Flash Point
Operational Model

Alek Bedard, Nuri Amiraslan, Francis Piché
Teymur Azim-zada, Seng Chiat Haw, Abhijay Gupta

November 2018

1 Game Setup

Operation: Flashpoint::login(credentials: String)
Scope: Player, Database
New: newPlayer: Player
Messages: Player::{promptHostOrJoin}
Post: The login operation looks for an existing instance of the Player that called this operation in the Database with the provided credentials. If there is no player with these credentials in the database, a *newPlayer* instance is created. and saved to the Database. Otherwise, an instance of the Player is retrieved from the Database. The *newPlayer* is then prompted to choose whether they would like to Host or Join a Game.

Operation: FlashPoint::replyHostGame()
Scope: Player
Messages: Player::{promptNewOrLoadExistingGame}
Post: The Player status is set to Host, and they are sent the *promptNewOrLoadExistingGame* message.

Operation: Flashpoint::replyJoinGame(gameIdentifier: String)
Scope: Player, Game
Messages: Player::{promptSelectRoleType; joinFailure_e}
Post: The *gameIdentifier* is used to check if the requested Game has an association to a Player who's status is Host. If not, the Player sent a *joinFailure_e* message. Otherwise, if the *difficultyLevel* of the Game matching the *gameIdentifier* is set to Family, the current Player's RoleType is set to BasicFirefighter. Otherwise *promptSelectRoleType* message is sent to the Player, and the Player's status is set to *Waiting*. The *Player* is associated with the Game which has the matching *gameIdentifier*.

Operation: FlashPoint::replyLoadExistingGame()
Scope: Player, Database, Game
Messages: Player::{promptChooseSavedGame}
Pre: The Player status must be Host.
Post: The Player indicates that they would like to use an existing saved game from the Database. The System sends the *promptChooseSavedGame* message back to the Player.

Operation: FlashPoint::replyChooseSavedGame(gameId: String)
Scope: Player, Database, Game
New:game: Game
Message: Player::{promptSelectRoleType}
Post: The System loads *savedGame* from the Database.(ie: it loads all Player positions, Tiles, Victims, Walls, Engine and Ambulance, ChatHistory, Fire, Smoke, HazMat, POI's, etc.) It then associates the Player which called the operation with the Game instance. The Player is then sent a *promptSelectRoleType* message.

Operation: FlashPoint::replyCreateNewGame(numberOfPlayers: Integer)
Scope: Player, Game, Database
New: newGame : Game
Messages: Player::{promptSelectDifficultyLevel}
Post: A *newGame* is created, and the *numberOfPlayers* of *newGame* is set. *newGame* is added to the *Database*. The *Player* is sent the *promptSelectDifficultyLevel* message. The possible difficulty levels are: Family, Recruit, Veteran and Heroic.

Operation: FlashPoint::replyChooseDifficultyLevel(game: Game, difficulty: DifficultyLevel)
Scope: Player, Game
Message: Player::{promptChooseBoardType}
Post: The difficultyLevel of *game* is set to the given difficulty. (This may be any of : Family, Recruit, Veteran and Heroic). The Player is then sent the *promptChooseBoardType* message.

Operation: FlashPoint::replyChooseBoardType(boardType: BoardType, game: Game)
Scope: Player, Game, Board, Fire, Smoke, Tile, Quadrant, POI, Victim, HotSpot, HazMat
New: newBoard: Board fires: Bag{Fire}, smokes: Bag{Smoke}, pointsOfInterest: Bag{POI}, hazmats: Bag{HazMat}, ambulance: Ambulance, engine: Engine, hotSpots: Bag{HotSpot}.
Messages: Player::{promptSelectRoleType}
Pre: The *game* must exist and must have no Board associated. The *game* must have a *difficultyLevel* previously set.

Post: The system creates a *newBoard* of type *option* and associates it with *game*. Then all Fires, Smokes, POI, HotSpot, Hazmat are created and associated with the *newBoard* in randomly selected Tiles. The number of the above markers depends on the *difficultyLevel* of *game*. If the *difficultyLevel* of the game is set to Recruit, 3 initial explosions are resolved and 3 HazMats are placed randomly and then the current Player is then sent the *promptSelectRoleType* message.If the *difficultyLevel* of the game is set to Veteran, 3 initial explosions are resolved and 4 HazMats are placed randomly and then the current Player is then sent the *promptSelectRoleType* message. If the *difficultyLevel* of the game is set to Heroic, 4 initial explosions are resolved and 5 HazMats are placed randomly and then the current Player is then sent the *promptSelectRoleType* message. If the *difficultyLevel* of the game is set to Family, the RoleType for the current Player is set to BaseFirefighter.
Operation: FlashPoint::replySelectRoleType(role: RoleType)
Scope: Player, Game
Messages: Player::{promptChooseStartingLocation}
Pre: The *Player* object of the *Player* which called this operation does not have a *RoleType* associated to them.
Post: The role is associated to the *Player*, and the *Player* is sent the *promptChooseStartingLocation* message.

Operation: FlashPoint::replyChooseStartingLocation(tile: Tile)
Scope: Player, Board, Tile
Messages: Player::{promptChooseVehicleLocations}
Pre: The Player must have a RoleType associated to them. The selected Tile must be a *OutsideTileType*.
Post: The *Player* is associated to the given *tile*. The player is sent the *currentGameBoard* message. If the Player status is Host, they are sent the *promptChooseVehicleLocations* message.

Operation: FlashPoint::replyChooseVehicleLocations(a: AmbulanceParkingSpot, e: EngineParkingSpot)
Scope: Player, Board, AmbulanceParkingSpot, Ambulance, EngineParkingSpot, Engine
New: amb: Ambulance, eng: Engine
Messages: Player::{currentGameBoard}
Post: A new Ambulance instance *amb* is created and associated to the AmbulanceParkingSpot *a*. A new Engine instance *eng* is created and associated the the EngineParkingSpot *e*.
Operation: FlashPoint::readyToPlay()
Scope: Player, Game, Board
Messages: Player::{currentGameBoard, informTurn}
Pre: The Player which calls this operations status must be *Waiting*. The Player must have a Role and Tile associated to them. (The Tile is their starting location).
Post: The status of the *Player* which called this operation is set to *Ready*.

If all players associated to *Game* are *Ready*, then a random player is sent the *informTurn* message, and the *currentGameBoard* message is sent to all Players.

2 In-Game

Operation: FlashPoint::endTurn()
Scope: Player, Game, Board, Fire, Smoke, HazMat, HotSpot, POI Victim, FalseAlarm, Tile
Messages: Player::{informTurn, currentGameBoard, promptChooseRespawnSpot, gameLost, promptChooseHazMatResolve}
New: Fire, Smoke, HotSpot, FalseAlarm, Victim
Post: The effect of *endTurn* is to resolve the advance fire (see below), replenish POI's (see below) and to determine the next *Player*. The Player's ActionPoints total is set to a minimum of 4. Any unspent ActionPoints are saved up to a total ActionPoint limit of 8. SpecialActionPoints are set to the minimum for the SpecialistType of the Player if not playing Family mode. The current *Player* of the *Game* is changed to the next *Player*, and they are notified with the *informTurn* message. All Players are sent the *currentGameBoard*.

- **Advance Fire**:

A random Tile with type House is selected. This will be referred to as the TargetSpace. If the Tile does not contain Smoke or Fire, and is not adjacent to a fire, a new Smoke is created on the TargetSpace. If the TargetSpace is associated with a Smoke then the Smoke is disassociated from the TargetSpace and a new Fire is created and associated to the TargetSpace. If the TargetSpace contains a Fire, see **Explosion.**

- **Explosion** When the fire advances into a space that is already on Fire, an **Explosion** occurs. An explosion radiates in all four directions from the TargetSpace. The Fire is associated with each adjacent space of TargetSpace without Fire or if there is a Smoke associated with TargetSpace, it is disassociated and a Fire is associated with the adjacent space. The TargetSpace can be a OutsideTile. A damage marker is placed on the wall(s) associated with TargetSpace. A door associated with the TargetSpace is disassociated(the door is removed). If any adjacent space of TargetSpace has a Fire marker, see **Shockwave**.

  - **Shockwave**

  Shockwave continues to travel in its respective direction passing through all the tiles which has a Fire associated with them until it encounters one of the following:

1) If the tile has no Fire or Smoke associations, a Fire is associated to the tile, even if the tile is OutsideTile.
2) If the tile has a Smoke associated with it, smoke is disassociated with the Tile and Fire is associated with the tile.
3) If the Tile has a Wall associated with it, a damage marker is placed on that Wall.
4) If the Tile has an association with a Door and the Door has a status closed, the Door is disassociated from the Tile.
5) If the Tile has an association with a Door and the Door has a status opened, the Door is disassociated from the tile and Shockwave continues.

If during Advance Fire a Fire is created on a Tile which contains a Player and the Game *difficultyLevel* is set to something other than Family, that Player is dissociated from the Tile and associated with the AmbulanceSpot which contains the Ambulance. If a Fire is created on a Tile which contains a Player and the Game difficultyLevel is set to Family, that Player is sent the *promptChooseRespawnSpot* message. Other Players are sent the *waitForPlayersMessage*. If the Player that was knocked down takes too long to choose, they will be associated to the Tile associated with the Ambulance.
If the Game *difficultyLevel* is set to something other than Family, the following changes also occur: If after resolving all FlashOvers, a Tile containing a HazMat now contains Fire, an explosion occurs. (See above). If two or more HazMat's now contain fire, the Player is sent the *promptChooseHazMatResolve* message. Additionally, if the TargetSpace contains a HotSpot, a new HotSpot is placed on the TargetSpace, and a second TargetSpace is selected and the process is repeated.
If at the end of resolving all fire, a new Fire has been associated to a Tile which contains an association to a POI, the POI is revealed as a Victim or FalseAlarm. If the POI is revealed to be a Victim, the Victim is removed from the System state, and the *lostVictims* counter in the current Game is incremented. If the number of lost players is greater than 3, the *gameLost* message is sent to all Players, and the *promptReturnToLobby* message is sent.

Initial Explosion
The system determines the TargetSpace. It associates a Fire and HotSpot marker with the TargetSpace and resolves an explosion as described in endTurn() operation in the TargetSpace. If the TargetSpace already has a Fire marker associated with it, the system determines a new TargetSpace, associates Fire and HotSpot marker with the new TargetSpace and resolves an explosion as described in endTurn() operation in the new TargetSpace.

- **Replenish POI**

If there are less than 3 POI markers on the Board, a new POI is placed on a random Tile. If the current Game associated to the Player's *difficultyLevel* is set to anything other than Family, POI's cannot be associated to Tiles which do not contain Fire, Smoke, Player, or another POI. If this the selected Tile contains one of those Markers, then the POI is shifted to an adjacent tile until it is in a valid location. If the *difficultyLevel* is Family, and the selected Tile contains a Smoke or Fire, this Smoke or Fire is dissociated from the Tile (and thus removed from the System). Additionally, if the selected Tile contains an association to a Player, a FalseAlarm or Victim is created and associated to the Tile.

Operation: FlashPoint::replyChooseHazmatResolve(h: HazMat)
Scope: Player, HazMat
Messages: Player::{currentGameBoard, promptCchooseHazmatResolve}
Pre: There must be at least two HazMat on the board which need to be resolved.
Post: The explosion caused by the Fire placed on h is resolved as per the Advance Fire explanation in the endTurn() operation. The *currentGameBoard-Message* is sent to all Players. If there are more unresolved HazMat objects with Fire on them, send the current Player the *promptChooseHazmatResolve*message.
Operation: FlashPoint::chooseRespawnSpot(a: AmbulanceParkingSpot)
Scope: AmbulanceParkingSpot, Player, Game
Messages: Player::{currentGameBoard}
Pre: The *difficultyLevel* of the Game associated to the current Player must be set to Family.
Post: The Player is associated to the selected AmbulanceParkingSpot a. The System sends the *currentGameBoard* message to all Players.

Operation: FlashPoint::saveGame()
Scope: Player, Database, Game, Board, Vehicle, Tile, ParkingSpot, HazMat, Victim, POI, HotSpot, Fire, Smoke
Messages: Database::{gameState}
Pre: The calling Player must be the Host.
Post: The entire state of the Game associated to the calling Player, and all objects associated to it are sent to the Database.

Operation: FlashPoint::quitGame()
Scope: Player, Game, Board, Vehicle, ParkingSpot, Tile, HazMat, Victim, POI, HotSpot, Fire, Smoke
Post: If the Player's status is not Host, the system removes the Player instance representing the sender from the instance of the Game, if the player was carrying a victim or a HazMat, the association with the player will be removed and new association with the tile on which player has quit is created. If there are no more participants or the Player's status is Host, then the game instance and all it's associations are removed from the System state.

Operation: Flashpoint::movePlayer(tile:Tile)

Scope: Player,Game,Board,Tile
Messages: Player::{currentGameBoard}
Pre: *tile* is adjacent to the Tile associated the calling Player, and is not a Wall or ClosedDoor, and the Tile does not contain Fire. The Player calling the operation must have sufficient Action Points. The Player calling the operation must be the *currentPlayer* in the *Game*.
Post: The system removes the Player from the tile they are associated with and associates with the *tile*. The *currentGameBoard* message is sent to all *Players* in the *Game*.

Operation: Flashpoint::openDoor(door: Door)
Scope: Player, Game, Board, Door
Messages: Player::{currentGameBoard}
Pre: The Door chosen by the Player calling the operation is adjacent to the Tile associated with the current Player, and the Door is closed. The Player calling the operation must have sufficient Action Points. The Player calling the operation must be the *currentPlayer* in the *Game*.
Post: The system changes the state of the *door* to open. The *currentGameBoard* message is sent to all *Players* in the *Game*.

Operation: Flashpoint::closeDoor(door: Door)
Scope: Player, Game, Board, Door
Messages: Player::{currentGameBoard}
Pre: The Door chosen by the Player calling the operation is adjacent to the Tile associated with the current Player, and the Door is open. The Player calling the operation must have sufficient Action Points. The Player calling the operation must be the *currentPlayer* in the *Game*.
Post: The system changes the state of the door on *tile* to closed. The *currentGameBoard* message is sent to all *Players* in the *Game*.

Operation: Flashpoint::chopWall(wall: Wall)
Scope: Player, Game, Board, Tile
Messages: Flashpoint::{ currentGameBoard, gameLost}
Pre: The Wall chosen by the Player calling the operation is adjacent to the Tile associated with the current Player, and the Wall has 0 or 1 damage markers. The Player calling the operation must have sufficient Action Points. The Player calling the operation must be the *currentPlayer* in the *Game*.
Post: The System increases the damage marker of the wall on *tile*. If the *wall* has 2 damage markers, the system changes the state of the *wall* to Open, else the state is not changed. The Game damageMarkers counter is incremented. If the damageMarkers of and the game is above the maxDamage for the Game, the Game is lost, and the *gameLost* message is sent. Otherwise, the *currentGameBoard* message is sent to all *Players* in the *Game*.

Operation: Flashpoint::moveVictim(victim: Victim, tile: Tile)
Scope: Player, Game, Board, Victim, Tile

Messages: Player::currentGameBoard, gameWon, promptReturnToLobby
Pre: The Tile to which the Victim is associated must be the same as the Tile associated to the current Player. The Player calling the operation must be the *currentPlayer* in the *Game*. There must not be a Fire in the path between, or on, the Tile associated to the current Player and *tile*.
Post: If the status of *victim* is NotTreated, moving costs double AbilityPoints. System associates the *victim* on with the calling Player, and removes the association with *victim* and it's Tile. If *tile* is an OutsideTile, the *victimsSaved* of the Game which is associated with the Player is incremented if the *tile* is outside tile. If *victimsSaved* of the Game is above 7, *gameWon* message is sent to all players, and all Players are sent the *promptReturnToLobby* message. Otherwise, the *currentGameBoard* message is sent to all *Players* in the *Game*.

Operation: Flashpoint::carryHazMat(hazmat:HazMat)
Scope: Player,Game,Board,Tile,HazMat
Messages: Flashpoint::currentGameBoard,
Pre: The Tile to which the HazMat is associated is adjacent to the Tile associated to the current Player. The Player calling the operation must be the *currentPlayer* in the *Game*.
Post: The system associates the *hazmat* on with the calling Player, and removes the association with *hazmat* and it's Tile. The *currentGameBoard* message is sent to all *Players* in the *Game*.

Operation: Flashpoint::extinguishFireOrSmoke(tile: Tile)
Scope: Player,Game,Board,Tile,Fire,Smoke, MultiStepMove
Messages: Player::promptRemoveOrFlipFire, currentGameBoard
New: m: MultiStepMove
Pre: *tile* is adjacent to the Tile associated to the current Player. *tile* must have a Fire or Smoke associated to it.The Player calling the operation must be the *currentPlayer* in the *Game*.
Post: If the Tile is associated to a Fire, a new MultiStepMove instance is created and associated to the calling Player as well as *tile*, and the Player is sent the *promptRemoveOrFlipFire* message. Otherwise, the Smoke is removed from the System state, and the Player is sent the *currentGameBoard* message.

Operation: FlashPoint::replyRemoveOrFlipFire(remove: boolean, m: MultiStepMove)
Scope: Player, Game, Board, Tile, Fire, Smoke, MultiStepMove
Messages: Player::{currentGameBoard, insufficientActionPoints_e}
Post: If the Player does not have sufficient Action Points to complete their choice of *remove*, the *insufficientActionPoints_e* message is sent. Otherwise, if *remove* is true, the Fire associated to the Tile associated to m is removed from the System state. If it is false, that Fire is replaced by a Smoke. The Player is sent the *currentGameBoard* message.

Operation: Flashpoint::fireTheDeckGun()

Scope: Player,Game, Board, Tile, Fire, Smoke Quadrant, EngineSpot, MultiStepMove;
Messages: Player::{currentGameBoard, promptReroll}
New: m: MultiStepMove; **Pre:**The Player calling the action must be associated with the EngineSpot associated to the Engine. The Quadrant associated with the EngineSpot must contain no Players. The Player must be the *currentPlayer* in the *Game.* The Player must have sufficient Action Points to complete the action.
Post: The System determines the Tile the deck gun hits, removes all Fire or a Smoke markers associated with this Tile and adjacent Tiles if the Player's role is not Driver/Operator otherwise, a new MultiStepMove instance m is created and associated to the calling Player. m is also then associated to the chosen Tile and adjacent Tiles, (and thus any Fires/Smokes associated to those Tiles) these Tiles are dissociated from the Tiles in the current Game's gameBoard. The *currentGameBoard* message is sent to all *Players* in the *Game.* If the currentPlayer's role is the Driver/Operator, the Player is also sent the *promptReroll* message.

Operation: FlashPoint::replyReroll(reroll: boolean, m: MultiStepMove)
Scope: Player, Game, Board, Fire, Smoke, Quadrant, MultiStepMove
Messages: Player::{promptSecondReroll, currentGameBoard}
Post: If *reroll* is true, the System undoes the previous operation by adding back associations of the Tiles associated to the MultiStepMove m associated to the currentPlayer to the board associated to the current Game. These associations are removed from m. The System then determines the Tile the deck gun hits, adds an association of this Tile and adjacent Tiles to m, and sends the *currentGameBoard* message to all *Players.promptSecondReroll* is sent to the current Player. Otherwise, the Smoke and Fire instances associated to the Tile instances associated to m are removed from the System state. The Tile instances associated to m are added back to the current Game's *board.* The MultiStepMove m is removed from the System state.

Operation: FlashPoint::replySecondReroll(reroll: boolean, m: MultiStepMove)
Scope: Player, Game, Board, Fire, Smoke, Quadrant, MultiStepMove
Messages: Player::{ currentGameBoard}
Post: If *reroll* is true, the System undoes the previous operation by adding back associations of the Tiles associated to the MultiStepMove m associated to the currentPlayer to the Board associated to the current Game. The System then determines the Tile the deck gun hits, removes all Fire and Smoke associations from this Tile and adjacent Tiles from the System state. The MultisStepMove m is removed from the System state. The System sends the *currentGameBoard* message to all *Players.*

Operation: Flashpoint::resuscitateVictim(victim: Victim)
Scope: Player,Game, Board, Tile, Victim
Messages: Flashpoint::{currentGameBoard}

Pre: The *victim*'s status must not be Treated. The Player's role must be a ParamedicRoleType. The calling Player must be the currentPlayer in the Game.
Post: The system changes the status of the *victim* to Treated . The *currentGameBoard* message is sent to all *Players* in the *Game*.

Operation: Flashpoint::removeHazMat(hazmat: HazMat)
Scope: Player, Game, Board, Tile, HazMat,RescuedSpot
Messages: Flashpoint::{currentGameBoard}
Pre: The Tile associated to the *hazmat* must be adjacent to the Tile associated with the calling Player. The Player's role must be a HazMatTechnicianRoleType. The calling Player must be the currentPlayer in the Game.
Post: The system removes the association of the and the associated tile and associates *hazmat* with RescuedSpot. The *currentGameBoard* message is sent to all *Players* in the *Game*.

Operation: Flashpoint::flipPOI(poi: POI)
Scope: Player, Game, Board, Tile, POI
Messages: Flashpoint::{currentGameBoard}
New: v:Victim, f: FalseAlarm
Pre: The Tile associated to the calling Player must be adjacent to the Tile associated with the POI, unless the Players RoleType is the ImagingTechnicianRoleType. If so, the Player must have sufficient ActionPoints.
Post: The *poi* is removed from the System state and either a FalseAlarm or Victim is created and associated to the Tile. The *currentGameBoard* message is sent to all *Players* in the *Game*.

Operation: FlashPoint::initiateDriveVehicle(vehicle:Vehicle, newSpot:ParkingSpot)
Scope: Player, Game, Board, Vehicle, ParkingSpot, MultiStepMove
Messages: Player::{ currentGameBoard, promptRideVehicle, waitingForOtherPlayers}
New: m: MutliStepMove;
Pre: The calling Player must be associated with the Tile which is associated with *vehicle*'s Tile if the *vehicle* is of type Engine.
Post: The systems sends *promptRideVehicle* to all players associated with the ParkingSpot containing the *vehicle*. A new MultiStepMove instance is created and *newSpot* is associated it. It is then associated to the calling Player and all Players associated with the ParkingSpot associated to the *vehicle*. The System sends the *waitingForOtherPlayers* message to the calling Player.
Operation: FlashPoint::replyRideVehicle(accept: Boolean, m : MultiStepMove)
Scope: Player, Game, Board, Vehicle, MultiStepMove
Messages: Player::{currentGameBoard, timeOut_e}
Post: If *accept* is true, the calling Player is associated with the Vehicle associated to their Tile, and their association with this Tile is removed. The system associates this Vehicle with *newSpot* from m and then removes the associations

of all players with the Vehicle. The MultiStepMove m is removed from the System state. The *currentGameBoard* message is sent to all Players.

Operation: Flashpoint::crewChange(role:RoleType)
Scope: Player, Game
Messages: Player::{ currentGameBoard,roleIsTaken_e}
Post: If another Player associated with the Game currently has the *role* , then *roleIsTaken_e* is send to the player. Otherwise, the system changes the Role of the Player to *role*. The *currentGameBoard* message is sent to all Players.

Operation: Flashpoint::initiateCommandOtherPlayer(otherPlayer: Player, command: CommandType)
Scope: Player,Game, MultiStepMove
New: m: MutliStepMove
Messages: Player::promptCommandOtherPlayerRequest, waitingForOtherPlayers, commandRequestDeclined
Pre: *currentPlayer*'s role must be set to FireCaptain, and must have enough ActionPoints to complete the action. If *command* is MoveCommand, for the precondition of the *movePlayer* operation for *otherPlayer* must be valid. If *command* is OpenDoorCommand the precondition of the *openDoor* operation for *otherPlayer* must be valid. If *command* is CloseDoorCommand the precondition of the *closeDoor* operation for *otherPlayer* must be valid.
Post: A new MultiStepMove instance m is created and associated to the current Player. The CommandType of the MultiStepMove is set to *command*. The System sends a *commandOtherPlayerRequest* to *otherPlayer*, and *waitingForOtherPlayers* message is sent to the current Player. IF the other Player takes too long to respond, the request is assumed to have been denied, and the Player is sent the *commandRequestDeclined* message.

Operation: FlashPoint::replyToCommandRequest(accepted: Boolean, m: MultiStepMove)
Scope: Player, Game, Board, MultiStepMove
Messages: Player::currentGameBoard; commandRequestDeclined
Post: If accepted is true, and the CommandType of m is MoveCommand, then the Player is moved to the location specified by the request. If accepted is true, and the CommandType of m is OpenDoorCommand, then the status of Door adjacent to the Player is set to Open. If accepted is true, and the CommandType of m is CloseDoorCommand, then the status of Door adjacent to the Player is set to Close. m is removed from the System state. The *currentGameBoard* is displayed to all Players. If accepted is false, send a *commandRequestDeclined* message to the player associated with m.

3 Communication

Operation: createChatMessage(text: String)
Scope: Player, ChatHistory, ChatMessage
Messages: Player::displayChatHistory
New: chatMessage: ChatMessage
Post: A *ChatMessage* instance *chatMessage* is created with the given *text* and the *chatMessage* is associated with *ChatHistory*. The *displayChatHistory* message is sent to all *Players*.