Flash Point: Fire Rescue

Operation Model

The following aims to be a complete operation model for the Flash Point: Fire Rescue game. Whenever an operation like *currentGameBoard* as one of the output messages, it means that all players are notified of the new game state once the effects of the operation have taken place. The in-game operations will be shown for when an experienced game mode is selected, as it encompasses the same operations and instance creations as the family mode and more.

Operation: FlashPoint::login(username: String, password: String)
Scope: Player;
Messages: Player::{verificationSuccessful(); failedtoLogin()}
Description: The *login* operation looks for an instance of the Player in the system with the provided username. If it exists, the Player is shown the main menu. If the credentials do not match those in the database, the player is taken back to the login menu.

Operation: FlashPoint::createNewUser (username : String, password : String, email : String)
Scope: Player;
New:
newPlayer: Player
Messages: Player::{newUserCreated()}
Description: The Player does not have an existing username, so the Player is creating a new account in order to play. Once they register, the Player is shown the main menu.

Operation: FlashPoint::cancelGameCreation()
Scope: Game
Message: Player::{gameCreationCancelledSuccessfully()
Description: A player can choose to cancel his game creation before launching the readyToPlay() operation while other players are joining his game.

Operation: FlashPoint::cancelJoinGame()
Scope: Game
Message: Player::{cancelJoinGameSuccessful()}
Description: While attempting to join a game or before sending the readyToPlay() operation, a player can choose to cancel

Operation: FlashPoint::createNewFamilyGame(numOfPlayersToStart : int, generateRandomBoard : boolean)
Scope: Game, Player, JoinGame, Board, Position, Wall, PointOfInterest, Firefighter, FireMarker, SmokeMarker, Door, DamageMarker;

New:

newGame: Game
newBoard: Board
newBlackDice: BlackDice
newRedDice: RedDice
newPositions: Set<Position>
newEdges: Set<Edges>
newPOIs: Set<PointOfInterest>
newFirefighters: Set<Firefighter>
newFireMarkers: Set<FireMarker>
newDoors: Set<Door>
newDamageMarkers: Set<DamageMarker>
newSmokeMarkers: Set<SmokeMarker>
newRedDie : RedDie
newBlackDie : BlackDie

Message:

Player::{currentGameBoard()}

Description:

The Player has successfully logged on and selects to create a new game. The game is created and then initializes all the objects necessary for the game to start. The game is set as *readyToJoin* and waits for the required number of players, *numOfPlayersToStart*, to connect before launching the game. One instance of *Firefighter* is now controlled by the player that sent the request. Once the required number of players joined, if the *numOfPlayersToStart* is less than 6, the instances of *Firefighter* not already attributed to a *player* are assigned to joined players on a round-robin basis (e.g. if numOfPlayersToStart is 4 and 4 players have joined, the sender will control *Firefighter* 1 and 5, *Player* 2 will control *Firefighter* 2 and 6, and the other two Players will control *Firefighter* 3 and 4 respectively.) Furthermore, if *generateRandomBoard* is *true*, the game will place the FireMarkers at random positions located within the borders of the house. It will also place the PointOfInterest markers at random positions that are not occupied by a FireMarker. If *generateRandomBoard* is set to *false* then both the *FireMarkers* and *PointOfInterests* will be placed at their default positions i.e. the same as positions as in the board game edition. Finally, when enough players have joined, the new game state is sent to the Player.

Operation: FlashPoint::createNewExperiencedGame(numOfPlayersToStart : int, level : String)
Scope: Game, Board, Position, Wall, HotSpot, PointOfInterest, FireFighter, Engine, FireMarker, SmokeMarker, Door, HotSpot, HazMat, Ambulance, HealMarker, DamageMarker;
New:

newGame: Game
newBoard: Board
newPositions: Set<Position>
newWalls: Set<Wall>

newHotSpotMarkers: Set<Hotspot>
newPOIs: Set<PointOfInterest>
newEngine: Engine
newFireMarkers: Set<FireMarker>
newSmokeMarkers: Set<SmokeMarker>
newDoors: Set<Door>
newHotSpotMarkers: Set<HotSpot>
newHazMatMarkers: Set<HazMat>
newAmbulance: Ambulance
newHealMarker: HealMarker
newDamageMarkers: Set<DamageMarker>
newRedDie: RedDie
newBlackDie: BlackDie
newImageTechnician: ImageTechnician
newParamedic : Paramedic
newHazmatTechnician : HazMatTechnician
newOperator : Operator
newFireCaptain: FireCaptain
newRescueSpecialist : RescueSpecialist
newCAFSFireFighter: CAFSFirefighter
newGeneralist: Generalist

Message: Player::{currentGameBoard()}
Description: This operation is similar to *createNewFamilyGame*, but this operation sets up an experienced game. Thus, a new game is created, its status is set to *readyToJoin* and in addition to setting up *FireMarkers* and *PointOfInterests*, it must also initialize *HotSpots, Engine, Ambulance,* and *HazMats.* Depending on the *level* provided, the amount of each instances created and their placement will vary. If *numOfPlayersToStart* is less than 6, the unassigned *Firefighter* instances are distributed on a round-robin basis, same as in *createNewFamilyGame*. Once enough players have reached, *readyToJoin* is set to false and the game state is sent to the player.

Operation: FlashPoint::loadExistingGame(g: Game)
Scope: Game, Board, Position, Wall, HotSpot, PointOfInterest, FireFighter, Engine, FireMarker, SmokeMarker, Door, HotSpot, HazMat, Ambulance, HealMarker, DamageMarker;
New:
newGame: Game
newBoard: Board
newPositions: Set<Position>
newWalls: Set<Wall>
newHotSpotMarkers: Set<Hotspot>
newPOIs: Set<PointOfInterest>

newFirefighters: Set<Firefighter>
newEngine: Engine
newFireMarkers: Set<FireMarker>
newSmokeMarkers: Set<SmokeMarker>
newDoors: Set<Door>
newHotSpotMarkers: Set<HotSpot>
newHazMatMarkers: Set<HazMat>
newAmbulance: Ambulance
newHealMarker: HealMarker
newDamageMarkers: Set<DamageMarker>
newRedDie: RedDie
newBlackDie: BlackDie
newImageTechnician: ImageTechnician
newParamedic : Paramedic
newHazmatTechnician : HazMatTechnician
newOperator : Operator
newFireCaptain: FireCaptain
newRescueSpecialist : RescueSpecialist
newCAFSFireFighter: CAFSFirefighter
newGeneralist: Generalist

Message: Player::{currentGameBoard}
Description: The *loadGame* operation loads the state of the game *g* from non-volatile storage (i.e. storage where the data stored is not deleted if the machine is shut off and is later retrievable). It sends a join status of the sender's player instance as *not ready*. The game state is sent to all players. If the game is a family game, then only a set of firefighters is initialized. If it is an experienced game, then all the specialists are initiatied.

Operation: FlashPoint::joinExistingGame(g : Game)
Scope: JoinGame, Players, Game;
Messages: Player::{currentGameBoard}
Description: The joinGame operation sends the current instance of the player to the list of instances existing in the *Game g*. The status of the sending player is set as *not ready.* The current game state is sent to all players.

Operation: FlashPoint::chatWithPlayer (p : Player, m : String)
Scope: Player
Messages: Player::{messageToPlayerSent, messageToPlayerNotSent}
Description: From the main menu, a Player can send a message to another player. He must select to whom he wishes to send the message and the include the message content. If he has a stable connection to the server, his message will be sent. Otherwise, he will get a be notified by the system that his message was not delivered.

Operation: FlashPoint::exitGame()
Scope: Game, Board, Position, Wall, HotSpot, PointOfInterest, FireFighter, Engine, FireMarker, SmokeMarker, Door, HotSpot, HazMat, Ambulance, HealMarker, DamageMarker;
Messages: Player::{handControl()}
Description: The exitGame() operation re-assigns the *Player* instance of the sending player controlling the least instances in the current game (e.g. if 4 people are playing and players 1 and 2 control 2 instances of FireFighter while players 3 and 4 control only 1, if player 4 quits, his Firefighter instance will be re-assigned to player 3.) If only 1 player controls all the Firefighter Instances and quits, then the game instance is deleted along with all the other game related instances, such as the positions, markers, vehicles, PointOfInterests, and walls. Once the sending player sends the exitGame() operation, the system sends the *handControl()* output to the Player who gets to control the sending Player's instance.

Operation: FlashPoint::readyToPlay()
Scope: Player, JoinGame;
Message: Player::{promptPickSpecialist, yourTurn}
Description: This operation sets the sending player's JoinGame to ready. Once all the *JoinGame* of each player is set to *ready*, and there is no selected current player (i.e. this is not a saved game), then the game state is set to *specialistSelection* the sending player is selected as the current player and is prompted to select his specialist from the available specialist cards. If there is a current player, then the game notifies him that it is his turn.

Operation: FlashPoint::specialistSelected(s : Specialist)
Scope: Player, Firefighter
Message: Player::{promptInitialPosition, promptPickSpecialist}
Description: the *specialistSelected* operation lets the Player select the specialist he wants to be. The order in which players get to choose the specialists is random. If the sender is not the last player to select his specialist, then the system sends a promptPickSpecialist message to the next randomly selected player. If he is the last player to choose, then the game state changes to *selectedInitialPosition* and a *promptInitialPosition* message is sent to Player 1, the first player to have chosen his specialist.

Operation: FlashPoint::help()
Scope: Player
Message: Player::{displayHelp()}
Description: At any moment during the game, a player can go to a help menu which will indicate the game rules. The system will display the help menu upon execution of this operation.

Operation: FlashPoint::endMyTurn()
Scope: Player, Firefighter

Message: Player::{storedAPLimitExceeded, yourTurn, promptPositionAfterKnockDown, promptMultiHazMatExplosionResolve, showWinningScreen, showLosingScreen}
Description: The *endMyTurn()* operation can have multiple effects, depending on what was achieved by the player during his turn. Here are the possible effects:

- If at the end of his turn, the number of *Victims* rescued exceeds 10, then the game state changes to *gameWon* and the system shows the winning screen, where statistics about the game are displayed and medals are handed out to Players.
- If the player decides to end his turn but the sum of his unused AP this turn and his stored AP exceeds the allowed limit, the system outputs a *storedAPLimitExceeded* message, and the *Player* is forced to make another move until he has spent enough APs to end his turn.

Once the player has successfully ended his turn, the game state changes to *advanceFire*, the system then rolls the dice the advance the fire, resolves explosions and flashovers. In addition, it places extra *PointsOfInterest* if there are less than the required amount present on the board at the end of the player's move. After the system has performed those actions,

- If the *Player* that is supposed to play next has been knocked down, the system will send a *promptPositionAfterKnockDown* output message to allow the player to select the *Position* at which he wants to continue the game and perform his turn. If any other player has been knocked down, the system will send him the same output message when his turn comes before he is allowed to perform another move.
- If the explosion have triggered a HazMat to explode a *promptMultiHazMatExplosionResolve* output message, where the sending player needs to choose which HazMat explosion he wants to resolve first.
- If the explosions have caused the amount of *DamageMarkers* to exceed the allowed limit, then the game state becomes *gameLost* and the system shows the losing screen to the players along with statistics on the performance of each player.
- If none of the above occurs, the next player receives a *yourTurn* output message.

Otherwise, the game state is changed to *advanceFireTurn* and the game randomly advances the fire, handles explosions and flashovers. Afterwards, it replenishes PointOfInterests, and changes the state back to *playersMove* and notifying the next player that it is now his turn.

Operation: FlashPoint::selectedHazMatResolve(h : Hazmat)
Scope: Player, Firefighter, HazMat
Message: Player::{promptMultiHazMatExplosionResolve, currentGameBoard}
Description: In the case of an explosion triggering several explosions caused by HazMat, the player sending the endMyTurn() input message that caused this explosion is prompted to select which HazMat caused explosion he wants to resolve first. If there are still more than 1 HazMat caused explosions after he has resolved it, the system sends the player another

proptMultiHaxMatExplosionResolve() message. Otherwise, the system resolves the subsequent explosions in the provided order and the players are shown the new game state.

Operation: FlashPoint::selectedInitialPosition(p: OutsidePosition)
Scope: Player, Firefighter, Position
Message: Player::{promptInitialPosition, yourTurn, currentGameBoard}
Description: The player selects the position in which he wants to start the game from the list of available positions (i.e. positions located outside of the house). His firefighter is then placed on the selected position. If the sender is not the last player to select his initial position, then a promptInitialPosition message is sent to the next player. If he is the last player to chose his starting position, then the game state changes to *playersMove* the system notifies Player 1 that it is his turn and the game state is shown to all players.

Operation: FlashPoint::selectedPositionAfterKnockDown(p : AmbulanceParkingPosition)
Scope: Player, Firefighter, Position
Message: Player::{currentGameBoard}
Description: If a player has been knocked down in between player moves, then he gets to select which ambulance spot he wants to continue the game from before being able to continue his turn. After he has select his position, game state is shown to all players.

Operation: FlashPoint::moveFirefighter(p : Position)
Scope: Player, Firefighter, Position
Message: Player::{currentGameBoard, insufficientAP}
Description: The moveFirefighter operation allows the player to move his firefighter to granted that it is not separated by a wall or door, and he has enough action points to do so. If he has enough action points, he may place his Firefighter on the specified location. If he does not, the system notifies him that he does not have enough action points.

Operation: FlashPoint::removeSmoke(s: SmokeMarker)
Scope: Player, SmokeMarker, Firefighter, Position
Message: Player::{currentGameBoard, insufficientAP}
Description: The removeSmoke operation allows a Player to remove a SmokeMarker from an adjacent or current position. If he has enough action points, then the selected SmokeMarker is removed, his action points are decremented, and the new game state is shown. Otherwise, the system notifies the player that he has insufficient AP.

Operation: FlashPoint::turnFireIntoSmoke(f: FireMarker)
Scope: Player, SmokeMarker, Firefighter, Position
Message: Player::{currentGameBoard, insufficientAP}
Description: The reduceFireToSmoke operation allows a Player to, using his FireFighter, reduce a FireMarker to a SmokeMarker, given that he has sufficient AP.

Operation: FlashPoint::removeFire(f: FireMarker)
Scope: Player, FireMarker, Firefighter, Position
Message: Player::{currentGameBoard, insufficientAP}
Description: The removeFire operation aims to combine the *turnFireIntoSmoke* and the *removeSmoke* operations into one operation. If the player has enough action points, he can completely remove a FireMarker in an adjacent or current position. If he does not have enough AP, then the system notifies him that he has insufficient AP.

Operation: FlashPoint::openDoor(d : Door)
Scope: Player, Firefighter, Door
Message: Player::{currentGameBoard, insufficientAP}
Description: This operation allows a Player to open a door that is located on one of the 4 edges of a Position. If the player does not have sufficient AP, then the the system notifies him that he cannot perform this action. Otherwise, the Door status is changed to *open*, his action points are decremented, and the new game state is sent to all the players.

Operation: FlashPoint::chopWall (w: Wall)
Scope: Player, Firefighter, Wall
Message: Player::{currentGameBoard, insufficientAP}
Description: A Player can place a damage marker on a wall located on one of the 4 edges of his *Position*. If he has sufficientAP, then at then of this operation, his APs are decremented, a damage marker is placed on that wall, and the total number of damage markers inflicted on the house is incremented by 1.

Operation : Flashpoint::carryVictim(v : Victim, p: Position)
Scope: Victim, Firefighter, Player,
Message: Player::{currentGameBoard, insufficientAP}
Description: When a player is located on the same position as a victim, he can choose to carry the victim as one of his moves. If he does not have sufficient AP, the system shows a insufficientAP message. Otherwise, the new game state is sent to all the players after the Firefighter has successfully carried the victim to the target position.

Operation : Flashpoint::carryHazmat(h : Hazmat, p : Position)
Scope: HazMatTechnician, Player, Hazmat
Message: Player::{currentGameBoard, insufficientAP, alreadyCarrying}
Description: A player located on a position can carry the HazMat the same way it can carry a victim. If the player does not have enough AP, the system gives him a insufficientAP output message. If he is already carrying a HazMat or a Victim, the system notifies him that he cannot carry more than he already is by sending the *alreadyCarrying* output message.

Operation: FlashPoint::crewChange(s : Specialist)

Scope: Player, Firefighter
Message: Player::{currentGameBoard, insufficientAP, notOnEnginePosition, notFirstActionOfTurn}
Description: The *crewChange* operation allows a specialist to select another specialist card among the unplayed specialists. If the user does not have sufficient AP, the system shows him an *insufficientAP* output message. If he has already made another move this turn, he cannot perform this operation and the system shows a *notFirstMove* message. If the *Firefighter* is not on the engine position when performing this move, the system displays a *notOnEnginePosition* message. Finally, if he fulfills all the above conditions, he can then become another specialist for the duration of this turn, and the *currentGameBoard* is shown to all the players.

Operation: treatVictim (v: Victim)
Scope: Player, Paramedic, Victim, HealMarker
Message: Player::{currentGameBoard, insufficientAP}
Description: If the player is operating as the Paramedic, then he can treat a victm, meaning he can assign the HealMarker to a Victim, which allows any Firefighter to carry that Victim at no additional Carry cost i.e. the cost of moving this victim is the same as only moving the Firefighter. If the player has sufficient AP, the HealMarker is placed on the Victim object and the new game state is shown to all players. Otherwise, a insufficientAP message is displayed.

Operation: disposeHazmat (h: Hazmat)
Scope: HazMatTechnician, Player, Hazmat
Message: Player::{currentGameBoard, insufficientAP}
Description: If a *Player* operates as the HazMat technician, he can remove the HazMat object in the same position as him. When he performs this operation, the system either alerts him that he does not have sufficient AP, or the game state changes and is shown to all players.

Operation: FlashPoint::commandFirefighter (f: Firefighter, m: CommandableMove)
Scope: Player, Firefighter, FireCaptain, BasicMove
Message: Player::{insufficientAP, promptPermissionToControl()}
Description: If a *Player* is a *FireCaptain*, he can operate another *Firefighter* by telling him to move and/or open doors on his turn if the other *Player* allows him to operate his *Firefighter* and he has sufficient APs to do it. If he has insufficient AP, the game shows that he has *insufficientAP*. If the other *Player* does not allow to be controlled, the system outputs a *permissionToControlDenied* message. Finally, after the target player has granted access, the move is performed and all the players are shown the new game state.

Operation: Flashpoint::repliedToPermissionToControl(answer : boolean)
Scope: Player, Firefighter
Message: Player::{currentGameBoard}

Description: A player, at any time that is not his turn, can be prompted to give control of his character to the Player that operates as the FireCaptain. If he accepts, the system will relay his message to the other player, and the new game state will be shown to all the players.

Operation: FlashPoint::flipPOI (poi: PointOfInterest)
Scope: Player, Firefigther
Message: Player::{currentGameBoard, insufficientAP}
Description: A player located on the same position as PointOfInterest object can chose to flip the *poi* to reveal if it's a *Victim* or a *FalseAlarm.* If he does not have sufficient AP, the system shows him the adequate message. Otherwise, all players are updated of the new game state.

Operation: FlashPoint::moveEngine (p: EngineParkingPosition)
Scope: Player, Firefighter, Engine, Position
Message: Player::{currentGameBoard, promptFirefighterToRideAlong, insufficientAP, notOnEnginePosition}
Description: A *Player* can select to move the *Engine* object if he is located on the same position as the Engine and has sufficient AP. If he does not have sufficient AP, the system shows him the *insufficientAP* message. If he is not located on the same position as the *Engine*, then the system shows the *notOnEnginePosition* message. If he chooses to move the *Engine* and another Firefighter is located on the same position, the system prompts that *Player* if he wants his *Firefighter* to travel along with the *Engine* with the sender. Otherwise, the *Engine* and the *Player* are moved to the target *Position* p and all players are notified of the new game state.

Operation: moveAmbulance (p: Position, rideAlong : boolean)
Scope: Player, Firefighter, Ambulance, Position
Message: Player::{currentGameBoard, promptFirefighterToRideAlong, insufficientAP}
Description: A *Player* can move the *Ambulance* without having to be located on the same *Position* as the *Ambulance*. He can also choose to ride along with the ambulance or not. If he does not have enough AP, the system displays a insufficientAP output message. If another player is located on the same *Position* as the *Ambulance*, he is prompted if he wants to ride along with the ambulance.

Operation: FlashPoint::invitationToRideAlongReturned(answer : boolean)
Scope: Player, Firefighter, Engine, Ambulance
Message: Player::{currentGameBoard()}
Description: If a *Player* is located on the same position as the *Engine* while another *Player* wants to perform the *moveEngine(p : Position)* or *moveAmbulance(p : Position)*, he is prompted if he wants his *Firefighter* to move along. If he accepts, his response is sent back and the initiator of either move operations and this *Player's Firefighter's* are transported to the target position. Otherwise, the initiator is transported but the prompted player remains at his current position. Either way, the resulting output is that the new game state is shown to all the players.

Operation: FlashPoint::showOptions()
Scope: Player
Message: Player::{displayedOptions()}
Description: At any moment, either at the menu screen or in game, a Player can select to show the options, such as turn sound on or off or adjust brightness. Once he selects this option, the system sends back a displayedOptions() output.

Operation: FlashPoint::saveGame()
Scope: Game, Player
Message: Player::{gameSavedSuccessfully(), failedToSaveGame()}
Description: At any point, any Player can select to save the current game state, which will be stored on the server and able to be reloaded by anyone else. If game was saved successfully, the server sends back a *gameSaved()* output message. Otherwise, it sends a *gameNotSaved()* output message.

Operation: FlashPoint::sendMessage(m : String)
Scope: Player
Message: Player::{messageSentSuccessfully(), messageNotSent()}
Description: At any point during the game, a player can send a message to all the other players in the game. If the message was sent successfully, the system sends a messageSentSuccessfully() output. Otherwise, it sends a messageNotSent() output message.

Operation : Flashpoint::fireDeckGun()
Scope: Player, Firefighter, BlackDie, RedDie
Message: Player::{promptDiceRollForDeckGun, insufficientAP, notOnEnginePosition, quadrantNotClear}
Description: The fireDeckGun() operation allows the *Player* to use his Firefighter to fire the deck gun on the engine vehicle to remove *SmokeMarkers* and *FireMarkers*. If a player attempts to perform this operation but is not located on the same tile as the Engine, then the system sends a *notOnEnginePosition* output message. If a player attempts this operation, but the quadrant in of the board covered by the *Engine* is not free of other *Firefighters*, then the system sends a *quadrantNotClear* output message. If the *Player* does not have sufficient AP to perform the operation, the *insufficientAP* message is sent to that player. Finally, if the *Player* matches all the pre-conditions to perform this operation, a *rollDiceForDeckgun* message is sent back to *Player*.

Operation : Flashpoint::diceRolledForDeckGun(r RedDie, b BlackDie, redResult : int, blackResult : int)
Scope: BlackDie, RedDie, Player, Firefighter, FireMarker, SmokeMarker, Position
Message: Player::{currentGameBoard, promptReRollForDeckGun}
Description: Once the *Player* has rolled the dice and the system, in case of landing outside of the quadrant, flipped the die to make them align with the adequate quadrant, the FireMarkers

and SmokeMarkers located in all adjacent positions to the target position are removed and all the players are made aware of the game state. However, if the *Player* performing this move is the *Operator* specialist, then the system will give him the option to re-roll each dice once if he is unsatisfied with the designated position.

Operation : Flashpoint::reRolledBoth(r:redDie, b:blackDie, redResult: int, blackResult: int)
Scope: Player, Position, FireMarker, SmokeMarker, RedDie, BlackDie, Engine
Message: Player::{currentGameBoard}
Description: After the player has decided to fire the deck gun, rolled the dice, and deciced to re-roll both dice, the resulting target *position* is final and all the *FireMarkers* and *SmokeMarkers* located in the target or adjacent positions are removed. All players are informed of the game state.

Operation : Flashpoint::reRolledRed(r: RedDie, redResult: int)
Scope: Player, Position, FireMarker, SmokeMarker, RedDie, BlackDie, Engine
Message: Player::{currentGameBoard, promptReRollBlack}
Description: After the player has decided to fire the deck gun and rolled the dice, he can then select this operation to re-roll only the red die. If he previously re-rolled the black die, then the target position is determined by the roll obtained from the red die on this roll and the roll of the black die on the previous roll. The new game state is shown to all players. Otherwise, he is prompted to re-roll the black die as well if he is unsatisfied with this result and hasn't done so already.

Operation : FlashPoint::reRolledBlack(b: BlackDie , blackResult: int)
Scope: Player, Position, FireMarker, SmokeMarker, RedDie, BlackDie, Engine
Message: Player::{currentGameBoard, promptReRollRed}
Description: After the player has decided to fire the deck gun and rolled the dice, he can then select this operation to re-roll only the black die. If he previously re-roll the red die, then the target position is determined by the roll obtained from the red die on this roll and the roll of the black die on the previous roll. The new game state is shown to all players. Otherwise, he is prompted to re-roll the black die as well if he is unsatisfied with this result and hasn't done so already.