# Lab sessions group 1. Analysis tasks

## 1. Introduction

In this session we are going to put in practice relevant object-oriented analysis tasks, namely: identification of user-level use cases, development of use cases, list of features, and domain model building.

Problem: a client wants you to build a software platform that will allows its subscribers to remotely play chess games.

## 2. Exercise 1

Prepare a list of questions that as analyst you would make to the client. This list must be primarily focussed at discover functional requirements.

At the session, the professor will act as client. He will engage in an exchange of questions/answers for some time. During this time, and as a result of the answers to the questions, write down:

* Features that the software system has to have.
* Requirements that the software system has to meet.
* User-level use cases.

Finally, review the user-level use cases and identify the most relevant ones. Order them so that the first one shall be the first use case that you will completely develop, and so on.

- Can we choose black or White pieces?

- Can we ask for draw?

- Can the user select the combination of colours for the board (daltonism)?

- Can we select different types of rules in a game?

- Do we only have public lobbies? Or can we invite someone?

- Can we have a friend list? Block list?

- Can spectators watch a game?

## 3. Exercise 2

Build the first user-level use case following the complete-formal format explained in class.

**USE CASES LIST:**

1. Play Game
2. Register
3. Propose a game to another person.
4. Replay game.
5. Resume unfinished game.

**Use case number:** 1

**Use case name:** Play game

**Use case goal:** allows two players play game.

**Actors:** Players (interested in using the system for playing).

**Preconditions:**

- Two players subscribed, and correctly logged in, and having agreed to play a game.

- System properly initialized.

- The system has selected who is the player it will give turn in first place.

**Postconditions:**

Game ended and stored

**Basic flow:**

1. System gives turn to one player.
2. System requests moments to the player.
3. Player that has the turn proposes a movement of one of his pieces from one square of the board to another square of the board.
4. System executes the movement.
5. System records movement in the trace.
6. System notifies movement to players.
7. System gives turn to the other player.

Repeat steps 2 to 7 while there is no winner.

1. Notify winner to players.
2. System saves trace of game.

**EXTENSIONS (Alternative Flow)**

3.a. Player proposes suspending the game.

1. The other player accepts.

2. System ends use case

3.b. Player proposes suspending the game.

1. The other player rejects.

2. Return to step 3 of basic flow.

CORRECTION OF THE PREVIOUS ONE

3.a Player proposes to suspend the game

1. Other player accepts

2. System notifies to players

3. Ends use case

1.a Other player rejects

1. System notifies to the user

2. Go to step 3 of basic flow

3.c. Player closes the game.

1. System ends use case.

3.d. The player ~~makes~~ proposes an invalid movement.

1. System notifies the user

2. Return to step 3 of basic flow.

7.a. The other player is disconnected.

1. System notifies to the user.

2. System ends use case.

Extension consisting in areas that can access any time appear at the end of the extension section

\* a. At anytime

8.a. There is a ~~draw~~ in the game. System checks that there is stalment

1. System notifies the user.

2. System ends the use case.

## 4. Exercise 2

From this use case and the lists of features/requirements, build the first version of domain model.

First we are going to start defining **list of requirements.**

1. One type of client, Player.
2. Two players play a game
3. A player must register and give basic info (email, password, name)
4. Two types of chess chips (16 black and 16 white)
5. A chess chip has to be moved between squares inside the board.
6. There are 32 chess chips distributed 6 types of chess chips: 2 Kings, 2 queen, 4 rooks, 4 knights, 4 bishops, 16 pawns.
7. Each type of chess chip contains the allowed movements and the colour.
8. For a player: invite another player
9. A board has 64 squares
10. A square has to provide basic info (position, chess chip that contains)
11. Store the information of the game (board, turn, time)
12. When the game is finished system has to propose to replay a game.

Once the list of requirements it is identified it is time to make the **textual analysis.**

1. **Take all the nouns that will be candidates for conceptual classes:**

* Player
* Game
* Info
* Email
* Password
* Name
* Chess Chip
* Square
* Board
* King
* Queen
* Rook
* Knight
* Bishop
* Pawn
* Movement
* Colour
* Position
* Turn
* Time

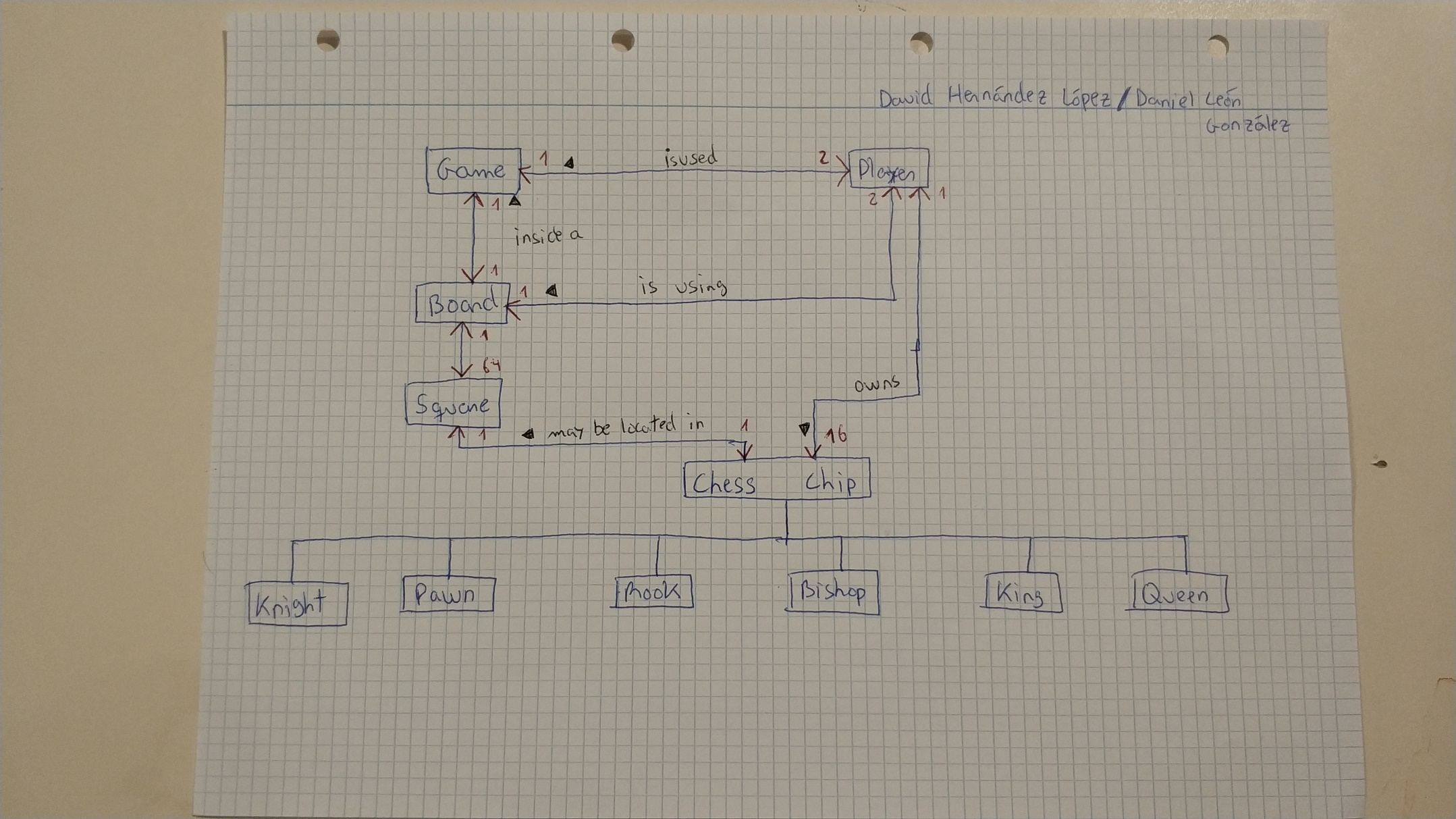
1. **DROP (Redundancy, vague concepts, names of attributes, name of operations, name of roles)**

* Info
* Email (Attribute)
* Password (Attribute)
* Name (Attribute)
* Movement
* Colour (Attribute)
* Position (Attribute)
* Turn (Attribute)
* Time

Now it is time to make the **relationships** between the kept classes which are: Player, Game, Square, Board, Chess Chip, King, Queen, Rook, Knight, Bishop, Pawn.

* A **square** is inside a **board.**
* A **board** is used in one **game.**
* A **chess chip** may be located in a **square.**
* A **player** owns **chess chips.**
* A **player** is in a **game.**
* A **player** is using a **board.**

**Chess chip** is a generalization. **Pawn, bishop, knight, rook, queen and king** are 6 types of **chess chips.** An object **pawn** is also an object **chess chip** as well as the rest. A **pawn** has the attributes that **chess chip** has and appear in **pawn**. Same for the rest of types of **chess chips.**

**Domain model scheme:**