**פרוייקט גמר - bombermen**

**מגישים:**

* שלמה זלמן קאהן - 327517066
* שמעון בריזל - 214552143

**תיאור הפרויקט:**

במהלך הפרויקט יצרנו את משחק הבומברמן תוך שימוש בספריית SFML על מנת לשלב אובייקטים מסוג – תמונה, סאונד וחלונות.

**מבנה הפרויקט:**

**מחלקות עיקריות בפרויקט:**

**Bomb:**

**תיאור כללי:**

מחלקה זו יורשת מObjectMove מכיוון והיא זהה בפעולתה לשאר האובייקטים הזזים.

מחלקה זו מנהלת את הפצצות במשחק, כולל הטיימרים לפיצוצים וההשפעה על אובייקטים סמוכים.

**פונקציות עיקריות:**

* : **update**מעדכנת ומדפיסה את הפצצה במהלך המשחק. אם הפצצה מגיעה לזמן הפיצוץ שלה(4 שניות), מתבצע מעבר לפעולת פיצוץ(Explotion).
* **explosion** : מטפלת בלוגיקת הפיצוץ וההשפעות שלו על הסביבה, כולל השמדת אובייקטים בטווח והתנעת אפקטים גרפיים וקוליים.

**Controller**

**תיאור כללי:** מחלקה זו אחראית על ניהול הלוגיקה הכללית של המשחק, כולל קריאת שלבים, עדכון תצוגה, ניהול אירועים וניהול קריאות לבדיקות של התנגשויות.

**פונקציות עיקריות:**

* **:run** מנהלת את לולאת המשחק הראשית, כולל עדכונים גרפיים וטיפול באירועים.
* **readLevels** : טוענת שלבים מתוך קובץ בתחילת המשחק ובתחילת שלב חדש ומאתחלת את מצב המשחק בהתאם.
* **:eventManager**מטפלת באירועי משתמש כמו לחיצות מקשים ותנועות עכבר.
* **:exceptionManager**בודקת מצבי קיצון כמו ניצחון או לחילופין מוות השחקן (Robot) ומטפלת בהתאם**.**
* **readLevelsGift:** מוסיפה מתנות בהתאם לסוג השלב, מקבלת את המיקום ממחלקת LoadFile. וממקמת את המתנה בהתאם לכך  
  במידה ומדובר בשלב ללא מגבלת זמן, לא תווצר מתנה להוספת חיים.
* **:collisionObjects** מזהה התנגשויות בין אובייקטים שונים במשחק ומבצעת קריאה לפונקציות הפנימיות שלהם על מנת לבצע טיפול.
* **resetObjects** : מאתחלת מחדש את מצב האובייקטים במשחק לאחר מעבר שלב או אירועים אחרים.
* **clearObjectsGame** : מנקה את האובייקטים במשחק כדי לאתחל מצב חדש.
* **updateWindow**: מעדכנת ומדפיסה את כל אובייקטי המשחק, ובנוסף מבצעת בדיקה של isDead ומוחקת בהתאם לזה.
* **printDataGame:** מדפיסה את המידע למשתמש, כגון זמן שעבר, כמות החיים ועוד.

**Door**

**תיאור כללי:**

מחלקה זו יורשת את Object מכיוון שמדובר באובייקט שאינו זז,

מחלקה זו מייצגת את דלת המשחק, כולל התנהגות בעת התנגשות עם הדלת.

**פונקציות עיקריות:**

* **collided** : בודקת האם יש התנגשות בין הדלת לבין אובייקט אחר, ומבצעת פעולות בהתאם לפתיחת השלב הבא.

**Gift**

**תיאור כללי:**

מחלקה זו יורשת את Object מכיוון שמדובר באובייקט שאינו זז,

מחלקה זו מייצגת את המתנות שבמשחק, ומשמשת כמחלקה מורישה עבור סוגי המתנות הספציפיות.

**פונקציות עיקריות:**

* בבנאי הוא מקבל ארגומנט של bool על מנת להגדיר אותו כמוסתר או גלוי, ואז במידה והוא על מקום של אבן (ston) המתנה לא תודפס.
* **Collided:** כאשר יש התנגשות של פצצה עם המיקום הזה, הפצצה תחזור להיות מודפסת.

**GiftAddLife / GiftAddTime / GiftKillOneGuard / GiftStopGuards**

**תיאור כללי:**

ארבעתם מחלקות היורשות מ Gift וכל אחד מהם מבצע פעולה בהתאמה, כגון הוספת חיים, הוספת זמן וכדומה.

**פונקציות עיקריות:**

**Collided:** כל אחד מהם מממש אותו לפי הצורך

**DataTexture**:

**תיאור כללי:**

מחלקה זו היא מסוג Singleton על מנת למנוע יצירת מופע נוסף ממחלקה זו. מחלקה זו אחראית לטעינה, ניהול והצגת טקסטורות במשחק. היא טוענת ומחזירה טקסטורות עבור אובייקטים שונים כמו שומרים, רובוטים, קירות, אבנים, דלתות, פצצות, פיצוצים ומתנות. בנוסף, המחלקה מכילה פונקציות להצגת טקסטורות מסוימות על המסך, כולל הצגת טקסטורות רקע ואירועים מיוחדים.

**פונקציות עיקריות:**

**printEventTexture**: מציגה טקסטורה(תמונה) באירועים מיוחדים (למשל הודעות) על המסך, תוך הצגת תמונה במרכז החלון.

**:printBackgroundTexture**מציגה את טקסטורת הרקע של המשחק, תוך התאמתו לגודל החלון.

**getTexture**: מחזירה את הטקסטורה(Texture) המתאימה לפי הchar שהתקבל.

**:poshTexture**מטעינה טקסטורה(Texture) ומוסיפה אותה לווקטור הטקסטורות, עם סוג מוגדר.הפונקציה משמשת בעיקר את הטעינה הראשונית של הTexture

**loadTexture**: מטעינה את כל הטקסטורות הנדרשות למשחק, כמו טקסטורות של שומרים, רובוטים, קירות, פצצות ועוד.

**Explosion**:

**תיאור כללי:**

המחלקה מייצגת את הפיצוץ של Bomb. הפיצוץ מתרחש במיקום ובכיוון מסוים, ומתפשט באופן שגורם להשפעה על אובייקטים אחרים בסביבה, כמו Guard, Robot, Wall, Stone

**פונקציות עיקריות:**

* **:Move**מעדכנת את מיקום הפיצוץ על פי הזמן שעבר ובתוך גבולות המסך.
* **Collision**: בודקת אם הפיצוץ מתנגש עם אובייקט אחר ומבצעת את ההתנהגות המתאימה בהתנגשות.
* **inWindow**: בודקת אם הפיצוץ נמצא בתוך גבולות המסך. אם הפיצוץ יצא מגבולות המסך, הוא מושמד.

**Guard**:

**תיאור כללי:**

מחלקה זו יורשת מObjet move כמו שאר האובייקטים הזזים.

מחלקה זו מייצגת את השומרים במשחק. השומרים מסוגלים לנוע במרחב המשחק, לבצע שינוי כיוון תנועה בהתנגשות עם קירות או אבנים. המחלקה כוללת לוגיקה לניהול מספר השומרים הפעילים ותגובה להתנגשויות שונות.

**פונקציות עיקריות:**

* **:Guard** מאתחל שומר במיקום אקראי עם כיוון תנועה אקראי. וגם מעדכן את מספר השומרים הפעילים במשחק.
* **:reset**מאפסת את מיקום השומר למיקום ההתחלתי שלו.
* **:move**מעדכנת את מיקום השומר על גבי המסך ומבצעת שינוי כיוון תנועה במרווחי זמן אקראיים או בהתנגשות.
* **:inWindow**בודקת אם השומר בתוך גבולות המסך. אם לא, מבוצע שינוי כיוון.
* **:changeDirection**משנה את כיוון התנועה של השומר באופן אקראי.
* **:firstCheck**מבצע בדיקה ראשונית לטיפול במצבים מיוחדים, כמו הפסקה זמנית של תזוזת השומר או סיום חייו.
* **:collided**מטפלת בהתנגשות עם רובוט.

**LoadFile:**

**תיאור כללי:**

מחלקה זו היא מסוג Singleton בשביל למנוע מצב של יצירת מופע נוסף ולא דרוש למחלקה זו.

מחלקה זו אחראית על טעינת קובץ השלבים של המשחק וניהול המידע הדרוש להפעלת המשחק.  
המחלקה מטפלת בקריאת נתוני השלב, מיקומי המתנות, ואספקת מידע למערכת לגבי השלב הנוכחי.

**פונקציות עיקריות:**

* **fillData**: קורא את נתוני השלב מהקובץ ומאחסן אותם במבני נתונים מתאימים.
* ובנוסף מחפש שורות המכילות מספרים ומפענח את מידע השלב.
* **:colSize** מחשב ומחזיר את הרוחב המקסימלי של השלב.
* **:getLegalGiftLoc**מוצא מיקום חוקי להצבת מתנה במפת המשחק ומחזיר אותו כווקטור מיקום, ובנוסף הוא קובע אם המיקום מוסתר מאחורי אובייקט מסוג אבן.
* **checkGiftsLoc**: בודק אם המיקום הנתון חוקי להצבת מתנה (וללא התנגשות עם אובייקטים אחרים).
* **getFromFile:** קורא את התו הבא מקובץ השלבים שאינו רווח ומחזיר אותו. ובנוסף הוא מנקה את ווקטור המתנות בסיום קריאה כהכנה לקריאת השלב הבא.
* **getPosition**: מחזיר את המיקום הנוכחי במפה עבור האובייקט הנקרא.
* **getSize**: מחזיר את גודל מפת המשחק במונחים של עמודות ושורות.

**Object:**

**תיאור כללי:**

מחלקה זו היא מחלקת בסיס המייצגת אובייקטים כלליים במשחק. המחלקה מספקת ממשקים ותכונות בסיסיות הנחוצים לכל האובייקטים המוצגים על המסך, כמו מיקום, טקסטורה, והגדרת מצב חיים (חי או מת).  
היא משמשת כמחלקת אב למרבית האובייקטים במשחק, כמו דמויות, קירות, מתנות ועוד.

**פונקציות עיקריות:**

* **:Object**יוצר אובייקט עם טקסטורה מוגדרת וממקם אותו בנקודה מסוימת על המסך.
* **Update:** מצייר את האובייקט על המסך במידה והוא לא מת.
* **isDead :** מחזיר האם האובייקט מת או חי.
* **getGlobalLoc:** מחזיר את המלבן החיצוני של גבולות האובייקט לצורך בדיקות התנגשות.

**ObjectMove:**

**תיאור כללי:**

מחלקה זו יורשת מ-**Object** ומייצגת אובייקטים שנעים בתוך חלון המשחק. המחלקה מטפלת בלוגיקה הקשורה לתנועה, כולל שמירה על גבולות החלון, כיווני תנועה ותיקונים למיקום האובייקט במקרה של חפיפות לא חוקיות עם קירות או אלמנטים אחרים.

**פונקציות עיקריות:**

* **:ObjectMov**קונסטרקטור ברירת מחדל המאתחל את מהירות התנועה ל-0.
* **:initPositionLevel**מאתחל את המיקום הראשוני והאחרון של האובייקט וממקם אותו על המסך.
* **inWindow**: בודק אם האובייקט נמצא בתוך גבולות חלון המשחק ומחזיר אותו למיקום הקודם אם הוא חורג.
* **move**מנהל את תנועת האובייקט בהתאם לזמן שעבר ולכיוון התנועה, כולל תיקוני מיקום במידת הצורך.
* **:setLastPosition** מחזיר את האובייקט למיקומו הקודם.
* **straightenUpByTop** : מבצע תיקון במרחקים אנכיים כאשר האובייקט אינו מיושר למיקום חוקי ברשת המשחק.
* **:straightenUpByLeft** מבצע תיקון במרחקים אופקיים כאשר האובייקט אינו מיושר למיקום חוקי ברשת המשחק.
* **spriteMove**: מעדכן את החלק הנכון בטקסטורה של האובייקט לפי כיוון התנועה, המספרים והחלוקות המוזרות בפונקציה זו היא מכיוון שה sprite -ים לא היו בגודל מתאים, ולא היה לנו זמן להחליפם, אנו מתנצלים על כך.

**:Robot**

**תיאור כללי:**

מחלקה זו יורשת מ-**ObjectMove** ומייצגת את השחקן הראשי במשחק. היא מטפלת בניהול התנועה של הרובוט, אינטראקציות עם אובייקטים שונים (כמו קירות, דלתות, מתנות וכו'), חישוב תוצאה, חיים, וזמן. כמו כן, המחלקה כוללת לוגיקה לזכייה והפסד במשחק.

בנוסף, מחלקה זו היא מסוג Singleton מכיוון שמשחק זה לא יוכל להיות בעל 2 רובוטים.

**פונקציות עיקריות:**

* **startLevel**: מאתחל את המיקום ההתחלתי של הרובוט ומגדיר האם השלב מוגבל בזמן.
* **:reset** מחזיר את הרובוט למיקומו הראשוני.
* **:move** מטפל בתנועת הרובוט בהתאם ללחיצות מקשים (חיצים למעלה, למטה, שמאלה וימינה).
* **getPosition** : מחזיר את המיקום המרכזי של הרובוט על המסך.
* **:collided**בודק ומבצע פעולות במקרים שבהם הרובוט מתנגש עם אובייקטים אחרים.
* **collided(Door& door)**  
  טיפול בהתנגשות עם דלת — מגדיר את מצב ניצחון ומעדכן את הניקוד.
* **collided(Wall& wall) / collided(Stone& stone)**  
  מחזיר את הרובוט למיקומו האחרון במקרה של התנגשות עם קיר או אבן.
* **void collided(Explosion& explosion) / void collided(Guard& guard)**  
  טיפול בהתנגשות עם פיצוצים או שומרים — מפחית חיים של הרובוט במקרה הצורך.
* **void collided(GiftAddLife& giftAddLife), void collided(GiftAddTime& giftAddTime)**  
  מטפל במתנות שמוסיפות חיים או זמן לרובוט.
* **:isLimitedTime**מגדיר האם השלב מוגבל בזמן ומעדכן את שעון הזמן בהתאם.
* **printRobotData** : מציג על המסך מידע על חיי הרובוט, ניקוד, וזמן שנותר.
* **isWin**: מחזיר אם השחקן ניצח בשלב הנוכחי.
* **:lostLife**בודק אם הרובוט איבד חיים ומפעיל צליל בהתאם.
* **timeLeft**: בודק האם הזמן לשלב הסתיים ומפחית חיים במקרה הצורך.

**SmartGuard:**

**תיאור כללי:**

מחלקה זו יורשת מ-**Guard** ומייצגת שומר חכם במשחק. תפקידו של השומר החכם הוא לנוע על המסך בצורה מחושבת ולהתקרב למיקומו של השחקן (הרובוט). המחלקה כוללת לוגיקה לניהול כיוון התנועה בהתאם למרחק ולמיקום הרובוט.

**פונקציות עיקריות:**

* **changeDirection**: משנה את כיוון התנועה של השומר בהתאם למרחק האופקי והאנכי מהרובוט.  
  האלגוריתם מחליט האם לנוע על ציר ה-X או ציר ה-Y בהתאם למרחק הקטן יותר בין השומר לרובוט.
* **:Check** בודקת ומעדכנת את כיוון התנועה של השומר בהתאם למרחק מהרובוט.

**SoundManager:**

**תיאור כללי:**

מחלקה זו מנהלת את השמעת האפקטים הקוליים והמוזיקה במשחק. זוהי מחלקת Singleton על מנת לוודא שיווצר מופע יחיד למחלקה זו.

**פונקציות עיקריות:**

* **SoundManager**: קונסטרקטור הטוען את כל קבצי הקול הדרושים למשחק, כולל מוזיקת רקע ואפקטים שונים כמו פיצוץ, קבלת מתנה והפסד חיים.
* **loadSound** : טוענת קובץ קול ל- sf::SoundBuffer ומשייכת אותו לsf::Sound -
* **:playBackgroundMusic**מפעילה את מוזיקת הרקע בלולאה עם עוצמת שמע מוגדרת מראש.
* **:stopAllSounds**עוצרת את כל הצלילים והמוזיקה ברקע.

**StartMenu:**

**תיאור כללי:**

מחלקה זו מנהלת את התפריט ההתחלתי של המשחק, שבו מוצגים כפתורים למעבר למשחק עצמו, הצגת עזרה או יציאה מהתוכנית. המחלקה אחראית על יצירת חלון התפריט, טיפול בלחיצות משתמש והצגת טקסטים שונים במסך.

**פונקציות עיקריות:**

* **:runMenu**מפעילה את תפריט המשחק הראשי. מכילה לולאת אירועים המאפשרת לחיצות כפתור, סגירת החלון וכו'.
* **:createWindow**יוצרת את חלון התפריט ומאתחלת את צורות הכפתורים.
* **:clickManagment**מטפלת בלחיצות עכבר על הכפתורים בתפריט הראשי.
* **fillText**: יוצרת טקסטים מותאמים לכל כפתור בתפריט על בסיס המיקומים שלהם.
* **:drawEndDisplay**מציגה את התפריט על המסך לאחר רינדור כל הכפתורים והטקסטים.
* **:loadHelpText**טוענת את טקסט העזרה מתוך קובץ help.txt
* **:showHelp**מציגה את טקסט העזרה ומאפשרת גלילה שלו על ידי המקשים Up ו-Down.
* **:createBackBotton**יוצרת כפתור "back" להצגת מסך התפריט לאחר הצגת טקסט העזרה.

**TextMaker:**

**תיאור כללי:**

מחלקה זו משמשת כמחולל טקסטים למשחק. המחלקה מנהלת יצירה ועיצוב טקסטים באופן גמיש, כולל התאמה של מיקום, צבע, גודל וסוג הנתונים המוצגים.

**תיאור לפונקציות**

* **:TextMaker**קונסטרקטור שטוען גופן מתוך הקובץ, מגדיר את הגודל והצבע הבסיסיים של הטקסט.
* **sf::Text makeText(const std::string& text, const sf::Vector2f position)**  
  יוצר טקסט עם מחרוזת נתונה ומגדיר את מיקומו על המסך.
* **sf::Text makeText(const std::string& text, const sf::Vector2f position, const sf::Color color)**  
  כמו הפונקציה הקודמת אך מאפשר גם שינוי צבע הטקסט.
* **sf::Text makeText(const std::string& text, const sf::Vector2f position, const int charSize)**  
  מייצר טקסט עם מחרוזת נתונה בגודל פונט מותאם.
* **sf::Text makeText(const sf::Clock& clock, const sf::Vector2f position)**  
  מחולל טקסט שמציג זמן שעבר מאז תחילת השעון בפורמט דקות:שניות.
* **sf::Text makeText(const sf::Time& time, const sf::Vector2f position)**  
  מחזיר טקסט שמציג זמן נתון בפורמט פורמט דקות:שניות.
* **sf::Text makeText(const int text, const sf::Vector2f position)**  
  מציג טקסט מספרי באמצעות המרה של מספר שלם למחרוזת.
* **std::string formatTime(const sf::Time& time)**  
  מעצב את הזמן שניתן לפונקציה לפורמט "MM:SS"

**איך בנוי קובץ הlevels.txt של השלבים:**

* ראשית יש את השלב עצמו, שהוא מורכב מסימנים מוסכמים כגון # = קיר וכו.
* לאחר מכן יש את שורת המידע על השלב, מכילה 3 int-ים, הראשון הוא מספר השלב, לאחר מכן – כמות מתנות, והשלישי – כמות הזמן המוקשה לסיום השלב (או 0 לשלב ללא מגבלת זמן), כאשר בין כל אחד יש "-".

**אלגוריתמים מורכבים:**

1. **בקרת שלבים:** המערכת מאפשרת טעינת שלבים דינמית מקבצים, תוך תמיכה בשמירת מצב המשחק הנוכחי. האלגוריתם מבצע בדיקות תקינות לקובץ לפני הטעינה ומספק טיפול בשגיאות, בסיום הקובץ, האלגוריתם מחזיר את מצביע הקריאה לתחילת הקובץ – בשביל שהמשתמש יוכל להמשיך במשחק גם אם סיים את כל השלבים.
2. **טעינת מתנות מותאמות לשלב + מיקום אפשרי:** על מנת שלא יווצרו מצבים בהם המתנות מודפסות מעל קיר או כל מקום לא הגיוני, ייצרנו פונקציה במחלקת Load File שבוחרת מיקום בrand, מוודאת את תקינות המיקום, ומחזירה אותו לקונטרולר כמיקום נכון להנחת המתנה.

**באגים ידועים:**

* במצב שבו החלון לא גדול דיו, יתכנו שיבושים בהדפסת הdata game, וזה מכיוון ובדרישות הפרויקט נדרשנו להדפיס במידע למשתמש מספר דברים (כגון מספר השלב, זמן שנותר ועוד).
* במידה ומוגדר בשלב יותר מתנות ממקומות פנויים – התוכנית לא תרוץ, מכיוון והיא תחפש מקום פנוי להנחת מתנה ולא תמצא.

**הערות:**