Code Refactoring Sample Snippet Codes

Removed all unnecessary empty lines, e.g. in the method

Labyrinth.cs

```
private bool ExitFound(Cell cell)
        {
            bool exitFound = false;
            if (cell.Row == LABYRINTH_SIZE - 1 ||



                cell.Col == LABYRINTH_SIZE - 1 ||
                cell.Row == 0 ||
                cell.Col == 0)
            {
                exitFound = true;
            }

            return exitFound;

          }
```

➔

Engine.cs

```
        public bool ExitFound(Cell cell)
        {
            bool exitFound = false;
            bool rowBorder = cell.Row == LABYRINTH_SIZE - 1 || cell.Row == 0;
            bool columnBorder = cell.Column == LABYRINTH_SIZE - 1 || cell.Column == 0;
            if (rowBorder || columnBorder)
            {
                exitFound = true;
            }

            return exitFound;
        }
```

Introduced new variables to simplify complex expressions.

Engine.cs

```
        bool rowBorder = cell.Row == LABYRINTH_SIZE - 1 || cell.Row == 0;
        bool columnBorder = cell.Column == LABYRINTH_SIZE - 1 || cell.Column == 0;
        if (rowBorder || columnBorder)
        {
            exitFound = true;
        }
```

Inserted empty lines between the methods.

Result.cs

```
public int MovesCount
{
    get
    {
        return this.movesCount;
    }
}
public string PlayerName
{
    get
    {
        return this.playerName;
    }
}
```

➔

Player.cs

```
/// <summary>
/// Represent the name of the player
/// </summary>
/// <exception cref="ArgumentNullException">
/// If the name is null or empty string</exception>
public string Name
{
    get
    {
        return this.name;
    }

    private set
    {
        if (string.IsNullOrEmpty(value))
        {
            throw new ArgumentNullException(
                "Invalid input! Name cannot be null or empty!");
        }

        this.name = value;
    }
}
```

Split the lines containing several parameters into several simple lines, e.g.:

Ladder.cs

```
Console.WriteLine("{0}. {1} --> {2} moves", index + 1,
                        topResults[index].PlayerName, topResults[index].MovesCount);
```

➔

Scoreboard.cs

```
                    string currentResult = string.Format(
                        "{0}. {1} --> {2} moves",
                        index + 1,
                        this.topPlayers[index].Name,
                        this.topPlayers[index].MovesCount);
                    resultList.AppendLine(currentResult);
```

Naming Convections used: variables and fields: camelCase; types, methods, properties and read-only fields: PascalCase. Constants: ALL_CAPS

Engine.cs

```
public const int LABYRINTH_SIZE = 7;
private readonly int StartRow = LABYRINTH_SIZE / 2;
```

Added input parameters validation for public methods and properties.

Player.cs

```
public string Name
        {
            get
            {
                return this.name;
            }

            private set
            {
                if (string.IsNullOrEmpty(value))
                {
                    throw new ArgumentNullException(
                        "Invalid input! Name cannot be null or empty!");
                }

                this.name = value;
            }

        }
```

Initialized all fields on declaration where possible

Game.cs

```
        private int movesCount = 0;

        private bool isRestart = false;
```