Cyber Space Odyssey Documentation

Kendra Graham, Bryce Heitmeyer & Conrad Rife

1 Overview

Cyber Space Odyssey is a 3D game designed to help the players become familiar with the structure of networks and gain practice understanding, reading, and sending network traffic. Gameplay consists of up to five players that are all client to a single server, and each client must use their knowledge of networks in order to progress in the game.

The general setup of the game is a race between the players to solve a mystery, which they do by obtaining clues, each of which is a piece of a logic puzzle whose solution gives the answer to the mystery. To obtain a clue, a player must move to a certain location in the game world and successfully hold a multi-step communication with the server that involves solving a simple cypher, decryption, and manually sending two different kinds of network packets. Submitting their answer to the mystery also involves a multi-step communication with the server. Players can also send network packets that hinder the other players' progress, and are encouraged to find other ways to exploit the network structure to their own advantage.

2 Gameplay

The game is set in space around a very small planet earth that has five space stations in orbit around it. Each space station is surrounded by a different-colored transparent bubble, and a spaceship is docked at each station. Attached to each spaceship is a green crosshairs that shows the pilot where the nose is pointed, and a colored bubble that makes the ship more visible from a distance that can be turned on and off.

Figure 1: Close-up of space station and ship

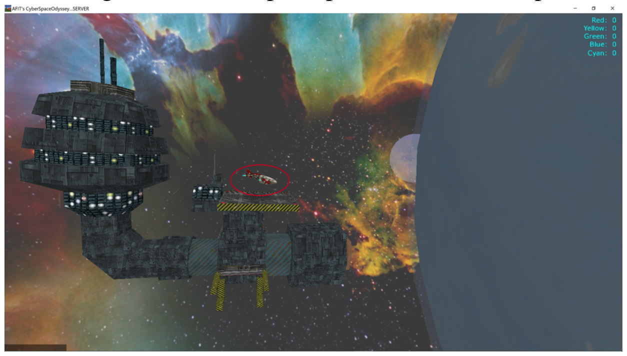
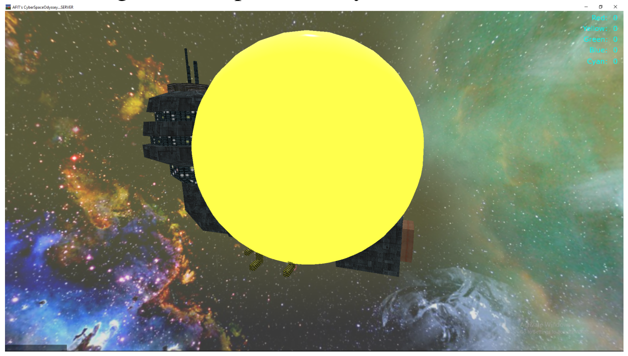
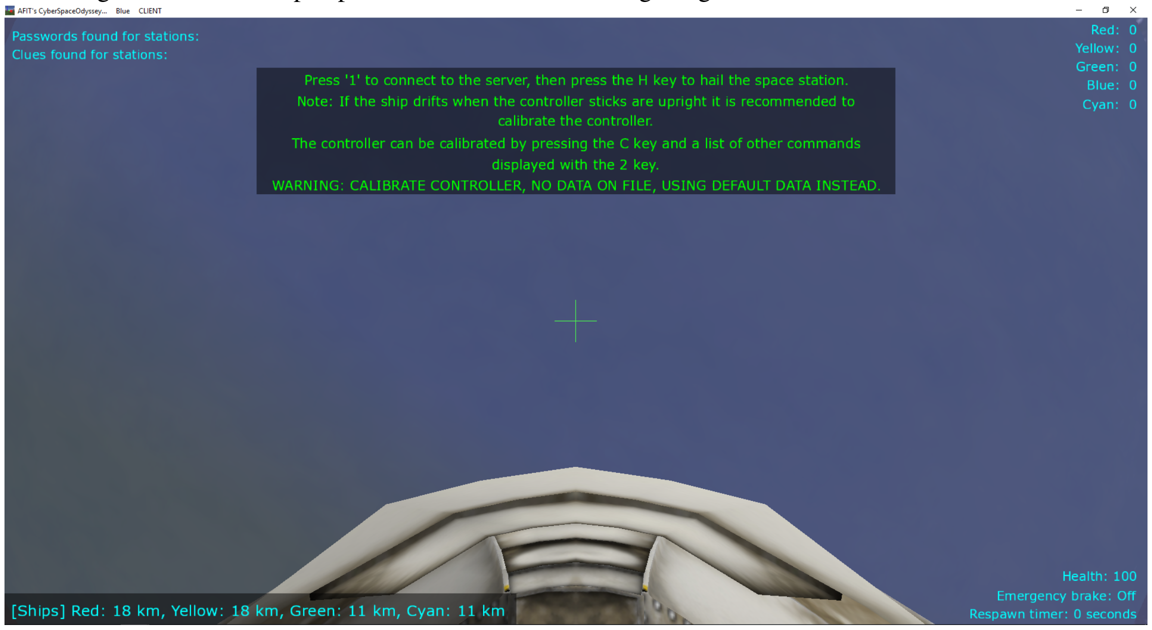


Figure 2: Ship's visibility bubble enabled



For both server and clients, there is a leaderboard displaying each team and its score in the top right corner of the screen. In addition to this, clients have a brief summary of game progress displayed in the top left corner, personal ship-related information in the bottom right corner, and the distance between their own spaceship and the other spaceships is shown on the bottom. The center of the screen also contains a GUI with information for the user throughout play.

Figure 3: The client perspective of the world, showing the green crosshairs and GUI elements.



AFIT's CyberSpaceOdyssey... Blue CLIENT

Passwords found for stations:
Clues found for stations:

Press '1' to connect to the server, then press the H key to hail the space station.
Note: If the ship drifts when the controller sticks are upright it is recommended to calibrate the controller.
The controller can be calibrated by pressing the C key and a list of other commands displayed with the 2 key.
WARNING: CALIBRATE CONTROLLER, NO DATA ON FILE, USING DEFAULT DATA INSTEAD.

Red: 0
Yellow: 0
Green: 0
Blue: 0
Cyan: 0

Health: 100
Emergency brake: Off
Respawn timer: 0 seconds

[Ships] Red: 18 km, Yellow: 18 km, Green: 11 km, Cyan: 11 km

Players must fly their spaceship to each space station in any order they choose. Once in range of a station, the player hails the station and receives a simple mathematical cypher to crack. When the player has cracked the cypher, they send a network packet back with the answer and the server sends an additional puzzle to solve that unveils a password, which may be encrypted. When the player has found and decrypted the password, they send a network packet back with the decrypted password and the server sends a clue, which also may be encrypted. Once they have found and decrypted all of the clues, they go to a station indicated by the answer to the mystery and submit their answer. Along the way, players can send a hack packet to the server that tampers with the motion of other spaceships.

Points are awarded for making progress in the game, and teams are ranked first by whether they completed the mission or not, and secondarily by points. In general, if one team does something before another team, the first team to do it earns more points for it.

3 Program Structure

`GLViewCyberSpaceOdyssey` is the top-level manager class. It stores and renders graphics information, sends and receives network messages; owns the master copy of all model classes, namely `JoystickState`, `TeamState`, and `Game State`; allows for controller calibration by linking the controller's input to the `Calibration` class; and handles device input from the keyboard, mouse and joystick, sending the raw joystick values received to `JoystickState` for processing and storage. If it is acting as a server, once per second it saves all of its data to an external file called `gamedataHH-MM.txt`, where `HH-MM` is the time that the server started running. It also stores a summary of game statistics to another file called `game results.txt`.

These classes handle graphics:

- **`CameraChaseSpaceShip`** allows us to see the world from a cockpit view.
- **`CyberSpaceOdysseyWayPoints`** is a graphics class that lets us use waypoints.
- **`WOPlanet`** draws a custom planet earth.
- **`WOSpaceShip`** draws a custom spaceship and calculates its motion.
- **`WOParticleSystem`** draws a spaceship's exhaust animation

These classes are for data modeling:

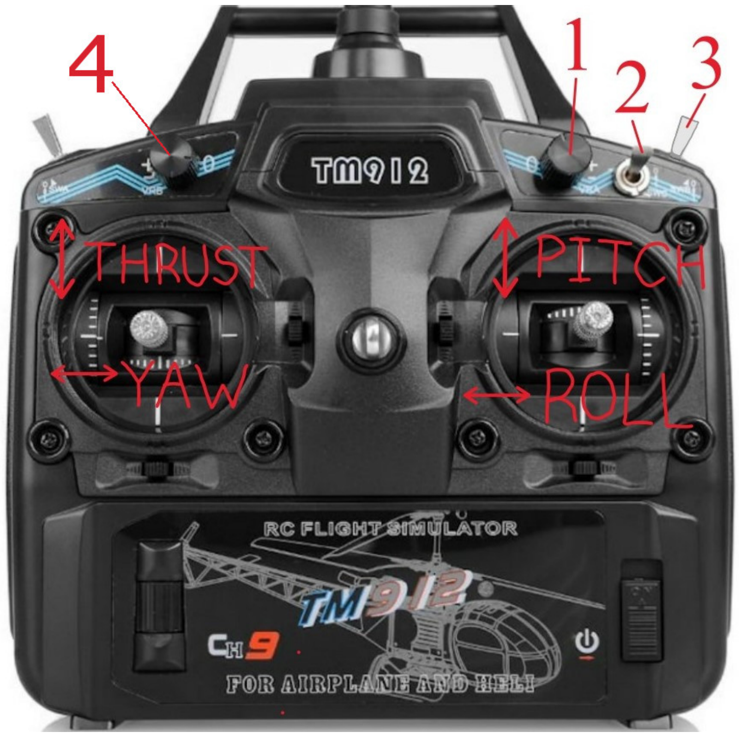
- **`GameState`** stores each team's progress in the game.
- **`TeamState`** stores the score and several boolean variables for each team.
- **`JoystickState`** is a wrapper for the raw joystick variables.

The following classes are all network messages:

- **`NetMsgAuthenticateGuess`** is part of a two-step authentication process in which clients submit the name of whom they think is guilty. In this step, the server sends a randomly generated key to the client.
- Clients connect to the server by pressing the number **1** on the keyboard, which automatically sends a **`NetMsgClient Join`**.
- Clients automatically notify the server of their position and shooting state at 60 hz with a **`NetMsgClientPose`**.
- Whenever a client earns a clue, the server sends a **`NetMsgClue`** to the client corresponding to that station, and the client who earned it can then use Wireshark to find it. If the client that ought to receive the packet is not connected, the server sends the clue to the client that earned it.
- The server sends a station cypher to a client with a **`NetMsgCypher`**, which does not print to the on-screen GUI.
- Clients manually send their answer to a station cypher with a **`NetMsgCypherResponse`**.
- Clients complete the two-step authentication process by manually sending a **`NetMsgGuess`**, which contains a string of the client's accusation along with the key received with the `NetMsgAuthenticateGuess`.
- The server prints to a client's on-screen GUI by sending a **`NetMsgGuiString`**.
- Clients can hack other clients by manually sending a **`NetMsgHack`** to the server that, once processed, will interfere with the motion of another spaceship.
- Client initiate communications with a station by pressing the **H** key, which automatically sends a **`NetMsgHailStation`** to the server.
- When clients have decrypted a password, they manually send a **`NetMsgInformationRequest`** to the station, which sends an encrypted clue in response.
- Clients begin the guessing process by sending a **`NetMsgInitiateGuess`** containing their client index to the server.
- Whenever a client earns a password, the server sends a **`NetMsgPassword`** to the client corresponding to that station, and the client who earned it must find it on Wireshark. If the client that ought to receive the packet is not connected, the server sends the password to the client that earned it.
- The server sends a **`NetMsgTeamState`** to all clients once per second, which contains the current `TeamState` and a brief summary of game progress specific to each client.
- The server sends the position, health, and laser visibility of each ship to all clients at 60 hz via **`NetMsgUpdatePose`**.

4 Spaceship Motion

The user controls spaceship motion through the joystick, and `JoystickState` is used to process the device input. `JoystickState` scales the raw joystick input to a range of -1 to 1, making sure that when the joystick is upright, its value is zero. It stores a running average of the most recent values such that at any given time, 30% of a given axis value comes from the most recent joystick input and the other 70% comes from the previously stored value. The rate of motion for each axis is directly related to the square of the current `JoystickState` value for that axis.



The joystick allows control of the spaceship over four axes: thrust, roll, pitch, and yaw. Tilting the left stick forward moves the ship forward and tilting it left turns the nose to the left. Tilting the right stick forward pitches the nose of the ship down, and tilting it to the right rolls the ship to the right. The right-hand vertical switch (see [2]) is an emergency brake; switching it towards you turns the brakes on and switching it away from you turns the brakes off. The right-hand horizontal switch (see [3]) is a windshield wiper; it clears the screen of GUI messages. The left-hand and right-hand dial (see [1] and [4]) is a laser shooter; turning the dial to the right turns it on and turning it to the left turns it off.

`TeamState` also influences motion through a set of boolean values which control spaceship motion along a certain axis. When set to true,

- `ailerons_jammed` causes the spaceship to roll to the right at a constant rate
- `elevator_jammed` causes the spaceship to pitch forward at a constant rate
- `rudder_jammed` causes the spaceship to yaw to the left at a constant rate
- `throttle_jammed` causes the spaceship to move backwards at a constant rate

Each spaceship has a copy of the main `JoystickState` and `TeamState` pointers so that it can correctly update its own position.

If a spaceship is hit by a laser, the ship loses health at a constant rate. Once a ship hits 0 health, its health is reset to 100 and the player cannot control the spaceship for 15 seconds, making the ship drift off in the direction it was facing when disabled. In addition, there are certain external limits to where the spaceship can go. The spaceship is not allowed to fly into earth or spend more than 3 seconds in the atmosphere. If it does, it is sent back to its original position, the brakes are forcibly applied for 30 seconds, and a message is displayed explaining what just happened. The ship is also notified when it enters and leaves a space station's bubble, when it enters and leaves the atmosphere, when it gets more than 30 km from the earth and when it comes back within 30 km of the earth.

5 Server-Client Relations

The same program can function as either server or client, depending on a boolean variable in the configuration file, `aftr.conf`. The variable is called `isServer`, and setting it to true causes the program to run as the server and setting it to false causes the program to run as a client. For a server, `aftr.conf` must have a # in front of `NetServerListenerPort`. The client must not have the #, and the `NetServerListenerPort` number must be set to the appropriate value for the team color that has been chosen, which should also be specified next to `clientTeamName` and spelled exactly as written in the nearby comment field.

During normal gameplay, there is one server and up to five clients - one client per team. All network communications are made directly between the server and a client, and in general, the server owns the authoritative copy of all data. The server has an array that stores a socket for each client, and the client stores a socket to the server. Based on its configuration file (`aftr.conf`), the client knows the server's IP address and port, and it also knows what index it will be in the server's array of clients. Therefore, it is the client's responsibility to connect itself to the server by sending a `NetMsgClientJoin` to tell the server its IP address, port, and client index. All network messages are UDP protocol, except for `NetMsgClientJoin`.

5.1 Continuous Broadcasts

Once a client has connected to the server, there are three kinds of network messages that are continuously broadcast over the connection:

- The client makes the server aware of its position, orientation, and laser shooting state by sending a `NetMsgClientPose` at 60 hz. This always overwrites the server's copy of that particular data.
- The server makes all clients aware of all spaceship positions, their healths, and what lasers should be visible by sending a `NetMsgUpdatePose` at 60 hz. This always overwrites the client's data for all other spaceships, but it never affects the client's own position.
- The server broadcasts its copy of `TeamState`, along with a short summary of game progress, to all connected clients via a **`NetMsgTeamState`**. The client always uses the data received to overwrite its own copy of `TeamState`.

5.2 Obtaining a Clue

Clients must send a specific sequence of network packets in order to obtain game clues from the various space stations. The process is as follows:

1. The client moves within range of a space station and presses the **H** on the keyboard to send a **`NetMsgHailStation`**, which contains the client index and the station index.

2. The server checks if client is truly in range of the requested station, and if so, the server sends a **`NetMsgCypher`** containing the string of the station's mathematical cypher to the client. The server also notifies `GameState` that the client hailed that particular station. The `NetMsgCypher` does not print to the on-screen GUI, so the client must use Wireshark to look at the packet contents and solve the cypher.

3. The client solves the cypher and manually crafts a **`NetMsgCypherResponse`** containing the answer to the cypher and sends it to the server.

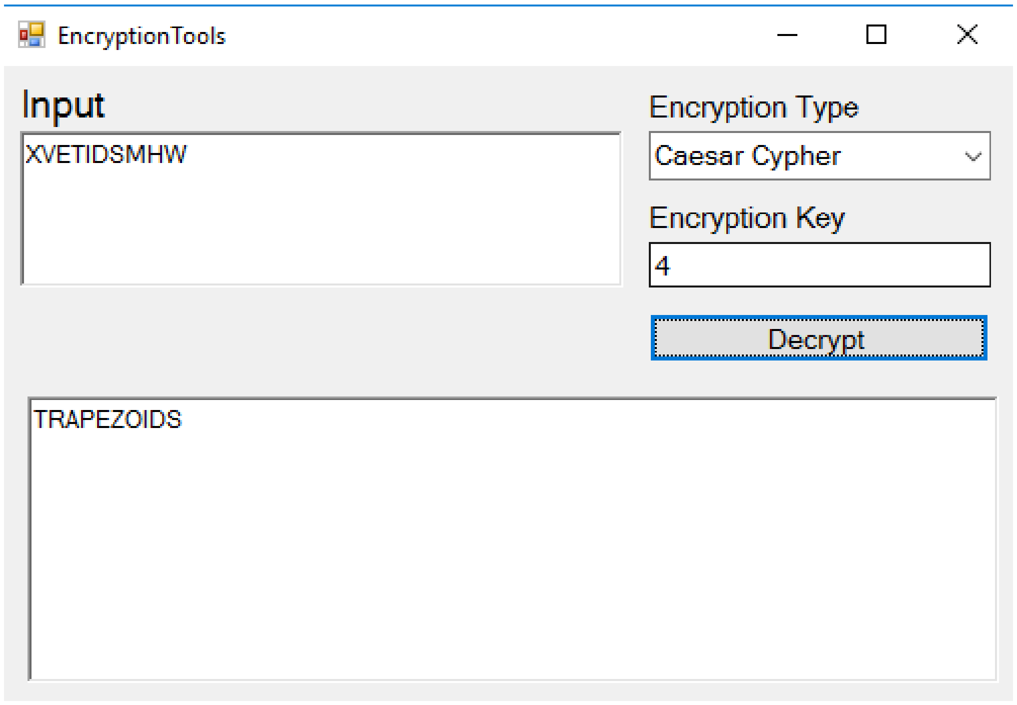
4. Upon receiving the `NetMsgCypherResponse`, the server checks if the client is still in range of the station. If so, it sends a puzzle containing the possibly encrypted password to the client corresponding to that station - Red station will send the puzzle to the Red client; Blue station will send to the Blue client, and so on. If the client corresponding to the station is not connected, the station instead sends the puzzle to the client that send the `NetMsgCypherResponse`. Additionally, the server also notifies `GameState` that the client answered the cypher correctly and sends a `NetMsgGuiString` telling the client that the cypher was answered correctly.

5. The client uses the cypher answer or other means to find and possibly decrypt the station password and manually crafts a **`NetMsgInformationRequest`** containing the decrypted station password and sends the packet to the server.

6. Upon receiving the `NetMsgInformationRequest`, the server checks if the client is still in range of the station and whether the password is correct. If so, the server sends the possibly encrypted station clue to the client corresponding to the station, just as it did with the puzzle, again sending the clue to the ship that sent the `NetMsg` if the client that ought to receive it is disconnected. The server also notifies `GameState` that the client decrypted the password and earned a clue. If the client sent an incorrect password, the server notifies `GameState` and the client loses points. The server also sends a message to the client telling them whether or not they received a clue.

7. The client can now move on to the next station and use the decrypted station password or other means to decrypt the clue at leisure.

In order to facilitate the decryption of some clues and passwords (but not all of them) there is an external application called EncryptionTools, which decrypts the input according to the encryption type selected and encryption key entered.



Input: XVETIDSMHW

Encryption Type: Caesar Cypher

Encryption Key: 4

TRAPEZOIDS

Figure 4: The EncryptionTools application

5.3 Making an Accusation

Once a player has obtained and decrypted all the clues, they have enough information to determine who the rogue operative is. The process of formally accusing the culprit is as follows:

1. The client flies to the rogue operative's home space station and sends a **`NetMsgInitiateGuess`**, which contains the client index.
2. When the server receives a `NetMsgInitiateGuess`, it generates a random signed nonzero integer, stores it in `this->guessKeys[clientIndex]`, and sends it to the client wrapped in a **`NetMsgAuthenticateGuess`**.
3. The client must find the random key generated by the server using Wireshark, and then send a `NetMsgGuess` to the server containing its client index, the random number it received, and the name of the person accused.
4. When the server receives a `NetMsgGuess`, it checks whether the number received matches the most recently generated random number for that client. If it does, and the client has not previously made a correct guess, the server checks whether the guess is correct. If it is, the server notifies `GameState` that the client guessed correctly, resets `this->guessKeys[clientIndex]`, and sends a message to the client telling it that the guess was correct.

5.4 Client Hacks

Clients have the ability to "hack" other clients by sending a packet to the server that, if successful, will temporarily alter the values broadcast in `NetMsgTeamState` in order to tamper with the motion of their own or another team's spaceship. A successful hack also earns a few extra points for the team that carries it out.

In order to hack another team, a client manually sends a `NetMsgHack` packet, which contains the index of the hacker client, the index of the target client, the index of the value in `TeamState` that the hacker wishes to modify, and the boolean value the hacker wishes the `TeamState` value to be set to. The indices of those booleans are as follows:

- Index [0] is `ailerons_jammed`, which causes the ship to roll at a constant rate
- Index [1] is `elevator_jammed`, which causes the ship to pitch at a constant rate
- Index [2] is `rudder_jammed`, which causes the ship to yaw at a constant rate

- Index [3] is `throttle_jammed`, which causes the ship to go backwards at a constant rate

When the server receives a `NetMsgHack`, it first checks that all internal values are valid. If so, it checks whether the hacker client is within range (5 km) of the target client and whether it's been at least 20 seconds since the hacker client's last hack. If all checks are cleared, the server notifies `GameState`, creates a tuple that contains the index of what to change in `TeamState`'s boolean array, the value to change it to, and a counter, and adds this tuple to a list of hacks. Then, every time the server broadcasts `TeamState`, it applies every change contained in the list of hacks to the `NetMsgTeamState`'s boolean array and decrements each counter in the list, removing hacks with expired counters. This way, the original copy of `TeamState`'s boolean array is never changed, only the temporary copy given to `NetMsgTeamState`.

5.5 Other Network Messages

Whenever the server sends a message for the client to read, it sends a **`NetMsgGuiString`**, which prints to the client's windshield for them to read.

5.6 Sending and Receiving Network Messages

Clients use Wireshark to find and look at any packet that is not automatically processed by the client program.

In order to send network packets, clients use an external program written as a Visual Studio project, which uses the C++ boost libraries. The project includes an example of how to send a correctly formatted `NetMsg` that the server will be able to recognize. Part of this involves entering the correct header ID that identifies which type of `NetMsg` the UDP packet contains. The mapping between `NetMsg`s and the header IDs may be found by scrolling to the top of the game's console window while it's running.

6 Game Modeling

All game progress and information is stored on the server and given to the client on a need-to-know basis. `GameState` randomly generates a unique set of cypher questions, puzzles, and clues each time the server is initialized, so there is no way for the client to know ahead of time what the cypher questions, passwords, and clues are or who the culprit is.

Station cyphers are randomly generated simple math problems with integer answers ranging from 2 to 25. Some station passwords are randomly chosen from a list of over 30 single-word isograms, ranging between 10 and 13 characters long, and some are public-private key encryption style math problems that involve having to find the two prime numbers that, when multiplied, produce the provided number in the `NetMsg` found on WireShark. Station clues come from a template and are filled in by randomly selecting a certain combination of names from a list, then randomly assigned to the different stations.

The server does its own encryption using a Caeser Cipher or Base64 for the passwords and a keyword substitution or Base64 encryption for the clue. The key for a station's password is the answer to its cypher question, and the key to a station's clue is the (unencrypted) station password. For a non-mathematical password of length n, the first n letters of the new alphabet are the letters of the password, and the rest of the letters of the new alphabet are all letters not used in the password itself, in alphabetical order. So, if the password were EXHAUSTING, then the alphabet would look like this:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
E	X	H	A	U	S	T	I	N	G	B	C	D	F	J	K	L	M	O	P	Q	R	V	W	Y	Z

Information about each client's progress in the game is stored as boolean values: whether a particular client has hailed a particular station or not, whether a particular client has received a clue from a particular station or not, whether a particular team has made a correct guess or not, and so on. `GameState` stores these boolean values in parallel arrays, some of them two-dimensional and some of them one-dimensional. The only exception is an array that stores the number of hacks each team has done; that array stores integers instead of booleans. `GameState` provides getters and setters for these values that internally use a pointer to the master copy of `TeamState` to update the score whenever called.

Clients win by both convicting the rogue operative and earning the most points. More points mean a higher ranking, but a team that has convicted the operative will always rank higher than a team that didn't. As a general rule, the first players to achieve something earns the most points for it, the second player to do it earns one point less than the first player did, the third player to do it earns one point less than the second player, and so on.

7 Miscellaneous

7.1 Setup

Clients play the game with three computers. The first runs the main program and has a keyboard, mouse, joystick, wireless network card, and either two monitors or one projector and one monitor. The second also has a wireless network card and has Microsoft Visual Studio 2015 installed, which allows it to run the UDPPacketSender program, which sends UDP packets. The third has a wireless network dongle that allows it to capture packets, and clients run Wireshark on this computer to see all network traffic. Besides this, there must be a seperate computer with a wireless network card to be the server.

Each client must have a joystick and choose a client color that is not already in use, and the aftr.conf file must have the following values set:

- isServer must be set to 0 for a client and 1 for a server
- NetServerListenPort must not have a # starting the line and must have have the port number that matches the team name, as indicated in the comments

When starting the server, it is advised to delete the `gamedataHH-MM.txt` files from the folder, so that in the event of a server crash it will be easier to find the most one that pertains to the current game.

7.2 Restoring the Game From a File

Once per second, the server saves all data relevant to the current game state to a file called **`gamedataHH-MM.txt`**, where `HH` is the hour and `MM` are the minutes of the time that the game began. If the server crashes in the middle of a game, data can be restored by pressing **3** on the server keyboard, which will prompt for a filename to be entered in the console window. Find the most recently modified `gamedataHH-MM.txt` file in the server's folder, type its name into the console window, and that game will be reloaded.

7.3 External Files

- **`gamedataHH-MM.txt`** contains all data about the current game state, updated once per second, and can be used to restore a previous game in the event of a server crash. These files can safely be deleted at any time.
- **`gameresults.txt`** contains a summary of game progress for each team, updated once per second. It can safely be deleted at any time.
- **`SSPECSfile.txt`** contains the third puzzle and can be altered slightly if desired before running the server. It is very important to not delete this file.
- **`EncryptionTools.exe`** is an external utility application used to decrypt the passwords and clues.

7.4 Keypresses

- Pressing **1** on a client will connect the client to the server if not already connected.
- Pressing **2** on a client or server will list all of these keypress commands to the in-game GUI.
- Pressing **3** on the server prompts the user to type a filename into the console, and then reloads all game data from that file. Entering the name of an incorrectly formatted file will crash the program.
- By default, every ship besides the client's own ship is surrounded with an opaque colored bubble to help others see it. Pressing **TAB** turns the visibility of the bubbles on and off.
- Pressing **DELETE** causes the ship to self-destruct.
- Clients press the **H** key to hail a nearby space station.
- Clients press the **C** key to start calibrating their joystick while following the in-game instructions. Once finished, the player presses **C** again to finish calibration. This is recommended to do if the ship movement seems off or their is no data on file.
- The spaceship can also be controlled with keypresses, if necessary. Pressing **W**, **S**, **A**, or **D** causes the spaceship to thrust forward and backward and yaw left and right respectively.
- Pressing **Q**, **E**, **R**, or **F** causes the spaceship to roll left and right and pitch down and up respectively.
- Holding **LEFT SHIFT** for the client, halves the speed of all movement (for increased precision).
- Pressing **SPACE** activates the laser.
- Pressing **LEFT ARROW** or **RIGHT ARROW** for the server, set's the server's camera view to each individual client's view. This allows the person running the server to view what the client ship's are doing without having to chase them manually.
- Pressing **RIGHT SHIFT** for the client, alternates the user's view between first and third person. However, for the server, sends the server's camera back to free roam mode, if the server was viewing a client spaceship.