This document explains the initial design for the TCP RCON support in King Arthur's Gold. At the end is an explanation of what ended up being changed for the real implementation. This document was written in "stream of consciousness" originally, so you will see how some design considerations evolved as you read onward. Originally written in ~May 2012, these comments and those at the end added November 2012.

--

- when the server starts, it creates a new thread used for managing/operating tcp rcon connections and requests
- communication is done with a std::list<some struct> that acts as a message queue between the main server thread and the tcp rcon thread. Probably there will be two of these, one for each direction. We probably could use a fifo/queue for this instead, we'll see which ends up best for other reasons
- a mutex is needed around reads and writes from the list being used for IPC
- the tcp rcon thread will use nonblocking IO with select() and support IPv6. Two options:
    - either set the timeout to something like 1 second, in which case this will not be exactly real time for executing commands
    - use NULL timeout so we block until a command arrives, however we end up with a problem that no output will be sent back to the client unless a command is received from one of them.
        - This can be avoided if the IPC used has a file descriptor, e.g. pipes. pipes are looking like a good and simple way to communicate, even if just to act as a trigger for when events are in the aforementioned list. hmm.
- Ok so the communication between the two will happen via pipes example. Reasoning is in the above bullet point nest. The events will still be pushed onto the list as structs, but the pipes will let us trigger select() for more nearly realtime behavior.
- nonblocking IO is used because it prevents the need for another thread for every admin connection (thus helping avoid a potential DoS vector and more IPC headache).
- For events coming from the server thread out to TCP rcon clients (I'm going to call it tcpr instead of typing TCP Rcon from now on), the server thread adds them onto the list (only if the config says it's enabled [and if the pipe is open for writing still?] then writes one byte (pretty much anything) to the pipe. This will unblock the select() in the tcpr thread, at which point it will flood that message out to all connected tcpr clients.
- For events coming from a tcpr client, the process is very similar. The tcpr thread locks the list with the mutex, adds the event (just a string with some metadata), and writes a byte to the pipe. The server thread picks this up because it is calling a nonblocking read from the pipe in every tick (maybe there's something better to do) to see if it thinks it should be reading from the thread. It could alternatively lock the list, check the size of it, and act if the list is length > 0. This might be better because it doesn't result in an extra system call every iteration. In this case, the pipe won't even be used in the direction of tcpr -> game thread. Some additional thoughts:
    - The game thread will handle these exactly as if they came from a connected game client who is an rcon admin

    - The tcpr thread will *not* send incoming commands directly back out to other clients, because that should be handled by the game thread (just like it does now, generationg [rcon from <player>] messages in the game console. This should generate outgoing messages from the game thread back to tcpr and take care of this for us unless the way it is implemented now really sucks.
- There will be a pair of available configurations to help prevent abuse by flooding the TCPR port (see below), probably more
- tcpr will require the rcon password to be sent as the first piece of data in the socket. If the first successful read() from a new connection doesn't result in bytes matching the beginning or entirety of the rcon password, the connection is immediately closed with no output.
- There probably should be a rate limit on how often a host can try to reconnect to the tcpr port.
- tcpr connections will take any valid rcon command, but typing /rcon won't be necessary since there's no need to differentiate between executing locally versus on the server (like there is when sending commands from in the client's console)

additional config settings added:

- sv_tcprcon boolean for whether tcp rcon is enabled
- sv_tcprcon_maxconn max number of simultaneous tcp admin connections (default 25)
- sv_tcprcon_maxconnhost max number of simultaneous tcp admin connections per host

--

Notes about differences with actual implementation:

- The messages from the main server thread to the tcpr thread (containing console output) ended up being written to the pipe fd directly. No metadata is currently needed, so plaintext with each message terminated by \n is fine.
- Commands from the tcpr clients to the server ended up being written to shared memory in a std::list<struct tcprCommand>. The net event loop in the main server thread just checks the size of this list every tick. It is protected by a mutex to prevent memory corruption. There are no blocking system calls inside the mutex-protected critical area, so no performance penalties will ever be incurred. This is actually described above, and is exactly how it ended up being implemented.
- The sv_tcprcon boolean config ended up being called sv_tcpr
- No additional configuration options have been added to fine tune tcpr for anti-abuse purposes yet.