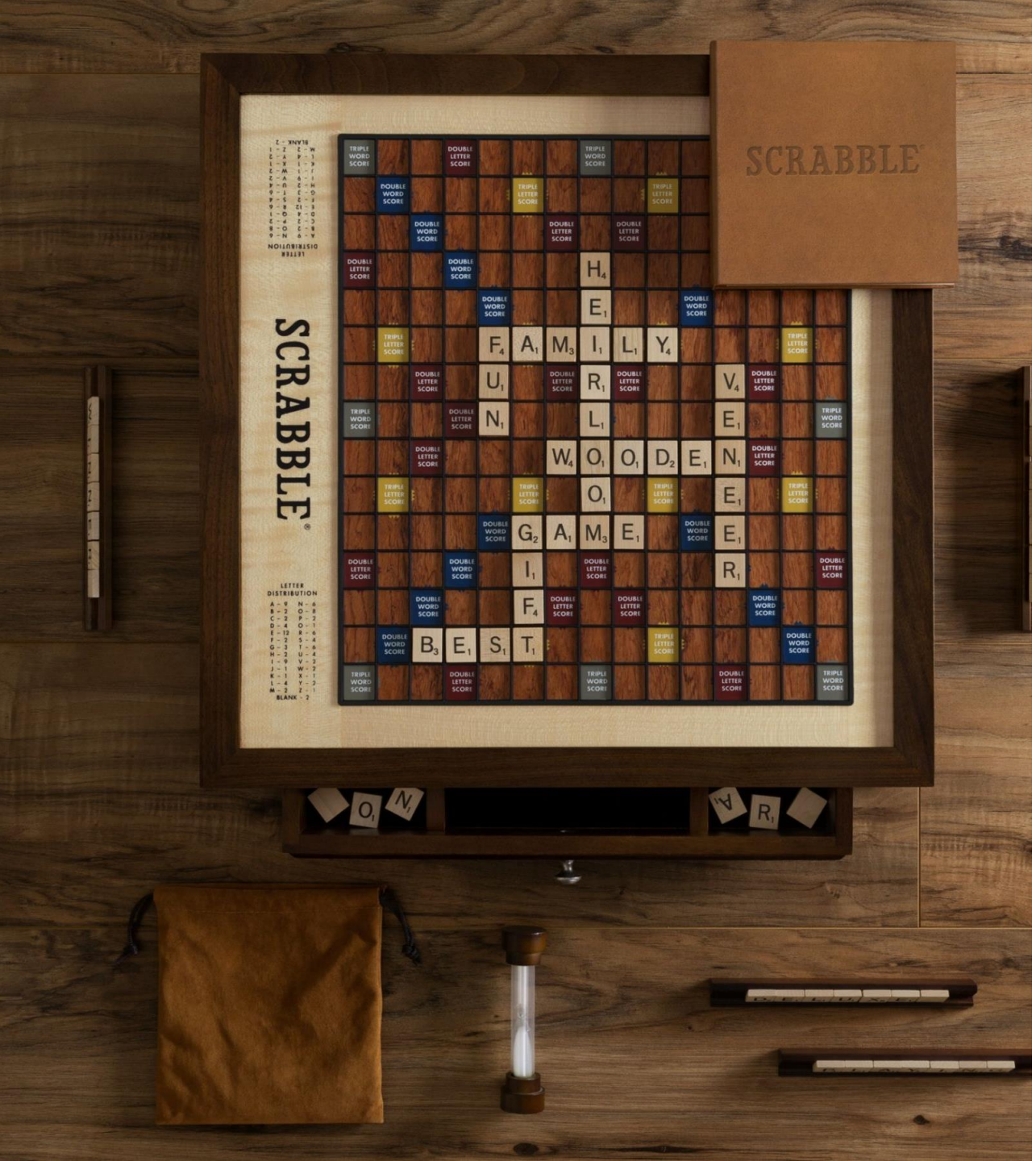
ALPHABET TILES

Presented by:

Batoul Haidar

CONTENTS

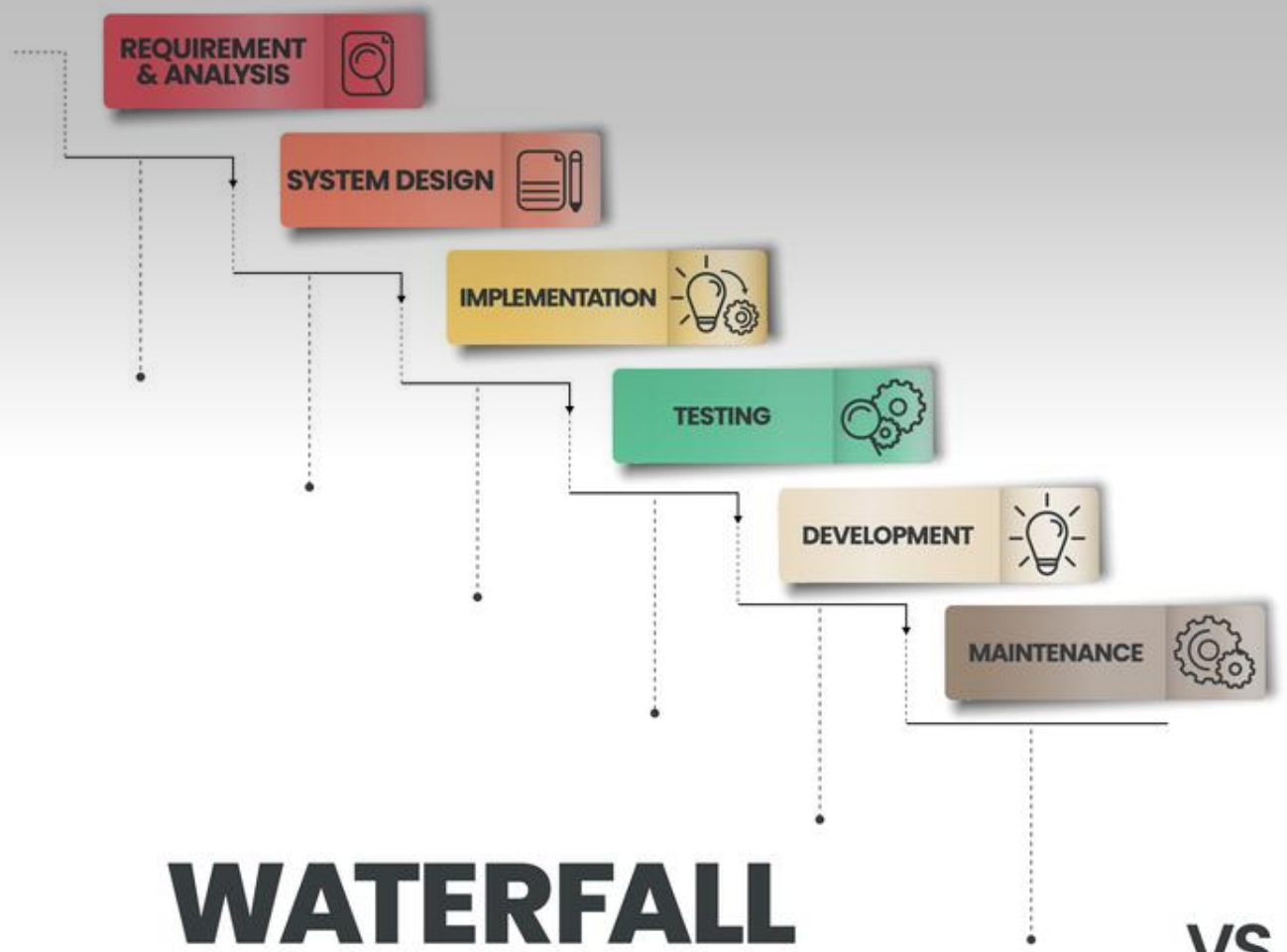


WELCOME TO ALPHABET TILES: A SCRABBLE-INSPIRED ADVENTURE.

- The main objective is to score the highest points by forming words on the game board using letter tiles. Words can be formed horizontally or vertically but not diagonally.

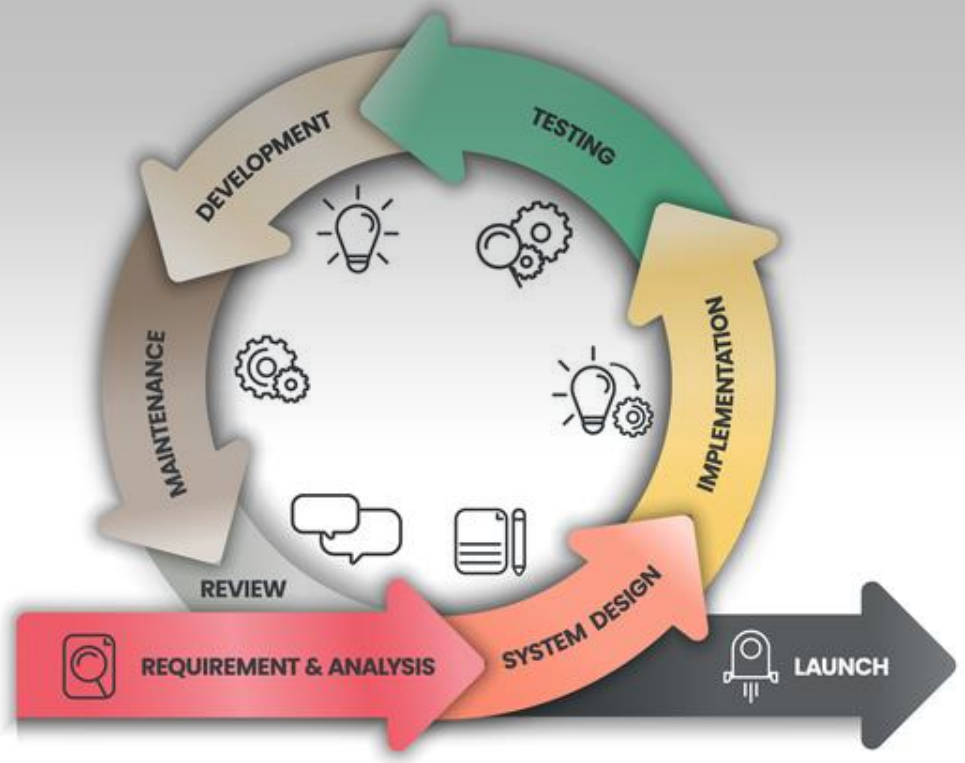
- It offers a unique blend of education and entertainment, effective tool for language learning, helping users enhance their vocabulary.

EVOLUTION OF DEVELOPMENT METHODOLOGIES: FROM WATERFALL TO AGILE

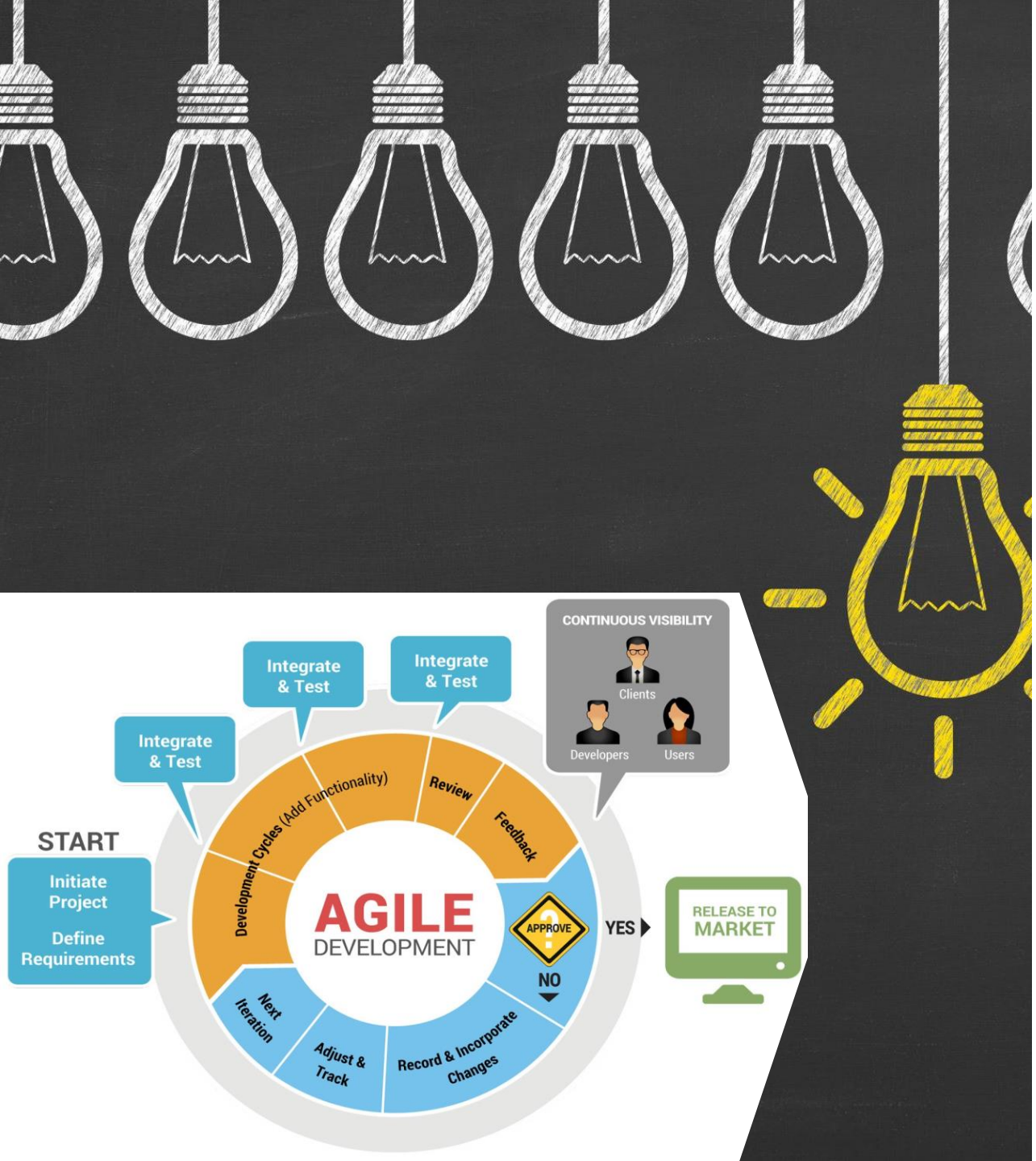


WATERFALL
MODEL

VS

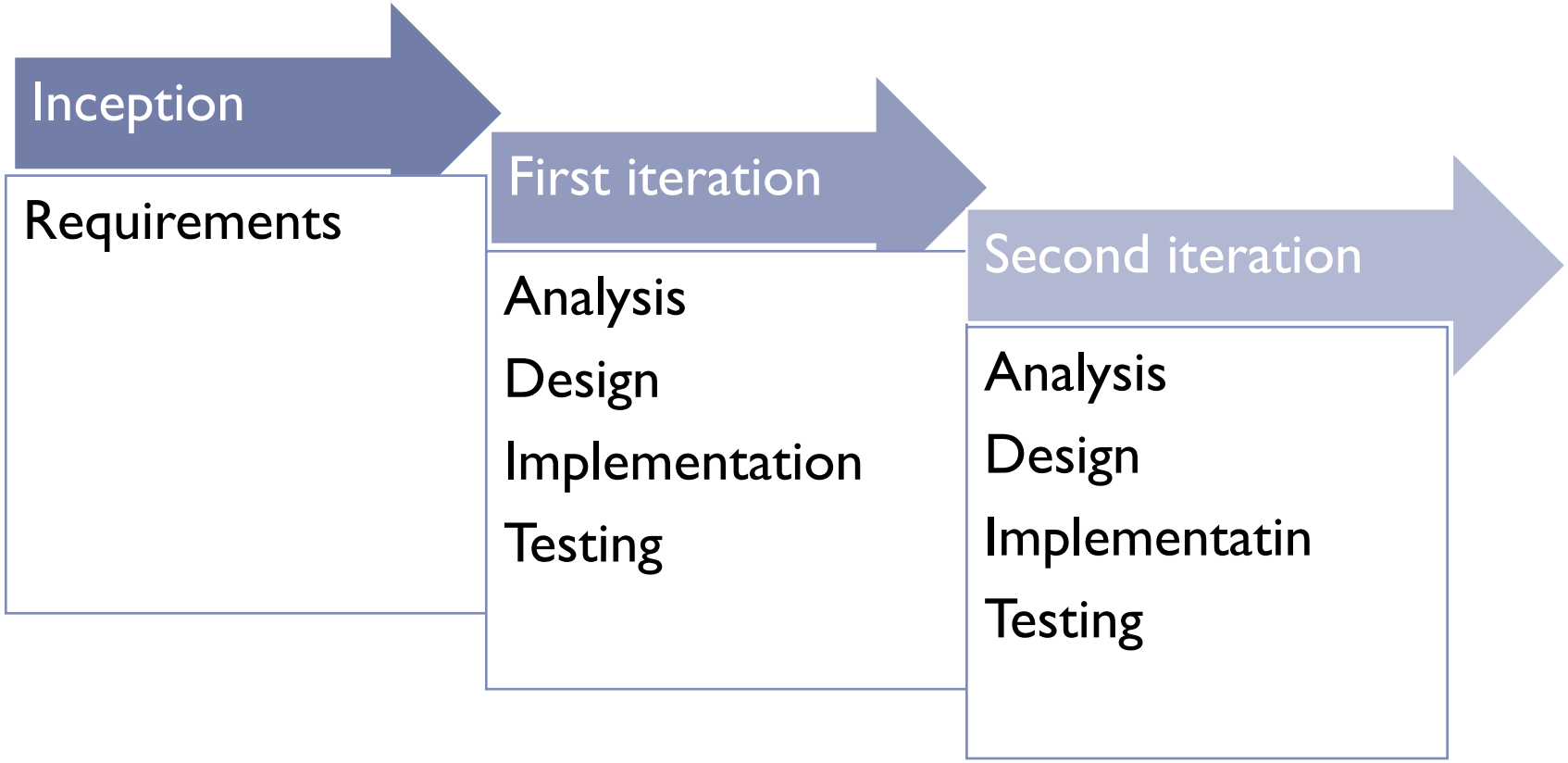


AGILE
MODEL



EMBRACING AGILE: ITERATIVE DEVELOPMENT APPROACH.

DEVELOPMENT JOURNEY: FROM INCEPTION TO ITERATIONS.



INCEPTION PHASE: LAYING THE FOUNDATION.

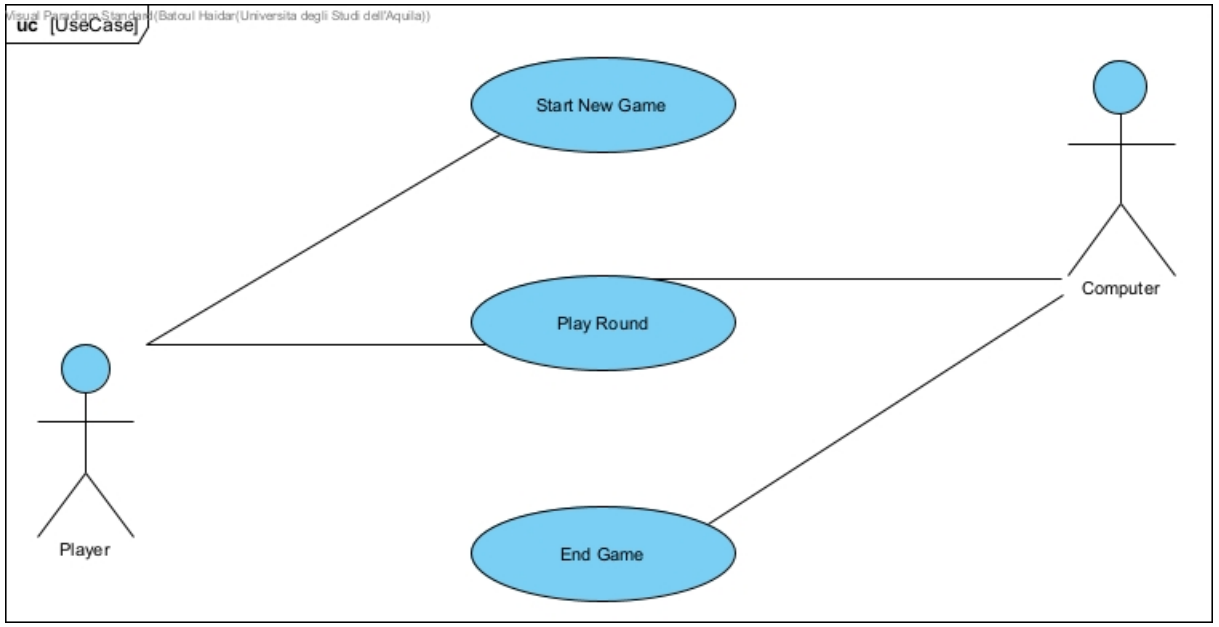
Vision : collect, analyze, and define high-level needs and features of the Alphabet Tiles. It focuses on the capabilities needed by the stakeholders and the target users, and why these needs exist. “it guides and inspire who working on the project”.

Glossary : The purpose of Business Glossary is to establish a clear and consistent understanding of specific terms and concepts used within the "Alphabet Tiles" project.

Supplementary specifications : The purpose of Supplementary Specification is to outline and detail the system requirements for the "Alphabet Tiles" project that are not explicitly addressed in the use-case model.

Business Rules: define all steps.

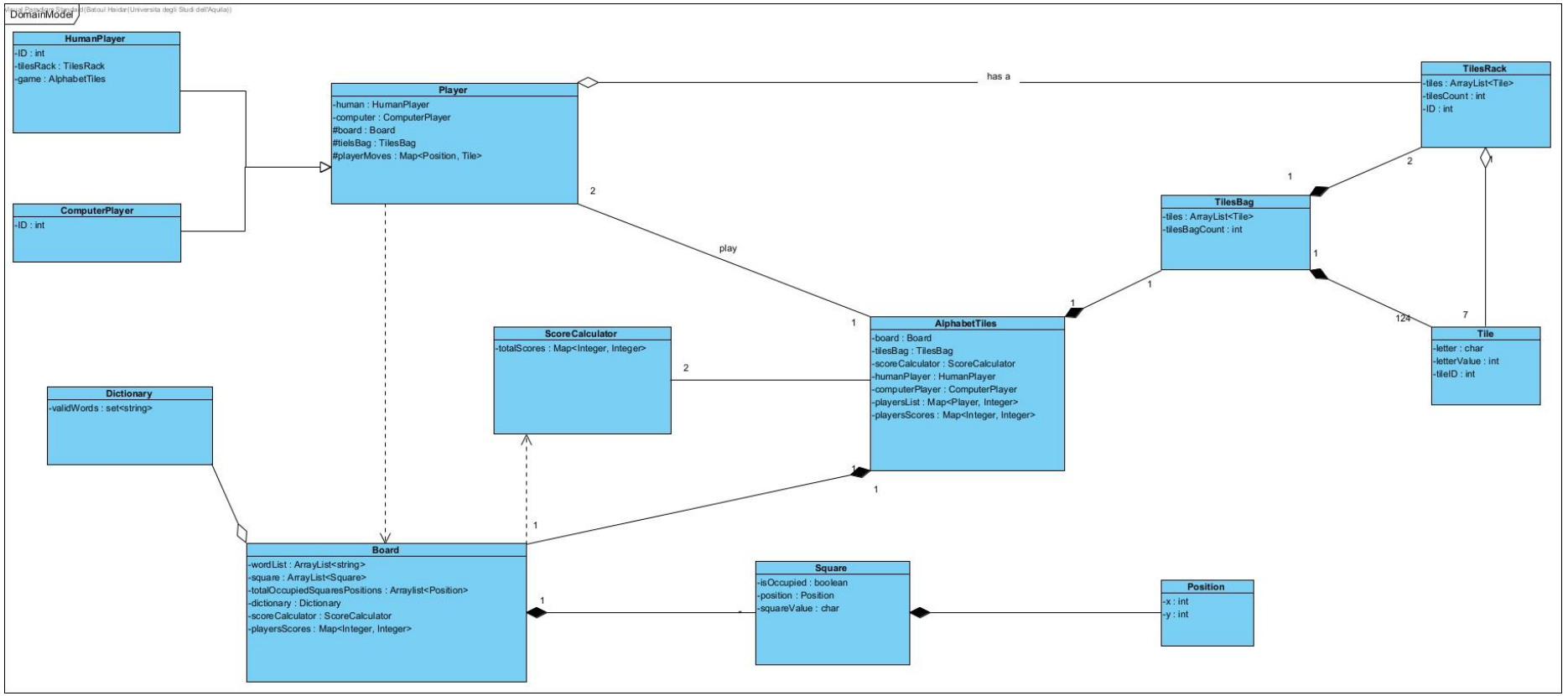
INCEPTION PHASE: LAYING THE FOUNDATION. USE CASES



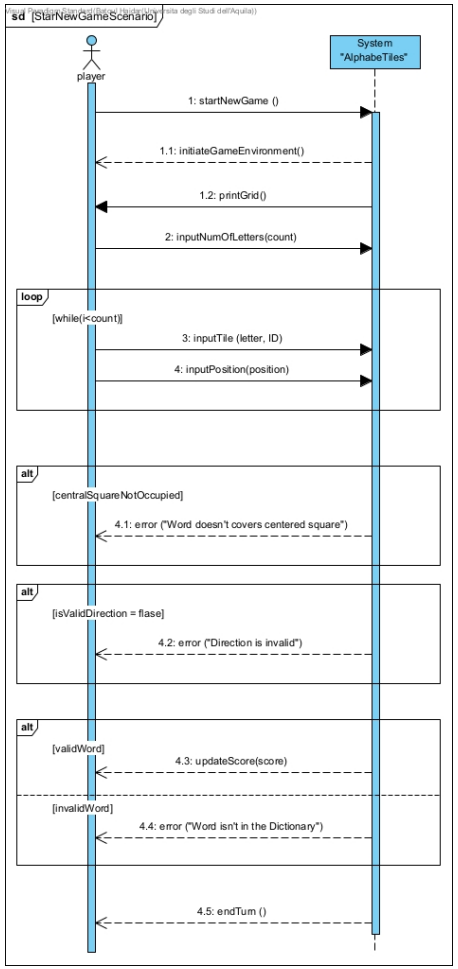
FIRST ITERATION: BUILDING THE CORE GAMEPLAY.

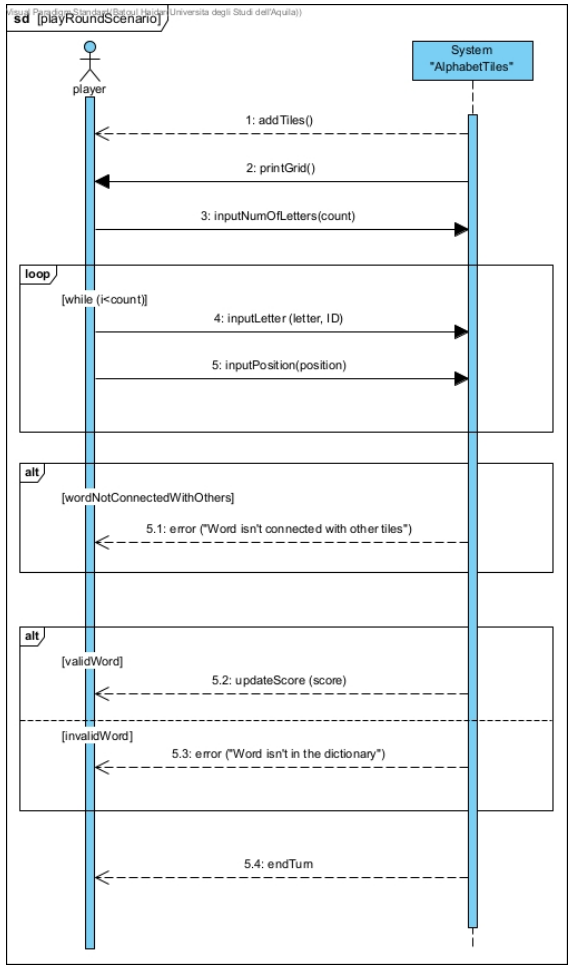
- **Objective:** Develop a working game prototype focusing on core gameplay mechanics.
- **Simplicity:** Prioritize simplicity over complexity to ensure a smooth development process.
- **Core Features:** Implement essential game features, such as:
  - Basic tile placement mechanics.
  - Player input for forming words.
  - Score tracking system.

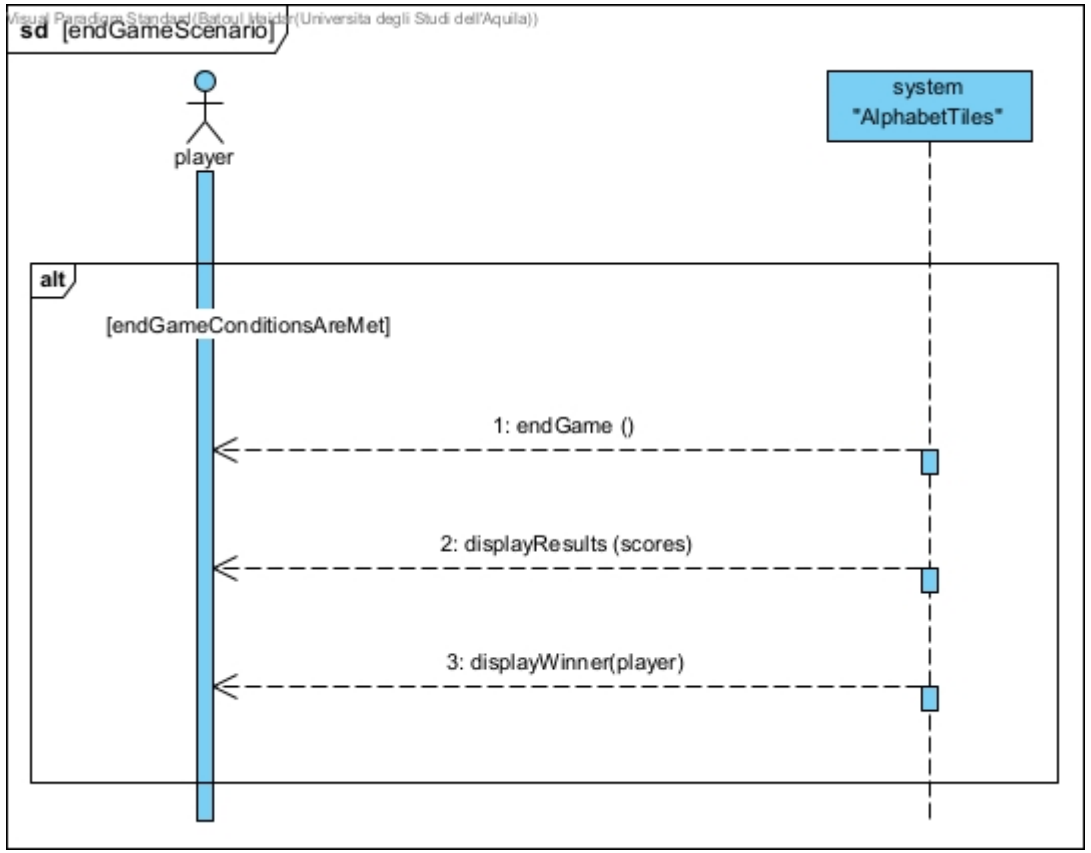
FIRST ITERATION: OOA DOMAIN MODEL DIAGRAM



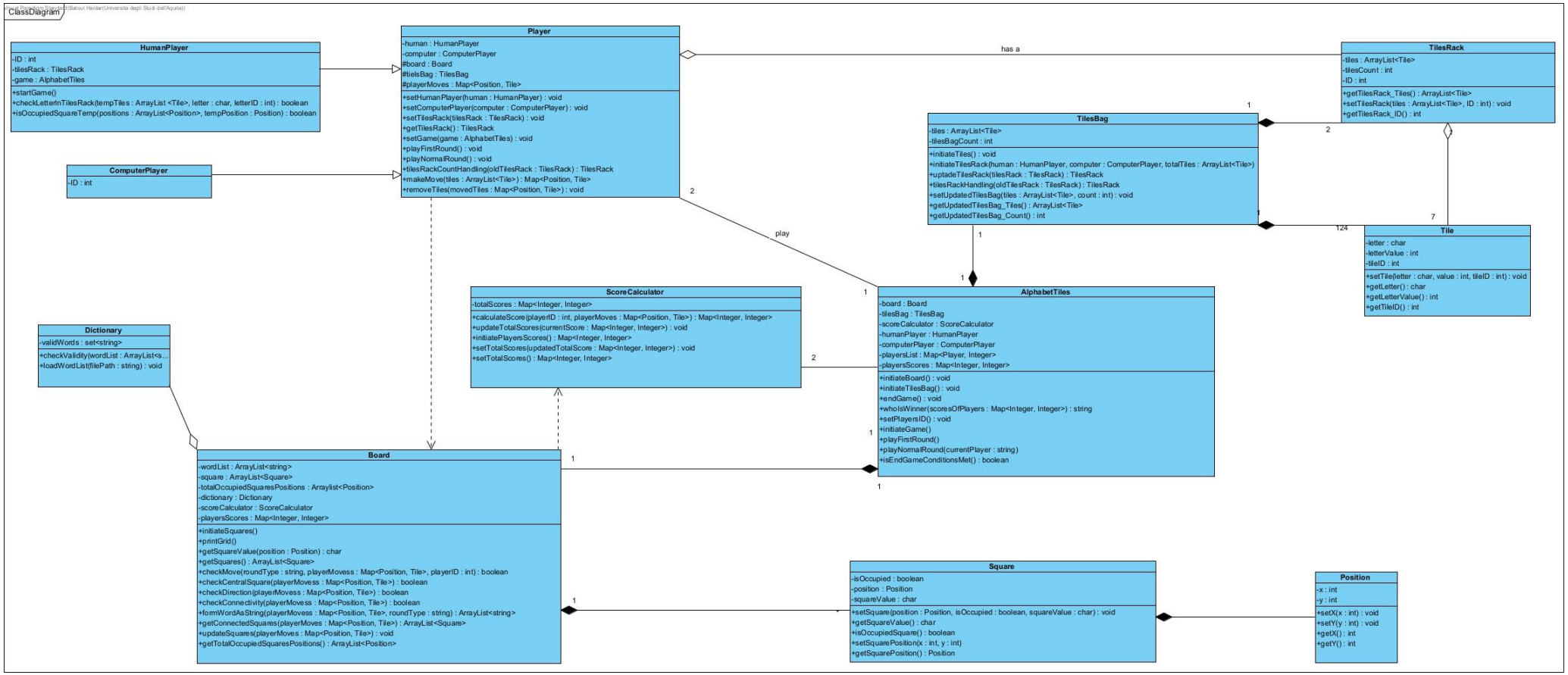
FIRST ITERATION: OOA
SYSTEM SEQUENCE DIAGRAMS



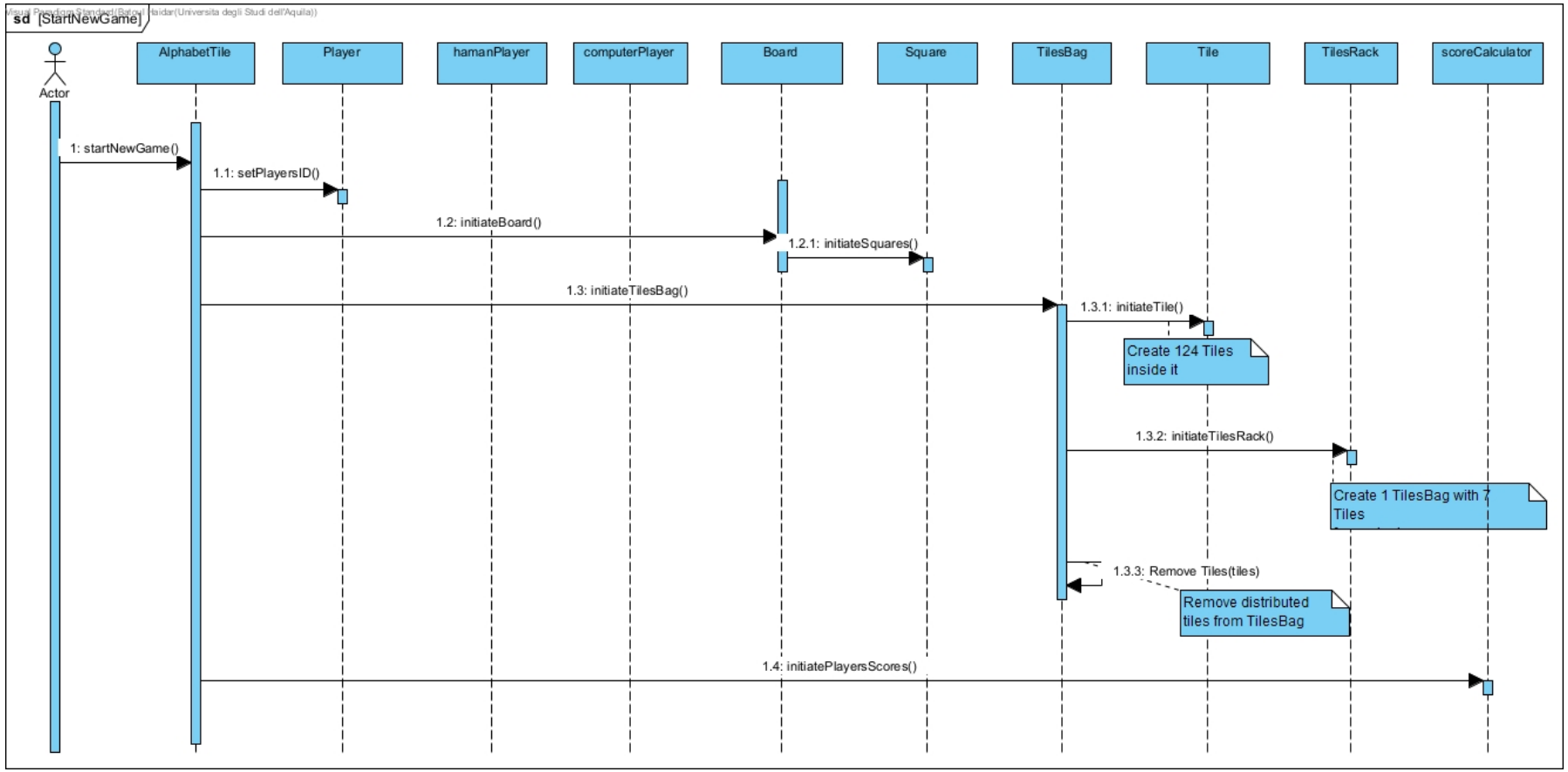




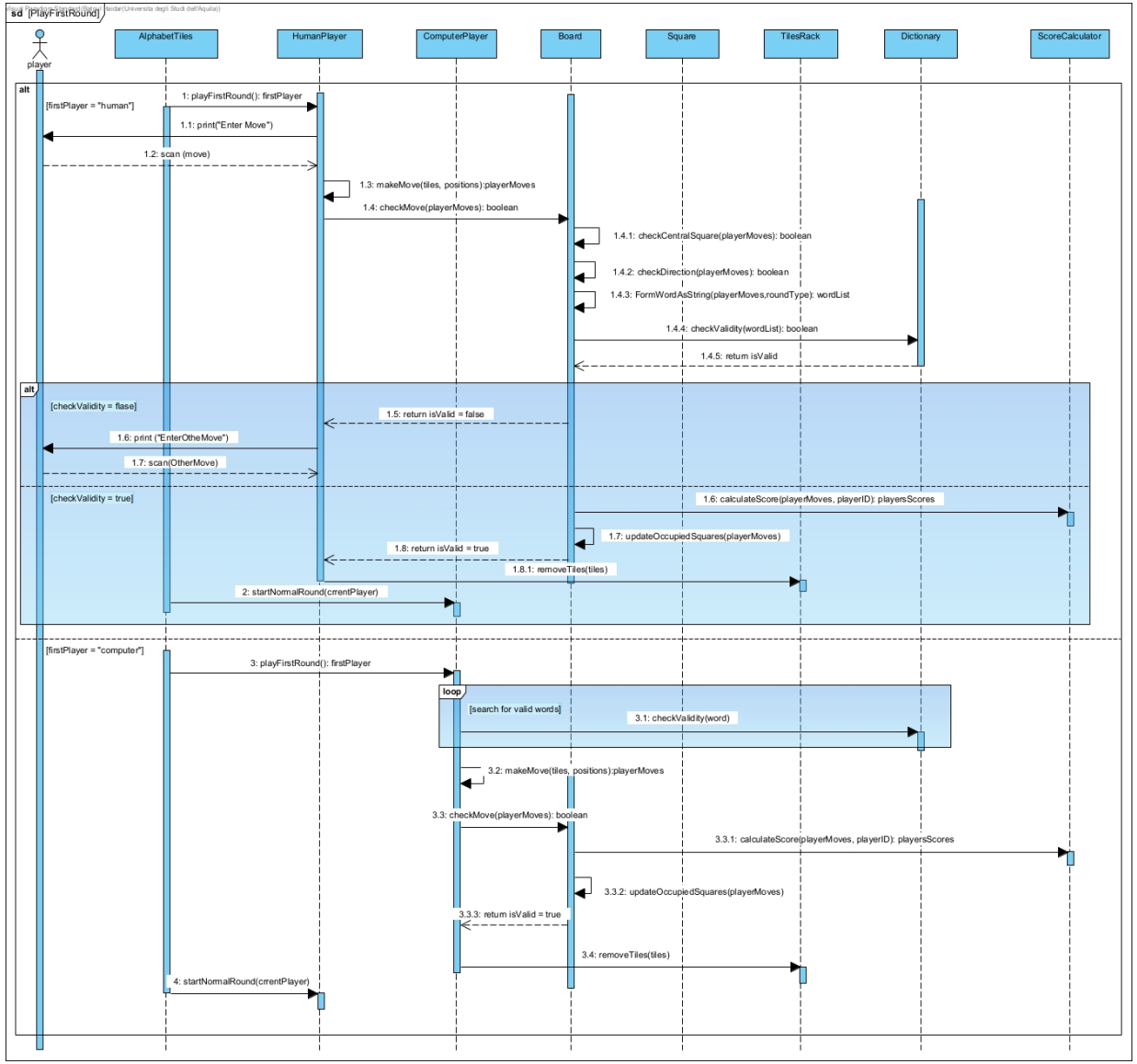
FIRST ITERATION: OOD CLASS DIAGRAM



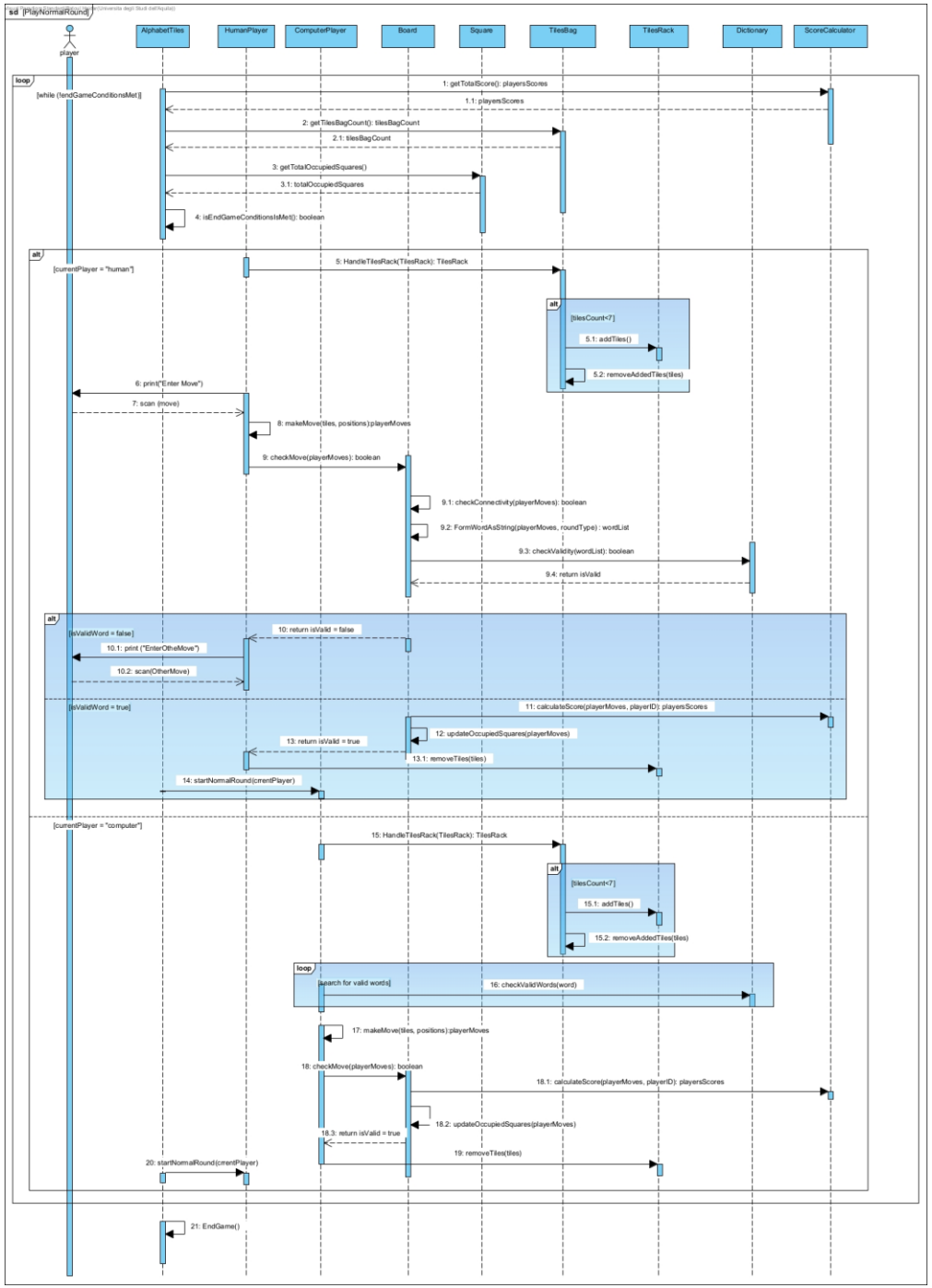
FIRST ITERATION: OOD SEQUENCE DIAGRAM USE CASE: START GAME



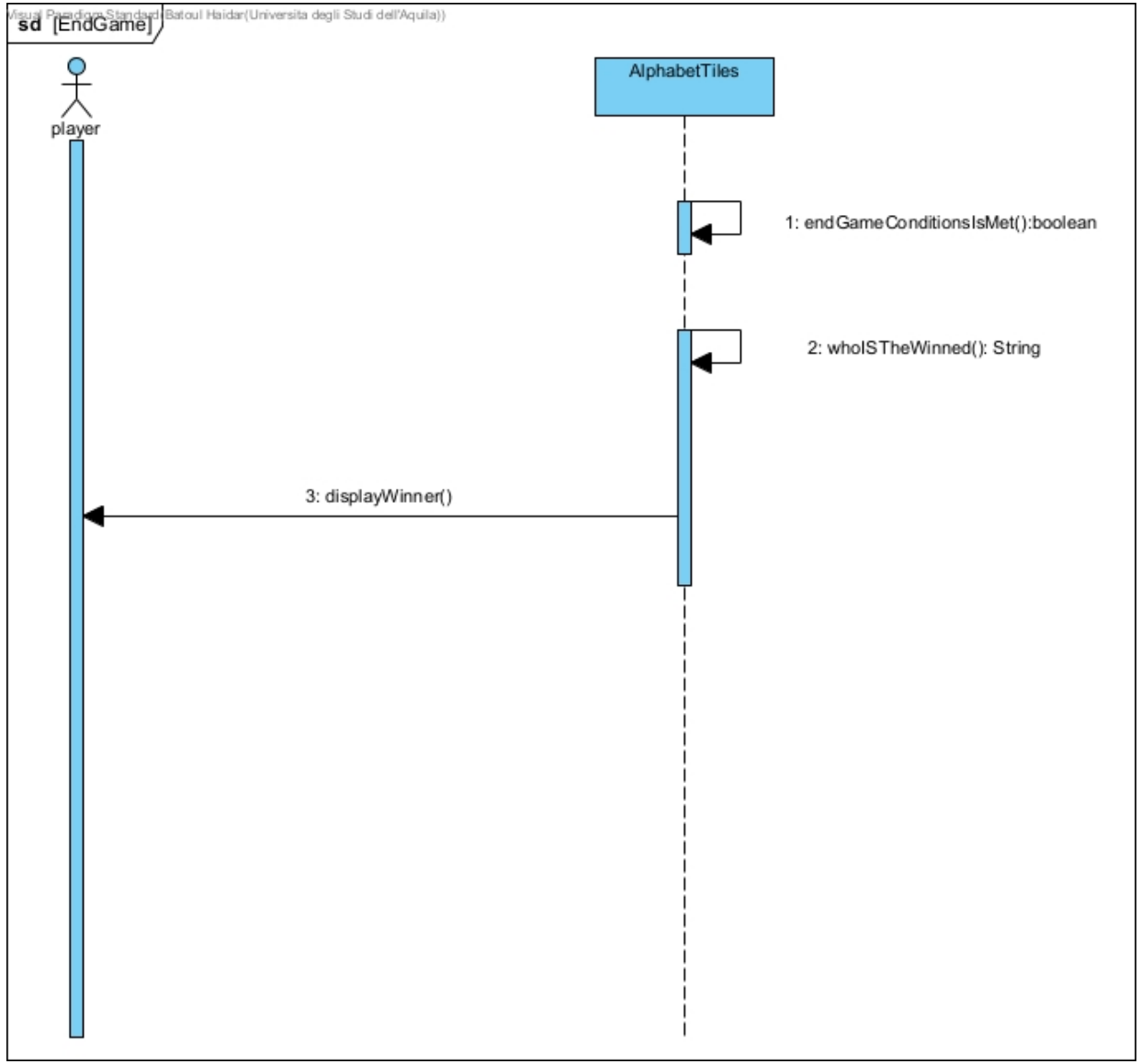
FIRST ITERATION: OOD SEQUENCE DIAGRAM USE CASE: PLAY FIRST ROUND



FIRST ITERATION: OOD SEQUENCE DIAGRAM USE CASE: PLAY NORMAL ROUND

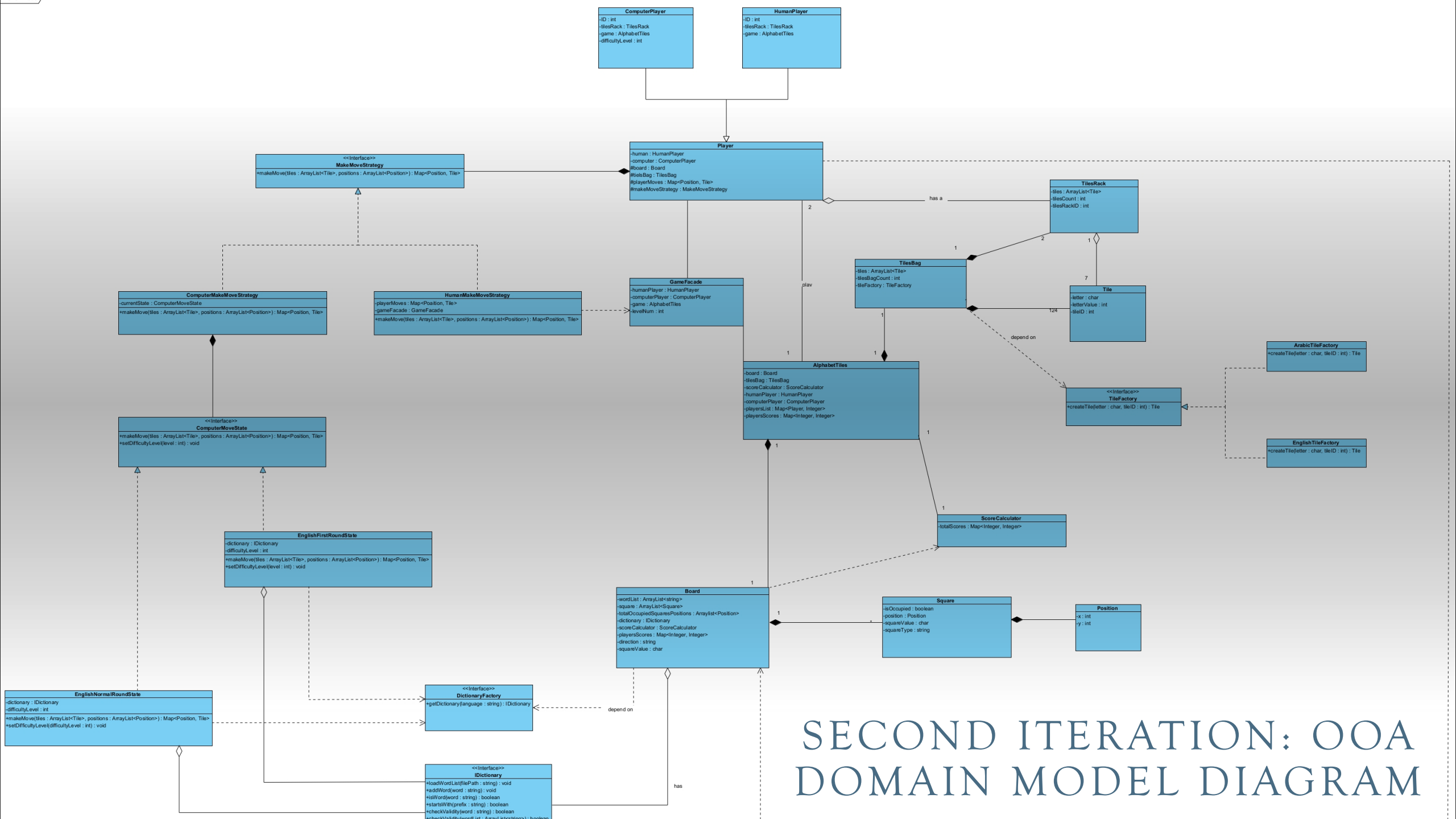


FIRST ITERATION: OOD SEQUENCE DIAGRAM USE CASE: END GAME



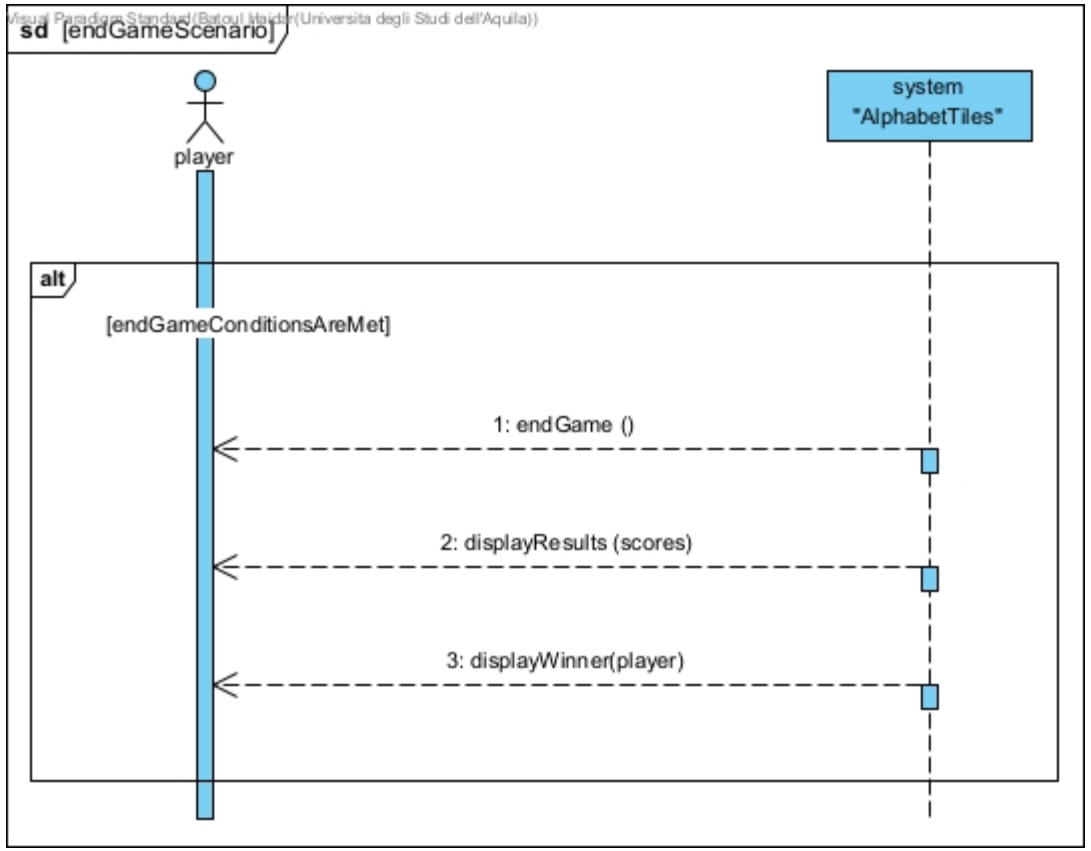
SECOND ITERATION: ENHANCING FEATURES AND FUNCTIONALITY.

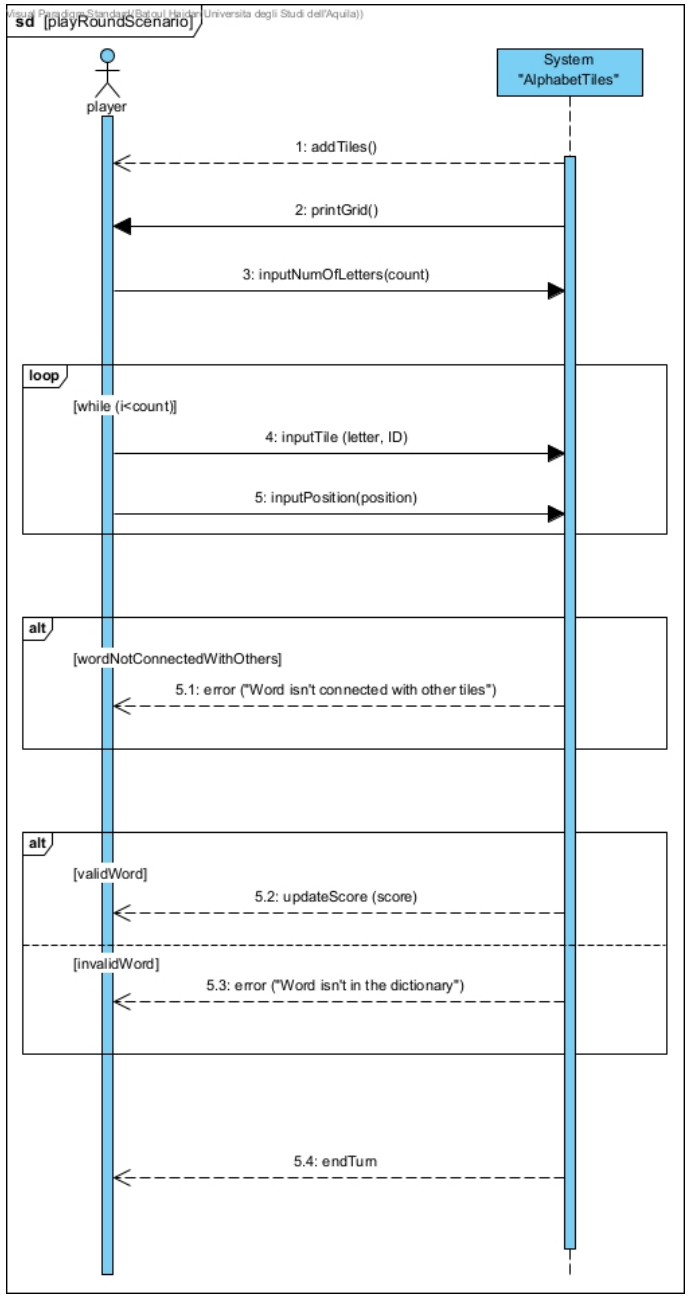
- **Objective:** Refine the gameplay experience by adding new features and enhancing existing functionalities to increase player engagement and enjoyment.
- **Simplicity:** Maintain a balance between complexity and simplicity
- **Core Features:**
  - Difficulty Levels: Introduce multiple difficulty levels (easy, medium, hard) to cater to a broader range of players, offering both casual and challenging gameplay experiences.
  - Square Points System: Implement a scoring system where certain squares on the game board provide bonus points when utilized in word formation.
  - Design Patterns: Incorporate design patterns to enhance code maintainability and scalability.

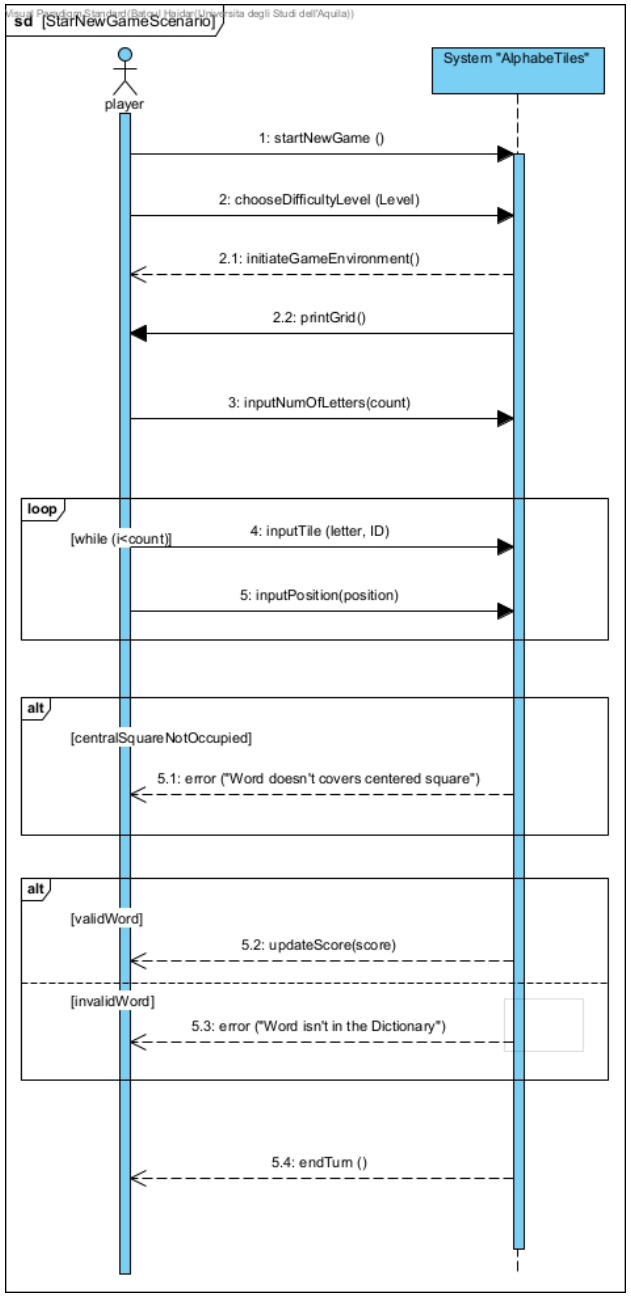


SECOND ITERATION: OOA DOMAIN MODEL DIAGRAM

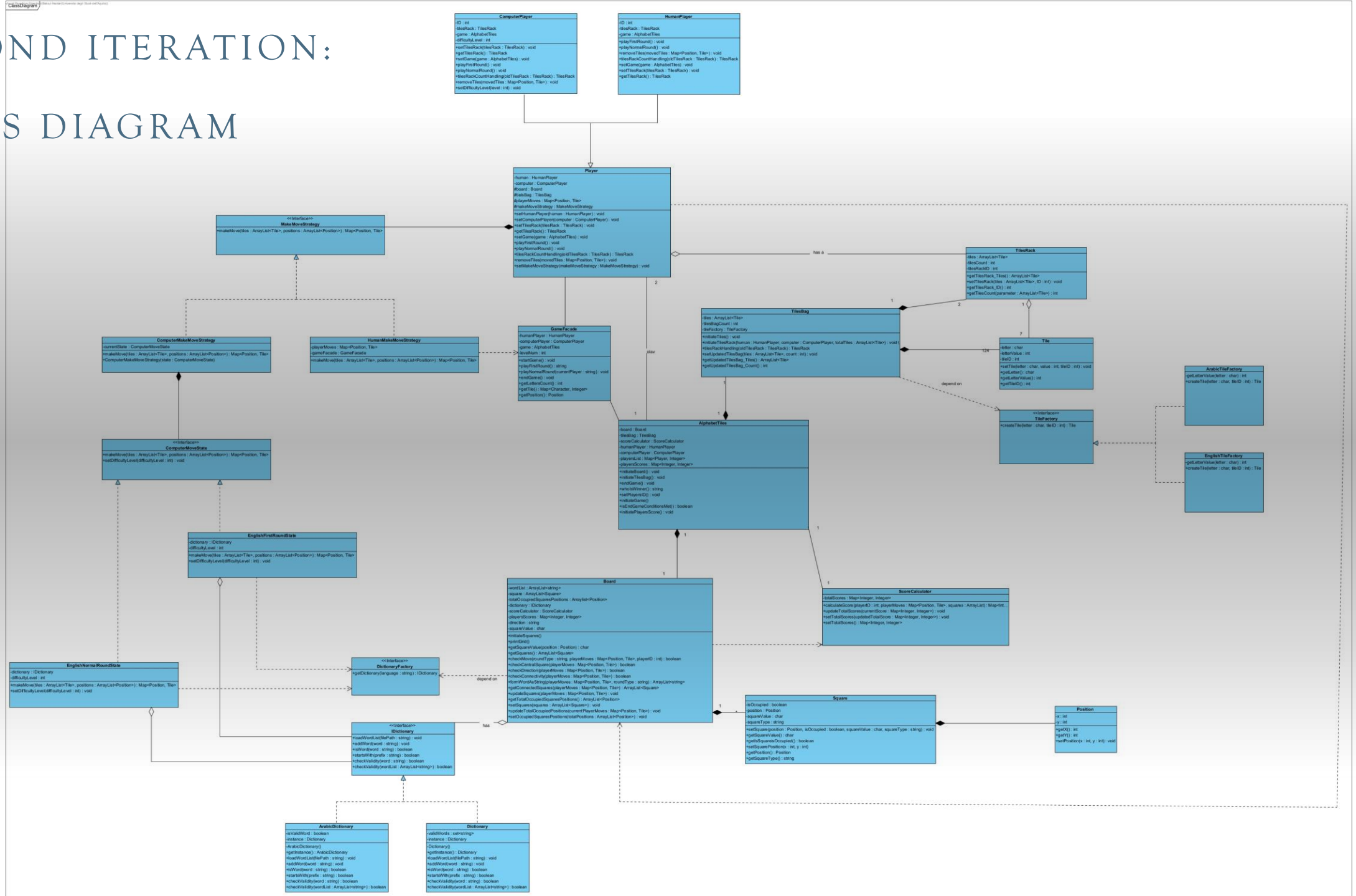
SECOND ITERATION: OOA SYSTEM SEQUENCE DIAGRAMS



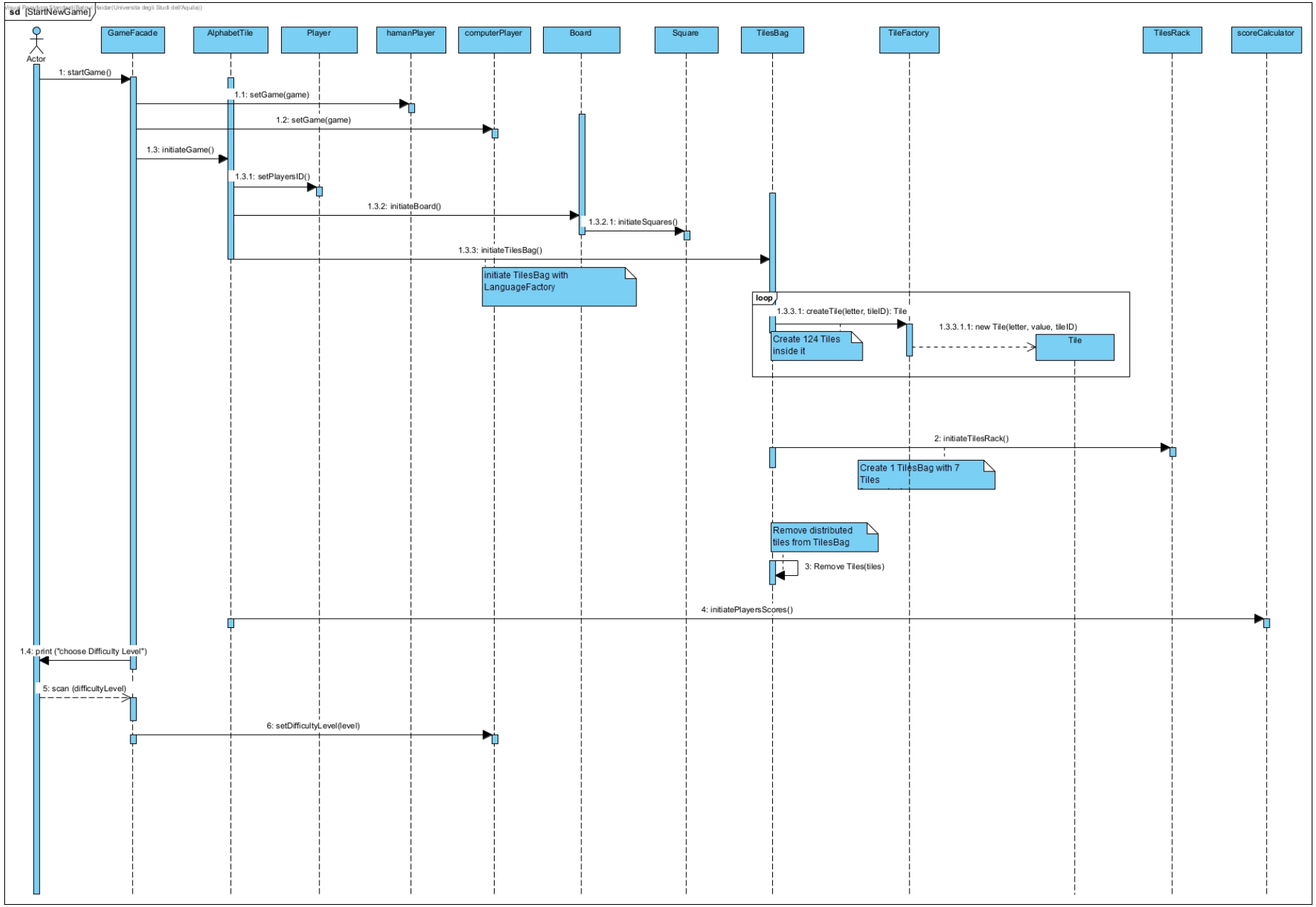




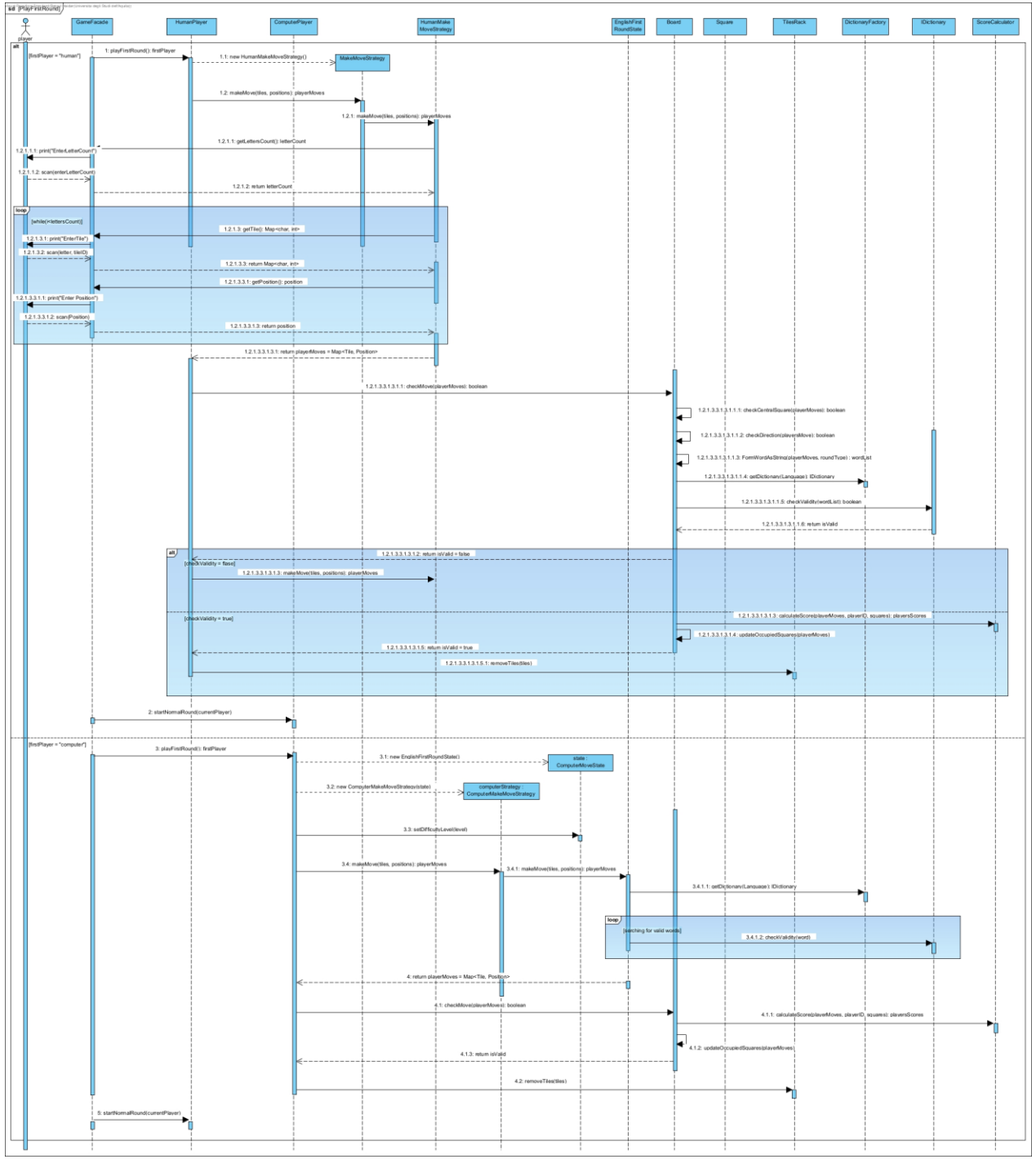
SECOND ITERATION: OOD CLASS DIAGRAM



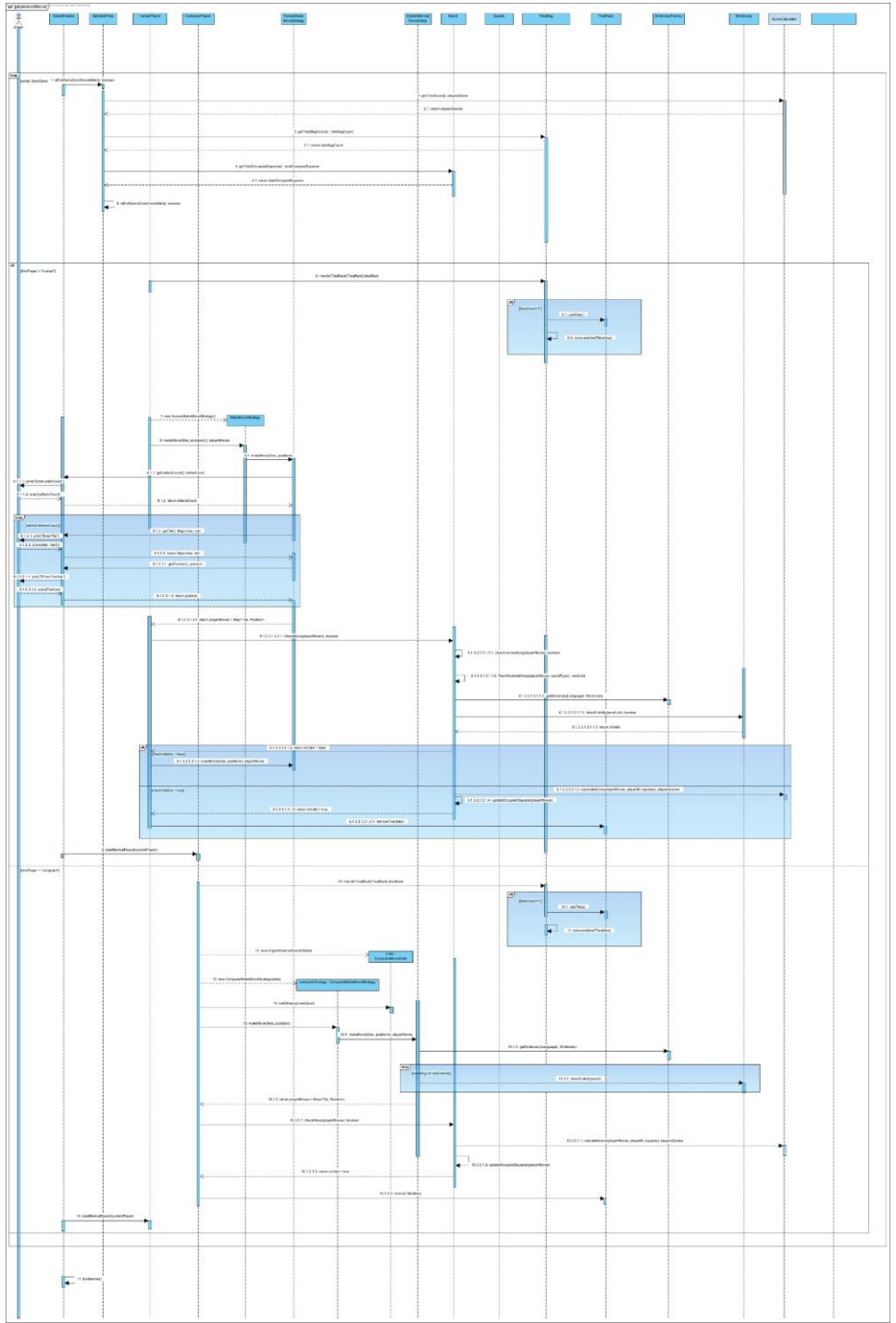
SECOND ITERATION: OOD SEQUENCE DIAGRAM USE CASE: START GAME



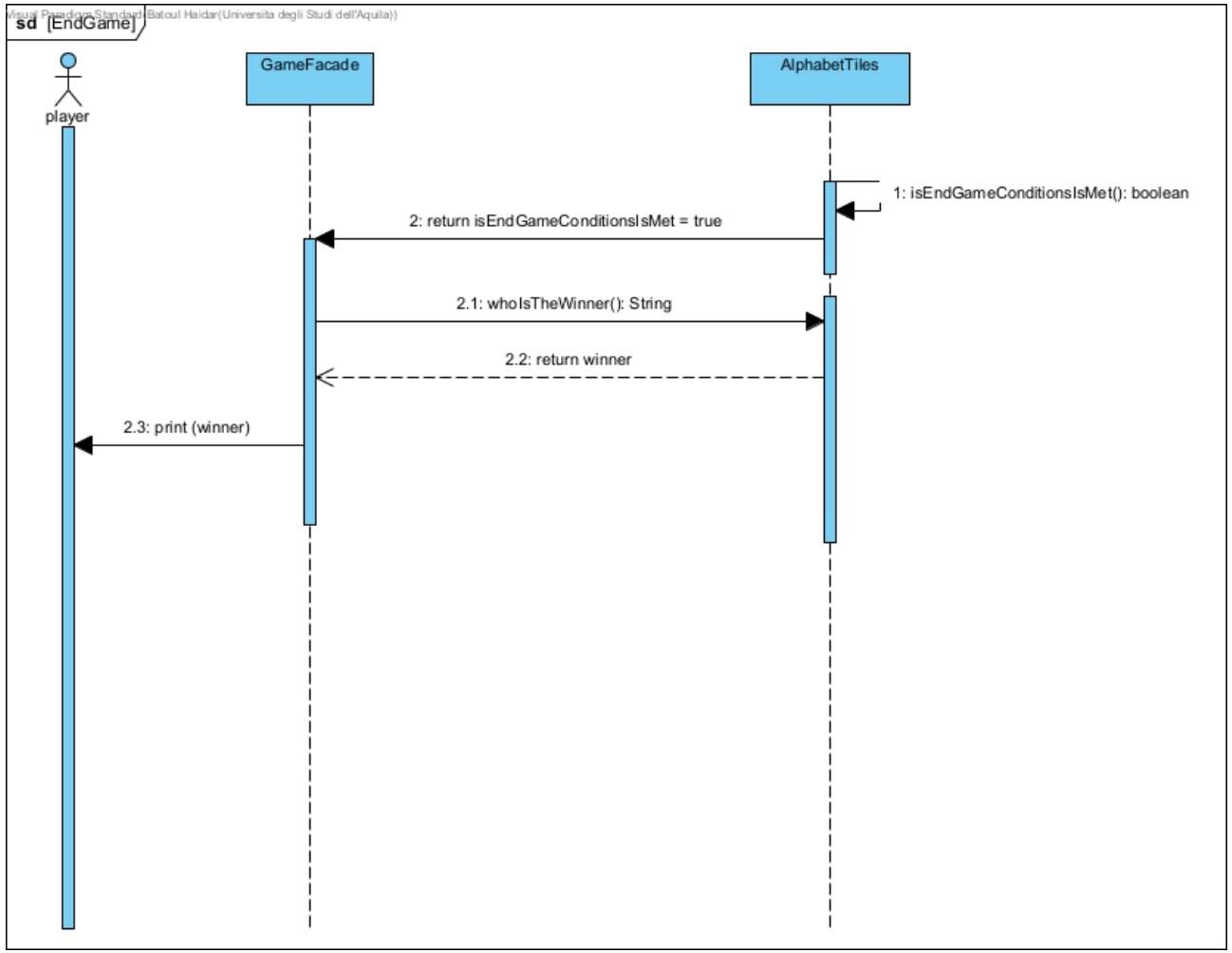
SECOND ITERATION: OOD SEQUENCE DIAGRAM USE CASE: PLAY FIRST ROUND



SECOND ITERATION: OOD SEQUENCE DIAGRAM USE CASE: PLAY NORMAL ROUND

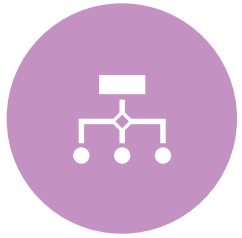


SECOND ITERATION: OOD SEQUENCE DIAGRAM USE CASE: END GAME



ARCHITECTURAL INSIGHTS: DESIGN PATTERNS IN ALPHABET TILES.

FAÇADE
DESIGN PATTERN.



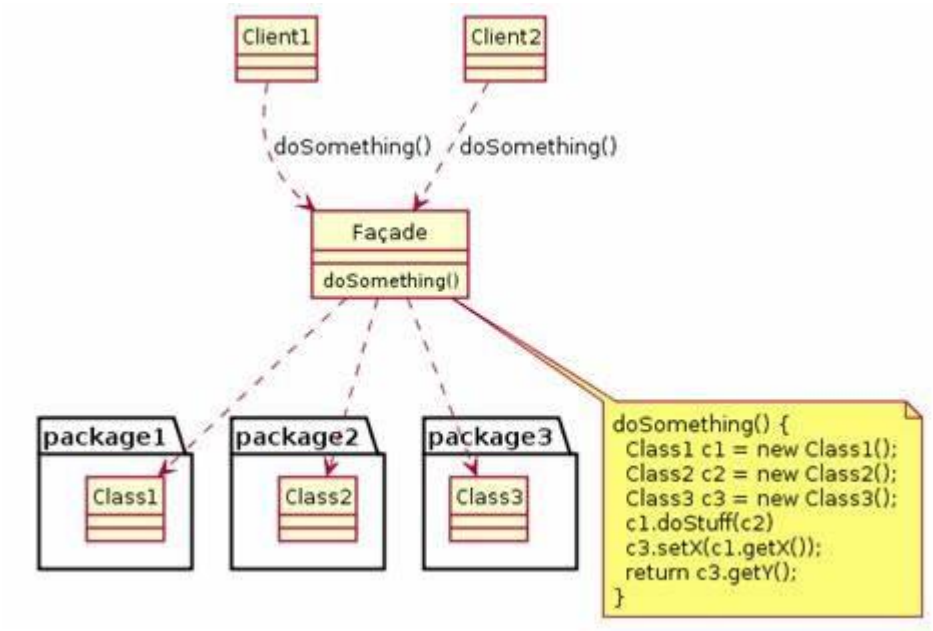
SINGLETON
DESIGN PATTERN.

FACTORY
DESIGN PATTERN.

STRATEGY
DESIGN PATTERN.

STATE
DESIGN PATTERN.

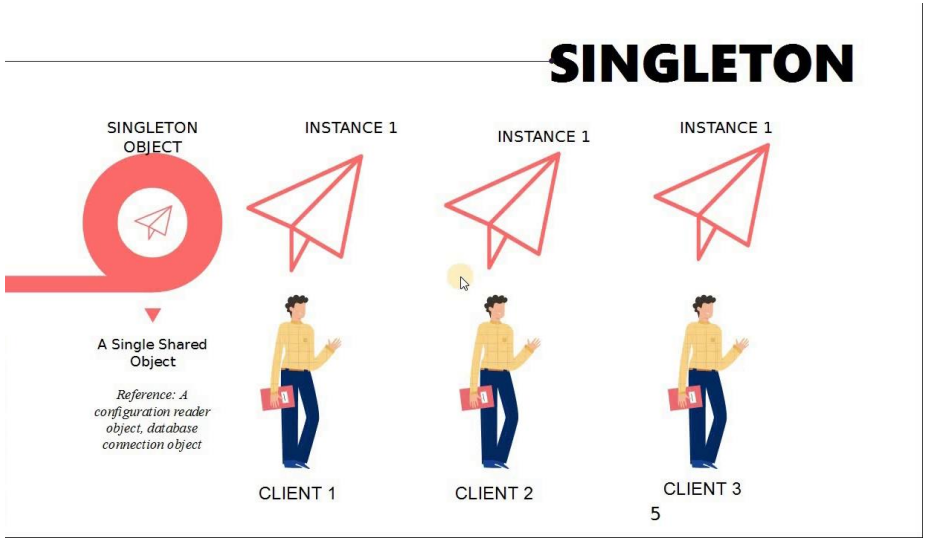
FAÇADE DESIGN PATTERN



- The Facade Design Pattern is a structural design pattern that provides a simplified interface to a complex system of classes, libraries, or frameworks. By doing so, it hides the complexities of the system and makes it easier to access and use for the client.

- By using “GameFacade” class, it offers a simpler interface to complex subsystems in the game, making the game easier to develop, extend, and maintain."

SINGLETON DESIGN PATTERN

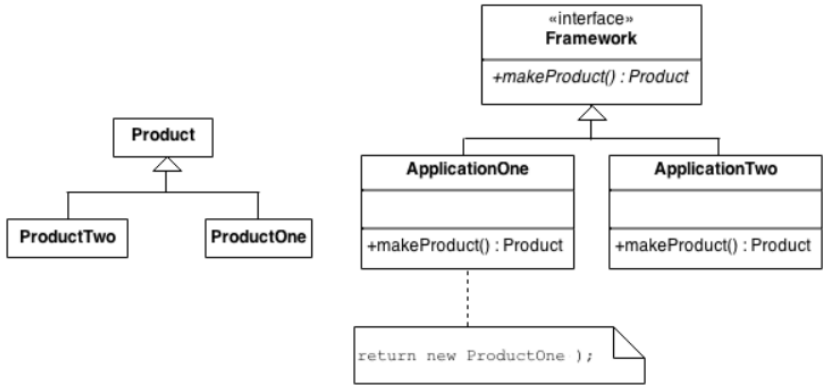


- The Singleton design pattern ensures that a class has only one instance and provides a global point of access to that instance. The benefits of using a Singleton, in general, include:

- It can be accessible only by public getInstence().

- Loading a dictionary, can be resource-intensive (due to memory usage). A Singleton ensures that the resource-intensive process of loading the dictionary data into memory happens only once, which can significantly improve performance and reduce memory usage.

FACTORY DESIGN PATTERN

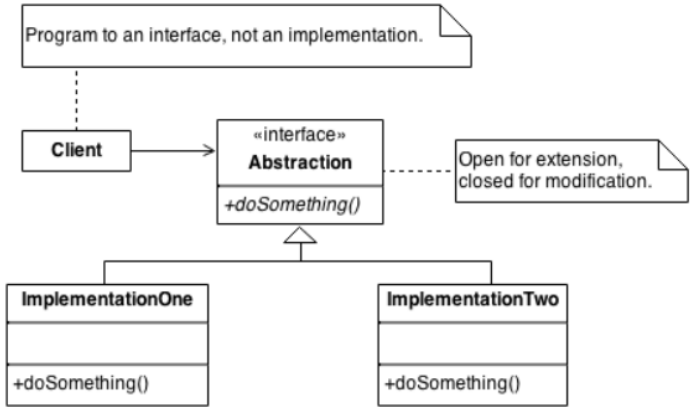


- Factory Method is a creational design pattern, used to creating objects as Template Method, which means create instances of different classes that share a common base class or interface.

- In Alphabet Tiles game, it is very necessary to use this design pattern due to additional languages in future extension.

- So, this design pattern make this extension easy by creating Tiles & dictionary by interface which could be implemented by any new language.

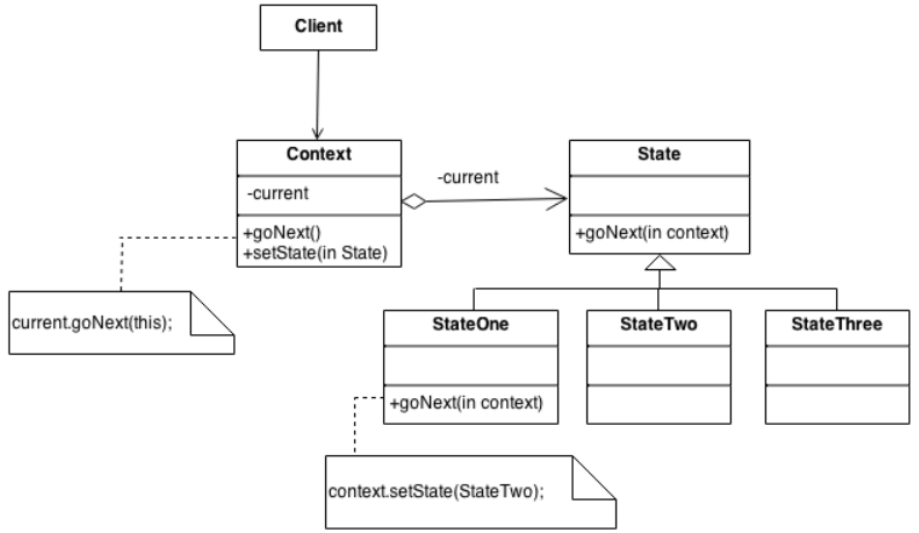
STRATEGY DESIGN PATTERN



- It is behavioural design pattern. It define a family of algorithms, encapsulate each one, and make them interchangeable.

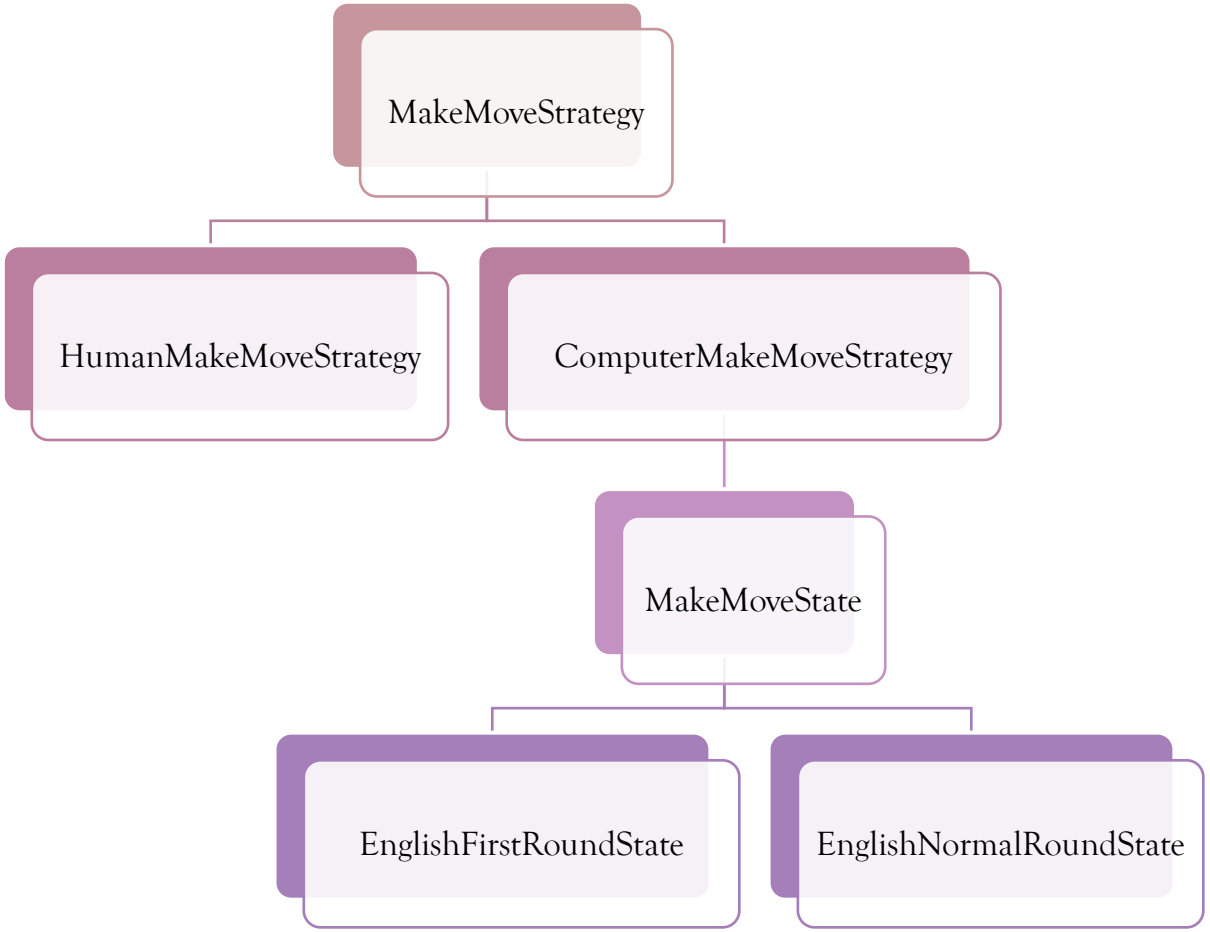
- In Alphabet Tiles, there is a strategy to form a player move contains of Map<Position, Tile>. But this strategy is different between human and computer players, because human will input tiles and positions then by makeMoveStrategy system will form the move. But computer player will form the word on positions by special strategy.

STATE DESIGN PATTERN



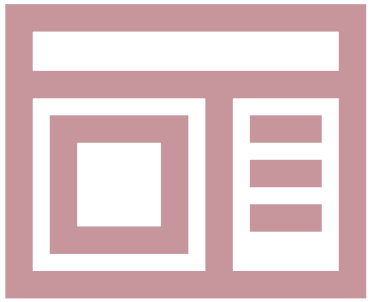
- It is behavioural design pattern. It allow an object to alter its behaviour when its internal state changes.

- In Alphabet Tiles, computer strategy to form a word will be different between first round and other rounds due to their special conditions. In addition, by adding other language, it also be different. So it is very useful to use state for strategies.



COMBINATION OF STRATEGY & STATE DESIGN PATTERNS

APPLYING GRASP PRINCIPLES FOR SOLID DESIGN.



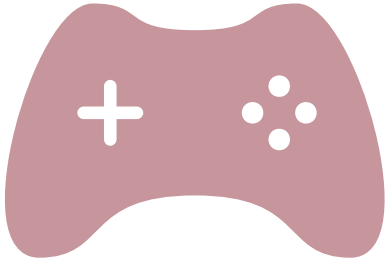
1. Creator : in Alphabet Tiles each class create classes that contained in itself. This leads to achieving :
   - low coupling
   - increased clarity
   - encapsulation
   - Reusability

   Examples from project:
   - Alphabet Tiles creates Board and TilesBag
   - Board creates Square.
   - TilesBag creates TilesRack and Tiles.

2. Information Expert : Assigning responsibilities to the expert class. In Alphabet Tiles each class is responsible of its contained class.

   Examples:
   - Board responsible of Square . In Board getSquare().

APPLYING GRASP PRINCIPLES FOR SOLID DESIGN.



3. Controller :

   To know if we have a controller, and who is the controller we should ask the following question:

   "What is the first object beyond the UI layer that receives and controls a system operation"

   In Alphabet Tiles : "gameFacade" is the controller, because it:

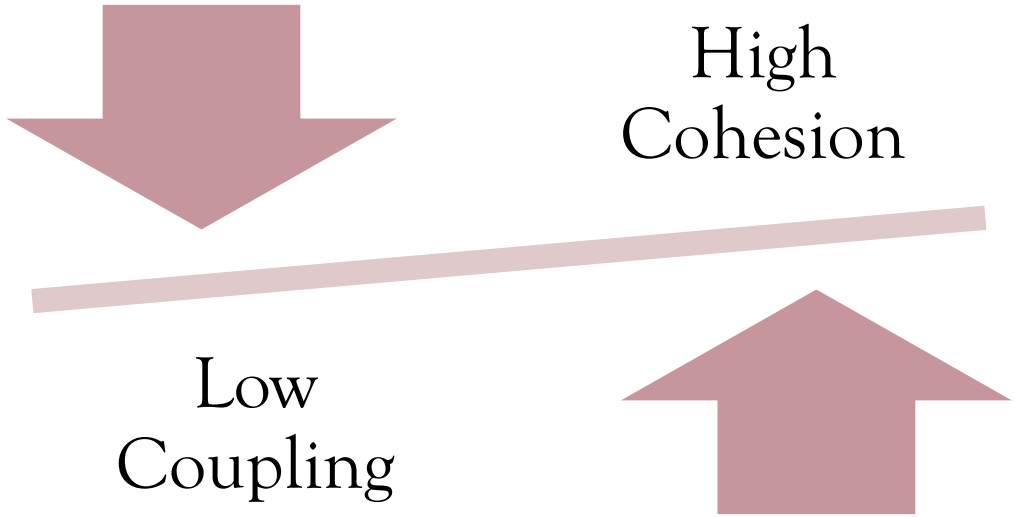
   - Receives and processes user inputs.
   - Manages the flow of the game or application.
   - Coordinates between the user interface (view) and the data/model.

APPLYING GRASP PRINCIPLES FOR SOLID DESIGN.

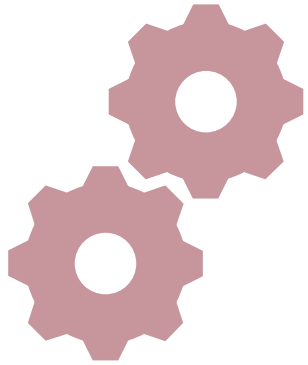
4. Low Coupling & High Cohesion:

As we noticed in the game, responsibilities are assigned and distributed from object to other, this leads to make each one of them focused , understandable and manageable. Which will achieve Low coupling with high cohesion.

So, the code now is easier to read, understand, modify, develop, test and has increased potential for reuse.



NAVIGATING CHALLENGES: OVERCOMING HURDLES IN ALPHABET TILES DEVELOPMENT



- **Performance Optimization:**
  - **Issue:** Handling searching in dictionary for word validation and suggestion can lead to performance bottlenecks.
  - **Solution:** Implementing more efficient searching algorithms Trie (prefix tree) can significantly reduce search times for word validation and suggestion.

FUTURE EXTENSION

- Multilingual Expansion: The next significant evolution could be a version in Arabic. This not only appeals to a vast new audience but also presents unique challenges due to the script and linguistic structure of Arabic, such as the use of root words and pattern formations.

THANK YOU