COMP 361 Software Development

Legends of Andor

M3: Requirements Specification Models

Hexanome Group 14

Chelsea Wright, Ailish Mak, Anahita Mohapatra,
Brendon Keirle, Ian Tsai, Max Boksem, Tingyu Shen

Architectural Decisions

We are choosing to implement our game using a client/server model which would require two executables: one for the client and one for the server. At run time, we will have one server instance with many client instances. One major benefit of a client/server model for a multiplayer game, such as Legends of Andor, is that one player's internet connection will not affect another player's game which means there is increased reliability. Additionally, the centralization of control allows a dedicated server(s) to maintain the integrity of access, resources, and data important to protect the game and thus a layer of security is naturally included within the architecture that prevents cheating. Our architecture will be server authoritative.

Server Executable:

- Saved games (in progress) for each client will be stored on the server .
- Game lobby with clients user name will be saved and displayed by server.

Client Executable:

- Clients may use all functions available in game to *play Legend of Andor* with a personal account.
- Clients may use the *chat function* to communicate and make collaborative decisions together.

Environment Model

1. Input Messages

- **Pre-Game**
    - login(username : String, password : String)
    - logout()
    - createAccount(username : String, password : String, confirmPassword : String)
- **Game Lobby Processes: [Tere can be n clients connected to Game Lobby server]**
    - replyCreateNewGame()
    - replySetGame(gameName : String, public : boolean, difficulty: int, numPlayers: int)
    - replyChooseHero(h: Hero)
    - invitePlayer(self : Player, username : String)
    - replyInvitation(join: boolean)
    - replyJoinGame()
    - replyOpenSavedGame()
    - replyDistributeWineskinAndGold(amountOfGold : int, amountOfWineskin : int)
    - replyLoadSavedGame(filePath : String)
    - replyJoinGameSession(gameName : String)
    - leaveLobby()
    - replyPlayerOrder(gamePlayerList: list)
    - readyPlayer(player_ready: boolean)   //only for non-hosts
    - startGame()   //only for host
- **In Game Processes**
    - takeTurn()
    - openFeatures(Chat, Card: Legend/Event#, Hero: h)
    - completeDay()
- **Menu Processes**
    - openMenu()
    - quitGame()

- endGame()
- saveGame()

- **Move**
  - endTurn()
  - endDay()
  - movePrince(s: Space)
  - move(s: Space)
  - pass()
- **Free Actions**
  - activateFog()
  - emptyWell()
  - pickUpGD(gold: int, gemstone: int)
  - trade()
  - buyItem(Article : item)
  - useItem(Article: item)
- **Trade**
  - displayOwn(itemList: mylist, itemList: playerlist)
  - selectTradeItem(Article : item)
  - replyTrade(decision: boolean)
- **Fight**
  - joinFight(c:creature)
  - leaveFight()
  - inviteHero(h: Hero)
  - rollDice(numDice: int)
  - useItem(itemList: item)
  - distributeRewards(Gold: int, Willpower: int)
- **Event Card**
  - replyEventCard(EventCard #n, EventVote())
- **Communication**
  - sendChat(message : String, p : Player[])
- **Collaborative Action**
  - initiateVotingSession()
  - sendVoteDesion(self : Player, option : int)

- makeRequest(request : Request, p : Player[])

- **Game Conclusion Processes**
  - GameConclusion(Game)

2. Output Messages

- **Pre-Game Processes**
  - promptLoginResponse()
  - logoutSuccess()
  - createAccountResponse(msgInvalidCredentials : String)
- **Game Lobby Processes [There can be n clients connected to Game Lobby server]**
  - promptCreateNewGame()
  - prompSetGame()
  - promptInvitePlayer()
  - promptChooseHero(Player: Hero)
  - promptHeroTaken_e
  - promptInvitationResponse()
  - promptJoinGame()
  - promptJoinGameSession(gameName[] : String[])
  - msgWaitForPlayers
  - promptDistributeWineskinAndGold(amountOfGold : int, amountOfWineskin : int)
  - promptPlayerOrder
  - promptOpenSavedGame()
  - promptJoin()
  - promptSessionFull_e
  - msgPlayerLeft
  - promptSavedGames(gameName[] : String[])
  - promptReadyPlayer(Player[])
  - promptStartGame()
- **In Game Processes**
  - promptTakeTurn()

- promptOutOfHours_e()
- promptConfirmOvertimeHours()
- promptChatWindow()
- showCard(Card: Legend/Event)
- showHeroBoard(Hero: h)
- promptNewDay()

- **Menu**
  - promptSaveGame()
  - promptQuitGame()
  - promptEndGame()
  - promptloggout()
  - promptGameLobby()
- **Move**
  - promptNextTurn(playerOrder.next)
  - promptSunriseBox()
  - promptMove()
  - promptExceedDayHours_e()
- **Free Actions**
  - clearFog()
  - shieldIneffective_e()
  - promptUseItem(item: telescope)
  - wellEmptied()
  - passingThrough_e()
  - goldgemAdded()
  - itemAdded()
  - insufficientFunds_e()
  - cannotUseItem_e()
- **Trade**
  - promptTrade(Player: Hero)
  - promptItemOwn(Player: itemlist)
  - itemTradeInvalid_e()
  - sendItem(Player: Hero, itemlist: item)
  - promptTradeResponse(d: decision)

- **Fight**
    - promptRollDice()
    - msgWinLose
    - promptInviteFight(Player: Hero)
    - msgHeroLeft
    - promptInviteResponse(d: decision)
    - promptDiceValue()
    - noMoreRolls_e()
    - promptUseItem(Article[]: item)
    - chooseItem(itemlist: item)
    - distributionResponse(Gold: int, Willpower: int)
- **Event Cards**
    - promptTask(Card: event#)
    - promptEventCardResponse()
- **Communication**
    - displayChat()
- **Collaborative Actions**
    - promptSendVote(self : Player, option : int)
    - promptVotingResult(String)
    - promptDecisionToMake(d: Decisions)
    - promptFinalDecision(d: Decision)
- **Game Conclusion Processes**
    - promptGameWinOrLose()