Prefixes:

CS\_: Sent from client to server

SC\_: Sent from server to client

IS\_: Internal Server

CLIENT TO SERVER Commands

|  |  |
| --- | --- |
| **Protocol** | **Description** |
| Connection Oriented |  |
| **CS\_CLOSE\_GAME** | Will close the game for the player, and the connection to the server.  ***void CloseGame(string playerName)*** |
| **CS\_LOGIN\_PLAYER** | Sends a request to the server for the player to connect.  ***boolean LoginPlayer(string playerName)*** |
| **CS\_LOGOUT\_PLAYER** | Requests that the server disconnect the player.  ***void LogoutPlayer(string playerName)*** |
|  |  |
| Game Viewer |  |
| **CS\_REFRESH\_GAMES** | Sends a request for a list of refreshed games.  ***void RefreshGames(string playerName, uint32 lobbyID)*** |
| **CS\_JOIN\_GAME** | Requests that the player joins the selected game.  ***boolean JoinGame(string playerName, uint32 gameID)*** |
| **CS\_CREATE\_GAME** | Request sent to the server to create a game with the player as the host.  ***boolean CreateGame(string playerName)*** |
|  |  |
| Game Lobby |  |
| **CS\_LEAVE\_GAME** | Requests that the player leave the game they are currently in.  ***void LeaveGame(string playerName, uint32 gameID)*** |
| **CS\_DISBAND\_GAME** | Requests that the game is disbanded.  ***void DisbandGame(uint lobby ID)*** |
| **CS\_JOIN\_TEAM** | Requests that the player joins the team they have selected  ***boolean JoinTeam(string playerName, uint lobbyID, uint teamID)*** |
| **CS\_READY\_STATUS** | Updates the player ready status, and sends it to the server  ***void UpdatePlayerReady(string playerName, boolean readyStatus)*** |
| **CS\_BENCH\_PLAYER** | Requests that the player is sent back to the bench if they are currently on a team.  ***void BenchPlayer(string playerName, uint lobbyID)*** |
| **CS\_SET\_GAMEPLAY\_OPTIONS** | *Host of game only.* Will send to the server the new gameplay options for the players.  ***void SetGameplayOptions(uint gameID, ????? options)*** |
|  |  |
| Chat |  |
| **CS\_SEND\_MESSAGE** | Will send a message to the server meant for all the players within that game.  ***void BroadcastMessage\_Game(string senderName, uint gameID, string message)*** |
| **CS\_SEND\_MESSAGE** | Will send a message to the server meant for all the players within the game prep. lobby.  ***void BroadcastMessage\_Lobby(string senderName, uint lobbyID, string message)*** |
| **CS\_SEND\_WHISPER** | Will send a whisper request to the server for a specified person.  ***bool Whisper(string senderName, string recipientName)***  *returns false if recipient is not found* |
| **CS\_SEND\_PARTY\_MSG** | Will send a message to the server meant for players in the same party - while in game.  ***void BroadcastMessage\_Team(string senderName, uint teamID, string message)*** |
| **CS\_SEND\_PARTY\_MSG** | Will send a message to the server meant for players in the same party - while in the game lobby.  ***void PartyMessage\_Lobby(string senderName, uint lobbyID, string message)*** |
|  |  |
| In-Game Commands |  |
| **CS\_KICK\_PLAYER** | Requests to the server that a designated player is kicked from the match.  *server:* ***void KickPlayer(string playerName)*** |
| **CS\_PAUSE\_GAME** | A request sent to the server to pause the game. - When two are received in X amount of time the game will pause.  ***boolean PauseRequest(string playerName, uint gameID)*** |
| **CS\_END\_GAME** | *Game host only.* Will send a request to the server to save and quit the current game.  ***boolean SaveAndQuit(uint gameID)*** |
| **CS\_RESUME\_GAME** | *Game host only.* Will send a request to server to resume game from a paused status.  ***boolean ResumeGame(string hostPlayerName)*** |
| **CS\_QUIT\_GAME** | Tells the server that a player has left the game  ***void QuitGame(string playerName, uint gameID)*** |
|  |  |
| Gameplay |  |
| **CS\_PLACE\_TOWER** | Sends a requests to the server to place a tower in the designated spot.  ***boolean PlaceTower (string playerName, uint towerID, uint gridspace)***  *returns true if the tower placement is allowed and false if not allowed* |
| **CS\_SUMMON\_MINION** | Sends a request to the server to place a minion at the starting portal.  ***boolean SummonMinion(uint teamID, string playerName, uint minionID)***  summons a minion for team by player at startPortal  returns true if successful |
| **CS\_RESEARCH\_MINION** | Sends a request to research a minion for the team.  ***boolean ResearchMinion(string playerName, uint minionID)***  *returns true if successful* |
| **CS\_RESEARCH\_TOWER** | Sends a request to research a tower for the team.  ***boolean ResearchTower(string playerName, uint towerID)***  unlocks the tower type for the player if successful  returns true if successful |

SERVER TO CLIENT protocols

|  |  |
| --- | --- |
| **Protocol** | **Description** |
| Connection |  |
| **SC\_LOGIN\_PLAYER** | the boolean response from a player requesting to join a game  ***void PlayerLogin(boolean response, uint gameID)*** |
|  |  |
| Game lobby |  |
| **SC\_REFRESH\_GAMES** | Returns the games in the lobby  ***void RefreshGames(string playerName, uid lobbyID, ??????[ ] games)*** |
| **SC\_JOIN\_GAME** | the boolean response from a player requesting to join a game  ***void PlayerJoinedGame( boolean response, uint gameID)*** |
| **SC\_CREATE\_GAME** | the boolean response from a player requesting to createa game  ***void PlayerCreatedGame( boolean response, uint gameID)*** |
| **SC\_DISBAND\_GAME** | the boolean response from a player requesting to disband a game  ***void PlayerDisbandedGame( boolean response)*** |
| **SC\_JOIN\_TEAM** | the boolean response from a player requesting to join a team  ***void PlayerJoinedTeam( boolean response, uint teamID)*** |
| **SC\_BENCH\_PLAYER** | the boolean response from a player requesting to bench another player  ***void PlayerBenched(string playerName, boolean response)*** |
| **SC\_UPDATE\_LOBBY** | updates the players with the current version of the game lobby (who is on what team, etc)  ***void UpdateLobby(????? LobbyInfo)*** |
|  |  |
| Chat |  |
| **SC\_SEND\_MESSAGE** | Will send a message to all players in the specified game.  ***void UserChatMessage(string playerName, string senderName, string message, MessageType type)*** |
|  |  |
| In-Game Commands |  |
| **SC\_PAUSE\_GAME** | the boolean response from a player requesting to pause a game  ***void PlayerPausedGame(boolean response)*** |
| **SC\_END\_GAME** | the boolean response from a player requesting to end a game  ***void PlayerEndedGame(boolean response)*** |
| **SC\_RESUME\_GAME** | the boolean response from a player requesting to resume a game  ***void PlayerResumedGame(boolean response, GameInfo gameInfo)*** |
| **SC\_QUIT\_GAME** | the boolean response from a player requesting to quit a game  ***void PlayerQuitGame(boolean response)*** |
|  |  |
| Gameplay |  |
| **SC\_SUMMON\_MINION** | ***void SendMinionToClient(Connection user, Command command, Message arguments)***  gets called multiple times by BroadcastMinionToGame(); |
| **SC\_PLACE\_TOWER** | boolean value sent back to client |
| **SC\_PLACE\_TOWER\_BC** | ***void SendTowerToClient(Connection user, Command command, Message arguments)***  gets called multiple times by BroadcastTowerToGame(); |
| **SC\_SUMMON\_MINION** | boolean value sent back to client  ***void sendMinionInfo(string playerName, boolean minionAcknowledgement)*** |
| **SC\_UPDATE\_ECONOMY** | ***void UpdateClientEconomy(Connection client, uint[] economyValues)***  updates one client’s economy  called multiple times by UpdateEconomy(); |
| **SC\_UPDATE\_BASE\_HP** | ***void UpdateBaseHP(uint TeamID)***  updates remaining HP for the 2 main bases |
| **SC\_ASSIGN\_RESOURCE\_NODE** | ***void AssignResourceNode(uint TeamID, uint ResourceNodeID)***  Assigns a particular resource node to a particular team once they have captured it using their ZoI |
| **SC\_RESEARCH\_MINION** | boolean value sent back to client |
| **SC\_RESEARCH\_TOWER** | boolean value sent back to client |

Internal SERVER protocols \* note: there will be many more here -Alex

|  |  |
| --- | --- |
| **Protocol** | **Description** |
| IS\_PLACE\_TOWER\_BC | void BroadcastTowerToGame(uint gameID, uint towerID, uint gridspace, string playerName)  broadcasts a newly placed tower to other players in a game |
| IS\_ADJUST\_ZOI | void adjustZoI(uint gameID)  goes through each tower and adjusts the ZoI map in the game model |
| IS\_SUMMON\_MINION\_BC | void BroadcastMinionToGame(uint gameID, uint minionID, string playerName)  broadcast new minion to clients |
| IS\_UPDATE\_MINION\_LIST | goes through minions and updates them - essentially moving them and checking current HP, etc |
| IS\_UPDATE\_TOWERLISTS | goes through towers and updates the towerlists (lists of enemies within range)  and checks if tower can fire then chooses an enemy and updates the game’s projectile list |
| IS\_UPDATE\_PROJECTILES | goes through all projectiles active in a game and updates them (projectiles are represented as the shooter and shootee so the client can just animate them based on speed stats) The server determins if a hit occurs and updates the correct minion |
| IS\_UPDATE\_ECONOMY | void UpdateEconomy(uint gameID, uint[] economyValues)  Pushes economy updates to clients in a game |

**Pseudocode**

//CLIENT TO SERVER Commands

//Connection Oriented

//CS\_CLOSE\_GAME

void CloseGame(string playerName)

{

//go through hashtable of online players

//remove reference of player from current game

//return player to main menu

}

//CS\_LOGIN\_PLAYER

boolean LoginPlayer(string playerName)

{

//add player to online players hashtable

}

//CS\_LOGOUT\_PLAYER

void LogoutPlayer(string playerName)

{

//remove player from online players hashtable

//return player to main menu (with the login option)

}

//Game Viewer

//CS\_REFRESH\_GAMES

void RefreshGames(string playerName, uint lobbyID)

{

//go through list of all avaialble games connected to the server

//display them to current player's client

}

//CS\_JOIN\_GAME

boolean JoinGame(string playerName, uint gameID)

{

//add player to the list of players in the gameID

}

//CS\_CREATE\_GAME

boolean CreateGame(string playerName)

{

//create a new game and add it to the list of running games on the server

//add the player to the list of players in that game

//flag player as host

}

//Game Lobby

//CS\_LEAVE\_GAME

void LeaveGame(string playerName, uint lobbyID)

{

//go through hashtable of online players

//remove reference of player from current game

//return player to main menu

}

//CS\_DISBAND\_GAME

void DisbandGame(uint lobby ID)

{

//return all players in the lobby to main menu

//remove lobby from lobby list

}

//CS\_JOIN\_TEAM

boolean JoinTeam(string playerName, uint lobbyID, uint teamID)

{

//check if teamID has room for the player

//if yes:

//remove player from bench

//add player to team list

//return true

//if no:

//return false;

}

//CS\_READY\_STATUS

void UpdatePlayerReady(string playerName, boolean readyStatus)

{

//set player's ready flag to readyStatus

}

//CS\_BENCH\_PLAYER

void BenchPlayer(string playerName, uint lobbyID)

{

//remove player from a team list if he/she is in one

//add player to bench list

}

//CS\_SET\_GAMEPLAY\_OPTIONS

void SetGameplayOptions(uint gameID, ????? options)

{

//go through game options model and set variables accordingly

}

//Chat

//CS\_SEND\_MESSAGE

void BroadcastMessage\_Game(string senderName, uint gameID, string message)

{

//find all players in the gameID

//send message to all those players' clients

}

//CS\_SEND\_MESSAGE

void BroadcastMessage\_Lobby(string senderName, uint lobbyID, string message)

{

//find all players in the lobbyID

//send message to all those players' clients

}

//CS\_SEND\_WHISPER

bool Whisper(string senderName, string recipientName)

{

//find the recipient from hastable of online players

//if found

//send message to recipient

//return true

//else

//return false

}

//CS\_SEND\_PARTY\_MSG

void BroadcastMessage\_Team(string senderName, uint teamID, uint gameID, string message)

{

//find all players in the teamID in the gameID

//send message to all those players' clients

}

//CS\_SEND\_PARTY\_MSG

void PartyMessage\_Lobby(string senderName, uint lobbyID, uint lobbyID, string message)

{

//find all players in the teamID in the lobbyID

//send message to all those players' clients

}

//In-Game Commands

//CS\_KICK\_PLAYER

void KickPlayer(string playerName)

{

//find which gameID the player is in

//remove player back to game viewer screen

//update the game representation

//notify other players involved in the game

}

//CS\_PAUSE\_GAME

boolean PauseRequest(string playerName, uint gameID)

{

//check for another pause request from the same gameID but from a different player

//if a 2nd one is received within X seconds

//pause the game

}

//CS\_RESUME\_GAME

boolean ResumeGame(string hostPlayerName, uint gameID)

{

//check to make sure player is host

//resume gameID

}

//CS\_END\_GAME

boolean SaveAndQuit(string playerName, uint gameID)

{

//make sure player is host

//save all game info to local db

//remove all players to game viewer/main menu

//remove gameID from list of running games

}

//CS\_QUIT\_GAME

void QuitGame(string playerName, uint gameID)

{

//remove player from game

//update game representation

//notify other players

}

//Gameplay

//CS\_PLACE\_TOWER

boolean PlaceTower (string playerName, uint towerID, uint gridspace)

{

//check if tower has been researched for the player

//check if player has enough mana to build it

//check if gridID is a buildable tile

//check if anything else is on that gridspace already

//check if gridspace is in player's team's zone of influence

//if the above is yes

//deduct cost from mana

//add towerID to gridspace

//add tower to tower list in the game representation

//check if gridspaces in ZoI of tower are already marked as ZoI for the team

//if not, mark as ZoI for team

//return true

//else return false

}

//CS\_SUMMON\_MINION

boolean SummonMinion(string playerName, uint minionID)

{

//check if minion has been researched for the player

//check if player has enough mana to summon the minion

//if yes:

//deduct cost from mana

//increase MpS (mana per second) as per minions MpS value

//spawn minion at player's team's portals

//add minion to minion list in the game representation

//update game representation

//return true

//else return false

}

//CS\_RESEARCH\_TOWER

boolean ResearchTower(string playerName, uint towerID)

{

//check if player has enough mana to research the tower

//if yes:

//flag tower as researched for player

//deduct cost from mana

//return true

//else

//return false

}

//CS\_RESEARCH\_MINION

boolean ResearchMinion(string playerName, uint minionID)

{

//check if player has enough mana to research the minion

//if yes:

//flag minion as researched for player

//deduct cost from mana

//return true

//else

//return false

}

//Internal SERVER protocols

//IS\_UPDATE\_MINION\_LIST

void UpdateMinions(uint gameID)

{

//loop trough all minions in the minion list in the game ID

//check for HP

//if dead

//despawn

//check if at enemy base

//if yes:

//do damage to enemy base

//remove MpS bonus from player who had spawned it

//despawn minion

//update position

}

//IS\_UPDATE\_TOWERLISTS

void UpdateTowers(uint gameID)

{

//loop through all towers in the tower list in the gameID

//loop through all minions in the game

//if any minion is in range

//fire at minion

}

//IS\_UPDATE\_PROJECTILES

void UpdateProjectiles(uint gameID)

{

//loop through all projectiles in the list in the gameID

//loop through all minions

//check for collision

//if collided

//damage calculation (gets info from tower that owns projectile)

//despawn projectile

//if projectile reaches max range/timer

//despawn projectile

}

//IS\_UPDATE\_ECONOMY

void UpdateEconomy(uint gameID, uint[] economyValues)

{

//loop through all players in the gameID

//add MpS value to total mana

//sends out SC\_UPDATE\_ECONOMY to each client

}