SKILLONOMY MODEL

The model is a description of the movement of tokens by period and distribution among agents. Agents are entities involved in skill-mining, profit-making, manipulating tokens. All data on agents and possible predicted and initial distribution of tokens are recorded in the environment description file. Each agent at a certain time does some action and changes the environment, namely the distribution of tokens. Actions are carried out in accordance with the behavior presented in a formal form in the file of behaviors. With this description, it is possible to simulate the movement of tokens and explore the properties of the entire project.

The model consists of an informal description of agent types and the corresponding attributes, formal actions, and a pattern of general behavior. The description is built on the principle of insertion modeling: if the condition of an agent performing an action is true, then a corresponding change in the environment is allowed. The appendices describe the formal parts of the model in the language of behavior algebra.

List roles in model:

1. Students:
   1.1. Excellent students.
   1.2. Good students.
   1.3. Good with minus students.
   1.4. Trey students.
   1.5. Two students.
   1.6. One students.
2. Coaches.
3. Managers.
4. Owners nodes.
5. Owner platform
6. Holders.
7. Stock exchange.

For details on the list of agent attributes and actions in which they participate, see below in Table #1.

Table #1

AGENT TYPE	**AGENT ATTRIBUTES**
Excellent students	● **income** - the number of tokens available; ● **tokenLocked** - the number of locked tokens; ● **tokenAvailable** - the number of tokens available;
Good students	● **tokenAvailable** - the number of tokens available; ● **tokenLocked** - the number of locked tokens;
Good with minus students	● **tokenAvailable** - the number of tokens available; ● **tokenLocked** - the number of locked tokens;

Trey students	• **tokenAvailable** - the number of tokens available; • **tokenLocked** - the number of locked tokens;
Two students	• **tokenAvailable** - the number of tokens available; • **tokenLocked** - the number of locked tokens;
One students	• **tokenAvailable** - the number of tokens available; • **tokenLocked** - the number of locked tokens;
Coaches	• **tokenAvailable** - the number of tokens available; • **tokenLocked** - the number of locked tokens;
Managers	• **tokenAvailable** - the number of tokens available; • **tokenLocked** - the number of locked tokens;
Owners nodes	• **tokenAvailable**- the number of tokens available; • **tokenSkillMining**- the number of tokens for the general skillmining pool; • **nodeReserve**- reserve fund of node owner; • **tokenLocked** - he number of locked tokens; • **income** - the income of the owner in the conventional units;
Owner platform	• **SMInvestToken** - planned quantity of tokens for skillmining from the platform for the corresponding period; • **tokenICOStageEmission**- the planned investment of tokens in each period; • **tokenICOStageForSale**- the number of emitted tokens for sale in each respective period; • **CommonEmission** - the total amount of emissions in tokens, reserved by the platform throughout the project period; • **RESERVE** - the number of tokens in the reserve fund; • **tokenSkillMining** - the current number of tokens obtained from skillmining monthly

	• **income** - the income of the owner in the conventional units; • **holderEmission** - relevant emissions for Holders to the respective period
Holders	• **tokenAvailable** - the number of tokens available; • **tokenLocked**- the number of locked tokens;
Stock exchange.	• **bought** -the number of tokens purchased per month. • **sold**- the number of tokens sold per month; • **tokens**- the number of available tokens on the exchange •

STUDENTS GROUP	MINING RATE	PERCENTAGE DISTRIBUTION OF STUDENT GROUPS
Excellent students	1.1	10%
Good students.	1.0	15%
Good with minus students	0.9	25%
Trey students	0.8	25%
Two students	0.7	15%
One students	0.1	10%
		100%

ATRIBUT	VALUE	DESCRIPTION
FUNDS	20	Count funds
allMonths	43	Number of months of the project
basicNeeds	0	Cost of skillmining, dependent on the planned number of skillmining
coefSoundnessDownBuy	0.1	The ratio signaling the purchase at a lower price.

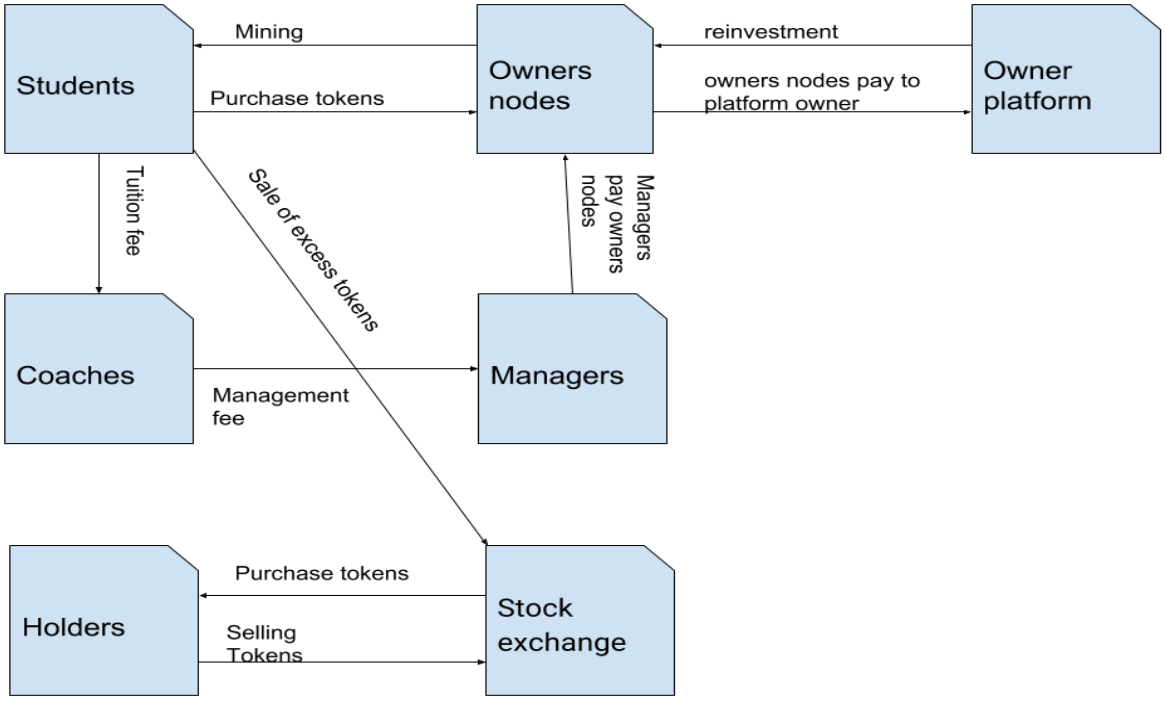
coefSoundnessDownSell	0.1	The ratio signaling the sale at a lower price
coefSoundnessUpBuy	0.05	The ratio signaling the purchase with a price increase
coefSoundnessUpSell	0.05	The ratio signaling the sale at a lower price
fundPrice	20	Price of funds
holdRation2	0.1	Unlock SEED tokens by investors. 60% - are locked up for 4 months and then unlocked at 10% per month
holderEmssionMonth	1. holderEmissionMonth(1)=1; 2. holderEmissionMonth(2)=2; 3. holderEmissionMonth(3)=3; 4. holderEmissionMonth(4)=4; 5. holderEmissionMonth(5)=5; 6. holderEmissionMonth(6)=6; 7. holderEmissionMonth(7) =7;	Emission periods for agents "holders". 1. holderEmissionMonth(1)=1; 2. holderEmissionMonth(2)=2; 3. holderEmissionMonth(3)=3; 4. holderEmissionMonth(4)=4; 5. holderEmissionMonth(5)=5; 6. holderEmissionMonth(6)=6; 7. holderEmissionMonth(7) =7;
holderRation	0.3	The value of relocked tokens from holders
holdersUnlock		Global attribute of quantities of locked tokens for agent "holders"
koefNeeds	0.9	The utilization rate of the reserve platforms fund for the skillmining.
koefSM	0.5	What part of the tokens need to spend in order to mine 1
koefSMWork_exelent **koefSMWork_good** **koefSMWork_goodm** **koefSMWork_one** **koefSMWork_trey** **koefSMWork_two**	1.1	Coefficient of received tokens for agent "students excellent" from total number invested tokens
	1.0	Coefficient of received tokens for agent "students good" from total number invested tokens
	0.9	Coefficient of received tokens for agent "students good with minus" from total number invested tokens
	0.8	Coefficient of received tokens for agent "students trey" from total number invested tokens
	0.7	Coefficient of received tokens for agent "students two" from total number invested tokens

	0.1	Coefficient of received tokens for agent "students one" from total number invested tokens
month	1	Current month of the project
monthEndStage	1. monthEndStage(SEED) =2 2. monthEndStage(PRE_TGE) =6 3. monthEndStage(TGE_ OPEN_SALE) =10	Closing months of periods. 1. monthStartStage(SEED) =2; 2. monthStartStage(PRE_TGE) =6; 3. monthStartStage(TGE_ OPEN_SALE) =10;
monthSM	1. monthSM(0)=1 2. monthSM(1)=3 3. monthSM(2)=7 4. monthSM(3)=42 5. monthSM(4)=43	Skillmining periods for investment. 1. monthSM(0)=1; 2. monthSM(1)=3; 3. monthSM(2)=7; 4. monthSM(3)=42; 5. monthSM(4)=43;

monthStartStage	1. monthStartStage(SEED) = 1
	2. monthStartStage(PRE_TGE) =3
	3. monthStartStage(TGE_ OPEN_SALE) =6
		Starting months of periods.
		monthStartStage(SEED) = 1;
		monthStartStage(PRE_TGE) =3;
		monthStartStage(TGE_ OPEN_SALE) =6;

monthlyPRE_TGE	250	The number of tokens emitted per month in the closed period ICO
monthlyTGE_OPEN_SALE	250	The number of tokens emitted per month in the open period ICO.
pForSale	0.25	The share of tokens in the node for sale, after the purchase of the node.
pOICOcoach	0.1	Share of purchased tokens on open sale for coaches.
pOICOhold	0.2	Share of purchased tokens on open sale for agents "holders".
pOICOmng	0.05	Share of purchased tokens on open sale for managers.
pOICOnode	0.2	Share of purchased tokens on open sale for owners nodes
pOICOstd_e **pOICOstd_g** **pOICOstd_gm** **pOICOstd_o** **pOICOstd_tr** **pOICOstd_tw**	0.2	Share of purchased tokens on open sale for students.
pReserve	0.25	The share of tokens in the node for the reserve, after the purchase of the node
pSale	0.8	Part of the sold tokens on open sale.

pSkillMining	0.9	The share of tokens in the node for skillmining, after the purchase of the node.
percStdE **percStdG** **percStdGM** **percStdON** **percStdTR** **percStdTW**	0.1 0.15 0.25 0.25 0.15 0.1	Percentage distribution "students"
priceDelta	0.7	Price difference of token the current period and past period
priceRation	0.05	The coefficient on which price changes
rationHoldersSeed1 **rationHoldersSeed2**	0.4 0.6	1. Unlock SEED tokens by investors. 60% - are locked up for 4 months 2. Unlock SEED tokens by investors 40% are locked
1. **revCoach** 2. **revHold** 3. **revManager** 4. **revNode** 5. **revPlatform**	0.5 0.25 0.2 0.05 1	1. revCoach- Profit share for coach from node 2. revHold- Profit share for agents "holders" from node 3. revManager- Profit share for managers from node 4. revNode- Profit share for node owner 5. revPlatform- Profit share for platform owner from node
sumRation	0	Sum of coefficients in the demand formula
tokenPrice	1	Price of token
tokenPriceLastPeriod	0	Price for a token in the last period
percReduc	0.05	The amount by which the number of students decreases (in percents)
criticalLimit	100	Student purchasing power
listPurchases	1. listPurchases(EXCELLENT)=0 2. listPurchases(GOOD)=0 3. listPurchases(GOOD_M)=0 4. listPurchases(TREY)=0 5. listPurchases(TWO)=0 6. listPurchases(ONE)=0	The number of purchased tokens by student groups
purchaseRate	1	The ratio of the purchase of tokens by students at the node to the purchase of tokens on the exchange



CIRCULATION OF TOKENS IN THE MODEL

MODEL FORMAIIZATION

NAME	**DESCRIPTION**	**FORMALIZATION**
basicNeedsCalculation	The calculation of the cost of skillmining for the mastering of the planned amount	**basicNeedsCalculation**=Operator((month >= 2)->> ("") (**basicNeeds:**=pSkillMining * nodes . tokenSkillMining / sumRation))
studentE_BuyTokens	Starting from the 7th month (the end of the closed ICO), students (excellent student) acquire tokens to replenish their skillmining account. Noda receives cash income	**studentE_BuyTokens**=Operator((month > 6&std_e . tokenAvailable < basicNeeds * percStdE)->> ("") (**std_e . tokenAvailable**:=std_e . tokenAvailable + (percStdE * basicNeeds - std_e . tokenAvailable) ; l**istPurchases(EXCELLENT)**:=listPurchases(EXCELLENT) + (percStdE * basicNeeds - std_e . tokenAvailable); **nodes . tokenAvailable:**=nodes . tokenAvailable - purchaseRate*(percStdE * basicNeeds - std_e . tokenAvailable) ; **nodes . income**:=nodes . income + purchaseRate*(percStdE * basicNeeds - (std_e . tokenAvailable)) * tokenPrice;

		stock . tokens:=stock . tokens - (1-purchaseRate)*(percStdE * basicNeeds - (std_e . tokenAvailable))))
studentG_BuyTokens	Starting from the 7th month (the end of the closed ICO), students (student good) acquire tokens to replenish their skillmining account. Noda receives cash income	**studentG_BuyTokens**=Operator((month > 6&std_g . tokenAvailable < basicNeeds * percStdG)->("") (**std_g . tokenAvailable**:=std_g . tokenAvailable + percStdG * basicNeeds - std_g . tokenAvailable; **listPurchases(GOOD)**:=listPurchases(GOOD) + percStdG * basicNeeds - std_g . tokenAvailable; **nodes . tokenAvailable:**=nodes . tokenAvailable - purchaseRate*(percStdG * basicNeeds - std_g . tokenAvailable) ; **nodes . income**:=nodes . income + purchaseRate*(percStdG * basicNeeds - std_g . tokenAvailable) * tokenPrice; **stock . tokens**:=stock . tokens - (1-purchaseRate)*(percStdG * basicNeeds - std_g . tokenAvailable)))
studentGM_BuyTokens	Starting from the 7th month (the end of the closed ICO), students (student good with minus) acquire tokens to replenish their skillmining account. Noda receives cash income	**studentGM_BuyTokens**=Operator((month > 6&std_gm . tokenAvailable < basicNeeds * percStdGM)->("") (**std_gm . tokenAvailable**:=std_gm . tokenAvailable + percStdGM * basicNeeds - std_gm . tokenAvailable ; l**istPurchases(GOOD_M):**=listPurchases(GOOD_M) + (percStdGM * basicNeeds - std_gm . tokenAvailable); **nodes . tokenAvailable**:=nodes . tokenAvailable - purchaseRate*(percStdGM * basicNeeds - std_gm . tokenAvailable) ; **nodes . income:**=nodes . income + purchaseRate*(percStdGM * basicNeeds - std_gm . tokenAvailable) * tokenPrice; **stock . tokens:**=stock . tokens - (1-purchaseRate)*(percStdGM * basicNeeds - (std_gm . tokenAvailable))))
studentTR_BuyTokens	Starting from the 7th month (the end of the closed ICO), students (students trey) acquire tokens to replenish their skillmining account. Noda receives cash income	**studentTR_BuyTokens**=Operator((month > 6&std_tr . tokenAvailable < basicNeeds * percStdTR)->("") (**std_tr . tokenAvailable**:=std_tr . tokenAvailable + percStdTR * basicNeeds - std_tr . tokenAvailable; **listPurchases(TREY)**:=listPurchases(TREY) + (percStdTR * basicNeeds - std_tr . tokenAvailable); **nodes . tokenAvailable:**=nodes . tokenAvailable - purchaseRate*(percStdTR * basicNeeds - std_tr . tokenAvailable) ; **nodes . income**:=nodes . income + purchaseRate*(percStdTR * basicNeeds - std_tr . tokenAvailable) * tokenPrice;

		stock . tokens:=stock . tokens - (1-purchaseRate)*(percStdTR * basicNeeds - std_tr . tokenAvailable)))
studentTW_BuyTokens	Starting from the 7th month (the end of the closed ICO), students (students two) acquire tokens to replenish their skillmining account. Noda receives cash income	**studentTW_BuyTokens**=Operator((month > 6&std_tw . tokenAvailable < basicNeeds * percStdTW)-> ("") (**std_tw . tokenAvailable:**=std_tw . tokenAvailable + (percStdTW * basicNeeds - std_tw . tokenAvailable) ; **listPurchases(TWO)**:=listPurchases(TWO) + percStdTW * basicNeeds - std_tw . tokenAvailable; **nodes . tokenAvailable**:=nodes . tokenAvailable - purchaseRate*(percStdTW * basicNeeds - std_tw . tokenAvailable) ; **nodes . income**:=nodes . income + purchaseRate*(percStdTW * basicNeeds - std_tw . tokenAvailable) * tokenPrice; **stock . tokens**:=stock . tokens - (1-purchaseRate)*(percStdTW * basicNeeds - std_tw . tokenAvailable)))
studentON_BuyTokens	Starting from the 7th month (the end of the closed ICO), students (students one) acquire tokens to replenish their skillmining account. Noda receives cash income.	**studentON_BuyTokens**=Operator((month > 6&std_o . tokenAvailable < basicNeeds * percStdON)-> ("") (**std_o . tokenAvailable**:=std_o . tokenAvailable + (percStdON * basicNeeds -std_o . tokenAvailable) ; **listPurchases(ONE)**:=listPurchases(ONE) + percStdON * basicNeeds - std_o . tokenAvailable; **nodes . tokenAvailable**:=nodes . tokenAvailable - purchaseRate*(percStdON * basicNeeds - std_o . tokenAvailable) ; **nodes . income**:=nodes . income + purchaseRate*(percStdON * basicNeeds - std_o . tokenAvailable) * tokenPrice; **stock . tokens**:=stock . tokens - (1-purchaseRate)*(percStdON * basicNeeds - std_o . tokenAvailable)))
managerBuyFund	Managers buy available tokens from node owners for money and buy funds for tokens. The owners of the nodes sell and then take the tokens back.	**managerBuyFund**=Operator(((month=monthStartStage(TGE_OPEN_SALE)))-> ("") (**nodes . income**:=nodes . income + FUNDS * fundPrice * tokenPrice))
priceChangeDOWN	With falling demand for tokens, the price of the token falling by 5%	**priceChangeDOWN**=Operator((stock . bought / stock . sold < 1)-> ("") (**tokenPrice**:=tokenPrice - (tokenPrice * priceRation)))

priceChangeUP	With the growth in demand for the token, the price of the token grows by 5%	**priceChangeUP**=Operator((stock . bought / stock . sold >= 1)->> ("") (**tokenPrice:**=tokenPrice + tokenPrice * priceRation))
stockSellStudent	Sale of tokens by excellent students; (1) -koefSMWork_exelent * percStdE * basicNeed - calculated by agent "students excellence" from the basic need, according to the percentage distribution. 2) -percStdE * basicNeed - the part that is assigned to excellent students. (1) - (2) - profit	**stockSellStudent**=Operator((month > 6)->> ("") (**stock . tokens**:=stock . tokens + koefSMWork_exelent * percStdE * basicNeeds - percStdE * basicNeeds ; **stock . bought**:=stock . bought + koefSMWork_exelent * percStdE * basicNeeds - percStdE * basicNeeds ; **std_e . tokenAvailable**:=std_e . tokenAvailable - (koefSMWork_exelent * percStdE * basicNeeds - percStdE * basicNeeds) ; **std_e . income**:=std_e . income + (koefSMWork_exelent * percStdE * basicNeeds - percStdE * basicNeeds) * tokenPrice))
unlockHolders	Unlock tokens. Starting from the 8 month agents holders, each month decompose 0.3 of their tokens..	**unlockHolders**=Operator((month >= 8)->> ("") (**hold . tokenLocked**:=hold . tokenLocked - holdersUnlock ; **hold . tokenAvailable**:=hold . tokenAvailable + holdersUnlock))
prepareUnlock	Unlock tokens On the 8th month, 0.3 of tokens is calculated from agent "holders".	**prepareUnlock**=Operator((month >= 8)->> ("") (**holdersUnlock**:=holderRation * hold . tokenLocked))
unlockICO	Unlock tokens. All that was bought on the open ICO is unlock down after the end of the ICO.	**unlockICO**=Operator(((month=monthStartStage(TGE_OPEN_SALE) + 1))->> ("") (**nodes . tokenAvailable**:=nodes . tokenAvailable + nodes . tokenLocked ; **coach . tokenAvailable**:=coach . tokenAvailable + coach . tokenLocked ; **mng . tokenAvailable**:=mng . tokenAvailable + mng . tokenLocked ; **std_e . tokenAvailable**:=std_e . tokenAvailable + std_e . tokenLocked ; **std_g . tokenAvailable**:=std_g . tokenAvailable + std_e . tokenLocked ; **std_gm . tokenAvailable**:=std_gm . tokenAvailable + std_gm . tokenLocked ; **std_tr . tokenAvailable**:=std_tr . tokenAvailable + std_tr . tokenLocked ; **std_tw . tokenAvailable:**=std_tw . tokenAvailable + std_tw . tokenLocked ;

		std_o . tokenAvailable:=std_o . tokenAvailable + std_o . tokenLocked ; **nodes . tokenLocked**:=0 ; **coach . tokenLocked**:=0 ; **mng . tokenLocked**:=0 ; **std_e . tokenLocked**:=0 ; **std_g . tokenLocked**:=0 ; **std_gm . tokenLocked**:=0 ; **std_tr . tokenLocked**:=0 ; **std_tw . tokenLocked**:=0 ; **std_o . tokenLocked**:=0))
tokenOpenICOSale	Sale of tokens in the period TGE_OPEN_SALE. Open sale period. 1.The platform sells a monthly rate of tokens. 2.The platform contributes to the reserve fund what is not sold. 3.Platform receives cash income. 4. Nodes, trainers and students buy their part of tokens according to the pOICOXX ratio. All purchased lochitsya until the end of the open sale. 5. Holders buy their share of tokens according to the pOICOhold ratio. Tokens are locked.	**tokenOpenICOSale**=Operator((month >= monthStartStage(TGE_OPEN_SALE)&month <= monthEndStage(TGE_OPEN_SALE))->("") (**platform . tokenICOStageForSale(TGE_OPEN_SALE)**:=platform . tokenICOStageForSale(TGE_OPEN_SALE) - (monthlyTGE_OPEN_SALE) ; **nodes . tokenLocked**:=nodes . tokenLocked + monthlyTGE_OPEN_SALE * pOICOnode ; **hold . tokenLocked**:=hold . tokenLocked + monthlyTGE_OPEN_SALE * pOICOhold ; **std_e . tokenLocked:**=std_e . tokenLocked + monthlyTGE_OPEN_SALE * pOICOstd_e ; **std_g . tokenLocked**:=std_g . tokenLocked + monthlyTGE_OPEN_SALE * pOICOstd_g ; **std_gm . tokenLocked**:=std_gm . tokenLocked + monthlyTGE_OPEN_SALE * pOICOstd_gm ; **std_tr . tokenLocked**:=std_tr . tokenLocked + monthlyTGE_OPEN_SALE * pOICOstd_tr ; **std_tw . tokenLocked**:=std_tw . tokenLocked + monthlyTGE_OPEN_SALE * pOICOstd_tw ; **std_o . tokenLocked:**=std_o . tokenLocked + monthlyTGE_OPEN_SALE * pOICOstd_o ; **coach . tokenLocked**:=coach . tokenLocked + monthlyTGE_OPEN_SALE * pOICOcoach ; **mng . tokenLocked**:=mng . tokenLocked + monthlyTGE_OPEN_SALE * pOICOmng ; **platform . RESERVE**:=platform . RESERVE + (1 - pSale) * monthlyTGE_OPEN_SALE ; **platform . income:**=platform . income + monthlyTGE_OPEN_SALE * pForSale * tokenPrice))
tokenSalePreTGE	Selling tokens during the PRE_TGE period.1. All that is not sold every month we contribute to the reserve fund. 2. Nodes buy tokens along with nodes. This is all that is sold in this period	**tokenSalePreTGE**=Operator((month >= monthStartStage(PRE_TGE)&month <= monthEndStage(PRE_TGE))->("") (**nodes . tokenSkillMining**:=nodes . tokenSkillMining + monthlyPRE_TGE * pSkillMining ; **nodes . nodeReserve**:=nodes . nodeReserve + monthlyPRE_TGE * pReserve ;

	only for the nodes. The received tokens are distributed between the fund for skill and the fund available for sale. 3. The platform returns tokens and receives a cash profit for them.	**nodes . tokenAvailable**:=nodes . tokenAvailable + monthlyPRE_TGE * pForSale ; **platform . tokenICOStageForSale(PRE_TGE)**:=platform . tokenICOStageForSale(PRE_TGE) - monthlyPRE_TGE) ; **platform . income**:=platform . income + monthlyPRE_TGE * pForSale * tokenPrice))
tokenSaleSeed	Selling during the SEED period. The corresponding planned number of tokens goes to the Holders agent completely in the first month. Agent platform gets cash returns.	**tokenSaleSeed**=Operator((month=monthStartStage(SEED))->> ("") (**hold . tokenLocked**:=hold . tokenLocked + platform . tokenICOStageEmission(SEED) ; **platform . tokenICOStageEmission(SEED)**:=0 ; **platform . income**:=platform . income + platform . tokenICOStageEmission(SEED) * tokenPrice))
stageEmission	Emit tokens for sale in periods (SEED, PRE_TGE, TGE_ OPEN_SALE) in the first month of the period (see Period Table) in which the emission is planned	**stageEmission**=Operator(Exist (i:ICO_STAGE)(((month=monthStartStage(i)))->> ("") (**platform . tokenICOStageForSale(i)**:=platform . tokenICOStageEmission(i) ; **platform . commonEmission:=**platform . commonEmission - platform . tokenICOStageEmission(i)))
nextMonth **lastMonth**		**nextMonth**=Operator((month < allMonths)->> ("") (**month**:=month + 1; **stock . bought**:=0; **stock . sold**:=0; **tokenPriceLastPeriod**:=tokenPrice)) **lastMonth**=Operator((month = allMonths)->> ("") (1))
skillMiningInvest	Planned investment in skillmining from the platform. Planned investment in skillmining from the platform. Tokens go to the common pool of skillmining, namely to nodes, to the corresponding attribute tokenSkillMining, according to token investment plan by months (SMInvestToken) from the emission pool.	**skillMiningInvest**=Operator(Forall (i:int)(((i > 0&i <= 4)->((monthSM(i)=month)))->> ("") (**nodes . tokenSkillMining**:=nodes . tokenSkillMining + platform . SMInvestToken(i) ; **platform . commonEmission**:=platform . commonEmission - platform . SMInvestToken(i))))

tokenHolders	Tokens are delivered to the agent Holders, according to the planned issue	**tokenHolders**=Operator(Forall (i:int)((i >= 1&i <= 7&(holderEmssionMonth(i)=month))->("") (**hold . tokenLocked**:=hold . tokenLocked + platform . holderEmission(i) ; **platform . commonEmission:**=platform . commonEmission -platform . holderEmission(i)))
studentInitialMining	Initial skillmining Students mine the first 6 months without committing costs. Each group of students earns in accordance with their coefficient:(1.1-"student excellent", 1-"stduent good",0.9-"student good with minus", 0.8-"student trey", 0.7-"student two", 0.1-"student one"). Nodes gives the planned number of received tokens from the account of skillmining. (pSkillMining coefficient) Not received goes to reserve	**studentInitialMining**=Operator((month >= 2&month <= 6)->("") (**std_e . tokenAvailable:**=std_e . tokenAvailable + koefSMWork_exelent * percStdE * basicNeeds ; **std_g . tokenAvailable**:=std_g . tokenAvailable + koefSMWork_good * percStdG * basicNeeds ; **std_gm . tokenAvailable**:=std_gm . tokenAvailable + koefSMWork_goodm * percStdGM * basicNeeds ; **std_tr . tokenAvailable**:=std_tr . tokenAvailable + koefSMWork_trey * percStdTR * basicNeeds ; **std_tw . tokenAvailable**:=std_tw . tokenAvailable + koefSMWork_two * percStdTW * basicNeeds ; **std_o . tokenAvailable**:=std_o . tokenAvailable + koefSMWork_one * percStdON * basicNeeds ; **nodes . tokenSkillMining**:=nodes . tokenSkillMining - (koefSMWork_exelent * percStdE * basicNeeds + koefSMWork_good * percStdG * basicNeeds + koefSMWork_goodm * percStdGM * basicNeeds + koefSMWork_trey * percStdTR * basicNeeds + koefSMWork_two * percStdTW * basicNeeds + koefSMWork_one * percStdON * basicNeeds) - (1-pSkillMining) * nodes . tokenSkillMining ; **nodes . nodeReserve**:=nodes . nodeReserve + (1-pSkillMining) * nodes . tokenSkillMining))
unlockSeed1	Unlock SEED tokens by investors. 40% are locked up for 3 months	**unlockSeed1=**Operator(((month=4))->("") (**hold . tokenLocked**:=hold . tokenLocked+(-1)*(unlocHolderSeeds * rationHoldersSeed1); **hold . tokenAvailable**:=hold . tokenAvailable +unlocHolderSeeds * rationHoldersSeed1; **unlocHolderSeeds**:=unlocHolderSeeds-unlocHolderSeeds*rationHoldersSeed1)),
unlocCalculationHolders	The value of 10% of locked tokens after unlocking 40%	**unlocCalculationHolders** = Operator((month=5)->("") (**unlocHold:=** unlocHolderSeeds* holdRation2))
unlockSeed2	Unlock SEED token investors. 60% - locked	**unlockSeed2**=Operator((month >= 5&month <= 15)->

	up for 4 months and then unlocked at 10% per month	("") (**hold . tokenLocked**:=hold . tokenLocked + (-1)*unlocHold; **hold . tokenAvailabl**e:=hold . tokenAvailable **+ unlocHold**)),
stockBuyIncreasedHolders	Protocol purchase tokens with an increase in the price. Available Holder agent tokens can be traded on the stock exchange according to the coefSoundnessUpBuy ratio	**stockBuyIncreasedHolders**=Operator((priceDelta >= 0)->> ("") (**stock . tokens**:=stock . tokens + coefSoundnessUpBuy * hold . tokenAvailable ; **hold . tokenAvailable**:=hold . tokenAvailable - coefSoundnessUpBuy * hold . tokenAvailable); **stock . bought**:=stock . bought + coefSoundnessUpBuy * hold . tokenAvailable))
stockBuyDecreasedHolders	Protocol purchase tokens at lower prices. Available Holder agent tokens can be traded on the stock exchange according to the coefSoundnessDownBuy ratio	**stockBuyDecreasedHolders=**Operator((priceDelta < 0)->> ("") (**stock . tokens**:=stock . tokens + coefSoundnessDownBuy * hold . tokenAvailable ; **hold . tokenAvailable**:=hold . tokenAvailable - coefSoundnessDownBuy * hold . tokenAvailable ; **stock . bought**:=stock . bought + coefSoundnessDownBuy * hold . tokenAvailable))
stockSellIncreasedHolders	Tokens sale protocol with price increase. The Holder agent can buy tokens from the stock exchange according to the coefSoundnessUpSell ratio set.	**stockSellDecreasedHolders**=Operator((priceDelta < 0)->> ("") (**stock . tokens**:=stock . tokens- (coefSoundnessDownSell * stock . tokens) ; **hold . tokenAvailable**:=hold . tokenAvailable + coefSoundnessDownSell * stock . tokens ; **stock . sold**:=stock . sold + coefSoundnessDownSell * stock . tokens))
stockSellDecreasedHolders	Tokens sale protocol at lower prices. Agent Holder can buy tokens from the stock exchange according to the established coefficient coefSoundnessDownSell	**stockSellDecreasedHolders**=Operator((priceDelta < 0)->> ("") (**stock . tokens**:=stock . tokens - (coefSoundnessDownSell * stock . tokens) ; **hold . tokenAvailable**:=hold . tokenAvailable + coefSoundnessDownSell * stock . tokens ; **stock . sold**:=stock . sold + coefSoundnessDownSell * stock . tokens))
studentMining	Students mining and pay according to the established cost ratios after the project grants for mining. In addition, students give coaches a share of the nominal rate. Coaches pay a share of managers according to	**studentMining**=Operator((month > 6)->> ("") (**std_e . tokenAvailable**:=std_e . tokenAvailable - koefSM * koefSMWork_exelent * percStdE * basicNeeds + koefSMWork_exelent * percStdE * basicNeeds - revCoach * percStdE * basicNeeds ; **std_g . tokenAvailable**:=std_g . tokenAvailable - koefSM * koefSMWork_good * percStdG * basicNeeds +

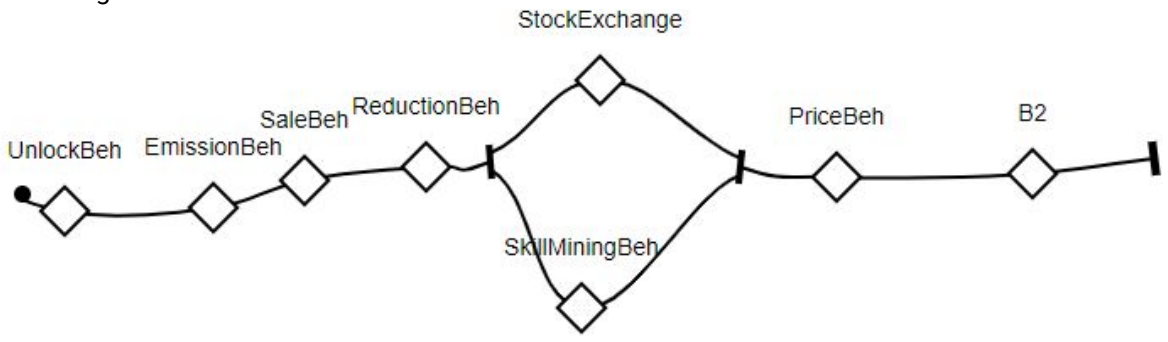
	the coefficient. Managers pay according to the coefficient for node. Students from available tokens subtract the number of tokens according to the coefficient of the cost of the students. We add received according to the percentage distribution by groups of students and the coefficient of tokens skillmining student to the basic needs.The balance (reserve) is distributed between the reserve of the node and the reserve of platforms in accordance with the ratio of the money invested in the mining.	koefSMWork_good * percStdG * basicNeeds - revCoach * percStdG * basicNeeds ; **std_gm . tokenAvailable**:=std_gm . tokenAvailable - koefSM * koefSMWork_goodm * percStdGM * basicNeeds + koefSMWork_goodm * percStdGM * basicNeeds - revCoach * percStdGM * basicNeeds ; **std_tr . tokenAvailable**:=std_tr . tokenAvailable - koefSM * koefSMWork_trey * percStdTR * basicNeeds + koefSMWork_trey * percStdTR * basicNeeds - revCoach * percStdTR * basicNeeds ; **std_tw . tokenAvailable**:=std_tw . tokenAvailable - koefSM * koefSMWork_two * percStdTW * basicNeeds + koefSMWork_two * percStdTW * basicNeeds - revCoach * percStdTW * basicNeeds ; **std_o . tokenAvailable**:=std_o . tokenAvailable - koefSM * koefSMWork_one * percStdON * basicNeeds + koefSMWork_one * percStdON * basicNeeds - revCoach * percStdON * basicNeeds; **coach . tokenAvailable**:=coach . tokenAvailable + revCoach * basicNeeds - revCoach * basicNeeds * revManager; **mng . tokenAvailable**:=mng . tokenAvailable + revCoach * basicNeeds * revManager - revCoach * basicNeeds * revManager * revNode; **nodes . tokenAvailable:**=nodes . tokenAvailable + revCoach * basicNeeds * revManager * revNode - revCoach * basicNeeds * revManager * revNode * revPlatform ; **nodes . tokenSkillMining**:=nodes . tokenSkillMining - (1-pSkillMining) * nodes . tokenSkillMining) - (koefSMWork_exelent * percStdE * basicNeeds + koefSMWork_good * percStdG * basicNeeds + koefSMWork_goodm * percStdGM * basicNeeds + koefSMWork_trey * percStdTR * basicNeeds + koefSMWork_two * percStdTW * basicNeeds + koefSMWork_one * percStdON * basicNeeds) ; **platform . RESERVE**:=platform . RESERVE + (nodes . tokenSkillMining - nodes . tokenSkillMining * pSkillMining) * platform . tokenSkillMining / monthlyPRE_TGE * pSale ; **nodes . nodeReserve**:=nodes . nodeReserve + (nodes . tokenSkillMining- (nodes . tokenSkillMining * pSkillMining)) * nodes . tokenSkillMining / monthlyP * pSale))
miningReinvest	Reinvestment into the pool for skillmining the node and the owners of the platform. The share of the owner of the platform and node is recalculated.	**miningReinvest**=Operator((month > 6)->("") (**nodes . tokenSkillMining**:=nodes . tokenSkillMining + nodes . nodeReserve + koefNeeds * platform . RESERVE ; **nodes . nodeReserve**:=0))
calculationSumRation	Calculation of sum of coefficients in the demand formula	**calculationSumRation**=Operator((month>1)->("") (**sumRation**:=koefSMWork_exelent * percStdE + koefSMWork_good * percStdG + koefSMWork_goodm * percStdGM + koefSMWork_trey * percStdTR +

		koefSMWork_two * percStdTW + koefSMWork_one * percStdON))
calculationPriceDelta	Calculates the price difference in the current period and last period.	**calculationPriceDelta**=Operator((~(tokenPrice=tokenPriceLastPeriod))-> ("") (**priceDelta**:= tokenPrice - tokenPriceLastPeriod))
percentageReduction_std_one	Recalculates percentage distribution after leaving 1 percent of “students one”.	**percentageReduction_std_one**=Operator((listPurchases(ONE)>=criticalLimit & percStdON>0.01)-> ("") (**percStdE**:=(percStdE * 100 + percStdE * 100 * percReduc*100/(100-percReduc*100))/100; **percStdG**:=(percStdG * 100 + percStdG * 100 * percReduc*100/(100-percReduc*100))/100; **percStdGM**:=(percStdGM * 100 + percStdGM * 100 * percReduc*100/(100-percReduc*100))/100; **percStdTR**:=(percStdTR * 100 + percStdTR * 100 * percReduc*100/(100-percReduc*100))/100; **percStdTW**:=(percStdTW * 100 + percStdTW * 100 * percReduc*100/(100-percReduc*100))/100; **percStdON**:=((percStdON * 100 - percReduc * 100) + ((percStdON * 100 - percReduc * 100) *percReduc*100/(100-percReduc*100)))/100)),
percentageReduction_std_two	Recalculates percentage distribution after leaving 1% of “students two”.	**percentageReduction_std_two**=Operator((listPurchases(TWO)>=criticalLimit & percStdTW>0.01)-> ("") (**percStdE**:=(percStdE * 100 + percStdE * 100 * percReduc*100/(100-percReduc*100))/100; **percStdG**:=(percStdG * 100 + percStdG * 100 * percReduc*100/(100-percReduc*100))/100; **percStdGM**:=(percStdGM * 100 + percStdGM * 100 * percReduc*100/(100-percReduc*100))/100; **percStdTR**:=(percStdTR * 100 + percStdTR * 100 * percReduc*100/(100-percReduc*100))/100; **percStdTW**:=((percStdTW * 100 - percReduc * 100) + ((percStdTW * 100 - percReduc * 100) *percReduc*100/(100-percReduc*100)))/100; **percStdON**:=(percStdON * 100 + percStdON * 100 * percReduc*100/(100-percReduc*100))/100))
percentageReduction_std_trey	Recalculates percentage distribution after leaving 1 % of “students trey”.	**percentageReduction_std_trey**=Operator((listPurchases(TREY)>=criticalLimit & percStdTR>0.01)-> ("") (**percStdE**:=(percStdE * 100 + percStdE * 100 * percReduc*100/(100-percReduc*100))/100; **percStdG**:=(percStdG * 100 + percStdG * 100 * percReduc*100/(100-percReduc*100))/100; **percStdGM**:=(percStdGM * 100 + percStdGM * 100 * percReduc*100/(100-percReduc*100))/100; **percStdTR**:=((percStdTR * 100 - percReduc * 100) +

		((percStdTR * 100 - percReduc * 100) *percReduc*100/(100-percReduc*100)))/100; **percStdTW**:=(percStdTW * 100 + percStdTW * 100 * percReduc*100/(100-percReduc*100))/100; **percStdON**:=(percStdON * 100 + percStdON * 100 * percReduc*100/(100-percReduc*100))/100)),
percentageReduction_std_goodm	Recalculates percentage distribution after leaving 1 % of “students goodm”.	**percentageReduction_std_goodm**=Operator((listPurchases(GOOD_M)>=criticalLimit & percStdGM>0.01)-> (“”) (**percStdE**:=(percStdE * 100 + percStdE * 100 * percReduc*100/(100-percReduc*100))/100; **percStdG**:=(percStdG * 100 + percStdG * 100 * percReduc*100/(100-percReduc*100))/100; **percStdGM**:=((percStdGM * 100 - percReduc * 100) + ((percStdGM * 100 - percReduc * 100) *percReduc*100/(100-percReduc*100)))/100; **percStdTR**:=(percStdTR * 100 + percStdTR * 100 * percReduc*100/(100-percReduc*100))/100; **percStdTW**:=(percStdTW * 100 + percStdTW * 100 * percReduc*100/(100-percReduc*100))/100; **percStdON**:=(percStdON * 100 + percStdON * 100 * percReduc*100/(100-percReduc*100))/100)),

BEHAVIORS AGENTS

Procedures are organized in the form of a cycle, which is repeated monthly. There is a parallel composition in the cycle, in which two procedures are performed in parallel - the sale of tokens and skillmining.



UnlockBeh	This procedure is unlocked tokens in the appropriate months for different types of agents
EmissionBeh	This procedure presents the sequential execution of emissions, emissions for agents of holders, as well as reinvestment into the pool of skillmining.

SaleBeh	This procedure presents a sequence of actions that regulates the sale of tokens in the period of SEED , the sale of nodes and distribution of tokens by groups
ReductionBeh	Changes percentage distribution of groups of students after they leave.
StockExchange	Holders buy and sell tokens in this way change demand on the tokens
SkillMiningBeh	Skillmining procedure presented. With a lack of tokens for skillmining, students buy tokens.
PriceBeh	Token price change. With a decrease in demand, the price of tokens falls, ,with an increase in demand, the price of the token increases. The difference in the price of tokens in the current period and the past is also calculated.
B2	Counter of months. Increases the month by one until it reaches 43 - months

BEHAVIORS AGENTS IN ALGEBRAIC FORM:

B0 = (UnlockBeh ; EmissionBeh ; ReductionBeh ; SaleBeh ; { SkillMiningBeh || StockExchange } ; PriceBeh; B2),

UnlockBeh = prepareUnlock . UB1 + ! prepareUnlock . UB1,
UB1 = unlockICO . UB2 + ! unlockICO . UB2,
UB2 = unlockHolders . UB3 + ! unlockHolders . UB3,
UB3 = unlockSeed1 . UB4 + ! unlockSeed1 . UB4,
UB4 = unlockSeed2 + ! unlockSeed2,

EmissionBeh = stageEmission . EB1 + not_stageEmission . EB1,
EB1 = tokenHolders . EB2 + ! tokenHolders . EB2,
EB2 = skillMiningInvest + not_skillMiningInvest,

ReductionBeh = percentageReduction_std_one . R1 + ! percentageReduction_std_one . R1,
R1 = percentageReduction_std_two . R2 + ! percentageReduction_std_two . R2,
R2 = percentageReduction_std_trey . R3 + ! percentageReduction_std_trey . R3,
R3 = percentageReduction_std_goodm + ! percentageReduction_std_goodm,

SaleBeh = tokenSaleSeed . SB1 + ! tokenSaleSeed . SB1,
SB1 = tokenSalePreTGE . SB2 + ! tokenSalePreTGE . SB2,
SB2 = tokenOpenICOSale . SB3 + ! tokenOpenICOSale . SB3,
SB3 = calculationSumRation . SB4 + ! calculationSumRation . SB4,
SB4 = basicNeedsCalculation + ! basicNeedsCalculation,

SkillMiningBeh = managerBuyFund . SM1 + ! managerBuyFund . SM1,
SM1 = studentE_BuyTokens . SM2 + ! studentE_BuyTokens . SM2,
SM2 = studentG_BuyTokens . SM3 + ! studentG_BuyTokens . SM3,
SM3 = studentGM_BuyTokens . SM4 + ! studentGM_BuyTokens . SM4,
SM4 = studentTR_BuyTokens . SM5 + ! studentTR_BuyTokens . SM5,
SM5 = studentTW_BuyTokens . SM6 + ! studentTW_BuyTokens . SM6,
SM6 = studentON_BuyTokens . SM7 + ! studentON_BuyTokens . SM7,
SM7 = studentInitialMining . SM8 + ! studentInitialMining . SM8,

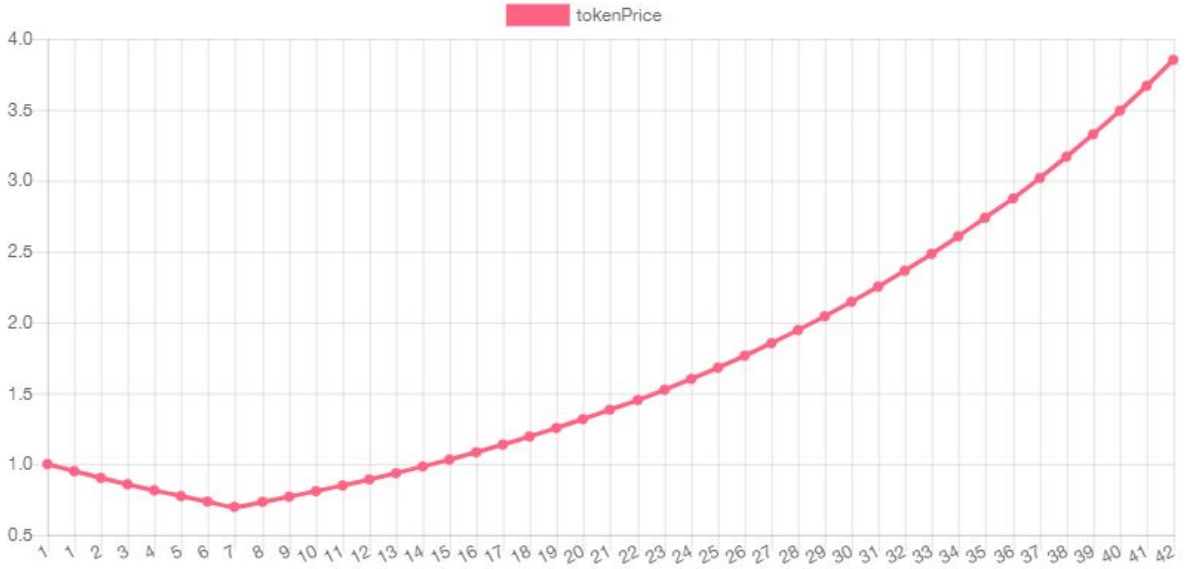
SM8 = studentMining . SM9 + ! studentMining . SM9,
SM9 = miningReinvest + ! miningReinvest,

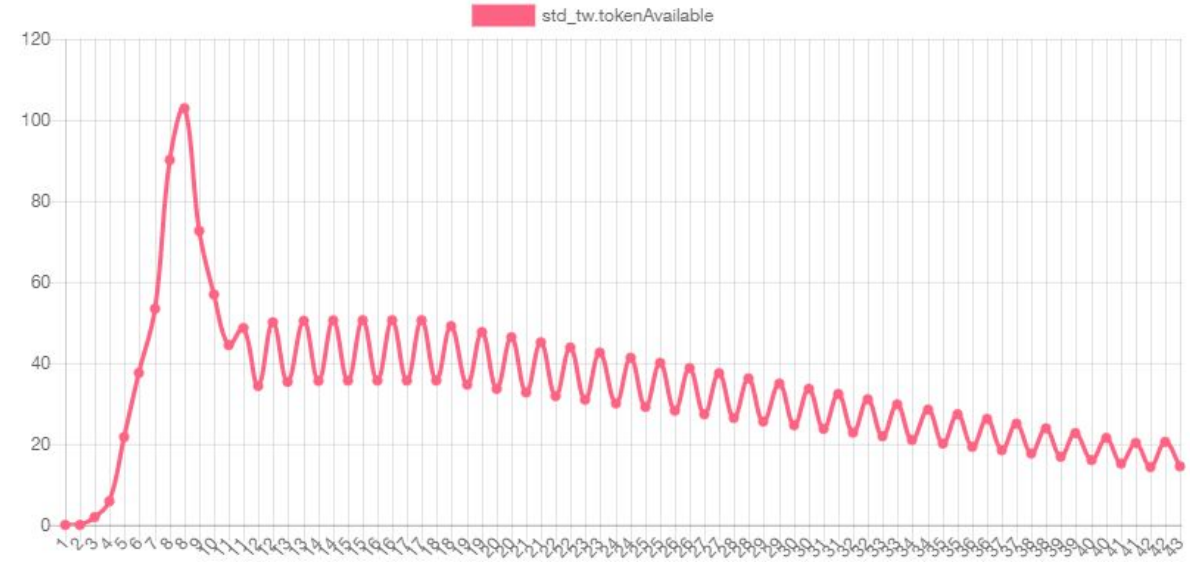
StockExchange = stockBuyIncreasedHolders . stockSellIncreasedHolders +
stockBuyDecreasedHolders . stockSellDecreasedHolders,

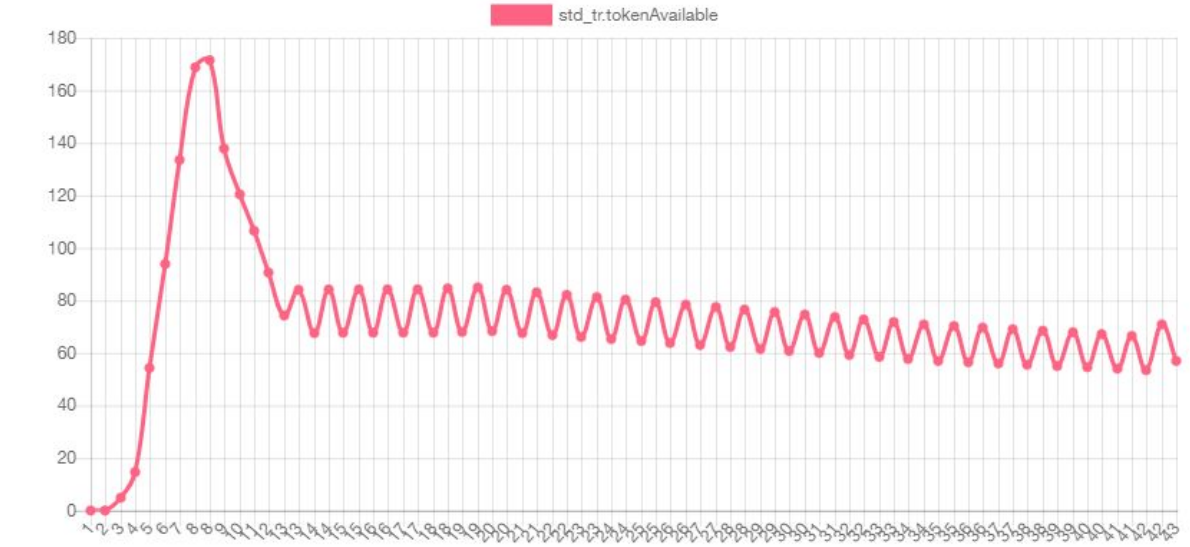
PriceBeh = priceChangeUP . PC1 + ! priceChangeUP . PC1,
PC1 = priceChangeDOWN . PC2 + ! priceChangeDOWN . PC2,
PC2 = stockSellStudent . PC3 + ! stockSellStudent . PC3,
PC3 = calculationPriceDelta + ! calculationPriceDelta,

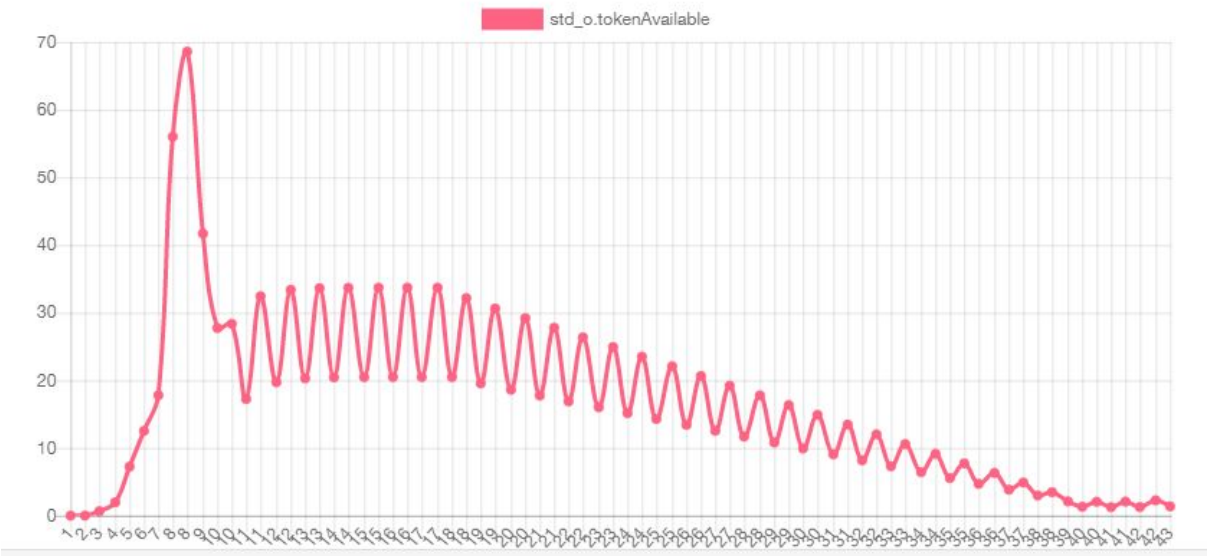
B2 = nextMonth . B0 + lastMonth . Delta

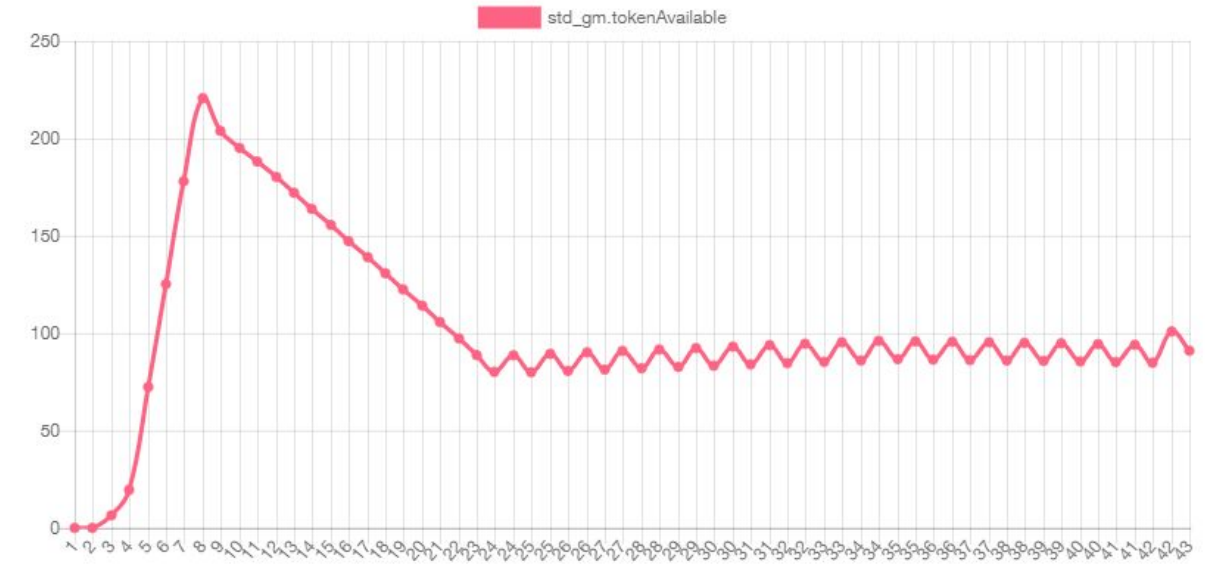
WORK RESULTS

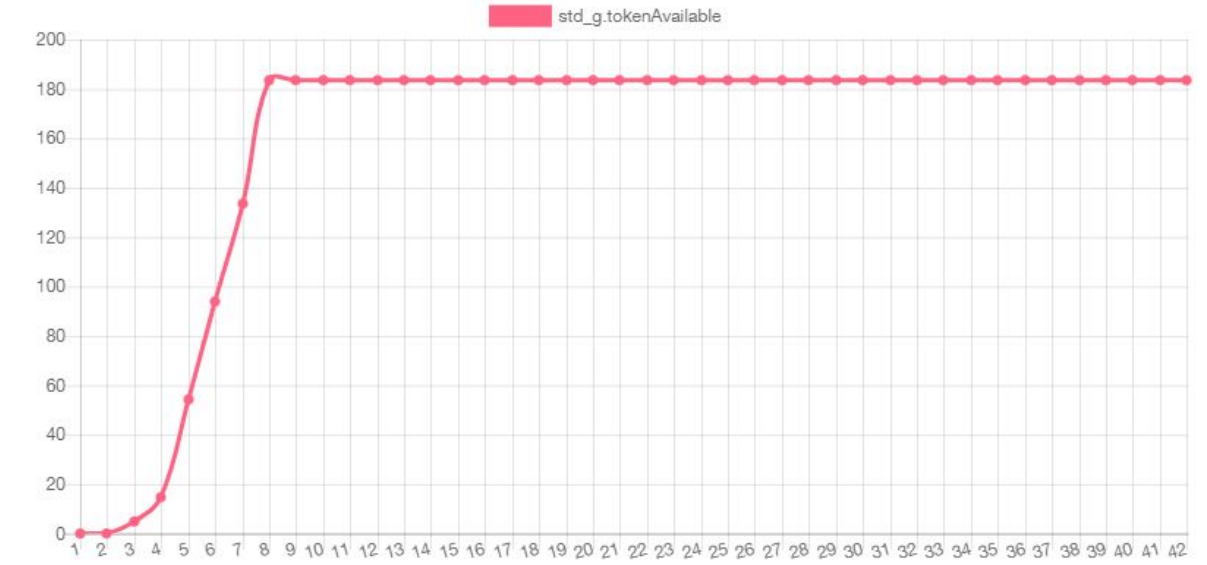

tokenPrice
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5

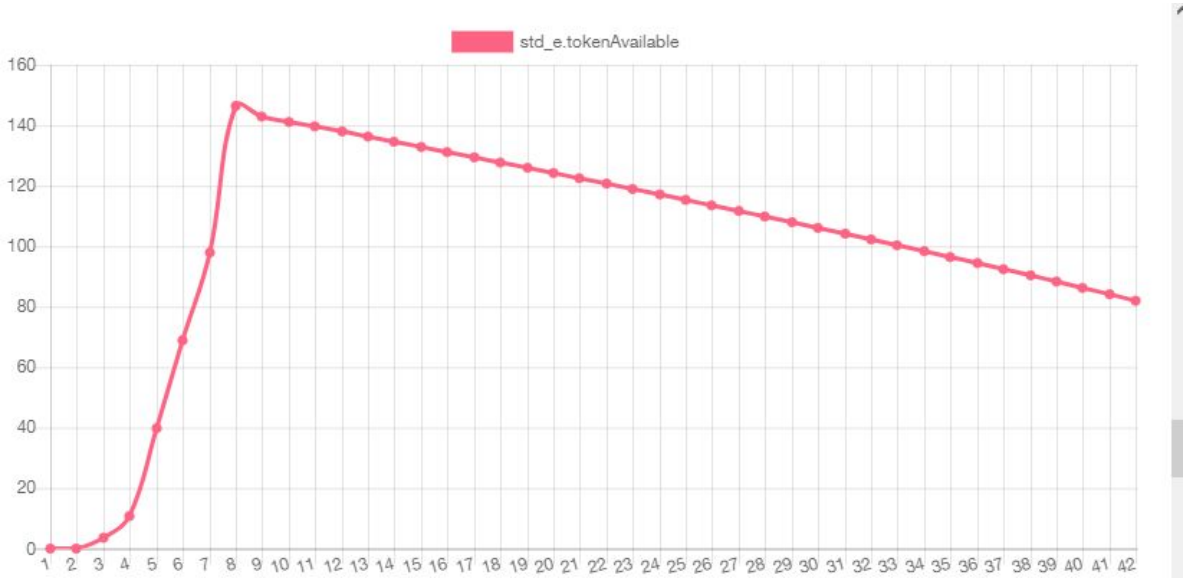

std_tw.tokenAvailable
120
100
80
60
40
20
0

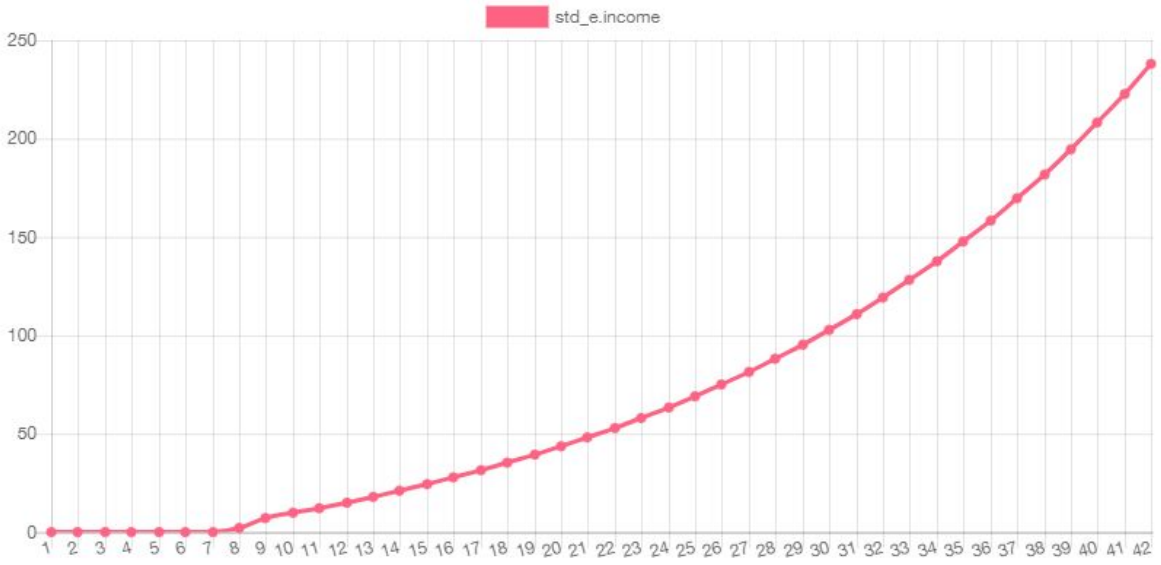

std_tr.tokenAvailable
180
160
140
120
100
80
60
40
20
0

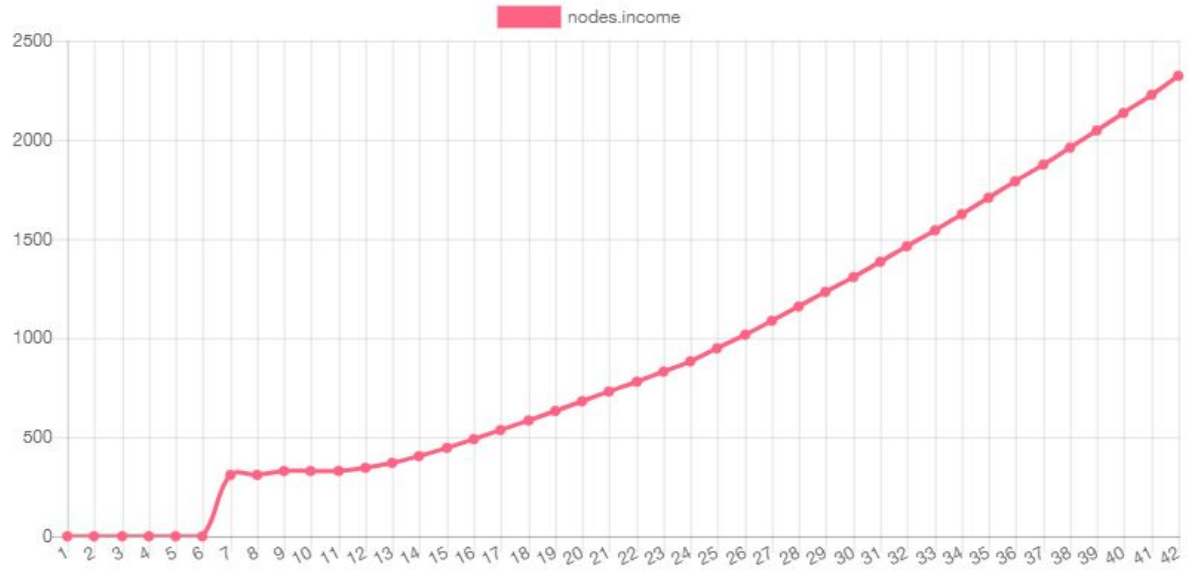

std_o.tokenAvailable
70
60
50
40
30
20
10
0

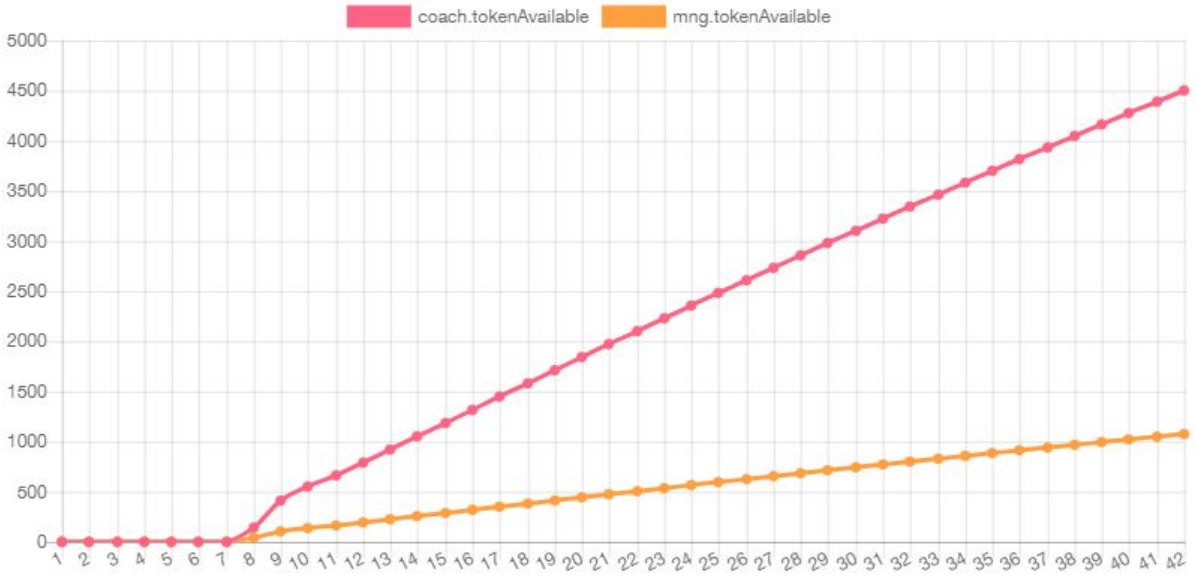

std_gm.tokenAvailable
250
200
150
100
50
0


std_g.tokenAvailable
200
180
160
140
120
100
80
60
40
20
0


std_e.tokenAvailable
160
140
120
100
80
60
40
20
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42


std_e.income
250
200
150
100
50
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42


nodes.income
2500
2000
1500
1000
500
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42


coach.tokenAvailable
mng.tokenAvailable
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42