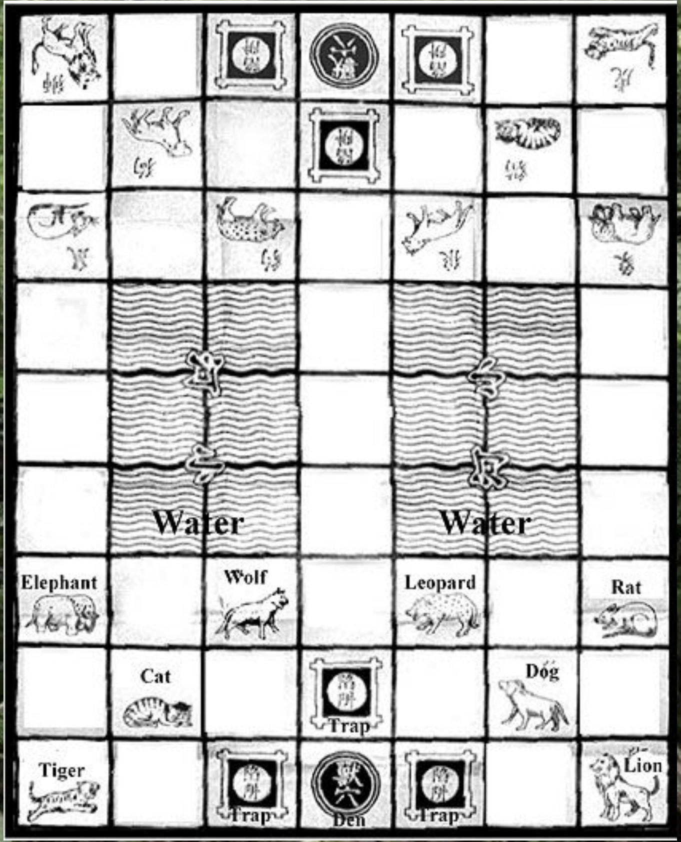
Jungle Game Application

Team.name()

David Wells
Alex Bailey
Tim Rooney
Brian Crane

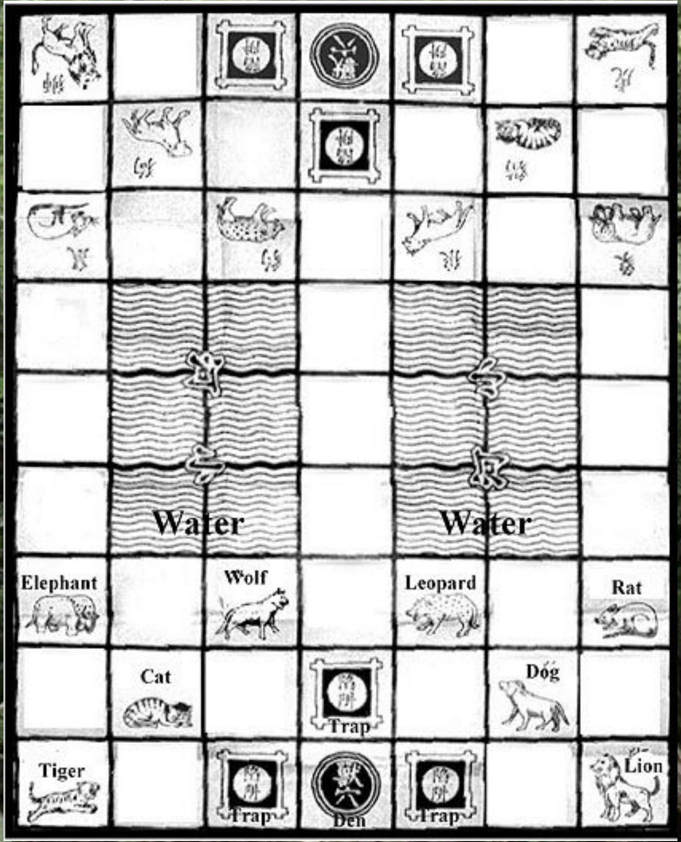
Jungle Board Game

- Modern Chinese board game similar to chess
- Consists of a 7x9 board
- Contains animals, river tiles, a den for each team, and traps surrounding the den
- 8 animals on each team with different ranks



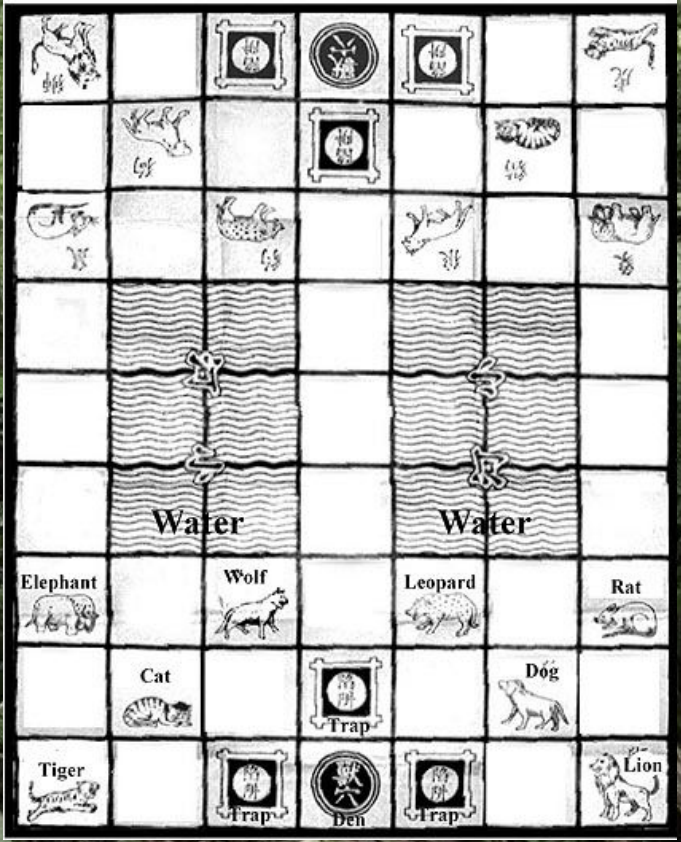
Game Rules

- Animals can capture pieces of equal or lower rank than themself
- Win by capturing all enemy pieces or by getting a piece to the enemy den
- Traps reduce enemy piece's rank to zero while on the tile
- Water tiles cannot be walked on (except by the rat)



Piece Abilities

- The rat
  - Can walk in the river
  - Cannot be captured while in the river
  - Can capture the elephant if not in the river
- Lion and Tiger
  - Can leap over the river horizontally or vertically if there is no piece between points

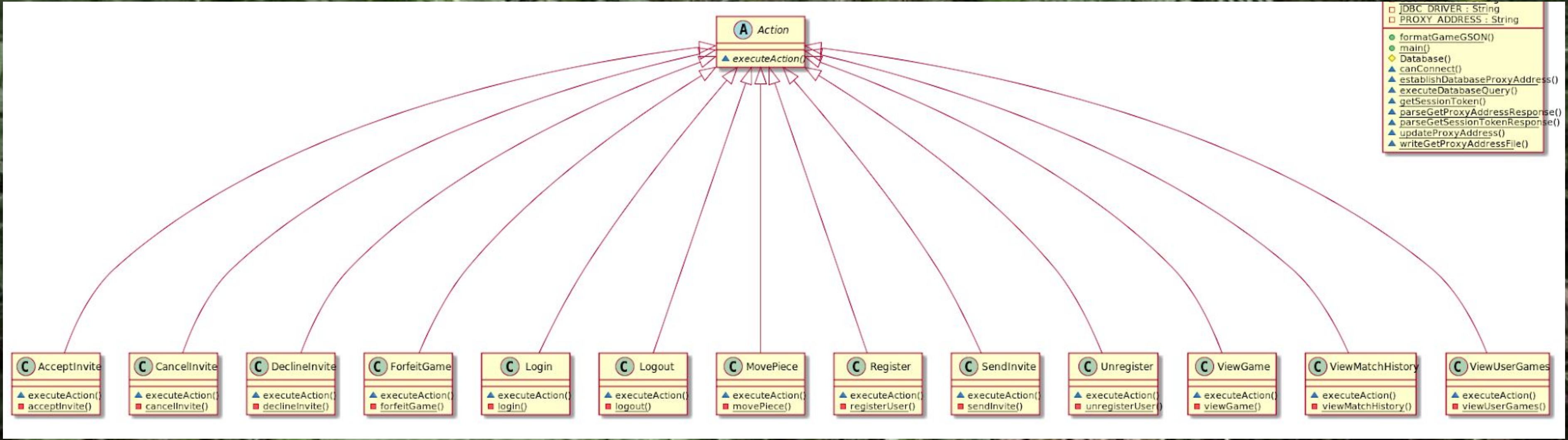


Development Process

- Followed Scrum methodologies
  - Met frequently
  - Discussed progress and upcoming goals
- Practiced Agile including:
  - Test Driven Development
  - Pair Programming
  - Common work area where we regularly develop together
- Tools
  - GitHub to collaborate code and deliverables
    - Required reviews of code before pushing
  - Slack for communication
  - IntelliJ for development
  - Java, React, Selenium

Design Patterns - Abstract Action class

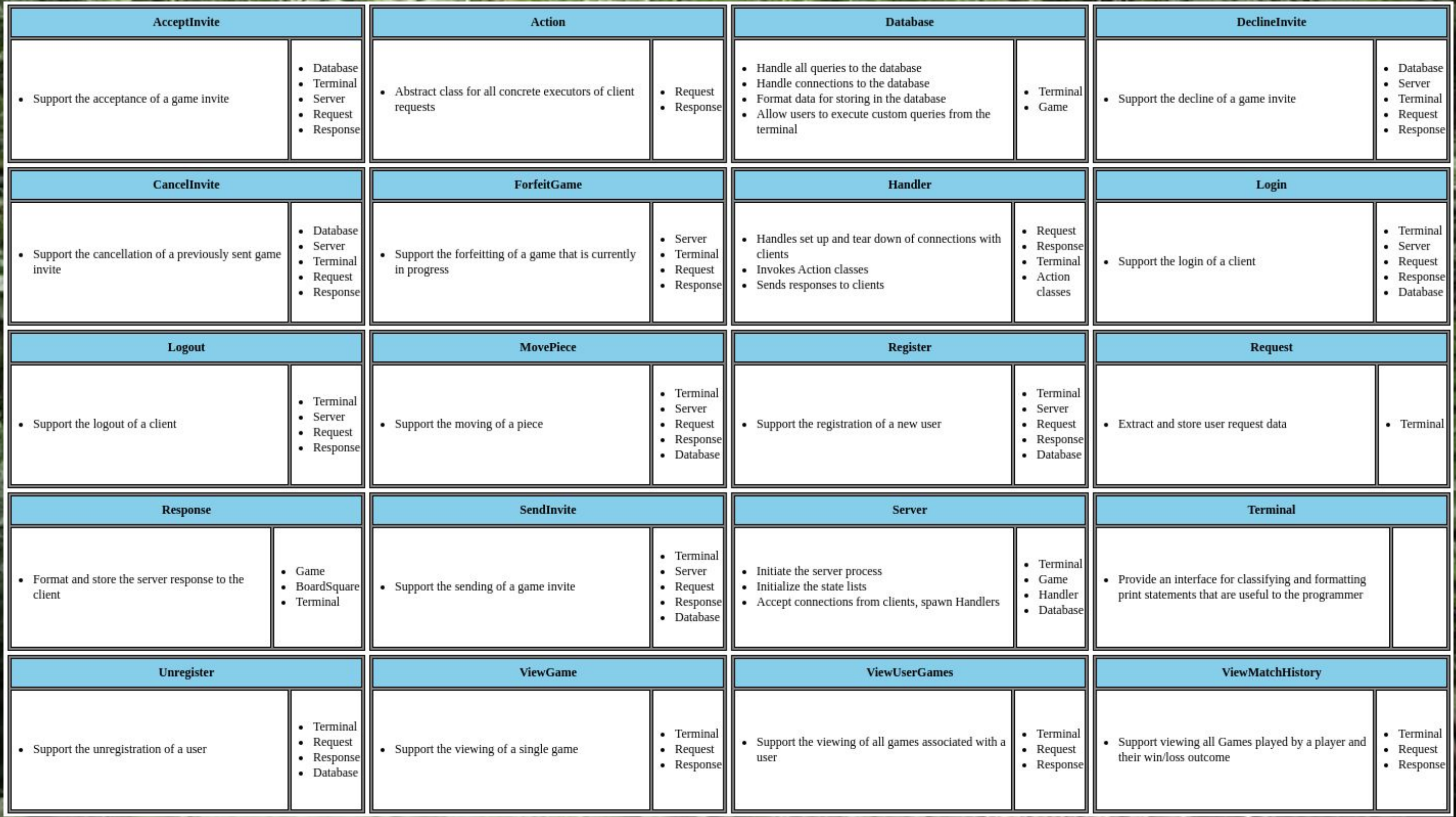
- Continued with design from P2
- Action is an abstract class
    - One abstract method: executeAction( );
- Each concrete action class extends Action
    - Represents a possible action an client can request the server to perform
    - Second private method that executeAction( ) calls
- Makes low coupling: each action user story affected exactly one server class (see TLM, CRC)



Design Decisions
Traceability Link Matrix

Issue Number	App.js	Board.js	Gamerules.js	Games.js	History.js	Home.js	Invite.js	Login.js	Register.js	Square.js	User.js	index.js	BoardSquare.java	Game.java	GamePiece.java	GameTest.java	GameUtils.java	Move.java	AcceptInvite.java	Action.java	CancelInvite.java	Database.java	DeclineInvite.java	ForfeitGame.java	Handler.java	Login.java	Logout.java	MovePiece.java	Register.java	Request.java	Response.java	SendInvite.java	Server.java	Terminal.java	Unregister.java	ViewGame.java	ViewMatchHistory.java	ViewUserGames.java
E14		x																						x														
T125																								x														
T172		x																																				
E12											x																								x			
T134											x																											
T136																																			x			
T137																																			x			
T138																																			x			
E13		x			x																																x	x
T126																																						x
T140					x																																	
T141																																					x	
T142		x																																				

Design Decisions
GameServer CRC Diagram



AcceptInvite	
• Support the acceptance of a game invite	• Database • Terminal • Server • Request • Response

Action	
• Abstract class for all concrete executors of client requests	• Request • Response

Database	
• Handle all queries to the database • Handle connections to the database • Format data for storing in the database • Allow users to execute custom queries from the terminal	• Terminal • Game

DeclineInvite	
• Support the decline of a game invite	• Database • Server • Terminal • Request • Response

CancelInvite	
• Support the cancellation of a previously sent game invite	• Database • Server • Terminal • Request • Response

ForfeitGame	
• Support the forfeitting of a game that is currently in progress	• Server • Terminal • Request • Response

Handler	
• Handles set up and tear down of connections with clients • Invokes Action classes • Sends responses to clients	• Request • Response • Terminal • Action classes

Login	
• Support the login of a client	• Terminal • Server • Request • Response • Database

Logout	
• Support the logout of a client	• Terminal • Server • Request • Response

MovePiece	
• Support the moving of a piece	• Terminal • Server • Request • Response • Database

Register	
• Support the registration of a new user	• Terminal • Server • Request • Response • Database

Request	
• Extract and store user request data	• Terminal

Response	
• Format and store the server response to the client	• Game • BoardSquare • Terminal

SendInvite	
• Support the sending of a game invite	• Terminal • Server • Request • Response • Database

Server	
• Initiate the server process • Initialize the state lists • Accept connections from clients, spawn Handlers	• Terminal • Game • Handler • Database

Terminal	
• Provide an interface for classifying and formatting print statements that are useful to the programmer	

Unregister	
• Support the unregistration of a user	• Terminal • Request • Response • Database

ViewGame	
• Support the viewing of a single game	• Terminal • Request • Response

ViewUserGames	
• Support the viewing of all games associated with a user	• Terminal • Request • Response

ViewMatchHistory	
• Support viewing all Games played by a player and their win/loss outcome	• Terminal • Request • Response

Design Patterns - Abstract Action class (continued)

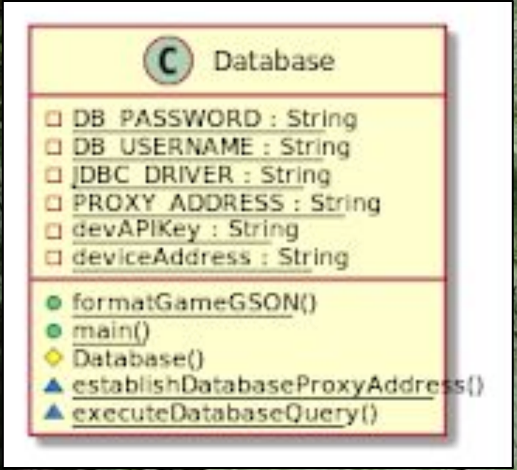
- Handler.java uses Java generics and parameterizable classes to map the action string to an Action class
  - No switch statements
- Easy to add new actions:
  - Create new class that extends Action.java…that’s it!

```
private void handleRequest() throws … {

      …

        String action = request.body.get("action");// e.g. “movePiece”
        Class<?> classSupportingAction = Class.forName(String.format("GameServer.%s", action));// get Action
                                                                                                // concrete class
        Object actionInstance = classSupportingAction.newInstance();// get instance of concrete class
        Method executeAction = classSupportingAction.getDeclaredMethod("executeAction", Request.class);
        response = ((Response)executeAction.invoke(actionInstance, request));// call executeAction()

      …

    }
```

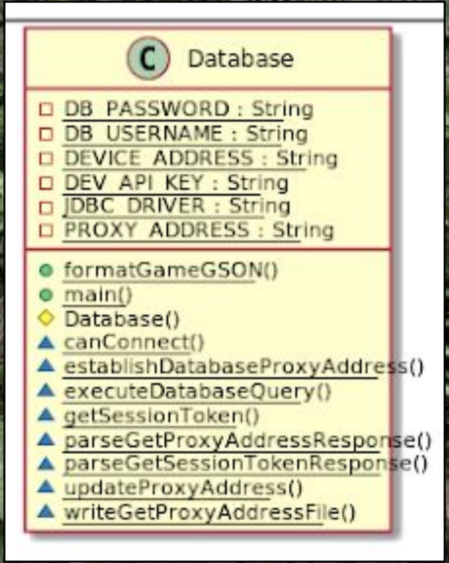
Refactoring Decisions - Extract method in Database.java

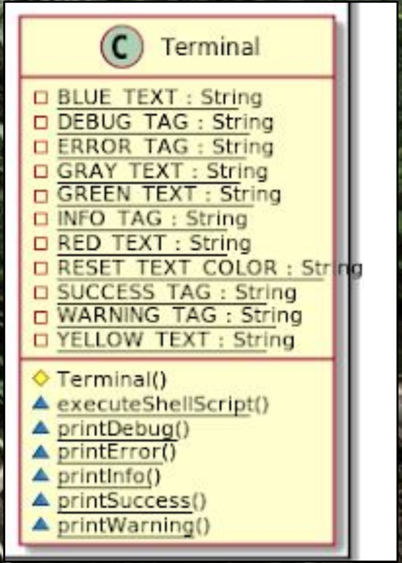
- Database.java had a single method for establishing a connection with the database
  - establishDatabaseProxyAddress( );
- Extract method was used to create eight new methods
  - Seven in Database.java
  - One in Terminal.java
- A bug was found in the refactoring process, too

BEFORE



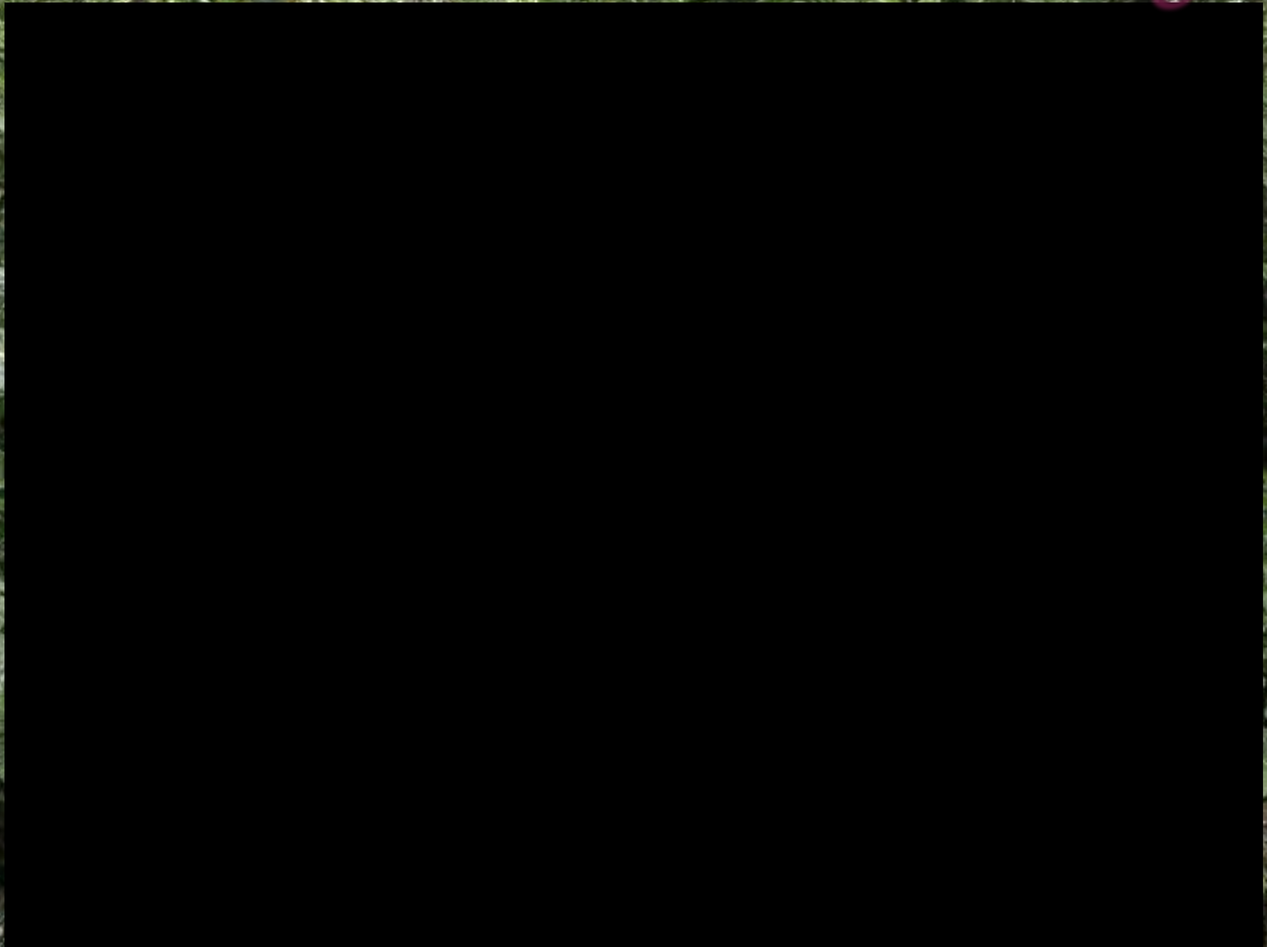
AFTER

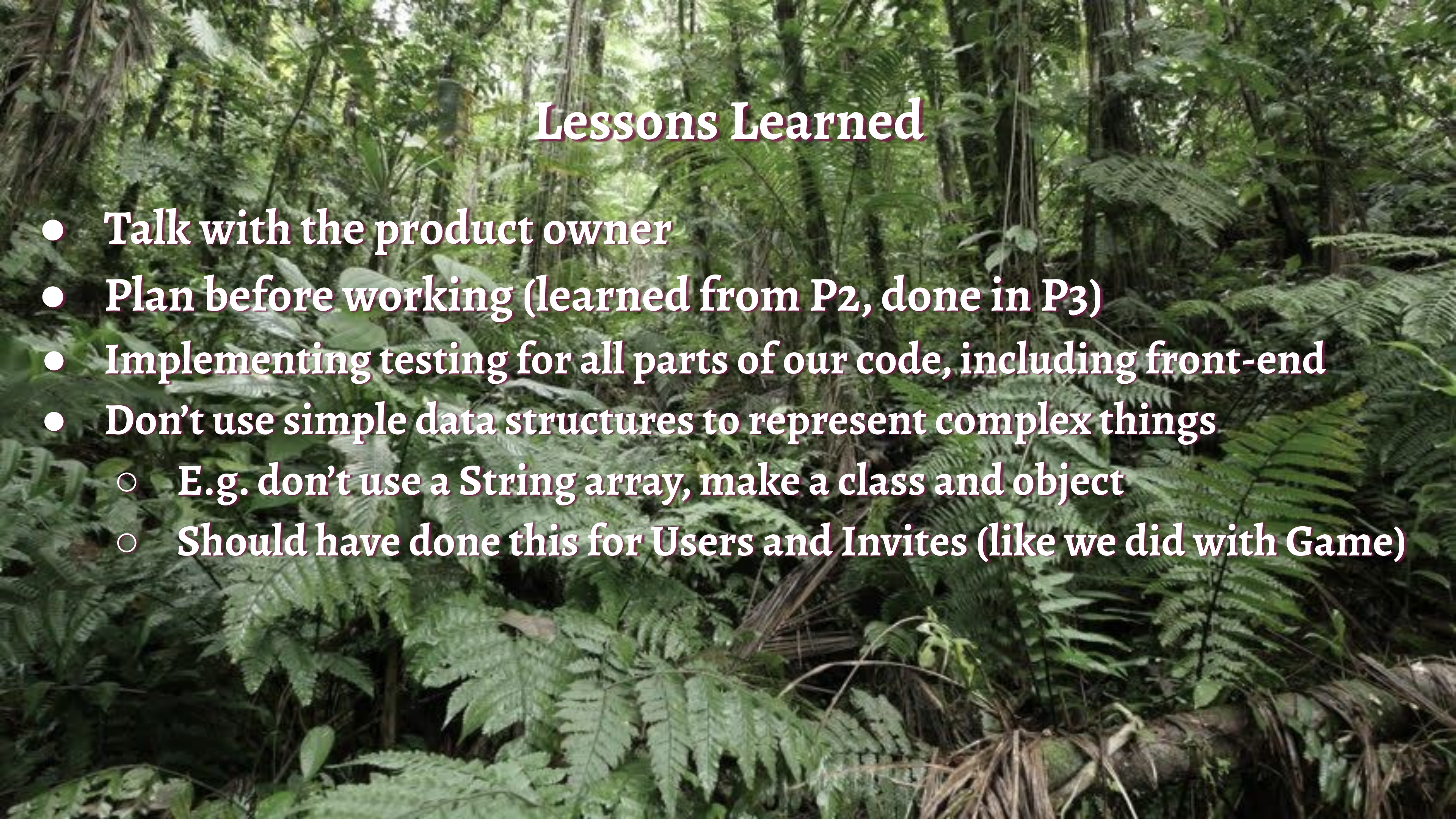




Demo

Selenium Front-End Testing





Lessons Learned

- Talk with the product owner
- Plan before working (learned from P2, done in P3)
- Implementing testing for all parts of our code, including front-end
- Don’t use simple data structures to represent complex things
  - E.g. don’t use a String array, make a class and object
  - Should have done this for Users and Invites (like we did with Game)

Questions?