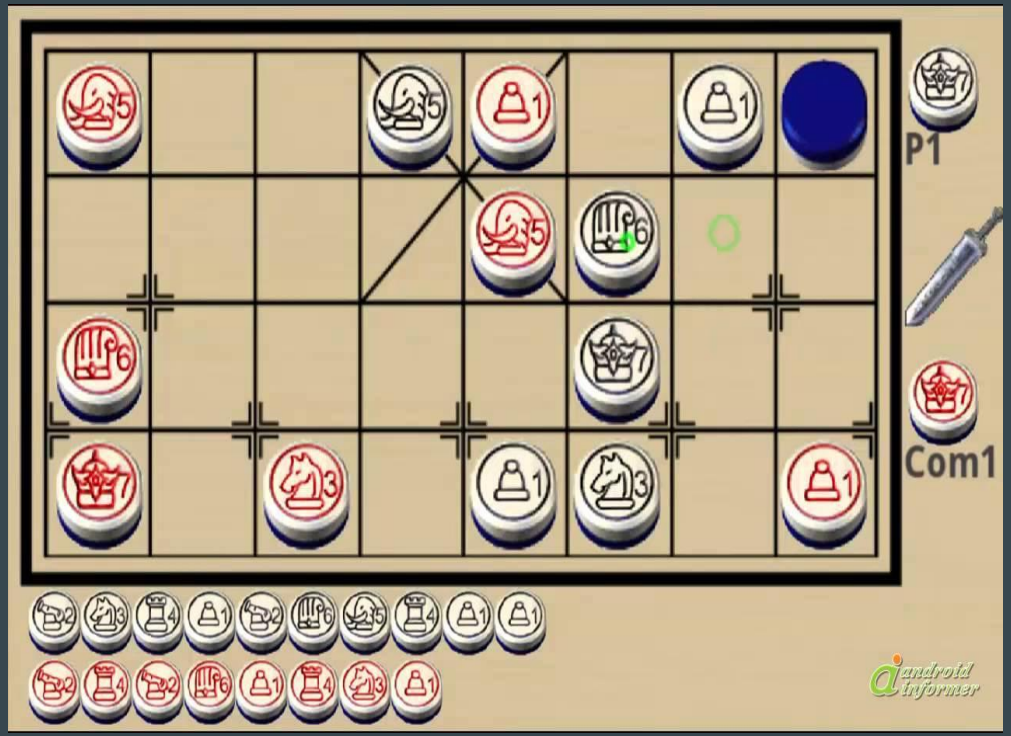
Banqi

Generic Team Name

Jared Zymbaluk, Kory Clark, Nick Wilson, Sam Maxwell, Tanner Pflager

What is Banqi?



- Known as Half Chess, Dark Chess, and Blind Chess
- Played on a 4 x 8 grid
- 16 pieces for each player
- Pieces:
  - General (1)
  - Advisor (2)
  - Elephant (2)
  - Chariot (2)
  - Horse (2)
  - Soldier (5)
  - Canon (2)

Piece Hierarchy

Name of piece	Number per side	Notes
General, marshal, "king" 帥 *shuai*, 將 *jiàng*	 × 1	Highest rank. Captures everything except soldiers.
Advisor, guard, minister*, assistant, mandarin, warrior 仕 *shi*, 士 *shì*	 × 2	Second-highest rank in Taiwanese version.
Elephant, war elephant, minister* 相 *xiàng*, 象 *xiàng*	 × 2	
Chariot, rook, cart 俥 *jū*, 車 *jū*	 × 2	Second-highest rank in Hong Kong version.
Horse, cavalry 傌 *mà*, 馬 *mǎ*	 × 2	
Soldier, private, pawn 兵 *bīng*, 卒 *zú*	 × 5	Lowest rank, except able to capture the general.
Cannon, catapult 炮 *pào*, 砲 *pào*	 × 2	Abilities differ in Taiwanese variations.

Rules

- Starts with all 32 pieces randomly placed face down on the board
- First player flips a piece, the color of the piece is the color they will play with
- Turns alternate until the game is finished
- In a turn a player can
  - Flip a face down piece
  - Move a piece up, down, left or right
  - Capture an opponent's piece that is lower in the hierarchy

  General > Advisor > Elephant > Chariot > Horse > Cannon > Soldier

- Exceptions
  - Cannons can capture any rank by moving any distance in a row or column and jumping over a single intermediate piece
  - Soldiers can capture the General

Decisions

- Implementing Taiwanese version
- Coded as a Java executable
- Database will be created to keep track of accounts
- Keeps track of user statistics (win loss ratio, etc.)
- GUI will be implemented using some Java API (most likely Swing)
- User can have any number of games with other users
- Will be implemented “Words With Friends” style

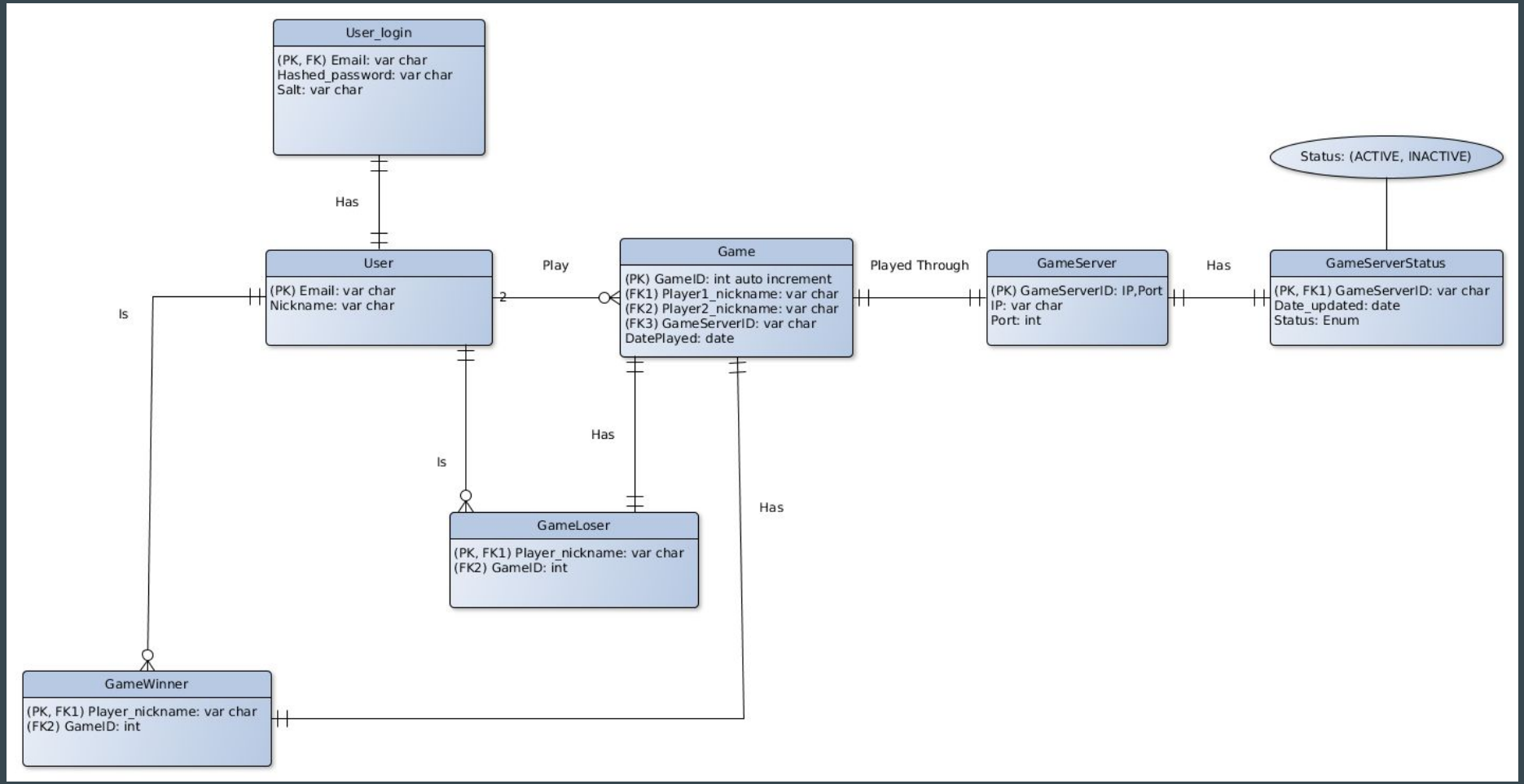
Web Server and Database

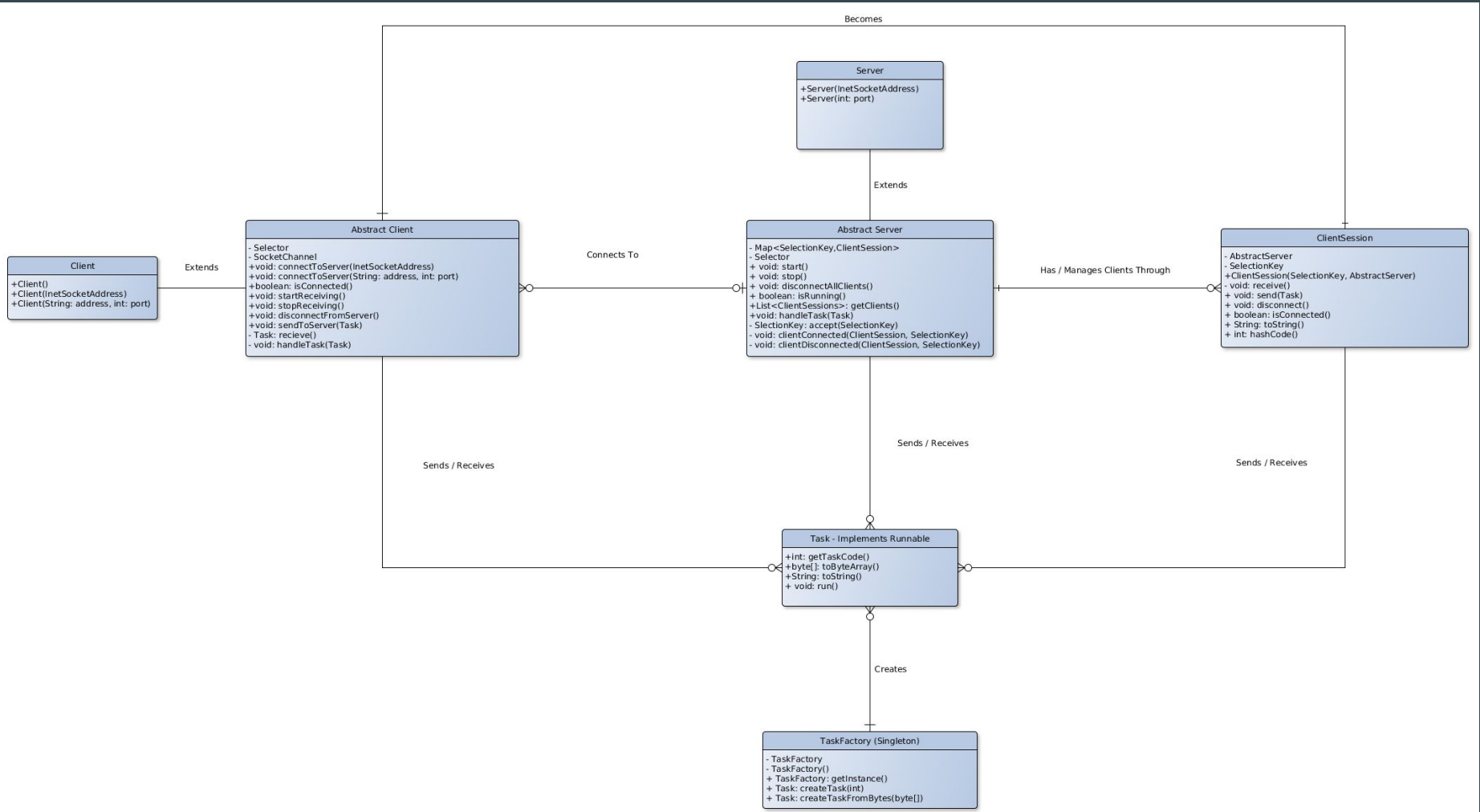
Web Server

- Clients will connect via TCP (Transmission control Protocol) using socket-based communication
- TCP vs UDP
  - Latency
  - Reliability
- Clients will send “requests” to the server

Database

- Keeps track of Account information
  - Store nickname, email, passwords (salt/hashed)
- Keeps track of active servers
- Will have an active MySQL server so queries can be made to retrieve an active list of server’s


User_login
(PK, FK) Email: var char
Hashed_password: var char
Salt: var char
Has
Status: (ACTIVE, INACTIVE)
User
(PK) Email: var char
Nickname: var char
Play
2
Game
(PK) GameID: int auto increment
(FK1) Player1_nickname: var char
(FK2) Player2_nickname: var char
(FK3) GameServerID: var char
DatePlayed: date
Played Through
GameServer
(PK) GameServerID: IP,Port
IP: var char
Port: int
Has
GameServerStatus
(PK, FK1) GameServerID: var char
Date_updated: date
Status: Enum
Is
Is
Has
Has
GameLoser
(PK, FK1) Player_nickname: var char
(FK2) GameID: int
GameWinner
(PK, FK1) Player_nickname: var char
(FK2) GameID: int


Becomes
Server
+Server(InetSocketAddress)
+Server(int: port)
Extends
Client
+Client()
+Client(InetSocketAddress)
+Client(String: address, int: port)
Extends
Abstract Client
- Selector
- SocketChannel
+void: connectToServer(InetSocketAddress)
+void: connectToServer(String: address, int: port)
+boolean: isConnected()
+void: startReceiving()
+void: stopReceiving()
+void: disconnectFromServer()
+void: sendToServer(Task)
- Task: recieve()
- void: handleTask(Task)
Connects To
Abstract Server
- Map<SelectionKey,ClientSession>
- Selector
+ void: start()
+ void: stop()
+ void: disconnectAllClients()
+ boolean: isRunning()
+List<ClientSessions>: getClients()
+void: handleTask(Task)
- SlectionKey: accept(SelectionKey)
- void: clientConnected(ClientSession, SelectionKey)
- void: clientDisconnected(ClientSession, SelectionKey)
Has / Manages Clients Through
ClientSession
- AbstractServer
- SelectionKey
+ClientSession(SelectionKey, AbstractServer)
- void: receive()
+ void: send(Task)
+ void: disconnect()
+ boolean: isConnected()
+ String: toString()
+ int: hashCode()
Sends / Receives
Sends / Receives
Sends / Receives
Task - Implements Runnable
+int: getTaskCode()
+byte[]: toByteArray()
+String: toString()
+ void: run()
Creates
TaskFactory (Singleton)
- TaskFactory
- TaskFactory()
+ TaskFactory: getInstance()
+ Task: createTask(int)
+ Task: createTaskFromBytes(byte[])

Use Case Diagram