1. תרגול פיתוח משחק Xonix.
2. אהרון משה שמסיאן: 205779598, עקיבא משה סודרנסקי: 326382322

**3. הסבר כללי של התרגיל:**

בתרגיל זה נבנה משחק גרפי xonix , מטרת השחקן לנסות לסגור שטחים פתוחים באמצעות תנועה מחוץ לאזור הסגור. כאשר השחקן יוצא מהאזור הסגור ויוצר מסלול סגור, השטח הכלוא מתווסף לאזור שבשליטתו. המטרה היא לסגור אחוז כמה שיותר גדול מהלוח מבלי להיפגע מאויבים.

במהלך הפיתוח יש ליישם מנגנוני תנועה לשחקן ולאויבים, טיפול בהתנגשויות, איתור שטח נסגר (באמצעות אלגוריתם כמו flood-fill), וזיהוי מצבים של הפסד (כגון פגיעה באויב או בדריכה עצמית על השובל.

המשחק יופעל על פי קובץ שלבים בטקסט פשוט, שבו מוגדרים פרטי השלבים, כולל גודל הלוח, מספר האויבים, אחוז הסגירה הנדרש לניצחון, מיקום התחלתי של השחקן, ועוד. האויבים נעים באלכסונים ומשנים כיוון לפי התנגשויות. יש לנהל נכון את מצבי המשחק, כולל סיום שלב, הפסד, ניקוד, חיי שחקן ופסילות.

מטרות התרגיל הן תרגול עבודה עם מבני נתונים ואלגוריתמים מה־STL, תכנון מחלקות וממשק גרפי, וכן כתיבה תקינה ומודולרית של קוד מונחה עצמים. בנוסף, יש דרישות מתקדמות לבעלי עניין, כמו הוספת אויבים "חכמים", טיימר, ניקוד, והתנהגויות מורכבות נוספות.

**4. רשימות קבצי התרגיל:**

* **GameManager**  
  אחראי על ניהול המשחק כולו – טעינת שלבים, הרצת הלולאה הראשית, ניהול זרימת המשחק, וסיום שלבים.
* **GameInfo**  
  מחזיק את מצב המשחק הנוכחי – מספר חיים, מספר שלב, אחוז סגירה, וניהול ניצחון או הפסד.
* **ErrorWindow**  
  מציג חלון שגיאה גרפי במקרים של שגיאות בקלט, חריגות, או בעיות קריטיות אחרות בהרצת המשחק.
* **GameException**  
  מחלקת חריגה ייעודית שמכילה הודעת שגיאה וניתנת ללכידה לצורך טיפול והצגה מסודרת.
* **ObjectType**  
  מונה (enum class) של סוגי האובייקטים בלוח – שחקן, אויב, קיר, שביל, רקע ועוד.
* **Images**  
  מחלקה סטטית שאחראית לטעינה מוקדמת של טקסטורות. מספקת ספרייטים מתאימים לפי סוג וגודל.
* **Object**  
  מחלקת בסיס לכל אובייקט גרפי בלוח. כוללת מידע על מיקום, סוג האובייקט, פעולה של ציור, והתנגשות.
* **MovingObject**  
  מחלקת בסיס לאובייקטים נעים. כוללת כיוון תנועה, מהירות, ועדכון מיקום לפי זמן.
* **Player**  
  מייצג את השחקן. מקבל קלט מהמשתמש, יוצר שבילים, סגור שטחים, ומנהל את אינטראקציות המשחק.
* **Enemy**  
  מייצג אויב שמסתובב בלוח. נע באלכסון, משנה כיוון כשהוא פוגע בקיר, ויכול לפגוע בשחקן או בשובל.
* **TileType**  
  מונה המציין את סוגי האריחים בלוח: ריק, מלא, מסגרת, או שביל.
* **Tile**  
  מייצג תא בודד בלוח. כולל מיקום, סוג אריח, וספרייט גרפי. משמש ליצירת הלוח ולבדיקות סגירה.
* **TileBoard**  
  מייצג את לוח המשחק – מטריצה של אריחים. כולל ניהול ציור הלוח, עדכונים, בדיקות סגירה, אתחול שלבים, וחישוב אחוזי שטח.
* **playlistLevels.txt**  
  קובץ טקסט שמכיל את רשימת השלבים לטעינה. כל שורה היא שם של קובץ שלב (כגון level001.txt).
* **levelXXX.txt**  
  קובץ שמכיל תיאור של שלב יחיד: גודל המסך, מספר אויבים, מיקומים התחלתיים, מפת האריחים, ואחוז סגירה נדרש.

**5. פירוט מבנה נתונים עיקריים:**

**וקטור דו מימדי של אריחים:**

לוח המשחק מיוצג כמערך דו־ממדי (וקטור של וקטורים) של מצביעים חכמים לאריחים מסוג Tile.  
כל תא במבנה מייצג אריח בודד בלוח, וכולל מידע על סוג האריח (ריק, מסגרת, שביל, סגור).

מבנה זה מאפשר גישה מהירה לפי שורה ועמודה, ומשמש לציור הלוח, איתור שטחים סגורים, ולוגיקת המשחק.

**וקטור אובייקטים נעים:**

כל האובייקטים שיכולים לזוז בלוח – כלומר השחקן והאויבים – נשמרים בווקטור הנעים, תוך שימוש בפולימורפיזם לבדיקת התנגשויות בין האובייקטים.

כל אובייקט כולל מיקום, מהירות, כיוון תנועה, ופעולות כמו תזוזה, ציור, והתנגשות.  
המבנה מאפשר לעבור על כל האובייקטים בצורה אחידה, לעדכן את מצבם בכל שלב של המשחק, ולטפל בהתנגשויות בצורה כללית.

**6. אלגוריתמים חשובים שהשתמשנו בהם.**

**תזוזת אויביים:**

כל אויב בלוח זז באלכסון – למשל למעלה וימינה או למטה ושמאלה – במהירות קבועה.  
כשהאויב נתקל בקיר או בשטח סגור, הוא מחליף כיוון לפי הכיוון שבו פגע:

אם פגע בקיר מלמעלה או מלמטה – הוא הופך את כיוון התנועה בציר האנכי.אם פגע בקיר מימין או משמאל – הוא הופך את כיוון התנועה בציר האופקי. אם פגע גם מלמעלה וגם מהצד – הוא מחליף את שני הכיוונים.

**אלגוריתם סגירת שטח – רקורסיה בארבעה כיוונים Backtracking**

כדי לבדוק אילו אריחים צריכים להפוך לשטח סגור, השתמשנו באלגוריתם רקורסיבי שפועל בארבעה כיוונים. המטרה היא לזהות אזורים סגורים שהשחקן הצליח לכלוא באמצעות השובל שלו, ולהפוך אותם לשטח סגור.

האלגוריתם מתחיל מאריח ריק ובודק את כל ארבעת הכיוונים ממנו – למעלה, למטה, ימין ושמאל. אם דרך המסלול הזה ניתן להגיע לשטח פתוח או לצאת מהמסגרת, זה אומר שהאזור לא באמת סגור ולא נצבע.

אבל אם כל הכיוונים חסומים והגישה החוצה לא קיימת, אפשר לדעת שהאזור סגור. במהלך הריצה, האלגוריתם מסמן את האריחים שכבר ביקר בהם כדי לא לבדוק אותם שוב.  
כך נמנע מצב של לולאה אינסופית וחוסכים בדיקות מיותרות. שיטה זו מזכירה את האלגוריתם.

**7. תיכון המשחק.**

תכננו את המשחק בגישה מונחית עצמים, עם הפרדה ברורה בין תחומי אחריות. כל מחלקה אחראית על תחום שונה במשחק, מה שמאפשר קוד מסודר, קריא ונוח להרחבה.

* GameManager הוא המנהל הראשי של המשחק. הוא אחראי על טעינת שלבים, הרצת הלולאה הראשית, ציור ועדכון של המסך, ניהול מצב המשחק, וזיהוי סיום שלבים.
* GameInfo מחזיק את מצב המשחק הנוכחי – מספר החיים, אחוז השטח שנסגר, מספר השלב ועוד.
* TileBoard מייצג את הלוח, שמורכב ממטריצה דו־ממדית של אריחים (Tile). כל אריח מייצג מצב אחר בלוח: ריק, מסגרת, שביל או שטח סגור – על ידי מזהה euna ולא להdraw של לוח המשחק, מדפיס בהתאם למזהה הtype של האריח הנוחכי.
* האובייקטים שזזים (שחקן ואויבים) נשמרים בווקטור אחיד של MovingObject, מחלקת בסיס לאובייקטים נעים. Player ו־Enemy יורשים ממנו, וכל אחד מממש את ההתנהגות שלו.

**פולימורפיזם בהתנגשויות double dispatch**

כדי לממש התנגשות בין אובייקטים מסוגים שונים בצורה גמישה ונכונה, השתמשנו בטכניקה שנקראת **double dispatch**.

המשמעות היא שכל אובייקט קורא לפונקציית handleCollision של האובייקט השני, והיא בתורה קוראת חזרה לפונקציה הספציפית עבור הסוג שקיים בפועל

**8. אין באגים ידועים.**

**9. הערות כלליות:**

**במהלך מימוש התרגיל נוספו מספר שדרוגים ובונוסים שמעשירים את חוויית המשחק ומאפשרים גמישות:**

1. מימשנו חלון גראפי לשגיאות כמו קובץ לא נמצא, וכדומה, ולא פלט לטרמינל.
2. **המשחק תומך בשינויים קלים דרך קובץ קבועים חיצוני – אפשר לשנות מהירויות, זמני שלב, אחוזי סגירה נדרשים ועוד, בלי לגעת בקוד עצמו, ניתן לשנות גודל חלון וכדומה, בקובץ הקבועים של המשחק, וגודל האריחים ולוח הנתונים למטה יתעדכן אטומאטי.**
3. הוספנו טיימר בתחתית המסך שמציג את הזמן הנותר לכל שלב. אם הזמן נגמר, השחקן נפסל.
4. המשחק כולל חמישה שלבים שונים, שכל אחד מוגדר בקובץ נפרד ונטען מתוך קובץ playlistLevells כך שהמימוש ממש גנרי, וניתן להוסיף ולהסיר שלבים של משחק בקלות.
5. אם השחקן סוגר יותר מ-10% מעל הדרישה המינימלית לשלב, הוא מקבל תוספת ניקוד (פקטור) שמתגמל על הביצוע.
6. המיקום ההתחלתי של השחקן קבוע בכל שלב – תמיד בחצי העליון של הלוח.
7. האויבים מתחילים כל שלב במיקום אקראי, מה שיוצר אתגר משתנה בכל הרצה.
8. רמת הקושי עולה משלב לשלב – יותר אויבים, פחות זמן, שטח הנדרש לכיבוש עולה, ובכך המשחק מאתגר יותר ככל שמתקדמים.