# תרגיל מס' 2 – "בול פגיעה" ל- Console

## מטרות

- הטמעה של עבודה עם מחלקות ותכנות מונחה עצמים
  - עבודה עם מערכים/מחלקות אוספים
    - שימוש במחלקה string
    - שימוש ב- DII (אסמבלי) חיצוני •

## ידע נדרש

- Microsoft Visual Studio .NET היכרות עם כלי הפיתוח
  - C# הכרות עם תחביר בסיסי של שפת
  - עבודה עם מערכים / מחלקות אוספים
- (Access modifiers, Constructors, Properties) עבודה עם מחלקות
  - שימוש במחלקה string
  - שימוש ב- DII (אסמבלי) חיצוני •

## הכינותי מראש

- מותקן על המחשב. Microsoft Visual Studio
- קובץ Ex02.ConsoleUtils.dll שנמצא כחלק מחבילת ההורדה מהאתר.

## התרגיל

עליכם לממש את המשחק "בול פגיעה" ל- Console

## המשחק:

המחשב בוחר רצף של 4 אותיות באופן אקראי מתוך 8 האותיות הראשונות של הא"ב האנגלי (A-H), באופן כזה שאות לא יכולה להופיע פעמיים.

בכל שלב השחקן מנחש מה הוא הרצף שהמחשב בחר, והמחשב נותן לו "פידבק" על הניחוש לפיו על השחקן לשפר את הניחוש הבא עד אשר נגמרים הניחושים או עד אשר ניחש נכונה.

## <u>התוכנית:</u>

תחילה יבקש המחשב מהמשתמש את מספר הניחושים המקסימאלי הרצוי (מספר השורות). מינימום 4, מקסימום 10.

המצב ההתחלתי יהיה לוח שיכיל שורות ריקות לפי מספר הניחושים המקסימאלי, כשבשורה הראשונה מופיעות סולמיות במקום הבחירה של מחשב. (ראה תמונה 1).

בכל שלב יתבקש המשתמש לבחור רצף של 4 אותיות בין A ל- H. אחרי שהמשתמש ניחש, יוצג הלוח שוב עם הבחירה של המשתמש בשורה המתאימה וכן ה"פידבק" מימין לניחוש שלו.

## אופן הניקוד:

אות שמופיעה <u>באותו מיקום ברצף</u> ("פגיעה") תזכה ב- "V" ואות שמופיעה ברצף של המחשב <u>אך לא באותו מיקום ברצף</u> ("פגיעה") תזכה ב- "X". שימו לב, התוצאה לא מציגה מי מהאותיות קיבלה V או X. התוצאה תמיד מיושרת לצד שמאל ותמיד מוצגים קודם ה-"V" ואחריהם ה- "X". (ראה תמונה 2). אם המשתמש ניחש נכונה, תוצג הודעה ושאלה אם לשחק שוב. (ראה תמונה 3). אם נגמרו למשתמש הניחושים תוצג הודעה מתאימה ושאלה אם לשחק שוב. במצב הזה, תיחשף למשתמש הבחירה של המחשב. (ראה תמונה 4).

אחרי סיום משחק, המשתמש יישאל אם ברצונו להתחיל סיבוב נוסף.

בכל עת, במקום להזין ניחוש ניתן לפרוש מהמשחק ע"י הזנת "Q" (שתציג הודעת "להתראות" והתוכנית תסתיים).

# הוראות כלליות

- יש להשתמש במתודה Next של המחלקה
- יש לבדוק תקינות קלט בכל בקשת קלט מהמשתמש ולהציג הודעה מתאימה במקרה של קלט לא תקין. יש להבדיל בין קלט לא תקין תחבירית לבין קלט לא תקין עניינית **–** 
  - ס קלט לא תקין תחבירית למשל: קלט שאמור להיות מספרי אבל הוא לא.
  - ס קלט לא תקין עניינית למשל: מספר שהוא מחוץ לתחום. או אות מחוץ לתחום
- לפני כל הדפסה של מצב הלוח יש לבצע ניקוי מסך. לשם כך, יש להיעזר בקובץ שמכילה Ex02.ConsoleUtils.Screen שמכיל את מחלקת השירות Ex02.ConsoleUtils.dll את המתודה הסטטית – (Clear).

הדרך לפניה ל-dll שלי מתוך הקוד שלכם היא פשוטה:

יש למקם את ה-dll בתיקייה C:\Temp בתיקייה dll. (חשוב מאוד. לא לשים במקום אחר!)

ללחוץ כפתור ימני של עכבר בחלון ה-Solution Explorer, מעל ה-References של הפרויקט. לבחור באופציה Add Reference, ואז לבחור ב-dll באמצעות האפשרות Browse.

כעת, ניתן לגשת ל-Namespace שמוגדר בתוך ה-dll ולהפעיל מתודות מתוך ה-All באופן זהה להפעלת מתודות ספריה רגילות.

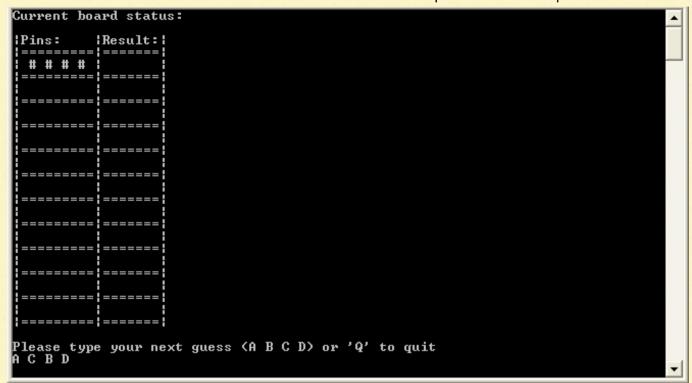
חשוב: אין לצרף את הקובץ הזה בהגשה (אחרת GMAIL ידחה את העבודה שלכם). (לבודק יש את הקובץ אצלו במקום זהה)

## ארכיטקטורה והנדסת תוכנה:

- יש להשתמש בתכנות מונחה עצמים! בחירה נכונה של מחלקות וחלוקה נכונה ⊙ למתודות מהווים נקודות חשובות בתרגיל זה.
  - יש להפגין שימוש נכון ביכולות 2.0 #C ודוט נט. ⊙
    - <u>אין</u> להשתמש בהורשה / פולימופריזם
- יש לבצע <u>הפרדה</u> מתאימה בין המחלקות שמנהלות את הלוגיקה והנתונים של המשחק לבין המחלקות שמציגות את מצב המשחק ומייצרות אינטראקציה עם המשתמש. במילים אחרות: הפרדה בין המימוש של ממשק המשתמש (UI) למימוש של הלוגיקה של המערכת. קחו בחשבון שחלק מהמימושים אמורים לשרת אתכם גם בשלב שבו נרצה לפתח את המשחק לסביבת "חלונות" והשאיפה היא להשאיר כמה שיותר חלקים ללא שינוי!
  - ניתן להעזר בקבוצת הפייסבוק של הקורס כדי לשאול שאלות בנוגע לתרגיל.
  - יש לעמוד בתקנים לכתיבת קוד כפי שמפורט במסמך הרלוונטי שניתן למצוא באתר הקורס. שימו לב בעיקר לתקנים לגבי שמות של שדות מחלקה ופרמטרים לפונקציה. נקודות יירדו למי שלא יעמוד בתקנים אלו.
    - יש לעמוד בהוראות ההגשה כפי שמפורט במסמך הרלוונטי שניתן למצוא באתר הקורס. נקודות יירדו למי שלא יפעל באופן מדוייק לפי הוראות אלה.
- כדי להימנע ממצב שגוגל חוסם את ההגשות שלכם מלהגיע לתיבת ההגשות של בודק התרגילים, יש לזכור לא רק למחוקת את תיקיות ה- bin/obj מהזיפ לפני שאתם שולחים, אלא גם את כל קבצי המטמון של StyleCop.Cache". לדוגמה- StyleCop (במידה והוספתם את StyleCop כ nuget packge לפרויקט <u>יש לדאוג למחוק את תיקיית ה- "packages"</u>)
  - נא להמנע מהעתקות (הן מתגלות מאוד בקלות)

## תמונה 1:

זהו מצב הלוח ההתחלתי, לאחר שהמשתמש בחר את מספר הניחושים המקסימאלי שהוא רוצה שהמחשב יאפשר לו והקליד את הניחוש הראשון שלו:



#### :2 תמונה

זהו מצב הלוח אחרי 8 ניחושים. המשתמש מזין את הניחוש השמיני שלו.

## תמונה 3:

המשתמש הזין ניחוש נכון.

## תמונה 4:

המשתמש לא הצליח לנחש במקסימום הניחושים שהועמדו לרשותו.

Certainly! Designing an Object-Oriented Programming (OOP) project requires careful thought and planning to ensure that the system :is modular, scalable, and maintainable. Here's a step-by-step guide to help you think about the architecture of your project

\*\*:Understand the Problem Domain\*\* .1
.Before diving into OOP design, ensure you have a deep understanding of the problem you are trying to solve .Write down the requirements and expected functionality of your system .Identify potential use cases and user interactions with the system 
\*\*:Identify Main Entities (Objects)\*\* .2
Based on the problem domain, identify the primary entities or objects that will exist in your system. For example, in a library -

Based on the problem domain, identify the primary entities or objects that will exist in your system. For example, in a library - .`system, you might have entities like `Book`, `User`, and `Library

\*\*:Determine Object Attributes (Fields)\*\* .3

List out the properties or characteristics that each object should have. For the `Book` object, attributes could be `title`, `author`, - .`ISBN`, etc

\*\*:Determine Object Behaviors (Methods)\*\* .4

What actions should an object be able to perform? For the `Book` object, methods might include `checkout()`, `returnBook()`, or - .`()`getDetails

\*\*:Establish Relationships\*\* .5

:Understand how objects relate to each other. There are several types of relationships -

.`Association:\*\* A generic relationship between objects. E.g., a `Library` has `Books\*\* -

Aggregation:\*\* A "whole-part" relationship but with a weaker connection. E.g., a `Library` has a `ReadingRoom` but the\*\* - .reading room can exist without the library

Composition:\*\* A stronger "whole-part" relationship. If the whole is destroyed, the part is as well. E.g., a `Library` has a\*\* -

.`Catalog`, and without the library, the catalog doesn't make sense

Inheritance:\*\* A "is-a" relationship where one class inherits properties and methods from another. E.g., `Ebook` is a type of\*\* - .``Book

\*\*:Identify Patterns and Principles\*\* .6

:Look for OOP principles and patterns that can be applied to your design -

.Encapsulation:\*\* Bundle the data (attributes) and methods that operate on the data into single units or classes\*\* -

.Polymorphism:\*\* Allow objects of different classes to be treated as instances of the same class through inheritance\*\* -

.Inheritance:\*\* As mentioned, this is about creating a new class based on an existing class\*\* -

SOLID principles:\*\* A set of design principles that help in making the software design more understandable, flexible, and\*\* - .maintainable

\*\*: Sketch Out the Architecture \*\* .7

Use UML (Unified Modeling Language) diagrams or similar tools to sketch out the class relationships, attributes, and methods. - .This can provide a clear visual of your architecture

\*\*:Refine and Review\*\* .8

.After your initial design, revisit the architecture, refine objects, relationships, attributes, and methods -

.Consider any edge cases or scenarios you might have missed -

.Get feedback from peers or mentors to ensure that the design is solid -

\*\*:Prototype\*\* .9

It can be beneficial to prototype some parts of your system to validate your design decisions. This doesn't need to be fully - .functional but can help in understanding potential pitfalls or design flaws

\*\*:Implementation\*\* .10

.Once you're satisfied with the architecture, begin implementing the classes and methods as defined in your design - .Remember to write tests for your code to ensure that the implemented methods behave as expected -

\*\*:lterate\*\* .11

As you implement, you might find that some design decisions need adjustment. This is normal. Design is iterative, and real-world - .coding can often bring clarity to the best architectural choices

Remember, while these steps provide a structured approach, the design process can be iterative, and you may need to loop back to !previous steps as you gain a better understanding of the problem and its requirements. Good luck with your project

