בס"ד

Bomberman - Game Project

2.1

שמות המגישים:

1. חנן גרשון כהן מ.ז. 213669229

2. צבי רולניק מ.ז. 24917965

3.1

פרויקט זה הוא יישום של משחק Bomberman הקלאסי באמצעות C++ ו-SFML. המשחק עוקב אחר מכניקת הליבה של בומברמן, שם השחקן נע בתוך שלב המוגדר מראש , מניח פצצות ומנסה להרוס מכשולים תוך הימנעות מאויבים.
המטרה הכללית של המשחק הוא להיכנס לתוך דלת ואז לגמור את השלב , לצבור כמה שיותר נקודות , ולא לאבד חיים וזמן.

4.1

Wall.h – קובץ כותרת המגדיר קיר במשחק.
Wall.cpp – מימוש ההתנהגות של הקיר, כולל זיהוי התנגשויות עם רובוטים ושומרים.
GameObject.h – קובץ כותרת שמגדיר מחלקת בסיס לאובייקטים סטטיים במשחק.
GameObject.cpp – מימוש המחלקה, כולל חישוב מיקום, ציור על המסך, וזיהוי התנגשויות.
SfmlManager.h – קובץ כותרת המגדיר מנהל משאבים של SFML, כולל טקסטורות וצלילים.
SfmlManager.cpp – מימוש הטענת קבצי תמונות וצלילים, ואחזורם לפי סוגי אובייקטים.
Rock.h – קובץ כותרת המגדיר מחלקת `Rock`, שמייצגת סלעים במשחק.
Rock.cpp – מימוש המחלקה, כולל טיפול בהתנגשויות עם רובוטים ושומרים.
MovingObject.h – קובץ כותרת למחלקת `MovingObject`, שמגדירה אובייקטים שיכולים לזוז.
MovingObject.cpp – מימוש המחלקה, כולל שינוי כיוון התנועה תוך מניעת קפיצות פתאומיות.
GameController.h – קובץ כותרת שמנהל את המשחק.
main.cpp – נקודת הכניסה למשחק, יוצרת את `GameController` ומפעילה את המשחק.
Information.h – קובץ כותרת המכיל מידע על מצב המשחק, כמו ניקוד, מספר שומרים ומצב החיים של הרובוט.
Information.cpp – מימוש ניהול חיי הרובוט, ספירת השומרים, עדכון השלב, הצגת נתוני המשחק על המסך, וטיפול במתנות.

Guard.h – קובץ כותרת שמגדיר את מחלקת `Guard`, המייצגת שומרים במשחק.
Guard.cpp – מימוש התנהגות השומרים, כולל מעקב אחרי הרובוט, שינוי כיוון בהתאם למיקומו, וטיפול בהתנגשויות.
Gift.h – קובץ כותרת שמגדיר מתנה כללית במשחק.
Gift.cpp – מימוש מחלקת `Gift`, כולל הורשה למחלקות מתנות ספציפיות.
Gift1.h – קובץ כותרת שמגדיר את המתנה הראשונה במשחק.
Gift1.cpp – מימוש האינטראקציה של `Gift1` עם הרובוט, כולל השפעתה בעת התנגשות.
Gift2.h – קובץ כותרת שמגדיר את המתנה השנייה במשחק.
Gift2.cpp – מימוש השפעת `Gift2`, שמעניקה לרובוט חיים נוספים.
Gift3.h – קובץ כותרת שמגדיר את המתנה השלישית במשחק.
Gift3.cpp – מימוש השפעת `Gift3`, המוסיפה זמן לשעון המשחק.
Gift4.h – קובץ כותרת שמגדיר את המתנה הרביעית במשחק.
Gift4.cpp – מימוש `Gift4`, שגורמת להסרת שומרים מהמשחק.
Bomb.h
קובץ כותרת שמגדיר את מחלקת הפצצה, כולל תכונות ופונקציות לניהול פיצוץ, זמן והתנגשות.
Bomb.cpp
מימוש מחלקת הפצצה, כולל ניהול זמן פיצוץ, אפקטים קוליים והתנגשויות עם דמויות וחפצים.
Button.h
קובץ כותרת שמגדיר מחלקת כפתור, כולל אירועים ולחיצות משתמש.
Button.cpp
מימוש הכפתור, כולל ציור הכפתור על המסך והתנהגות בעת לחיצה.
CountdownTimer.h
קובץ כותרת לטיימר שסופר זמן אחורה, כולל פונקציות עדכון ומעקב אחר הזמן שנותר.
CountdownTimer.cpp
מימוש הטיימר, כולל התחלה מחדש, עדכון הזמן והצגתו בפורמט של דקות ושניות
.**FirstWindow.h** הקובץ מכיל את ההכרזות עבור מחלקת `FirstWindow`. הוא מגדיר את הממשק של החלון הראשי במשחק, כולל משתנים כמו מיקום הכפתורים, פונקציות כמו `draw()` לציור הכפתורים, `customerChoice()` לטיפול בלחיצות על הכפתורים, ו-`playMusic()` לניגון מוזיקת רקע. הקובץ מגדיר גם את מבנה החלון ואת כל המתודות שקשורות אליו, כולל קריאה לטקסט של עזרה מתוך קובץ.

5. מבני נתונים עיקריים ותפקידיהם.

std::vector <std::unique_ptr<GameObject>> m_staticObjVec;

ווקטור שמחזיק את כלל האובייקטים הנייחים במשחק , תפקידו לשמור את המיקום שלהם כאשר האובייקט נקרא מן הקובץ , וכן מיקומו במקום הנוכחי בכל רגע נתון.

std::vector <std::unique_ptr<MovingObject>> m_movingObjVec;

ווקטור שמחזיק את כלל האובייקטים הנעים במשחק , תפקידו לשמור את המיקום שלהם כאשר האובייקט נקרא מן הקובץ , וכן מיקומו במקום הנוכחי בכל רגע נתון.

std::vector <std::unique_ptr<Bomb>> m_BombVec;

ווקטור שמחזיק את הפצצות במשחק , תפקידו לשמור את המיקום שלהם כאשר האובייקט נקרא מן הקובץ , וכן מיקומו במקום הנוכחי בכל רגע נתון.
הפצצות בדווקא הם מחוץ הווקטור הראשון כי יש להם התנהגות שאינה תואמת את הנעים ולא את הניחיים אלא משהו אמצעי.

SfmlManager

אובייקט שמחזיק את התמונות וקבצי השמע שSFML מטפל בהם

Information

אובייקט שמחזיק את המידע על המשחק

std::vector<Button> m_buttons;

6 אלגוריתמים הראויים לציון.

6.1

handleCollision ..

אלגוריתם בסיסי (מימוש כללי)

1. **זיהוי התנגשויות**
   - כשאובייקט נתקל באובייקט אחר, יש לבדוק את סוגו.
   - אם מדובר באובייקט סטטי, ההתנגשות מטופלת בהתאם לכללים שהוגדרו עבור אותו סוג.
   - אם מדובר באובייקט נע (כמו שחקן או אויב), ההתנגשות מטופלת דרך הפונקציה הווירטואלית `handleCollision`.
2. **מימוש `Double Dispatch`**
   - כדי למנוע `if-else` מסורבל לכל סוג של אובייקט, כל אובייקט מפעיל את `handleCollision`, שמעבירה את השליטה לפונקציה המתאימה (באמצעות פולימורפיזם).
   - דוגמה לפעולה:
     - אם `Robot` נתקל ב-`Rock`, הקריאה תהיה `robot.handleCollision(rock)`, שתפעיל את המימוש הספציפי של `&handleCollision(Rock)` ב-`Robot`.
3. **טיפול בהתנגשות (מימוש בפועל)**

- **Robot & Wall** → הרובוט לא יכול לעבור דרך הקיר, ולכן נדרשת חסימה.
- **Robot & Guard** → תלוי במנגנון המשחק, הרובוט יכול להיפגע, להיהרג, או להילחם בשומר.
- **Robot & Gift** → הרובוט אוסף את המתנה, ויש להפעיל את האפקט שלה.
- **Bomb & Object** → אם פצצה מתפוצצת על אובייקט, יש לבדוק אם יש להשמידו (למשל קיר הרוס).

סיכום האלגוריתם

1. מזהים התנגשויות בין שני אובייקטים.
2. משתמשים ב-`handleCollision` כדי להפנות את הבדיקה לאובייקט המתאים.
3. מיישמים טיפול שונה לכל מקרה של התנגשות בהתאם להיגיון המשחק.

6.2

Guard:: move

השומר במשחק מתנהג לפי מספר מצבים עיקריים:

1. **רדיפה אחרי הרובוט (מעקב אחר מיקומו)** – מתבצע באמצעות `trackRobotX()` ו-`trackRobotY()`.
2. **תנועה אקראית** – כאשר אין מידע ברור על מיקום הרובוט או כשהשומר פוגע במכשול.
3. **טיפול בהתנגשויות** – אם השומר פוגע במכשול או ברובוט, מופעלים מנגנוני טיפול בהתנגשות.

אלגוריתם הרדיפה אחר הרובוט

1. **קבלת מיקום הרובוט:**
   - השומר מקבל את מיקום הרובוט מהמשחק דרך `m_information.getRobotLoc()`.
2. **עיגול המיקום לרשת המשחק:**
   - מיקום הרובוט (Grid) **מעוגל** לגריד של המשחק כדי לוודא שהשומר נע בדיוק לפי הרשת.
   - החישוב מתבצע כך שהערכים יהיו מיושרים לרשת בגודל `m_pixelSize` (50 פיקסלים x50 כגון).
   - למשל, אם הרובוט נמצא ב- `(126 ,207)`, זה יעוגל ל- `(100 ,200)`.
3. **חישוב כיוון התנועה:**
   - אם השומר רחוק מהרובוט (יותר מ-10% מגודל X תא ברשת), הוא קודם כל ינוע **אופקית** לעבר הרובוט **באופן משמעותי** על ציר X.

- אחרת, הוא יתחיל לנוע **אנכית** לכיוונו.

4. **קביעת כיוון השומר:**
   - אם הרובוט **מימין** לשומר, השומר יזוז ימינה `(direction.x = 1)`.
   - אם הרובוט **משמאל** לשומר, השומר יזוז שמאלה `(direction.x = -1)`.
   - אם אין צורך לזוז בציר X, השומר יתחיל לנוע בציר Y (למעלה או למטה).
5. **עדכון כיוון התנועה:**
   - שמשנה `setDirection(direction)`, הכיוון החדש נשלח לפונקציה את כיוון התנועה בפועל.

6.3

מחיקת אובייקטים מהווקטור :

המערכת משתמשת ב-`std::erase_if` (שהתווסף ב-C++20) כדי למחוק את האובייקטים בצורה יעילה.
הפונקציה `IsDead()` היא פונקציה של האובייקטים שמחזירה `true` אם האובייקט כבר לא רלוונטי וצריך להסיר אותו מהמשחק.
השימוש ב-`std::erase_if` מונע בעיות כמו מחיקה בתוך לולאה תוך כדי מעבר על הוקטור (שזה עלול לגרום לקריסה).

6.4

תזוזה לפי שעון
בכל פעם שמתבצע אירוע במשחק (כגון לחיצה על מקש או תנועה של דמות), הפונקציה `handleEvent()` מופעלת כדי לעדכן את מצב המשחק.

השלב הראשון בפונקציה הוא מדידת הזמן שחלף מאז העדכון הקודם באמצעות `m_gameClock.restart()`. הדבר מאפשר חישוב תנועה אחידה, כך שאם המשחק רץ בקצב שונה (FPS משתנה), האובייקטים עדיין ינועו באותו קצב יחסי.

לאחר מכן, הפונקציה עוברת על כל האובייקטים הנעים במשחק (כגון הרובוט והשומרים). אם אחד מהם מסומן כאובייקט שצריך לאתחל את מיקומו מחדש (למשל, אם שומר פגע ברובוט), מתבצע אתחול מיקום באמצעות `restartObjPlace()`.

בהמשך, כל אובייקט מעדכן את כיוון התנועה שלו (`updateDirection()`). לדוגמה, השומרים מחליטים אם לעקוב אחרי הרובוט או לזוז בצורה רנדומלית.

לאחר עדכון הכיוון, מתבצע זיהוי התנגשויות בין האובייקטים (`handleCollisionController()`). אם שומר פוגע בקיר, ברובוט או באובייקט אחר, הוא עשוי לשנות כיוון או להפעיל פעולה אחרת.

לבסוף, כל אובייקט נע לפי הזמן שחושב קודם (`move(deltaTime)`). השומרים עשויים להתקדם לעבר הרובוט אם הם מזהים אותו קרוב אליהם, או להמשיך לנוע בכיוון שנקבע להם מראש.

6.5

חישוב מיקומי הפתורים ביחס לגודל החלון.
הקוד מחשב את המידות והמיקום של שלושה כפתורים בתוך חלון הגרפי, תוך שמירה על מראה מאוזן ומרוכז.

1 חישוב גודל הכפתורים:

- רוחב הכפתורים נקבע לפי `m_width / 1.5`, כלומר, הם תופסים כשני שלישים מרוחב החלון.
- גובה הכפתורים מחושב כ- `m_height / 4`, כלומר, כל כפתור תופס כרבע מגובה החלון.

2 חישוב מיקום הכפתורים:

כדי לוודא שהכפתורים ממוקמים בצורה אסתטית, יש צורך למרכז אותם גם אופקית וגם אנכית.

אופקית (ציר X):

- המטרה היא שהכפתורים יהיו ממורכזים באמצע הרוחב של החלון.
- החישוב `(m_width - buttonWidth) / 2` מוצא את המיקום ההתחלתי של הכפתור הראשון כך שהוא יהיה באמצע המסך.

אנכית (ציר Y):

- הכפתורים מסודרים אחד מתחת לשני, עם רווחים ביניהם (`spacing = 20.f`).
- החישוב `(m_height - 3 * buttonHeight - 2 * spacing) / 2` ממקם את הכפתור הראשון כך שכל שלושת הכפתורים יהיו ממורכזים בגובה המסך.
  - `buttonHeight * 3` → גובה של שלושת הכפתורים יחד.
  - `spacing * 2` → שני רווחים בין הכפתורים.
  - החיסור מחלק את יתרת הגובה בין החלק העליון והתחתון של החלון.

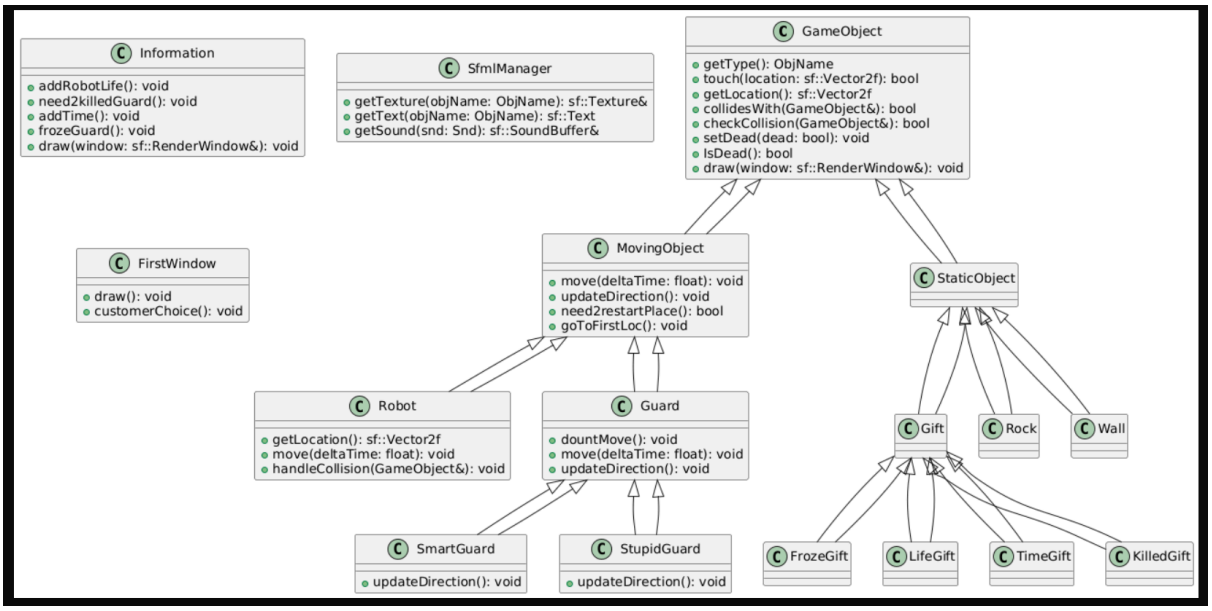
3 מיקום הכפתורים הסופיים:

- הכפתור הראשון (`button1`) ממוקם בנקודה `(x, y)`.
- הכפתור השני (`button2`) ממוקם מתחתיו, כך שהמרחק הוא `buttonHeight + spacing`.

- הכפתור השלישי (`button3`) ממוקם עוד יותר למטה, כלומר `2 *` `(buttonHeight + spacing)`.

לסיכום, הכפתורים מסודרים בטור במרכז המסך, עם רווחים שווים ביניהם, תוך ניצול נכון של השטח הקיים כדי לשמור על עיצוב נקי ומסודר.

7.1
תיכון הפרוייקט 👍



מבנה המערכת

המערכת מורכבת ממספר מחלקות המחולקות לקטגוריות עיקריות:

1. **אובייקטים כלליים**
   - `GameObject` - מחלקת בסיס לכל האובייקטים במשחק. כוללת תכונות כמו מיקום, סוג האובייקט, ותמיכה בהתנגשויות כפולות (Double Dispatch).
   - `StaticObject` - מחלקת בסיס לאובייקטים סטטיים כמו קירות וסלעים.
   - `MovingObject` - מחלקת בסיס לאובייקטים נעים, כמו `Robot` ו-`Guard`.
2. **דמויות במשחק**
   - `Robot` - הדמות הנשלטת על ידי השחקן.
   - `Guard` - מחלקת בסיס לשומרים במשחק.
     - `SmartGuard` - שומר חכם שמחשב מסלול לעבר השחקן.
     - `StupidGuard` - שומר טיפש שמתנהג באופן אקראי.
3. **אובייקטים אינטראקטיביים**
   - `Gift` - מחלקת בסיס למתנות במשחק.
     - `FrozeGift` - מקפיא שומרים.
     - `LifeGift` - מוסיף חיים לרובוט.
     - `TimeGift` - מוסיף זמן למשחק.
     - `KilledGift` - מחסל שומרים.

4. **מערכות ניהול**
   - `SfmlManager` - מנהל משאבים (טקסטורות, צלילים וכו').
   - `Information` - מנהל מידע המשחק (חיים, נקודות, זמן וכו').
   - `CountdownTimer` - מנהל שעון המשחק.
   - `FirstWindow` - מנהל את תפריט הפתיחה.

אינטראקציה בין המחלקות

- `Robot` ו-`Guard` יורשים מ-`MovingObject`, ומטפלים בהתנגשויות מול אובייקטים שונים (כגון `Gift`, `Wall` וכו').
- `Gift` מופיעים כאשר מתקיימים תנאים מסוימים (כגון הריגת שומר).
- `SfmlManager` מספק לכל המחלקות גישה לטקסטורות, צלילים וכו'.
- `Information` מכיל את המידע המרכזי כמו חיי השחקן, מספר השומרים שנותרו וזמן המשחק.

8. באגים ידועים.

לא נמצאו באגים בתכנית.

9. הערות אחרות.

אין כרגע הערות .