets[_who].lastDividendPeriod = uint16(last);
471.     LogDividend(_who,balance,last);
472. }
473.
474. /* lottery functions */
475.
476. function betPrize(Bet _player, uint24 _hash) constant private returns (uint) { // house fee 13.85%
477.     uint24 bethash = uint24(_player.betHash);
478.     uint24 hit = bethash ^ _hash;
479.     uint24 matches =
480.         ((hit & 0xF) == 0 ? 1 : 0 ) +
481.         ((hit & 0xF0) == 0 ? 1 : 0 ) +
482.         ((hit & 0xF00) == 0 ? 1 : 0 ) +
483.         ((hit & 0xF000) == 0 ? 1 : 0 ) +
484.         ((hit & 0xF0000) == 0 ? 1 : 0 ) +
485.         ((hit & 0xF00000) == 0 ? 1 : 0 );
486.     if(matches == 6){
487.         return(uint(_player.value) * 7000000);
488.     }
489.     if(matches == 5){
490.         return(uint(_player.value) * 20000);
491.     }
492.     if(matches == 4){
493.         return(uint(_player.value) * 500);
494.     }
495.     if(matches == 3){
496.         return(uint(_player.value) * 25);
497.     }
498.     if(matches == 2){
499.         return(uint(_player.value) * 3);
500.     }
501.     return(0);
502. }
503.
504. /**
505.  * @dev Check if won in lottery
506.  */
507. function betOf(address _who) constant external returns (uint) {
508.     Bet memory player = bets[_who];
509.     if( (player.value==0) ||

```

```

510.     (player.blockNum<=1) ||
511.     (block.number<player.blockNum) ||
512.     (block.number>=player.blockNum + (10 * hashesSize))){
513.     return(0);
514. }
515. if(block.number<player.blockNum+256){
516.     return(betPrize(player,uint24(block.blockhash(player.blockNum))));
517. }
518. if(hashFirst>0){
519.     uint32 hash = getHash(player.blockNum);
520.     if(hash == 0x1000000) { // load hash failed :-(), return funds
521.         return(uint(player.value));
522.     }
523.     else{
524.         return(betPrize(player,uint24(hash)));
525.     }
526. }
527. return(0);
528. }
529.
530. /**
531.  * @dev Check if won in lottery
532.  */
533. function won() public {
534.     Bet memory player = bets[msg.sender];
535.     if(player.blockNum==0){ // create a new player
536.         bets[msg.sender] = Bet({value: 0, betHash: 0, blockNum: 1});
537.         return;
538.     }
539.     if((player.value==0) || (player.blockNum==1)){
540.         payWallet();
541.         return;
542.     }
543.     require(block.number>player.blockNum); // if there is an active bet, throw()
544.     if(player.blockNum + (10 * hashesSize) <= block.number){ // last bet too long ago, lost !
545.         LogLate(msg.sender,player.blockNum,block.number);
546.         bets[msg.sender] = Bet({value: 0, betHash: 0, blockNum: 1});
547.         return;
548.     }
549.     uint prize = 0;
550.     uint32 hash = 0;
551.     if(block.number<player.blockNum+256){
552.         hash = uint24(block.blockhash(player.blockNum));
553.         prize = betPrize(player,uint24(hash));
554.     }
555.     else {
556.         if(hashFirst>0){ // lottery is open even before swap space (hashes) is ready, but player must
            collect results within 256 blocks after run
557.             hash = getHash(player.blockNum);
558.             if(hash == 0x1000000) { // load hash failed :-(), return funds
559.                 prize = uint(player.value);
560.             }

```

```

561.         else{
562.             prize = betPrize(player,uint24(hash));
563.         }
564.     }
565.     else{
566.         LogLate(msg.sender,player.blockNum,block.number);
567.         bets[msg.sender] = Bet({value: 0, betHash: 0, blockNum: 1});
568.         return();
569.     }
570. }
571. bets[msg.sender] = Bet({value: 0, betHash: 0, blockNum: 1});
572. if(prize>0) {
573.     LogWin(msg.sender,uint(player.betHash),uint(hash),prize);
574.     if(prize > maxWin){
575.         maxWin = prize;
576.         LogRecordWin(msg.sender,prize);
577.     }
578.     pay(prize);
579. }
580. else{
581.     LogLoss(msg.sender,uint(player.betHash),uint(hash));
582. }
583. }
584.
585. /**
586.  * @dev Send ether to buy tokens during ICO
587.  * @dev or send less than 1 ether to contract to play
588.  * @dev or send 0 to collect prize
589.  */
590. function () payable external {
591.     if(msg.value > 0){
592.         if(investStart>1){ // during ICO payment to the contract is treated as investment
593.             invest(owner);
594.         }
595.         else{ // if not ICO running payment to contract is treated as play
596.             play();
597.         }
598.         return;
599.     }
600.     //check for dividends and other assets
601.     if(investStart == 0 && balances[msg.sender]>0){
602.         commitDividend(msg.sender);}
603.     won(); // will run payWallet() if nothing else available
604. }
605.
606. /**
607.  * @dev Play in lottery
608.  */
609. function play() payable public returns (uint) {
610.     return playSystem(uint(sha3(msg.sender,block.number)), address(0));
611. }
612.

```

```

613. /**
614.  * @dev Play in lottery with random numbers
615.  * @param _partner Affiliate partner
616.  */
617. function playRandom(address _partner) payable public returns (uint) {
618.     return playSystem(uint(sha3(msg.sender,block.number)), _partner);
619. }
620.
621. /**
622.  * @dev Play in lottery with own numbers
623.  * @param _partner Affiliate partner
624.  */
625. function playSystem(uint _hash, address _partner) payable public returns (uint) {
626.     won(); // check if player did not win
627.     uint24 bethash = uint24(_hash);
628.     require(msg.value <= 1 ether && msg.value < hashBetMax);
629.     if(msg.value > 0){
630.         if(investStart==0) { // dividends only after investment finished
631.             dividends[dividendPeriod] += msg.value / 20; // 5% dividend
632.         }
633.         if(_partner != address(0)) {
634.             uint fee = msg.value / 100;
635.             walletBalance += fee;
636.             wallets[_partner].balance += uint208(fee); // 1% for affiliates
637.         }
638.         if(hashNext < block.number + 3) {
639.             hashNext = block.number + 3;
640.             hashBetSum = msg.value;
641.         }
642.         else{
643.             if(hashBetSum > hashBetMax) {
644.                 hashNext++;
645.                 hashBetSum = msg.value;
646.             }
647.             else{
648.                 hashBetSum += msg.value;
649.             }
650.         }
651.         bets[msg.sender] = Bet({value: uint192(msg.value), betHash: uint32(bethash), blockNum: uint32(hashNext)});
652.         LogBet(msg.sender,uint(bethash),hashNext,msg.value);
653.     }
654.     putHash(); // players help collecting data
655.     return(hashNext);
656. }
657.
658. /* database functions */
659.
660. /**
661.  * @dev Create hash data swap space
662.  * @param _sadd Number of hashes to add (<=256)
663.  */

```

```

664. function addHashes(uint _sadd) public returns (uint) {
665.     require(hashFirst == 0 && _sadd > 0 && _sadd <= hashesSize);
666.     uint n = hashes.length;
667.     if(n + _sadd > hashesSize){
668.         hashes.length = hashesSize;
669.     }
670.     else{
671.         hashes.length += _sadd;
672.     }
673.     for(;n<hashes.length;n++){ // make sure to burn gas
674.         hashes[n] = 1;
675.     }
676.     if(hashes.length>=hashesSize) { // assume block.number > 10
677.         hashFirst = block.number - ( block.number % 10);
678.         hashLast = hashFirst;
679.     }
680.     return(hashes.length);
681. }
682.
683. /**
684.  * @dev Create hash data swap space, add 128 hashes
685.  */
686. function addHashes128() external returns (uint) {
687.     return(addHashes(128));
688. }
689.
690. function calcHashes(uint32 _lastb, uint32 _delta) constant private returns (uint) {
691.     return( ( uint(block.blockhash(_lastb )) & 0xFFFFFFF )
692.         | ( ( uint(block.blockhash(_lastb+1)) & 0xFFFFFFF ) << 24 )
693.         | ( ( uint(block.blockhash(_lastb+2)) & 0xFFFFFFF ) << 48 )
694.         | ( ( uint(block.blockhash(_lastb+3)) & 0xFFFFFFF ) << 72 )
695.         | ( ( uint(block.blockhash(_lastb+4)) & 0xFFFFFFF ) << 96 )
696.         | ( ( uint(block.blockhash(_lastb+5)) & 0xFFFFFFF ) << 120 )
697.         | ( ( uint(block.blockhash(_lastb+6)) & 0xFFFFFFF ) << 144 )
698.         | ( ( uint(block.blockhash(_lastb+7)) & 0xFFFFFFF ) << 168 )
699.         | ( ( uint(block.blockhash(_lastb+8)) & 0xFFFFFFF ) << 192 )
700.         | ( ( uint(block.blockhash(_lastb+9)) & 0xFFFFFFF ) << 216 )
701.         | ( ( uint(_delta) / hashesSize) << 240));
702. }
703.
704. function getHash(uint _block) constant private returns (uint32) {
705.     uint delta = (_block - hashFirst) / 10;
706.     uint hash = hashes[delta % hashesSize];
707.     if(delta / hashesSize != hash >> 240) {
708.         return(0x10000000); // load failed, incorrect data in hashes
709.     }
710.     uint slotp = (_block - hashFirst) % 10;
711.     return(uint32((hash >> (24 * slotp)) & 0xFFFFFFF));
712. }
713.
714. /**
715.  * @dev Fill hash data

```



```

716.  */
717.  function putHash() public returns (bool) {
718.      uint lastb = hashLast;
719.      if(lastb == 0 || block.number <= lastb + 10) {
720.          return(false);
721.      }
722.      uint blockn256;
723.      if(block.number<256) { // useless test for testnet :(
724.          blockn256 = 0;
725.      }
726.      else{
727.          blockn256 = block.number - 256;
728.      }
729.      if(lastb < blockn256) {
730.          uint num = blockn256;
731.          num += num % 10;
732.          lastb = num;
733.      }
734.      uint delta = (lastb - hashFirst) / 10;
735.      hashes[delta % hashesSize] = calcHashes(uint32(lastb),uint32(delta));
736.      hashLast = lastb + 10;
737.      return(true);
738.  }
739.
740.  /**
741.   * @dev Fill hash data many times
742.   * @param _num Number of iterations
743.   */
744.  function putHashes(uint _num) external {
745.      uint n=0;
746.      for(;n<_num;n++){
747.          if(!putHash()){
748.              return;
749.          }
750.      }
751.  }
752.
753.}

```

