מכללת הדסה, החוג למדעי המחשב
מבוא לתכנות מונחה עצמים והנדסת תוכנה
סמסטר א', תשפ"ד

תרגיל 2

תאריך אחרון להגשה:

הנביאים – יום ה', 25/01/2024, בשעה 23:59
שטראוס גברים – יום ב', 22/01/2024, בשעה 23:59

מטרת התרגיל:

בתרגיל זה נתרגל תכנון ממשק (Interface designing), שימוש במחלקות מורכבות (מחלקות המכילות אובייקטים של מחלקות אחרות), העברת מידע בין אובייקטים, שימוש נכון בהפניות (References) וב־const.

גם הפעם נתונים קובצי פרויקט שתתבססו עליהם, עם הגדרות להעתקת קבצים אוטומטית מתיקיית resources (לטובת קובץ/י הלוח, כפי שמוסבר בהמשך) ומספר פונקציות עזר לקריאה מהמקלדת ולכתיבה למסך. התאימו את הפרויקט לצוות (הוספת השמות בהתאם להנחיות ההגשה) והוסיפו קבצים לפי הצורך.

תיאור כללי:

בתרגיל זה נבנה משחק. כדי לממש את המשחק, התוכנית שלנו תשתמש במספר אובייקטים שנגדיר, ויצטרכו להעביר מידע ביניהם.

המשחק שנממש הוא "חתול ועכבר". השחקן שולט בדמות העכבר, שעליה לאסוף את כל הגבינה שעל הלוח כדי לסיים את השלב הנוכחי, ובחלק מהשלבים יהיו על הלוח גם חתול אחד או מספר חתולים ששומרים על הגבינה מפני העכבר. אם החתול תופס את העכבר, יורדים לו "חיים". אחרי שלוש פסילות, המשחק מסתיים בהפסד. אם העכבר אסף את הגבינה בכל שלבי המשחק, המשחק מסתיים בניצחון.

סרטון המדגים את המשחק: https://www.youtube.com/watch?v=pTLBO319rLo

עדיין לא למדנו כלים שיאפשרו לנו ליצור משחק גרפי, ולכן נישאר עדיין בטרמינל (console) ונשתמש בתווים שונים כדי לייצג את הדמויות והאובייקטים השונים, כמפורט בהמשך.

להלן נפרט את הגדרות המשחק עבור התרגיל שלנו.

חלק משמעותי מהאתגר בתרגיל הזה הוא לחשוב על תיכון (Design) מתאים, כלומר אלו מחלקות צריכות להיות בקוד, חלוקת האחריות ביניהן, אלו פונקציות יהיו בכל אחת וכו'. שימו לב לדרישה לתאר את התיכון

ולהסביר אותו בקובץ ה־README (בנוסף לסעיפי ה־README האחרים כמו בתרגיל הקודם. הפירוט נמצא בסוף הקובץ, בחלק המתאר את קובץ ה־README).

פירוט הדרישות:

מטרת המשחק וסיומו

כאמור, המטרה של השחקן (-העכבר) היא לאסוף את כל הגבינה בכל שלב, מבלי להיתפס על ידי החתולים. השלב מסתיים בהצלחה כאשר נגמרות כל פיסות הגבינה שעל המסך, והמשחק ממשיך למסך (השלב) הבא. המשחק ממשיך כל עוד יש שלבים נוספים ונשארו לשחקן "חיים". בתחילת המשחק יש לעכבר שלושה "חיים". המשחק יסתיים (בהצלחה) בסיום השלב האחרון, או (בהפסד) אחרי שלוש פסילות. יש להגיש משחק עם לפחות שלושה שלבים.

האובייקטים במשחק

הדמויות:

- העכבר. כאמור, מטרת המשחק היא לאסוף את כל פיסות הגבינה שבשלב, אם כן כשהעכבר מגיע לגבינה הוא "אוכל" אותה, ומקבל נקודות כמפורט בהמשך. בכל התנגשות עם חתול ירדו לו "חיים". כמובן שהוא לא יכול לעבור דרך קירות. העכבר יסומן בתו % (שקצת מזכיר את האוזניים או העיניים שלו).
- חתול. רודף אחרי העכבר, ומוריד לו חיים בהתנגשות. הוא יסומן בתו ^ (שקצת מזכיר את האוזן שלו).

סוגי העצמים הנייחים שבמשחק

בלוח המשחק יכולים להופיע העצמים הבאים:

- קיר. אף דמות לא יכולה לעבור קירות. יסומן בתו #.
- גבינה. העכבר אוכל אותן. כשחתול עולה על גבינה לא קורה לה כלום והיא נשארת במקומה. במקרה כזה, נשים לב שעלינו להציג על המסך את החתול, ולא את הגבינה (למרות שהיא עדיין שם). תסומן בתו *.
- מתנה. כשהעכבר לוקח מתנה, אחד החתולים נעלם מהשלב. כשחתול דורך על מתנה, לא קורה כלום, וגם כאן המתנה נשארת במקומה. בדומה להנחיה לעיל עבור הגבינה, גם כאן נקפיד במקרה כזה להציג על המסך את החתול ולא את המתנה. תסומן בתו $.
- דלת. אף דמות לא יכולה לעבור דרך דלת. לעומת זאת, אם יש לעכבר מפתח, הוא יכול לפתוח את הדלת (היא נעלמת) והיא הופכת להיות מעבר רגיל (אריח ריק). מסומנת בתו D (עבור Door).

- מפתח. העכבר יכול לאסוף מפתחות כדי שיוכל לעבור בדלתות. כשחתול עובר על מפתח לא קורה כלום (ושוב, נציג במצב הזה את החתול, לא את המפתח). מסומן בתו F (קצת מזכיר צורה של מפתחות ישנים).
- אריח ריק. זה מקום שהיה ריק מתחילת השלב וכרגע אין בו שום דמות או מקום שהתרוקן אחרי שמה שהיה שם נאסף. מוצג כרווח.

מהלך המשחק ותנועת הדמויות

מהלך המשחק

לשם פישוט, המשחק לא מתנהל "בזמן אמת" לפי שעון, כמו המשחק הרגיל, אלא לפי תור. המשתמש יעשה את התור של העכבר, ואז המחשב (כלומר הקוד שתכתבו...) יבצע את התור של כל אחד מהחתולים לפי הסדר. בין תזוזה של חתול אחד לתזוזה של חתול אחר ניתן (אך לא חובה) להוסיף המתנה קצרה, כדי שיהיה קל יותר לעקוב על המסך מי זז ולהיכן. איך? על ידי הוספת השורות הבאות בתחילת הקובץ:

```
#include <thread>
#include <chrono>
using namespace std::chrono_literals;
```

כעת נוכל לגרום להשהיית ריצת הקוד בהוספת השורה הבאה במקום הרצוי (בדוגמה יצרנו השהייה למשך 200 מילישניות (זו המשמעות של ms), אבל כמובן משך הזמן הזה נתון לשיקול דעתכם, והמטרה החשובה היא לוודא שהתוצאה משפרת את המשחקיות):

```
std::this_thread::sleep_for(200ms);
```

הזזת הדמויות

בכל תור דמות יכולה לזוז לאחד מארבעת הכיוונים (אך לא באלכסון). דמות זזה בכל תור רק משבצת אחת.

השחקן מזיז את הדמות שלו בעזרת מקשי החיצים (ראו הסבר בהמשך), או לחיצה על מקש הרווח כדי לוותר על התור.

השיטה להזזת החתולים נתונה לשיקול דעתכם. מספר נקודות מציון התרגיל מוקדשות לנושא זה וככל שהשיטה מוצלחת יותר, הניקוד לסעיף זה יעלה. האפשרות הפשוטה ביותר היא תזוזה אקראית, אולם היא גם "מתומחרת" בהתאם. בכל מקרה, יש לציין בקובץ ה־README, בסעיף 6 (בסעיף האלגוריתמים), מה שיטת התזוזה שמימשתם ולהסביר את הבחירה בה, איך היא עובדת וכדומה.

ביצוע התנועה

נבחין בין המקרים הבאים לגבי התא המבוקש:

- אם התא המבוקש חסום (יש בו קיר או דלת, למעט המקרה של עכבר עם מפתח, ראו להלן), התנועה לא תתבצע. מה כן יקרה? נבדיל בין השחקן לבין דמויות המחשב (החתולים): אם השחקן ניסה לזוז בכיוון המכשול ונחסם, התנועה לא תתבצע אך גם התור נשאר שלו, עד שהוא יזוז למקום חוקי או עד שהוא יוותר במפורש על התור שלו (בעזרת מקש הרווח, כאמור לעיל). לעומת זאת, במקרה שאלגוריתם התזוזה של החתול ניסה צעד כזה ונחסם, הוא מפסיד את התור, והתור עובר לדמות הבאה, לפי הסדר שתואר.
- אם התזוזה היא לכיוון אחד מגבולות השלב (למשל, תנועה ימינה מהתא הכי ימני בשורה, או תנועה מטה מהשורה הכי תחתונה), זה נחשב כמו התנגשות בקיר, כלומר, גם אם ניתן להגיע למשל לקצה השמאלי של המסך בשורה מסוימת, לחיצה שמאלה תיחשב כמו ניסיון כניסה לתא חסום (עם הכללים כאמור לעיל). שימו לב שגם אם בשלבים שתגישו עם המשחק תמיד יהיו קירות מסביב זה לא נחשב לפתרון מספיק, מכיוון שייתכן שנבדוק את ההגשה על קובץ שלב שלא בנוי כך! (כן ניתן להוסיף תמיד קירות מסביב, מעבר למה שיש בקובץ השלב.)
- אם העכבר נע לתא שיש בו חתול, המהלך חוקי (הוא מתבצע), אולם הוא "ייאכל" מיד.
- כצפוי, גם אם חתול נע לתא שבו נמצא העכבר, העכבר "נאכל".
- אם חתול נע לתא שיש בו חתול אחר, המהלך חוקי. כלומר, שני חתולים (או יותר) יכולים להיות בתא אחד.
- אם חתול מגיע לתא שיש בו גבינה או מתנה, המהלך חוקי. כאמור לעיל, נראה על המסך רק את החתול, אבל עדיין הגבינה או המתנה נמצאים שם (ויוצגו שוב ברגע שהוא יזוז משם).
- אם העכבר נע לתא שיש בו גבינה, הוא "אוכל" את הגבינה, והניקוד שלו עולה (ראו פירוט להלן לגבי הניקוד).
- אם העכבר נכנס לתא שיש בו מתנה, הוא "לוקח" את המתנה (היא מוסרת מהלוח), ואחד החתולים נעלם.
- אם העכבר נכנס לתא שיש בו מפתח, הוא לוקח את המפתח ומונה המפתחות שלו עולה.
- אם העכבר מנסה להיכנס לתא שיש בו דלת, ויש לו מפתח, הדלת "נפתחת" (נעלמת מהמסך), יורד מפתח אחד ממונה המפתחות של העכבר, והוא עובר לתא שבו הייתה הדלת עד כה.

נדגיש שוב את ההבדל בין השחקן לדמויות המחשב בכך שהשחקן יכול לבחור לוותר על תורו (בעזרת מקש הווח), אולם המחשב לא יכול "לבחור" לוותר על תורו, אלא רק להפסיד את התור במקרה שניסה לזוז לתא חסום, כנ"ל.

סיום השלב

אחרי שהעכבר אסף את כל פיסות הגבינה בשלב הנוכחי, הוא סיים בהצלחה את השלב, והתכנית צריכה לטעון את השלב הבא (אם יש). שימו לב שהדמויות לא שומרות על מקומן בין השלבים וצריך להציב את האובייקטים ואת הדמויות מחדש לפי השלב המופיע בקובץ (כפי שיוסבר בהמשך). כמו כן גודל השלב יכול להיות שונה בין שלב לשלב.

ניקוד

- כל פיסת גבינה שהעכבר אוסף מזכה אותו ב־10 נקודות
- כל מתנה שהעכבר לוקח מזכה אותו ב־5 נקודות
- כל פתיחת דלת מזכה ב־2 נקודות.
- כל סיום שלב מוצלח מזכה ב־25 נקודות + 5 נקודות בונוס עבור כל חתול שהיה בשלב. כלומר סיום שלב שהיו בו 2 חתולים יזכה את השחקן ב־35 נקודות (ההנחה היא שככל שיש יותר חתולים קשה יותר לסיים את השלב, לכן הניקוד בהתאם).

הצגת נתוני המשחק בעת המשחק

בעת המשחק, יש להציג את נתוני המשחק לכל אורך המשחק. כלומר יש להציג את מצב הניקוד, מצב החיים (=מספר החיים שנותרו), מספר המפתחות שהעכבר מחזיק כרגע ומספר השלב. כמובן, כל שינוי יעדכן את התצוגה. התצוגה תהיה בשורה (או שורות) שאחרי השלב.

דרישות לפרטי מימוש

קובץ main.cpp קטן

כמוסבר בתרגול, הציפייה היא שהקובץ main.cpp יהיה מינימלי ככל האפשר, כך שפונקציית ה־main המופיעה בו תכיל לכל היותר 2 שורות. כלומר, יש ליצור את האובייקט שמנהל את כל התוכנית בשורה אחת, ולקרוא לפונקציה שתריץ את המשחק בשורה השניה. דוגמא לקוד כזה ניתן למצוא במודל בדוגמא של ForwardDeclaration, בקובץ main.cpp שבתוכו.

לוח המשחק וטעינת השלבים

כאמור, הדרישה היא ליצור לפחות שלושה שלבים שונים. השלבים האלה צריכים להיקרא מקובץ טקסט חיצוני (או מספר קבצים כאלה). בכל מקרה, המשחק צריך לתמוך בקלות באפשרות של יותר משלושה שלבים ובגדלים שונים (ככל שמאפשר חלון ה־console) גם בלי צורך בשינוי קוד או קומפילציה מחודשת אלא רק שינוי של הקבצים החיצוניים האלה או הוספה של קבצים נוספים.

פורמט הקובץ המדויק (למשל, האם צריך לציין בפירוש את מספר השורות בשלב לפני שמופיע התוכן שלו) וההחלטה אם כל השלבים יהיו בקובץ אחד ברצף או בקבצים נפרדים נתונים להחלטתכם, אבל באופן כללי הדרישה היא שכל שלב ייראה בקובץ באופן דומה למה שיוצג בסופו של דבר על המסך, כלומר ייצוג הדמויות והאריחים השונים בקובץ חייב להיות זהה לצורה שבה הם מופיעים על המסך במשחק עצמו (ורווחים עבור תאים ריקים), למשל, שלב מינימלי של שורה אחת, שתי עמודות, המכיל עכבר ופיסת גבינה אחת ייראה בקובץ כך: %*. כמו כן, אם כל השלבים נמצאים בקובץ אחד ברצף, יש להפריד בין שלב לשלב בעזרת שורה ריקה.

שימו לב שאם השלבים נתונים בקבצים נפרדים, עדיין צריכה להיות אפשרות להוסיף שלבים חדשים בלי לשנות את הקוד או לקמפל מחדש. שתי דרכים אפשריות לבצע זאת הן:

1. שמות הקבצים יתאימו לתבנית מסוימת (לדוגמה Level001.txt), ובקוד שפותח את הקבצים ניצור את המחרוזת המתאימה לפי התבנית. כך נמשיך לנסות לפתוח גם את Level004.txt (בדוגמה הנ"ל), עד שנגיע לקובץ שלא נמצא וכך נדע שאין עוד שלבים.
2. יהיה קובץ "Playlist", עם שם ידוע מראש, שהקוד פותח אותו, ובו יופיעו שמות קובצי השלבים לפי הסדר הרצוי, וכך הקוד עובר עליו בלולאה וקורא שורה אחרי שורה כדי לדעת מה שם השלב הבא. כשנרצה להוסיף שלב, ניצור את קובץ השלב ואז נוסיף את השם שלו לסוף הרשימה הזו.

בכל מקרה, תעדו את ההחלטות האלה ואת פורמט הקובץ בקובץ ה־Readme כך שניתן יהיה להבין איך ליצור ולהוסיף שלב חדש.

את הקובץ (או הקבצים) יש לשים בתיקיית resources, ובקובץ CMakeLists.txt שבתיקייה הזו להוסיף שורה של פקודת configure_file (כמו שמופיעה שם לדוגמה) עבור כל אחד מהקבצים, ולשנות את הפרמטר הראשון של הפקודה להיות שם הקובץ. לדוגמה, אם יש לנו קובץ בשם LevelList.txt, נוסיף את השורה:

```
configure_file ("LevelList.txt" ${CMAKE_BINARY_DIR} COPYONLY)
```

הקבצים האלה יועתקו אוטומטית לתיקייה שבה גם יווצר קובץ ה־exe של התוכנית שלנו. כך, כדי לפתוח את הקובץ מהקוד, לא צריך לציין שום שם של תיקייה, ומספיק להעביר את שם הפרמטר לבנאי של std::ifstream או לפונקציית ()open שלו, למשל:

```
auto levelList = std::ifstream("LevelList.txt");
```

מיקום הדמויות

בעת כתיבת השלב, יש לוודא את הדברים הבאים:

- העכבר מופיע פעם אחת בדיוק.
- מופיעה לפחות פיסת גבינה אחת.

כלומר, אין צורך לתמוך בקובץ שלב שלא מקיים את התנאים הללו, אפשר להניח שזה המצב ואין צורך להוסיף בקוד התמודדות עם מצב שבו אחד מאלה לא מתקיים.

בתחילת כל שלב, המיקום הראשוני של הדמויות השונות יתאים למיקום שבו הן מופיעות בשלב בקובץ. אם השחקן "נאכל" ויש לו עוד "חיים", כל הדמויות הנעות שהיו על המסך (כלומר העכבר והחתולים, למעט אלה שהועלמו על ידי מתנות) תמוקמנה מחדש כפי המתואר בקובץ, כך נמנע מצב שבו העכבר מיד "ייאכל" שוב על ידי אותו חתול שהרגע תפס אותו. שאר האובייקטים במשחק (למשל, פיסות הגבינה והמתנות) ישארו כמו שהם (מה שנאכל לא יופיע מחדש).

הערות (שיעזרו) למימוש

ניקוי המסך

כדי לאפשר הצגה ברורה של עדכוני הלוח במסך ה־Console, נצטרך לנקות את המסך בין הדפסה להדפסה.

קיימת אפשרות לנקות את המסך על ידי הפקודה system("cls"); (אחרי הוספה של #include <cstdlib>), אבל זו אפשרות לא מוצלחת שגורמת להבהובים וריצודים של חלון המשחק. (אגב נציין שפקודת system מפעילה פקודות של מערכת ההפעלה, ולכן הפקודה המסוימת הזו, עם cls, תעבוד רק בסביבת Windows ולא במערכות הפעלה אחרות.)

לכן הוספנו בקובץ io.h שמופיע כחלק מקובצי הבסיס לתרגיל, את הפונקציות Screen::resetLocation() ו־Screen::setLocation(). הפונקציות האלה מאפשרות להזיז את הסמן לתחילת מסך ה־Console או למיקום הרצוי במסך (בהתאמה). לכן כל מה שנדפיס לאחר הקריאה לפונקציות האלה, ישכתב את מה שכבר מופיע באותו מיקום. שימו לב שכדי "למחוק" תווים קיימים יש להדפיס רווחים. כמו כן, אם יש צורך "לשכתב" שורה, חשוב להדפיס את כל רוחב השורה (כולל רווחים) כדי למחוק את הטקסט שהיה שם קודם.

במעבר בין שלב לשלב ניתן להיעזר בפקודת cls הנ"ל, כדי לפשט את ניקוי המסך.

שימוש במקשי החיצים

עד היום הכרנו איך לקבל מהמשתמש קלט שמכיל תווי ASCII, כאלה שיש להם ייצוג מודפס על המסך. כמו כן, שיטות הקלט הרגילות (בין אם זו cin, scanf או אפילו getc) דורשות מאתנו להמתין עד שהמשתמש ילחץ על מקש ה־Enter לפני שהן מתחילות לקבל את הקלט. המגבלה הראשונה מונעת מאתנו להשתמש במקשי החיצים בכלל. המגבלה השנייה מונעת מאתנו לקבל את הקלט כל עוד לא נלחץ גם מקש ה־Enter, דרישה שאיננה סבירה במשחק.

כדי להשתמש במקשי החיצים, וכדי לקבל את המקש שנלחץ גם בלי המתנה ל־Enter, השתמשו בפונקציה _getch() שנמצאת ב־conio.h. הפונקציה מחזירה int ובאמצעותו תוכלו להחליט מה לעשות עכשיו, לדוגמה לאן להתקדם וכו'. (כמו שהזכרנו קודם: אין להשתמש במספרים מפורשים, literals, ישירות בקוד hard-coded, אלא עליכם לעשות זאת בעזרת הגדרת קבועים). גם כאן, כדי להקל, הוספנו מספר קבועים שימושיים בקובץ io.h שאתם מקבלים מאיתנו.

מותר לשנות את הקבצים הנתונים.

שיטת עבודה מומלצת

ההמלצה היא לעבוד בשלבים קטנים באופן כזה שבכל פעם מה שכתבנו מתקמפל ורץ, כך שאפשר מיד לראות האם מה שכתבנו עד כה אכן עובד כמצופה, במקום לעבוד במשך מספר ימים, לכתוב קוד, ורק אז לגלות שהוא בכלל לא מתקמפל או ששום דבר לא עובד כצפוי.

למשל:

- שלב ראשון: לקרוא את השלב מהקובץ ולהדפיס אותו למסך כמו שהוא בלי שינוי.
- שלב שני: ליצור את האובייקט של אחת הדמויות לפי המיקום שקראנו מהשלב.
- שלב שלישי: להזיז את הדמות לפי לחיצות המקשים (זה בסדר אם הדמות כרגע תזוז בלי שום הגבלות של קירות וכדומה, רק שנראה שהצלחנו להזיז אותה ולהדפיס אותה בצורה נכונה).

וכן הלאה.

כדאי לתכנן רגע לפני כן מה המחלקות הנדרשות, כמו שדיברנו בתרגול איך להסיק מהדרישות מה המחלקות שנצטרך. מצד שני, לפעמים לנסות לתכנן הכול מראש זה רק מסבך יותר, כי יש הרבה דברים לא מוכרים או לא ודאיים מה שמשאיר אותנו עם הרבה סימני שאלה. לכן, לפעמים כדאי להתחיל לכתוב את השלבים הראשונים אחרי התכנון הבסיסי, ואז דברים נהיים יותר ברורים ואפשר להמשיך הלאה.

הערה כללית

הרבה פרטים הוגדרו לעיל במדויק, אולם תמיד, בכל פרויקט מספיק מורכב, יש התנהגויות שלא מוגדרות במדויק על ידי הדרישות. אפשר להתייעץ במקרה הצורך, אבל ההנחייה הכללית היא שעל דברים שלא נאמרו בהגדרות תוכלו להחליט עצמאית מה הדרך הטובה יותר לביצוע הפעולה או לפתרון הבעיה (כלומר, טובה יותר מבחינת משחקיות, או סבירה מבחינת משחקיות וקלה מבחינה תכנותית :)).

קובץ ה־README:

יש לכלול קובץ README שיקרא README.docx, README.doc או README.txt (ולא בשם אחר). הקובץ יכול להיכתב בעברית ובלבד שיכיל את הסעיפים הנדרשים.

קובץ זה יכיל לכל הפחות:

1. כותרת.
2. פרטי הסטודנט: שם מלא כפי שהוא מופיע ברשימות המכללה, ת"ז.
3. הסבר כללי של התרגיל.
4. רשימה של הקבצים שיצרנו, עם הסבר קצר (לרוב לא יותר משורה או שתיים) לגבי תפקיד הקובץ.
5. מבני נתונים עיקריים ותפקידיהם.
6. אלגוריתמים הראויים לציון.
7. **תיכון (design): הסבר קצר מהם האובייקטים השונים בתוכנית, מה התפקיד של כל אחד מהם וחלוקת האחריות ביניהם ואיך מתבצעת האינטראקציה בין האובייקטים השונים.**
8. באגים ידועים.
9. הערות אחרות.

יש לתמצת ככל שניתן אך לא לוותר על אף חלק. אם אין מה להגיד בנושא מסוים יש להשאיר את הכותרת ומתחתיו פסקה ריקה. **נכתוב ב־README כל דבר שרצוי שהבודק ידע כשהוא בודק את התרגיל.**

אופן ההגשה:

הקובץ להגשה: ניתן ליצור בקלות קובץ zip המותאם להגדרות ההגשה המפורטות להלן ישירות מתוך VS, כפי המוסבר תחת הכותרת "יצירת קובץ ZIP להגשה או לגיבוי" בקובץ "הנחיות לשימוש ב־Visual Studio 2022". אנא השתמשו בדרך זו (אחרי שהגדרתם כראוי את שמות הצוות ב־MY_AUTHORS: **נשים את שמות המגישים בתוך המרכאות, נקפיד להפריד בין השם הפרטי ושם המשפחה בעזרת קו תחתי ואם יש יותר ממגיש אחד, נפריד בין השמות השונים בעזרת מקף (מינוס '-')**) וכך תקבלו אוטומטית קובץ zip המותאם להוראות, בלי טעויות שיגררו אחר כך בעיות בבדיקה.

באופן כללי, הדרישה היא ליצור קובץ zip בשם exN-firstname_lastname.zip (או במקרה של הגשה בזוג – exN-firstname1_lastname1-firstname2_lastname2.zip), כשהקובץ כולל את כל קובצי הפרויקט, למעט תיקיות out ו־vs. כל הקבצים יהיו בתוך תיקייה ראשית אחת.

את הקובץ יש להעלות ל־Moodle של הקורס למשימה המתאימה. בכל מקרה, **רק אחד** מהמגישים יגיש את הקובץ ולא שניהם.

הגשה חוזרת: אם מסיבה כלשהי החלטתם להגיש הגשה חוזרת יש לוודא ששם הקובץ זהה לחלוטין לשם הקובץ המקורי. אחרת, אין הבודק אחראי לבדוק את הקובץ האחרון שיוגש.

כל שינוי ממה שמוגדר פה לגבי צורת ההגשה ומבנה ה־README עלול לגרור הורדת נקודות בציון.

מספר הערות:

1. נשים לב לשם הקובץ שאכן יכלול את שמות המגישים.
2. נשים לב לשלוח את תיקיית הפרוייקט כולה, לא רק את קובצי הקוד שהוספנו. תרגיל שלא יכלול את כל הקבצים הנדרשים, לא יתקבל וידרוש הגשה חוזרת (עם כללי האיחור הרגילים).

המלצה כללית: אחרי הכנת הקובץ להגשה, נעתיק אותו לתיקייה חדשה, נחלץ את הקבצים שבתוכו ונבדוק אם ניתן לפתוח את התיקייה הזו ולקמפל את הקוד. הרבה טעויות של שכחת קבצים יכולות להימנע על ידי בדיקה כזו.

<u>בהצלחה!</u>