Scrabble Game Project Report

Submitted by:

Hafiz Muhammad Zain Sharif (24L-0603)

Instructor:

Usman Anwar

Department of Computer Science

University Name

May 13, 2025

Abstract

This project develops a digital Scrabble game using C++ and the SFML library, replicating the classic word game where players form words on a 15x15 board to score points. The game aims to provide an accessible, interactive experience with automated rule enforcement, word validation, and scoring, including bonus tiles. It demonstrates object-oriented programming (OOP) principles such as inheritance, polymorphism, composition, and aggregation, ensuring modular and extensible code. Features include player racks, a tile bag, dictionary validation, and a user-friendly interface with move cancellation and dictionary updates.

Contents

1 Introduction

1.1 Problem Statement

Physical Scrabble games require manual setup, score tracking, and rule enforcement, which can be cumbersome and limit accessibility. A digital version overcomes these challenges by automating gameplay mechanics, validating words against a dictionary, and calculating scores with bonuses, making the game portable and user-friendly.

1.2 Objectives

- Create a functional Scrabble game with a graphical interface using C++ and SFML.
- Implement core mechanics: tile placement, word validation, scoring, and turn management.
- Utilize OOP principles to ensure modular, reusable, and maintainable code.
- Enhance user interaction with features like move cancellation and dictionary updates.

1.3 Motivations

The project was motivated by the desire to digitize a beloved word game, enhancing accessibility on modern platforms. It provided a practical opportunity to apply OOP concepts, improving programming proficiency. The complexity of implementing word formation, validation, and scoring with bonuses offered an engaging challenge.

2 OOP Concepts Used

The Scrabble game employs key OOP principles to achieve robust, maintainable code:

- **Inheritance**: The `Tile` class inherits from `GameEntity` (for position management) and `Drawable` (for rendering), enabling code reuse and a clear hierarchy.
- **Polymorphism**: The `Drawable` interfaces virtual `draw` method allows polymorphic rendering, enabling different components (e.g., `Tile`) to be drawn uniformly.
- **Composition**: The `Board` class owns a 15x15 array of `Tile` objects, managing their lifecycle. Similarly, `Bag` and `YesNoDialog` compose `Piece` and `Button` objects, respectively.
- **Aggregation**: The `Rack` class loosely holds `Piece` objects, which can be reassigned to the `Bag`, supporting flexible tile management.

These principles minimized code duplication, enhanced modularity, and simplified future extensions, such as adding new tile types or game modes.

3 Class Diagrams

The UML class diagram, shown in Figure 1, visualizes the Scrabble games core architecture, focusing on 9 key classes for simplicity, inspired by the Monopoly games UML diagram. Classes are represented as rectangular boxes with compartments for attributes and methods. Relationships are depicted with clear arrows: double arrows for inheritance, filled diamonds for composition, hollow diamonds for aggregation, and solid lines for association, using thin black lines for clarity. The diagram is defined using PlantUML code, which generates a professional UML diagram when rendered in a PlantUML-compatible tool (e.g., `https://www.plantuml.com`).

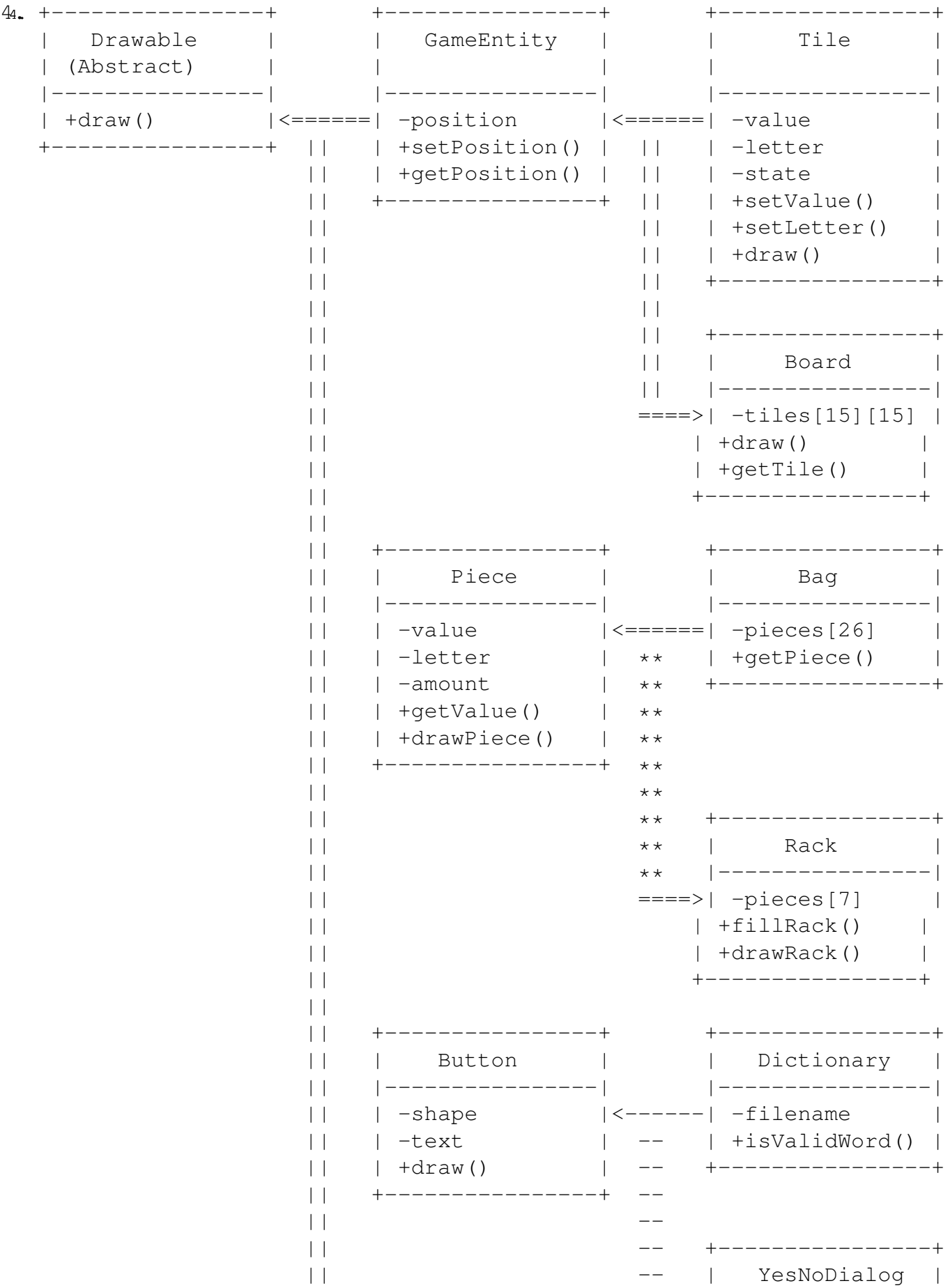
Important: Generate the Visual Diagram

1. Copy the PlantUML code below into `https://www.plantuml.com` or a tool like Visual Studio Code with the PlantUML extension.
2. Press "Generate" to render the diagram with boxes and arrows.
3. Download the PNG file (e.g., $\texttt{uml}_d iagram.png).Place$

Figure 1: Simplified UML Class Diagram for Scrabble Game

ASCII Representation of UML Diagram To provide a textual sense of the diagrams structure, below is an ASCII art approximation showing boxes and arrows for key classes and relationships. For the full visual diagram, render the PlantUML code.

4.


Drawable
(Abstract)
+draw()
GameEntity
-position
+setPosition()
+getPosition()
Tile
-value
-letter
-state
+setValue()
+setLetter()
+draw()
Board
-tiles[15][15]
+draw()
+getTile()
Piece
-value
-letter
-amount
+getValue()
+drawPiece()
Bag
-pieces[26]
+getPiece()
Rack
-pieces[7]
+fillRack()
+drawRack()
Button
-shape
-text
+draw()
Dictionary
-filename
+isValidWord()
YesNoDialog

```
                    ||                              ====>| -yesButton      |
                    ||                                   | -noButton       |
                    ||                                   | +draw()         |
                    ||                                   +-----------------+
                    ||
                    \/
Legend:
  ||   = Inheritance (Double Arrow)
  ===> = Composition (Filled Diamond)
  **   = Aggregation (Hollow Diamond)
  --   = Association (Solid Line)
```

PlantUML Code for Visual UML Diagram

```
@startuml ScrabbleGameUML
' Styling inspired by Monopoly report: clean, thin arrows
skinparam class {
    BackgroundColor #FFFFFF
    BorderColor #000000
    BorderThickness 1
    ArrowColor #000000
    ArrowThickness 1
    FontName Times
    FontSize 12
    FontColor #000000
}
skinparam shadowing false
skinparam dpi 150
skinparam ranksep 30
skinparam nodesep 20
skinparam monochrome true

' Abstract class Drawable
abstract class Drawable {
    +draw(window: sf::RenderWindow&): void <<virtual>>
}

' Base class GameEntity
class GameEntity {
    -position: sf::Vector2f
    +setPosition(x: float, y: float): void
    +getPosition(): sf::Vector2f
}

' Tile class
class Tile {
    -value: int
    -letter: char
    -state: string
```

```
    +setValue(value: int): void
    +setLetter(letter: char): void
    +getValue(): int
    +getLetter(): char
    +draw(window: sf::RenderWindow&): void
}
Tile <|-- Drawable : <<Inheritance>>
Tile <|-- GameEntity : <<Inheritance>>

' Board class
class Board {
    -tiles: Tile*[15][15]
    +draw(window: sf::RenderWindow&): void
    +getTile(i: int, j: int): Tile*
}
Board *--> "15x15" Tile : <<Composition>>

' Piece class
class Piece {
    -value: int
    -letter: char
    -amount: int
    +getValue(): int
    +getLetter(): char
    +drawPiece(window: sf::RenderWindow&, x: int, y: int): void
}

' Bag class
class Bag {
    -pieces: Piece*[26]
    +getPiece(index: int): Piece*
}
Bag *--> "26" Piece : <<Composition>>

' Rack class
class Rack {
    +turn: int <<static>>
    -pieces: Piece*[7]
    +fillRack(bag: Bag&): void
    +drawRack(window: sf::RenderWindow&, startX: int, startY: int): void
}
Rack o--> "7" Piece : <<Aggregation>>

' Button class
class Button {
    -shape: sf::RectangleShape
    -text: sf::Text
    +draw(window: sf::RenderWindow&): void
```

```
    +isMouseOver(window: sf::RenderWindow&): bool
}

' Dictionary class
class Dictionary {
    -filename: string
    +isValidWord(word: string): bool
    +addWordToDictionary(word: string): bool
}

' YesNoDialog (not in simplified diagram)
class YesNoDialog {
    -yesButton: Button
    -noButton: Button
    -promptText: sf::Text
    +show(words: string): void
    +draw(window: sf::RenderWindow&): void
}
YesNoDialog *--> "2" Button : <<Composition>>

' Layout for clarity
Drawable -[hidden]down- GameEntity
GameEntity -[hidden]down- Tile
Tile -[hidden]right- Board
Piece -[hidden]down- Bag
Bag -[hidden]right- Rack
Button -[hidden]down- Dictionary
Dictionary -[hidden]down- YesNoDialog

@enduml
```

3.1 Detailed UML Description

The simplified UML diagram (Figure 1) includes 9 core classes (`Drawable`, `GameEntity`, `Tile`, `Board`, `Piece`, `Bag`, `Rack`, `Button`, `Dictionary`) with 7 key relationships for clarity, inspired by the Monopoly games UML structure. All 10 classes, including `YesNoDialog`, are described below with attributes, methods, and relationships. Visibility: (+) public, (-) private.

- **Drawable** (Abstract)
  - *Methods*: `+draw(sf::RenderWindow&): void` (virtual)
- **GameEntity**
  - *Attributes*: `-position: sf::Vector2f`
  - *Methods*: `+setPosition(float, float): void`, `+getPosition(): sf::Vector2f`
- **Tile** (Inherits `GameEntity`, `Drawable`)

  - *Attributes*: -value: int, -letter: char, -state: string, -isplaced: bool, -isTemporary: bool, -tileTexture: sf::Texture, -tileSprite: sf::Sprite, -x: int, -y: int
  - *Methods*: +setValue(int): void, +setLetter(char): void, +setState(str void, +getValue(): int, +getLetter(): char, +getState(): string, +getIsPlaced(): bool, +setIsPlaced(bool): void, +getIsTempo bool, +setIsTemporary(bool): void, +draw(sf::RenderWindow&): void, +draw(sf::RenderWindow&, int, int): void, +getX(): int, +getY(): int

- **Board**
  - *Attributes*: -tiles: Tile*[15][15]
  - *Methods*: +draw(sf::RenderWindow&): void, +getTile(int, int): Tile*, +getTemporaryTiles(Tile*[], int&): void, +clearTemporaryTile void, +confirmTemporaryTiles(): void, +getWords(string&, string&, int, int): bool, +getAllWords(string[], int&): void
  - *Relationship*: Composes Tile (15x15)
- **Piece**
  - *Attributes*: -value: int, -letter: char, -amount: int, -remaining: int, -pieceTexture: sf::Texture, -pieceSprite: sf::Sprite
  - *Methods*: +getValue(): int, +getLetter(): char, +getAmount(): int, +getRemaining(): int, +pieceOut(): void, +pieceIn(): void, +setSprite(string): void, +drawPiece(sf::RenderWindow&, int, int): void, +getGlobalBounds(int, int): sf::FloatRect
- **Bag**
  - *Attributes*: -pieces: Piece*[26]
  - *Methods*: +getPiece(int): Piece*
  - *Relationship*: Composes Piece (26)
- **Rack**
  - *Attributes*: +turn: int (static), -pieces: Piece*[7], -number: int, -filledAmount: int
  - *Methods*: +fillRack(Bag&): void, +drawRack(sf::RenderWindow&, int, int): void, +getTileAt(int): Piece*, +removeTileAt(int): void, +addLetter(char, int): void, +getNumber(): int
  - *Relationship*: Aggregates Piece (up to 7)
- **Button**
  - *Attributes*: -shape: sf::RectangleShape, -text: sf::Text, -font: sf::Font

  - *Methods*: `+isMouseOver(sf::RenderWindow&): bool`, `+setBackColor(sf::Co`
    `void`, `+draw(sf::RenderWindow&): void`

- **YesNoDialog**
  - *Attributes*: `-yesButton: Button`, `-noButton: Button`, `-promptText: sf::Text`, `-visible: bool`, `-invalidWords: string`
  - *Methods*: `+show(string): void`, `+hide(): void`, `+isVisible(): bool`, `+getInvalidWords(): string`, `+isYesClicked(sf::RenderWindow&): bool`, `+isNoClicked(sf::RenderWindow&): bool`, `+update(sf::RenderWinc`
    `void`, `+draw(sf::RenderWindow&): void`
  - *Relationship*: Composes `Button` (2)

- **Dictionary**
  - *Attributes*: `-filename: string`
  - *Methods*: `+isValidWord(string): bool`, `+addWordToDictionary(string): bool`

Relationships (as shown in Figure 1):

- **Inheritance**: `Tile` inherits from `Drawable` and `GameEntity` (double arrows).
- **Composition**: `Board` composes `Tile` (15x15), `Bag` composes `Piece` (26) (filled diamonds).
- **Aggregation**: `Rack` aggregates `Piece` (up to 7) (hollow diamond).
- **Association**: `Button` and `Dictionary` are associated with other classes (solid lines).

Additional relationship: `YesNoDialog` composes `Button` (not shown in simplified diagram).

4 Test Cases

The following test cases verify the games functionality:

1. **Tile Placement**
   - *Input*: Click board tile (7,7), select rack piece 'A', click Place.
   - *Output*: 'A' displayed on (7,7), 'A' removed from rack, status: "Piece placed."
2. **Word Validation**
   - *Input*: Place tiles for "CAT" horizontally at (7,7), click Place.
   - *Output*: Status: "Valid word(s)! +[score] points", tiles confirmed, score updated.
3. **Invalid Word Handling**
   - *Input*: Place tiles for "XYZ" vertically, click Place.
   - *Output*: Dialog: "Add invalid word(s) to dictionary? XYZ", Yes/No buttons shown.
4. **Dictionary Update**

- *Input*: For invalid "XYZ", click Yes in dialog.
- *Output*: "XYZ" added to dictionary, tiles confirmed, status: "Words added to dictionary!"

5. **Cancel Move**
   - *Input*: Place 'B' on (8,7) temporarily, click Cancel.
   - *Output*: 'B' removed from board, returned to rack, status: "Move canceled."
6. **Scoring with Bonuses**
   - *Input*: Place "DOG" on (0,0) (Triple Word), click Place.
   - *Output*: Score: (D:2, O:1, G:2) * 3 = 15, status: "+15 points."
7. **Turn Management**
   - *Input*: Player 1 places valid word, clicks Place.
   - *Output*: Active rack switches to Player 2, highlighted rack updates.

5 Future Directions

Potential improvements to the Scrabble game include:

- **AI Opponent**: Implement a computer player with word-finding algorithms.
- **Online Multiplayer**: Add networking for remote play.
- **Turn Timers**: Introduce time limits to enhance challenge.
- **UI Animations**: Include animations for tile placement and scoring.
- **Save/Load Game**: Enable saving and resuming game states.
- **Dictionary UI**: Provide an interface to manage the dictionary.