Environment Model

1. Server Version

Input Message:

- DemandLobbyList()
- SendLobbyUpdate(modifiedLobby: Lobby)
- RemoveLobby(lobby: Lobby)
- DemandLobbyUI(lobbyToPresent: Lobby)
- SendVote(vote: Vote)
- SendMessage(message: String)
- SendAudio(voiceMessage: Audio)
- SendGameStatus(gameStatus: GameStatus)
- SendCommandToPlayer(command: Action, player: Player)
- SendEndTurn()
- DriveFireVehicle(position: Grid)
- RadioAmbulanceVehicle(position: Grid)
- SendIndividualVotes(Set{Vote})

Output Message:

- UpdateLobbyList(lobbyList: Set{Lobby})
- LobbyRemoved(Set{Lobby})Gue
- UpdateLobbyUI(lobbyToPresent: UI)
- VoteResult(result: GameStatus)
- DeliverMessage(message: String)
- DeliverAudio(voiceMessage: Audio)’
- DeliverGameState(gameStatus: GameStatus)
- DeliverCommandToPlayer(command: Action)
- WinLoseCheck()
- DemandVehicleVote()

2. Client Version

Input Message:

- BackToMainMenu()
- JoinGamesButton()
- JoinALobby(wishToJoin: Lobby)
- ChooseFirefighter(role: Role)
- GuestReady()
- GuestExit()
- LoadGamesButton()
- LoadAGame(game: Game)
- HostExit()
- StartGame()
- CreateGameButton()
- CreateAGame(gameSettings: GameSetting)
- EditSettings(newGameSettings: GameSetting)
- PlaceInitialFireFighterPosition(position: Grid)
- VoteInitialAmbulancePosition(position: Grid)

- VoteInitialFireEnginePosition(position: Grid)
- SendText(message: String)
- SendAudio(voiceMessage: Audio)
- Move(position: Grid)
- ChopWall(wall: Wall)
- InteractDoor(door: Door)
- ExtinguishFireSmoke(position: Grid, fully: boolean)
- Carry(carry: Carryable)
- Lead(victim: POI)
- Abandon(carrying: Carryable)
- FireDeckGun()
- ReRoll(changeX: boolean, changeY: boolean)
- DriveFireEngine(fireEngine: FireEngine, position: Grid)
- RadioAmbulance(ambulance: Ambulance, position: Grid)
- CrewChange(role: Role)
- Resuscitate_Paramedic(victim: POI)
- RemoveHazmat_Technician(hazmat: Hazmat)
- Command_Captain(player: Player, command: Action)
- RevealPOI_Imaging(poi: POI)
- EndCurrentTurn()

Output Message:

- MainMenu(mainMenu: UI)
- AvailableLobbies(Set{Lobby})
- GuestLobby(guestLobby: UI)
- PromptChosenFirefighter()
- PromptReadyStatus()
- PromptForVote()
- AvailableSavedGames(Set{SavedFile})
- HostLobby(hostLobby: UI)
- UpdateGameBoard(board: GameBoard)
- ErrorMessage(message: String)
- PromptSettings()
- ChatLog(chatLog: String)
- PlayAudio(voiceMessage: Audio)
- YourTurn()
- PromptReRollCoordinate(position: Grid)
- PromptIfWantToRideVehicle()
- PromptChooseRole(availableRoles: Set{Role})
- PromptInvalidAction(message: String)
- Command(command: Action)
- YouWinOrLose()

Operation Model

Operation: FlashPoint_Server:DemandLobbyList()
Scope: Set{Lobby}
Message: FlashPoint_Client::{UpdateLobbyList}
New: lobbyList: Set{Lobby}
Description: The behavior of *AquireLobbyList* is to get all current existing lobbies and return them as a set to *FlashPoint_Client*.

Operation: FlashPoint_Server::SendLobbyUpdate(lobby: Lobby)
Scope: Lobby,Player,UI
Message: FlashPoint_Client::{UpdateLobbyUI}
New: modifiedLobby：UI
Description: The behavior of *SendLobbyUpdate* is to get the current lobby ,update the lobby information and then return the modified lobby UI to client.

Operation: FlashPoint_Server::RemoveLobby(lobby: Lobby)
Scope: Lobby,Player
Message: FlashPoint_Client::{LobbyRemoved}
New:
Description: The behavior of *RemoveLobby* is to get the current lobby ,and remove the current lobby.When the host of the lobby exits the game, the lobby is not existed.So the current lobby needs to be removed.

Operation: FlashPoint_Server::DemandLobbyUI(lobby: Lobby)
Scope: Lobby, UI
Message: FlashPoint_Client::{UpdateLobbyUI}
New: lobbyToPresent: UI
Description: The behavior of *DemandLobbyUI* is to return the chosen lobby's UI to the *FlashPoint_Client*.

Operation: FlashPoint_Server::SendVote(vote: Vote)
Scope: Vote, GameStatus
Message: FlashPoint_Client::{VoteResult}
New: result: GameStatus
Description: The behavior of *SendVote* is to acquire all the Votes comes from *FlashPoint_Client*, these Votes are either for the position of vehicles or if they want to be a passenger when a vehicle is driven. Then find the most voted option and return the result as a *GameStatus* to all *FlashPoint_Client* in the current game.

Operation: FlashPoint_Server::SendMessage(message: String)
Scope: String
Message: FlashPoint_Client::{DeliverMessage}
New:
Description: The behavior of *SendMessag*e is to receive a String message then send it to all *FlashPoint_Client* in the current game.

Operation: FlashPoint_Server::SendAudio(voice: Audio)
Scope: Audio
Message: FlashPoint_Client::{DeliverAudio}
New:
Description: The behavior of *SendAudio* is to receive an Audio voice then send it to all *FlashPoint_Client* in the current game.

Operation: FlashPoint_Server::SendGameStatus(gameStatus: GameStatus)
Scope: GameStatus
Message: FlashPoint_Client::{DeliverGameStatus}
New:
Description: The behavior of *SendGameStatus* is to receive a GameStatus gameStatus, which is modified after the player took an action. And send it to all *FlashPoint_Client* in the current game so they would know what the player has done if it is a valid Action.

Operation: FlashPoint_Server::SendCommandToPlayer(command: Action, player: Player)
Scope: Action, Player
Message: FlashPoint_Client::{DeliverCommandToPlayer}
New: command: Action
fireCaptain: Role
Description: The behavior of *SendCommandToPlayer* is to send a message to the client that belongs to the one who is asked by the *FireCaptain* to ask the player if he/she would give his/her consent to do what the *FireCaptain* is asking for.

Operation: FlashPoint_Server::SendEndTurn()
Scope: GameStatus
Message: FlashPoint_Client::{DeliverGameStatus, YourTurn}
New: gameStatus: GameStatus
Description: The behavior of *SendEndTurn* is to end the current player's turn, prepare for the next turn (Advancing fire, replenishing POI) and notify next player that it is his/her turn through YourTurn.

Operation: FlashPoint_Server::DriveFireVehicle(position: Grid)
Scope: Grid, Server, Player
Message: FlashPoint_client::{UpdateClientStatus}
Description: The behavior of *DriveFireVehicle* is to send the *position* to the sever.

Operation: FlashPoint_Client::RadioAmbulanceVehicle(position: Grid)
Scope: Grid, Server, Player
Message: FlashPoint_client::{UpdateClientStatus}
Description: The behavior of *DriveAmulanceVehicle* is to send the *position* to the sever.

Operation: FlashPoint_Client::BackToMainMenu()
Scope: MainMenu, UI, Player
Message: Player:: {MainMenu}
New: MainMenu: UI
Description: The behavior of *BackToMainMenu* is to return to main menu and present the main menu UI to player.

Operation: FlashPoint_Client::JoinGamesButton()
Scope: UI, Game, Player
Message: FlashPoint_Server::{DemandLobbyList}:
New: Set{Lobby}
Description: The behavior of *JoinGameButton* is to join a game by pressing the joingame button then send a lobby list acquirement request to *FlashPoint_Server*

Operation: FlashPoint_Client::JoinALobby(wishToJoin: Lobby)
Scope: UI, Game, Player, set{Lobby}
Message: FlashPoint_Server:: {DemandLobbyUI}
New: Lobby: UI
Description: The behavior of *JoinALobby* is for player to choose one lobby from the lobby list, and the chosen lobby UI will be requested.

Operation: FlashPoint_Client::ChooseFirefighter(role: Role)
Scope: Game, Player,role
Message: FlashPoint_Server:: {UpdateLobbyStatus}
New: role: Role
Description: The behavior of *ChooseFirefighter* is for player to choose the firefighter role that he/she wants to play.

Operation: FlashPoint_Client::GuestReady()
Scope: UI,Player, ReadyStatus,Game
Message: Player:: {PromptReadyStatus}
FlashPoint_Server:: {UpdateLobbyStatus}
New:
Description: The behavior of GuestReady is that guest is ready for the game by pressing the ready button, and will prompt the ready status.

Operation: FlashPoint_Client::GuestExit()
Scope: UI, Player
Message: Player:: {AvailableLobbies}
FlashPoint_Server::{DemandLobbyList}:
New: Set{Lobby}
UI
Description: The behavior of *GuestExit* is that the player other than the host want to exit the current lobby, and return to a UI that contains a list of available lobbies

Operation: FlashPoint_Client::LoadGamesButton()
Scope: UI, Game, Player, GameStatus
Message: Player:: {AvailableSavedGames}
New: Set{SavedFile}
UI
Description: The behavior of *LoadGamesButton* is to load the saved game by pressing the button, and a list of saved games will be presented to the player

Operation: FlashPoint_ Client::LoadAGame(game: Game)
Scope: Game, Player, Lobby, UI, ReadyStatus, Role, GameSetting
Message: Player::{ HostLobby, ErrorMessage, PromptReadyStatus, PromptChooseRole }
FlashPoint_Server::{ SendLobbyUpdate, DemandLobbyUI }
New: lobby: Lobby
hostLobby: UI
error: String
role: Role
Description: The loadAGame operation loads the state of Game game from storage, and creates a lobby for the host. If creation fails, A ErrorMessage will be sent to Player. After creation of the Lobby, the saved GameStting and Role will be restored to that lobby. From its UI, Player can receive constantly updates for ReadyStatus and chosen roles from other players.
This operation also sends current lobby to server's lobby list, and constantly sends current lobby UI to the sever.

Operation: FlashPoint_ Client:: HostExit()
Scope: Player, UI
Message: Player::{ MainMenu }
FlashPoint_Server::{ RemoveLobby }
New: menu: UI
Description: The HostExit operation allows Host to exit current lobby, this operation sends RemoveLobby to server and removes current lobby from the list. At the same time, user returns to MainMenu UI.

Operation: FlashPoint_ Client:: StartGame()
Scope: Game, Player, Lobby, UI, ReadyStatus, Role, GameSetting, Grid
Message: Player::{ UpdateGameBoard, PromptReadyStatus, ErrorMessage }
FlashPoint_Server::{ RemoveLobby SendGameStatus }
New: error: String
board: GameBoard
gameStatus: GameStatus
Description: In order to perform this operation, Client has to check current ReadyStatus, if not everybody is a ready state, it will prompt an Error to Player. Otherwise, continues. The StartGame operation allows Host to start a game in Lobby, this operation creates/loads a GameBoard and GameStatus. GameBoard will be sent to Player, GameStatus will be sent to Sever. This operation also removes current lobby from the lobby list by sending RmoveLobby to server.

Operation: FlashPoint_ Client:: CreateGameButton ()
Scope: Player, UI, Lobby, GameSetting
Message: Player::{ PromptSettings }
New: gamesettings: GameSetting
Description: The CreateGameButton operation allows Player to set initial settings for Lobby.

Operation: FlashPoint_ Client:: CreateAGame(gameSettings: GameSetting)
Scope: Game, Player, Lobby, UI, ReadyStatus, Role, GameSetting
Message: Player::{ HostLobby, ErrorMessage, PromptReadyStatus, PromptChooseRole }
FlashPoint_Server::{ SendLobbyUpdate, DemandLobbyUI }
New: lobby: Lobby
hostLobby: UI
error: String
role: Role
Description: The CreateAGame operation allows Player to create a Lobby from initialized game settings. It creates a Lobby and an UI (including Role, ReadyStatus, GameSetting) for that Player. If creation fails, this operation then prompts an error message to User.
This operation sends current lobby to server's lobby list, and constantly sends current lobby UI to the sever.

Operation: FlashPoint_ Client:: EditSettings(newGameSettings: GameSetting)
Scope: Game, Player, Lobby, UI, GameSetting
Message: Player::{ HostLobby, PromptSettings, ErrorMessage }
FlashPoint_Server::{ SendLobbyUpdate, DemandLobbyUI }
New: settingsUI: UI
Description: The EditSettings operation allows Player to edit GameSetting in exist(current) Lobby. Host is able to change game settings in the prompted settingsUI, if change fails, an error message will prompt to Host. After editing, the operation will send current Lobby and LobbyUI to server for updates.

Operation: FlashPoint_ Client:: PlaceInitialFireFighterPosition(position: Grid)
Scope: Game, Player, Role, Grid, GameSetting
Message: Player::{ UpdateGameBoard, PromptInvalidAction}
FlashPoint_Server::{ SendGameStatus}
New: gameStatus: GAmeStatus
Description: The operation allows Players to place initial Firefighter positions. This operation receives the Grid Players want to place. Flashing_Client will decide if the chosen grid is valid after receiving the grid. If not, Client will prompt invalid Action. Otherwise, operation will then updates current GameBoard and send the updated GameStatus to server.

Operation: FlashPoint_Client::ExtinguishFireSmoke(position: Grid, fully: boolean)
Scope: Game, Player, FireFighter, Grid
Message: FlashPoint_Server::{SendGameStatus}
New:
Description: The behavior of *ExtinguishFireSmoke* is to receive a Grid on which the player wants to extinguish fire, and if the player would like to Fully extinguish it. After receiving the information then *FlashPoint_Client* will decrease the *Player's FireFighter's* AP by 1 if the player would like to extinguish a smoke or a fire to smoke, or by 2 if the player would like to fully eliminate a fire to nothing (these AP

decreasing are doubled for *FireFighter* of type *RescueSpecialist*). The fire status on the specified Grid is also changed to desired status. These changes are then applied to *GameStatus* and sent to *FlashPoint_Server* as well as displaying the changes to current *Player*. If the *Player* performs invalid extinguish fire, then *FlashPoint_Client* will tell the player about their invalid action.

Operation: FlashPoint_Client::Carry(carrying: Carryable)
Scope: Game, Player, FireFighter, Grid, POI, POIManager
Message: FlashPoint_Server::{SendGameStatus}
New:
Description: The behavior of *Carry* is to receive a carryable object of which the player would like to carry. The *FlashPoint_Client* will then change the RescueStatus of the POI to rescuing after validating the POI and the *Player's FireFighter's* carrying list of whether this *FireFighter* has already carried anything. This POI will then be added to the *Player's FireFighter's* carrying list. All these changes are then applied to *GameStatus* and are sent to *FlashPoint_Server* as well as displaying the changes to current *Player* using *POIManager.* If the *Player* performs invalid carry, then *FlashPoint_Client* will tell the player about their invalid action.

Operation: FlashPoint_Client::Lead(victim: POI)
Scope: Game, Player, FireFighter, Grid, POI, POIManager
Message: FlashPoint_Server::{SendGameStatus}
New:
Description: The behavior of *Lead* is to receive a *POI* object of which the player would like to lead. The *FlashPoint_Client* will then change the RescueStatus of the *POI* to rescuing after validating the POI and the *Player's FireFighter's* carrying list of whether this *FireFighter* is leading another victim currently. This POI will then be added to the *Player's FireFighter's* carrying list. All these changes are then applied to *GameStatus* and are sent to *FlashPoint_Server* as well as displaying the changes to current *Player* using *POIManager.* If the *Player* performs invalid carry, then *FlashPoint_Client* will tell the player about their invalid action.

Operation: FlashPoint_Client::Abandon(carrying: Carryable)
Scope: Game, Player, FireFighter, Grid, POI, POIManager, Hazmat
Message: FlashPoint_Server::{SendGameStatus}
New:
Description: The behavior of *Abandon* is to receive a *Carryable* object of which the player would like to abandon. The *FlashPoint_Client* will first remove the *Carryable* from the *FireFighter's* carrying list and then check if the *Carryable* is a *POI* or is a Hazmat. If the *Carryable* is a *POI* the RescueStatus of the POI is changed to WaitingForRescue and its state is also updated to the *POIManager*. All these changes are then applied to *GameStatus* and are sent to *FlashPoint_Server* as well as displaying the changes to current *Player.* If the *Player* performs invalid abandon, then *FlashPoint_Client* will tell the player about their invalid action.

Operation: FlashPoint_Client::FireDeckGun()
Scope: Game, Player, FireFighter, Grid, FireEngine
Message: Player::{PromptReRollCoordinate (only for player whose current FireFighter is the Driver/Operator), PromptInvalidAction}
FlashPoint_Server::{SendGameStatus}
New:

Description: This operation is either triggered by a *FireFighter* that is not a *Driver/Operator* or it is triggered by a *Driver/Operator.* The behavior of *FireDeckGun* is to generate a random *Grid* on the game board to extinguish fire near it. First *FlashPoint_Client* will check if the *Player's FireFighter's* position is the same as the fire engine, and it checks if there are no *FireFighter* in the target quadrant, it will then check if the *FireFighter* has enough AP to operate the deck gun. If the *FireFighter* is feasible for doing the action, a random grid is selected for the deck gun to fire at.

- If this is triggered by a *FireFighter* that's not a *Driver/Operator,* the fire on the randomly selected *Grid* and *Grids* adjacent to it are extinguished to smoke, if on the *Grids* are smoke, it is distinguished to nothing. 4AP is taken from the *FireFighter* and the updated game status is shown to *Player* also send to *FlashPoint_Server.*
- If a *Driver/Operator* type of *FireFighter* triggered this operation, 2AP is taken away and the *Player* is prompted the randomly generated *Grid* position and a choice on whether the *Player* want to *ReRoll* any/both of the coordinates to generate a new *Grid* coordinate to fire the deck gun, in this case, no message are sent to *FlashPoint_Server.*

Operation: FlashPoint_Client::ReRoll(changeX: boolean, changeY: boolean, coordinate: pair<int, int>)
Scope: Game, Player, Grid
Message: FlashPoint_Server::{SendGameStatus}
New:
Description: The behavior of *ReRoll* is to receive the *Player's* choice of rechoose a new X or Y coordinate or both or neither. The *FlashPoint_Client* will then generate a new coordinate based on the *Player's* choice and on the newly generated *Grid* position and *Grids* adjacent to it, fire are extinguished to smoke, smoke are extinguished to nothing. All these changes are then applied to *GameStatus* and are sent to *FlashPoint_Server* as well as displaying the changes to current *Player.*

Operation: FlashPoint_Client::DriveFireEngine(fireEngine: FireEngine, position: Grid)
Scope: Game, Player, Grid, FireEngine, Vote
Message: FlashPoint_Server::{SendVote, SendGameStatus}
New: Vote
Description: The behavior of *DriveFireEngine* is to receive a *FireEngine* that the *Player* is willing to drive and a *Grid* as target position. *FlashPoint_Client* will first check if the *Player's FireFighter's* location is the same as the *FireEngine* and check if the *FireFighter's* AP is enough for driving the *FireEngine.* Also *FlashPoint_Client* will check the target position's validity (target position has to be another *Grid* of type *FireEngineParking*). If the player performs invalid *DriveFireEngine,* then *FlashPoint_Client* will tell the player about their invalid action. For valid condition, a *Vote* is initiated on riding the *FireEngine* along with the current *Player.* This *Vote* is then sent to server and server distributes the *Vote* to other *Players* of whose *FireFighter* is placed on the same *FireEngineParking* as the current *Player.* The *Players* voted to ride along are added to the *FireEngine's* passenger list and then the *FireEngine* is moved to the target position along with all passengers, and 2 or 4AP (for target *FireEngineParking Grid* on opposite side) are taken away from the *FireFighter*. These changes are then applied to the *GameStatus* and sent to *FlashPoint_Server* as well as displayed to current *Player.*

Operation: FlashPoint_Client::RadioAmbulance (ambulance: Ambulance, position: Grid)
Scope: Game, Player, Grid, Ambulance, Vote
Message: FlashPoint_Server::{SendVote, SendGameStatus}
New: Vote

Description: The behavior of *RadioAmbulance* is to receive a *Ambulance* that the *Player* is willing to radio and a *Grid* as target position. *FlashPoint_Client* will check if the *FireFighter's* AP is enough for driving the *Ambulance.* Also *FlashPoint_Client* will check the target position's validity (target position has to be another *Grid* of type *AmbulanceParking*). If the player performs invalid *RadioAmbulance,* then *FlashPoint_Client* will tell the player about their invalid action. For valid condition, a *Vote* is initiated on riding the *Ambulance* along with the current *Player.* This *Vote* is then sent to server and server distributes the *Vote* to other *Players* of whose *FireFighter* is placed on the same *AmbulanceParking* as the current *Player.* The *Players* voted to ride along are added to the *Ambulance's* passenger list and then the *Ambulance* is moved to the target position along with all passengers. These changes are then applied to the *GameStatus* and sent to *FlashPoint_Server* as well as displayed to current *Player.*

Operation: FlashPoint_Client::VoteInitialAmbulancePosition(position: Grid)
Scope: Game, Player, Grid, Vote, Ambulance
Message: Player::{PromptInvalidAction}
FlashPoint_Server::{SendVote}
New: vote: Vote
Description: The *VoteInitialAmbulancePosition* operation allows players to create a *Vote* of the position that they want the ambulance to initially be. There are four special spots that are designated to ambulance, and if the vote ended up in an invalid spot, then *FlashPoint_Client* will notify the players with *PromptInvalidAction*. If the vote is valid, the *Vote* is then sent to the server by *SendVote*. An ambulance is then placed at the most popular position.

Operation: FlashPoint_Client::VoteInitialFireEnginePosition(position: Grid)
Scope: Game, Player, Grid, Vote, FireEngine
Message: Player::{PromptInvalidAction, PromptForVote }
FlashPoint_Server::{SendVote}
New: vote: Vote
Description: The *VoteInitialFireEnginePosition* operation allows players to create a *Vote* of the position that they want the fireEngine to initially be. There are four special spots that are designated to fireEngine, and if the vote ended up in an invalid spot, then *FlashPoint_Client* will notify the players with *PromptInvalidAction*. If the vote is valid, the *Vote* is then sent to the server by *SendVote*. A fireEngine is then placed at the most popular position.

Operation: FlashPoint_Client::SendText(message: String)
Scope: Game, Player
Message: FlashPoint_Client::{DelieverMessage}
FlashPoint_Server::{SendMessage}
New: None
Description: The *SendText* operation allows players to communicate with each other. Players can choose to send some text to the server by *SendMessage*.

Operation: FlashPoint_Client::SendAudio(voiceMessage: Audio)
Scope: Game, Player, Voice
Message: FlashPoint_Server::{SendAudio}
New: voice: Voice
Description: The *SendAudio* operation allows players to communicate with each other by voice chat. Players can choose to use their microphones, and send the data to server by *SendAudio*.

Operation: FlashPoint_Client::Move(position: Grid)
Scope: Game, Player, Grid, FireFighter, POIManager, POI
Message: FlashPoint_Server::{SendGameStatus}
Player::{PromptInvalidAction}
Description: The *Move* operation allows players to move their firefighters around the board. *Player* chooses a position, and *FlashPoint_Client* then verifies if the location is reachable and calculate the cheapest cost to get there. *FlashPoint_Client* will again check if the *Player* has enough AP to perform the action. If the action is invalid at anypoint, the *Player* will be notified by *PromptInvalidAction*. After that, the *FireFighter* that the player owns will be moved to the position. If there happens to be an reveale POI, *POIManager* will reveal it. Finally, the game status is updated to the server end by *SendGameStatus*.

Operation: FlashPoint_Client::ChopWall(wall: Wall)
Scope: Game, Player, Grid, FireFighter, Wall
Message: FlashPoint_Server::{SendGameStatus}
Player::{PromptInvalidAction}
Description: The *ChopWall* operation allows players to deal damage to a *Wall* object. *Player* chooses a *Wall* object adjacent to his/her *FireFighter*. *FlashPoint_Client* then verifies if the chosen object is valid, and if the *Player* has enough AP.If the action is not valid, the *Player* gets warned by *PromptInvalidAction*. Then, some damage is dealt to the wall object. *FlashPoint_Client* checks the total damageTaken of the *Wall*, and determines whether it should be destroyed or not. Finally, *Player* sees the change by *UpdateGameboard*, and the server end is notified by *SendGameStatus*.

Operation: FlashPoint_Client::InteractDoor(door: Door)
Scope: Game, Player, Grid, FireFighter, Door
Message: FlashPoint_Server::{SendGameStatus}
Player::{PromptInvalidAction}
Description: The *InteractDoor* opeartion allows players to open/close a *Door* object. Player chooses a *Door* object adjacent to his/her *FireFighter*. *FlashPoint_Client* then verifies if the chosen object is valid, and if the *Player* has enough AP. If the action is not valid, the *Player* gets warned by *PromptInvalidAction.* Then, the *Door* object's status is flipped. Finally, *Player* sees the change by *UpdateGameBoard*, and the server end is notified by *SendGameStatus*.

Operation : FlashPoint_Client::RideVehicle(decision: Boolean)
Scope: GameStatus
Message: FlashPoint_Server::{SendGameStatus}
Description: When other firefighter invites firefighters to go to the vehicle, the effect of *RideVechicle* is that the firefighter decide whether to ride a vehicle or not and notify the server with *SendGameStatus* message.

Operation: FlashPoint_Client::CrewChange(role: Role)
Scope: Role, GameStatus
Message: FlashPoint_Server::{SendGameStatus}
Description: The effect of *CrewChange* is to change the role of crew, obtain the special abilities of that role and notify the server with *SendGameStatus* message.

Operation: FlashPoint_Client::Resuscitate_Paramedic(victim: POI)
Scope: GameStatus, POI
Message: FlashPoint_Server::{SendGameStatus}
Description: The effect of *Resuscitate_Paramedic* is that the victim can walk by himself without cost extra action point of firefighter and notify the server with *SendGameStatus* message.

Operation: FlashPoint_Client::RemoveHazmat_Technician(hazmat: Hazmat)
Scope: GameStatus, Hazmat
Message: FlashPoint_Server::{SendGameStatus}
Description: The effect of *RemoveHazmat_Technician*is is to remove a hazmat from the firefighter space and place it in a rescued spot and notify the server with the *SendGameStatus* message.

Operation: FlashPoint_Client:: Command_Captain(player: Player, command: Action)
Scope: Player, Action
Message: FlashPoint_Server::{SendGameStatus}
New:
Description: The effect of *Command_Captain*is to send a *command* and the *player* he want to command to sever.

Operation: FlashPoint_Client::RevealPOI_Imaging(poi: POI)
Scope: POI, GameStatus
Message: FlashPoint_Server::{SendGameStatus}
New:
Description: The effect of *RevealPOI_Imaging*is to flip a POI marker anywhere on the board and notify the server with the *SendGameStatus* message.

Operation: FlashPoint_Client::endCurrentTurn()
Scope:
Messages: FlashPoint_Server::{SendEndTurn}
Description: The player wants to end the current turn. The effects of the *endCurrentTurn* operation is to notify the server he want to end current turn with the *SendEndTurn*. Also, if the current firefighter's action points have more than the maximum capacity, if so, subtract it to the maximum capacity.

Operation: FlashPoint_Client::DeliverGameState(gameStatus: GameStatus)
Scope: Game, GameStatus, Player
Message: Player::{UpdateGameBoard}
Description: The *DeliverGameState* operation allows the client to present the updated *GameStatus* by using *UpdateGameBoard*.

Operation: FlashPoint_Client::DeliverCommandToPlayer(command: Action)
Scope: Game, Action, Player, FireFIghter
Message: FlashPoint_Player::{Command}
FlashPoint_Server::{SendGameStatus}
Description: The *DeliverCommandToPlayer* operation allows the client to notify the *Player* of the *Action* that s/he receives. If the *Player* accepts, the *FlashPoint_Client* checks if s/he has enough AP and if the action is valid. The *FireFighter* that the *Player* controls then performs the action. The updated gamestatus is sent to the server by *SendGameStatus*.

Operation: FlashPoint_Client::UpdateLobbyList(lobbyList: Set{Lobby})
Scope: Set{Lobby}
Message: Player::{AvailableLobbies}
New: lobbyList: Set{Lobby}
Description: The behavior of *UpdateLobbyList* is to receive a set of lobbes from *FlashPoint_Server* and demonstrate them to the *Player*.

Operation: FlashPoint_Client::LobbyRemoved(Set{Lobby})
Scope: Set{Lobby}
Message: Player::{AvailableLobbies}
New:
Description: The behavior of *LobbyRemoved* is to return the lobby set after the given lobby was removed to *FlashPoint_Client* to demonstrate them to all *Player* who is acquiring the lobby set.

Operation: FlashPoint_Client::UpdateLobbyUI(lobbyToPresent: UI)
Scope: Lobby, UI
Message: Player::{GuestLobby, HostLobby}
New:
Description: The behavior of *UpdateLobbyUI* is to return the UI according to input Lobby to all *FlashPoint_Client* to demonstrate it to all *Player*.

Operation: FlashPoint_Server::VoteResult(result: GameStatus)
Scope: Vote, GameStatus
Message: Player::{UpdateGameBoard}
New:
Description: The behavior of *VoteResult* is to acquire the *GameStatus* result that is generated by the server from each *Player's* vote an apply this change in *GameStatus* to *FlashPoint_Client* and this change is also showed to *Player* with *UpdateGameBoard.*

Operation: FlashPoint_Client::DeliverMessage(message: String)
Scope: String
Message: Player::{Chatlog}
New:
Description: DeliverMessage operation allows server to deliver Messages to Players through *FlashPoint_Client*.

Operation: FlashPoint_Client::DeliverAudio(voice: Audio)
Scope: Audio
Message: Player::{Chatlog}
New:
Description: DeliverAudio operation allows server to deliver Audios to Players through *FlashPoint_Client*.

Operation: FlashPoint_Client::WinLoseCheck()
Scope: GameStatus
Message: Player::{YouWinOrLose}
New: WinSlogan: String
LoseSlogan: String
Description: *FlashPoint_Server* keeps checking if a win/lose status is reached. If so, end current player's turn immediately. And notify all *FlashPoint_Client* to display the win/lose slogan through YouWinOrLose.

Operation: FlashPoint_Client::DemandVehicleVote()
Scope: Vote
Message: Player::{PromptForVote}
New:
Description: All *FlashPoint_Client* generate a Vote from *Players* to let them decide if they want to ride the car.