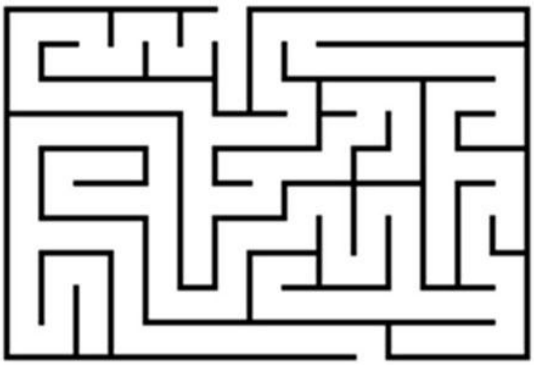
תכנות מתקדם 2 – תרגיל מס' 1

בתרגיל זה נבנה מימוש למשחק של מבוכים.

המשתמש במערכת יוכל לבקש יצירה של מבוך חדש, לקבל פתרון של מבוך, וכן לשחק במבוך נתון כנגד שחקן אחר.



כמו-כן, נרצה לבנות את התרגיל בצורה של שרת/לקוח כאשר השרת יכול לטפל בהרבה לקוחות במקביל.

אופן ייצוג המבוך:

המבוך מיוצג כמערך דו-מימדי של תאים. כל תא יכול להיות ריק (0) או חסום ע"י קיר (1).

דוגמא:

```
0000010001
0111110101
0000010101
0111010101
0001000101
1101111101
0101000101
0101110101
1*01#00000
1111111111
```

נקודת הכניסה למבוך מסומנת ע"י * ונקודת היציאה ע"י #.

לרשותכם שתי ספריות קוד מוכנות בשם MazeLib.dll ו- MazeGeneratorLib.dll.

ספריית הקוד MazeLib.dll מכילה את הטיפוסים הבאים המשמשים לייצוג מבוך:

1. המבנה Position – מייצג מיקום של תא במבוך, מכיל את המאפיינים Row ו- Col.
2. enum CellType – מייצג את סוג התא במבוך (Free או Wall).
3. enum Direction – מייצג כיוון תנועה במבוך (Left, Right, Up, Down).
4. המחלקה Maze – מייצגת את המבוך עצמו.

המחלקה Maze מכילה את המאפיינים הבאים:

הסבר	שם המאפיין
שם המבוך	string Name
מספר השורות במבוך	int Rows
מספר העמודות במבוך	int Cols
מיקום הכניסה למבוך	Position InitialPos
מיקום היציאה מהמבוך	Position GoalPos

כמו-כן המחלקה Maze מכילה את המתודות הבאות:

הסבר	שם המתודה
בנאי המקצה מטריצה של תאים בגודל rows x cols. כל תא הוא מטיפוס CellType.	Maze(int rows, int cols)
אינדקסר שמחזיר את סוג התא במיקום (row, col)	CellType [int row, int col]
מחזירה ייצוג מחרוזתי של המבוך	string ToString()
מחזירה את מאפייני המבוך בייצוג JSON	string ToJSON()
מייצרת אובייקט Maze עפ"י מחרוזת JSON נתונה	Maze FromJSON(string str)

ספריית הקוד MazeGeneratorLib.dll מכילה מימוש של אלגוריתם ליצירת מבוך. ספריה זו מכילה את הטיפוסים הבאים:

1. IMazeGenerator – ממשק ליצירת מבוכים. מגדיר את המתודה

Maze Generate(int rows, int cols)

2. DFSMazeGenerator – מחלקה המממשת את הממשק IMazeGenerator באמצעות אלגוריתם DFS ליצירת מבוכים.

ניתן למצוא הסבר כיצד האלגוריתם הזה עובד בויקיפדיה:

https://en.wikipedia.org/wiki/Maze_generation_algorithm#Depth-first_search

חשבו מה היתרון בלפצל את הטיפוסים הנ"ל לשני dll-ים שונים במקום לממש אותם ב- dll אחד?

משימה א' – אלגוריתם לפתרון מבוך (מפסיאודו-קוד ל- OOP)

צרו ספריית קוד (Class Library) בשם SearchAlgorithmsLib.

1. ממשו את התשתית עבור הגדרה של בעיית חיפוש ואלגוריתמים לפתירת בעיות חיפוש, בהתאם להסבר במצגת של הרצאה מס' 2 (הממשקים ISearchable, ISearcher, המחלקה האבסטרקטית Searcher וכו').
2. השלימו את המימוש של אלגוריתם BFS המופיע בשקפים 59-60.
3. ממשו אלגוריתם חיפוש נוסף בשם DFS. תוכלו למצוא פסיאודו-קוד שלו בויקיפדיה: https://en.wikipedia.org/wiki/Depth-first_search

הערה: לצורך מימוש האלגוריתם אתם רשאים להוריד ספריית קוד מוכנה מהרשת הממשת תור קדימויות (Priority Queue). עדיף להשתמש בספריות שארוזות כ- NuGet Packages (מדוע?).

צרו עתה פרויקט מסוג Console Application.

צרו בתוכו Object Adapter שמבצע אדפטציה ממבוך (מופע של Maze) לבעיית חיפוש (Searchable).

במחלקה Program הגדירו מתודה בשם CompareSolvers אשר:

1. יוצרת מבוך באמצעות DFSMazeGenerator
2. מדפיסה אותו
3. פותרת אותו באמצעות BFS
4. פותרת אותו באמצעות DFS
5. מדפיסה למסך כמה מצבים כל אלגוריתם פיתח. עליכם ליצור מבוך מספיק גדול כדי שתחושו בהבדל.

משימה ב' – כתיבת שרת/לקוח

השרת ממתין כל הזמן לקבלת חיבורים מלקוחות חדשים. מספר הפורט שהוא מאזין לו יוגדר בקובץ הגדרות (קובץ app.config ה). השרת צריך לשרת את כל הלקוחות המתחברים אליו בו זמנית. לאחר שקליינט התחבר לשרת הוא יכול לשלוח לו בקשות.

כאשר השרת מקבל בקשה מלקוח הוא מקבל אותה בצורה של טקסט. הוא אינו מכיר (כלומר, אינו כבול) לסט פקודות ספציפי. בפרט, הוא אינו מנסה להבין מה הלקוח שלח לו באמצעות if או switch או משהו דומה. במקום זאת, השרת בודק בעזרת Dictionary איך יש לטפל בבקשה בעזרת תבנית העיצוב של ה-Command.

הטיפול בבקשות השונות שמגיעות מהקליינטים ינוהל ע"י Thread Pool (באמצעות שימוש ב- Task Parallel Library).

את השרת עליכם לכתוב בתצורת MVC בהתאם למוסבר במצגת שיעור מס' 3.

פרוטוקול התקשורת בין הלקוח לשרת

להלן הפקודות שהלקוח יכול לשלוח לשרת:

תיאור	פקודה
הפקודה תגרום לשרת לייצר מבוך בגודל rows x cols. המבוך שיווצר יישמר בשם name.	generate <name> <rows> <cols>
פקודה זו תגרום לשרת לפתור את המבוך בשם name באמצעות האלגוריתם algorithm. כאשר השרת פותר מבוך הוא שומר את הפתרון, כך שבפעם הבאה לא יצטרך לפתור אותו שוב (כלומר הוא מחזיק סוג של cache של פתרונות).	solve <name> <algorithm> algorithm: 0 – for BFS (Best First Search) 1 – for DFS (Depth First Search)
הלקוח מבקש לקיים משחק בשם name מול שחקן נוסף. המשחק יהיה לפתור מבוך בגודל rows x cols. כאשר השחקן השני ישלח הודעת join לשרת – השרת ישלח לשניהם הודעה והם יתחילו לשחק אחד מול השני.	start <name> <rows> <cols>
מציגה את רשימת שמות המשחקים שניתן להצטרף אליהם	list
הלקוח מבקש להצטרף למשחק בשם name.	join <name>
השחקן מעדכן את השרת שהוא ביצע מהלך במבוך שלו (פקודה זו רלוונטית רק במקרה של שחקן מול שחקן יריב).	play <move> <move> - up/down/left/right
סיום משחק multiplayer בשם name	close <name>

התשובות שהשרת שולח ללקוח יהיו בפורמט JSON. אתם יכולים להיעזר בספריה Json .NET על-מנת לייצר את הודעות ה- JSON ישירות מתוך אובייקטים ב- #C. להלן קישור לספריה (ניתן להוריד אותה גם דרך NuGet Package Manager):

http://www.newtonsoft.com/json

להלן הסבר לפורמט של התשובות ודוגמאות.

עבור פקודה מס' 1 (generate)


```
{
  "Name": "mymaze",
  "Maze":
"0001010001010101011101010100011101011101000100010101111101010000000101011111110101000000000011111111111",
  "Rows": 10,
  "Cols": 10,
  "Start": {
    "Row": 0,
    "Col": 4
  },
  "End": {
    "Row": 2,
    "Col": 8
  }
}
```


כאשר Name הוא שם המבוך, Maze הינו ייצוג של המבוך המורכב מ-0/1 בלבד בצורה של מחרוזת רציפה המייצגת את המבוך משמאל לימין מלמעלה למטה, Rows הוא מספר השורות במבוך, Cols מספר העמודות במבוך, Start הינו התא במבוך שממנו מתחילים ו- End היא נקודת הסיום.

עבור פקודה מס' 2 (solve)


```
{
  "Name": "mymaze",
  "Solution": "3333113300000033111111111222222",
  "NodesEvaluated": 314
}
```


כאשר Name הוא שם המבוך, Solution הוא ייצוג של הפתרון המורכב מסדרת הצעדים שיש לבצע מנקודת ההתחלה כדי להגיע לנקודת הסיום (0 – ללכת שמאלה, 1 – ימינה, 2 – למעלה, 3 – למטה), ו- NodesEvaluated מציין את מספר הצמתים שאלגוריתם החיפוש פיתח בגרף.

עבור פקודה מס' 3 (start)

במקרה זה השרת לא מחזיר תשובה מידית. רק לאחר ששחקן נוסף הצטרף למשחק, השרת ישלח לשני השחקנים תשובה בפורמט הבא:


```
{
  "Name": "mygame",
  "Maze":
"0100010001010101010101010001010101111101000100010101110111010100010001010101011111100010000011111111111",
  "Rows": 10,
  "Cols": 10,
  "Start": {
    "Row": 6,
    "Col": 6
  },
  "End": {
    "Row": 7,
    "Col": 2
  }
}
```

כאשר Name הוא שם המשחק, Maze הוא בפורמט שהוצג עבור פקודה מס' 1 (כלומר מבוך, מספר שורות, מספר עמודות, נקודת התחלה וסיום).

עבור פקודה מס' 4 (list)


```
[
  "mygame",
  "game2",
  "game3"
]
```


פקודה זו מחזירה את רשימת שמות המשחקים שניתן להצטרף אליהם.

עבור פקודה מס' 5 (join)

במקרה זה השחקן יקבל אותה התשובה כמו התשובה שמקבל השחקן שפתח את המשחק בפקודה מס' 3:


```
{
  "Name": "mygame",
  "Maze":
"0100010001010101010101010001010101111101000100010101110111010100010001010101011
1110001000001111111111",
  "Rows": 10,
  "Cols": 10,
  "Start": {
    "Row": 6,
    "Col": 6
  },
  "End": {
    "Row": 7,
    "Col": 2
  }
}
```


עבור פקודה מס' 6 (play)

בכל פעם שהשרת מקבל עדכון שאחד השחקנים זז בלוח – הוא מעדכן את השחקן השני באמצעות שליחת הודעה בפורמט הבא:


```
{
  "Name": "mygame",
  "Direction": "right"
}
```


כאשר Name הוא שם המשחק ו- Direction יכול להיות left/right/up/down.

עבור פקודה מס' 7 (close)

כאשר אחד השחקנים יבקש לסיים את המשחק ע"י שליחת פקודת close, השרת יעדכן את השחקן השני שהמשחק הסתיים (למשל ע"י שליחת אובייקט JSON ריק).

צורת עבודה:

את התרגיל ניתן להגיש בזוגות. ניתן, אך לא חובה.

את התרגיל יש לעשות בעזרת SOURCE CONTROL. מומלץ להשתמש ב- git בעזרת bitbucket. גם אם אתם עובדים לבד, חובה לעבוד ולהשתמש בעזרת - source control. צורת העבודה איתו נלמדה ותורגלה בקורס תכנות מתקדם 1. עם זאת, במודל תוכלו למצוא קישורים שיסייעו לכם, בפרט הדגמה של עבודה איתו ב- VS.

את התרגיל כולו כותבים ב- #C. אין להשתמש בספריות אשר מחייבות התקנה נוספת של דברים מעבר ל- Net Framework. בפרט, לא ניתן להשתמש בספריות או קוד מוכן אשר עושה עבורכם דברים שאתם אמורים לממש (פרט לשימוש ב- PriorityQueue). למשל, אסור להשתמש בספריה שפותרת למשל בעיות BFS או קוד מהאינטרנט שעושה זאת.

לתרגיל ישנו פורום ייעודי בפיאצה. את כל השאלות יש לשאול אך ורק דרך הפורום במודל. חובה להתעדכן בפורום. כל הנאמר בו מחייב את כולם.

עליכם לכתוב את הקוד ע"פ ה Naming Convention המקובלים של שפת #C. פירוט תוכלו למצוא בקישור הבא:

https://msdn.microsoft.com/en-us/library/ms229002(v=vs.110).aspx

עליכם להגיש את כל הקוד של הפרויקט מתועד ע"פ הסטנדרט הבא:

https://msdn.microsoft.com/en-us/library/aa288481%28v=vs.71%29.aspx

צורת הגשה:

עליכם להגיש קובץ zip שיכיל 3 תיקיות:

1. תיקיית src: אשר תכיל את כל הקוד של ה- Solution הכולל את כל הפרוייקטים.
2. תקיית exe: אשר תכיל את קבצי ההרצה (קבצי exe) של כל אחד מהפרוייקטים – כל אחד בתיקיה נפרדת. במקרה שלנו את קובץ ההרצה של הקליינט ושל הסרבר (כמובן בנפרד), וכמובן את הקבצים הנלווים הנדרשים כמו ה-dll-ים וקובץ ההגדרות. שימו לב שעל הקבצים להיות standalone ולא להסתמך על קבצים מחוץ לתיקייה.
3. תיקייה בשם etc: אשר תכיל קובץ בשם info.txt אשר יכיל את שמות המגישים, ת"ז, קבוצת תרגול של כל אחד. בנוסף, היא תכיל את ה log של העבודה מול ה- git.

כל קבצי ההרצה יורצו על מערכת windows עם net framework. חדש אשר אמור לתמוך בכל הגרסאות אחורה. מחובתכם לוודא שכל הקבצים שהגשתם תקינים, ובפרט שהגשתם את כל הקבצים הנדרשים. מומלץ לוודא שהקבצים שלכם רצים תקין במחשב שונה משלכם טרם ההגשה.

לא ניתן יהיה להגיש קבצים מחדש או רטרואקטיבית או תיקונים או שינויים לקבצים לאחר מועד ההגשה.

רק אחד מבני הזוג מגיש את התרגיל. **חובה** להגיש אך ורק לקבוצת התרגיל שאתם רשומים אליה.

את התרגיל יש להגיש עד ה- 20.4 בשעה 22:00 דרך המודל.

בפועל המערכת תאפשר הגשה עד 23:59 של היום הנקוב, אך הגשה לאחר 22:00 היא "על אחריותכם" בלבד. לא ניתן להגיש באיחור כלל.

בהצלחה

רועי ואלי