Server-Client commands

This documents explains the commands used in the communication between the front and back end of the system.

All commands are of the form “command data”, where command is the name of the command and data is the data send with the command.

# Messages from client to server

### Startup “startup token”

This is the first command the client should send to the server, together with the token of the session. This token is linked to the user and problem information. If the token is wrong, outdated or no token is send, then the connection is closed and the game cannot be played. If the token is correct, then other commands are accepted.

### start “start”

This command is send when the client wants to start the game. When this command is received by the server, then information about the game is send to the front end.

### end “end”

This command is send when the client wants to end the game.

### setPre “setPre precondition”

This command is used to send a precondition to the server, but not check if it is correct.

### setPost “setPost postcondition”

This command is used to send a postcondition to the server, but not check if it is correct.

### checkPre “checkPre”

This command checks the precondition send earlier on correctness and sends new weights for the spawning of sparks for the precondition path back.

### checkPost “checkPost”

This command checks the postcondition send earlier on correctness and sends new weights for the spawning of sparks for the postcondition path back.

### submitPre “submitPre precondition”

Combines the setPre and checkPre commands, therefore first setting the precondition and then checking it on correctness.

### submitPost “submitPost postcondition”

Combines the setPost and checkPost commands, therefore first setting the postcondition and then checking it on correctness.

### waveDone “waveDone score;deltaScore;money;health;towerCount;preSpawned;prePassed;postSpawned;postPassed;preDeltaHealth;postDeltaHealth;moneySpent;timeSpent”

This command is send when the client finished a wave of sparks. The data send with the command contains information about the wave and state of the game afterwards. The data contains the following:

* Score (int): Score after the wave.
* deltaScore (int): Score gained during the wave.
* money (int): Money after the wave.
* health (int): Health after the wave.
* towerCount (int): Total amount of towers.
* preSpawned (int[]): Amount of different sparks spawned for the precondition path.
* prePassed (int[]): Amount of different sparks that passed the precondition path.
* postSpawned (int[]): Amount of different sparks spawned for the postcondition path.
* postPassed (int[]): Amount of different sparks that passed the postcondition path.
* preDeltaHealth (int[]): Amount of health lost during the wave, because of the precondition path.
* postDeltaHealth (int[]): Amount of health lost during the wave, because of the postcondition path.
* moneySpent (int[]): Amount of money spent during the wave.
* timeSpent (int[]): Amount of time spent on certain parts of the game during the wave.

### finalScore “finalScore score”

Send the final score that is used in the highscores list, if the score is legitimate.

# Messages from server to client.

### startup “startup state”

Sends whether the startup command previously received by the server was correct or something went wrong. Possible returns are: “startup done” when the startup went succesful, “startup wrong” when the token was incorrect, “startup timeout” when the token is outdated or it took too long before receiving the startup command from the client.

### difficulty “difficulty value”

Send the difficulty of the current problem to the client.

### validPreTokens “validPreTokens tokens”

Send the tokens that may be used by the player for constructing the preconditions.

### validPostTokens “validPostTokens tokens”

Send the tokens that may be used by the player for constructing the postconditions.

### description “description text”

Send the description of the current problem.

### spawnWave “spawnWave amount;preWeights;postWeights;health;speed;spawnTime;specialSparkPercentages”

Send information about the next wave of sparks. The data contains the following

* Amount (int): Amount of sparks that should spawn.
* preWeights (int[]): Weights for what type of sparks should spawn for the precondition path.
* postWeights (int[]): Weights for what type of sparks should spawn for the postcondition path.
* Health (int): Health of the sparks that will spawn.
* Speed (int): Base speed of the sparks that will spawn.
* Spawntime (int): Time inbetween the spawning of sparks.
* SpecialSparks (int[]): Percentages on what type of special sparks should spawn.

### resultPre “resultPre newWeights”

Sends the result of checking the precondition, by sending new weights to modify the spawning of sparks for the precondition path.

### resultPost “resultPost newWeights”

Sends the result of checking the postcondition, by sending new weights to modify the spawning of sparks for the postcondition path.

### preFeedback “preFeedback counterExample”

Sends a counterexample for the precondition to the front end. The counterexample consists of several variables with their respective values.

### postFeedback “postFeedback counterExample”

Sends a counterexample for the postcondition to the front end. The counterexample consists of several variables with their respective values.

### path “path pathData”

Sends the path for the sparks to following during the game.

### hint “hint text”

Sends a hint about writing formal specifications.

### availableScore “availableScore score”

Sends the amount of score the player is allowed to gain during the current problem.

### progression “progression number”

Send the current progression to the client.

### finish “finish”

Send to the client when the game is finished.