## תיאור כללי של התוכנית:

התוכנית מורכבת מארבע מחלקות:

**Main** –

אחראית על ריכוז והפעלת הפרויקט. קולטת את מצב הפאזל, ופותרת אותו באמצעות ארבעה אלגוריתמי חיפוש.

**EightPuzzle –**

הדרך בה בחרתי לייצג מצב של הפאזל.

**מצב** הפאזל מיוצג באמצעות תכונת מערך חד מימדי, **והמעברים** מיוצגים באמצעות רשימה של "בנים" – אוביקטי EightPuzzle שהם השלבים הבאים האפשריים מהמצב הזה.

המחלקה הזו היא מעין רקורסיבית – מכילה כתכונה הצבעה לאוביקט מהטיפוס שלה, שהוא מייצג את המצב הקודם של הפאזל, וזה לצורך קבלת דרך הפתרון כאשר ימצא כזה.

**UnInformedSearch –**

מחלקה האחראית על מימוש הפתרון באמצעות האלגו' העיוורים.

רכזתי את זה במחלקה אחת ע"מ למנוע חזרת קוד, ומתוך הבנה שההבדלים בינהם במימוש עצמו הם מינוריים.

מה ששונה מימשתי בתוך השיטה עצמה, כמו הגדרת מבנה הנתונים שמיצג את הfrontier.

**HeuristicSearch –**

מחלקה האחראית על מימוש הפתרון באמצעות אלגו' היוריסטיים.

הבנאי של המחלקה מקבל את סוג הפתרון הנדרש ועל פי זה מאתחל את סוג הקדימות של התור frontier, וזה מה שיוצר את ההבדל במימוש שני האלגוריתמים, ומונע חזרתיות קוד.

**היוריסטיקה שבחרתי:**

מעריכה את מספר הצעדים המינימלי ע"י שסוכם את מספר הצעדים המינימלי הנדרש כדי להביא כל אריח שלא במקומו למקומו.

מניחה שכל אריח צריך 2 צעדים מינימום כדי להגיע למקומו, ונותנת בונוס (צעד אחד פחות) לאריחים שנמצאים במרחק של צעד אחד מהמיקום המטרתי שלהם – כלומר – נמצאים באחד האריחים השכנים, ובמקומם המטרתי נמצא 0 ולכן הוא נגיש בצעד אחד.

יוריסטיקה זו **קבילה**, מכיוון שהיא אינה מעריכה יתר על המידה את עלות ההגעה ליעד. היא פשוט סופרת את מספר הצעדים הדרושים כדי להביא את כל האריח שלא במקומם למקומם, ללא קשר למסלול שנבחר.

יוריסטיקה זו **עקבית**, מכיוון שהיא תמיד מחזירה את אותו ערך עבור מצבים זהים. היא לא תשתנה אם תעבור במסלול מעגלי.

כפי שניתן לראות בדוגמאות הבאות -->
עבור A* נותנת הפתרון באמת האופטימלי.
עבור GBFS הפתרון הוא לא תמיד (לרוב לא ☹ ) האופטימלי מבחינת מספר צעדים. (אבל ניתן לשים לב שהוא האופטימלי מבחינת מספר הצמתים הנחקרים, אז אולי מבחינת סיבוכיות מקום הוא אופטימלי...)