Chad

Team WareWolves

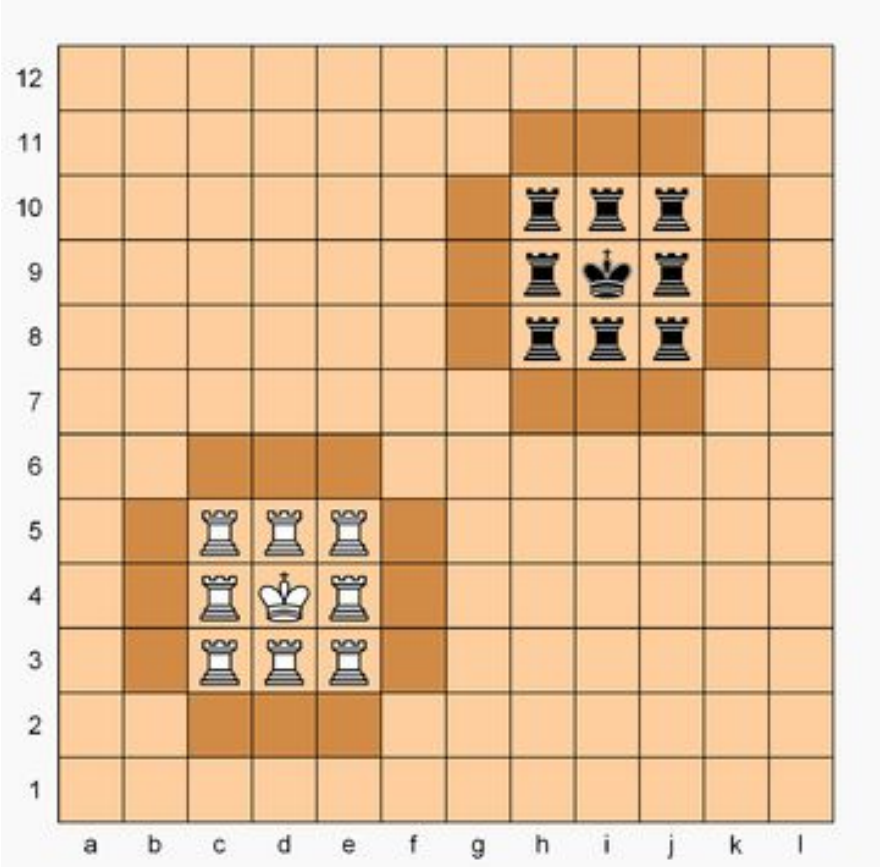
A Simple and Elegant game of
Tactical and Strategic depth

Ben Goodwin, Joshua Munoz, Josiah May, Luis Rodriguez, Miles Wood

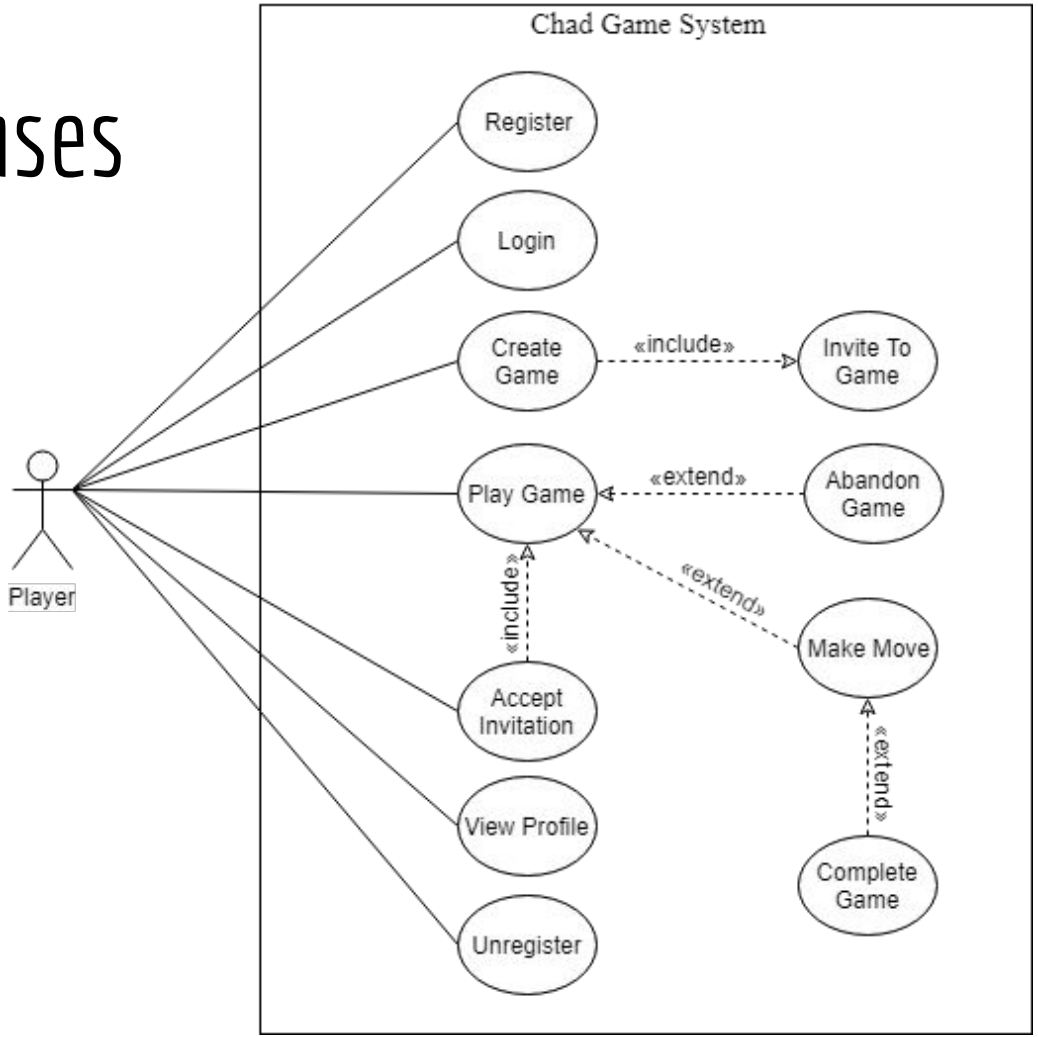
Chad: Game Overview

- Played on a 12x12 gameboard with two 3x3 ‘castles’ .
- Castles are surrounded on all four sides by ‘walls’
- Each player has 8 rooks and one king which start in their own castle.

- Rooks are promoted to queens if they enter their opponent's castle.
- The king cannot leave its castle.
- The king moves as a combination of the king and knight in normal chess.
- A player wins by forcing a checkmate on the opponent's king.

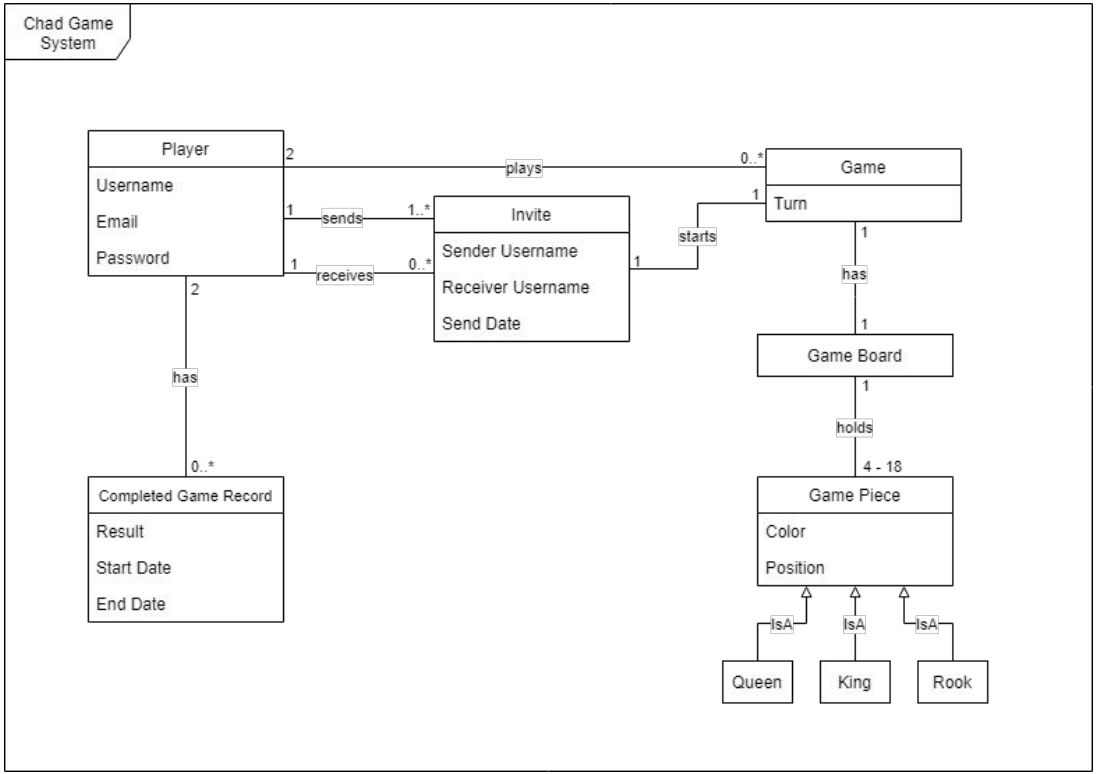


Use Cases


Chad Game System
Register
Login
Create Game
«include»
Invite To Game
Play Game
«extend»
Abandon Game
«include»
«extend»
Make Move
Accept Invitation
«extend»
View Profile
Complete Game
Unregister
Player

Domain Model

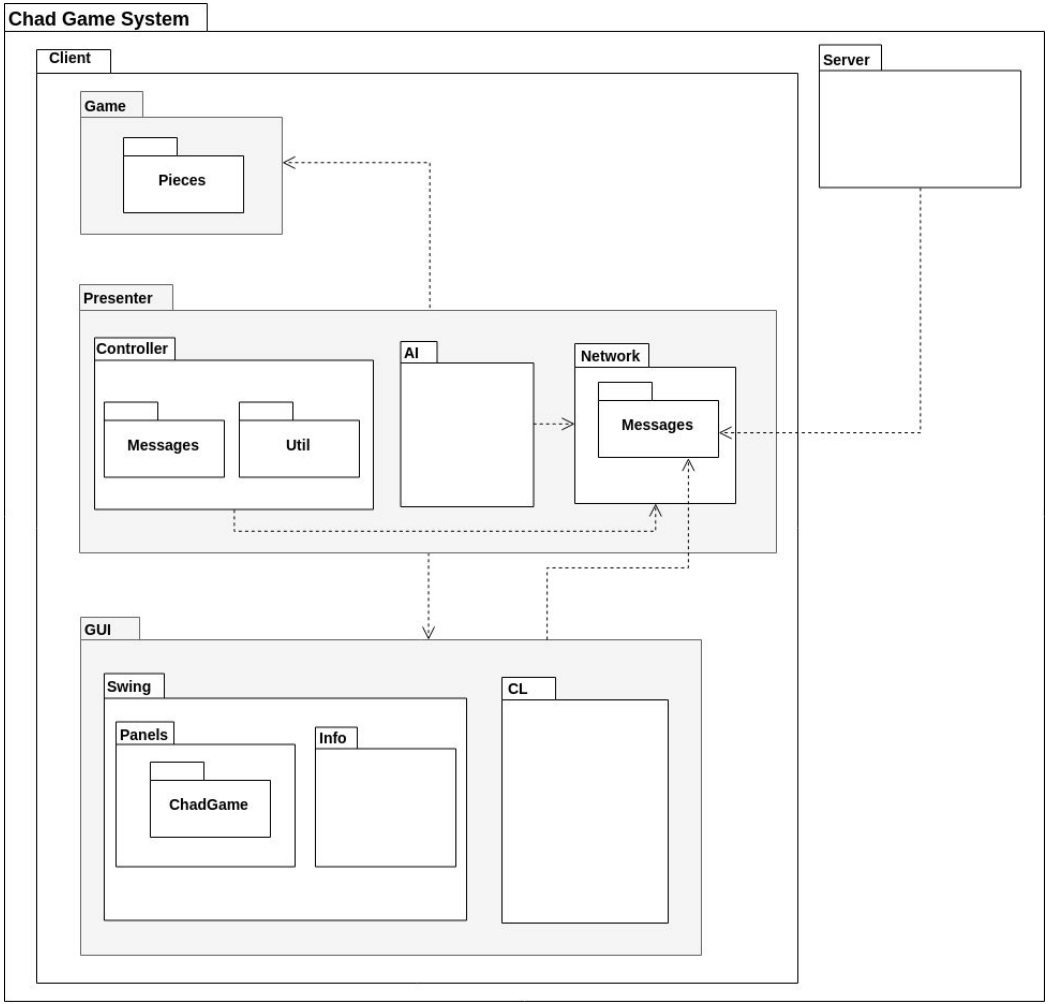
- Broke Use Cases into this Domain Model
- Invites essential
- Defined key interactions



Package Diagram

- Primary packages in Model-View-Presenter

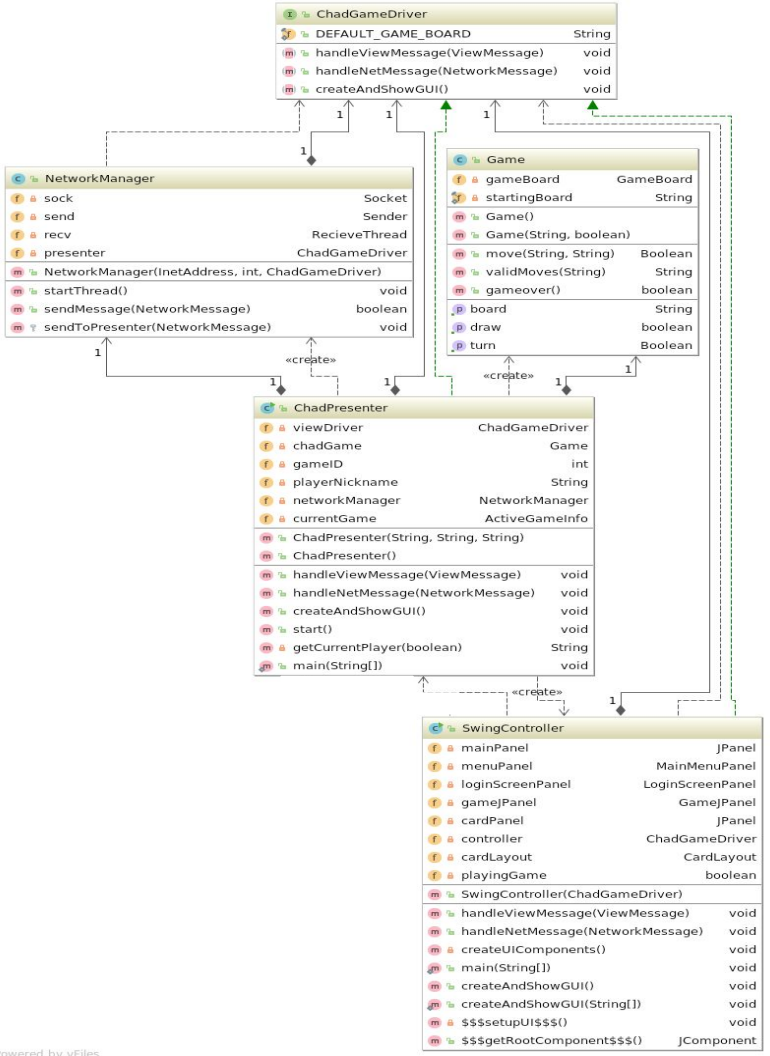
- Client-Server for login, game tracking, game history and asynchronous games



Class Diagram

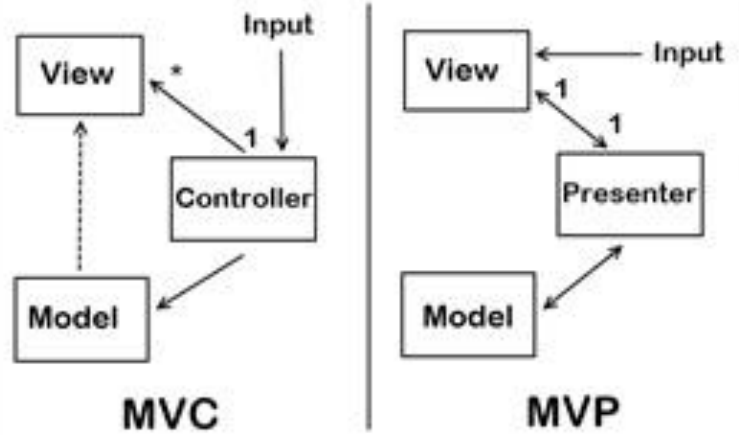
- Primary Classes of the Model-View-Presenter

- Full Class Diagram in our Design Document much too large to show on slides



🏰 Source: Design Patterns

- Main Architecture: Model View Presenter
- GUI Chad Piece Factory
- GUI Drivers strong compositions
- Game Model is a Facade
  - Template Method for Game Pieces
- Client-Server Architecture for Networking



 Technology

- Swing GUI
- MySQL DB
- Maven Build
- Travis CI
- Code Climate

Testing

- Unit Testing
  - Tests at every class

- System Testing
  - Server and Network interactions
  - Play testing
  - AI testing

Traceability Link Matrix

	ChadPresenter	HashPasswords	Game	GameBoard	Point	*Piece*	NetworkManager	RecieveThread	Sender	*NetworkMessage*	*ViewMessage*	ViewMessageFactory	GameView	LoginView	MenuView	InviteView	ProfileView	SwingController	AIDriver	ChadServer	Query
P-01 (Register)	X	X					X	X	X	X	X	X		X				X		X	X
P-02 (Login)	X	X					X	X	X	X	X	X		X				X		X	X
P-03 (Create Game)	X		X	X	X	X	X	X	X	X	X	X			X			X	X	X	X
P-04 (Invite to Game)	X						X		X	X	X	X			X	X		X	X	X	X
P-05 (Play Game)	X		X	X	X	X	X	X	X	X	X	X	X		X			X	X	X	X
P-06 (Accept Invite)	X						X	X	X	X	X	X			X	X		X		X	X
P-07 (Make Move)	X		X	X	X	X	X	X	X	X	X	X	X					X	X	X	X
P-08 (Complete Game)	X		X				X	X	X	X	X	X	X					X	X	X	X
P-09 (Abandon Game)	X						X		X	X	X	X	X					X		X	X
P-10 (Unregister)	X	X					X		X	X	X	X			X			X		X	X
P-11 (View Profile)	X						X	X	X	X	X	X			X		X	X	X	X	X
	Classes in *italics* are representing the abstract classes that the concrete classes extend and actually fulfil the use case for sake of simplicity and keeping the matrix small enough to read																				

🏰 Challenges

- Defining the database tables and columns
  - As we continued development we had to add additional columns and expand varchar ranges to fit the whole system as it evolved.

- Communicating about implementation of MVP
  - For each message some knowledge of the logic for other components was needed to send the appropriate information.

- Integrating Java and Python together for the AI Agent
  - The AI was trained through Python and querying the neural net for AI Agent moves required a Python call.

Lessons Learned

- Pick NIO or IO for networking interactions, not both.
  - Picking one package from the start would have made the networking part of the system more coherent and easier to maintain

- Messages to the Viewer should have been created alongside new features in the GUI/CLI
  - Each message should have been added to the system when a new feature was created instead of creating them all as a whole in the beginning.

Demonstration

Questions?