Chess on Golem / doc v1.02

Summary

This project was created as an entry to Golem Hackathon 12/2020.

It's purpose is to show that any state based game / problem can be run in Golem Network and solved interactively by provider nodes.

This particular example shows classical chess game played by two AI players that facilitate golem network for computing.

Whole game is managed by Node.js server which distributes chess computing tasks across Golem Network providers.

Each move request is sent to Golem Market and calculated by provider with best bid.

The aim is to show how computing power may affect the outcome of the game.

Right now the depth is fixed by allowing particular players to calculate moves with specific search depth. In youtube demos white player plays with depth of 17 / 20 and black player plays with depth of 1.

In future with more providers in the Golem Network particular players may parallelize their calculations by using more or less providers to achieve better or worse moves in shorter time. It would also probably impact of final cost of game for each player.

Hybrid strategies might be also used, e.g.: use more computing power at the beginning of the game and less computing power in the endgame.

Outcome:

Node Chess App

Node.js Server (can be run on linux or windows machines) is responsible for handling chess game and requesting Golem Network for aid with calculating next moves for each AI player.

Moves are calculated on Node alpine docker image transformed to .gvim with a help of a stockfish.js chess engine.

Node chess app creates a request to golem network for each move that is being performed by AI players. For demo purposes one player asks for best move with depth precision of 20 and the other one uses depth of 1.

This can be changed in chess/index.js on line 32:

```
moveData.depth = moveData.turnId == "w" ? 20 : 1;
```

Typical calculation times:

Depth	< 10	=>	< 1s
	~ 20	=>	~ 3s
	~ 30	=>	~ 157s

Example of a file with task description that is sent to Golem Providers:

```
hash_00000132_0003
20
position fen rnbqkbnr/ppp1pppp/8/3p4/4P3/8/PPPP1PPP/RNBQKBNR w KQkq d6 0 2
```

Line 1 : id of an operation used to distinguish different tasks by chess server.
Line 2 : contains depth that stockfish.js algorithm needs to consider.
Line 3: describes current chess game state in fen notation.

Correct output should look similar to this file:

```
bestmove e4d5 ponder g8f6
exec time:8672.225822
depth:20
hash:hash_00000132_0003
```

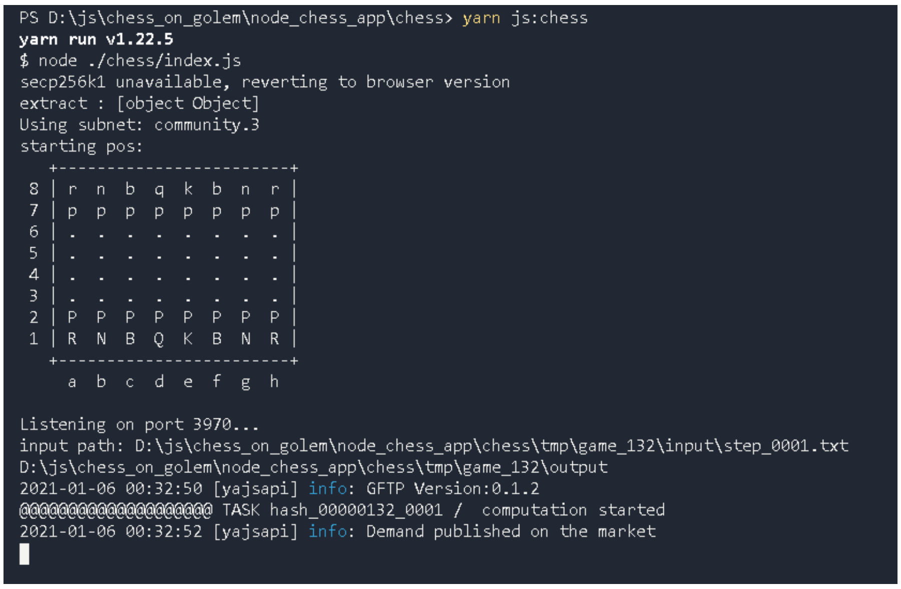
With lines describing suggested move, calculation time[ms], depth of calculations and operation id.

Node Chess app is also used as backend server for GUI App that displays chess game in real time with some statistics regarding provider nodes work.

Demo of Node Chess app currently runs at http:// 20.52.154.16/3970 on Linux Ubuntu VM in MS Azure cloud.

To run Node Chess app please do the following:

```
cd chess
yarn install
yarn js:chess
```



```
PS D:\js\chess_on_golem\node_chess_app\chess> yarn js:chess
yarn run v1.22.5
$ node ./chess/index.js
secp256k1 unavailable, reverting to browser version
extract : [object Object]
Using subnet: community.3
starting pos:
   +------------------------+
 8 | r  n  b  q  k  b  n  r |
 7 | p  p  p  p  p  p  p  p |
 6 | .  .  .  .  .  .  .  . |
 5 | .  .  .  .  .  .  .  . |
 4 | .  .  .  .  .  .  .  . |
 3 | .  .  .  .  .  .  .  . |
 2 | P  P  P  P  P  P  P  P |
 1 | R  N  B  Q  K  B  N  R |
   +------------------------+
     a  b  c  d  e  f  g  h

Listening on port 3970...
input path: D:\js\chess_on_golem\node_chess_app\chess\tmp\game_132\input\step_0001.txt
D:\js\chess_on_golem\node_chess_app\chess\tmp\game_132\output
2021-01-06 00:32:50 [yajsapi] info: GFTP Version:0.1.2
@@@@@@@@@@@@@@@@@@@@ TASK hash_00000132_0001 /  computation started
2021-01-06 00:32:52 [yajsapi] info: Demand published on the market
█
```

Script runs until game is finished, when some calculation fails or timeouts golem network is being asked to perform it again.

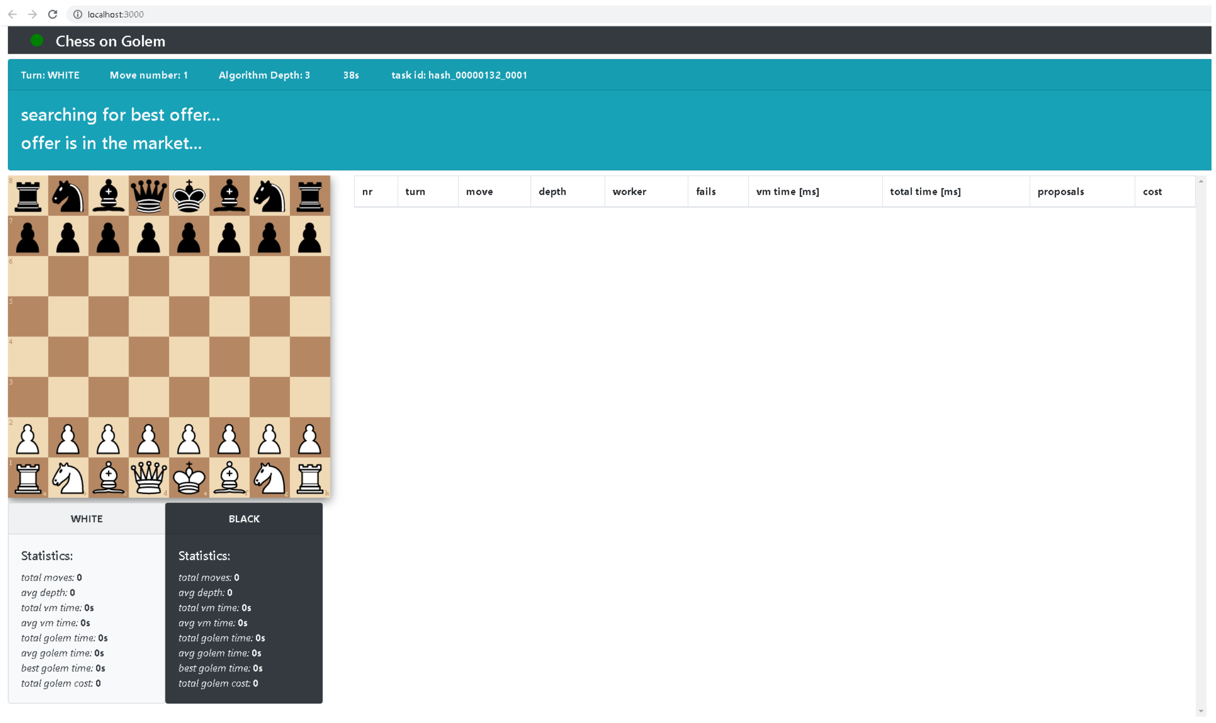
Multiple clients can connect to socket.io websocket server and listen for events that describe current game state

Events:

- currentTurnEvent
- providerFailed
- computationStarted
- movesRefreshed
- gameFinished
- offersReceived
- agreementCreated
- agreementConfirmed
- computationFinished
- invoiceReceived
- moveEvent
- positionEvent

When client reconnects server sends him automatically current state of the game.

Chess on Golem Viewer


localhost:3000

Chess on Golem

Turn: WHITE   Move number: 1   Algorithm Depth: 3   38s   task id: hash_00000132_0001

searching for best offer...
offer is in the market...

nr	turn	move	depth	worker	fails	vm time [ms]	total time [ms]	proposals	cost

WHITE	BLACK
Statistics:	Statistics:
total moves: **0**	*total moves:* **0**
avg depth: **0**	*avg depth:* **0**
total vm time: **0s**	*total vm time:* **0s**
avg vm time: **0s**	*avg vm time:* **0s**
total golem time: **0s**	*total golem time:* **0s**
avg golem time: **0s**	*avg golem time:* **0s**
best golem time: **0s**	*best golem time:* **0s**
total golem cost: **0**	*total golem cost:* **0**

React application that serves as GUI for displaying chess game progress for Chess on Golem.

It displays game progress and some interesting stats regarding provider nodes that took part in calculations.

There is live demo available at:

http://chess-on-golem-viewer.herokuapp.com/

If It's not currently running you can request start at pawel.burgchardt [A-T] gmail.com

You can run it locally by going to chess-viewer and executing

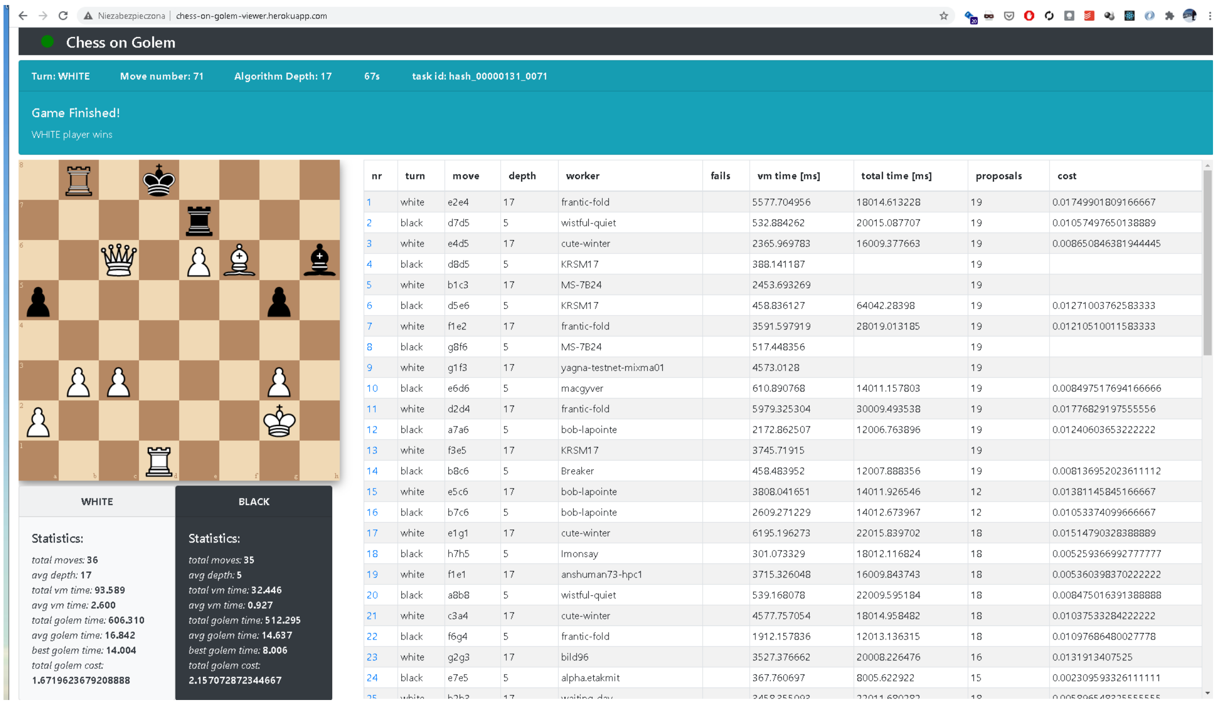
```
npm install
npm start
```

You can then open the browser and see the result at http://localhost:3000/

Chess on Golem Viewer connects automatically to node chess app server on 127.0.0.1:3970

To change it please update .env.development file

```
REACT_APP_NAME=Chess on Golem 1
REACT_APP_VERSION=0.0.1
REACT_APP_SOCKET_SERVER_URL=http://127.0.0.1:3970/
REACT_APP_API_URL=http://127.0.0.1:3970/api
```


Niezabezpieczona | chess-on-golem-viewer.herokuapp.com

Chess on Golem

Turn: WHITE   Move number: 71   Algorithm Depth: 17   67s   task id: hash_00000131_0071

Game Finished!

WHITE player wins

WHITE	BLACK
Statistics:	Statistics:
total moves: **36**	*total moves:* **35**
avg depth: **17**	*avg depth:* **5**
total vm time: **93.589**	*total vm time:* **32.446**
avg vm time: **2.600**	*avg vm time:* **0.927**
total golem time: **606.310**	*total golem time:* **512.295**
avg golem time: **16.842**	*avg golem time:* **14.637**
best golem time: **14.004**	*best golem time:* **8.006**
total golem cost: **1.6719623679208888**	*total golem cost:* **2.157072872344667**

nr	turn	move	depth	worker	fails	vm time [ms]	total time [ms]	proposals	cost
1	white	e2e4	17	frantic-fold		5577.704956	18014.613228	19	0.017499018091666667
2	black	d7d5	5	wistful-quiet		532.884262	20015.087707	19	0.010574976501388889
3	white	e4d5	17	cute-winter		2365.969783	16009.377663	19	0.008650846381944445
4	black	d8d5	5	KRSM17		388.141187		19	
5	white	b1c3	17	MS-7B24		2453.693269		19	
6	black	d5e6	5	KRSM17		458.836127	64042.28398	19	0.012710037625833333
7	white	f1e2	17	frantic-fold		3591.597919	28019.013185	19	0.012105100115833333
8	black	g8f6	5	MS-7B24		517.448356		19	
9	white	g1f3	17	yagna-testnet-mixma01		4573.0128		19	
10	black	e6d6	5	macgyver		610.890768	14011.157803	19	0.008497517694166666
11	white	d2d4	17	frantic-fold		5979.325304	30009.493538	19	0.017768291975555556
12	black	a7a6	5	bob-lapointe		2172.862507	12006.763896	19	0.012406036532222222
13	white	f3e5	17	KRSM17		3745.71915		19	
14	black	b8c6	5	Breaker		458.483952	12007.888356	19	0.008136952023611112
15	white	e5c6	17	bob-lapointe		3808.041651	14011.926546	12	0.013811458451666667
16	black	b7c6	5	bob-lapointe		2609.271229	14012.673967	12	0.010533740996666667
17	white	e1g1	17	cute-winter		6195.196273	22015.839702	18	0.015147903283888889
18	black	h7h5	5	lmonsay		301.073329	18012.116824	18	0.005259366992777777
19	white	f1e1	17	anshuman73-hpc1		3715.326048	16009.843743	18	0.005360398370222222
20	black	a8b8	5	wistful-quiet		539.168078	22009.595184	18	0.008475016391388888
21	white	c3a4	17	cute-winter		4577.757054	18014.958482	18	0.010375332842222222
22	black	f6g4	5	frantic-fold		1912.157836	12013.136315	18	0.010976864800277778
23	white	g2g3	17	bild96		3527.376662	20008.226476	16	0.0131913407525
24	black	e7e5	5	alpha.etakmit		367.760697	8005.622922	15	0.002309593326111111

Fails is count of failed computations for each task => it usually means that something went wrong on providers side.

Vm time is a time of running bestmove.js script running in VM.

Total time is a time between offer being sent to golem network and yajsapi events (WorkerFinished or ComputationFinished) => if that events are not triggered by yajsapi or sth goes wrong with processing it is not displayed.

Fails is count of failed computations for each task => it usually means that something went wrong on providers side.

Proposals column displays count of proposals sent to requestor before start of computation.

Cost is total cost of single task.

DEMOs

- Demo v0.3

https://www.youtube.com/watch?v=Wp_lJEeN7UA&feature=youtu.be&ab_channel=Pawe%C5%82Burgchardt

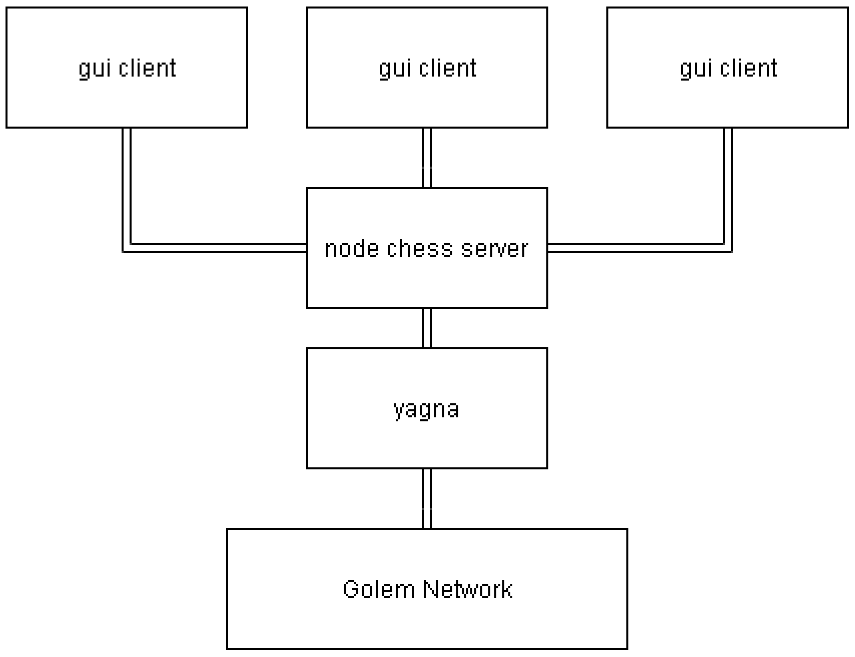
- Demo v0.2

https://www.youtube.com/watch?v=C65uTAZAsRA&list=UUxg1Vq50vwy7Pm3kFwb0ZQg&index=2&ab_channel=Pawe%C5%82Burgchardt

- Demo v0.1 (problem with some providers' payments)

  https://www.youtube.com/watch?v=cTD0zq7jURM&list=UUxg1Vq50vwy7Pm3kFwb0ZQg&index=3&ab_channel=Pawe%C5%82Burgchardt

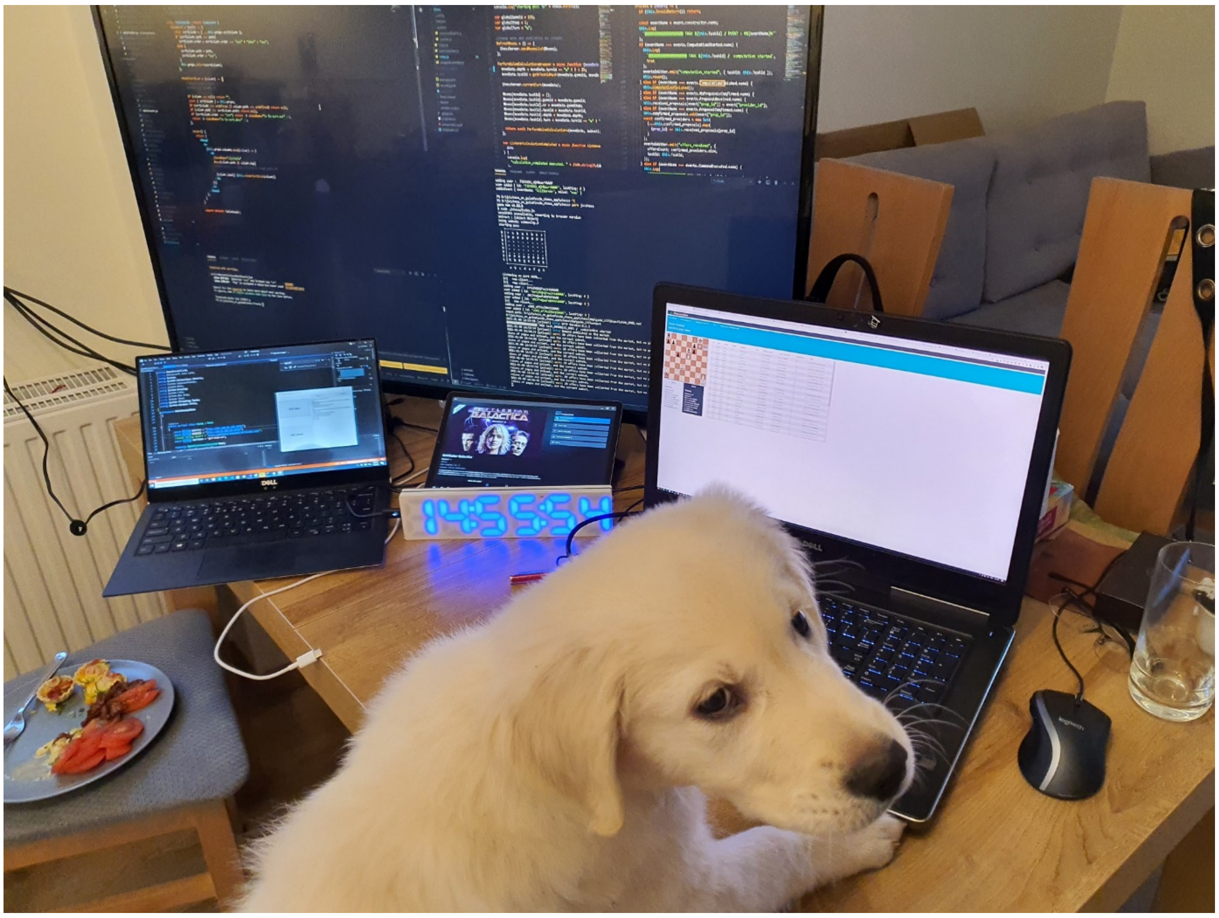
System architecture:



Authors:

Paweł Burgchardt & Indiana

12.2020 / 01.2021



License: