COMP20050 - Software Engineering Project II

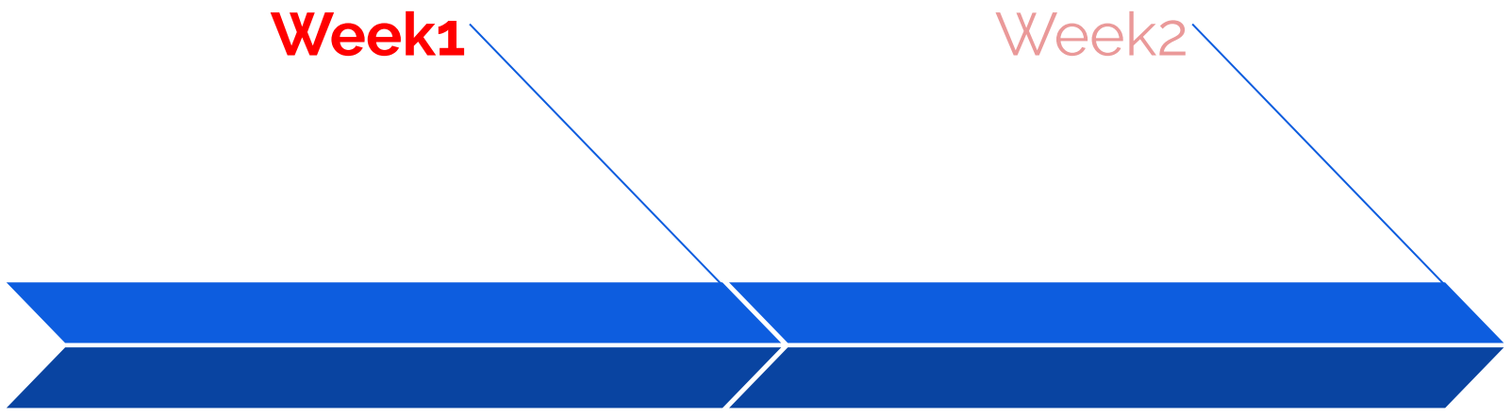
Software Engineering Project
(Hex Oust Board Game)

Ravi Reddy Manumachu
ravi.manumachu@ucd.ie

UCD School of Computer Science.

Scoil na Ríomheolaíochta
UCD.

COMP20050 - Weeks 1 & 2



Module Introduction

Software Project Specification

Version Control

Software Architectural Design

Outline (Learning Objectives)

- Understand the concepts of **user** and **system** requirements.
- Understand the differences between **functional** and **nonfunctional** software requirements.
- Understand the software requirements of your project HexOust.
- Become aware of what is software architectural design.

Software Engineering Project

- You are required to develop a software implementation of **HexOust (Base-7 hexagonal board)**.

- **HexOust** rules are in the locations below:

https://www.marksteeregames.com/Oust_rules.pdf
https://mindsports.nl/index.php/the-pit/614-hexoust

Project Handbook

- All you need to know about the project is provided in this handbook.

My Learning > Software Project Materials > Project Handbook
https://brightspace.ucd.ie/d2l/le/content/297464/viewContent/3490448/View

- It is the definitive reference.

Software Specification of HexOust

Requirements Engineering

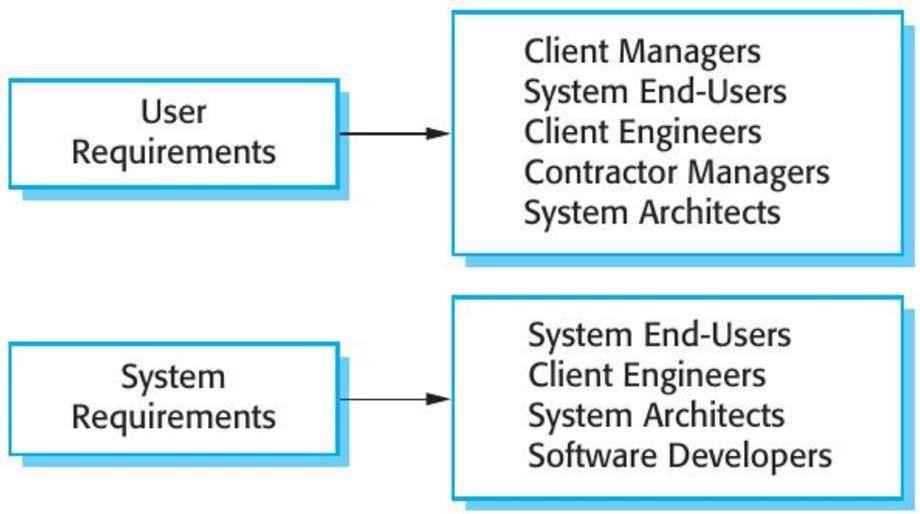
- The **requirements** for a system are the descriptions of what the system should do, the services that it provides and the constraints on its operation.

- The process of finding out, analyzing, documenting and checking these services and constraints is called **requirements engineering** (RE).

Two Description Levels: User and System

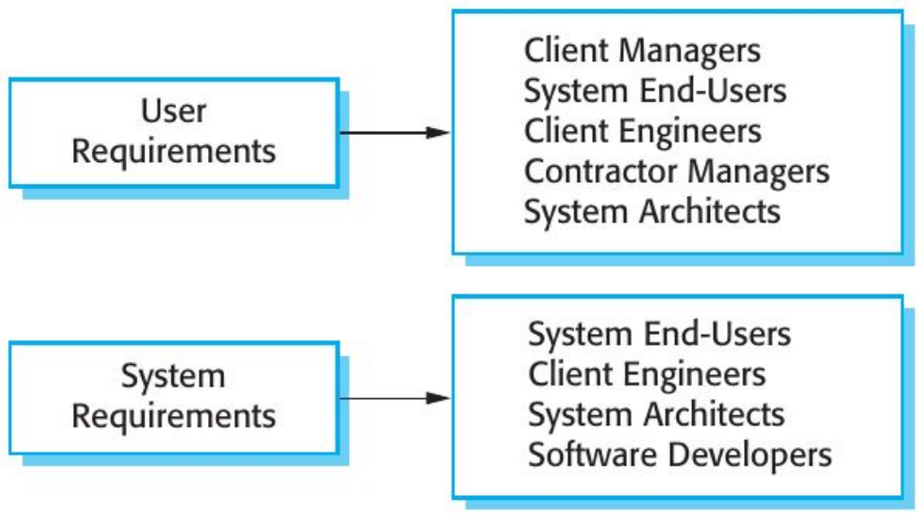
- **User requirements** are statements (natural language plus diagrams), of what services the system is expected to provide to system users and the constraints under which it must operate.

- The **user requirements** should specify only the external behavior of the system.



System Requirements

- **System requirements** are more detailed descriptions of the software system's functions, services, and operational constraints.
- System requirements are expanded versions of the user requirements that are used by software engineers as the starting point for the system design.
- All **system requirements** that define exactly what is to be implemented should go into a **functional specification**.



System Requirement Types: Functional

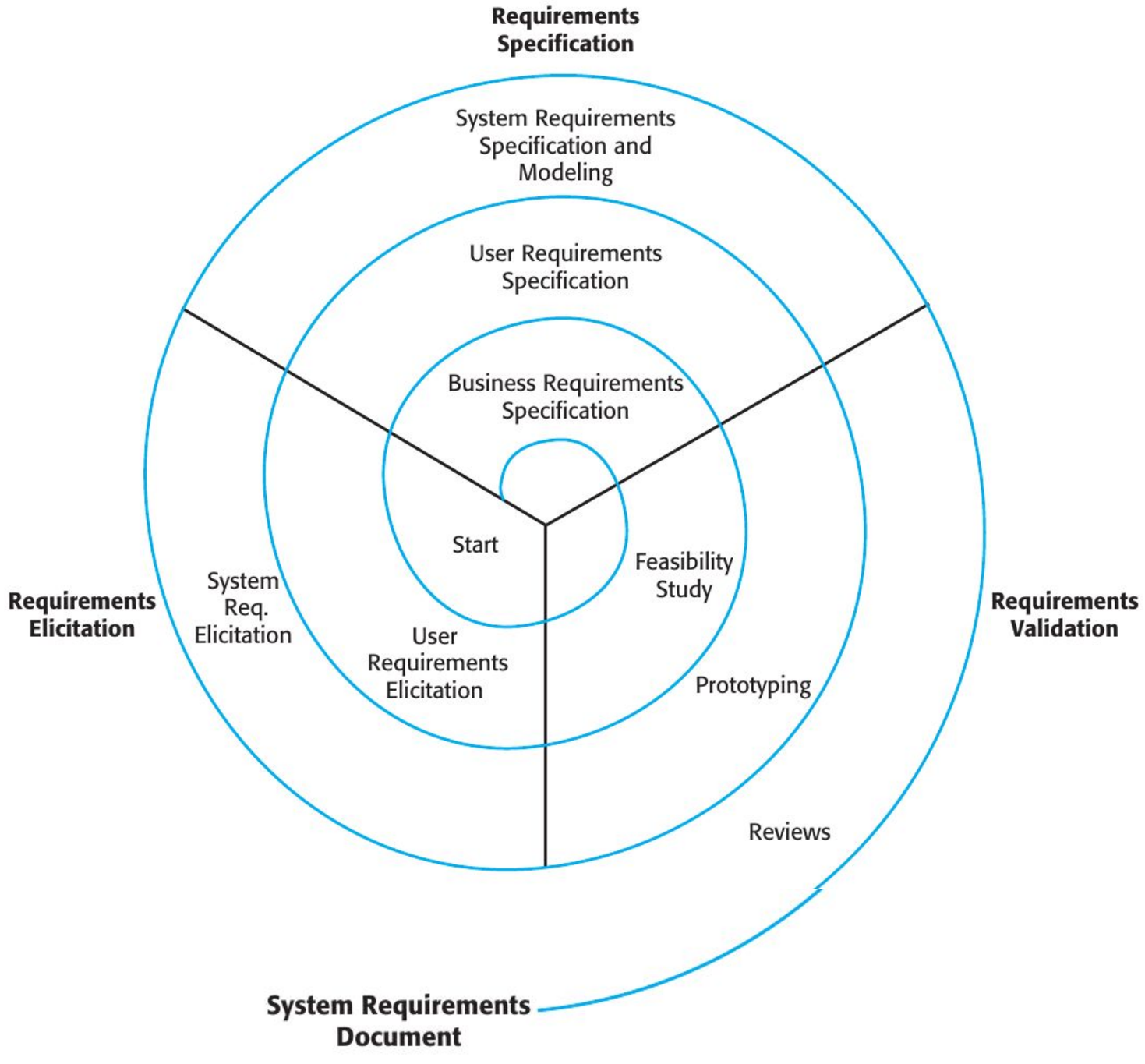
- **Functional requirements** for a system describe what the system should do.
- The requirements are statements of services the system should provide.
- The functional requirements specification of a system should be both **complete** and **consistent**.
  - **Completeness** means that all services required by the user should be defined.
  - **Consistency** means that requirements should not have contradictory definitions.

System Requirement Types: Non-functional

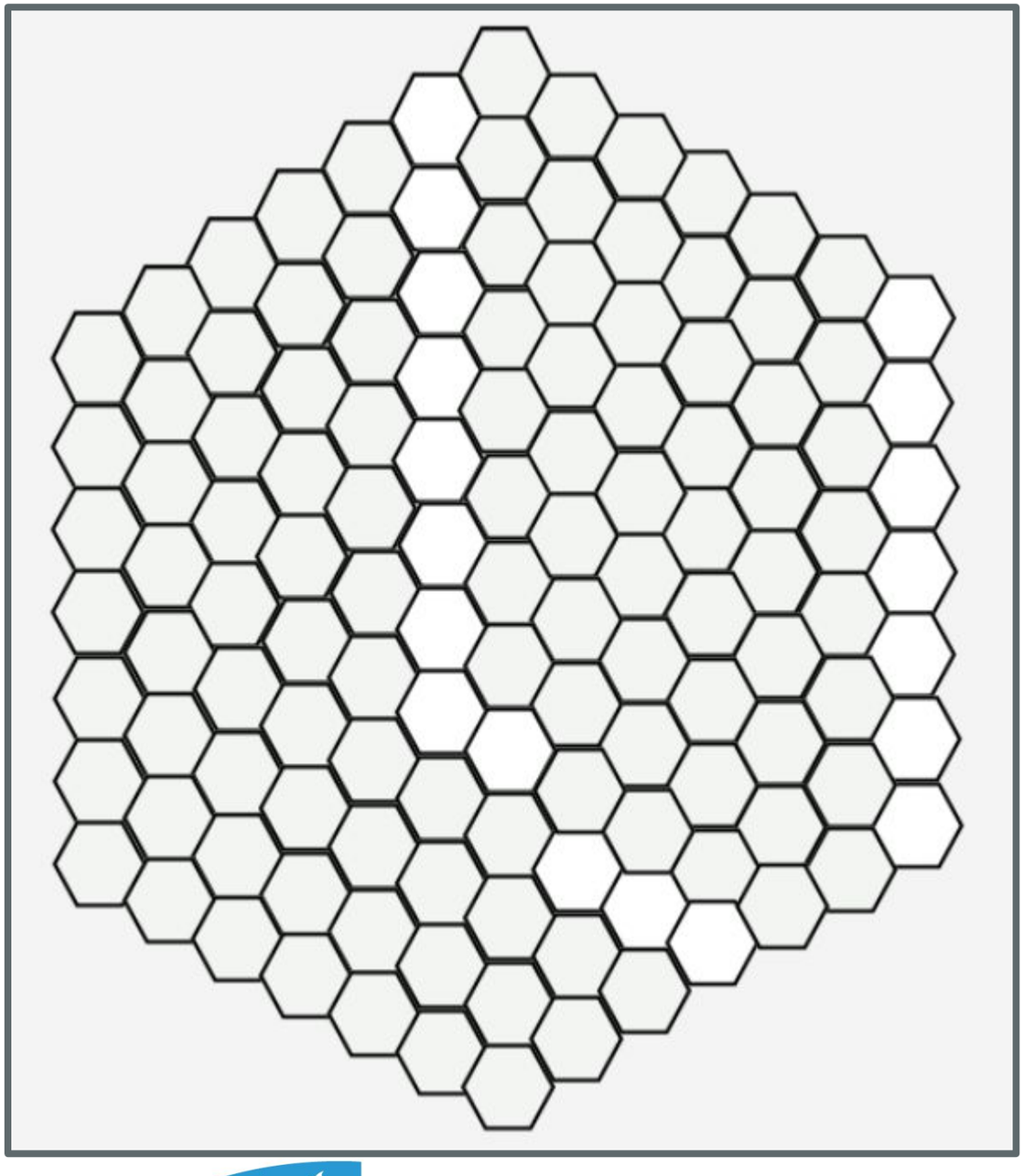
- **Non-functional requirements** are requirements that are not directly concerned with the specific services delivered by the system to its users.
  - Examples: Performance, security, or availability.
- **Non-functional requirements** are constraints on the services or functions offered by the system.
- Failing to meet a non-functional requirement can render the whole system unusable.
- Non-functional requirements may affect the overall architecture of a system rather than the individual components.

Requirements Engineering Process

- In practice, **requirements engineering** is a detailed iterative process with several high-level activities.

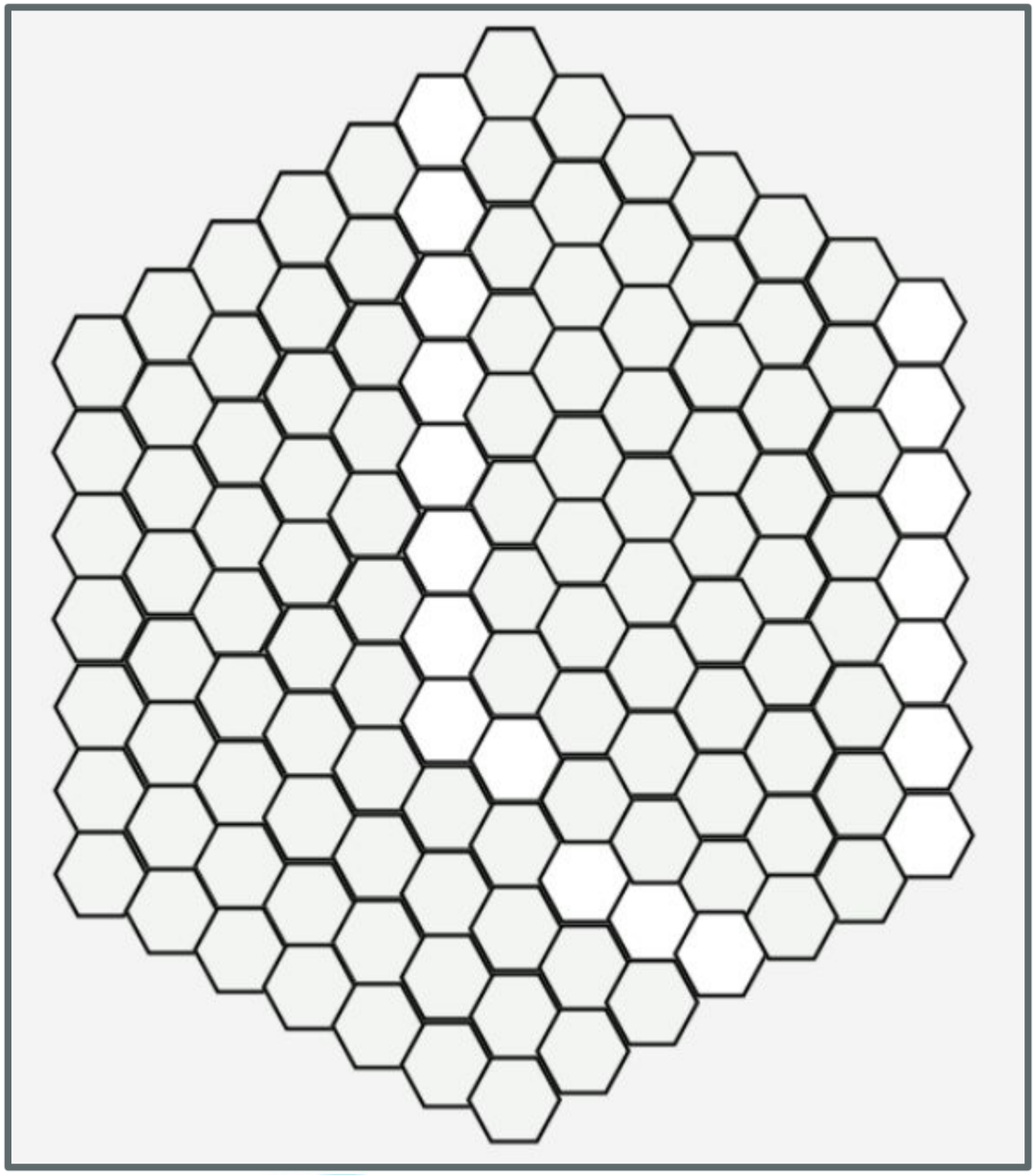


HexOust Brief (1/2)



- **HexOust** is a two-player game on a base-7 **hexagonal board**.
  - Seven hexagons on each side.
  - In total, there are 127 hexagons .
- The board starts out empty.
- The two players, **RED** and **BLUE**, take turns placing stones of their own color on **unoccupied** cells on the board.

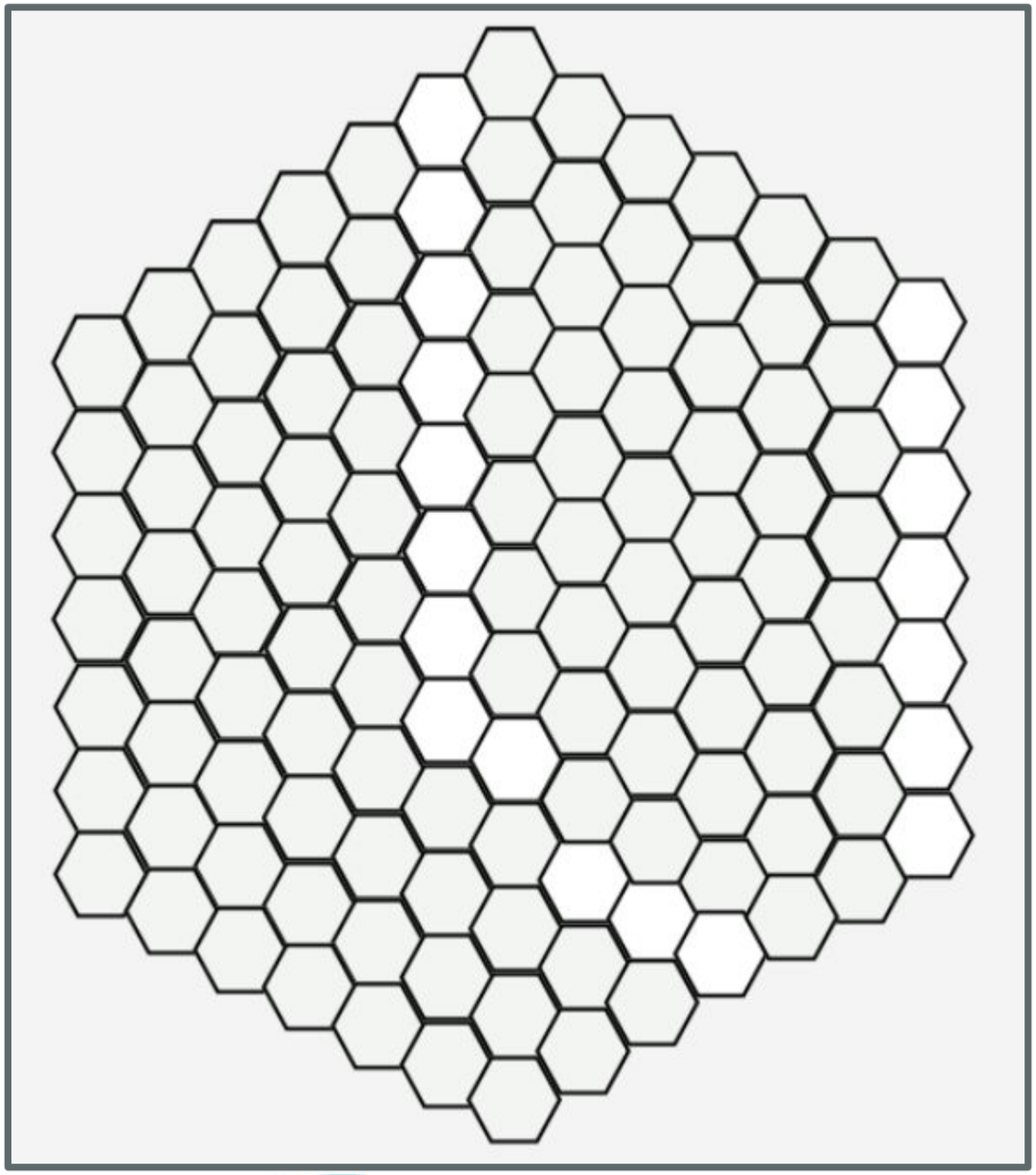
HexOust Brief (2/2)



- **RED** player moves first.
- The goal is to **oust** your opponent by completely clearing the board of her stones or capturing all of her stones.
- HexOust comprises what are known as **non-capturing** and **capturing** moves.
- Draws cannot occur in **HexOust**.

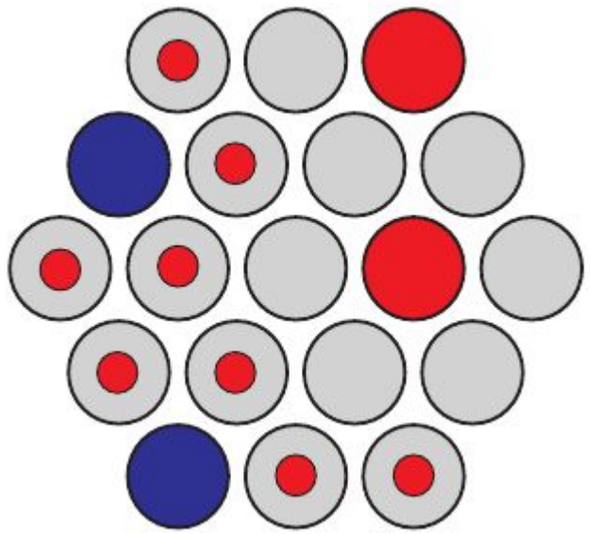
UCD DUBLIN

HexOust: Glossary



- **HexOust** software system will be abbreviated as **HOS**.
- The two players will be called **RED** and **BLUE**.
- A **GROUP** is a set of interconnected like-colored stones.
- **NCP** and **CP** are non-capturing and capturing placements.

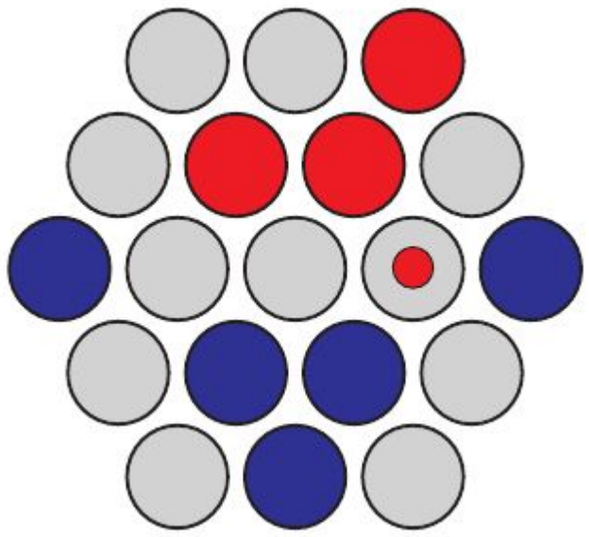
HexOust Rules: Non-capturing Placements



RED dots represent the only placements available to the RED player.

- *I will illustrate the rules using a simple circular board.*
- A **non-capturing placement** either forms no connections (adjacencies) with any stones.
- **OR** forms one or more connections only with enemy stones.
- A **non-capturing placement** does not form any connections with stones of its own color.
- A player concludes her turn by making a **non-capturing placement**.

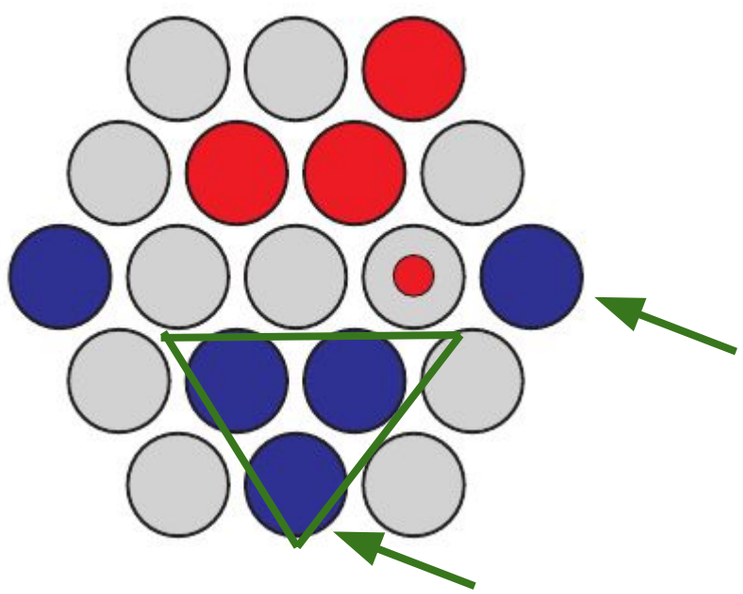
HexOust Rules: Capturing Placements



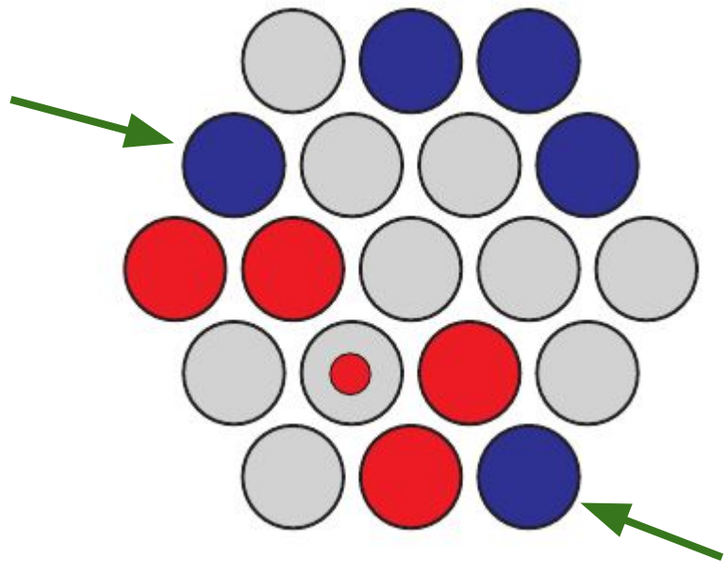
RED placement (the dotted cell) captures two BLUE groups.

- When a player places a stone which forms one or more connections with the player's own groups, the player creates a **new, larger group** of her own stones.
- A player can only make such a placement if the **new group** will have one or more connections with the **opponent's groups** upon its creation.
- **AND** if all **opponent's groups** are **smaller** than the new group.
- Upon making such a placement, all **opponent's groups** with connections with the placed stone are removed from the board.

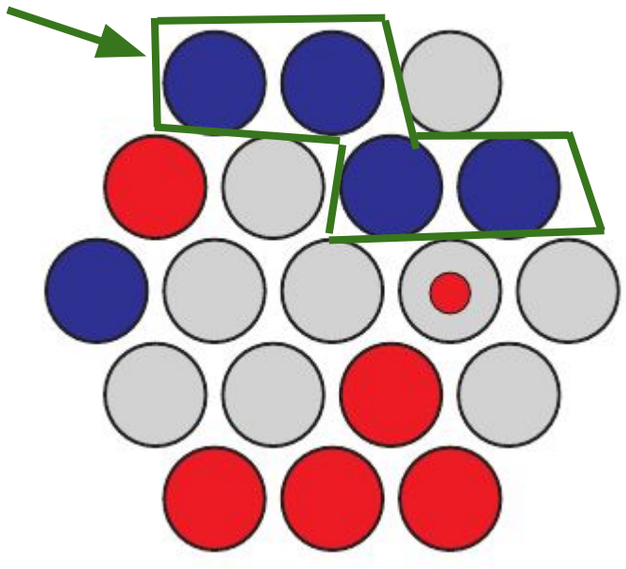
HexOust Rules: Capturing Placements



RED placement (the dotted cell) captures two **BLUE** groups.



RED placement (the dotted cell) captures two **BLUE** singleton groups.



RED placement (the dotted cell) captures **BLUE's** group of four.

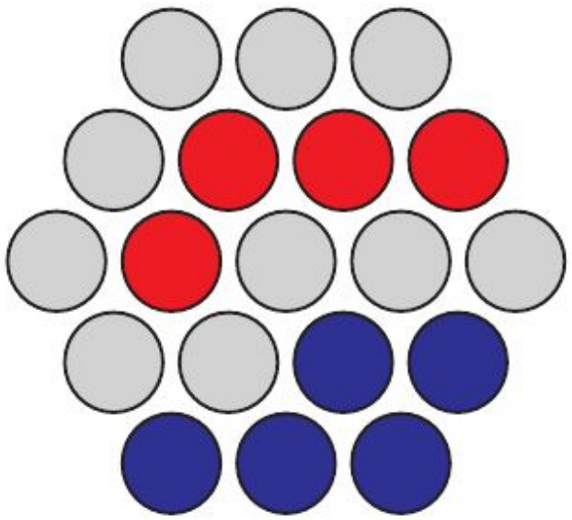
HexOust Rules: Multiple Placements Per Turn

- A player will continue to add stones after capturing one or more opponent's groups until the player makes a non-capturing placement.

- Making a non-capturing placement concludes the turn of a player.

- Therefore, a player can make multiple placements per turn and can potentially clean up all the opponent's groups.

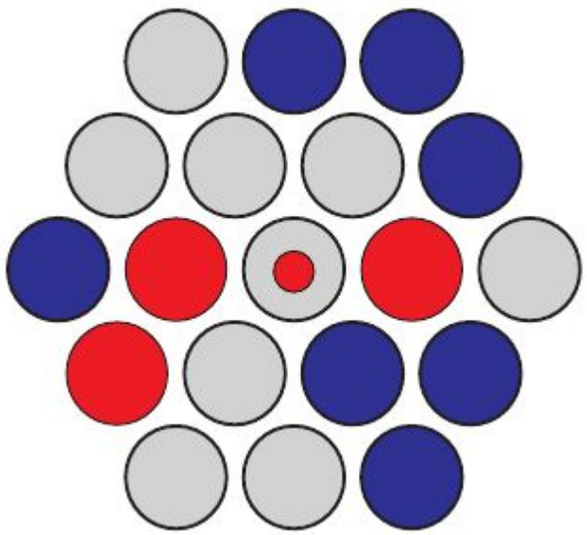
HexOust Rules: Making a Placement



RED player will pass her turn.

- If the player has a placement available on her turn, then she must make the placement.
- If the player has no placements available, then the player must pass her turn.
- There will always be a placement available to at least one of the two players.
- In the figure, **RED** player has no placements available and so must pass.

HexOust Rules: Winning Move

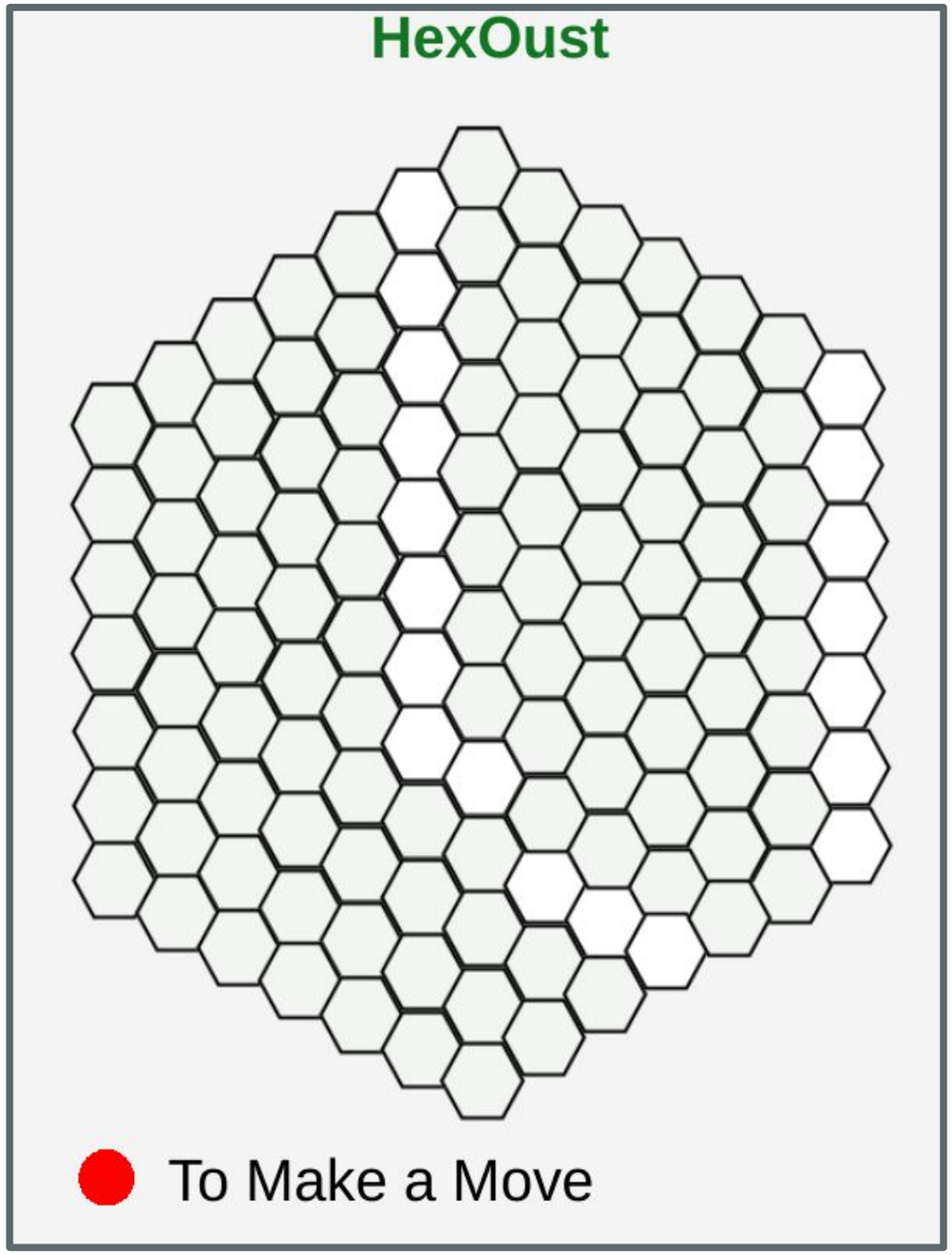


RED placement (the dotted cell) captures all the BLUE's groups.

- A player wins by making a placement which captures all of the enemy stones on the board.
- In the figure, **RED** captures all of the **BLUE**'s stones and wins the game.
- Group of 4 **RED** stones is bigger than the three **BLUE** groups. Therefore, the move captures all the **BLUE** stones.

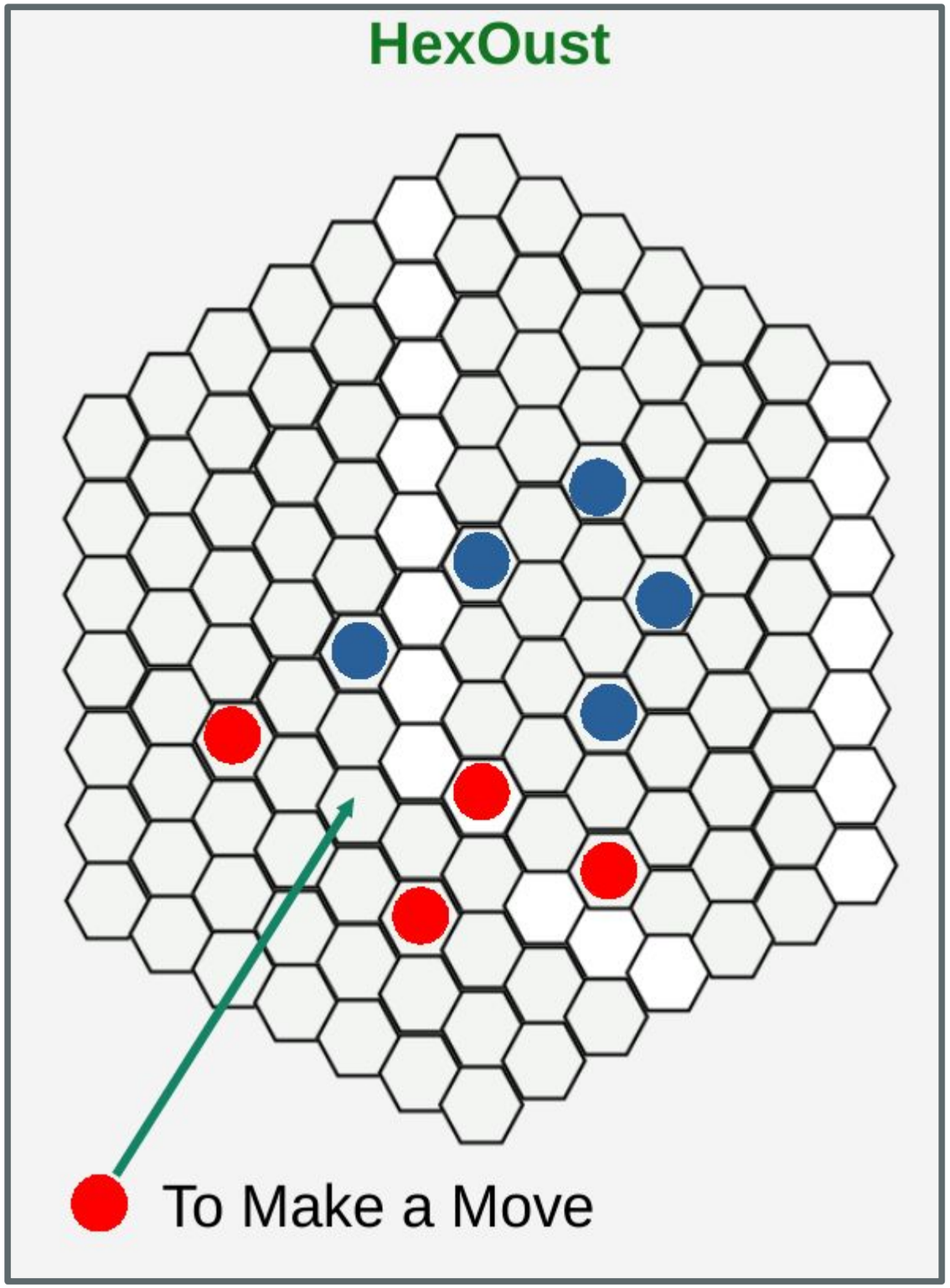
HexOust Requirements Specification (SRS)

- **SR1:** On launch of HOS, an empty base-7 hexagonal board shall be displayed with a **RED** sphere indicating the **RED** player's turn.
- **SR1.1:** **RED** player's stones shall be represented by 2D **RED** spheres.
- **SR1.2:** **BLUE** player's stones shall be represented by 2D **BLUE** spheres.
- **SR1.3:** A **RED** sphere followed by a text shall be displayed (*not necessarily below the board*) to indicate the **RED** player's turn.

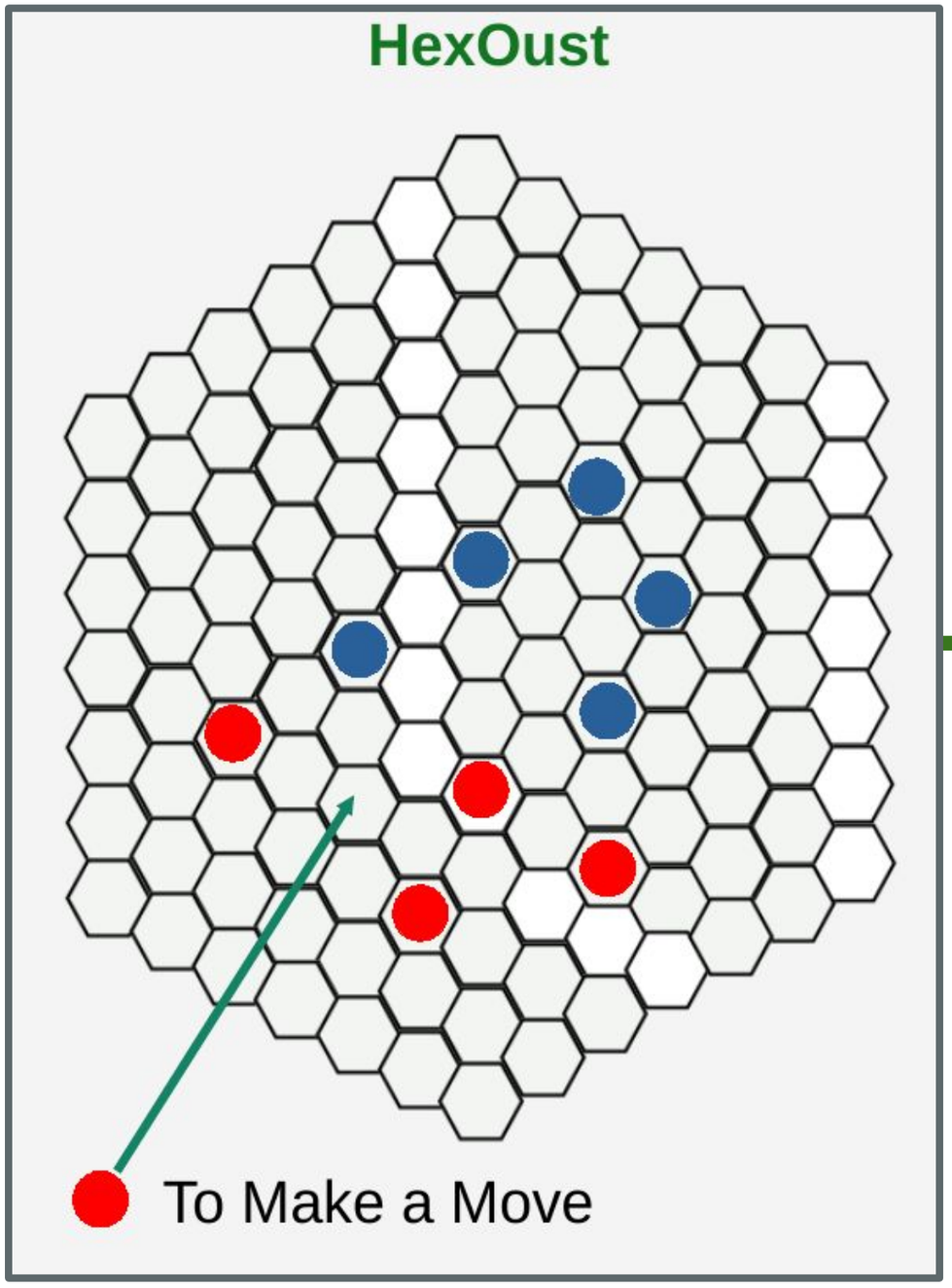


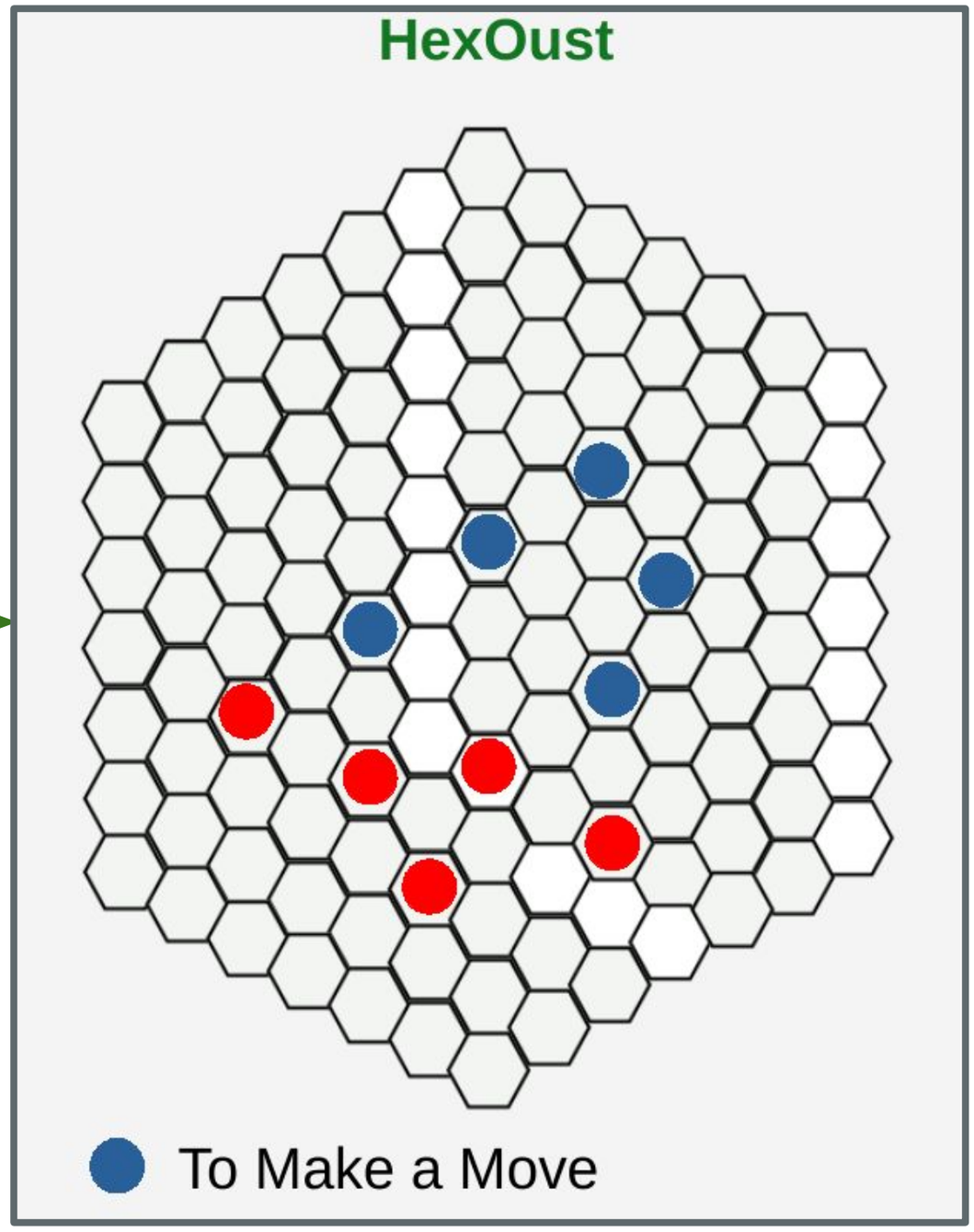
HexOust SRS 2 (Non-capturing Placements)

- **SR2:** A player (**RED** or **BLUE**) should place her stone in a valid cell by drag-and-drop or clicking the cell.
- **SR2.1:** If the player made an NCP, the stone shall be placed in the valid cell, and
- **SR2.2:** If the player made an NCP, a stone of the opponent player's color followed by a text shall be displayed to indicate the opponent player's turn.



HexOust SRS 2 (Example)

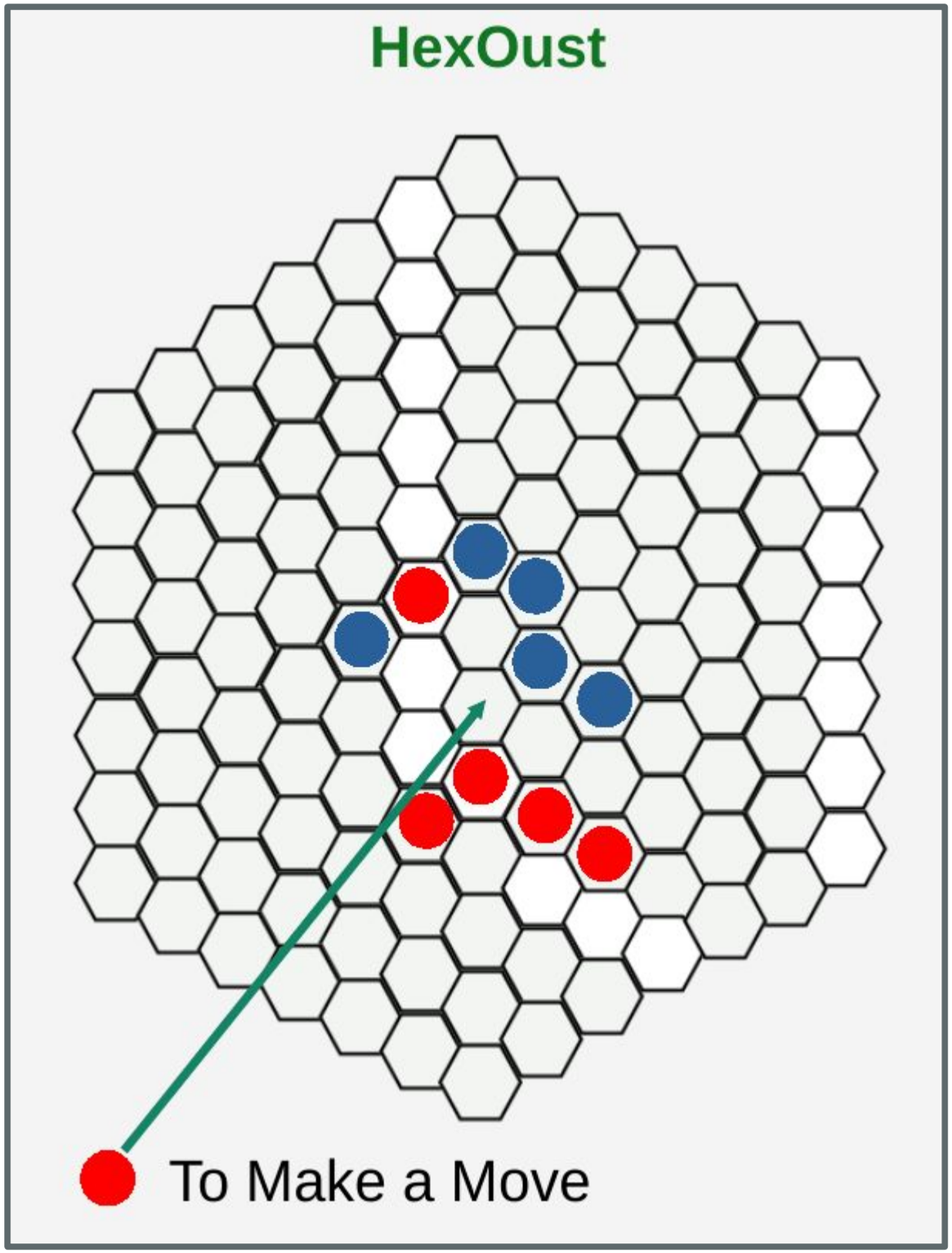




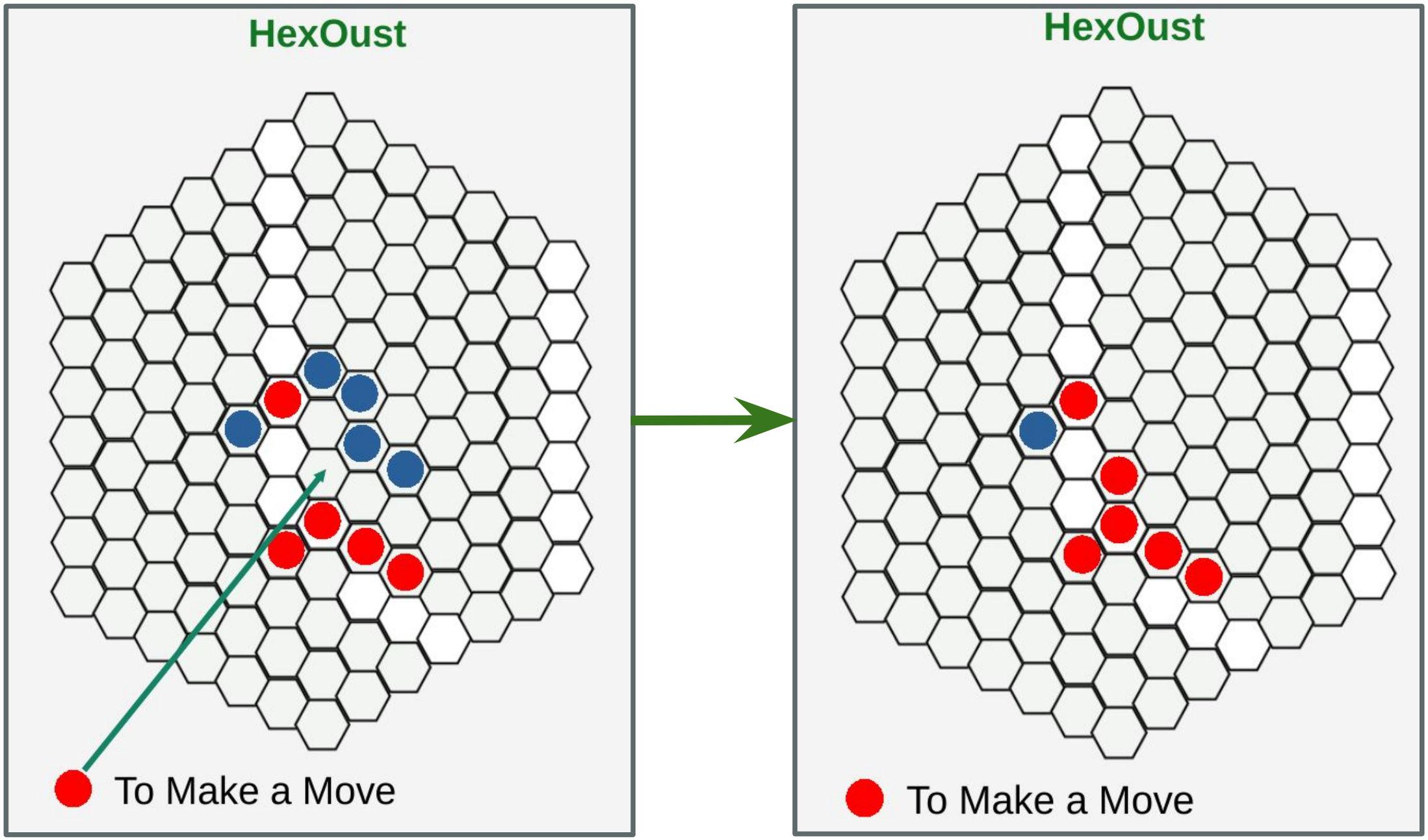
HexOust SRS 3 (Capturing Placements)

- **SR3:** A player (**RED** or **BLUE**) should place her stone in a valid cell by drag-and-drop or clicking the cell.
- **SR3.1:** If the player made a CP, then all the opponent's groups that are captured shall be removed from the board.
- **SR3.2:** A stone of the capturing player's color followed by a text shall be displayed to indicate the capturing player's turn.

In reality, this requirement can be further broken down into several interesting CP scenarios.

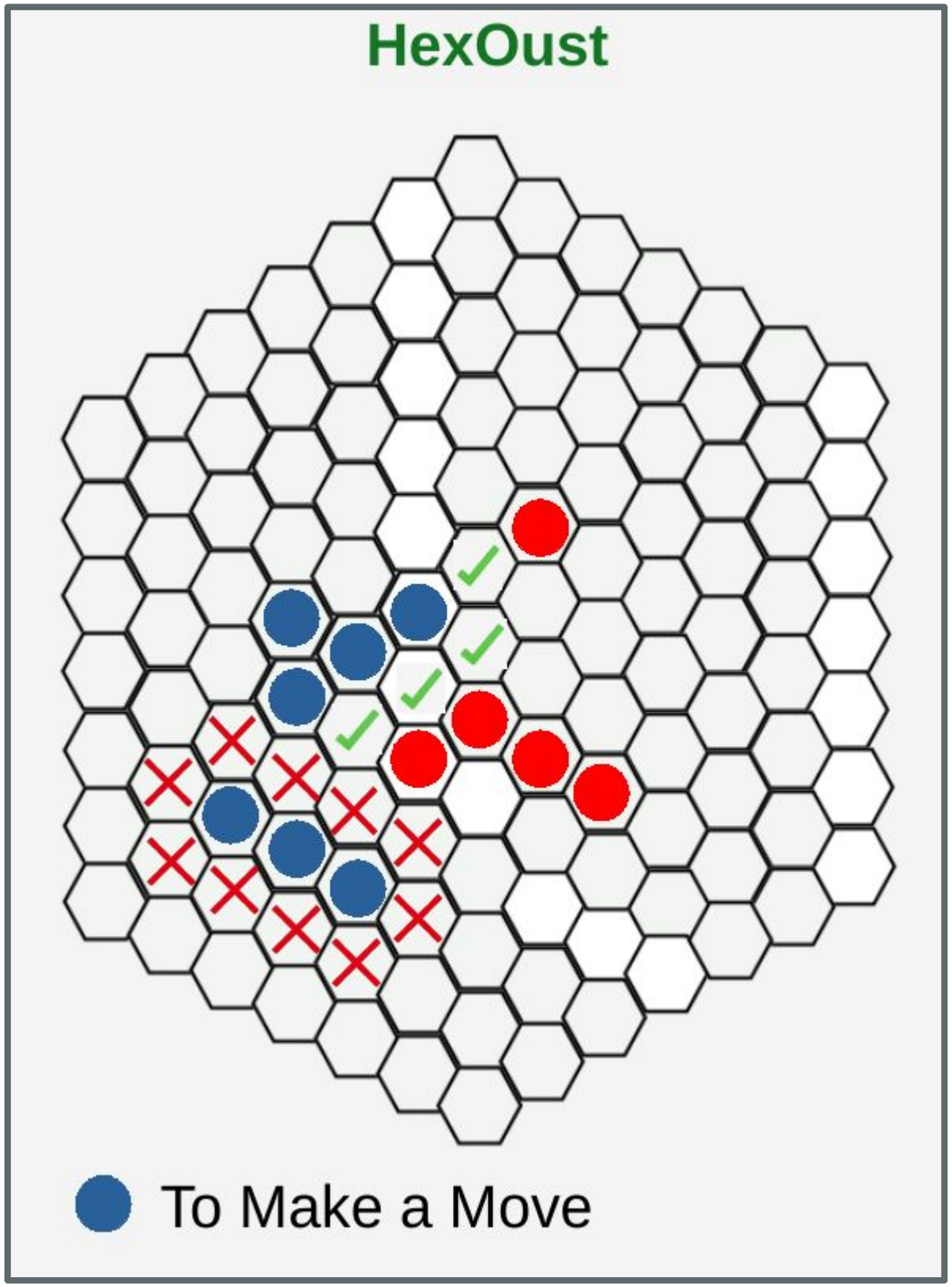


HexOust SRS 3 (Example)



HexOust SRS 4 (Invalid Placements)

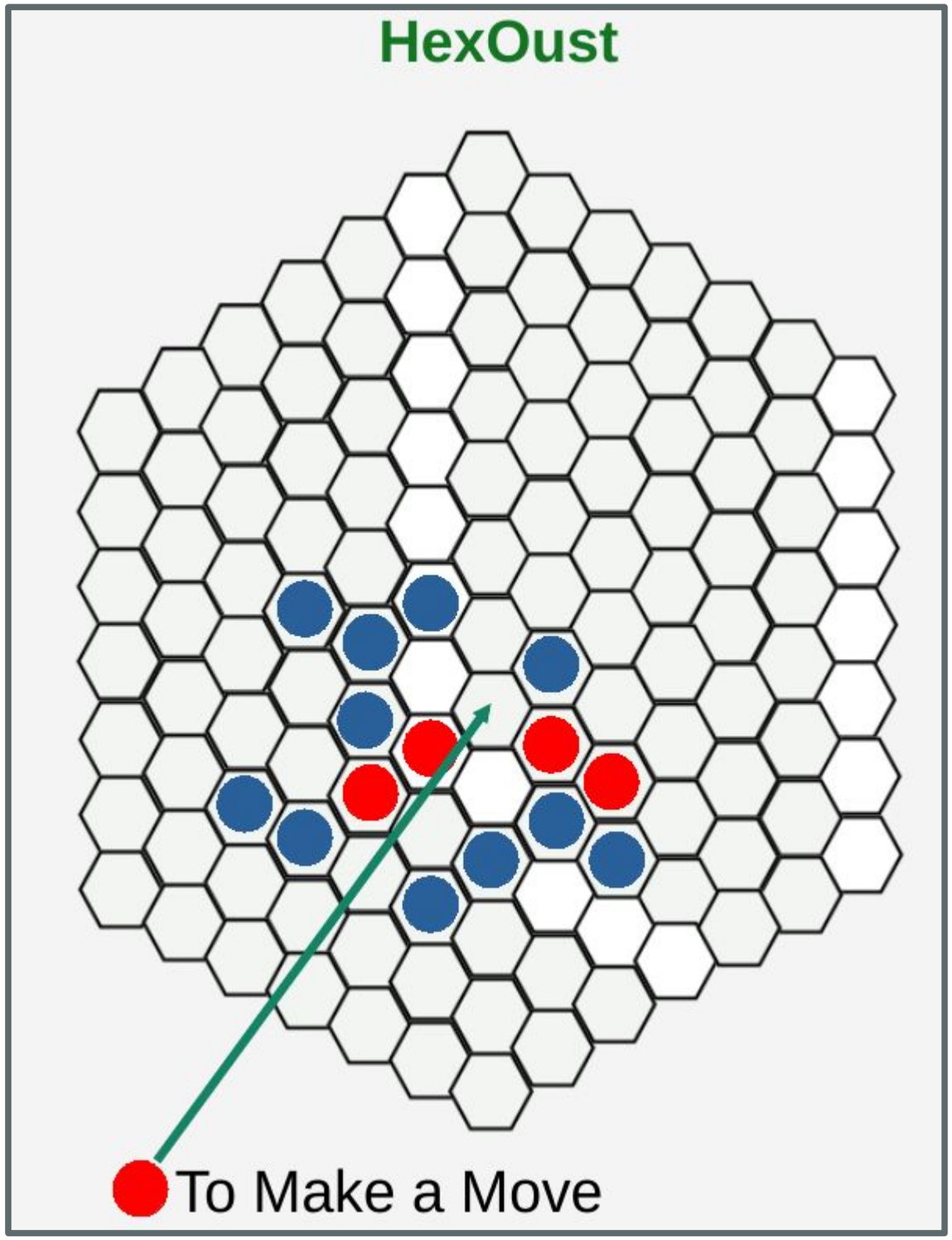
- **SR4:** If a player (**RED** or **BLUE**) places her stone in an invalid cell, then an **error message shall be displayed.**
- **SR4.1:** The error message shall be **Invalid Cell Placement**.
- *An enhanced requirement (not mandatory to implement) follows:*
- **SR4.E1:** If the player hovers her stone over a **valid cell**, then the cell should display a **green tick symbol** to indicate that the player can place a stone there.
- **SR4.E2:** Similarly, for an **invalid cell**, a red cross should be displayed.



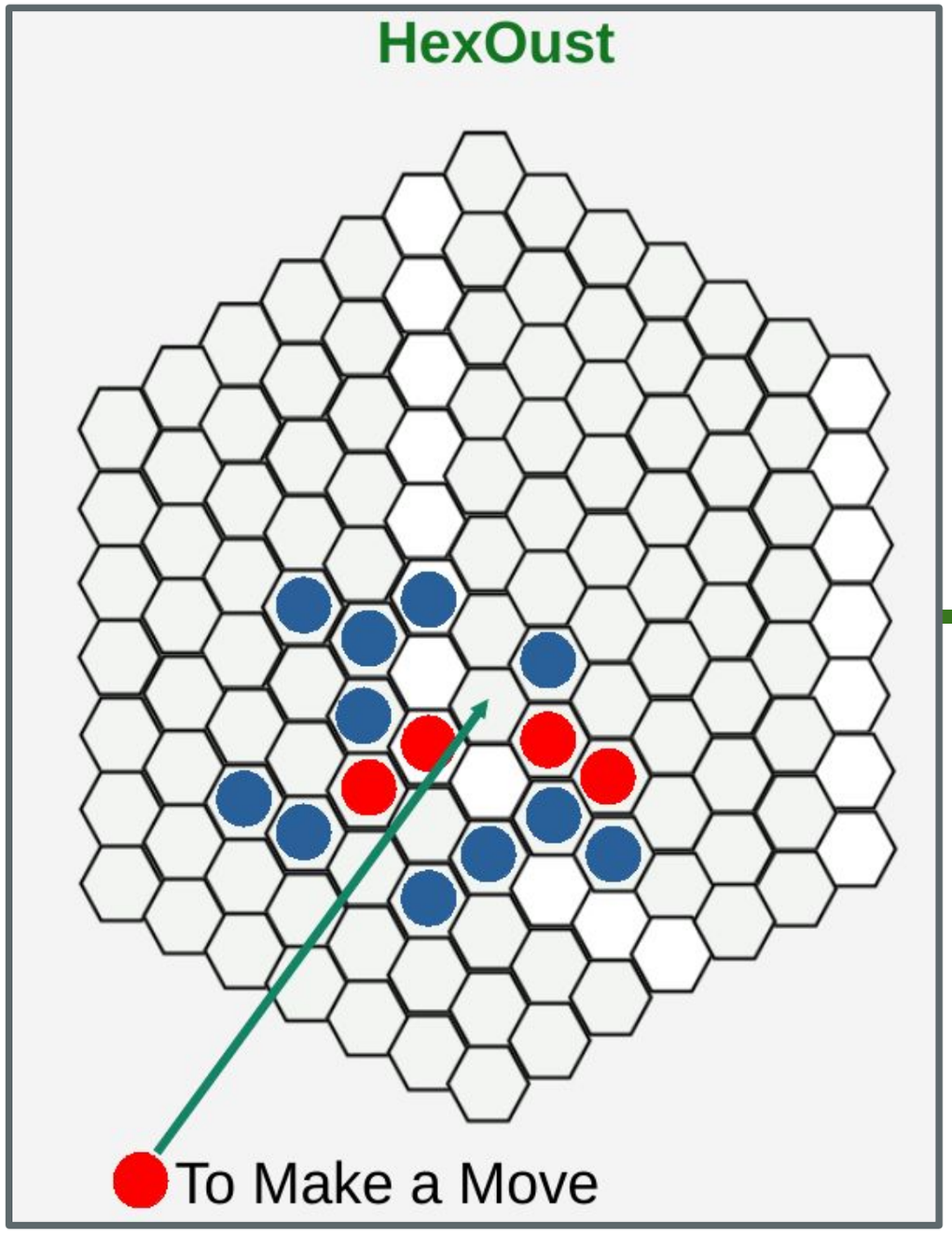
HexOust SRS 5 (Winning Move)

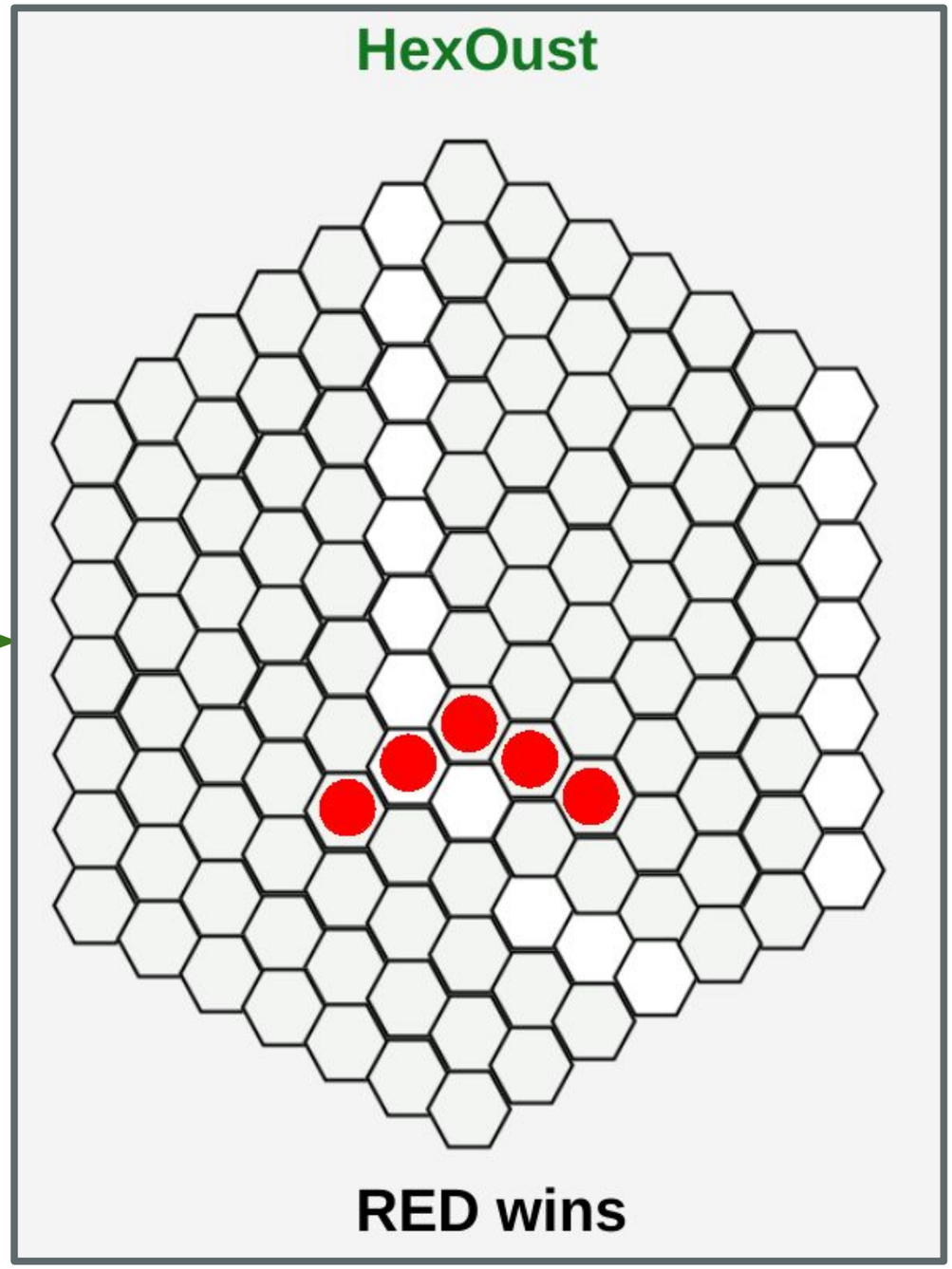
- **SR5:** If a player (**RED** or **BLUE**) places her stone that eliminates all the opponent's groups, then a message shall be displayed mentioning that the player won (**RED** or **BLUE**).

- **SR5.1:** The messages shall be
  RED wins
  BLUE wins

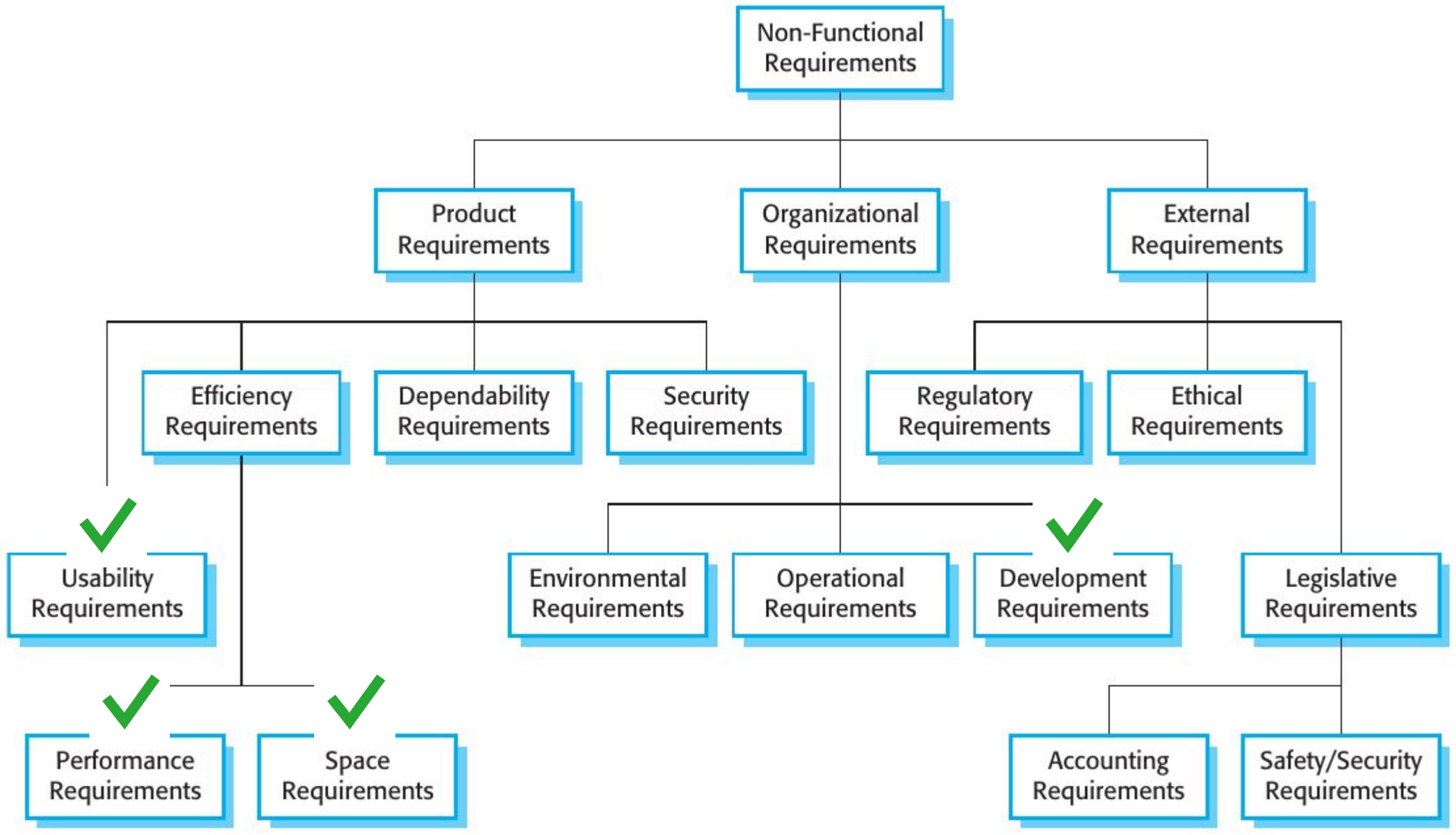


HexOust SRS 5 (Winning Move Example)





Non-Functional Requirements for HexOust



Green ticks ✓ are mandatory for your project.

Non-functional Requirements for HexOust

- **Performance requirements**:
  - The event response time should not be more than **5 seconds**. An event is a move by a player.
- **Space requirements**:
  - The final release package for HexOust should not be more than **10 MB**.
- **Development requirements**:
  - The system must be implemented using **Java programming language**.
- **Usability requirements**:
  - Experienced players should not make more than **3 errors** on average during one game play.

Non-functional Requirements for HexOust

- **Performance requirements**:
  - The event response time should not be more than **5 seconds**. An event is a move by a player.
- **Space requirements**:
  - The final release package for HexOust should not be more than **10 MB**.
- **Development requirements**:
  - The system must be implemented using **Java programming language**.
- **Usability requirements**:
  - Experienced players should not make more than **3 errors** on average during one game play.

Software Architectural Design

Architectural Design Using PBL (1/3)

- Your first task is to work on a challenge using **problem-based learning (PBL).**

- The challenge is **software architectural design** of your **software project**.

- You (in a **group of three**) will research and gather information related to designing high-level software architecture of your project.

Software Design Using PBL (2/3)

- While addressing the challenge, you will seek answers to the following questions.
  - **What is software architectural design?**
  - **What are different software architecture styles and characteristics?**
  - **What are the software architecture patterns best suited for this project?**
  - **What are the best tools for software architectural design?**

- Once you have the high-level design completed, then you can start thinking about low-level design and implementation.

- **Your software design work will be vital to formulating your project plan and software implementation.**

Group Roles in PBL (3/3)

- **Chairperson:**
  - Encourages the participation of all team members.
  - Facilitates the team to work within agreed ground rules.

- **Scribe:**
  - Summarizes and synthesizes the ideas and learning in the team.

- **Timekeeper:**
  - Helps the team to manage their time.

Role of Tutors

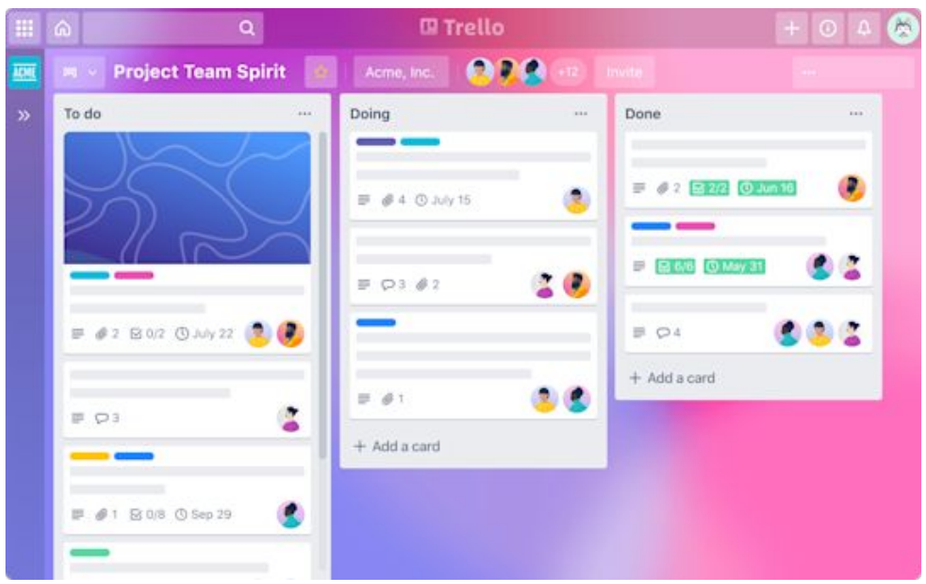
- The lecturer, TA, and demonstrators will be tutors in this module.

- TA and demonstrators will facilitate the PBL process in the labs.
  - They ask questions that encourage critical thinking in the labs.
  - Encourage student to link theory and practice.
  - Facilitate students to reflect on their learning, the development of key skills and the performance of the team.

Software Project Management Details

Project Management Details

- You must use **GitHub** for collaborative project work.
- Check if **GitHub** allows you to track your tasks. If No, you can use **Trello** for tracking project activities using Kanban Board.
- You can use **Slack** for communications on project work. However, it is not mandatory.
- **NOTE: Software version control lecture covered next week.**

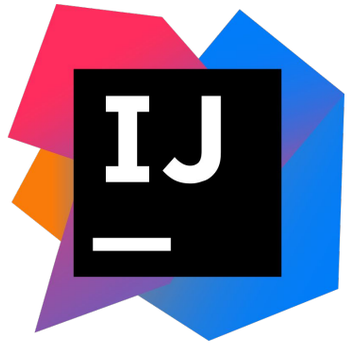


Software Implementation Details

- You will use **Java Programming Language**.

- No restrictions on the **development environment**.
  - Windows, Unix, Linux, macOS.
  - Eclipse IDE, IntelliJ IDEA.

- No restrictions on **User Interface (UI)**.
  - Graphical User Interface (GUI) using **JavaFX** or **LibGDX**.
  - Text-based User Interface (TUI).
  - **NOTE: JavaFX and LibGDX introductory lectures are given in Weeks 4-6.**



First Set of Deliverables (1/2)

- You will propose a **software architectural design.**
  - Low-level design containing classes and algorithms highly recommended.

- You are allowed to use any modelling tool (like UML) to show the external, interaction, structural, and behavioral perspectives of the software system.

- See project handbook for submission instructions.

- **Deadline: 10 February**

First Set of Deliverables (2/2)

- You will use your **software design** to formulate a **project plan** that breaks the development of the project into **four sprints.**

- See project handbook for a sample project plan.

- In each sprint, you will deliver 3 or more features that meet the SRS requirements.

- See project handbook for submission instructions.

- **Deadline: 10 February**

Q&A

To follow...

Software Version Control
Software Architectural Design