ינואר 2017

מגישה: הגר זיסר

תעודת זהות: 206377087

מנחה: כוכב בראימן

רכזת: לימור שיאון

בית ספר: עירוני ד' על שם אהרון קציר

Pathfinding.exe

עבודת גמר בנושא אלגוריתמי מציאת מסלולים במשחקי מחשב

תוכן עניינים

[מבוא 3](#_Toc472959749)

[הבעיה המרכזית 3](#_Toc472959750)

[מבוא לגרפים 4](#_Toc472959751)

[ייצוג מפה על ידי גרף 5](#_Toc472959752)

[מבוא לאלגוריתמי חיפוש 6](#_Toc472959753)

[אלגוריתם חיפוש לרוחב (Breadth-first search) 7](#_Toc472959754)

[אלגוריתם דייקסטרה (Dijkstra’s algorithm) 10](#_Toc472959755)

[חיפוש היוריסטי (Greedy Best-first search) 13](#_Toc472959756)

[אלגוריתם A\* 16](#_Toc472959757)

[מימוש איי סטאר 18](#_Toc472959758)

[טיפוס Node 19](#_Toc472959759)

[מחלקת Pathfinder 21](#_Toc472959762)

[מחשבות נוספות 27](#_Toc472959766)

[מימוש המשחק 28](#_Toc472959767)

[הגדרות 28](#_Toc472959768)

[דיאגרמת המחלקות 30](#_Toc472959769)

[מחלקות עזר 31](#_Toc472959770)

[טופס Main 32](#_Toc472959771)

[סיכום 33](#_Toc472959772)

[ביבליגרפיה 34](#_Toc472959773)

[נספחים 35](#_Toc472959774)

[מבוא לתכנות מונחה עצמים ב C# 35](#_Toc472959775)

[קוד מחלקת Node 37](#_Toc472959776)

[קוד מחלקת Pathfinder2 38](#_Toc472959777)

[טופס Main 42](#_Toc472959778)

[מחלקות עזר 53](#_Toc472959779)

# מבוא

מטרת הפרויקט המקורית הייתה למידת השפה C# ותכנות מונחה עצמים על מנת לבנות משחק. התוצאה הסופית היא פרויקט המדגים את הידע שרכשתי בתחום מציאת מסלולים ומימוש האלגוריתם שכתבתי בעזרת ידע זה. המשחק נבנה סביב האלגוריתם ומדגים את יתרונותיו.

החלק התיאורטי של עבודה זו עוסק באלגוריתמי מציאת מסלולים המתבססים על תורת הגרפים. מבוא לתכנות מונחה עצמים מצורף בנספחים. החלק המעשי מציג את מימוש התוכנית, בדגש על מחלקות האלגוריתם שכתבתי.

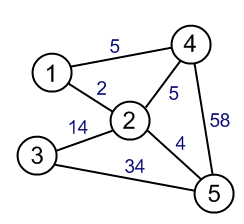
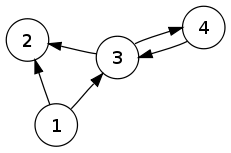
## הבעיה המרכזית

אחד הנושאים הראשונים שנגשתי אליהם בעת תכנון הפרויקט הוא כיצד האויבים במשחק יעבדו. הקונספט של משחק דו-ממדי ממבט על דרש דרך בה האויבים יכולים לעקוב אחר פעולות השחקן ובמידת הצורך לרדוף אחריו. בתיאוריה להגיע ממקום למקום זהו קונספט פשוט: אם יש מכשול בדרך אז עוקפים אותו. כשמגיעים למימוש, לעומת זאת, מתחילות לצוץ שאלות שהתשובה אליהן לאו דווקא ברורה. כיצד עוקפים מכשול מסוים? האם עקיפת מכשולים תמיד תוביל ליעד? איך מוצאים את הדרך המהירה ביותר? על מנת לענות על שאלות אלו חקרתי תחום במדעי המחשב בשם Pathfinding, אשר מתייחס למציאת מסלולים על ידי שימוש בתורת הגרפים. חקירת מסלולים הוא תחום מרחב המיושם מילולית באפליקציות התמצאות כגון Waze, אך גם בתחומים אחרים כגון רשתות וניתוב, ובינה מלאכותיות. בעבודה אסביר את העקרונות המרכזיים בתחום זה.

# מבוא לגרפים

גרף הוא קונספט מתמטי בעל שימוש נרחב במדעי המחשב. קיימות הרבה גרסאות של גרפים, אך כולן מוגדרות על ידי שני מאפיינים: קודקודים וצלעות. כל צלע מצייגת קישור בין שני קודקודים, ובאופן דומה אם שני קודקודים הם שכנים יש צלע המחברת ביניהם.

קיים הבדל בין גרפים מכוונים ולא מכוונים. בגרפים מכוונים הצלע מובילה מקודקוד אחד לשני בכיוון אחד, לכן השם, כאשר בגרפים לא מכוונים כל צלע היא דו כיוונית. בנוסף, גרף משוקלל הוא גרף בו לכל צלע יש משקל המייצג מספר, עלות, אורך או כל מדד אחר.



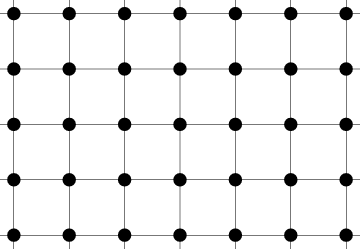
דוגמאות של גרף לא מכוון משוקלל (ימין) וגרף מכוון (שמאל)

# ייצוג מפה על ידי גרף

הצעד הראשון בעבודה עם אלגוריתם הוא להבין מה הקלט והפלט. באלגוריתם חיפוש מסלול הקלט הוא גרף המוגדר על ידי רשימת קודקודים וצלעות. אם הגרף משוקלל יהיו לצלעות גם ערך מספרי. חשוב להבין שמבחינת האלגוריתם אין אף חשיבות למה הגרף מייצג. הפלט יהיה הקודקודים והצלעות המרכיבים את המסלול.

מכאן ניתן להסיק כי קיימות דרכים רבות לייצג את מפת המשחק בתור גרף. ניתן לייצג חדרים בתור קודקודים ודלתות בתור צלעות. במקרה של עיר ניתן לייצג צמתות בתור קודקודים ורחובות בתור צלעות. הגרף יכול להיות מכוון ועל ידי כך לייצג תנועה חד כיוונית או משוקלל ובכך לייצג קושי תנועה. ההצגה השכיחה ביותר במשחקים היא של טבלה. טבלה היא מקרה פרטי של גרף מכוון משוקלל, בו תאים הם קודקודים ובין התאים הסמוכים מוגדרות הצלעות. לכל תא יש שמונה או ארבעה שכנים, בתלות או קיימת או לא קיימת תנועה אלכסונית.

הסיבה שאלגוריתמי החיפוש מותאמים לעבודה עם גרף היא מאוד פשוטה – גרף יכול לייצג כמעט הכל.



טבלה בתור גרף

# מבוא לאלגוריתמי חיפוש

כאשר מתייחסים לתחום מציאת מסלולים במדעי המחשב, וספציפית לאלגוריתמי חיפוש, הכוונה לרצף פעולות שניתן לבצע על גרף נתון המחזיר רצף קודקודים המהווה דרך מקודקוד אחד לאחר. בתחום הזה קיימים הרבה פתרונות אפשריים, בעלי אופן פעולה, עקרונות, ויעדים שונים. אנסה להסביר כיצד מספר אלגוריתמי חיפוש עובדים ועל איזה עיקרון הם מבוססים.

כל האלגוריתמים שאציג בנויים על עיקרון של חזית מתפשטת המיוצגת על ידי תור. בכל איטרציה של לולאה החיפוש מוציא קודקוד מהתור, מוצא את שכניו, מעריך כל שכן בנפרד ואם נמצא מתאים על פי התנאים מוסיף אותו לתור. החיפוש מפסיק כאשר כל הקודקודים נבדקו, דבר המיוצג על ידי כך שהתור ריק. לחילופין, החיפוש נגמר כאשר קודקוד היעד נבדק.

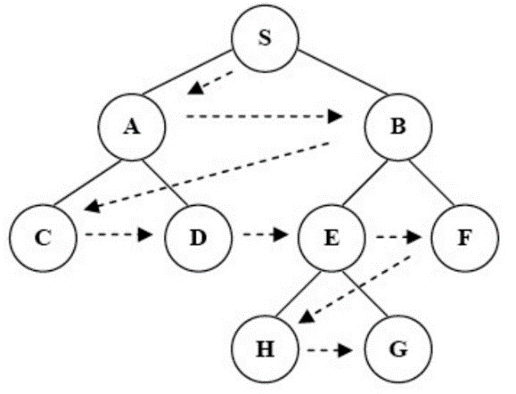
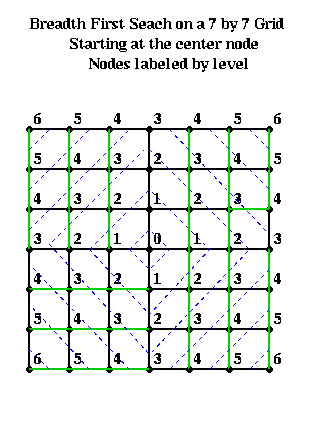
הווריאציות השונות של האלגוריתם תלוית ביישום הפונקציה של הוצאה מהתור. כאשר נבחר האלמנט הישן ביותר, כמו בדוגמה הנתונה, ייתקבל חיפוש לרוחב. כאשר נבחר האלמנט החדש ביותר ייתקבל חיפוש לעומק, Best-First search כאשר נבחר האלמנט בעל הערך הכי טוב ועוד.

ניתן לחלק חיפושים לשני סוגים: חיפוש עץ וחיפוש גרף. בחיפוש גרף קיים תנאי הבודק האם קודקוד נבדק לפני שהוא מתווסף לתור על מנת למנוע חזרות. חיפוש עץ, לעומת זאת, יכול לעבור על אותו הקודקוד מספר פעמים, אפילו אינסוף אם הגרף מכיל לולאה, אבל כל מעבר מתאים לשכן שונה בעץ. לאלגוריתמים המוצגים במסמך זה מצב בו יש חזרה לא מתאים מכיוון שכאשר קודקוד מסויים נבדק הדרך המובילה אליו היא בוודאות הטובה ביותר. לפיכך, לכולם תנאי המונע מעבר חוזר, דבר המגדיר אותם כחיפושים גרפיים.

בפרקים הבאים אשווה אלגוריתמים שונים העונים על בעית מציאת המסלולים. אתייחס לאופן בו מתבצעת פעולת ההוצאה מהתור, למאפיינים אשר נשמרים בכל קודקוד ולתנאי המונע חזרה.

## אלגוריתם חיפוש לרוחב (Breadth-first search)

האופן הכי בסיסי ליישם חיפוש מבוסס על התפשטות בשכבות עד ההגעה אל היעד נקרא "חיפוש לרוחב". החיפוש מתחיל בקודקוד מסוים, וכל פעם השכבה הבאה של שכנים הופכת לחזית.



הדגמת ריצת ההאלגוריתם על גרף כללי וטבלה

Breadth\_first\_search(graph, start, finish)

{

frontier = new queue()

frontier.enqueue(start)

start.parent = null //property

while(frontier.count != 0)

{

current = frontier.dequeue() //get next function

if(next.equals(finish)) //early exit

break

foreach(node next in graph.neighbors(current))

{

if(next.parent == null) //check condition

{

next.parent = current

frontier.enqueue(next)

}

}

}

**עצירה מוקדמת:** רלוונטית כאשר רוצים למצוא דרך לקודקוד מסויים. ברירת המחדל היא למצוא את הדרך מקודקוד מסויים לכל הקודקודים האחרים. כאשר אין בכך צורך ניתן להפסיק את החיפוש לאחר שמגיעים ליעד. במקרים שבהם דרך לא קיימת החיפוש יימשך כרגיל – עד שנבדקו כל הקודקודים, מצב אשר נקרא "הגרוע ביותר" לצורך הערכת סיבוכיות.

פעולת ההוצאה בחיפוש לרוחב מחזירה את הקודקוד הותיק ביותר, ולכן יהיה בעל מספר הצעדים הקטן ביותר הנדרש על מנת להגיע אליו. התוצאה היא שהאלגוריתם מתרחב באופן סימטרי בכל הכיוונים. המאפיין היחיד הנשמר לכל קודקוד הוא השכן ממנו התווסף, והוא גם התנאי המונע מעבר חוזר. אם קיים הורה משמע הקודקוד נבדק ואין צורך לבצע בדיקה חוזרת.

סיבוכיות הזמן של חיפוש לרוחב היאO(|V| + |E|), מאחר שבמקרה הכי גרוע (אין מסלול) כל קודקוד וצלע ייבדקו פעם אחת. |V| הוא מספר הקודקודים ו |E| מספר הצלעות. הטווח של O(|E|) נע בין O(1) ל )²O(|V|, בתלות בכמות הצלעות בין קודקודים בגרף. סיבוכיות המרחב תלויה ביצוג של הגרף, אך בדוגמה שנתתי היא תהיה O(|V|) מכיוון שנשמרת כמות זיכרון קבועה לכל קודקוד.

חיפוש לרוחב לאו דווקא בונה את הדרך אך מסביר איך לבקר בכל הגרף. זהו אלגוריתם עם שימושים רבים מעבר למציאת מסלולים. אפשר להשתמש בו בתור חיפוש אם שומרים לכל קודקוד את ה"הורה" שלו – השכן שלו אשר ממנו ניתן להגיע אליו. ניתן לשחזר את הדרך מקודקוד המטרה על ידי מעקב אחרי ההורה, כמו רשימה מקושרת רגילה.

Get\_path(graph, finish)

{

path = new list()

node next = finish.parent

while(next != null) //traces the parent back until the start

{

path.add(next)

previous = next.parent

}

path.reverse()

return(path) //if a path wasnt found the path returned will be empty

}

פסאודו קוד המשחזר את המסלול הנבנה על ידי חיפוש

זהו האלגוריתם הפשוט ביותר למציאת דרכים העובד על כל סוג של גרף. הוא נחשב לפשוט ביותר מכיוון שהוא ממצה את כל המסלולים המכוונים האפשריים ומחזיר את האחד הנגמר ביעד. במקרה בו הגרף לא משוקלל זו תהיה הדרך הקצרה ביותר, אך בגרף משוקלל תיתכן דרך יותר קצרה מפני שהאלגוריתם מתייחס רק לשכבה או מספר הצעדים של מהקודקוד ההתחלתי. על מנת למצוא את הדרך האופטימלית בגרף משוקלל, בו לצלעות יש ערכים, יש צורך באלגוריתם המתייחס לערכים אלו.

## אלגוריתם דייקסטרה (Dijkstra’s algorithm)

יצירתו של אדסחר דייקסטרה אשר פורסמה ב 9195, לאלגוריתם הזה אלמנטים נוספים על חיפוש לרוחב אשר מאפשרים עבודה עם גרף משוקלל. המשמעותי ביותר הוא שמירת סכום ערכי הצלעות אשר מרכיבות את המסלול אל אותו הקודקוד בתור משתנה הנקרא עלות תנועה. משתנה זה מאפשר להעריך כל דרך מתמטית על מנת למצוא את הטובה ביותר.

Dijkstra’s\_algorithm(graph, start, finish)

{

frontier = new priorityQueue()

start.cost = 0

start.parent = null

frontier.enqueue(start, start.cost)

while(frontier.count != 0)

{

current = frontier.dequeue()

if(next.equals(finish))

break

foreach(node next in graph.neighbors(current))

{

newCost = current.cost + next.distance(current)

if(next.parent == null || newCost < next.cost)

{

next.cost = newCost

next.parent = current

frontier.enqueue(next, next.cost)

}

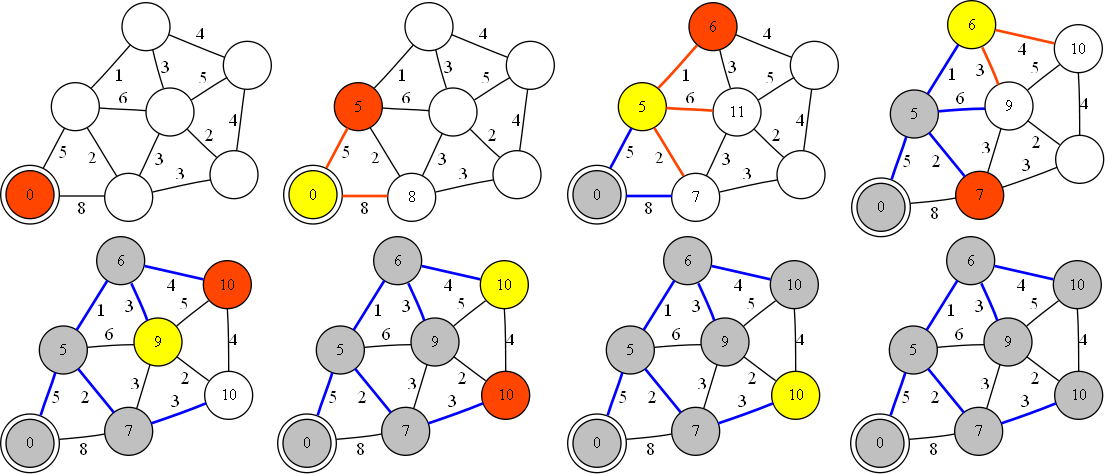
}

}

מימוש כללי של האלגוריתם של דייקסטרה המשתמש בתור עדיפויות

**תור עדיפויות:** תור אשר סדר איבריו נקבע על פי עדיפות מספרית הניתנת לכל איבר בעת הכנסתו לתור. הצצה והוצאת איבר מחזירים את האיבר בעל העדיפות המספרית הגבוהה ביותר. קיימים מספר מימושים של תור עדיפויות בעלי סיבוכיויות שונות. הסיבוכיות של ההכנסה (או ההוצאה, תלוי במימוש) היא או O(n) או O(log n) מכיוון שבהכנסה יש צורך לעבור על האיברים הקיימים בתור על מנת לקבוע היכן להכניס את האיבר החדש.

פעולת ההוצאה באלגוריתם של דייקסטרה מחזירה את הקודקוד בעל עלות התזוזה הכוללת הנמוכה ביותר, לפיכך גם הקרוב ביותר להתחלה. האלגוריתם ימצא את הדרך הקצרה ביותר לכל קודקוד שיחקור. נשמרים השכן ממנו התווסף ועלות התנועה. התנאי המונע חזרה שונה, ועכשיו יש שני תנאים אפשריים לבדיקה חוזרת. האחד הוא אם הקודקוד לא נבדק, כמו בחיפוש לרוחב. השני הוא אם הדרך החדשה שנבדקת יותר טובה מהדרך השמורה.



הדגמת ריצת האלגוריתם על גרף לא מכוון על מנת למצוא את כל המסלולים מקודקוד מסויים

לאלגוריתם של דייקסטרה יש הרבה מימושים שונים בעלי סיבוכיות שונה ויעוד שונה. באלגוריתם המקורי לא היה שימוש בתור עדיפויות, ולכן הסיבוכיות שלו הייתה )²O(|V| בהשוואה ל O(|V|log |V| + |E|) כאשר משתמשים בערימת פיבונאצ'י (סוג מבנה נתונים המשמש מטרה דומה לתור עדיפויות). בדוגמה הנתונה סיבוכיות המקום תהיה O(|V|).

השימוש במשתנה עלות התנועה כעדיפות בתור מבטיח כי הקודקודים בעלי הערך הכי טוב נבדקים קודם, ולפיכך החזית מתפשטת על פי המרחק מהמרכז ולא סדר ההכנסה. במקרה הזה הדרך שתימצא תמיד תהיה הקצרה ביותר.

## חיפוש היוריסטי (Greedy Best-first search)

כאשר יש צורך למצוא דרך בין שני מקומות ולא בין מקום אחד לרבים ניתן לגרום לחזית להתפשט בכיוון המבטיח ביותר. האופן שבו קובעים את המשתנה ההיוריסטי הזה תלוי בסוג הגרף – אם מדובר בעץ ניתן לבחור בצלע הקצרה לקודקוד הנוכחי, אך האלגוריתם הכי אפקטיבי כאשר ניתן להעריך את הבחירה של כל קודקוד ביחס לכל הגרף. לדוגמה, במקרה של טבלה המשתנה ההיוריסטי יוערך על ידי מדידת מרחק מקודקוד היעד.

האלגוריתם ההיוריסטי מתקדם על פי הבחירה הטובה ביותר בכל שלב מקומי בתקווה למצוא דרך התהווה פתרון. אלגוריתם כזה נקרא Greedy (חמדן). ברוב המקרים אלגוריתם חמדן לא ימצא את הפתרון האופטימלי הכללי, אבל יש סיכוי כי ימצא פתרונות מקומיים המדמים כזה בזמן סביר.

Greedy\_best\_first\_search(graph, start, finish)

{

frontier = new priorityQueue()

start.cost = Heuristic(start, finish) //heuristic function

start.parent = null

frontier.enqueue(start, start.cost)

while(frontier.count != 0)

{

current = frontier.dequeue()

if(current.equals(finish))

break

foreach(node next in graph.neighbors(current))

{

if(next.parent == null)

{

next.cost = next.Heuristic(next, finish)

next.parent = current

frontier.enqueue(next, next.cost)

}

}

}

int Heuristic(a, b) //Manhattan distance on a square grid

{

return(Math.Abs(a.X - b.X) + Math.Abs(a.Y - b.Y))

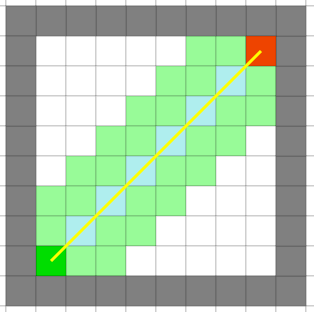
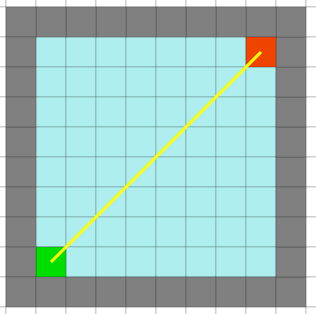
}

**מרחק מנהטן:** משתנה היוריסטי בגרפים מסוג טבלה. המרחק הוא סכום הפרשי הקורדינטות של שתי הנקודות. כאשר אין תנועה אלכסונית מרחק מנהטן יחזיר את המרחק הקצר ביותר מהיעד בלי התחשבות במכשולים. השם מבוסס על הגיאוגרפיה המוכרת של רחובות מנהטן.

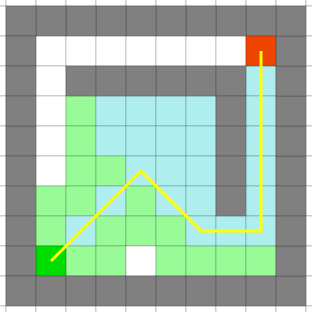
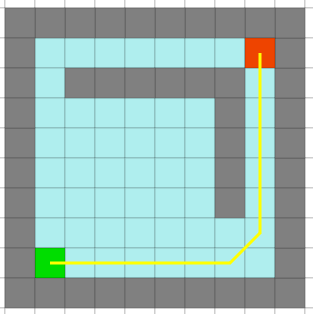
**Best-first search:** אלגוריתם המתקדם כל איטרציה לקודקוד המבטיח ביותר על פי חוק קבוע מראש. החוק הזה בדרך כלל מיוצג על ידי ערך מספרי משתנה התלוי באלגוריתם. אלגוריתם של דייקסטרה המשתמש בתור עדיפויות, למשל, משתמש בסכום עלות התנועה. כאשר BFS מסתמך אך ורק על משתנה היוריסטי הוא נקרא Greedy Best-first search.

בדומה לאלגוריתם של דייקסטרה, חיפוש היוריסטי מחזיר את הקודקוד בעל ערך התזוזה הטוב ביותר, רק שהפעם הוא המרחק המשוער מהסוף. האלגוריתם יתפשט בכיוון הכי מבטיח, היעד, ורק אם לא יגיע אליו יחזור ויבדוק דרכים שונות. אין חובה לשמור את המשתנה ההיוריסטי, אך בדוגמה שלי בחרתי לשמור אותו בנוסף להורה. אין טעם לבדוק מחדש קודקודים מכיוון שהמשתנה ההיוריסטי לא משתנה, ולפיכך התנאי היחיד הוא האם קיים הורה.

חיפוש היוריסטי לא תמיד ימצא את הדרך הטובה ביותר, ובמספר מימושים יכול להיתקע ולא למצוא דרך בכלל, אך יעשה זאת יותר מהר מחיפוש רוחבי. היתרונות והחסרונות של האלגוריתם נעשים ברורים כאשר הדרך האופטימלית אינה לינארית.

כאשר הדרך לא חסומה ניתן לראות כי אלגוריתם היוריסטי (מימין) מבקר רק בקודקודים הרצויים

ההחלטות המקומיות מובילות לדרך לא אופטימלית, בהשוואה לחיפוש רוחב המתייחס למכשול

## אלגוריתם A\*

השימוש בדייקסטרה תמיד יוביל לדרך האופטימלית, אך הוא מבזבז זמן בלחקור כיוונים לא מבטיחים. אלגוריתם היוריסטי חוקר כיוונים מבטיחים אבל הוא לאו דווקא ימצא את הדרך האופטימלית. איי סטאר משתמש גם בעלות התנועה הצבורה וגם במרחק המוערך ליעד על מנת למצוא את המסלול.

A\_star\_algorithm(graph, start, finish)

{

frontier = new priorityQueue()

start.g = 0 //g -> cost value to reach the node

start.h = start.Heuristic(finish) // h -> heuristic value

start.f = start.g + start.h //f -> combined movement cost

start.parent = null

frontier.enqueue(start, start.f)

while(frontier.count != 0)

{

current = frontier.dequeue()

current.state = closed //node state

if(current.equals(finish))

break

foreach(node next in graph.neighbors(current))

{

newCost = current.g + next.distance(current)

if(next.state != closed || newCost < next.g)

{

next.g = newCost

next.h = next.Heuristic(finish)

next.f = next.g + next.h

next.parent = current

next.state = open

frontier.enqueue(next, next.f)

}

}

}

אלגוריתם איי סטאר הוא השילוב של דייקסטרה וחיפוש היוריסטי. המשתנה על פיו מוצאים איברים מהתור מייצג את הערך הכולל של הדרך, מכיוון שהוא הסכום של עלות התנועה מההתחלה והמרחק המוערך מהסוף. התוצאה היא שהאלגוריתם יתקדם קודם כל בכיוון היעד כמו אלגוריתם חמדן, אך כאשר הדרך מתארכת יותר מדי (עלות תנועה גבוהה מצטברת) ייבדקו כיוונים אחרים. כל עוד המשתנה ההיוריסטי לא גדול יותר מהמרחק האמיתי ליעד האלגוריתם תמיד ימצא את הדרך הטובה ביותר, ובדרך כלל יותר מהר מדייקסטרה – למרות שבמקרה "הגרוע ביותר" איי סטאר יחקור את כל הקודקודים גם כן.

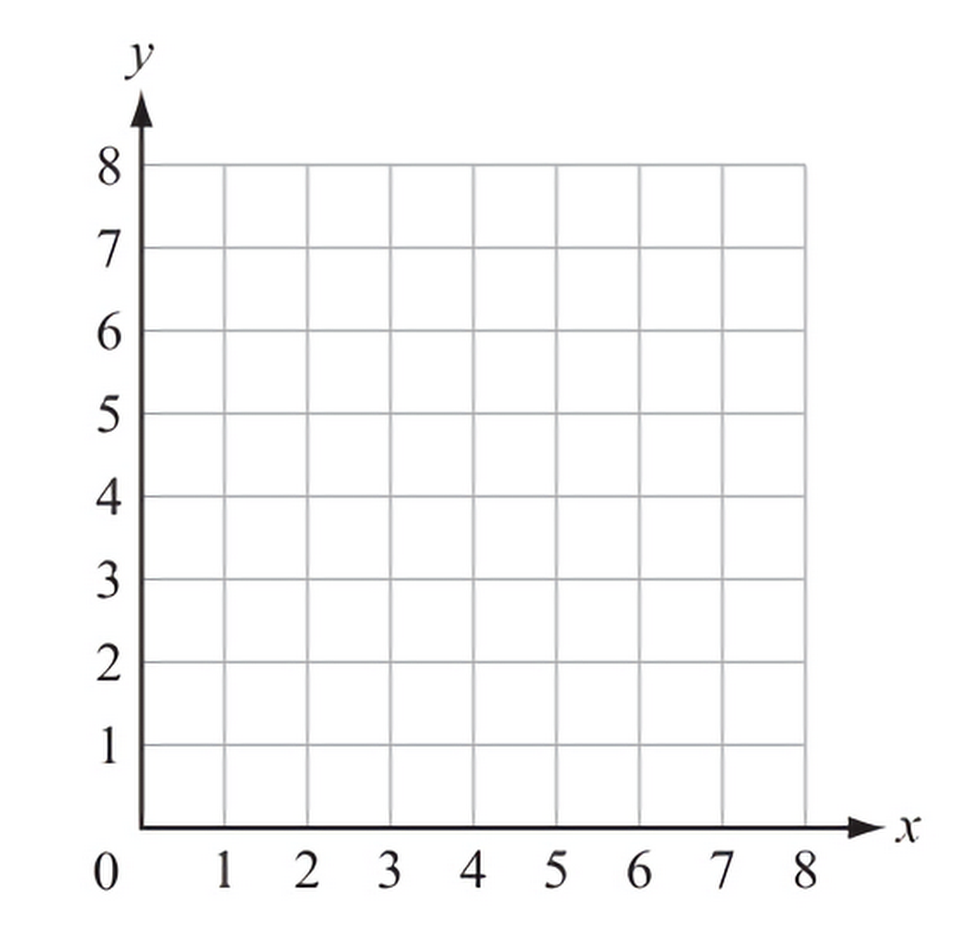
מכיוון שהאלגוריתם מסתמך גם על מרחק תנועה ומשתנה היוריסטי יש חשיבות לאופן בו ההיוריסטיקה מתבצעת, והיחס בין שני המשתנים. על מנת למצוא את הדרך האופטימלית המרחק המוערך מכל קודקוד ליעד חייב להיות קטן יותר או שווה למרחק האפשרי האמיתי, אחרת האלגוריתם יכול להתעלם מהפתרון האופטימלי עקב הערכת יתר של המרחק. מקרי הקצה מדגימים רעיון זה היטב. באלגוריתם של דייקסטרה, אשר ניתן להתייחס אליו כמקרה פרטי של איי סטאר, המרחק המוערך הוא אפסי וכפי שכבר הוסבר תמיד ימצא את הדרך הטובה ביותר. בחיפוש חמדן, בו מתייחסים רק למרחק המוערך, המסלול המוחזר בדרך כלל לא יהיה אופטימלי. ניתן להתייחס ליחס בין שני משתנים אלו בתור סקלה כאשר איי סטאר קלאסי הוא המרכז בין דייקסטרה וחיפוש חמדן.

השמירה של "מצב" הקודקוד מבטיחה כי אף קודקוד לא ייבדק יותר מפעם אחת. מצב הקודקוד יכול להיות אחד משלושה – לא בדוק, פתוח, וסגור. לא בדוק הוא ברירת המחדל. מצב פתוח מתייחס לקודקודים שכבר הוספו לתור החזית, ונקבע ערכם לפחות פעם אחת. מצב סגור מתייחס לקודקודים ההוצאו מהתור והדרך אליהם היא בוודאות הטובה ביותר, ולכן ניתן לדלג על בדיקה חוזרת שלהם על ידי קביעת המצב בתור התנאי. בנוסף, אם מצב התא שמור אין צורך לברר האם הוא כבר נבדק, פעולה הלוקחת זמן לינארי O(N) בהתאם לגודל החזית בהשוואה ל O(1).

הדיוק והיעילות של איי סטאר הפכו אותו לאלגוריתם מאוד נפוץ. בעיקר במשחקים, בהם יש צורך למצוא מסלולים בזמן אמת, מהירות הביצוע מהווה גורם מכריע. עקב סיבות אלו בחרתי להשתמש באיי סטאר במשחק שבניתי.

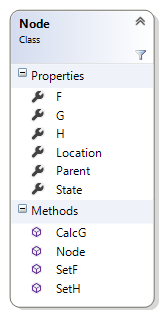
# מימוש איי סטאר

לפני שניגשים למימוש אלגוריתם יש צורך להבהיר מה הקלט והפלט הרצויים. כבר בוסס כי באלגוריתמי מציאת מסלולים הקלט הוא סוג מסוים של גרף. במשחק מדובר על טבלה בה אורכי הצלעות קבועים. בחרתי לייצג טבלה זו בתור מערך דו מימדי של בוליאנים, כאשר התנאי מסמל האם תא מסויים פנוי (ניתן לעבור דרכו). מספר התא במערך הדו מימדי זהה למיקומו בטבלה, בדומה ללו. הפלט יהיה הדרך הרצויה, מוצג בתור רשימה של תאים. מספרי התאים מוצגים בתור נקודות: טיפוס בו שני משתנים מסוג שלם המייצגים מיקום מסויים. על פי הנחות אלה בחרתי ליצור מימוש הנועד לעבוד עם טבלאות בלבד, אך עושה זאת באופן יעיל ככל הניתן. בפועל המימוש מחולק לשתי מחלקות – אחת המכילה את הגדרת הטיפוס תא Node והשנייה את האלגוריתם עצמו Pathfinder.



טבלה ממוספרת. במערכות הפעלה מודרניות ציר הגובה הפוך, ונקודת האפס נמצאת משמאל למעלה.

## טיפוס Node

דיאגרמה של מחלקת Node

### משתנים

public enum NodeState { Closed, Open, Untested }

public class Node

public Point Location { get; set; } //set by the Pathfinder class

public Node Parent { get; set; } //the node from which this one was reached

public int G { get; set; } //cost to reach node

public int H { get; set; } //estimated cost to reach finish (heuristic cost)

public int F { get; set; } //combined G + H or "path cost", the lower the better

public NodeState State { get; set; } = NodeState.Untested;

המשתנים השלמים הם ערך התנועה המשוקלל F אשר מחושב על ידי חיבור עלות התנועה לאותו התא G והמרחק המוערך ליעד H. שמור גם מיקומו של התא במערך בתור נקודה. אין חובה לשמור את המיקום, אך בחרתי לעשות זאת מטעמי נוחות מכיוון שאל משתנים ניתן לגשת בקלות. בנוסף, בכל תא יש הפניה אל התא הקודם לו במסלול, למרות שבתא ההתחלה ובתאים שלא נבדקו הפניה זו תהיה ריקה. המשתנה האחרון הוא מצב התא אשר שמור בתור enum (טיפוס המכיל ערכים קבועים), ובמקרה הזה אחד משלושה מצבים: סגור, פתוח, או לא בדוק. בזמן הגדרת תא מצבו הוא "לא בדוק" או Untested. כל המשתנים מוגדרים בתור properties על מנת לאפשר גישה נוחה.

### מתודות

public Node(Point location)

//initialize a new Node and sets location

{

Location = location;

}

public void SetH(Point goal)

//calculates heuristic by the manhattan approach

{

H = (Math.Abs(goal.X - Location.X) + Math.Abs(goal.Y - Location.Y)) \*10;

}

public void SetF()

//sets F to combined movement cost

{

F = G + H;

}

public static int CalcG(Directions dir, Node parent)

//calculates movement cost to current node

{

if((int)dir < 4)

return (parent.G + 10); //horizontal directions

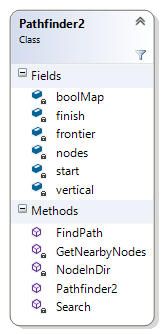
else

return (parent.G + 14); //14 = sqrt(200) rounded to save time

}

המתודות במחלקה הן חישוביות בלבד. המתודה האיתחולית קובעת את מיקום התא. SetH מחשבת את המרחק המוערך על פי גישת מנהטן כפי שהוצגה מקודם. ב SetF ניתן לקבוע את היחס בין עלות התנועה G והמרחק המוערך H, אך כרגע היחס הוא 1:1 על מנת למצוא את הדרך האופטימלית כמה שיותר מהר. המתודה הסטטית CalcG מחזירה את עלות התנועה עד לתא הנוכחי על ידי סיכום של G התא הקודם והוספת המרחק בין התאים, ולכן יש צורך לכלול את הכיוון בתור קלט. אם שני התאים אנכיים העלות המתווספת היא 10 ואם אלכסוניים 14. הסיבה שהשתמשתי ב 14 היא מכיוון שלא רציתי להשתמש במשתנה עשרוני, והערכת המרחק מספיקה בהחלט.

## מחלקת Pathfinder

דיאגרמה של מחלקת [[1]](#footnote-1)Pathfinder2

### משתנים

public enum Directions { Up, Right, Down, Left, UpRight, DownRight, DownLeft, UpLeft }

public class Pathfinder2

private bool[,] boolMap; //input map

private Point start;

private Point finish;

private Node[,] nodes; //2D array of nodes corresponding the input map

private SimplePriorityQueue<Node> frontier = new SimplePriorityQueue<Node>(); //the expanding frontier

private bool vertical; //enables vertical movement

boolMap, start ו finish הם הקלט של האלגוריתם, המהווים את הגרף לחקירה, נקודת התחלה ונקודת יעד. מערך התאים nodes מאותחל בתחילת החיפוש בהתאם למערך הבוליאנים שנקלט. SimplePriorityQueue הוא טיפוס מיובא של תור עדיפויות מכיוון שבשפת C# לא קיים טיפוס מובנה זה. frontier הוא תור החזית ממנו מוציאים תאים לבדיקה. ה enum Directions הוא לשימוש בזמן בדיקת תאים שכנים. vertical מכיל האם תנועה אלכסונית מאופשרת.

### מתודות

public Pathfinder2(bool[,] boolMap, Point start, Point finish, bool vertical)

//initialize a grid(graph) of nodes and resets them

{

this.boolMap = boolMap;

this.start = start;

this.finish = finish;this.vertical = vertical;

nodes = new Node[boolMap.GetLength(0), boolMap.GetLength(1)];

for (int x = 0; x < boolMap.GetLength(0); x++)

{

for (int y = 0; y < boolMap.GetLength(1); y++)

{

nodes[x, y] = new Node(new Point(x, y));

}

}

nodes[start.X, start.Y].SetH(finish);

nodes[start.X, start.Y].G = 0;

nodes[start.X, start.Y].SetF();

nodes[start.X, start.Y].State = NodeState.Open;

frontier.Enqueue(nodes[start.X, start.Y], nodes[start.X, start.Y].F);

}

מתודת אתחול המחלקה. קולטים את הטבלה, ההתחלה והסיום, בנוסף להאם תנועה אלכסונית מאופשרת. לכל מיקום בטבלה שנקלטה נוצר תא. נקודת ההתחלה נבדקת ומוכנסת לחזית. פעולת ההכנסה לתור עדיפויות Enqueue() מתבצעת על ידי ההזנה של האיבר בתור טיעון ראשון, והעדיפות שלו בתור הטיעון השני. באלגוריתם איי סטאר העדיפות היא עלות התנועה המשוכללת F, ובה משתמשים בתור.

public List<Point> FindPath()

//runs the search, if a path was found returns it

{

List<Point> path = new List<Point>();

if (Search())

//if a path was found, follow it from finish node backwards

{

Node curNode = this.nodes[finish.X, finish.Y];

while (curNode != null)

{

path.Add(curNode.Location);

curNode = curNode.Parent;

}

path.Reverse();

//since the returned path is flipped

}

return path;

}

FindPath() היא המתודה אליה קוראים על מנת להשתמש באלגוריתם. החיפוש Search() נקרא רק מכאן ומחזיר תשובה בוליאנית. אם נמצאה דרך, המתודה תשחזר אותה מהסוף על ידי מעקב אחרי ההורה מתא הסיום. לאחר מכן הדרך מהופכת מכיוון שבזמן המעקב היא מוחזרת מהסוף להתחלה. אם לא נמצאה דרך תוחזר רשימה ריקה.

private bool Search()

//main function

{

while (frontier.Count != 0)

//while there are nodes left to check

{

Node current = frontier.Dequeue();

//fetches the node with the best (lowest) movement cost which wasnt checked

current.State = NodeState.Closed;

//marks it as checked ("closed")

if (current.Location.Equals(this.finish))

//if end was reached, break

return true;

foreach (Node n in GetNearbyNodes(current))

//fetches adjecent nodes to the one currently tested

{

frontier.Enqueue(n, n.F);

//adds each to the frontier

}

}

return false;

}

הלולאה החיונית לחיפוש נמצאת ב Search(). כפי שהוסבר בחלק התיאורטי, הלולאה נמשכת כל עוד החזית לא ריקה. כל איטרציה מוצא הקודקוד בעל הערך הכי טוב, מסומן כסגור על מנת למנוע חזרות, ושכניו, אם עונים על הקרטריונים, מוספים לחזית. התאים מוספים בנפרד מכיוון שכל תא מוכנס לתור בהתאם לעדיפותו. במימוש שלי הקודקודים נבדקים במתודת ההבאה GetNearbyNodes() ולא בחיפוש הראשי. הסיבה היא שעל מנת להעריך את המרחק בין שני תאים צריך לדעת האם הם אנכיים או אלכסוניים, מידע אשר זמין בזמן ההבאה אך לא בהוספה. תנאי הסיום הם האם התא שנבדק הוא היעד, ואז החיפוש מחזיר תוצאה חיובית, או לאחר שהחזית מתרוקנת ואז מוחזרת תוצאה שלילית.

private List<Node> GetNearbyNodes(Node myNode)

//returns nodes adjacent to argument

{

List<Node> nodes = new List<Node>();

int dirs = 8;

if (!vertical)

//if vertical movement is disabled check only four directions

dirs = 4;

for (int i = 0; i < dirs; i++) //reads for each direction

{

Node \_node = NodeInDir(myNode, (Directions)i);

//fetches node in specified direction

if (\_node.State != NodeState.Closed && boolMap[\_node.Location.X, \_node.Location.Y])

//skips nodes that are closed or untraversable

{

if (\_node.State == NodeState.Open && \_node.G > Node.CalcG((Directions)i, myNode))

//if in open list, checks if the current path is better

{

\_node.Parent = myNode;

\_node.G = Node.CalcG((Directions)i, myNode);

nodes.Add(\_node);

}

if (\_node.State == NodeState.Untested)

//if untested, the node is initialized and added to the open list

{

\_node.Parent = myNode;

\_node.State = NodeState.Open;

\_node.G = Node.CalcG((Directions)i, myNode);

\_node.SetH(finish);

\_node.SetF();

nodes.Add(\_node);

}

}

}

return nodes;

}

מתודת הבאת השכנים. המתודה מחזירה רשימה של תאים קבילים אשר יוספו לחזית ב Search(). אם תנועה אלכסוניץ מאופשרת יבדקו שמונת הכיוונים, אחרת יבדקו רק ארבעת האנכיים. כל איטרציה מובא תא בכיוון מסויים NodeInDir(), ואם הוא עביר ולא במצב סגור נבדק לעומק. הבדיקה של המצב חשובה מכיוון שכאשר NodeInDir() מתבקש להחזיר תא מחוץ לגבולות הטבלה מוחזר התא הנוכחי. אם תא פתוח והדרך שנבדקת יותר טובה מהדרך השמורה היא מעודכנת ומוספת שוב פעם לחזית. במקרה הזה אכן אותו התא יןפיע פעמיים בתור, אך לאחר הפעם הראשונה התא ייחשב כסגור ולא ייבדק פעם נוספת. לבסוף, אם תא לא בדוק הוא מוסף לחזית גם כן.

private Node NodeInDir(Node myNode, Directions dir)

//fetches node in specified direction

{

switch (dir)

{

case Directions.Up:

if (myNode.Location.Y > 0)

return (nodes[myNode.Location.X, myNode.Location.Y - 1]);

break;

case Directions.Right:

if (myNode.Location.X < boolMap.GetLength(0) - 1)

return (nodes[myNode.Location.X + 1, myNode.Location.Y]);

break;

case Directions.Down:

if (myNode.Location.Y < boolMap.GetLength(1) - 1)

return (nodes[myNode.Location.X, myNode.Location.Y + 1]);

break;

case Directions.Left:

if (myNode.Location.X > 0)

return (nodes[myNode.Location.X - 1, myNode.Location.Y]);

break;

case Directions.UpRight:

if (myNode.Location.Y > 0 &&

myNode.Location.X < boolMap.GetLength(0) - 1)

return (nodes[myNode.Location.X, myNode.Location.Y - 1]);

break;

case Directions.DownRight:

if (myNode.Location.Y < boolMap.GetLength(1) - 1 &&

myNode.Location.X < boolMap.GetLength(0) - 1)

return (nodes[myNode.Location.X + 1, myNode.Location.Y]);

break;

case Directions.DownLeft:

if (myNode.Location.Y < boolMap.GetLength(1) - 1 &&

myNode.Location.X > 0)

return (nodes[myNode.Location.X, myNode.Location.Y + 1]);

break;

case Directions.UpLeft:

if (myNode.Location.Y > 0 &&

myNode.Location.X > 0)

return (nodes[myNode.Location.X - 1, myNode.Location.Y]);

break;

default:

return myNode;

}

return myNode;

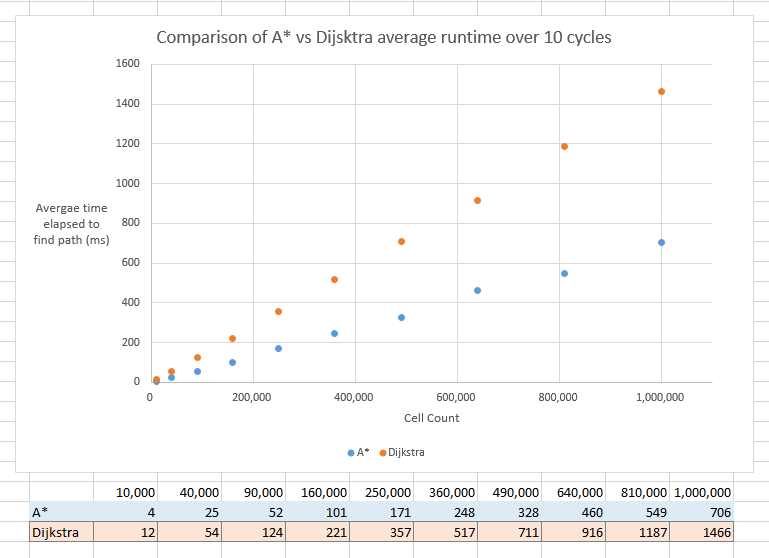
}

המתודה מחזירה את התא השכן בכיוון המסויים של התא הנבדק, לא לפני שהיא בודקת שהוא נמצא בגבולות הטבלה. אם התא לא בטבלה NodeInDir() מחזיר את התא המקורי.

### אופן שימוש

השימוש במחלקה נועד להיות פשוט. תחילה יוצרים אוביקט מטיפוס Pathfinder2 עם קלט מתאים המכיל טבלה, נקודת התחלה ונקודת סיום, וקובעים האם לאפשר תנועה אלכסונית. לאחר מכן קוראים למתודה FindPath(), והיא תריץ את החיפוש Search(). אם נמצאה דרך היא תוחזר בתור רשימה של תאים המהווים את הדרך ליעד. אחרת הרשימה שתוחזר תהיה ריקה.

## מחשבות נוספות

מבחינתי מימוש זה של אלגוריתם היה A\* הצלחה. הוא ידידותי למשתמש, רץ בזמן טוב וקל להבנה. כנראה שאפשר לבצע ייעולים קטנים נוספים, בעיקר בעת אתחול האלגוריתם ובשימוש חוזר, אך המחלקות מהוות בסיס טוב לשימוש עתידי. על מנת להשוות את זמן הריצה של A\* לאחד של דייקסטרה הרצתי מספר בדיקות עם איזונים שונים של המרחק המשוכלל F כאשר ב A\* הם שווים ובדייקסטרה אין התייחסות כלל למשתנה ההיוריסטי.

# מימוש המשחק

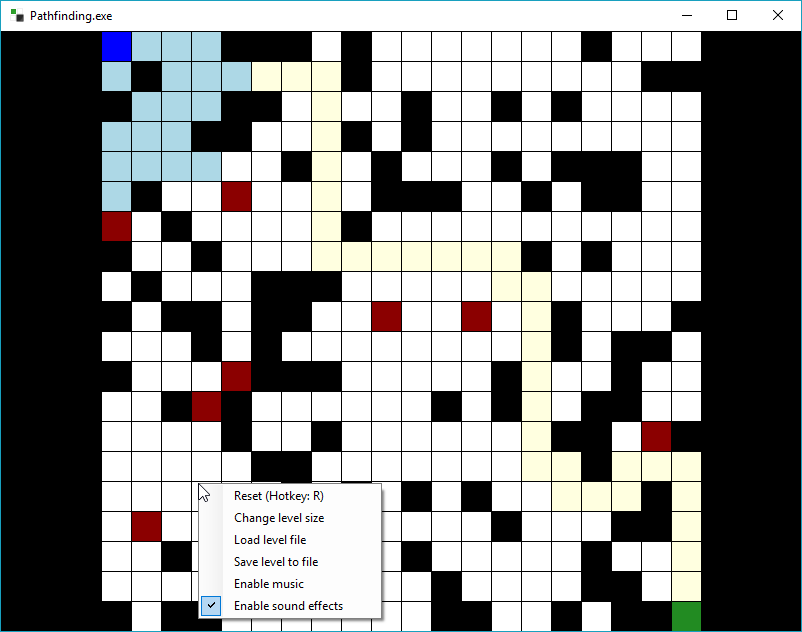
## הגדרות

Pathfinding.exe הוא שם המשחק שבניתי. מטרתו היא להדגים את עבודתי על מימוש אלגוריתם החיפוש איי סטאר, ועל סמך זה נגזר האופן בו המשחק עובד. חוקי המשחק מבוססים על מודלים קיימים, אך באופן שרירותי בלבד. משחקים אשר היוו חזון רעיוני כוללים את Monaco, Invisible, ו Stealth inc.

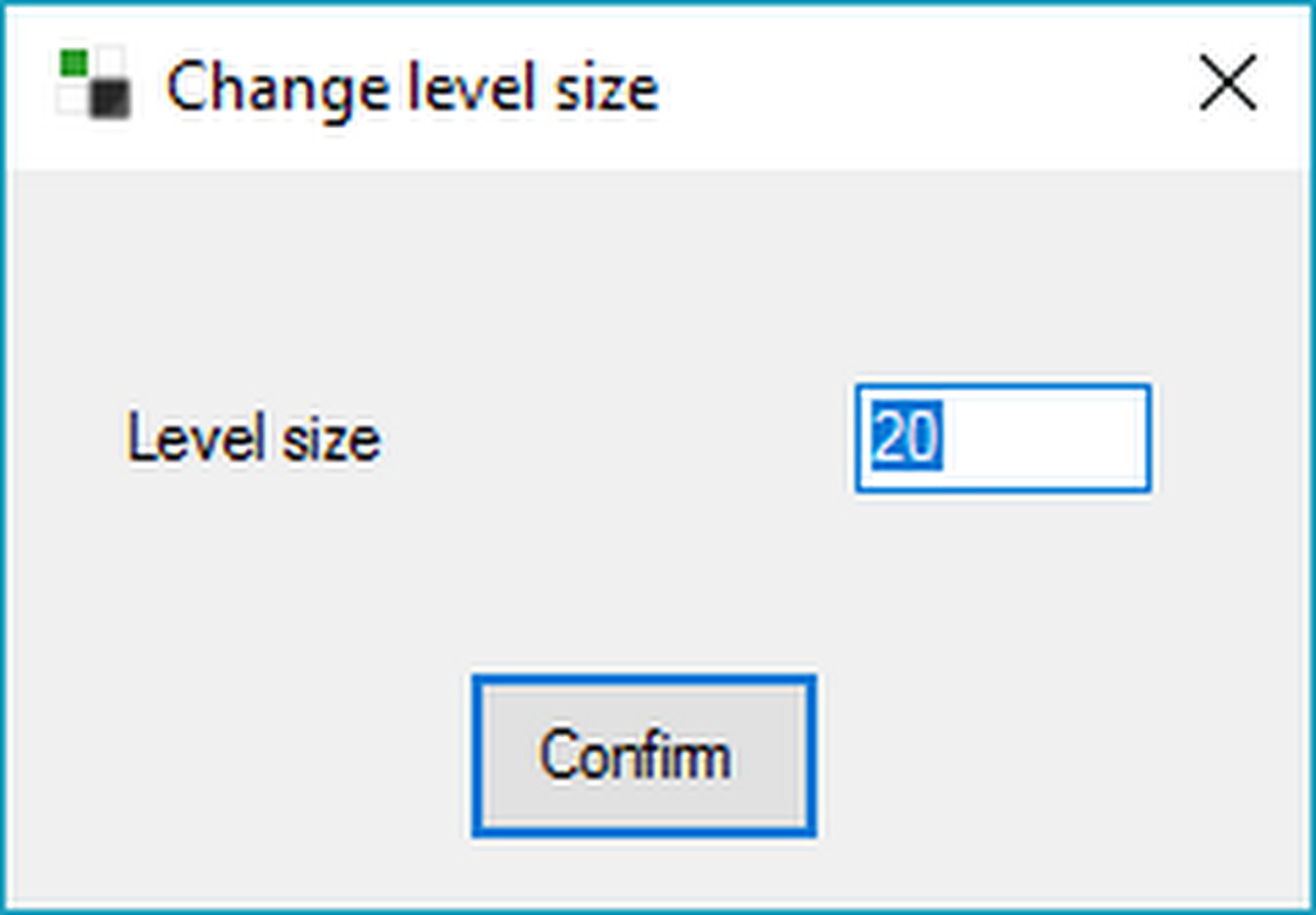
המשחק הוא דו מימדי ממבט על בו השחקן מנווט דרך שלב, מתחמק מאויבים המנסים לסגור עליו, ומנסה להגיע למשבצת המטרה. לוח המשחק או המפה הוא טבלה ריבועית בה התאים יכולים להיות עבירים או חסומים. קיים שחקן אחד, ומולו מספר אויבים הזזים לאחר כל מהלך של המשתמש. המשחק נגמר כשהשחקן מגיע לנקודת המטרה המוגדרת בשלב, במקרה הנקרא נצחון, או כאשר האויבים משיגים את השחקן, הנקרא הפסד. השחקן יזוז בהתאם לקלט המשתמש בכיווני החצים, אך לא יוכל לחצות משבצות חסומות. האויבים יתחילו לרדוף אחרי השחקן כאשר הוא במרחק מסויים מהם, הנקרא טווח. הרדיפה נעשית על ידי תזוזה בדרך הקצרה ביותר לכיוון השחקן כפי שהיא נקבעת על ידי אלגוריתם מציאת המסלולים.

על פי ברירת המחדל השלבים נוצרים אקראית, אך ניתן לשמור שלבים לקובץ טקסט ולטעון אותם. ניתן גם לערוך קבצים אלו ידנית. ניתן תיאורטית לשחק בכל גודל של לוח, אך קיימות הגבלות מטעמים מעשיים כגון זיכרון או גודל החלון.

אציג את המימוש של המשחק בפרקים: לוגיקה, קלט, וגרפיקה ושמע. אסביר כיצד הם עובדים ובאיזה אופן הם מתקשרים למחלקה הראשית.

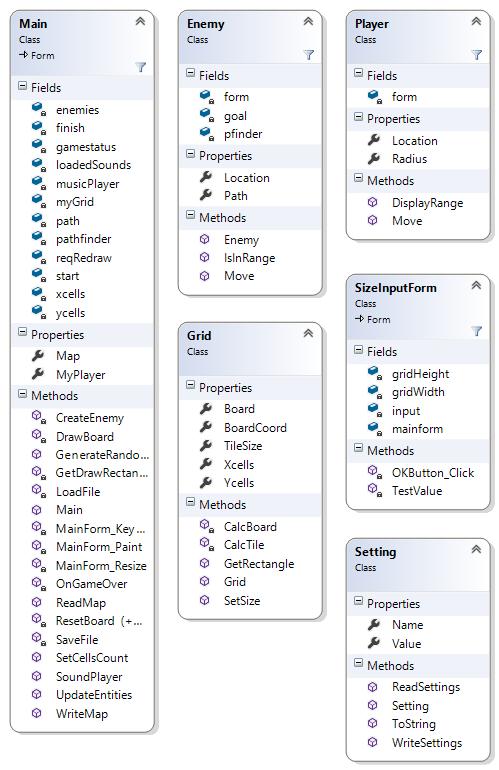


חלון המשחק (Main) עם תפריט האפשרויות פתוח



הטופס (SizeInputForm) שנפתח כאשר לוחצים על Change level size

## דיאגרמת המחלקות



## מחלקות עזר

**Player:** טיפוס מסוג שחקן. מסוגל לזוז בכיוון שננקב. בנוסף מסוגל להחזיר את כל התאים הנמצאים בטווח שלו.

**Enemy:** טיפוס מסוג אויב. מסוגל לקבוע האם השחקן בטווח האיתור, לקבוע מסלול אל השחקן, ולהתקדם במסלול צעד אחד כל מהלך.

**SizeInputForm:** טופס באמצעותו השחקן מסוגל לקבוע את גודל לוח המשחק.

**Setting:** טיפוס המיועד לאפשר שמירה גנרית של משתנים כקובץ טקסט, כאשר שם ההגדרה וערכה נשמרים בתור שני טיפוסי string. המתודה WriteSettings() שומרת רשימה של טיפוסים מסוג Setting בקובץ טקסט, ו ReadSettings() מחזירה רשימה זו על ידי קריאת קובץ מסוים.

**Grid:** מחלקה הממפה את הלוח לחלון המשחק. קובעת את הקורדינציה והגודל של כל תא בהתאם לגודל החלון ומספר התאים בלוח, ומסוגלת להחזיר תא מבוקש בתור מלבן (משתנה המכיל מיקום וגודל).

## טופס Main

המחלקה הראשית, שבהתאם לשמה מכילה את מירב הקוד. המתודות בה מתחלקות לשני סוגים. הסוג הראשון הוא מתודות ממשק. מתודות אלו הכרחיות על מנת לטפל באירועים אשר נגרמים דרך חלון המשחק, הידוע בתור ממשק המשתמש (User Interface). אירועים אלו כוללים טעינה או שינוי גודל החלון, לחיצה על מקשי המקלדת, או לחיצה על כפתור בתפריט. יוצא הדופן הוא אירוע הציור, הנורה על ידי התוכנית עצמה. מתודות אלה, על פי רוב, מכילות הפניה לסוג השני של מתודות – המתודות החישוביות.

המתודות החישוביות הן אלו שמקדמות את המשחק. זה כולל איפוס הלוח, עדכון הלוח או עדכון היישויות בו, שמירת הלוח הקיים לקובץ וקריאתו, ועוד. אסביר בקצרה איך המשחק בנוי, וכיצד המתודות החישוביות משפיעות עליו.

לוח המשחק שמור בתור מספר משתנים: כמות התאים (שלמים), המפה (מערך בוליאני דו מימדי), נקודות ההתחלה והסיום, הטיפוסים מסוג שחקן, ורשימת אויבים. משתנים אלו מגדירים שלב, ונשמרים או נקראים בעת טעינת קובץ. כאשר נפתח המשחק, ולאחר מכן כל אתחול, מורצת המתודה ResetBoard(). המתודה קובעת את נקודות ההתחלה והסיום, ומנפיקה מפה בעלי מימדים זהים לכמות התאים עד שקיימת דרך בין נקודת ההתחלה לסיום. בדיקה זו נעשית בעזרת אלגוריתם מציאת המסלולים. לאחר מכן מאותחל הטיפוס Grid כך שיכיל את קורדינציות הלוח בהתאם לגודל החלון, מאותחל טיפוס השחקן, ומוספים מספר אויבים בהתאם לגודל הלוח.

עדכון המשחק נעשה על ידי UpdateEntities(), והוא מופעל כל פעם שהמשתמש לוחץ על אחד מארבעת מקשי החצים. אם מיקום השחקן זהה לנקודת הסיום מופיעה הודעת הניצחון. לאחר שהשחקן זז (או מנסה לזוז, במקרה שמנסה לעבור למשבצת חסומה), נבדק כל אויב. במקרה שהאויב בטווח השחקן, האויב יתקדם בדרך הנקבעה בעזרת אלגוריתם מציאת המסלולים. אם מיקומו של האויב זהה לזה של השחקן מופיעה הודעת סיום המשחק.

# סיכום

בחלק התיאורטי של העבודה סקרתי אלגוריתמי מציאת מסלולים שונים. הסברתי כיצד כל אחד עובד, השוויתי ביניהם ונתתי דוגמאות למקרים בהם כדאי להשתמש באחד על פני האחר. את האלגוריתם A\*, המתאים ביותר למשחק שלי עקב יעילותו, בחרתי לממש בשפת C#.

מימוש האלגוריתם וקוד המשחק מהווים את החלק המעשי. הסברתי למה בניתי את מימוש האלגוריתם באופן מסויים, מה התפקיד של כל קטע קוד, וכיצד משתמשים במחלקה. בנוסף הצגתי את המשחק Pathfinding.exe ואת המחלקות המרכיבות אותו.

המשחק עצמו הוא תוצר של הידע שרכשתי במהלך הפרויקט. ידע זה כולל עבודה עם שפה וסביבת עבודה חדשים, תכנון מראש של מתודות וחלוקה למחלקות, מחקר של אלגוריתמי מציאת מסלולים קיימים, ועוד. כל הכלים האלו נדרשו על מנת לממש את הרעיון המקורי איתו ניגשתי לעבודה לפני שנה, ואני חושבת שהמשחק משקף זאת.

# ביבליגרפיה

* קלר, ארז (2003) תכנות מונחה עצמים בשפת C#
* Dijkstra, E. W. (1959). "A note on two problems in connexion with graphs".
* Zeng, W.; Church, R. L. (2009). "Finding shortest paths on real road networks: the case for A\*".
* http://www.policyalmanac.org/games/aStarTutorial.htm
* http://blog.two-cats.com/2014/06/a-star-example
* http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html
* https://en.wikipedia.org/
* https://github.com/BlueRaja/High-Speed-Priority-Queue-for-C-Sharp
* https://qiao.github.io/PathFinding.js/visual
* https://msdn.microsoft.com/en-us/library
* http://stackoverflow.com

# נספחים

## מבוא לתכנות מונחה עצמים ב C#

תכנות מונחה עצמים הוא שם של פרדיגמת תכנות המשתמשת ב"עצמים" לשם תכנות. השיטה נועדה להיות אינטואיטיבית וקרובה יותר לדרך החשיבה של האדם. אתגר התכנות, אשר בעבר היה כיצד לכתוב את הלוגיקה, הוא עכשיו כיצד להגדיר את המידע.

עצמים מוגדרים עלי ידי טיפוסים. **טיפוס** (גם נקרא מחלקה) הוא סוג העצם, ועצם הוא מופע של טיפוס איתו ניתן לעבוד. כל טיפוס יכול לכלול סוגים שונים של איברים. מאפיינים מתארים את הנתונים הנשמרים בטיפוס, מתודות אשר מגדירות פעולות אשר ניתן לבצע, ואירועים המאפשרים תקשור בין מחלקות או עצמים שונים.

כאשר מתייחסים לתכנות מונחה עצמים מועלים שלושה עקרונות. **כימוס** אומר שקבוצה של מאפיינים, מתודות ואיברים אחרים מיוחסת כאיבר אחד – עצם. הגישה אל איברים אלו נעשית רק דרך הפונקציות של האובייקט המהוות את הממשק. בכך שכימוס לא מאפשר שינוי ישיר של משתני האובייקט ניתן למנוע שינויים לא מבוקרים.

**הורשה** מאפשרת להגדיר מחלקה חדשה אשר משתמשת, מרחיבה ומשנה את ההתנהגות המוגדרת במחלקה אחרת. המחלקה ממנה נורשים האיברים נקראת מחלקת הבסיס, והמחלקה היורשת איברים אלו נקראת מחלקה נגזרת. ב C# אין תמיכה בהורשה מרובה, משמע לכל מחלקה נגזרת יכול להיות רק מחלקת בסיס אחת. יכולת זו מאפשרת חיסכון בכתיבת קוד ושימוש חוזר בקוד קיים, מכיוון שאין צורך להגדיר טיפוס חדש כל פעם.

עקרון **הפולימורפיזם**, או בעברית רב צורתיות, הוא היכולת לתת לפונקציה במחלקת הבסיס מימוש שונה במחלקות הנגזרות כך שבזמן ריצה תופעל הפונקציה שמתאימה לטיפוס האובייקט. העיקרון מאפשר להתייחס לאובייקטים מסוגים שונים באופן זהה אם יש להם מחלקת בסיס משותפת. אפשרות זו יוצרת תוכניות נוחות לעבודה מכיוון שניתן להגדיר פונקציות שעל כל מחלקה להכיל, ולהפעיל פונקציות אלו על אובייקט מבלי לדעת מהו הסוג שלו.

C# הינה שפת תכנות המותאמת לעבודה עם תכנות מונחה עצמים. שימוש במאפיינים אשר אפשר באמצעותם לגשת למשתנים באופן מבוקר מחזקת את עיקרון הכימוס. קיימת תמיכה במתודות וירטואליות, אשר בקריאתם מזומנת המתודה במחלקה הנגזרת. יש גם טיפוסי מידע struct וגם טיפוסי התייחסות class. תכונות אלו, בנוסף לעוד רבות, משקפות בשפה את עקרנות התכנות מונחה עצמים.

## קוד מחלקת Node

using System;

using System.Drawing;

namespace aStar

{

public enum NodeState { Closed, Open, Untested } // closed = does not require further check, open = added but a better route may be possible, untested... yeah

public class Node

{

public Point Location { get; set; } //set by the Pathfinder class

public Node Parent { get; set; } //the node from which this one was reached

public int G { get; set; } //cost to reach node

public int H { get; set; } //estimated cost to reach finish (heuristic cost)

public int F { get; set; } //combined G + H or "path cost", the lower the better

public NodeState State { get; set; } = NodeState.Untested;

public Node(Point location) //initialize a new Node and sets location

{

Location = location;

}

public void SetH(Point goal) //calculates heuristic by the manhattan approach

{

H = (Math.Abs(goal.X - Location.X) + Math.Abs(goal.Y - Location.Y)) \* 10;

}

public void SetF() //sets F to combined movement cost

{

F = G + H;

}

public static int CalcG(Directions dir, Node parent) //calculates movement cost to current node

{

if ((int)dir < 4) //horizontal directions

return (parent.G + 10);

else

return (parent.G + 14); //14 = sqrt(200) rounded to save process time

}

}

}

## קוד מחלקת Pathfinder2

using System.Collections.Generic;

using System.Drawing;

using Priority\_Queue; //NuGet by BlueRaja which implements priority queue in c# https://github.com/BlueRaja/High-Speed-Priority-Queue-for-C-Sharp

namespace aStar

{

public enum Directions { Up, Right, Down, Left, UpRight, DownRight, DownLeft, UpLeft }

public class Pathfinder2

{

private bool[,] boolMap; //input map

private Point start;

private Point finish;

private Node[,] nodes; //2D array of nodes corresponding the input map

private SimplePriorityQueue<Node> frontier = new SimplePriorityQueue<Node>(); //the expanding frontier

private bool vertical; //enables vertical movement

public Pathfinder2(bool[,] boolMap, Point start, Point finish, bool vertical) //initialize a grid(graph) of nodes and resets them

{

this.boolMap = boolMap;

this.start = start;

this.finish = finish;

this.vertical = vertical;

nodes = new Node[boolMap.GetLength(0), boolMap.GetLength(1)];

for (int x = 0; x < boolMap.GetLength(0); x++)

{

for (int y = 0; y < boolMap.GetLength(1); y++)

{

nodes[x, y] = new Node(new Point(x, y));

}

}

nodes[start.X, start.Y].SetH(finish);

nodes[start.X, start.Y].G = 0;

nodes[start.X, start.Y].SetF();

nodes[start.X, start.Y].State = NodeState.Open;

frontier.Enqueue(nodes[start.X, start.Y], nodes[start.X, start.Y].F);

}

public List<Point> FindPath() //initializes the search, if a path was found returns it

{

List<Point> path = new List<Point>();

if (Search()) //if a path was found, follow it from finish node backwards

{

Node curNode = this.nodes[finish.X, finish.Y];

while (curNode != null)

{

path.Add(curNode.Location);

curNode = curNode.Parent;

}

path.Reverse(); //since the returned path is flipped

}

return path;

}

private bool Search() //main function

{

while (frontier.Count != 0) //while there are nodes left to check

{

Node current = frontier.Dequeue(); //fetches the node with the best (lowest) movement cost which wasnt checked

current.State = NodeState.Closed; //marks it as checked ("closed")

if (current.Location.Equals(this.finish)) //if end was reached, break

return true;

foreach (Node n in GetNearbyNodes(current)) //fetches adjecent nodes to the one currently tested

{

frontier.Enqueue(n, n.F); //adds each to the frontier

}

}

return false;

}

private List<Node> GetNearbyNodes(Node myNode) //returns nodes adjacent to argument

{

List<Node> nodes = new List<Node>();

int dirs = 8;

if (!vertical) //if vertical movement is disabled check only four directions

dirs = 4;

for (int i = 0; i < dirs; i++) //reads for each direction

{

Node \_node = NodeInDir(myNode, (Directions)i); //fetches node in specified direction

if (\_node.State != NodeState.Closed && boolMap[\_node.Location.X, \_node.Location.Y]) //skips nodes that are closed or untraversable

{

if (\_node.State == NodeState.Open && \_node.G > Node.CalcG((Directions)i, myNode)) //if in open list, checks if the current path is better

{

\_node.Parent = myNode;

\_node.G = Node.CalcG((Directions)i, myNode);

nodes.Add(\_node);

}

if (\_node.State == NodeState.Untested) //if untested, the node is initialized and added to the open list

{

\_node.Parent = myNode;

\_node.State = NodeState.Open;

\_node.G = Node.CalcG((Directions)i, myNode);

\_node.SetH(finish);

\_node.SetF();

nodes.Add(\_node);

}

}

}

return nodes;

}

private Node NodeInDir(Node myNode, Directions dir) //fetches node in specified direction

{

switch (dir)

{

case Directions.Up:

if (myNode.Location.Y > 0)

return (nodes[myNode.Location.X, myNode.Location.Y - 1]);

break;

case Directions.Right:

if (myNode.Location.X < boolMap.GetLength(0) - 1)

return (nodes[myNode.Location.X + 1, myNode.Location.Y]);

break;

case Directions.Down:

if (myNode.Location.Y < boolMap.GetLength(1) - 1)

return (nodes[myNode.Location.X, myNode.Location.Y + 1]);

break;

case Directions.Left:

if (myNode.Location.X > 0)

return (nodes[myNode.Location.X - 1, myNode.Location.Y]);

break;

case Directions.UpRight:

if (myNode.Location.Y > 0 && myNode.Location.X < boolMap.GetLength(0) - 1)

return (nodes[myNode.Location.X, myNode.Location.Y - 1]);

break;

case Directions.DownRight:

if (myNode.Location.Y < boolMap.GetLength(1) - 1 && myNode.Location.X < boolMap.GetLength(0) - 1)

return (nodes[myNode.Location.X + 1, myNode.Location.Y]);

break;

case Directions.DownLeft:

if (myNode.Location.Y < boolMap.GetLength(1) - 1 && myNode.Location.X > 0)

return (nodes[myNode.Location.X, myNode.Location.Y + 1]);

break;

case Directions.UpLeft:

if (myNode.Location.Y > 0 && myNode.Location.X > 0)

return (nodes[myNode.Location.X - 1, myNode.Location.Y]);

break;

default:

return myNode;

}

return myNode;

}

}

}

## טופס Main

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Windows.Forms;

using System.IO;

using System.Media;

using aStar;

namespace Pathfinding

{

public enum Sounds { Collision, Move, GameOver, Win } //sound effects list

public partial class Main : Form

{

private int gamestatus = 0; //is the splash screen open

private Random rng = new Random();

private int xcells = 20; //initial board dimensions

private int ycells = 20;

public bool[,] Map { get; set; }

private Point start; //player starting position

private Point finish; //player goal

private bool reqRedraw = true; //toggles drawing the board

private Grid myGrid; //board grid

private Pathfinder2 pathfinder;

private List<Point> path = new List<Point>();

public Player MyPlayer { get; set; }

private List<Enemy> enemies = new List<Enemy>();

private static List<SoundPlayer> loadedSounds = new List<SoundPlayer>();

private static System.Windows.Media.MediaPlayer musicPlayer = new System.Windows.Media.MediaPlayer();

public Main()

{

InitializeComponent();

ResizeRedraw = true;

DoubleBuffered = true;

}

//static calculation functions

public static int DisPwr(Point p1, Point p2) //distance power

{

return ((p1.X - p2.X) \* (p1.X - p2.X) + (p1.Y - p2.Y) \* (p1.Y - p2.Y));

}

public static bool[,] GenerateRandomMap(int width, int height) //generates random "map" in the specified dimensions

{

bool[,] grid = new bool[width, height];

Random rng = new Random();

for (int i = 0; i < width; i++)

{

for (int j = 0; j < height; j++)

{

grid[i, j] = (rng.Next(4) % 4 != 0); //4 = the portion of the map blocked, calculated 1/density(4)

}

}

grid[0, 0] = true; //start &finish are always unblocked

grid[grid.GetLength(0) - 1, grid.GetLength(1) - 1] = true;

return grid;

}

//save and load functions

private void LoadFile() //loads a text file details a game board

{

OpenFileDialog dlg = new OpenFileDialog();

dlg.DefaultExt = ".txt";

dlg.Filter = "Text documents (.txt)|\*.txt"; // Filter files by extension

dlg.ShowDialog();

string file = dlg.FileName;

if (file != "")

{

try

{

List<Setting> settings = Setting.ReadSettings(file);

Point \_player = new Point(0, 0);

List<Point> \_enemies = new List<Point>();

foreach (Setting s in settings)

{

switch (s.Name)

{

case "player":

\_player.X = Convert.ToInt32(s.Value.Split(',')[0]);

\_player.Y = Convert.ToInt32(s.Value.Split(',')[1]);

break;

case "enemy":

Point \_enemy = new Point(0, 0);

\_enemy.X = Convert.ToInt32(s.Value.Split(',')[0]);

\_enemy.Y = Convert.ToInt32(s.Value.Split(',')[1]);

\_enemies.Add(\_enemy);

break;

case "xcells":

int \_xcells = Convert.ToInt32(s.Value);

if (\_xcells <= 300 && \_xcells >= 6)

this.xcells = \_xcells;

break;

case "ycells":

int \_ycells = Convert.ToInt32(s.Value);

if (\_ycells <= 300 && \_ycells >= 6)

this.ycells = \_ycells;

break;

case "finish":

this.finish.X = Convert.ToInt32(s.Value.Split(',')[0]);

this.finish.Y = Convert.ToInt32(s.Value.Split(',')[1]);

break;

default:

break;

}

}

bool[,] \_map = ReadMap(file, xcells, ycells);

ResetBoard(\_map, \_player, \_enemies);

}

catch

{

MessageBox.Show("Invalid map file.", "Error", MessageBoxButtons.OK);

}

}

}

private void SaveFile() //saves current board to file

{

SaveFileDialog dlg = new SaveFileDialog();

dlg.FileName = "myLevel";

dlg.DefaultExt = ".txt";

dlg.Filter = "Text documents (.txt)|\*.txt"; // Filter files by extension

dlg.ShowDialog();

string file = dlg.FileName;

if (file != "")

{

List<Setting> \_s = new List<Setting>();

\_s.Add(new Setting("xcells", this.xcells.ToString()));

\_s.Add(new Setting("ycells", this.ycells.ToString()));

\_s.Add(new Setting("player", MyPlayer.Location.X + "," + MyPlayer.Location.Y));

\_s.Add(new Setting("finish", finish.X + "," + finish.Y));

foreach (Enemy e in enemies)

{

\_s.Add(new Setting("enemy", e.Location.X + "," + e.Location.Y));

}

Setting.WriteSettings(\_s, file, "//This is a level file generated by Pathfinding.exe.\r\n");

WriteMap(file, this.Map);

}

}

public static bool[,] ReadMap(String filename, int width, int height) //reads a character map from file

{

bool[,] myMap = new bool[width, height];

using (StreamReader reader = new StreamReader(filename))

{

string line = "";

while (!reader.EndOfStream)

{

if (line != "" && line[0] == '\*') //skips input until it reaches the map "drawing" character

break;

line = reader.ReadLine();

}

for (int y = 0; y < height; y++)

{

line = reader.ReadLine();

for (int x = 0; x < width; x++)

{

if (x < line.Length && line[x].Equals('O'))

myMap[x, y] = true;

else

myMap[x, y] = false;

}

}

}

return myMap;

}

public static void WriteMap(string filename, bool[,] myMap) //appends the boolean map to the end of the save file

{

List<string> map = new List<string>();

for (int x = 0; x < myMap.GetLength(0); x++)

{

string column = "";

for (int y = 0; y < myMap.GetLength(1); y++)

{

if (myMap[y, x])

column += 'O';

else

column += 'M';

}

map.Add(column);

}

using (StreamWriter writer = new StreamWriter(filename, true))

{

writer.WriteLine("\*");

foreach (string s in map)

{

writer.WriteLine(s);

}

}

}

//sound related

public void SoundPlayer(Sounds s) //plays sounds

{

if (enableSoundEffectsToolStripMenuItem.Checked)

{

switch (s)

{

case Sounds.Collision:

loadedSounds[0].Play();

break;

case Sounds.Move:

loadedSounds[1].Play();

break;

case Sounds.GameOver:

loadedSounds[2].Play();

break;

case Sounds.Win:

loadedSounds[3].Play();

break;

default:

break;

}

}

}

private List<SoundPlayer> LoadSounds() //loads all sound files

{

musicPlayer.Open(new Uri(Application.StartupPath + "/Ttrs\_-\_BgMusic.wav"));

musicPlayer.MediaEnded += MediaEndedEvent;

List<SoundPlayer> loaded = new List<SoundPlayer>();

loaded.Add(new SoundPlayer(Properties.Resources.Ttrs\_\_\_Land));

loaded.Add(new SoundPlayer(Properties.Resources.Ttrs\_\_\_Move));

loaded.Add(new SoundPlayer(Properties.Resources.Ttrs\_\_\_GameOver));

loaded.Add(new SoundPlayer(Properties.Resources.Ttrs\_\_\_Win));

return loaded;

}

private void MediaEndedEvent(object sender, EventArgs e) //replays the background music

{

musicPlayer.Position = TimeSpan.Zero;

musicPlayer.Play();

}

//graphics

private Rectangle GetDrawRectangle() //the area in which to draw

{

Rectangle rect = new Rectangle();

rect.Size = new Size(this.ClientSize.Height, this.ClientSize.Height);

rect.Location = new Point((this.ClientSize.Width - this.ClientSize.Height) / 2, 0);

return rect;

}

private void DrawBoard(object sender, PaintEventArgs e) //paint event

{

Graphics g = e.Graphics;

if (gamestatus == 0)

g.DrawImage(Properties.Resources.pathfindingexe, GetDrawRectangle());

else

{

Pen myPen = new Pen(Color.Black, 1);

SolidBrush blackBrush = new SolidBrush(Color.Black);

SolidBrush whiteBrush = new SolidBrush(Color.White);

SolidBrush blueBrush = new SolidBrush(Color.Blue);

SolidBrush forestGreenBrush = new SolidBrush(Color.ForestGreen);

SolidBrush redBrush = new SolidBrush(Color.Red);

SolidBrush darkRedBrush = new SolidBrush(Color.DarkRed);

SolidBrush yellowBrush = new SolidBrush(Color.LightYellow);

SolidBrush lightBlueBrush = new SolidBrush(Color.LightBlue);

g.Clear(Color.Black);

if (reqRedraw)

{

for (int x = 0; x < Map.GetLength(0); x++)

{

for (int y = 0; y < Map.GetLength(1); y++)

{

if (Map[x, y])

{

g.FillRectangle(whiteBrush, myGrid.GetRectangle(x, y));

g.DrawRectangle(myPen, myGrid.GetRectangle(x, y));

}

else

{

g.FillRectangle(blackBrush, myGrid.GetRectangle(x, y));

}

}

}

}

foreach (Point p in path)

{

g.FillRectangle(yellowBrush, myGrid.GetRectangle(p.X, p.Y));

g.DrawRectangle(myPen, myGrid.GetRectangle(p.X, p.Y));

}

if (reqRedraw)

{

List<Point> points = MyPlayer.DisplayRange();

foreach (Point \_p in points)

{

if (Map[\_p.X, \_p.Y])

{

g.FillRectangle(lightBlueBrush, myGrid.GetRectangle(\_p.X, \_p.Y));

g.DrawRectangle(myPen, myGrid.GetRectangle(\_p.X, \_p.Y));

}

}

g.FillRectangle(blueBrush, myGrid.GetRectangle(MyPlayer.Location.X, MyPlayer.Location.Y));

g.DrawRectangle(myPen, myGrid.GetRectangle(MyPlayer.Location.X, MyPlayer.Location.Y));

}

g.FillRectangle(forestGreenBrush, myGrid.GetRectangle(finish.X, finish.Y));

g.DrawRectangle(myPen, myGrid.GetRectangle(finish.X, finish.Y));

foreach (Enemy enemy in enemies)

{

if (enemy.Active)

g.FillRectangle(redBrush, myGrid.GetRectangle(enemy.Location.X, enemy.Location.Y));

else

g.FillRectangle(darkRedBrush, myGrid.GetRectangle(enemy.Location.X, enemy.Location.Y));

g.DrawRectangle(myPen, myGrid.GetRectangle(enemy.Location.X, enemy.Location.Y));

}

}

}

private void OnGameOver() //message displayed on game over

{

DialogResult result = MessageBox.Show("Restart?", "Game over", MessageBoxButtons.YesNo);

switch (result)

{

case DialogResult.Yes:

ResetBoard();

break;

case DialogResult.No:

this.Close();

break;

default:

break;

}

}

//configuration

public void SetCellsCount(int x, int y) //the grid resize function, normally triggered by filling the pop up form

{

xcells = x;

ycells = y;

ResetBoard();

}

private Point CreateRandomEnemy()

{

Point \_location = new Point(0, 0);

while ((DisPwr(\_location, MyPlayer.Location) <= (MyPlayer.Radius \* MyPlayer.Radius)) || //makes sure enemies dont spawn too close to the player

(DisPwr(\_location, new Point(xcells - 1, ycells - 1))) <= (MyPlayer.Radius \* MyPlayer.Radius)) //or the finish point

{

\_location = new Point(rng.Next(xcells), rng.Next(ycells));

}

enemies.Add(new Enemy(\_location, this));

return (\_location);

}

private Point CreateEnemy(Point \_location)

{

enemies.Add(new Enemy(\_location, this));

return (\_location);

}

//game update

public void UpdateEntities() //moves all enemies each time the player moves

{

if (MyPlayer.Location.Equals(finish)) //winning message

{

this.Invalidate();

SoundPlayer(Sounds.Win);

MessageBox.Show("You win!", "Game over", MessageBoxButtons.OK);

OnGameOver();

}

foreach (Enemy e in enemies)

{

if (e.Location.Equals(MyPlayer.Location)) //checks to see if the player moved on an enemy

{

reqRedraw = false;

this.Invalidate();

SoundPlayer(Sounds.GameOver);

OnGameOver();

break;

}

if (e.IsInRange())

e.Detected(MyPlayer.Location); //if enemy is in the player's range, return enemy location

e.Move(); //moves

if (e.Location.Equals(MyPlayer.Location)) //checks to see if the enemy moved on the player

{

reqRedraw = false;

this.Invalidate();

SoundPlayer(Sounds.GameOver);

OnGameOver();

break;

}

this.Invalidate(); //forces redraw

}

}

private void ResetBoard() //generates a new map, pathfinder and grid

{

start = new Point(0, 0);

finish = new Point(xcells - 1, ycells - 1);

path.Clear();

while (path.Count == 0) //to prevent a blocked grid from generating

{

Map = GenerateRandomMap(xcells, ycells);

pathfinder = new Pathfinder2(Map, start, finish, false);

path = pathfinder.FindPath();

}

myGrid = new Grid(GetDrawRectangle(), Map.GetLength(0), Map.GetLength(1));

reqRedraw = true;

MyPlayer = new Player(start, this);

enemies.Clear();

for (int i = 0; i <= (xcells / 2 - 3); i++) //creates new enemies

{

Point ePoint = CreateRandomEnemy();

Map[ePoint.X, ePoint.Y] = true;

}

this.Invalidate(); //to trigger redrawing of the form

}

private void ResetBoard(bool[,] \_map, Point \_player, List<Point> \_enemies) //resets board to loaded file

{

path.Clear();

Map = \_map;

pathfinder = new Pathfinder2(Map, \_player, finish, false);

path = pathfinder.FindPath();

myGrid = new Grid(GetDrawRectangle(), Map.GetLength(0), Map.GetLength(1));

reqRedraw = true;

MyPlayer = new Player(\_player, this);

enemies.Clear();

foreach (Point p in \_enemies) //creates new enemies

{

CreateEnemy(p);

Map[p.X, p.Y] = true;

}

this.Invalidate(); //to trigger redrawing of the form

}

#region Event handlers

private void Main\_Load(object sender, EventArgs e)

{

ResetBoard();

loadedSounds = LoadSounds();

}

private void MainForm\_Resize(object sender, EventArgs e)

{

myGrid.SetSize(GetDrawRectangle());

}

private void MainForm\_KeyDown(object sender, KeyEventArgs e)

{

if (gamestatus == 0)

{

gamestatus = 1;

//musicPlayer.Play();

this.Invalidate();

}

else

{

switch (e.KeyCode)

{

case Keys.Up:

MyPlayer.Move(Directions.Up);

break;

case Keys.Right:

MyPlayer.Move(Directions.Right);

break;

case Keys.Down:

MyPlayer.Move(Directions.Down);

break;

case Keys.Left:

MyPlayer.Move(Directions.Left);

break;

case Keys.Escape:

this.Close();

break;

case Keys.Tab:

gamestatus = 0;

this.Invalidate();

break;

case Keys.R:

ResetBoard();

break;

default:

break;

}

}

}

private void Main\_Click(object sender, EventArgs e)

{

if (gamestatus == 0)

{

gamestatus = 1;

this.Invalidate();

}

}

private void ResetToolStripMenuItem\_Click(object sender, EventArgs e)

{

ResetBoard();

}

private void ChangeDimensionsToolStripMenuItem\_Click(object sender, EventArgs e)

{

SizeInputForm inputForm = new SizeInputForm(this);

inputForm.Show();

}

private void LoadFileToolStripMenuItem\_Click(object sender, EventArgs e)

{

LoadFile();

}

private void SaveBoardToFileToolStripMenuItem\_Click(object sender, EventArgs e)

{

SaveFile();

}

private void MusicOnToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (musicOnToolStripMenuItem.Checked)

musicPlayer.Play();

else

musicPlayer.Stop();

}

#endregion

}

}

## מחלקת Player

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Drawing;

namespace Pathfinding

{

public enum Directions { Up, Right, Down, Left }

public class Player

{

public Point Location { get; set; }

public int Radius { get; set; } = 3; //detection radius

private Main form;

public Player(Point location, Main form) //initialize Player

{

Location = location;

this.form = form;

}

public void Move(Directions dir) //checks if the cell is traversable, if so moves in said direction

{

switch (dir)

{

case Directions.Up:

if ((Location.Y > 0) && form.Map[Location.X, Location.Y - 1])

{

Location = new Point(Location.X, Location.Y - 1);

form.SoundPlayer(Sounds.Move);

}

else

form.SoundPlayer(Sounds.Collision);

break;

case Directions.Right:

if ((Location.X < form.Map.GetLength(0) - 1) && form.Map[Location.X + 1, Location.Y])

{

Location = new Point(Location.X + 1, Location.Y);

form.SoundPlayer(Sounds.Move);

}

else

form.SoundPlayer(Sounds.Collision);

break;

case Directions.Down:

if ((Location.Y < form.Map.GetLength(1) - 1) && form.Map[Location.X, Location.Y + 1])

{

Location = new Point(Location.X, Location.Y + 1);

form.SoundPlayer(Sounds.Move);

}

else

form.SoundPlayer(Sounds.Collision);

break;

case Directions.Left:

if ((Location.X > 0) && form.Map[Location.X - 1, Location.Y])

{

Location = new Point(Location.X - 1, Location.Y);

form.SoundPlayer(Sounds.Move);

}

else

form.SoundPlayer(Sounds.Collision);

break;

default:

break;

}

form.UpdateEntities();

}

public List<Point> DisplayRange() //gets the area that is the player's detection radius

{

List<Point> points = new List<Point>(); //points to color

for (int x = (Location.X - this.Radius); x <= (Location.X + Radius); x++) //2d loop that checks all points in the square that encloses the circle which its center is the player and radius is detection radius

{

for (int y = (Location.Y - Radius); y <= (Location.Y + Radius); y++)

{

if (Main.DisPwr(new Point(x, y), this.Location) <= (Radius \* Radius)) //if said point is in said circle

{

if ((x >= 0 && x < form.Map.GetLength(0) && y >= 0 && y < form.Map.GetLength(1))) //if the point is inside the map

{

points.Add(new Point(x, y)); //adds to list

}

}

}

}

return points;

}

}

}

## מחלקת Enemy

using System.Collections.Generic;

using System.Drawing;

using aStar;

namespace Pathfinding

{

public class Enemy

{

public Point Location { get; set; }

private Point goal;

private Main form;

private Pathfinder2 pfinder;

public Queue<Point> Path { get; set; } = new Queue<Point>();

public bool Active { get; set; } = false;

public Enemy(Point position, Main form) //initalizer

{

this.Location = position;

this.form = form;

pfinder = new Pathfinder2(form.Map, Location, Location, false);

}

public void Move() //moves one step

{

if (Path.Count > 0)

Location = Path.Dequeue();

}

public void Detected(Point at) //runs when detects the player in Main

{

if (!at.Equals(goal)) //when the goal is changed

{

Active = true;

goal = at; //sets goal to new point

pfinder = new Pathfinder2(form.Map, Location, at, false); //finds new path

Path = new Queue<Point>(pfinder.FindPath());

if (Path.Count > 0)

Path.Dequeue(); //remove the top of the list since the first node is the enemy's location

}

}

public bool IsInRange() //checks if this enemy is in the player's detection range

{

int radius = form.MyPlayer.Radius;

if (Main.DisPwr(this.Location, form.MyPlayer.Location) <= (radius \* radius))

return true;

else

return false;

}

}

}

## מחלקת Grid

using System.Drawing;

namespace Pathfinding

{

class Grid

{

public int Xcells { get; set; }

public int Ycells { get; set; }

public Rectangle Board { get; set; } //grid location and size

public Point[,] BoardCoord { get; set; } //stores individual cells location

public Size TileSize { get; set; }

public Grid(Rectangle board, int xcells, int ycells) //creats a resizeable grid then maps cells to given coordinates;

{

this.Xcells = xcells;

this.Ycells = ycells;

this.Board = board;

BoardCoord = new Point[xcells, ycells];

SetSize(board);

}

public Rectangle GetRectangle(int x, int y) //gets the specific tile's coord + size as Rectangle

{

return new Rectangle(BoardCoord[x, y], TileSize);

}

public void SetSize(Rectangle rect) // Resize function;

{

this.Board = rect;

TileSize = new Size(Board.Height / Xcells, Board.Height / this.Ycells);

CalcBoard();

}

private void CalcBoard()

{

for (int i = 0; i < Ycells; i++)

{

for (int j = 0; j < Xcells; j++)

BoardCoord[i, j] = CalcTile(new Point(i, j));

}

}

private Point CalcTile(Point tileCoord)

{

Point location = new Point();

location.X = (TileSize.Width \* tileCoord.X) + (Board.Left);

location.Y = TileSize.Height \* tileCoord.Y;

return location;

}

}

}

## מחלקת Setting

using System.Collections.Generic;

using System.IO;

namespace Pathfinding

{

public class Setting //general class used to save properties between runs;

{

public string Name { get; set; }

public string Value { get; set; }

public Setting(string name, string value)

{

this.Name = name;

this.Value = value;

}

public override string ToString()

{

return (Name + "=" + Value);

}

public static void WriteSettings(List<Setting> settings, string file, string comments) //Writes list of settings to file in the format "NAME=VALUE" (function ToString());

{

using (StreamWriter writer = new StreamWriter(file, false))

{

writer.WriteLine(comments);

foreach (Setting s in settings)

{

writer.WriteLine(s.ToString());

}

}

}

public static List<Setting> ReadSettings(string file) //Retrieves list of settings from file;

{

List<Setting> myList = new List<Setting>();

using (StreamReader reader = new StreamReader(file))

{

string line = "";

while (!reader.EndOfStream)

{

line = reader.ReadLine();

if (line != "")

{

if (line[0] == '\*') //quits reader when reaches the map flag

break;

if (line.Contains("=") && line[0] != '/')

{

string sName = line.Split('=')[0];

string sValue = line.Split('=')[1];

myList.Add(new Setting(sName, sValue));

}

}

}

}

return (myList);

}

}

}

/\* Example of implementing the settings list retrieved;

\*

public void runSettings(List<Setting> myList)

{

foreach(

{

switch (myList[i].Name)

{

case "top":

this.Top = Convert.ToInt32(myList[i].Value);

break;

case "left":

this.Left = Convert.ToInt32(myList[i].Value);

break;

case "isMaximized":

if (Convert.ToBoolean(myList[i].Value))

this.WindowState = FormWindowState.Maximized;

break;

default:

break;

}

}

}

\*/

## טופס SizeInputForm

using System;

using System.Windows.Forms;

namespace Pathfinding

{

public partial class SizeInputForm : Form

{

private Main mainform; //the form to which SizeInputForm works on

private int gridWidth;

private int gridHeight;

private int input;

public SizeInputForm(Main mainform) //initialize form

{

InitializeComponent();

this.mainform = mainform;

gridWidth = mainform.Map.GetLength(0); //sets default size to current map dimensions

gridHeight = mainform.Map.GetLength(1);

textBoxWidth.Text = gridWidth.ToString(); //sets textbox text to defaults

textBoxHeight.Text = gridHeight.ToString();

}

private bool TestValue(string \_in) //tests the input is valid

{

int number;

bool result = Int32.TryParse(\_in, out number);

if (result)

{

this.input = number;

return true;

}

else

return false;

}

private void OKButton\_Click(object sender, EventArgs e) //on user confirmation checks the values are valid then updates them in Main

{

if (TestValue(this.textBoxWidth.Text) && input <= 300 && input >= 10) //tests the input isn't too big

{

//gridWidth = input; //for simplicity's sake only square grids are allowed.

if (TestValue(this.textBoxHeight.Text) && input <= 300 && input >= 10)

{

gridHeight = input;

gridWidth = input;

}

else

{

MessageBox.Show("Error: invalid input. Enter a number between 10 and 300.", "Error", MessageBoxButtons.OK);

}

}

else

{

MessageBox.Show("Error: invalid input. Enter a number between 10 and 300.", "Error", MessageBoxButtons.OK);

}

mainform.SetCellsCount(gridWidth, gridHeight); //sets the grid size of the main form to the user's input

}

}

}

1. הסיבה לשם Pathfinder2 היא שזו הגרסה השנייה של הקוד. הראשונה לא השתמשה בתור עדיפויות ונטתה לא לעבוד כראוי עם קלט גדול, מה שגרם לי להפיל אותה לטובת הגרסה הנוכחית. [↑](#footnote-ref-1)