public class Tetris {

טטריס:

}

אני מניח שלא משנה בני כמה אתם, אתם כנראה מכירים את השם "טטריס". היו ימים בהם טטריס היה משחק המחשב הפופולרי ביותר, שכבש את העולם והכניס לחברות שונות מיליארדים של דולרים. אני הכרתי אותו לראשונה כ"משחק רוסי" (ככה הוא שווק אז) שהגיע לאחד מחברי לכיתה על דיסק פלופי שחור וגדול. המשחק הציג באנימצית ASCII פשוטה בלוקים נופלים מראש המסך, שצריך להתאים אותם ולסדר אותם כדי לסגור שורות. ברקע היתה מנגנינה (באמצעות טכנולוגית האודיו של שנות ה-80, היא נשמעה כמו סדרת צפצופים) עם שורשים רוסיים, ותמונות של הקרמלין ועוד סמלים רוסיים.

בשנות התשעים המוקדמות המשחק היה פופולרי גם במין מכשירים יעודיים קטנים שהריצו אך ורק טטריס, בשחור לבן (מדהים היום לחשוב שמכשיר שהיה בגודל סמארטפון ממוצע, היה מוגבל למשחק אחד בלבד וזה מה שהוא ידע לעשות). המכשירים האילו באמת כבשו את הארץ, אנשים היו משחקים בזה בכל זמן נתון, היה קשה מאוד להפסיק, התמכרות של ממש.

בכל אופן, המשחק עדיין נחשב למאסטרפיס – הוא פשוט להבנה, מכיל מעט חוקים וקל למשחק, ועם זאת פוטנציאל העניין שהוא מעורר בשחקן הוא עצום. לכן בפרק זה נפתח משחק טטריס. השתדלתי להשאר נאמן למקור ככל האפשר, אולם כיוון שגרסאות הטטריס הן רבות ומשונות, קשה לאתר מהו בדיוק ה"מקור" הזה, ככה שהיכן שהיה נוח, נקטתי גישה יותר גמישה לפיתוח. ככה או ככה, מדובר באתגר אמיתי, אתגר עיצובי ותכנותי שמערב כמעט כל מה שהנדסת תוכנה טובה מכילה. אני נהניתי לפתח את המשחק הזה ונהניתי לחשוב על הבעיות והאתגרים שהוא הציב עבורי, מקווה שגם אתם תהנו.

1. קיצור תולדות הטטריס

המשחק פותח ב-1984 בבריה"מ דאז, ע"י שני מהנדסי מחשוב מהאקדמיה למדעים במוסקווה – אלכסיי פז'יטנוב ודימיטרי פבלובסקי. המשחק הוא שילוב של המילה "טטרה" (ארבע ביוונית, על שם ארבע הקוביות שמרכיבות כל בלוק במשחק) ו"טניס" – המשחק האהוב על פז'יטנוב. השניים פיתחו את המשחק עבור מחשב ספציפי שעליו הם עבדו, ולכן המשחק לא הופץ בתפוצה רחבה.

אולם לאחר שנער בשם ואדים גראסימוב פיתח גרסת MS-DOS למשחק, המשחק הדביק את כל בריה"מ ונהיה מה שהיינו קוראים לו היום "ויראלי".

דרך הונגריה, שאז היתה שער ליצוא תוכנה מהמזרח למערב, המשחק הגיע גם לאירופה ולארה"ב, שם פותחו גרסאות שלו על כל פלטפורמה ממוחשבת קיימת. מפתחי המשחק המקוריים, אגב, לא הרוויחו אף לא רובל מזכויות המשחק, כיוון שהם היו עובדים ממשלתיים, והחוק הסובייטי בזמנו היה פרוץ לחלוטין בכל הנוגע לזכויות יוצרים על תוכנה. בשל הקושי בתקשורת בין הגוש המזרחי למערבי באותם זמנים והאלמוניות של מפתחי המשחק המקוריים, חברות רבות פיתחו גרסאות למשחק בלי לתת את הדין לגבי זכויות היוצרים שלו. מאוחר יותר ניטשו מאבקים משפטיים בין ענקיות התוכנה השונות על זכויות היוצרים של המשחק.

2. תיאור המשחק

משחק הטטריס נחשב למשחק פאזל, כלומר משחק שבו צריך להרכיב צורות שנות כך שיתאימו או יבנו צורה אחרת. המשחק מורכב מלוח דו ממדי של N שורות על M עמודות (בדר"כ מדובר ב-20 שורות על 10 עמודות). חלקי הפאזל נקראים טטרומינו (Tetromino) – כלומר חלקים שבנויים מארבע משבצות, בצורות שונות. טטרומינו אחד נופל בקצב מסויים מראש הלוח. המשתמש יכול להזיז אותו ימינה או שמאלה, או לסובב אותו על צירו ימינה או שמאלה. כאשר מגיע החלק לשורה בה הוא לא יכול להמשיך להתקדם, הוא נשאר שם, וטטרומינו חדש מתחיל לפול מראש הלוח. אם כתוצאה מהגעת החלק לשורה, כל משבצות השורה מלאות, השורה נעלמת וכל השורות שמעליה "נופלות" שורה אחת למטה.

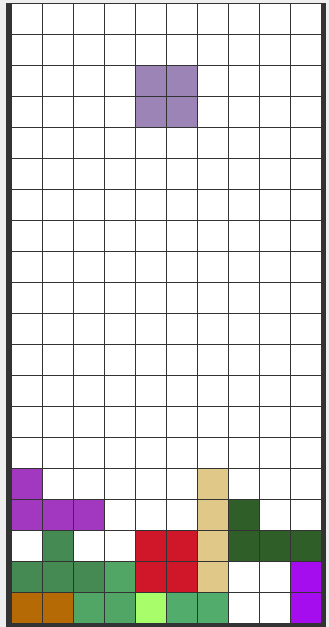
אם לא נותר מקום להפיל חלקים נוספים, המשחק נגמר בהפסד.

באופן עקרוני, המשחק הוא אינסופי – אין איזושהי מטרה שצריך להגיע אליה. הצורות ממשיכות ליפול, והשורות ממשיכות להיסגר, וכל עוד המבנה נשאר נמוך מספיק ולא מגיע לראש הלוח, אפשר בעקרון לשחק לנצח.

איור 1 מראה את חמשת הטטרומינו הסטנדרטים, וכינוייהם. איור 2 מראה את לוח הטטריס לאחר מספר תורות.

Line L J S Z Q

**איור 1 – אבני הבניין של הטטריס – כל חמשת סוגי הטטרומינו.**



**איור 2 – לוח טטריס אחרי מספר תורות.**

3. שיקולי עיצוב

כשהתחלתי לחשוב על המשחק התברר שלמרות פשטותו היחסית, הוא מכיל לא מעט "חלקים נעים" שצריך לקחת בחשבון, והרבה פרטים טכניים שצריך לפתור. מבחינת שיקולי העיצוב החשובים, הנה כמה מהשאלות הרלבנטיות שצריך לשאול בקשר לפיתוח המשחק –

1. איך נייצג את הצורה? איך נייצג את הצורות השונות? ברור שיש להן משהו משותף, לכן הגיוני להשתמש בירושה, מצד שני, האם הן מספיק שונות אחת מהשניה בכדי להצדיק יצירת מחלקת אב אחת ועוד חמש מחלקות יורשות? האם זה לא סיבוך מיותר?
2. איך נייצג את הלוח? הלוח מורכב ממשבצות – האם כל משבצת צריכה להיות אובייקט או שאפשר להסתפק במערך דו ממדי של int?
3. איך נתכנת את האינטראקציה בין הצורות ללוח? בזמן שהצורה נמצאת בתנועה, היא עדיין לא חלק מהלוח. מצד שני, ברגע שהצורה מפסיקה לזוז היא "תורמת את גופה למדע" – המשבצות מהן הצורה מורכבת הופכות להיות חלק מהמשבצות התפוסות של הלוח, והצורה כבר לא קיימת כצורה. שימו לב באיור 2, למשל, שחלק מהשורות מורכבות מחלקי צורות שכבר נעלמו בגלל שורות שנמחקו. הצורה לא משמרת את המהות שלה ברגע שהיא מפסיקה לנוע אלא להיפך, היא מתפרקת למרכיביה, שמוטמעים בתוך הלוח והופכים להיות חלק ממנו.
4. בנוסף לסעיף 3 – אם הצורה אינה חלק מהלוח בזמן התנועה, איך נוכל לבצע את התנועה שלה בצורה נכונה? למשל, איך נוכל לכתוב את הרוטציה של הצורה בזמן התנועה בלי לדעת איפה היא נמצאת בלוח ומה יש מסביבה?
5. וכמובן – בסופו של דבר אנחנו מעוניינים להפיק מעבר למשחק טטריס, גם תוכנה טובה. כלומר, תוכנה איכותית, שכתובה נכון, מודולרית וקלה לתחזוקה. אני לא מחפש מפלצת של מאות שורות קוד שעובדת אבל שאי אפשר לגעת בקרביים שלה. אני רוצה מוצר תוכנה חזק שניתן לשינוי בקלות יחסית ושקל לתחזק אותו.

אני חושב שדווקא השאלה האחרונה היא זאת שגורמת להכי הרבה כאב ראש. כיוון שאני מניח שכל מי שתכנת כמה דברים בחייו יוכל לתכנת טטריס בצורה כזאת או אחרת. אני חושב שאם לא לוקחים בחשבון את כל שאלות הנדסת התוכנה ואת ענייני הגמישות לשינויים, מקבלים גרסה יותר קלה לכתיבה. אפשר לוותר על כלליות, אפשר לוותר על תחזוקה, על קבועים, על תבניות.

אולם אני ממליץ בחום לא ליפול לפח הזה. האמת שזאת אולי אחת הבעיות הגדולות שיש לאקדמיה בהכשרת מהנדסי תוכנה – אנחנו לא מצליחים ללמד את הסטודנטים מספיק טוב את החשיבות של כתיבת מוצר איכותי ולא רק מוצר שעובד. אני חושב שלא משנה מה כותבים – אם זה טטריס או תוכנה לניהול מלאי – הרגלי תכנות נכונים חייבים להיות תמיד, וחייבים להתרגל לכתוב בצורה נכונה מההתחלה. לכן אני אתעקש לעצב את המשחק בצורה כללית ככל שאוכל, בצורה שתאפשר לי לבצע שינויים עתידיים בקוד בצורה קלה יחסית ושתאפשר לי להשתמש ברכיבים שפיתחתי לצורך מודולים אחרים בעתיד.

אז מקווה שהשתכנעתם, כי זה הולך להיות עיצוב ארוך, הנה זה בא:

4. עיצוב הצורה

בתור התחלה, בחרתי להתמודד עם עיצוב הצורה. מצד שני, שאלות קשות העיקו עלי ולא הניחו לי להמשיך לפתח בצורה ישרה, כי כל הזמן נזרקתי חזרה לאותן התלבטויות כמו איך הצורה תיוצג, איך היא תישמר, איך היא תסתובב במישור וכו'. זה היה מעגלי ומייאש, עד שנזכרתי שבשביל זה המציאו את הממשקים – כדי שנוכל לכתוב כותרות לפונקציות בלי לכתוב כלום.

אז ניסיתי לעקוב אחרי עקרון חשוב שאומר שצריך לתכנת לממשק ולא למימוש. לפחות בשלב הזה, אופן יצוג הצורה הוא לא בעיה מהותית, או בעיה בכלל. אפשר לייצג את הצורות בהרבה דרכים, והאמת שאף אחת מהן לא משפיעה כהוא זה על המהות של הצורה מהבחינה המופשטת. בעצם ניסיתי להבין מה עושה את המושג "צורה", מה מרכיב אותה ומהווה את המהות שלה, ולא איך המהות הזאת נבנית. נשמע אולי פילוסופי קצת, אבל זה גם הבסיס לעיצוב נכון של תוכנה – לחשוב על הפונקציונליות הנדרשת, ולא על דרך המימוש שלה. מימוש יכול להשתנות ועדיין לספק את אותה פונקציונליות, אז זה מה שאני מנסה לתפוס.

אז בואו נראה מה מרכיב את המושג "צורה":

כיוון שאנחנו יודעים שהמשחק הוא צבעוני, אנחנו גם יודעים שלכל צורה יש צבע שמרכיב את הקוביות שלה, אז צבע היא תכונה של צורה.

הצורה גם נמצאת איפשהו במישור, בתנועה כלפי מטה מראש הלוח לתחתיתו. הלוח מורכב משורות ועמודות, לכן המיקום של הצורה הוא השורה והעמודה בהם היא נמצאת, ולכן גם שורה ועמודה הם תכונות של צורה.

הנה נקודה שצריך לשים לב אליה – מי שחונך מספיק זמן על ברכי העיצוב מונחה העצמים בטח יתרעם על הפסקה האחרונה – האם מיקום הצורה לא יוצר קשר יותר מידי גדול בין הצורה ללוח? האם זה לא יוצר coupling שרצוי להמנע ממנו בין שני רכיבים?

התשובה היא כן ולא. כן, זה יוצר coupling במידה מסויימת, אבל אי אפשר להמנע ממנו במיוחד, כיוון שאנחנו לא מפתחים פה מושג אבסטרקטי שנקרא צורה גיאומטרית שאמור להתאים לכל אפליקציה גיאומטרית ובמקרה כרגע אנחנו משתמשים בה לטטריס. אנחנו מפתחים צורה שאמורה לתפקד בתוך משחק הטטריס, ובמשחק הזה יש גם לוח. בלי זה אין לצורה משמעות. ולכן הצימוד פה, בדמות המיקום, הוא משהו טבעי. לצורה במשחק הטטריס יש מיקום. אי אפשר לדבר על צורה ללא מיקום, לפחות לא במשחק טטריס, ולכן מיקומי הצורה הם תכונות לגיטימיות ונחוצות.

ומהצד השני, למרות זאת, עצם העובדה שהגדרנו "שורה" ו-"עמודה" בתור מיקום הצורה לא בהכרח רומז על המימוש, לפחות לא בצורה שממש תפתיע אותנו – אנחנו מדברים על משחק שמתנהל על לוח דו ממדי. לוח כזה אפשר לדמיין כמערכת צירים קרטזית דו ממדית. אפשר למקם כל דבר על המערכת ע"י ציון קואורדינטות x ו-y. ואפשר גם לחשוב על המיקום כשורה ועמודה, אם מחלקים את מערכת הצירים למשבצות. ככה או ככה, זה לא חשוב, העובדה היא ש"מיקום" היא תכונה מוגדרת היטב גם בלי לדעת איך בסוף זה יראה. להגדיר את המיקום כתכונה של הצורה לא אומר עוד שום דבר לגבי המימוש, **כל עוד ברור לנו שאנחנו עדיין בלוח דו ממדי**.

אז בנתיים יש לנו את צבע הצורה ומיקומה.

האם יש צורך בתכונות נוספות? כרגע קשה לראות, אז בואו נמשיך ונחקור את הפונקציונליות הנדרשת מהצורות, ואולי דרכה נגלה עוד תכונות רצויות.

מה הצורה אמורה לעשות?

דבר ראשון, ליפול, או לזוז כלפי מטה, בקצב מסויים. הקצב הוא לא העניין של הצורה, כי היא לא אמורה לשלוט בשאלה של מתי היא זזה (זה תלוי באלגוריתם המשחק עצמו, ולצורה אין נגיעה בזה), אבל התזוזה עצמה היא כן עניינה של הצורה. לכן נצטרך שיטה אחת שנקרא לה step. השיטה תקדם את הצורה צעד אחד למטה, אם אפשר ותחזיר true, או שתשאיר את הצורה במקומה אם אי אפשר לזוז, ותחזיר false. שוב, המימוש עצמו יבוא אח"כ, אבל בגדר פונקציונליות, זה נשמע לא רע. אגב, אפשר לחשוב שניה על העתיד הקרוב ולשאול את השאלה – רגע, אבל איך הצורה תדע אם אפשר לזוז או לא? היא תצטרך לדעת האם המיקום שלה נמצא בסמוך לשורה תפוסה בלוח. זאת אומרת, שהיא צריכה לדעת משהו על הלוח... נשוב לזה מיד.

בנתיים נמשיך לחשוב על עוד דברים שהצורה צריכה לדעת לבצע. למשל, לזוז ימינה או שמאלה בצורה אופקית. אז נגדיר שיטה בשם move שתקבל כפרמטר את כיוון ההתקדמות (זמן טוב להגדיר משתנים קבועים סטטיים במחלקה עבור הכיוונים) ותזיז את הצורה צעד אחד בכיוון זה, אם אפשר. אם אי אפשר, הצורה לא תזוז. ושימו לב שהשיטה לא תחזיר ערך בוליאני עבור הצלחת התזוזה או כשלונה, כי זה לא משנה. בניגוד לתזוזה כלפי מטה – חוסר הצלחה בתזוזה ימינה או שמאלה לא משנה לאלגוריתם המשחק. פשוט לא זזים. מצד שני, כשלון בביצוע תנועה כלפי מטה אומר הרבה – צריך לצרף את הצורה ללוח, לבדוק שורות, הפסד, לייצר צורה חדשה ועוד, ככה שלאלגוריתם המשחק כנראה יהיה חשוב לדעת האם התזוזה מטה הצליחה או לא, ולכן השיטה step מחזירה תשובה בוליאנית, והשיטה move לא.

פונקציונליות נוספת היא כמובן סיבוב הצורה על צירה, ימינה או שמאלה. נצטרך שיטה בשם rotate, שבדומה לשיטה move, תקבל כפרמטר את כיוון הרוטציה ותבצע אותה אם אפשר.

שתי פונקציות נוספות שקיימות במשחק המקורי הן הנחתה והאצה. הנחתה היא האפשרות להנחית צורה ישר לתחתית הלוח, פעולה שנקראת drop, כלומר בלחיצה על מקש לא צריך לחכות עד שהצורה תיפול בקצב שלה אלא נוצר "קיצור דרך" והצורה מתמקמת בסוף נתיב ההתקדמות הנוכחי שלה.

האצה היא פעולה בה ניתן להאיץ באופן זמני את קצב נפילת הצורה. לחיצה מתמשכת על מקש תאיץ את התקדמות הצורה כל עוד המקש נלחץ. הצורה תשוב לקצב ההתקדמות המקורי שלה ברגע שנעזוב את המקש.

בכל הפעולות האילו הצורה צריכה לדעת בצורה מאוד חזקה את מבנה הלוח ובפרט את המקומות התפוסים והפנויים בו, כיוון שכל תנועה של הצורה על הלוח אפשרית רק אם יש מקום פנוי. יש שתי אופציות לטפל בעניין – אפשרות אחת היא להעביר לפונקציות כפרמטר את הלוח. אפשרות שניה היא להגדיר את הלוח כתכונה של המחלקה ולהעביר אותו כפרמטר בבנאי.

מה עדיף? אני חושב שמבחינת תכנון טהור אולי עדיף להעביר את הלוח כפרמטר לשיטות, מתוך העקרון שאומר שמה שהשיטה צריכה לצורך עבודתה, כל מידע שהיא צריכה, היא תקבל מהמשתמש שמפעיל אותה.

מצד שני, אם מסתכלים בראיה רחבה יותר של OOP, האובייקט הוא ישות גדולה יותר משיטה והוא גם יכול לספק מידע לשיטה. אנחנו כבר עושים את זה – במחלקת הצורה כבר מוגדרות תכונות כמו row ו-col שהשיטות ישתמשו בהן לצורך חישוביהן. למה לא להגדיר גם תכונה נוספת מסוג הלוח?

בכל אופן, אני לא בטוח שיש עדיפות לפתרון אחד או אחר (מלבד העובדה שאם אנחנו מוסיפים את הלוח כתכונה אנחנו בעצם אומרים שהצורה "מורכבת מלוח", דבר שאולי נשמע קצת לא הגיוני).

אז בשעה טובה אפשר להגדיר את מחלקת Shape – מחלקה מופשטת שמייצגת צורה ומכילה שיטות ממשק בלי מימוש: