



פרויקט טריוויה – מסמך עבודה

רקע

ברוכים הבאים וברוכות הבאות!

הגענו לרגע שבו מה שלמדנו בסמסטר א', והפרויקטים שבהם התנסנו הכינו אותנו מספיק כדי להתחיל לעבוד על פרויקט בסדר גודל שלא ראינו עד עכשיו...

את הפרויקט נעשה בזוגות, וכל אחד או אחת מחברי הצוות יתנסה בכל סוגי הפיתוח, החל מהרמת התשתיות ועד העיצוב הגרפי של ה-GUI, מהיום הפכנו להיות מפתחי full stack במשרה מלאה.

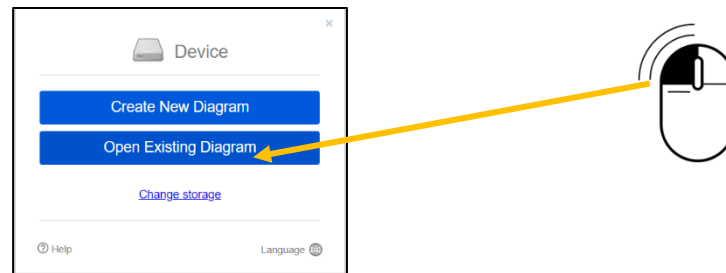
עד סוף הסמסטר נהיה עסוקים בבניית משחק טריוויה, ע"פ UML והנחיות כלליות.



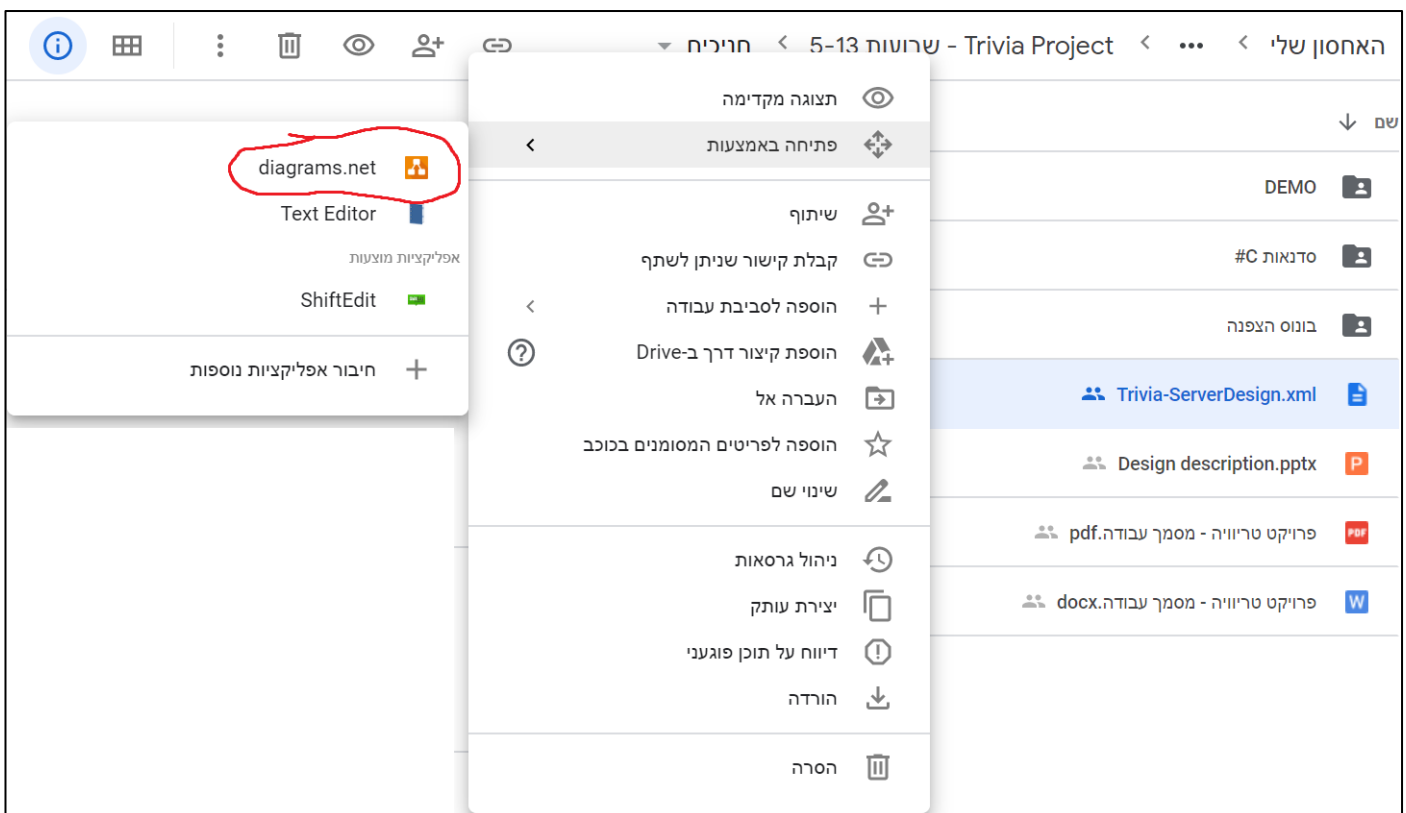
גרסאות עבודה ו-UML

ראינו שהפרויקט הזה ממש לא קטן, ושמאוד קל ללכת לאיבוד. מהסיבה הזו הפיתוח של הפרויקט מחולק לגרסאות שבכל אחת מהן נרחיב את התשתיות עד שבסופו של דבר נסיים עם פרויקט יעיל, מודולרי וקל להרחבה. בתוך תיקיית התכולה של פרויקט הטריוויה תוכלו למצוא את ה-UML, שמור כקובץ XML שאותו ניתן לפתוח בצורה הבאה:

1. הורידו את הקובץ Trivia-ServerDesign.xml מתיקיית הפרויקט.
2. היכנסו לאתר <https://www.draw.io>
3. לחצו על "Open Existing Diagram" ובחרו את קובץ ה-Trivia-ServerDesign.xml שהורדתם/ן.



לחלופין, ניתן ללחוץ right click על קובץ ה-XML ← פתיחה באמצעות ← diagrams.net



אפשרות זו עדיפה מכיוון שככה תמיד נפתח את ה-UML המעודכן.

את הפרויקט ננהל ב-gitlab repository בדומה למה שעשינו בגלריה.
זכרו לעבוד בצורה הנכונה ולהקפיד על העקרונות שלמדנו:

- לא דוחפים קוד ל-Master (ה-Master הוא קדוש 🙏), רק המדריך/ה מאשר/ת את הקוד שנכנס לשם.
- Commit Early, Commit Often...
- על כל פיצ'ר, תיקון באגים או patch נפתח branch ובסוף העבודה נעשה Merge Request ל-Develop
- בכל פעם שחבר הצוות או חברת הצוות עשו MR ל-Develop נעשה **Code Review**
- נשתף את המדריך/ה והאחב"ג בדרגת Maintainer.

חלוקת משימות

בכל גרסה ישנן מס' משימות שנוגעות בכל מיני תחומים, לפני שמתחילים לעבוד חשוב לשבת עם המדריך ולאשר מולו את המשימות, בנוסף יש **לפתוח issue** ב-gitlab ולסמן את ה-Assignee – כלומר מי שעושה את המשימה.

סיכום גרסה

כאשר סיימנו גרסה גדולה (גרסה 1, 2, 3, 4, 5) נעשה (כל חברי הצוות) שיחת סיכום גרסה עם המדריך שבה יקרו כמה דברים:


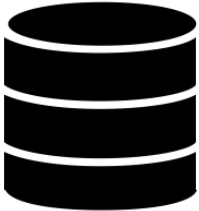
1. נסביר למדריך/ה על פתרונות מיוחדים שהיו לנו, בונסים שעשינו ובאופן כללי איך הייתה העבודה על הגרסה. חשוב שהמדריך יידע בדיוק איפה השקענו, אם יש באגים קטנים, ואם סטינו מה-Design.
2. נכין Demo שבו נראה שאנחנו עומדים בכל מה שדרשו מאיתנו בגרסה. במידה ואנחנו לא בטוחים באיזה פיצ'רים או מקרים צריך לתמוך, ניתן להיעזר [בתיקיית Demo](#) שנמצאת בתיקיית הפרויקט, שם יש אפשרות להריץ דוגמא של פרויקט טריוויה (שרת ולקוח) ולראות במה תומכת התוכנית.
3. נצא למשימות של הגרסה הבאה, ונפתח issues עבור המשימה הבאה שלנו.

עבודה בצוות

כמו בפרויקט השחמט, כמו שיהיה בשנה הבאה וגם כמו שתצטרכו להתנהל שתמשיכו לצבא או לעבודה, חשוב שתדעו איך עובדים בצוות.
תנסו תמיד לתמוך, לעזור ולהבין את חבר/ת הצוות השני/ה.
אם אנחנו בתקופה לחוצה של בגרריות תבקשו מחבר או חברת הצוות לקחת משימות גדולות יותר, ומתי שניהיה פנויים אנחנו ניקח משימות גדולות יותר.
בסוף תראו שהתוצר הסופי לא יהיה דומה לשום דבר שראיתם/ן, אין גבול לרמת האיכות ולמגוון הפיצ'רים שתוכלו להטמיע וזה גם יהיה כיף 😊




גרסה	משימה	תיאור המשימה ודרישות המערכת
1.0.1	<p>יצירת שרת MT</p> 	<p>המשימה הראשונה שלנו הפרויקט נרים את התשתית הבסיסית של השרת. המטרה היא להגיע למצב שיש לנו שרת שמקבל לקוחות ושאפשר להחליף הודעות "Hello" בין השרת ללקוח.</p> <p>דרישות:</p> <ul style="list-style-type: none"> - כל לקוח מקבל thread נפרד - במקביל לשירות הלקוחות, אפשר להקליד פקודות לשרת ב-Console ואם מקלידים "EXIT" התכנית נסגרת. - כל לקוח שמתחבר לשרת מקבל הודעת "Hello", לאחר מכן שולח הודעת "Hello" בחזרה שאותה נדפיס (5 תווים) - ההודעות לא צריכות להיות במבנה שראינו במצגת אלא טקסטואליות (לא צריך להקפיד על הפרוטוקול בינתיים). - על כל פונקציית מערכת (socket, bind, listen, accept, read, write) צריכה להיות בדיקת קלט והדפסת שגיאה - יש ליצור ממשק ריק שנקרא IRequestHandler וממנו יירשו כל ה-Handlers שיבואו בגרסאות הבאות. - יש ליצור מחלקת Handler ריקה בשם LoginRequestHandler - כל לקוח שהתחבר (accept) יירשם בתוך std::map שבשרת, כלומר יש להכניס את ה-socket שלו וליצור מופע חדש של LoginRequestHandler. <p>הערות</p> <ul style="list-style-type: none"> - כמעט כל מה שדרוש לגרסה כבר נעשה בתרגיל 13, כדאי להתחיל משם, חבל לכתוב דברים שוב. - כדאי להיעזר במפת ה-thread-ים שבמסמך design (ה-UML)
1.0.1	<p>בניית סקריפט python שמדמה לקוח</p> 	<p>כדי לבדוק את השרת שלנו נכתוב סקריפט בשפת Python שיממה לקוח. בסקריפט תצטרכו ליצור socket, להתחבר לשרת באמצעות פורט מוסכם, ולשלוח הודעת Hello</p> <p>דרישות:</p> <ul style="list-style-type: none"> - על כל פונקציית מערכת (socket, bind, listen, accept, read, write) צריכה להיות בדיקת קלט והדפסת שגיאה. - לא ניתן לקלוט מס' פורט שלא בטווח 65535 – 1024 - אחרי שנוצר חיבור מוצלח התוכנית תקלוט 5 תווים ותדפיס אותם במידה והתווים הם המחזורות "Hello" התכנית תחזיר "Hello" בחזרה. <p>הערות</p> <ul style="list-style-type: none"> - עדיין לא צריך להקפיד על הפרוטוקול, רק לשלוח 5 תווים של "Hello"

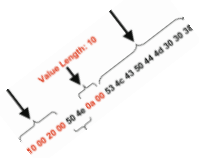

<div data-bbox="1329 604 1409 645" data-label="Text">1.0.2</div> <div data-bbox="1098 450 1295 607" data-label="Text"> <p>תמיכה ב- Serialization של הודעות יוצאות מהשרת</p> </div> <div data-bbox="1109 663 1249 790" data-label="Text"> <p>byte-stream 01110000 01101001 01100001</p> </div>	<p>גרסה 1.0.2 מתמקדת בפרוטוקול ומבנה ההודעות. במשימה זו תצטרכו ליצור את מחלקת JsonResponsePacketSerializer שמבצעת סריאליזציה לתגובות שיוצאות מהשרת לכיוון הלקוחות, כלומר ממירות אובייקט של Response (struct) לרצף בתים (שלו או קוראים buffer) והוא בנוי בצורה שהראינו במצגת (פרוטוקול).</p> <p>תוכן ההודעות צריך להיות בפורמט JSON שמחזיק attribute מסוג status עם ערך מספרי שמתאר קוד הצלחה/כישלון, בנוסף תהיה תמיכה בהחזרת שגיאה, עם הודעה אינפורמטיבית על השגיאה שקרתה.</p> <p>דרישות:</p> <ul style="list-style-type: none"> - מחלקה סטטית - התוצר הסופי של המשימה הוא מערך בתים או std::vector של בתים (בית unsigned char = במבנה שראינו במצגת (קוד, גודל, תוכן הודעה), שאותו נשלח ללקוח. - התו/בית הראשון ב-buffer צריך להיות קוד ההודעה (קוד שהחניכים בוחרים) - 4 התווים הבאים אחרי קוד ההודעה מייצגים את גודל ההודעה (גודל ה-JSON בתווים) - יש לתמוך בפונקציה serializeLoginResponse שתקבל LoginResponse - יש לתמוך בפונקציה serializeSignUpResponse שתקבל SignUpResponse - תמיכה בפונקציה serializeErrorResponse שתקבל ErrorResponse - תוכן ההודעה שב-buffer (ה-JSON) צריך להיות מהצורה: <ul style="list-style-type: none"> o {message: "ERROR"} במקרה של ErrorResponse o {status: 1} במקרה של LoginResponse או SignUpResponse - יש ליצור struct שמייצג כל Response שמגיע מהלקוח <p>הערות</p> <ul style="list-style-type: none"> - תיזכרו איך ממירים מספר עשרוני לבינארי - בהערות המצגת יש המלצה על ספרייה שתעזור להפוך מחלקה ל-JSON - זכרו לעשות את המשימה בהתאם ל-Design, מה שקשור לסריאליזציה שמים במחלקה JsonResponsePacketSerializer, מה שקשור לתקשורת שמים במחלקה Communicator
<div data-bbox="1329 1628 1409 1668" data-label="Text">1.0.2</div> <div data-bbox="1098 1451 1295 1608" data-label="Text"> <p>תמיכה ב- Deserialization של הודעות נכנסות לשרת</p> </div> <div data-bbox="1090 1686 1281 1836" data-label="Text"> <p>Value Length: 10 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200</p> </div>	<p>במשימה זו תצטרכו ליצור את מחלקת JsonRequestPacketDeserializer שמבצעת די-סריאליזציה לבקשות שמגיעות לשרת מכיוון הלקוחות. בנוסף תצטרכו לבנות את כל מה שקשור לאובייקטים של Request.</p> <p>רצף ההודעות מגיע כמערך בתים או std::vector של בתים שנקרא ישירות מה-socket, במשימה הזו תצטרכו לחלץ את תוכן ההודעה מתוך רצף הבתים, שמגיע בפורמט JSON. המבנה של רצף הבתים המתקבל מהלקוח Request:</p> <p>קוד הודעה (byte) גודל ההודעה (4 bytes) ההודעה עצמה (בגודל שנקרא)</p> <p>דרישות:</p> <ul style="list-style-type: none"> - מחלקה סטטית - כל בקשה שנקראת מה-socket נכנסת ל-RequestInfo struct שבו יש: <ul style="list-style-type: none"> o קוד ההודעה (int) o הזמן שבו התקבלה ההודעה o ווקטור/מערך בתים שבו נמצאת תכולת ההודעה (ה-JSON). - יש לממש את הפונקציות הוירטואליות ב-LoginRequestHandler <ul style="list-style-type: none"> o isRequestRelevant – שבודקת שמדובר בקוד הודעה של Login או SignUp (ולמנוע מצב שמישהו עוקף את החיבור שלנו וניגש ישיר לתפריט למשל) o handleRequest – שתקבל את ה-RequestInfo ותחזיר את התשובה אחרי סריאליזציה, כולל את ה-Handler הבא שמטפל בלקוח. - תמיכה בפונקציה deserializeLoginRequest שתקבל LoginRequest - תמיכה בפונקציה deserializeSignUpRequest שתקבל SignUpRequest - הפונקציות יחזירו struct שמחזיק את ערכי ה-attributes שהיו ב-JSON. - בניית struct לכל Request <p>הערות</p> <ul style="list-style-type: none"> - תיזכרו איך ממירים מספר בינארי לעשרוני

1.0.2	<p>הטמעת Serialization Deserialization בתכנית</p>	<p>החלק הזה נשען על משימות קודמות, וכאן תצטרכו לממש את הפונקציה handleNewClient שב-Communicator. המטרה היא להשתמש במחלקות ה-Serializer וה-Deserializer שבנינו קודם ולהרכיב את הכל ביחד ב-Communicator.</p> <p>דרישות:</p> <ul style="list-style-type: none"> - תמיכה בשליחת הודעות ע"פ המבנה שהצגנו במצגת כרצף בתיים. - תמיכה בקבלת הודעות מהלקוח והפיכתן לאובייקטים שאיתם ניתן לעבוד בתכנית
1.0.2	<p>הרחבת סקריפט python שמדמה לקוח</p> 	<p>במשימה זו נרחיב את הסקריפט שכתבנו בגרסה 1.0.1 ככה שישלח הודעות במבנה שמוגדר במצגת.</p> <p>דרישות:</p> <ul style="list-style-type: none"> - תוכן ההודעות נשלח בפורמט JSON - להודעה יש להוסיף קוד (בגודל בית אחד) וגודל הודעה (4 בתיים) - יש לתמוך בשני סוגי ההודעות <ul style="list-style-type: none"> ○ Login {username: "user1", password: "1234"} ○ Signup {username: "user1", password: "1234", mail: user1@gmail.com} - יש להדפיס את התגובה שהתקבלה מהשרת <p>הערות</p> <ul style="list-style-type: none"> - אין חשיבות לערך (status code) שהתקבל, העיקר שהגיע בפורמט JSON. (בגרסה זו לא ממש אכפת לנו מה עושים עם ההודעות, רק שהן מגיעות בפורמט נכון ועוברות סריאליזציה ודי-סריאליזציה)
1.0.3	<p>בניית DB לתכנית עם תמיכה בטבלת משתמשים</p> 	<p>את החלק הזה ניתן לממש ללא תלות בחלקים אחרים. המשימה היא ליצור מסד נתונים שבו יישמר מידע על המשתמש.</p> <p>דרישות:</p> <ul style="list-style-type: none"> - טבלת משתמשים שתשמור את התכונות הבאות: <ul style="list-style-type: none"> ○ שם המשתמש ○ סיסמא ○ כתובת מייל - שאילתות: <ul style="list-style-type: none"> ○ האם משתמש קיים? ○ האם הסיסמא מתאימה? - פעולות: <ul style="list-style-type: none"> ○ הוספת משתמש חדש ל-DB <p>הערות:</p> <ul style="list-style-type: none"> - לא מממשים את הממשק IDatabase אלא מחלקה יורשת בשם SqliteDatabase

<div>יצירת LoginManager</div>  <div>1.0.3</div>	<p>את החלק הזה מומלץ לתת לחניך/ה שמימש/ה את ה-DB. המשימה היא לממש את מחלקת LoginManager שניגשת ל-DB ובה יש את כל הלוגיקה הקשורה לחיבור משתמשים.</p> <p>דרישות:</p> <ul style="list-style-type: none"> - ניהול וקטור של משתמשים מחוברים (שמחוברים ברגע נתון לתכנית) כשדה במחלקה. - יצירת מחלקה של LoggedUser - תמיכה בפעולות הבאות: <ul style="list-style-type: none"> ○ חיבור משתמש (login) – הפונקציה מקבלת שם משתמש וסיסמא, ניגשת ל-DB כדי לראות אם המשתמש קיים והסיסמא נכונה ומחברת את המשתמש (מוסיפה לרשימת המשתמשים המחוברים) ○ הרשמת משתמש (signup) – הפונקציה מקבלת את כל המידע של המשתמש (שם, סיסמא, מייל) ורושמת אותו ב-DB ○ ניתוק משתמש (הוצאתו מרשימת המשתמשים המחוברים) <p>הערות:</p> <ul style="list-style-type: none"> - הגישה ל-DB נעשית באמצעות המחלקה SqliteDataBase ולא ישירות.
<div>מימוש מכונת המצבים (Factory-Handlers)</div>  <div>1.0.3</div>	<p>את החלק הזה ניתן לממש ללא תלות בחלקים אחרים. המשימה דורשת עבודה עם כמה מחלקות מרכזיות:</p> <ul style="list-style-type: none"> - RequestHandlerFactory – מייצרת את המצבים - LoginRequestHandler – המצב שבו מתחיל כל משתמש – שלב ה-Login או ה-Signup. <p>דרישות:</p> <ul style="list-style-type: none"> - החזרת RequestResult שמכיל את ה-buffer (שיישלח ללקוח) והמצב החדש: <ul style="list-style-type: none"> ○ MenuRequestHandler (ממומש באופן ריק) במידה והמשתמש הצליח להתחבר ○ LoginRequestHandler (אותו מצב) במידה וההתחברות נכשלה ○ nullptr במידה והייתה שגיאה - יש לממש את handleRequest – מקבלת את הבקשה, מוציאה את מידע המשתמש ושולחת ל-LoginManager כדי לבצע התחברות, בסוף גם תחזיר את ה-RequestResult - יש לממש את RequestHandlerFactory שמייצר Handlers, בגרסה זו הוא מייצר את LoginRequestHandler ו-MenuRequestHandler שממומש ריק. <p>הערות:</p> <ul style="list-style-type: none"> - המצב הראשוני שבו נמצא כל משתמש (אחרי שלקוח יצר חיבור עם השרת והשרת קיבל אותו באמצעות accept) הוא LoginRequestHandler. - דרך ה-Factory ניתן לגשת ל-Managers ול-DB.
<div>ביצוע tests באמצעות ה-Python script</div>  <div>1.0.3</div>	<p>במשימה זו תשתמשו ב-Python script כדי לשלוח הודעות Login ו-Signup ולבדוק באמצעותו את תפקוד השרת.</p> <p>דרישות:</p> <ul style="list-style-type: none"> - יש להדפיס את התגובה שהתקבלה מהשרת בכל אחד מהמקרים ולראות שזה נכון <ul style="list-style-type: none"> ○ משתמש חייב להיות רשום כדי להתחבר ○ משתמש לא יכול להירשם עם אותו שם משתמש פעמיים ○ משתמש לא יכול להתחבר אם הוא כבר מחובר ○ בדיקות קלט על שמות משתמש

<div data-bbox="1329 510 1414 548" data-label="Text">2.0.0</div> <div data-bbox="1098 360 1295 472" data-label="Text"> <p>תמיכה בחדרי משחק ומימוש RoomManager</p> </div> <div data-bbox="1098 510 1295 698" data-label="Image"> </div>	<div data-bbox="667 100 1066 129" data-label="Text"> <p>במשימה זו נוסיף תמיכה בחדרי משחק.</p> </div> <div data-bbox="979 165 1066 192" data-label="Section-Header"> <p>דרישות:</p> </div> <div data-bbox="215 197 1018 891" data-label="List-Group"> <ul style="list-style-type: none"> - בכל חדר יישמר מידע על החדר: <ul style="list-style-type: none"> ○ מס' חדר ○ שם החדר ○ מס' מקסימלי של שחקנים ○ זמן ממוצע לשאלה ○ האם החדר פעיל ברגע זה - יש ליצור מחלקה שמייצגת חדר ותומכת בפעולות והמאפיינים הבאים <ul style="list-style-type: none"> ○ ווקטור של שחקנים בחדר ○ הפונקציה addUser שמוסיפה משתמש לחדר ○ הפונקציה removeUser שמסירה משתמש מהחדר ○ הפונקציה getAllUsers שמחזירה רשימה (גם יכול להיות מחרוזת) של השחקנים בחדר. - יש ליצור ולממש את ה-RoomManager שתומך במאפיינים והפעולות הבאות: <ul style="list-style-type: none"> ○ ווקטור של חדרים פעילים (כאלו שיש שחקן שמחכה או שמתנהל בהם משחק) ○ הפונקציה createRoom שמקבלת את שם המשתמש שיצר אותו. ○ הפונקציה deleteRoom שמקבלת מספר חדר ומוחקת את החדר ○ הפונקציה getRoomState שמחזירה את מצב החדר (מחכה להתחיל משחק או שמתנהל בו משחק) ○ הפונקציה getRooms שמחזירה רשימה של החדרים הפעילים בשרת. </div>
<div data-bbox="1329 1500 1414 1538" data-label="Text">2.0.0</div> <div data-bbox="1129 1263 1295 1335" data-label="Text"> <p>הוספה של שאלות ל-DB</p> </div> <div data-bbox="1098 1384 1295 1512" data-label="Image"> </div> <div data-bbox="1098 1556 1295 1767" data-label="Image"> </div>	<div data-bbox="252 1025 1066 1055" data-label="Text"> <p>במשימה זו נכניס שאלות למסד הנתונים, ואלו השאלות שיופיעו במשחק בהמשך.</p> </div> <div data-bbox="979 1122 1066 1149" data-label="Section-Header"> <p>דרישות</p> </div> <div data-bbox="215 1182 1018 1532" data-label="List-Group"> <ul style="list-style-type: none"> - ליצור טבלה שתאחסן שאלות טריוויה אמריקאיות עם 4 תשובות שאחת מהן נכונה. - יש לאפשר הצגה של השאלה עצמה, 4 תשובות, ולדעת מה התשובה הנכונה. - בנוסף, יש להכניס 10 שאלות (עדיף הזויות ומצחיקות 😊), או למצוא מנגנון אוטומטי שיכניס שאלות למסד הנתונים (כלומר סקריפט כלשהו). ○ הנה רמז קטן לאתר שיוכל לעזור לנו במידה ובחרנו לכתוב סקריפט שיעשה את הכנסת השאלות בצורה אוטומטית. https://opentdb.com, לאתר יש API שמאפשר לקבל שאלות רנדומליות, והן חוזרות מהאתר ב... פורמט JSON ! </div>

<div> <div>יצירת StatisticsManager</div>  </div>	<p>המשימה היא לממש את מחלקת StatisticsManager שניגשת ל-DB ובה יש את כל הלוגיקה הקשורה לסטטיסטיקות משחק וטבלת השיאים.</p> <p>דרישות:</p> <ul style="list-style-type: none"> - המחלקה נקראת מתוך MenuRequestHandler וניגשת ל-DB על מנת להוציא סטטיסטיקות משחק. - המידע שחוזר מהמחלקה (ובסופו של דבר ייכנס ל-JSON שיחזור ל-client, צריך לאפשר ל-client להציג: <ul style="list-style-type: none"> o את הסטטיסטיקות האישיות של שחקן (של כל המשחקים ששיחק מאז שנרשם) o טבלת השיאים (5 התוצאות הטובות ביותר), ממוינות באמצעות פונקציית הניקוד שעליה חשבו החניכים כשתכננו את ה-DB. <p>הערות:</p> <ul style="list-style-type: none"> - הגישה ל-DB נעשית באמצעות המחלקה SqliteDatabase ולא ישירות.
<div> <div>הוספה של סטטיסטיקות ל-DB</div>  </div>	<p>במשימה זו נוסיף תמיכה בסטטיסטיקות משחק. החלק הראשון של המשימה הוא תכנון, שצריך להעשות בהסכמה של שני חברי הצוות. צריך להחליט על מה מבוסס ניצחון, ואך לשקלל את הפרמטרים ששמורים במסד הנתונים כדי להכריע מי מנצח.</p> <p>דרישות:</p> <ul style="list-style-type: none"> - לחשוב על נוסחה שבאמצעותה נשקלל את הפרמטרים ונחליט מי תפקד יותר טוב במשחק. - יש ליצור טבלה חדשה ב-DB בשם statistics עם פרמטרים שיעזרו להחליט מי ניצח במשחק - יש לממש שאילתות ופונקציות מתאימות במחלקה SqliteDatabase <ul style="list-style-type: none"> o getPlayerAverageAnswerTime – מחזירה את הזמן הממוצע שלקח למשתמש לענות על שאלה o getNumOfCorrectAnswers – מספר התשובות הנכונות שענה המשתמש בכל המשחקים שבהם השתתף o getNumOfTotalAnswers – מס' התשובות הכולל שענה המשתמש (לא רק הנכונות) o getNumOfPlayerGames – כמה משחקים שיחק המשתמש
<div> <div>תמיכה ב-Serialization של הודעות יוצאות מהשרת</div>  </div>	<p>במשימה זו נוסיף פונקציונליות למחלקה JsonResponsePacketSerializer</p> <p>דרישות:</p> <ul style="list-style-type: none"> - יש ליצור את כל ה-structs הרלבנטיים לתגובות השרת בגרסה זו - יש לתמוך בפונקציה serializeLogoutResponse - יש לתמוך בפונקציה serializeGetRoomResponse - יש לתמוך בפונקציה serializeGetPlayersInRoomResponse - יש לתמוך בפונקציה serializeJoinRoomResponse - יש לתמוך בפונקציה serializeCreateRoomResponse - יש לתמוך בפונקציה serializeHighScoreResponse - תוכן ההודעה שב-buffer (ה-JSON) צריך להיות מהצורה: <ul style="list-style-type: none"> o {status: 1} במקרה של LogoutResponse או o CreateRoomResponse או JoinRoomResponse o {Rooms: "room1, room2, ... roomN"} במקרה של getRoomResponse o {PlayersInRoom: "user1, user2, ... userN"} במקרה של getPlayersInRoomResponse o {UserStatistics: "<...>", HighScores: "<...>"} במקרה של HighScoreResponse כאשר לחניכים ניתן חופש בכל הנוגע להצגת המידע והעיקר שיוכלו לפרסר את זה בצד של הלקוח. <p>הערות</p> <ul style="list-style-type: none"> - יש לנו חופש בכל הנוגע לתוכן ההודעות (ה-JSON) ורק חשוב שהוא יכיל את כל הפרמטרים הדרושים כדי שאפשר יהיה להציג את הדברים ב-GUI של הלקוח. כלומר לא חשוב מה יש ב-JSON העיקר שהצד השני יבין את התגובה.

<p>במשימה זו נוסיף פונקציונליות למחלקה JsonRequestPacketDeserializer</p> <p>דרישות:</p> <ul style="list-style-type: none"> - יש ליצור את כל ה-structs הרלבנטיים לבקשות הלקוח בגרסה זו - יש לתמוך בפונקציה deserializeGetPlayersInRoomRequest - יש לתמוך בפונקציה deserializeJoinRoomRequest - יש לתמוך בפונקציה deserializeCreateRoomRequest <p>הערות</p> <ul style="list-style-type: none"> - ישנן בקשות נוספות שאין צורך לפרסר כי הן לא מכילות data וניתן להסתפק בקוד ההודעה: <ul style="list-style-type: none"> HighScoreRequest ○ LogoutRequest ○ GetRoomRequest ○ 	<p>תמיכה ב- Deserialization של הודעות נכנסות לשרת</p> 	<p>2.0.0</p>
<p>יש להרחיב את הפונקציונליות של ה-Factory כדי שיוכל ליצור את ה-handlers הרלבנטיים לגרסה 2, ולאפשר גישה ל-managers הרלבנטיים.</p> <p>דרישות:</p> <ul style="list-style-type: none"> - יש להוסיף ל-Factory את המתודות: <ul style="list-style-type: none"> createMenuRequestHandler – מחזירה מופע של MenuRequestHandler שיטפל בהודעות הרלבנטיות. Getters ל-RoomManager ול-StatisticsManager - יש ליצור את MenuRequestHandler שירש מ-IRequestHandler ומחויב לממש את הפונקציות: <ul style="list-style-type: none"> isRequestRelevant – בודק שמדובר בקוד הודעה של אחת מהבקשות: <ul style="list-style-type: none"> CreateRoomRequest GetRoomsRequest GetPlayersInRoomRequest JoinRoomRequest GetStatisticsRequest LogoutRequest handleRequest – מקבלת את הבקשה, מוציאה את מידע המשתמש ושולחת ל-Manager המתאים כדי שביצע את הלוגיקה (גישה ל-DB או למבנה הנתונים), בסוף גם תחזיר את ה-RequestResult 	<p>הרחבת מכונת המצבים Factory &) (Handlers</p> 	

בתת גרסה זו נעלה רמה ונשדרג את חווית המשתמש ע"י הוספת GUI. החל מגרסה 2 לכל פיצ'ר שפותח בשרת יהיה מסך מתאים (form) שאיתו מתקשר המשתמש. ה-GUI שיפותח בשפת C# ייצר בקשות נבדומה לתכנית ה-Python שכתבנו בגרסה 1.

אחד הדברים העיקריים שנצטרך לעשות בתת הגרסה זה ללמוד שפה חדשה 🤖
נוכל לעשות זאת בשתי דרכים:

1. להיעזר בסדנאות הלימוד שלנו שנמצאות בתיקיית הפרויקט
 2. ללמוד מהמקורות הרגילים (StackOverflow GeeksforGeeks) ומהתיעוד הרשמי של .NET, בכל מקרה אפשר להיות רגועים, C# היא שפה מאוד נפוצה ויש לה תיעוד מעולה.
- דבר נוסף זה שבשונה מהשרת כאן אין Design... תצטרכו לחשוב על Design משלכם/ן. תזכרו את העקרונות שלמדנו (DRY, KISS, SOLID).

דרישות:

- יש לתמוך בכל המסכים שהוצגו ב-Demo של הטיריוויה (נמצא בתיקייה של הפרויקט)

- מסך Login
- מסך Signup
- תפריט ראשי
 - כפתור יצירת חדר
 - כפתור הצטרפות לחדר
 - כפתור Statistics
 - כפתור יציאה מהתכנית
- מסך CreateRoom
 - הכנסת שם חדר, זמן לשאלה, כמות שחקנים
 - אחרי יצירת החדר ניתן לצפות בשמות המשתמש של השחקנים המחוברים לחדר ומי ה-Admin
- מסך JoinRoom
 - מציג את כל החדרים שאליהם ניתן להתחבר
- אחרי התחברות לחדר ניתן לצפות בשמות המשתמש של השחקנים המחוברים לחדר ומי ה-Admin
- מסך Statistics
 - כפתור לסטטיסטיקות אישיות
 - כפתור לטבלת שיאים
- מסך סטטיסטיקות אישיות לשחקן הנוכחי
 - מציג את מס' הסטטיסטיקות ששמרנו ב-DB עבור השחקן המחובר
- מסך HighScores
 - מציג את 3 השחקנים עם הניקוד הגבוה ביותר


יצירת Client עם
GUI בשפת C#



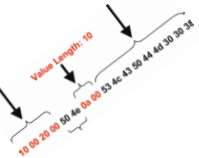


2.0.1

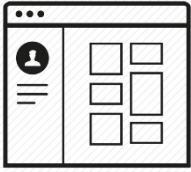
הערות


- יש להתפרע על העיצוב 😊
- בסדנא מס' 5 יש הסבר על Nugets וזה יכול לעזור עם תהליך הסריאליזציה והדיסריאליזציה.
- אפשר לעבוד ב-WPF או ב-WinForm, מה שנוח, העיקר שניהיה עקביים.
- יש לחלק את פיתוח המסכים בין שני חברי הצוות ושלא ייצא מצב שאחד מפתח את הכל.

<p>יש להרחיב את הפונקציונליות של ה-Factory כדי שיוכל ליצור את ה-handlers הרלבנטיים לגרסה 3, ולאפשר גישה ל-managers הרלבנטיים. דרישות:</p> <ul style="list-style-type: none"> - יש להוסיף ל-Factory את המתודות: <ul style="list-style-type: none"> ○ createRoomAdminRequestHandler – מחזירה מופע חדש של RoomAdminRequestHandler שיטפל בהודעות הרלבנטיות. ○ Getters ל-RoomManager ול-StatisticsManager - יש ליצור את RoomAdminRequestHandler שיומש את IRequestHandler ומחויב לממש את הפונקציות: <ul style="list-style-type: none"> ○ isRequestRelevant – בודק שמדובר בקוד הודעה של אחת מהבקשות: <ul style="list-style-type: none"> ▪ CloseRoomRequest ▪ StartGameRequest ▪ GetRoomStateRequest ○ handleRequest – מקבלת את הבקשה, מוציאה את מידע המשתמש ושולחת ל-Manager המתאים כדי שיבצע את הלוגיקה (גישה למבנה הנתונים), בסוף גם תחזיר את ה-RequestResult <ul style="list-style-type: none"> ▪ במקרה של CloseRoomRequest יש לשלוח לכל חברי החדר LeaveRoomResponse כדי שיידעו לעזוב את החדר ▪ במקרה של StartGameRequest יש לשלוח לכל חברי החדר StartGameResponse כדי שיידעו להתחיל את המשחק ▪ במקרה של GetRoomStateRequest יש לפנות ל-RoomManager - יש ליצור את RoomMemberRequestHandler שיומש את IRequestHandler ומחויב לממש את הפונקציות: <ul style="list-style-type: none"> ○ isRequestRelevant – בודק שמדובר בקוד הודעה של אחת מהבקשות: <ul style="list-style-type: none"> ▪ LeaveRoomRequest ▪ GetRoomStateRequest ▪ GetRoomStateRequest ○ handleRequest – מקבלת את הבקשה, מוציאה את מידע המשתמש ושולחת ל-Manager המתאים כדי שיבצע את הלוגיקה (גישה למבנה הנתונים), בסוף גם תחזיר את ה-RequestResult <ul style="list-style-type: none"> ▪ במקרה של CloseRoomRequest יש לשלוח לכל חברי החדר LeaveRoomResponse כדי שיידעו לעזוב את החדר ▪ במקרה של StartGameRequest יש לשלוח לכל חברי החדר StartGameResponse כדי שיידעו להתחיל את המשחק ▪ במקרה של GetRoomStateRequest יש לפנות ל-RoomManager <p>הערות</p> <ul style="list-style-type: none"> - יש כפל קוד, חשבו איך אפשר לפתור את הבעיה בלי לשבור עקרון Design אחר 	<p>הרחבת מכונת המצבים (Factory & Handlers)</p> 	<p>3.0.0</p>
<p>בסוף הגרסה יש להשתמש ב-GUI כדי לבדוק את הפונקציונליות דרישות:</p> <ul style="list-style-type: none"> - יש להקפיץ MessageBox שמראה את התגובה שהתקבלה מהשרת בכל אחד מהמקרים ולראות שהקדנו על הדברים החשובים: <ul style="list-style-type: none"> ○ יש להדפיס את המידע שקשור לחדר (RoomState) ○ רק Admin יכול לסגור חדר או להתחיל משחק ○ כאשר Admin עשה אחת מהפעולות הנ"ל, כל הלקוחות שמחוברים לחדר צריכים לקבל הודעה מתאימה (LeaveRoom/StartGame) ○ חברי החדר מתעדכנים אצל כולם אחרי שמישו עזב 	<p>ביצוע tests באמצעות ה-GUI</p>	<p>3.0.0</p>

<div>3.0.0</div>	<div>הוספת ריענון מסך אוטומטי</div>	<p>חלק מאתגר בגרסה הוא דווקא בצד הלקוח, והוא לרענן את המידע שמוצג ב-GUI בהתאם למידע שמתקבל מהשרת</p> <p>דרישות:</p> <p>רענון המסך רלבנטי בשני מקומות:</p> <ul style="list-style-type: none"> - מסך ה-Join Room, שבו משתמש צריך לראות את רשימת החדרים שאליהם יוכל להצטרף, והרשימה צריכה להתעדכן במידה ונפתחו/נסגרו חדרים. - מסך ההמתנה של החדר, שאליו נכנס המשתמש אחרי שהצטרף לחדר קיים, יש לעדכן את רשימת חברי החדר. - יש לרענן את המסך בכל 3 שניות. <p>הערות:</p> <p>שימו לב, רענון מסך דורש קצת חשיבה מחוץ לקופסא. כדי לממש את רענון המסך אנו ממליצים ליצור thread נוסף בצד של ה-Client שבכל 3 שניות ישלח בקשה מסוג מסוים לשרת. אם יש דרך אחרת שתמצאו לממש אנא התייעצו עם המדריך/ה.</p>
<div>4.0.0</div> <div>תמיכה ב-Serialization של הודעות יוצאות מהשרת</div> <div>byte-stream 01000001 01101101 01110000</div>	<div>תמיכה ב-Serialization של הודעות יוצאות מהשרת</div>	<p>בחלק הזה נוסיף פונקציונליות למחלקה JsonResponsePacketSerializer</p> <p>דרישות:</p> <ul style="list-style-type: none"> - יש ליצור את כל ה-structs הרלבנטיים לתגובות השרת בגרסה זו - יש לתמוך בפונקציה <code>serializeGetGameResultsResponse</code> - יש לתמוך בפונקציה <code>serializeSubmitAnswerResponse</code> - יש לתמוך בפונקציה <code>serializeGetQuestionResponse</code> - יש לתמוך בפונקציה <code>serializeLeaveGameResponse</code> - תוכן ההודעה שב-buffer (ה-JSON) צריך להיות מהצורה: <ul style="list-style-type: none"> ○ במקרה של <code>LeaveGameResponse</code> או <code>SubmitAnswerResponse</code> מחזירים JSON שמכיל {status: 1} ○ במקרה של <code>GetGameResultsResponse</code> מחזירים JSON שמכיל את הפרמטרים הבאים: <ul style="list-style-type: none"> ▪ status – קוד שגיאה/הצלחה <ul style="list-style-type: none"> ◀ במידה והסתיים המשחק אז הקוד יהיה 1 (ומחזירים את התוצאות) ◀ במידה והמשחק עדיין לא הסתיים יש להחזיר 0 (אין עדיין תוצאות) ▪ Results (list<PlayerResults>) – שבה יש רשימה של תוצאות של כל שחקן: <ul style="list-style-type: none"> ◀ Username ◀ CorrectAnswersCount ◀ WrongAnswerCount ◀ averageAnswerTime ○ במקרה של <code>GetQuestionResponse</code> מחזירים JSON שמכיל את הפרמטרים הבאים: <ul style="list-style-type: none"> ▪ status – קוד שגיאה/הצלחה <ul style="list-style-type: none"> ◀ במידה ואין עוד שאלות (השחקן ענה על כל השאלות בחדר) מחזירים 0 ◀ במידה ויש עוד שאלות שהשחקן עדיין לא ענה עליהן במשחק מחזירים 1 (ואת השאלה) <ul style="list-style-type: none"> ▪ question – השאלה עצמה ▪ Answers – מילון של מס' התשובה והתשובה עצמה <p>הערות</p> <ul style="list-style-type: none"> - במידה ששלחו בקשה לתוצאות והמשחק לא הסתיים צריך להחזיר קוד שגיאה ולהחזיר רשימה ריקה של תוצאות. - במידה והמשתמש ענה על כל השאלות וביקש שאלה חדשה צריך להחזיר קוד שגיאה ולהחזיר רשימה ריקה של תוצאות.

<p>בגרסה זו יש להוסיף פונקציונליות ל-JsonRequestPacketDeserializer - יש לתמוך בפונקציה deserializerSubmitAnswerRequest</p> <p>בנוסף, קודים שבהם צריך לתמוך: LeaveGameRequest - GetQuestionRequest - SubmitAnswerRequest - GetGameResultRequest -</p> <p>הערות - משתמש יכול להיות במשחק אחד בלבד, לכן אין צורך להעביר את מזהה החדר בבקשה.</p>	<p>תמיכה ב- Deserialization של הודעות נכנסות לשרת</p> 	<p>4.0.0</p>
<p>יש להרחיב את הפונקציונליות של ה-Factory כדי שיוכל ליצור את ה-handlers הרלבנטיים לגרסה 4, ולאפשר גישה ל-managers הרלבנטיים. דרישות:</p> <ul style="list-style-type: none"> - יש להוסיף ל-Factory את המתודות: <ul style="list-style-type: none"> o createGameRequestHandler – מחזירה מופע של GameRequestHandler שיטפל בהודעות הרלבנטיות. o Getter ל-GameManager - יש ליצור את GameRequestHandler שיוורש מ-IRequestHandler ומחויב לממש את הפונקציות: <ul style="list-style-type: none"> o isRequestRelevant – בודק שמדובר בקוד הודעה של אחת מהבקשות: <ul style="list-style-type: none"> ▪ LeaveGameRequest ▪ GetQuestionRequest ▪ SubmitAnswerRequest ▪ GetGameResultRequest o handleRequest – מקבלת את הבקשה, מוציאה את מידע המשתמש ושולחת ל-Manager המתאים כדי שיבצע את הלוגיקה , בסוף גם תחזיר את ה-RequestResult <ul style="list-style-type: none"> ▪ במקרה של LeaveGameRequest יש לפנות ל-RoomManager שיסיר את המשתמש מהחדר ▪ במקרה של GetQuestionRequest יש לפנות ל-GameManager שיחזיר את השאלה הבאה. ▪ במקרה של SubmitAnswerRequest יש לפנות ל-GameManager שיעדכן את תוצאות המשחק ▪ במקרה של GetGameResultRequest יש לפנות ל-GameManager כדי שיחזיר את התוצאות. 	<p>הרחבת מכונת המצבים Factory &) (Handlers</p> 	<p>4.0.0</p>
<p>המשימה היא לממש את מחלקת GameManager שניגש ל-DB, וגם מנהל מבנה נתונים שמכיל מידע על המשחקים שמתנהלים בשרת. דרישות:</p> <ul style="list-style-type: none"> - המחלקה נקראת מתוך GameRequestHandler ועושה את הדברים הבאים: <ul style="list-style-type: none"> o ניגש ל-DB כדי לקבל שאלות טריוויה (כולל תשובות) רנדומליות o מנהל מבנה נתונים (מילון) שבו המפתח זה שחקן (משתמש) והערך זה מידע על המשחק שהשחקן משחק בו ברגע הנתון. מידע שנשמר: <ul style="list-style-type: none"> ▪ השאלה הנוכחית שעליה עונה השחקן ▪ כמה תשובות נכונות ענה עד עכשיו במשחק הנוכחי ▪ כמה תשובות שגויות ענה עד עכשיו במשחק הנוכחי ▪ זמן ממוצע שלקח לענות על שאלה במשחק הנוכחי. <p>הערות: - במידה ומשתמש פרש מהמשחק התוצאות שלו עדיין יוצגו.</p>	<p>יצירת GameManager</p> 	

4.0.0	<p>הוספת מסכי משחק ל-GUI</p> 	<p>במשימה זו נוסיף מסכים ל-GUI כדי לתמוך בגרסת המשחק.</p> <ul style="list-style-type: none"> - מסך משחק שבו מוצג המידע הבא: <ul style="list-style-type: none"> ○ השאלה הנוכחית (טקסט השאלה) ○ 4 תשובות אפשריות – שאפשר ללחוץ עליהן ○ כמה שאלות נשאר ○ כמה שאלות נכונות ענה עד עכשיו ○ כמה זמן נשאר לענות על השאלה הנוכחית - מסך תוצאות שמוצג אצל כל השחקנים בסוף המשחק ומכיל: <ul style="list-style-type: none"> ○ שמות השחקנים, מס' התשובות הנכונות, זמן ממוצע שלקח לענות על השאלה. ○ מי ניצח (במידה והתוצאות לא ממוינות)
4.0.0	<p>ביצוע tests באמצעות ה-GUI</p>	<p>יש להשתמש ב-GUI כדי לבדוק את הפונקציונליות של הגרסה דרישות:</p> <ul style="list-style-type: none"> - בדיקות בקשר לשאלות <ul style="list-style-type: none"> ○ כל השחקנים מקבלים את אותן שאלות (לא חייב באותו סדר) ○ התוצאות מתעדכנות בצורה נכונה אחרי תשובה נכונה/שגויה - משחק עצמו <ul style="list-style-type: none"> ○ המשחק מתחיל אצל כולם בבת אחת ○ אם מישהו סיים את המשחק הוא מחכה עד שכולם מסיימים ואז מגיעות התוצאות.
4.0.1	<p>בונוס – הוספת שאלה</p>	<p>אם הגעתם/ן לשלב הזה אז עבדתם/ן מדהים 😊 בונוס קצרצר שאולי תשקלו לעשות זה הוספת מסך חדש, שבו ניתן יהיה להכניס שאלה ל-DB.</p> <p>דרישות:</p> <ul style="list-style-type: none"> - יש להוסיף מסך חדש ל-GUI - יש לבקש מהמשתמש את המידע הבא: <ul style="list-style-type: none"> ○ השאלה עצמה. ○ התשובה הנכונה. ○ 3 תשובות נוספות שהן לא נכונות. - השאלה צריכה להירשם ב-DB ועשויה להופיע במשחקי הטריוויה.
4.0.1	<p>בונוס – מצב "ראש בראש"</p>	<p>אם הגעתם/ן לשלב הזה אז עבדתם/ן מדהים 😊 בונוס שתשקלו לעשות הוא הוספת מצב "1 על 1"</p> <p>דרישות:</p> <ul style="list-style-type: none"> - יש להוסיף מסך וכפתור חדשים ל-GUI עבור מצב "ראש בראש" - כשמשתמש נכנס למצב ראש בראש הוא מחכה למשתמש נוסף (אחד) שגם הוא יילחץ על "ראש בראש". - ברגע ששני המשתמשים מחוברים המשחק ה-GUI יקפיץ התראה של 5 שניות ואז שני השחקנים יתחילו את המשחק.

<p>5.0.0</p>	<p>בונוס – הצפנה והוספת פיצ'רי אבטחה. (OTP)</p>	<p>אם נשאר לכם/ן כוח אחרי 4 גרסאות מפרכות, גרסה 5 היא תרגיל מעולה כדי ללמוד קצת על הצפנה. בגרסה זו תוסיפו הצפנה לתקשורת שבין השרת ללקוח, תחילה באמצעות אלגוריתם הצפנה משלכם/ן, ואחר כך בעזרת אלגוריתמי הצפנה נפוצים שכבר מומשו עבורנו.</p> <p>דרישות:</p> <ul style="list-style-type: none"> - יש להוסיף מחלקה בשם CryptoAlgorithm עם שתי מתודות וירטואליות טהורות: להצפנה, ופיענוח <ul style="list-style-type: none"> o <code>std::string encrypt(std::string message)</code> המתודה encrypt מקבלת הודעה (מחרוזת או buffer מהסוג שתמצאו) ומחזירה מחרוזת חדשה = הצופן. o <code>std::string decrypt(std::string cypher)</code> המתודה decrypt עושה את ההפוך ל-encrypt, מקבלת צופן ומחזירה את ההודעה המקורית. o שימו לב, אלגוריתמי ההצפנה שיממשו המחלקות היורשות יכולים להיות מאוד שונים, אבל בכולם מתקיים הדבר הבא: $decrypt(encrypt(m)) = m$ תמיד יתקיים o ניתן להוסיף שדות/מתודות פרטיות למחלקה במידת הצורך o במידה והשתמשם במפתח הצפנה יש לאחסנו ב-DB. - יש ללמוד על האלגוריתם One Time Pad (OTP) וליצור את המחלקה OTPCryptoAlgorithm אשר יורשת מ-CryptoAlgorithm וממשת את המתודות encrypt ו-decrypt ע"פ אלגוריתם OTP
<p>5.0.1</p>	<p>בונוס – הצפנה והוספת פיצ'רי אבטחה. (Crypto++)</p>	<p>בגרסה 5.0.0 התנסיתם/ן בכתיבת אלגוריתם הצפנה משלכם/ן, ותאמינו או לא, OTP הוא אלגוריתם שנחשב <i>perfectly secret</i>, ובלי להיכנס למושגים מהסתברות זה אומר שהסיכוי לנחש את ההודעה המקורית אחרי שראינו את הצופן, זהה לסיכוי לנחש את ההודעה בלי לראות את הצופן.</p> <p>כלומר האלגוריתם עצמו לא חושף משהו על ההודעה המקורית, ולכן נחשב <i>perfectly secret</i>.</p> <p>למרות זאת, ל-OTP יש בעיה רצינית וזה שהמפתח הצפנה צריך להיות בגודל ההודעה, וזה לא משהו שאפשר לעמוד בו ב"עולם האמיתי" (לדוגמא כדי לשלוח קובץ בגודל 1GB נצטרך מפתח בגודל 1GB).</p> <p>בגרסה זו תשתמשו באלגוריתמים שמומשו עבורנו, ספציפית בספריית Crypto++ שאליה תורמים מפתחים מחברות מהתעשייה, בעיקר Microsoft. בתיקיית "בונוס הצפנה" תוכלו למצוא מדריך לשימוש ב-Crypto++ ואיך להתקין שיעבוד ב-Visual Studio.</p> <p>דרישות:</p> <ul style="list-style-type: none"> - התקינו את Crypto++ וודאו שאפשר להשתמש בפונקציות של הספרייה מתוך הקוד של הפרויקט. - הוסיפו שתי מחלקות הצפנה נוספות וממשו את המתודות encrypt ו-decrypt באמצעות הפונקציות שמומשו בספריית Crypto++: <ul style="list-style-type: none"> o RSACryptoAlgorithm – משתמשת באלגוריתם RSA. o AESCryptoAlgorithm – משתמשת באלגוריתם AES. * שימו  אין צורך לממש את האלגוריתמים בעצמכם, רק להבין איך להשתמש בספרייה כדי להפעיל אותם. - יש לממש את אותו מנגנון בצד הלקוח, מומלץ להשתמש בספריות המובנות של .net. כגון System.Security.Cryptography שגם בהן יש מימושים לאלגוריתמים הנ"ל. לדוגמא RSA - הוסיפו ל-Communicator שדה מסוג ICryptoAlgorithm, ובחרו אלגוריתם מהאלגוריתמים שמימשתם/ן. - הצפינו את המידע ביציאתו מהשרת, פענחו את המידע בכניסה לשרת. - עשו את אותה פעולה בצד הלקוח.