Warszawa 01-2010

Universal Game System Documentation

Prepared by:

Małgorzata Janik
Krzysztof Bujniewicz
Bartosz Kubiński
Przemysław Kozioł

Universal Game System Documentation:

Section	Page
Glossary of important terms	[3]
Project description	[4]
Communication protocol design	[7]
Class diagrams	[8]
Use case diagrams	[13]
Event flow diagrams	[17]
State diagrams	[21]
Important activity diagrams	[24]

UGS Documentation: Glossary of important terms

Terms:

- Game Client – is any application connecting to the server via appropriate protocol and identifying itself with the game-id.
- Game Player – is a client application that hosts actual player and provides appropriate communication interface and message processing
- Game Master – is a client application that hosts actual game master and provides appropriate communication interface and message processing. Game master application must identify itself also by sending required number of players information.
- Game Server – mediates between Game Clients, keeps only the current pools of game players and game masters and tries to initiate game if any new client arrives. It is also responsible for running championship.
- End game message – regular end of game message containing winner id.
- Terminate game message – erroneous end of game message containing id of client that caused error and error type.
- Game state message – message contacting current game state – it is delivered to all players participating in game.
- Move request message – message directed to particular game player only informing him about his turn.
- Move message – response of the game player containing information about move.

UGS Documentation: Project description (1/4)- Overview

UGC overview:

- Universal Game System is designed to organize games between game applications.
- It consists of:
  - □ Game player – application hosting actual player programs, providing necessary communication interface for processing received move requests, sending responses and receiving game state and final result. The game player application is also responsible for sending game termination message if it detects the game master timeout;
  - □ Game master – application hosting actual game master program, providing necessary communication interface for sending move requests, receiving responses and sending game state and final result. The game master application is also responsible for sending game termination message if the actual game master detects the illegal movement or one of the players violates the timeout;
  - □ Game server - application responsible for providing communication between game players and game masters interfaces, delivering all messages sent in the system. Server is able to work in normal mode – when it supports all games independently or in an championship mode – when it supports only one type of the two-player game and only one game master.
- All host applications support the illegal movement processing and timeout detection. The system is designed to close the application responsible for error and finish game without recording any results. The other applications are placed back into the waiting pool and as soon as it would be possible they would start next game.
- Games are identified and matched by the server on the basis of unique game-id. Each client must identify itself by sending game-id and only then is granted client-id and can participate in games.

UGS Documentation: Project description (2/4) - Connctions

UGC – connection interfaces:

- There are no connections between game masters and players – all communication is managed by server.
- Server supports forwarding of:
  - Game state messages
  - Move request messages
  - Move messages
  - Game end messages
  - Game termination messages
- Game termination messages are handled in a different way depending on what is the reason of termination.
- All clients must register to the game server.
- All applications must handle the game termination messages.

UGS Documentation: Project description (3/4) - Server

Game server:

- Game server assigns id to any client that is trying to connect and provides the game-id.
- Game server tries to start game as any new client application arrives. If it fails it adds client to appropriate waiting pool.
- In an ordinary mode server is not recording any game results. In championship mode server records all game results, chooses the winner and sends championship results. As soon as the championship is over the server cleans all championship data.
- Game server is also responsible for organizing championship (if it is started in appropriate mode). In such case it accepts only the first game master supporting selected game and only players willing to play selected game can register to the server. The server can start championship only if required number of players is gathered. Then it prepares all championship stages and starts championship.

UGS Documentation: Project description (4/4) - Clients

Game master:

- The initial activity of game master application is sending client-id request containing supported game-id and required number of players information to the game server.
- Game master starts activity as soon as it receives complete list of players willing to participate in the game – then its application performs preparing initial state and chooses first player. Game master delivers initial state to players and it addressees appropriate player move request. As the move is delivered from player game master application verifies its correctness and game master can send updated game state and pick next player messages or can send terminate game message if it detects inappropriate move.
- Game master also supports timeout detection.
- Game master also is responsible for sending end game state with additional information about winner.

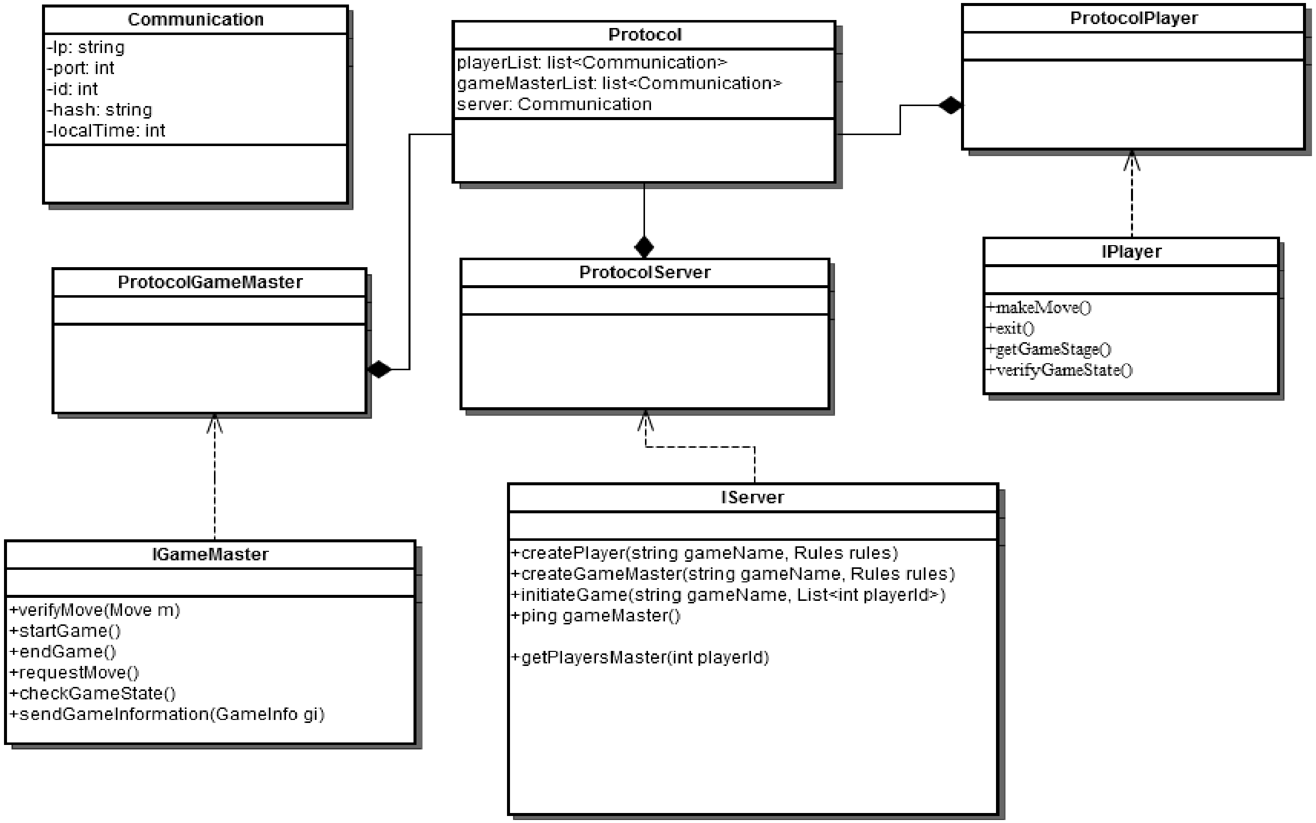
Game player:

- The initial activity of any game client application is sending client-id request containing supported game-id to the game server.
- Game client is responsible for receiving game state and move request messages and it responds with the new move message.
- Game client also supports timeout detection.

UGS Documentation: Communication protocol design

UGS Documentation: Class diagram (1/4):Overview

Protocol Class diagram



Communication

-Ip: string
-port: int
-id: int
-hash: string
-localTime: int

Protocol

playerList: list<Communication>
gameMasterList: list<Communication>
server: Communication

ProtocolPlayer

ProtocolGameMaster

ProtocolServer

IPlayer

+makeMove()
+exit()
+getGameStage()
+verifyGameState()

IGameMaster

+verifyMove(Move m)
+startGame()
+endGame()
+requestMove()
+checkGameState()
+sendGameInformation(GameInfo gi)

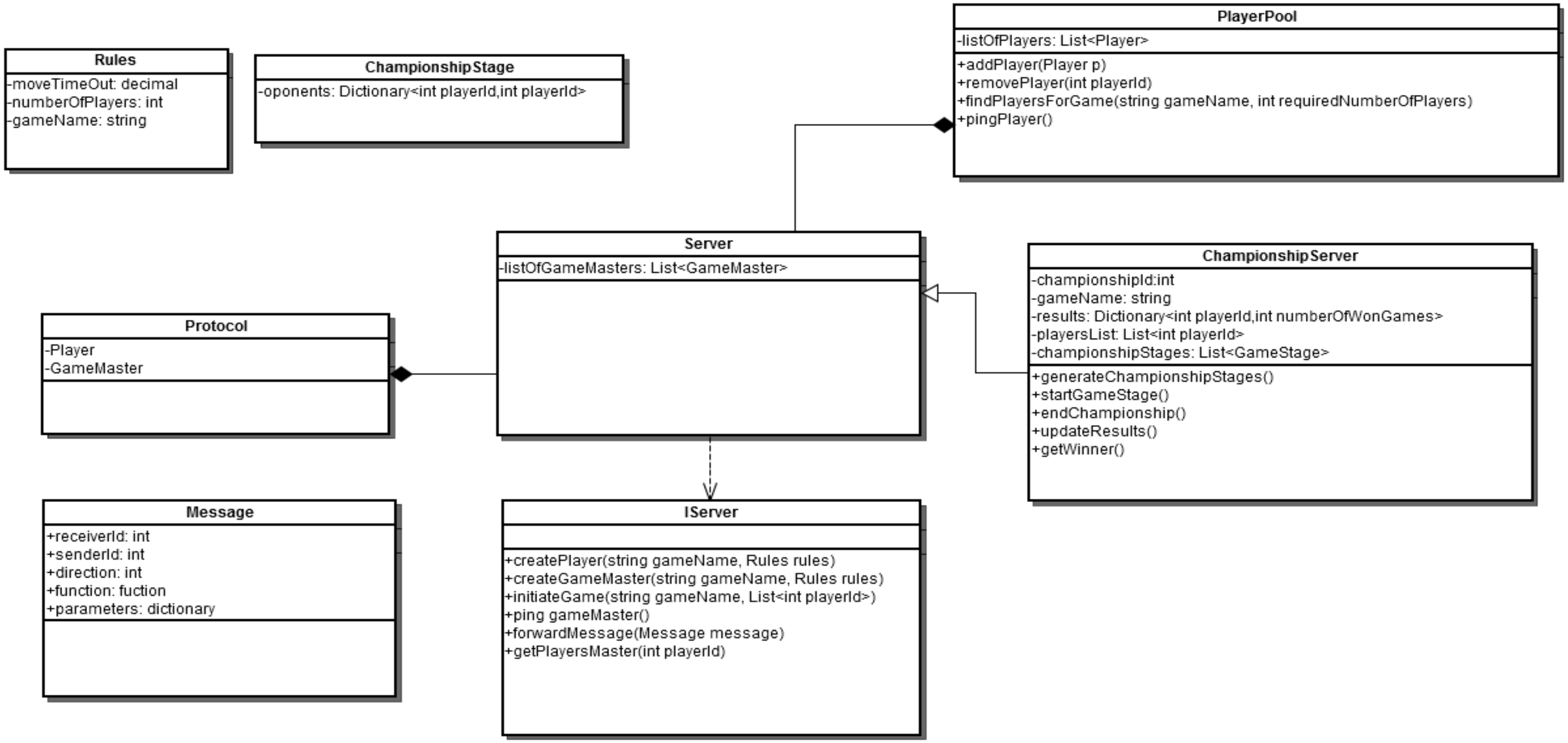
IServer

+createPlayer(string gameName, Rules rules)
+createGameMaster(string gameName, Rules rules)
+initiateGame(string gameName, List<int playerId>)
+ping gameMaster()

+getPlayersMaster(int playerId)

UGS Documentation: Class diagram (2/4):Game server

Server Class diagram



Rules
-moveTimeOut: decimal
-numberOfPlayers: int
-gameName: string

ChampionshipStage
-oponents: Dictionary<int playerId,int playerId>

PlayerPool
-listOfPlayers: List<Player>
+addPlayer(Player p)
+removePlayer(int playerId)
+findPlayersForGame(string gameName, int requiredNumberOfPlayers)
+pingPlayer()

Server
-listOfGameMasters: List<GameMaster>

Protocol
-Player
-GameMaster

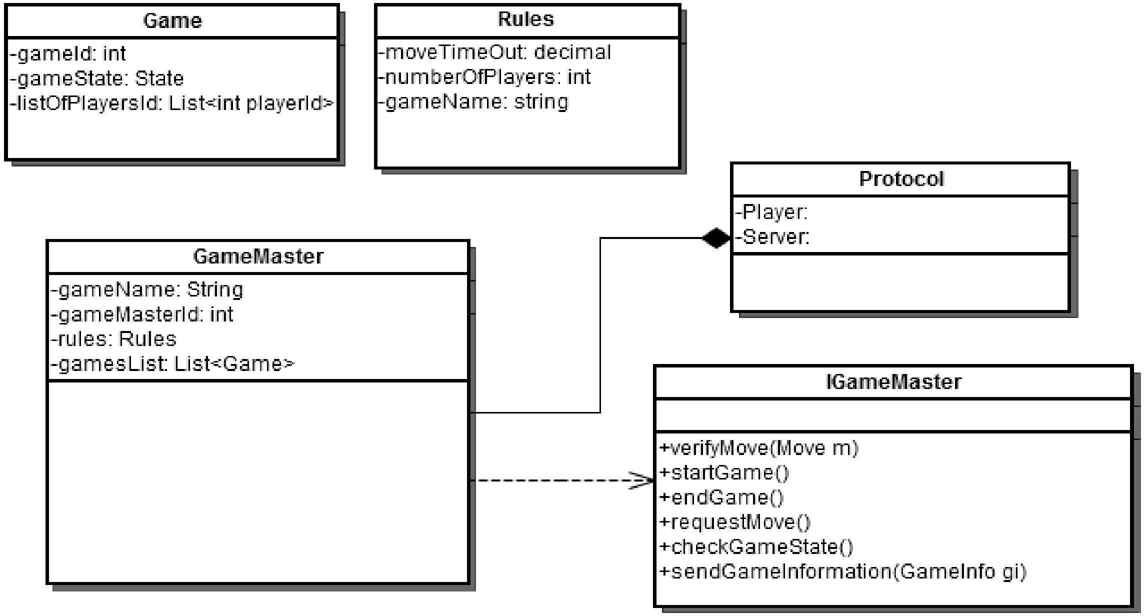
ChampionshipServer
-championshipId:int
-gameName: string
-results: Dictionary<int playerId,int numberOfWonGames>
-playersList: List<int playerId>
-championshipStages: List<GameStage>
+generateChampionshipStages()
+startGameStage()
+endChampionship()
+updateResults()
+getWinner()

Message
+receiverId: int
+senderId: int
+direction: int
+function: fuction
+parameters: dictionary

IServer
+createPlayer(string gameName, Rules rules)
+createGameMaster(string gameName, Rules rules)
+initiateGame(string gameName, List<int playerId>)
+ping gameMaster()
+forwardMessage(Message message)
+getPlayersMaster(int playerId)

UGS Documentation: Class diagram (3/4):Game master

Game Master Class diagram



Game
-gameId: int
-gameState: State
-listOfPlayersId: List<int playerId>

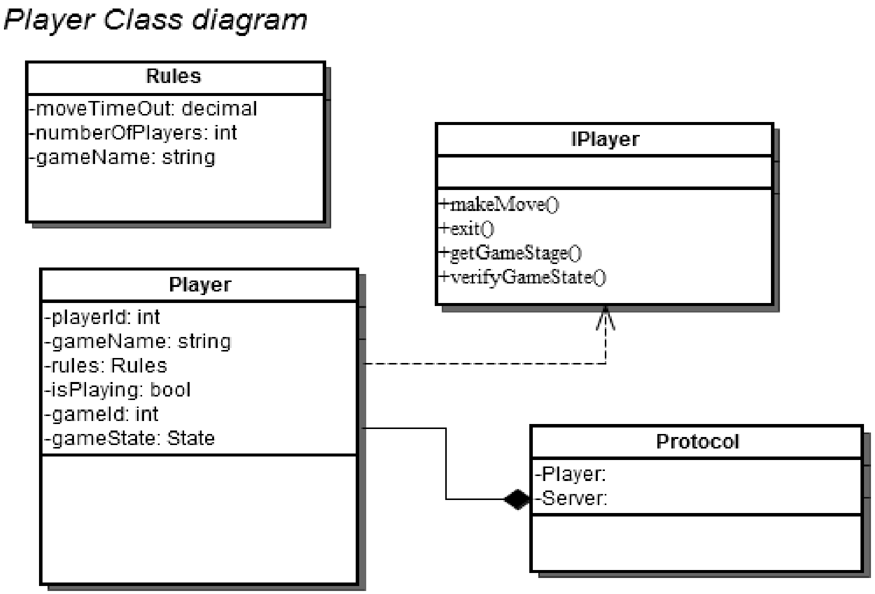
Rules
-moveTimeOut: decimal
-numberOfPlayers: int
-gameName: string

Protocol
-Player:
-Server:

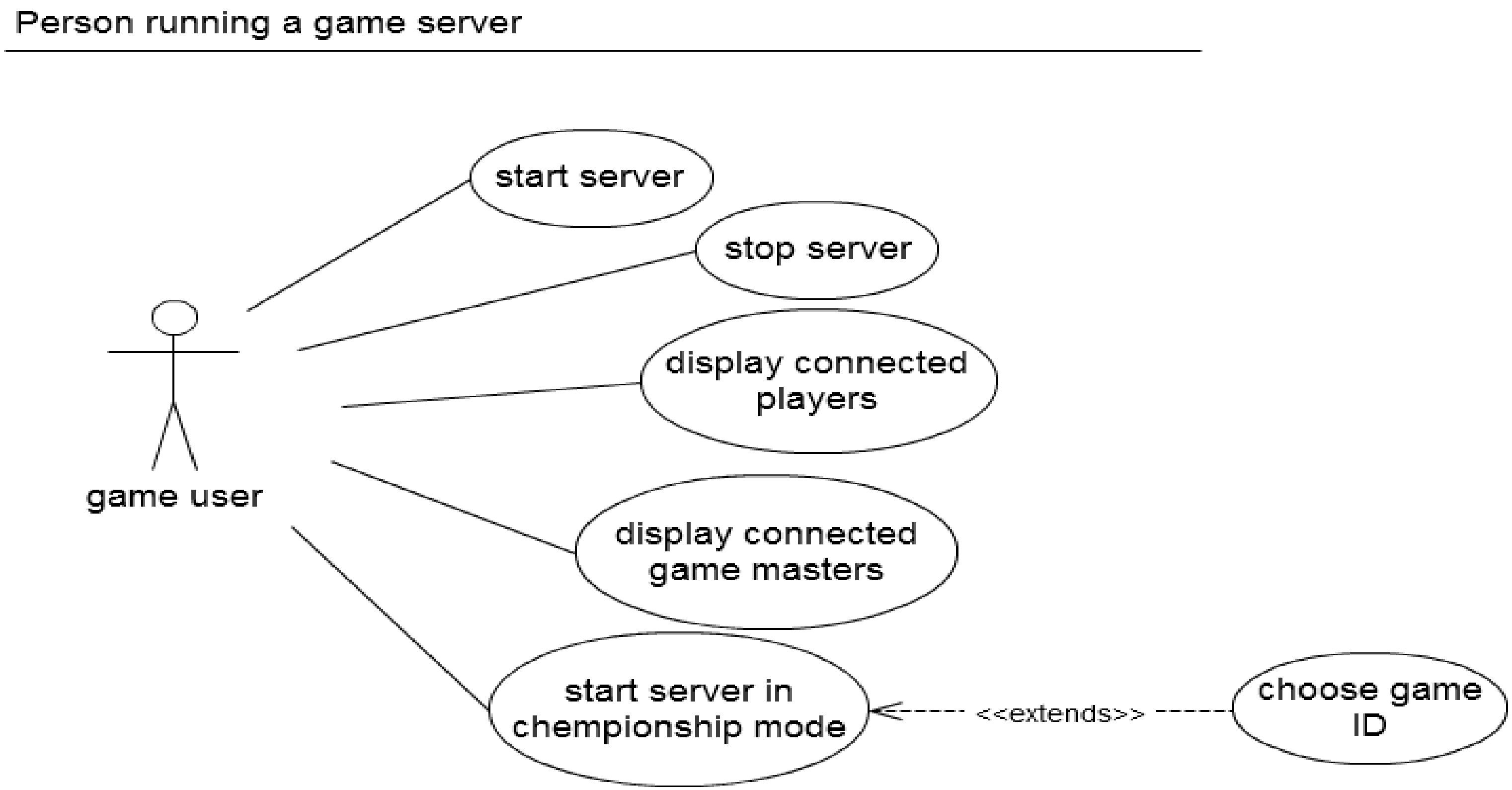
GameMaster
-gameName: String
-gameMasterId: int
-rules: Rules
-gamesList: List<Game>

IGameMaster
+verifyMove(Move m)
+startGame()
+endGame()
+requestMove()
+checkGameState()
+sendGameInformation(GameInfo gi)

UGS Documentation: Class diagram (4/4):Game player

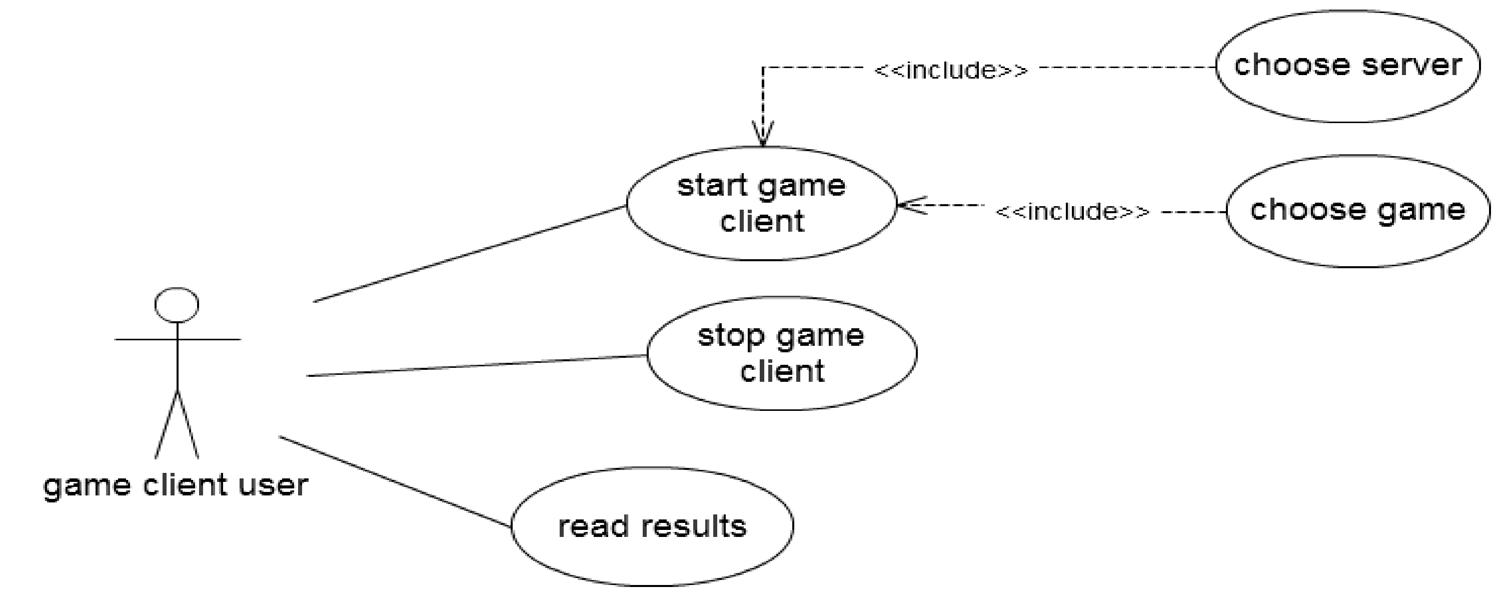


UGS Documentation: Use case diagrams (1/4) - Game server

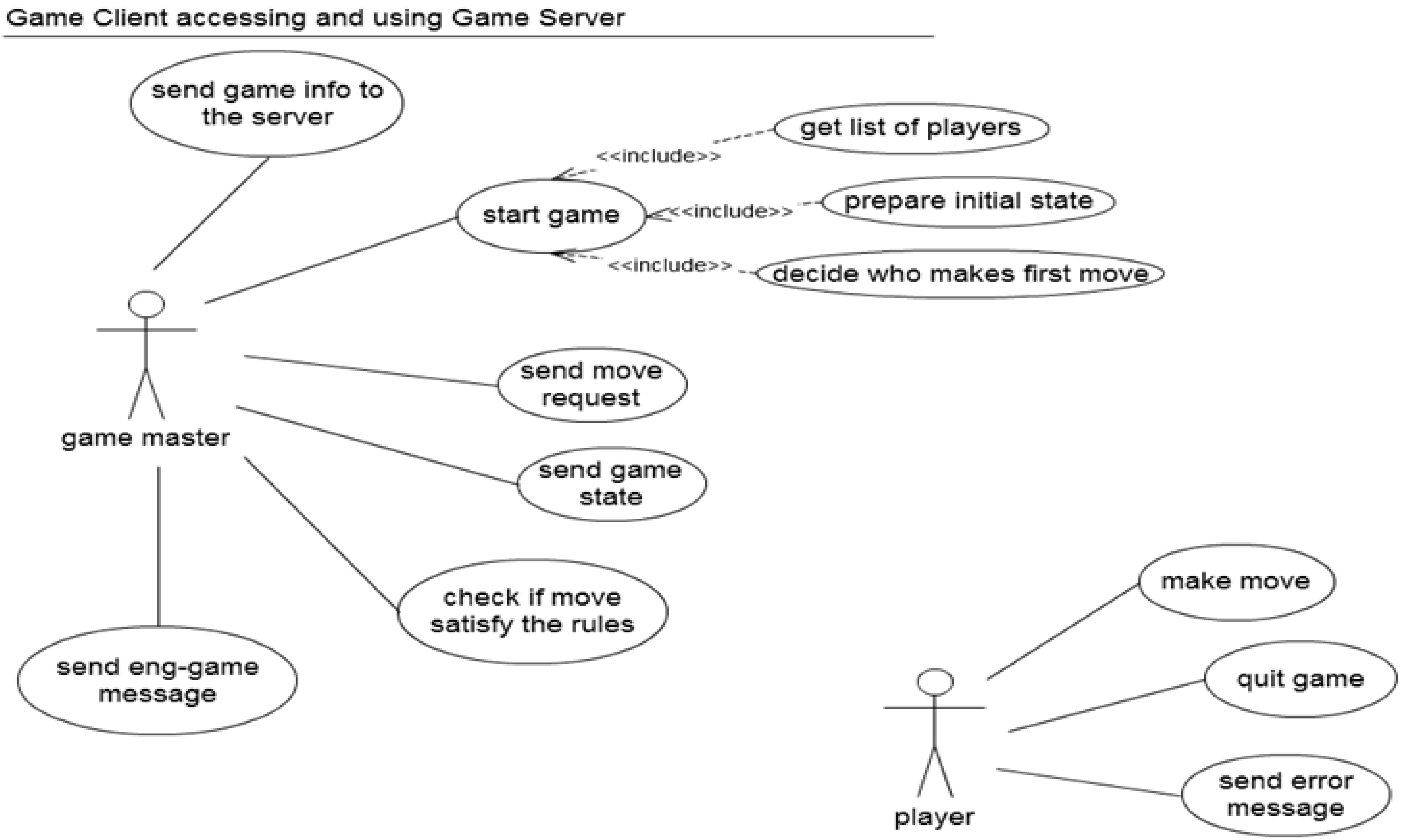

Person running a game server
game user
start server
stop server
display connected players
display connected game masters
start server in chempionship mode
<<extends>>
choose game ID

UGS Documentation: Use case diagrams (2/4) - Game client

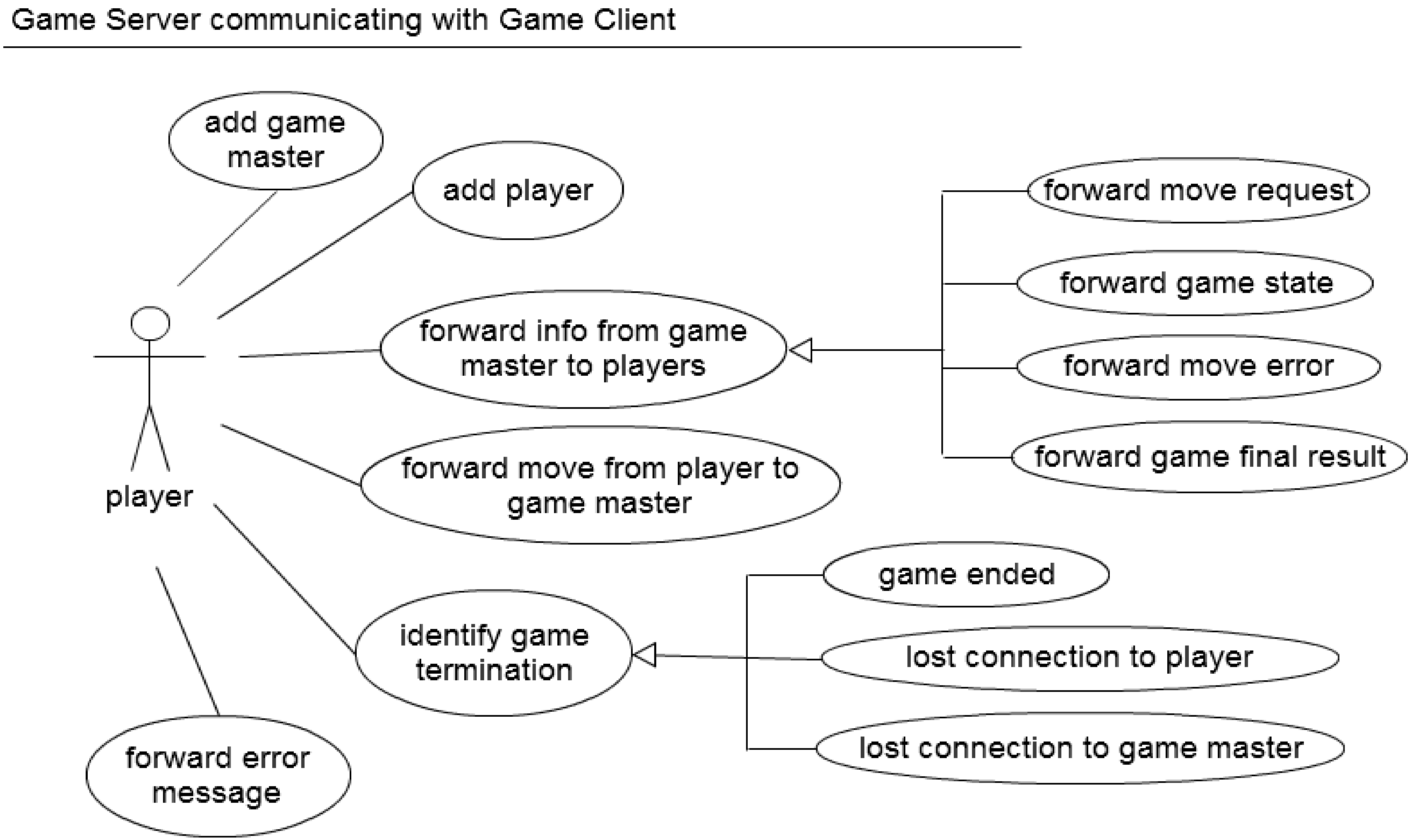
Person running a game client



UGS Documentation: Use case diagrams (3/4) – Server clients

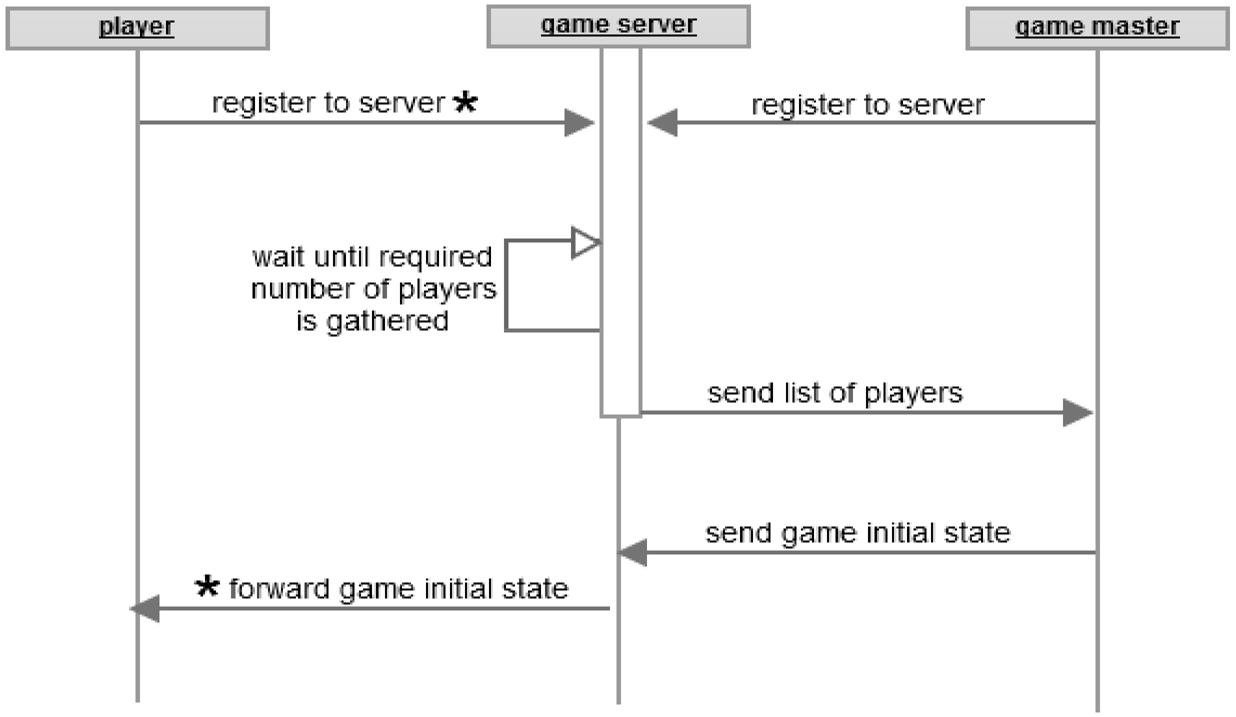


UGS Documentation: Use case diagrams (4/4) - Game server


Game Server communicating with Game Client
add game master
add player
forward info from game master to players
forward move from player to game master
identify game termination
forward error message
player
forward move request
forward game state
forward move error
forward game final result
game ended
lost connection to player
lost connection to game master

UGS Documentation: Event flow diagram (1/4) – Game start

Event Flow Diagram: Game Initialization



player

game server

game master

register to server *

register to server

wait until required number of players is gathered

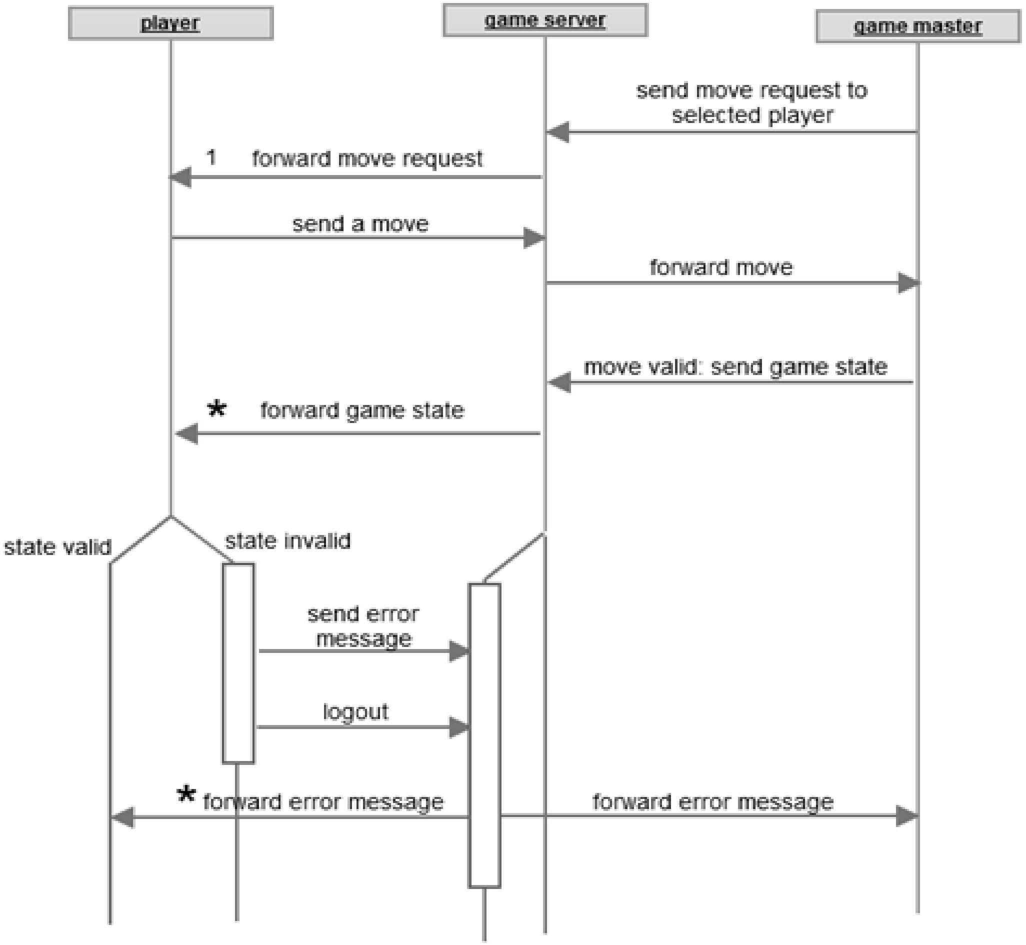
send list of players

send game initial state

* forward game initial state

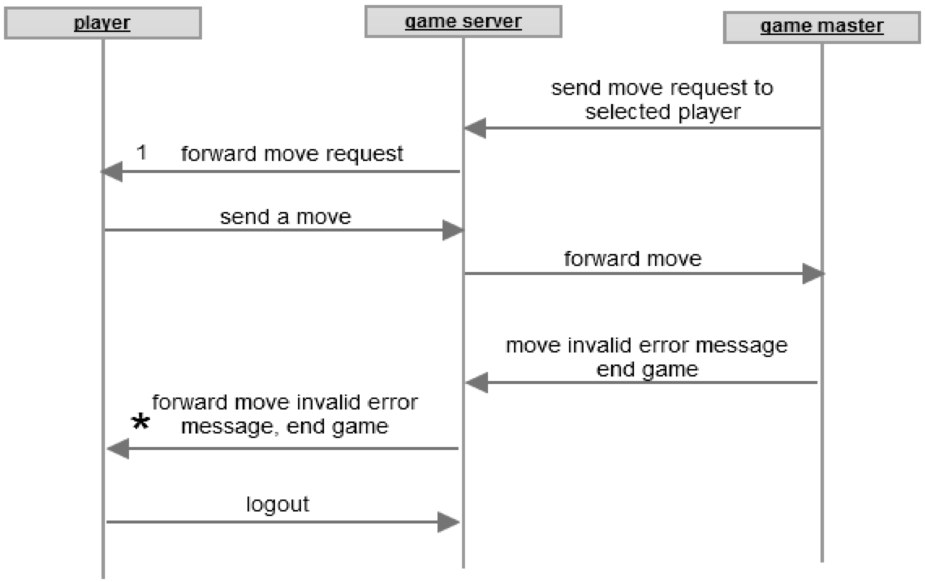
UGS Documentation: Event flow diagram (2/4) – Valid move

Event Flow Diagram: Game Flow - move valid



UGS Documentation: Event flow diagram (3/4) – Invalid move

Event Flow Diagram: Game Flow - move invalid



player

game server

game master

send move request to selected player

1 forward move request

send a move

forward move

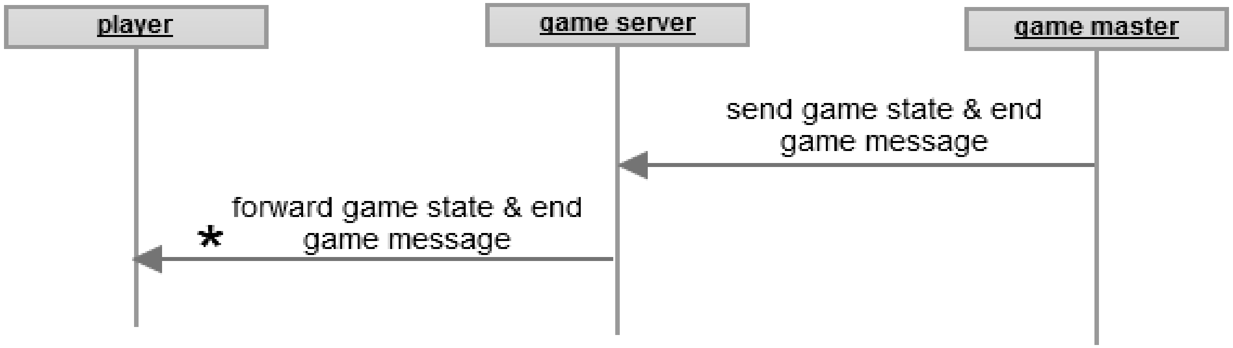
move invalid error message end game

* forward move invalid error message, end game

logout

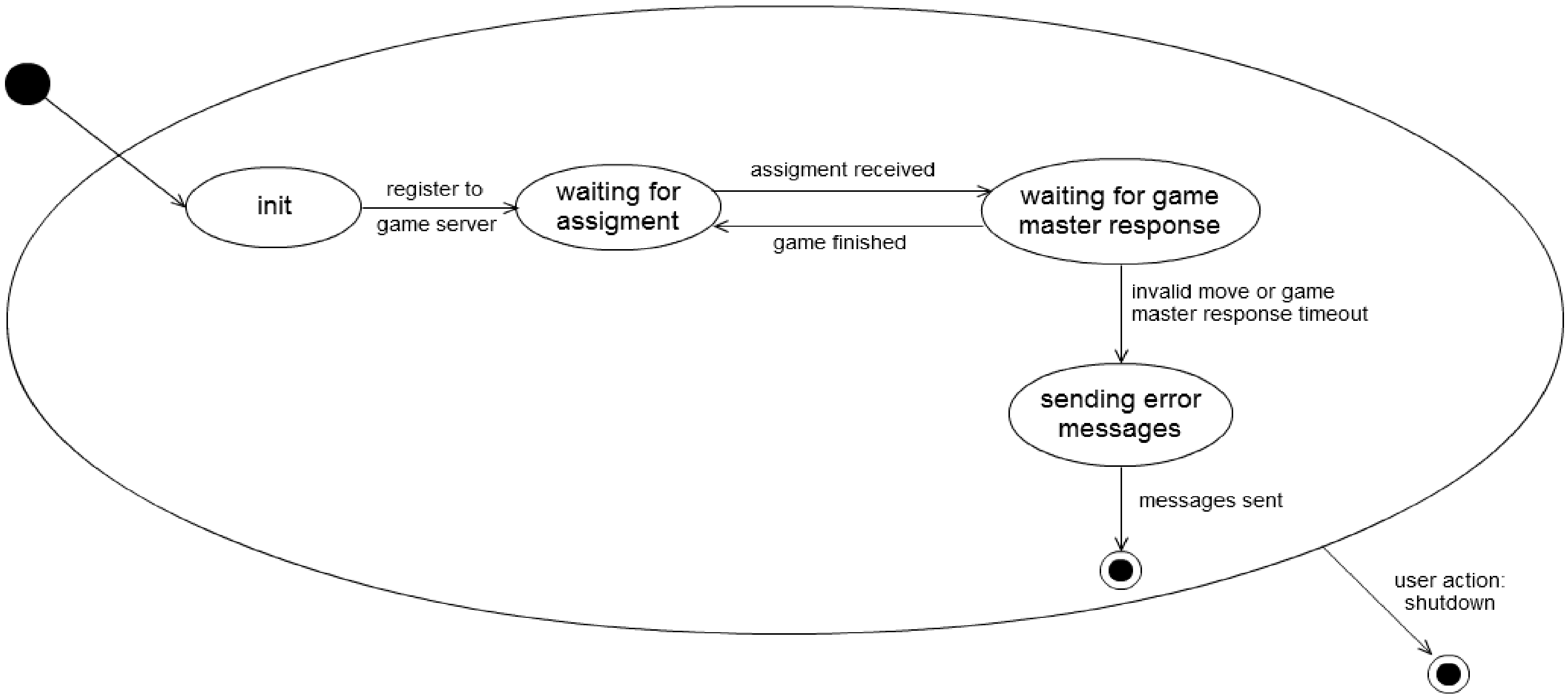
UGS Documentation: Event flow diagram (4/4) – End game

Event Flow Diagram: End Game

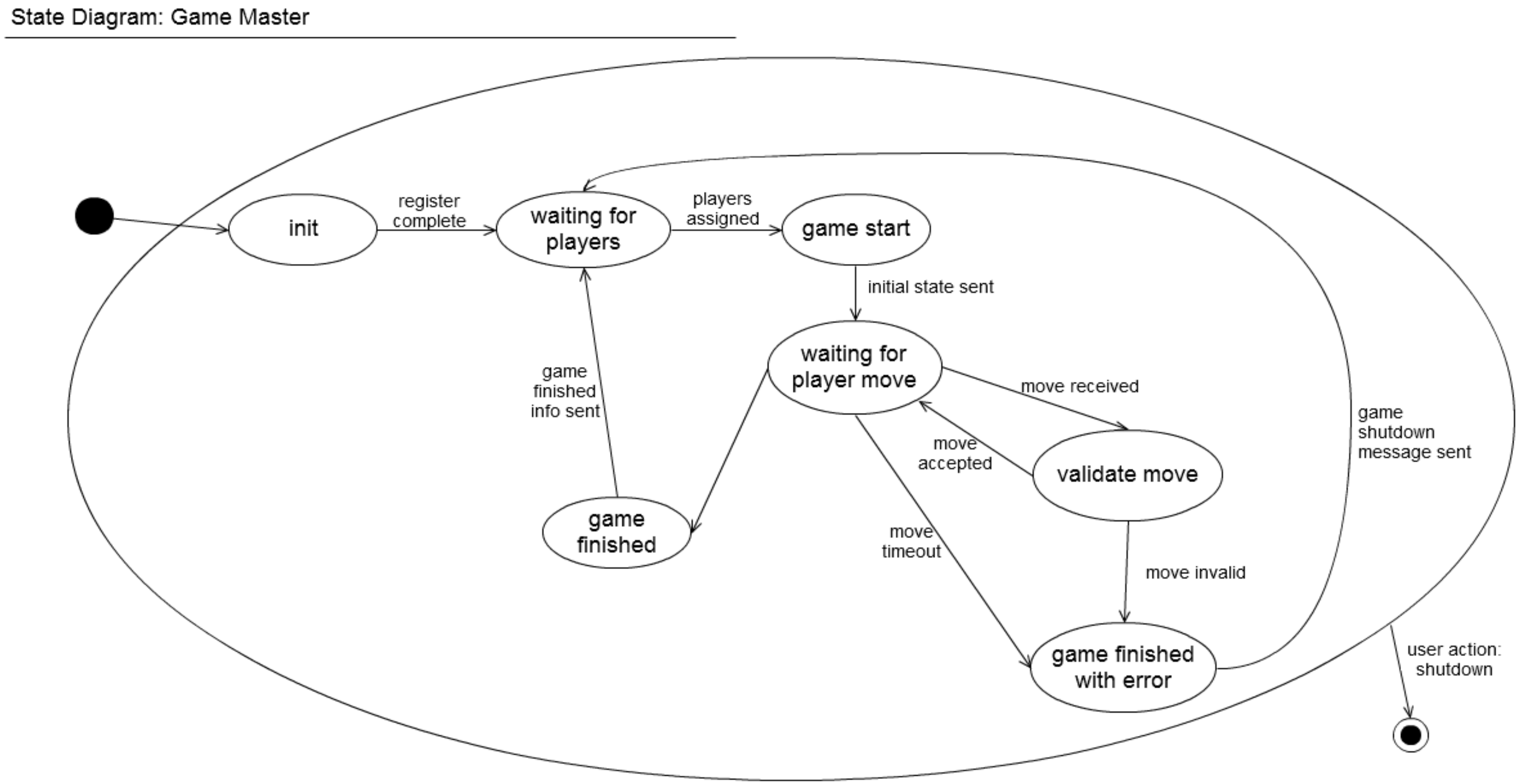

player
game server
game master
send game state & end game message
forward game state & end game message
*

UGS Documentation: State diagram (1/3) – Game client

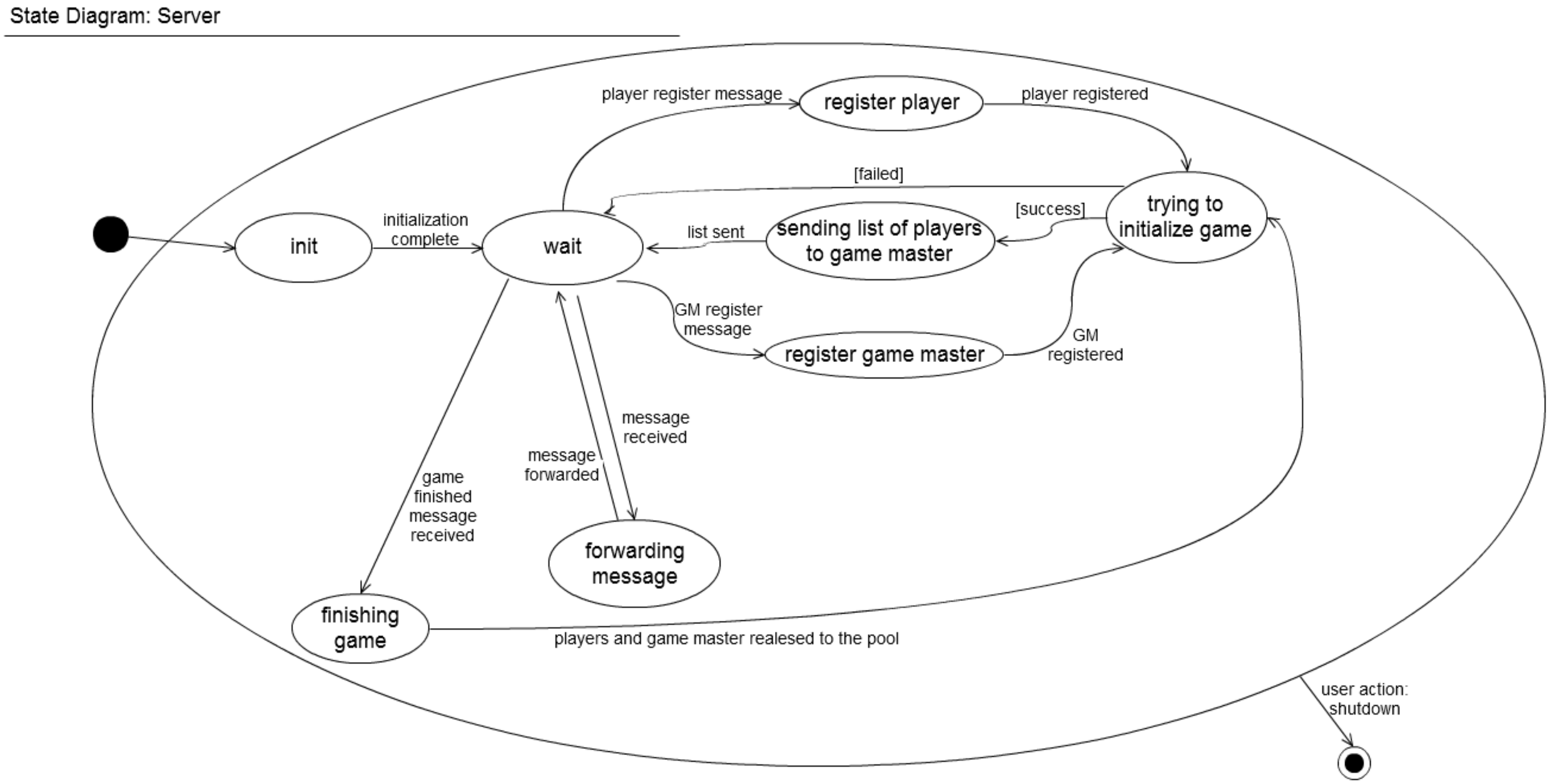
State Diagram: Client



UGS Documentation: State diagram (2/3) – Game master

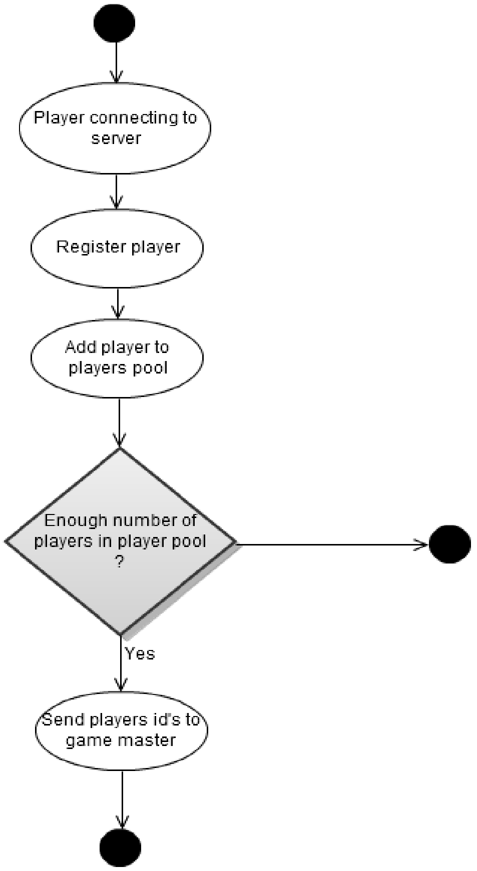


UGS Documentation: State diagram (3/3) – Game server



UGS Documentation: Activity diagram (1/5) – Loging to server

Server player log in



Player connecting to server

Register player

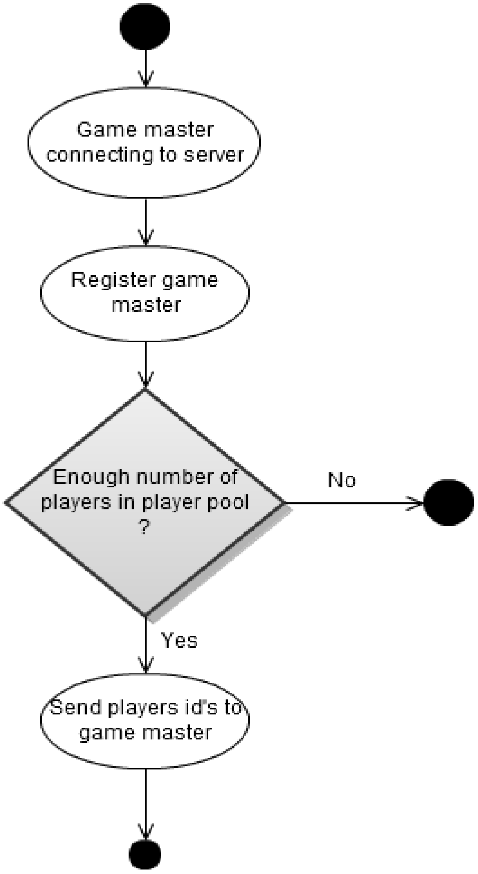
Add player to players pool

Enough number of players in player pool ?

Yes

Send players id's to game master

Server game master log in



Game master connecting to server

Register game master

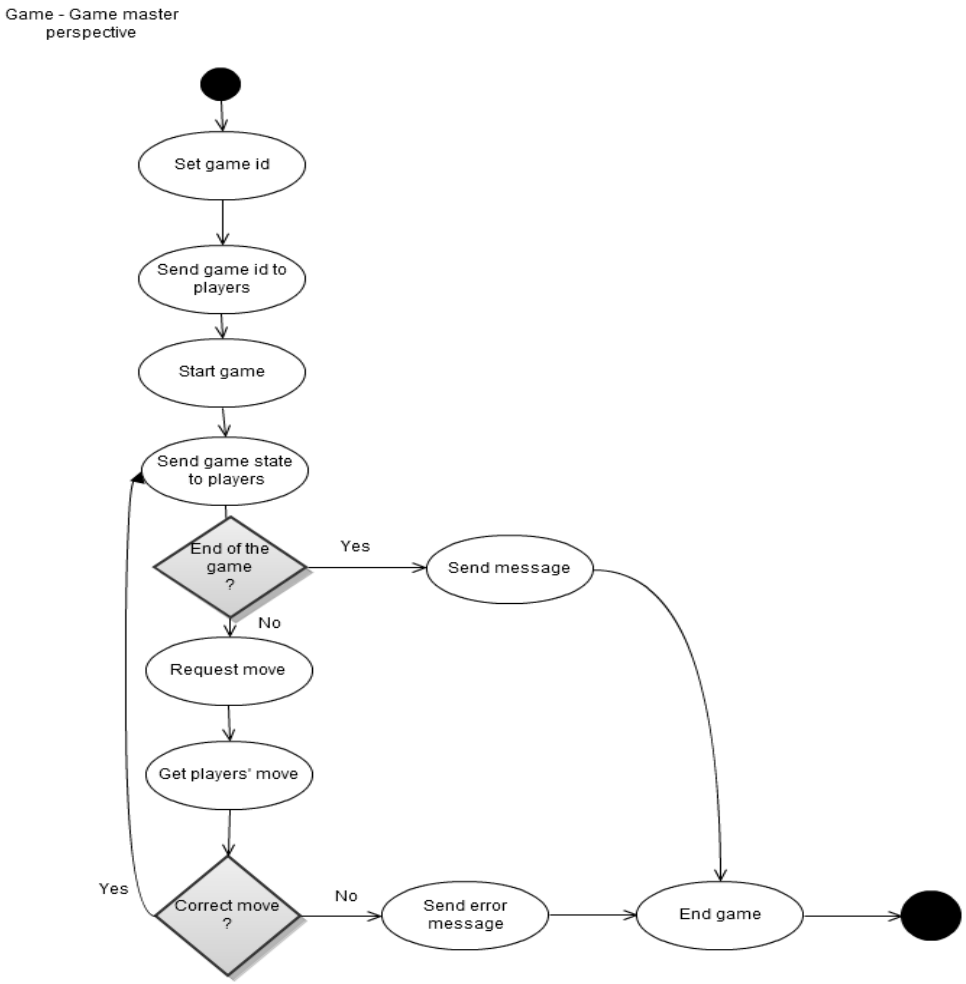
Enough number of players in player pool ?

No

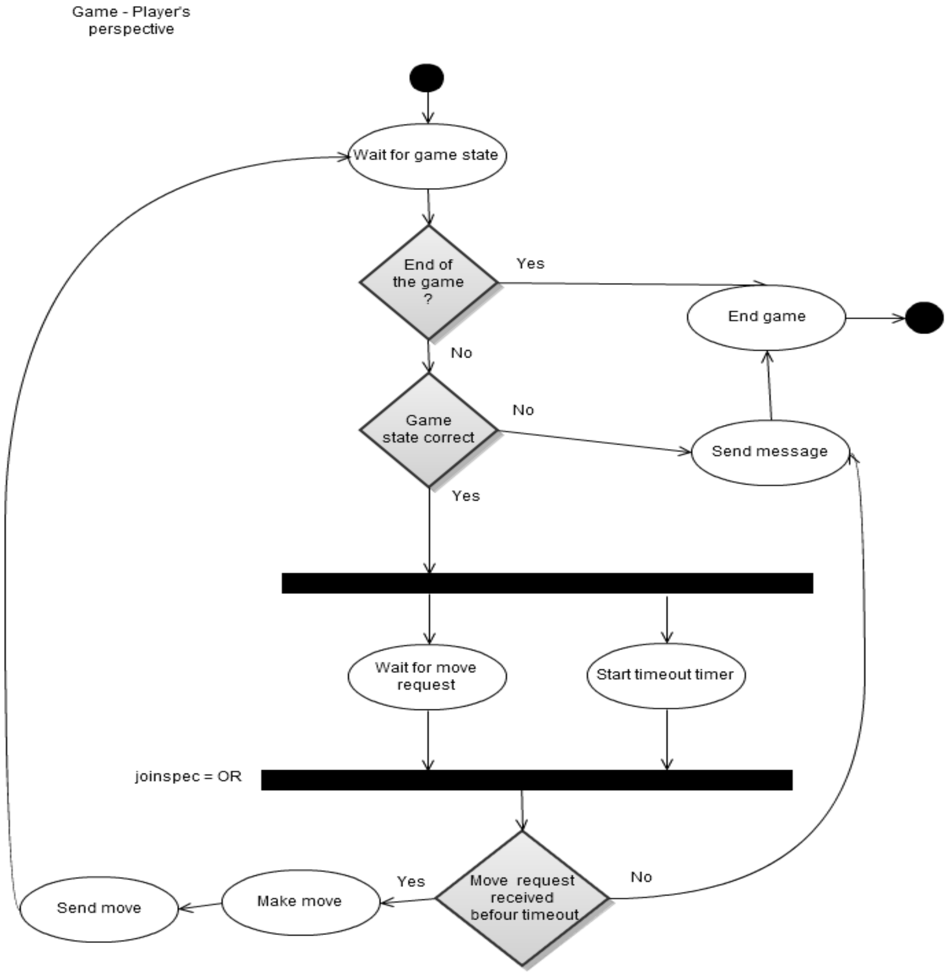
Yes

Send players id's to game master

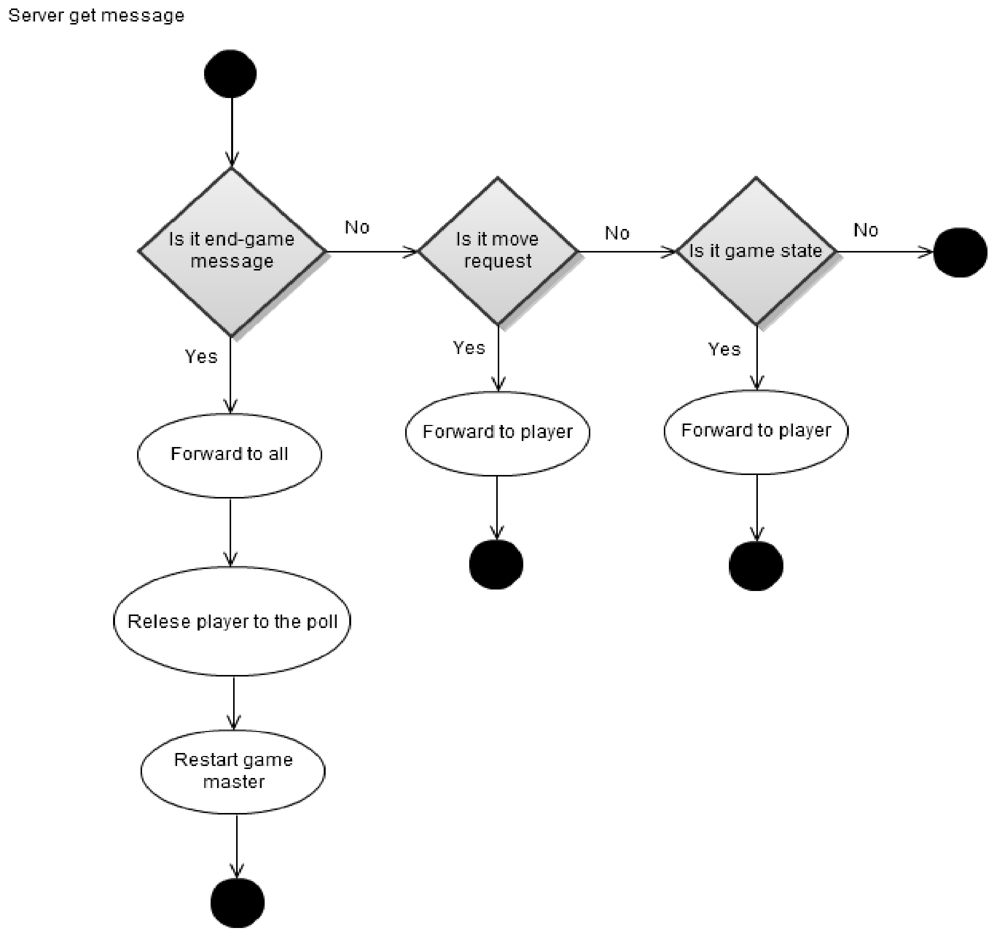
UGS Documentation: Activity diagram (2/5) – Game master



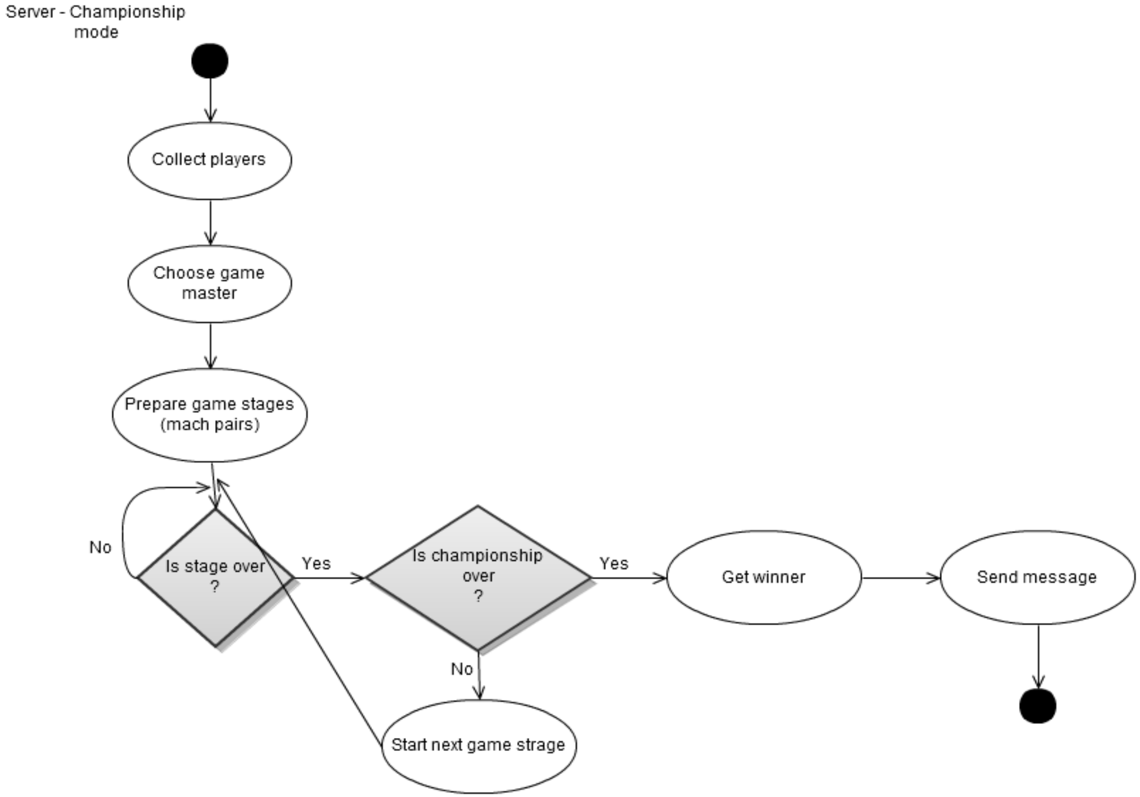
UGS Documentation: Activity diagram (3/5) – Game player



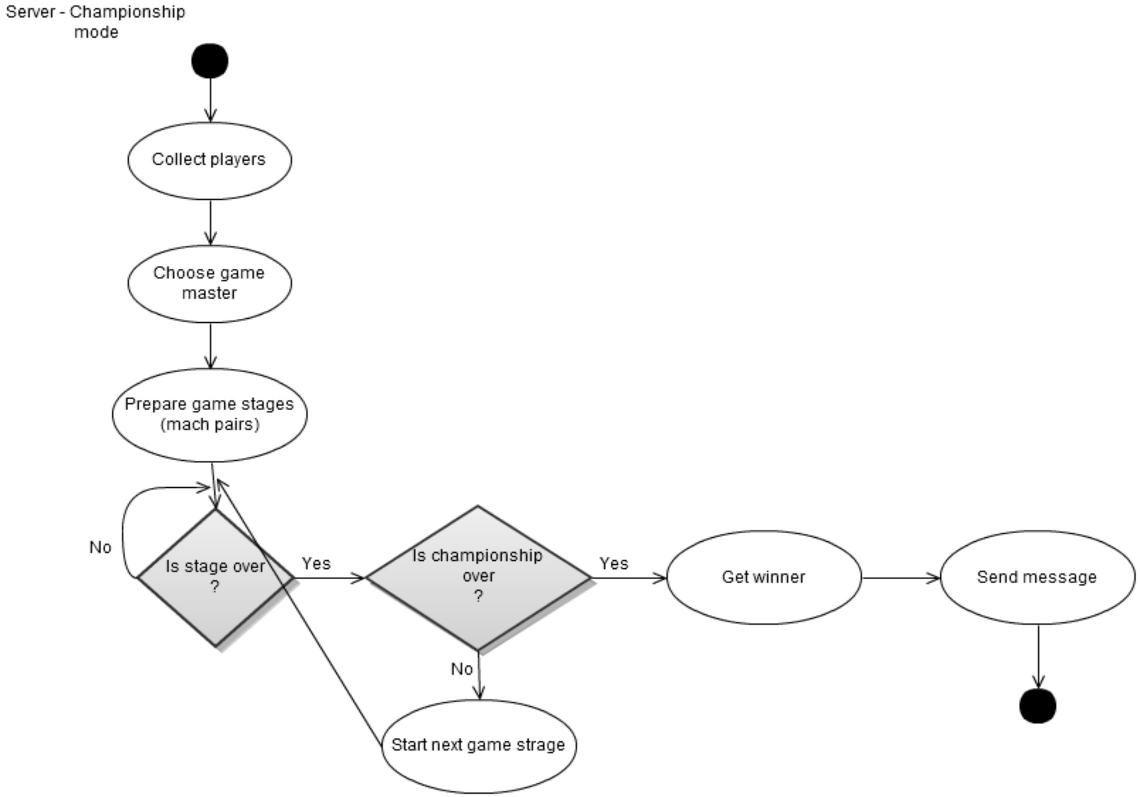
UGS Documentation: Activity diagram (4/5) – Server message processing



UGS Documentation: Activity diagram (5/5) – Server championship mode



UGS Documentation: Activity diagram (5/5) – Server championship mode


Server - Championship mode
Collect players
Choose game master
Prepare game stages (mach pairs)
No
Is stage over ?
Yes
Is championship over ?
Yes
Get winner
Send message
No
Start next game strage