הבהרה 1:

כפי שמוסבר בתרגול, API זה ההוראות לשימוש באובייקט מסוים. מי שכתב את האובייקט מספר לעולם איך להשתמש בו (אילו מתודות יש, מה הן מקבלות ומחזירות וכו')

בתרגיל זה, אנחנו סיפקנו לכם שלד של מספר מחלקות, אשר מהוות את ה-API של כל מחלקה בהתאם. לדוגמא, שימוש במחלקת הלוח תיעשה רק על ידי המתודות שקיבלתם. ולכן אם מחלקה אחרת צריכה מידע ממחלקת הלוח, מותר לה להשתמש רק במתודות שב-API.

מותר לכם להוסיף עוד מתודות לכל מחלקה, שיעזרו לכם לממש את המתודות שב-API, אבל הן לא יהיו חלק מה-API. לדוגמא, אם אתם רוצים להוסיף למחלקת המשחק שדה שסופר את כמות התורות שהיו (ואולי מתודה שתעדכן אותו בכל תור), אתם יכולים, אבל דבר זה לא יהיה חלק מה-API, כלומר לא יהיה נגיש למחלקות אחרות.

מה שזה אומר, זה שהקוד שלכם אמור להיות כזה שאם נחליף כל אחד מהמחלקות שכתבתם, במחלקה שאנחנו מימשנו (שתומכת באותו API), המשחק אמור לעבוד. וזה רק בגלל שאנחנו עובדים בשיטה של design by contract, כלומר רק ה-API קובע את התפקוד של המחלקה, ולא המימוש הספציפי.

ובאופן כללי זה עקרון מאוד מאוד חשוב שמאפשר לכותבי תוכנה לשנות ולשפר את התוכנה שלהם בלי לגרום לכל מי שתלוי בו לעשות שינויים אצלו.

הבהרה 2:

ב-API כלולים השמות של המתודות, הפרמטרים שהם מקבלים(כולל הסוג שלהם) ומה הם מחזירים. כמו כן גם שדות מסוימים יכולים להיות חלק מה-API.

ולכן ברגע שמגדירים API, הכותב מתחייב שהקוד יעבוד עבור הקלטים הנכונים לכל מתודה. במידה ובכל זאת מוזן קלט לא תקין במתודה, הדרך לפתור את זה, זה דרך exceptoins (תלמדו השבוע).

אבל אתם לא צריכים להשתמש בזה בתרגיל הנוכחי. מה שזה כן אומר זה שלא צריך להתעסק עם קלט לא חוקי למתודות. כלומר, אם מתודה צפויה לקבל שני פרמטרים, אחד רשימה ואחד int, אז אתם יכולים להניח שזה מה שהיא תקבל (עם ערכים חוקיים).

**עוד כמה הערות רק כדי לוודא שלא פספסתם -**

**1. זה אולי לא היה מספיק ברור אבל הרצף של המשחק אמור להיות:**

**א. יצירת לוח**

**ב. קריאת ה-JSON, ייצור מכוניות והוספה שלהם ללוח**

**ג. רק עכשיו ליצור משחק ולהעביר אליו את הלוח**

**כל זה קורה ב- name==main או בפונקציות (לא מתודות) שנקראות משם**

**2. שימו לב איפה המקומות היחידים שעליכם להדפיס שגיאות. אין לעשות הדפסות בשום מקום אחר**

**3. שימו לב שאם הוספתם מתודה למחלקה כלשהי, היא לא חלק מה-API שלה ואתם לא יכולים להשתמש בה במחלקה אחרת!**

הערה כללית

חלק מהעניין בעבודה לפי API וחלוקת אחריות בין מחלקות היא להפוך את הקוד למודולרי. דבר זה מאפשר לנו לקחת מחלקות קיימות לשימוש חוזר בתוכנות שונות (לדוגמא להשתמש ב-car עם מחלקת game אחרת עם חוקים קצת אחרים), או יקל עלינו מאוד אם נרצה להרחיב את התוכנית (לדוגמא להוסיף סוגי מכוניות, להוסיף תאי מטרה על הלוח וכו'). ולכן בתרגיל הנוכחי, המכונית מניחה כמה שפחות הנחות (=לא מכירה את חוקי המשחק או את הלוח) כדי שבעתיד אם נרצה לשנות את המשחק זה יהיה יותר קל. כמובן שלתרגיל הזה אין "עתיד", ולא נרצה לשנות אותו אבל זאת חשיבה מאוד חשובה להפנים כי תוכנות אמיתיות, בין אם הן תוכנות מסחריות או קוד המשמש למחקר (בכל זאת אנחנו באוניברסיטה), כמעט תמיד יעברו שינויים והוספות וכו' (תחשבו על כל עדכוני האפליקציות שיש לכם בפלאפון).

לדוגמא אם אתם רוצים לשמור רשימה של צבעים חוקיים, היה אפשר לשים את זה במחלקת המכונית, אבל אז אנחנו מגבילים מכוניות להיות בצבעים הללו, גם אם נרצה לקחת את המחלקה הזו למשחק אחר. ולכן צבעים חוקיים צריכים להיות תכונה של המשחק ולא של המכונית. אפשר לקחת את זה צעד אחד קדימה ולהגיד שמכונית גם לא תדע מה זה אוריינטציה ואז זה עוד יותר גנרי. הבעיה בזה היא שמה שנוריד מהמכונית נצטרך לשים במחלקה אחרת (כנראה במחלקת המשחק) ואז יכול להיווצר מצב שמחלקת המשחק תהיה עצומה עם הרבה תחומי אחריות, דבר שאנחנו לא רוצים. לכן יש איזון בין גנריות של מחלקות לבין חלוקת אחריות ביניהם וה-API הוא בדיוק מה שמגדיר את האיזון שבחרנו.

לכן הרבה פעמים בפורום הופיעו תשובות קצת מעורפלות כי הכוונה שלהן הייתה בעלת ראיה לטווח ארוך, ולא ספציפית לתרגיל הנוכחי. אבל חשוב להדגיש שלמרות שאנחנו מתכננים את הקוד ככה שיהיה קל לשנות אותו בעתיד, כרגע הוא צריך לתמוך במשחק ספציפי המתואר בתרגיל (ולכן אין דבר כזה מכונית שזזה שני ריבועים במהלך אחד - למרות שזה אולי משהו ש"בעתיד" נרצה להוסיף).

הערות נקודתיות

שימו לב ששדות (=משתנים שמגדירים ב-init) הם חלק מה-API. בפרט בתרגיל, אין שדות שהם חלק מה-API ולכן אם לדוגמא יש לכם רשימת מכוניות בלוח ומחלקת המשחק צריכה מידע ממכונית, היא לא יכולה פשוט לגשת לרשימה שקיימת בלוח.

צריך למצוא דרך חלופית (דרך ה-API) להשיג את המידע.

באותו עניין, מותר להוסיף שדות למחלקות לשימוש פנימי, בפרט מותר להוסיף שדות גם אם הם לא קשורות לפרמטרים המתקבלים ב-init. לדוגמא אם המכונית מקבלת 4 פרמטרים ב-init אפשר להוסיף עוד שדות מעבר לארבעת הפרמטרים. רק שימו לב שלמחלקות אחרות אין גישה לשדות בכלל.

המתודות שניתנו לכם כחלק מהAPI הם שם לשימושכם. ממשו אותן ואז העזרו בהן למימוש הלוגיקה של המשחק. לדוגמא בשביל לדעת את צבע האוטו, כיוון שלא ניתן לגשת לשדה self.name של car, אז צריך להשתמש במתודה get\_name. וככה, מהבנה טובה של המתודות ואיך הן עובדות ביחד אפשר להרכיב את המשחק שיעמוד ב-API במוגדר לכם.

לגבי possible\_moves של **הלוח**, אנחנו נקבל את שתי האפשרויות (לפי המצב הנתון, או לפי אוריינטציה כללית בלי התחשבות בלוח). מחלקת ה-Board שלנו תחזיר תזוזות לפי מצב הלוח

ניתן להניח כי נתיב קובץ הJSON חוקי וגם הקובץ עצמו יהיה חוקי (לא צריך לוודא שהוא קיים). הוא כן יכול להכיל מילון ריק מה שאומר שלא יהיו מכוניות וכל קלט מהמשתמש לא יעשה כלום. בנוסף ניתן להניח שהקלט למתודות תקין (זה הרעיון ב-API, שדרכו אומרים מה אמור להתקבל ואם מעבירים פרמטר לא תקין לא מובטח שהקוד יעבוד). אבל כן צריך לוודא את הקלט של המשתמש שמשחק במשחק (שימו לב שקלט זה אינו חלק מה-API)

זה כבר נאמר, אבל שימו לב להיררכיה בין המחלקות. המכונית לא מכירה (= אין לה גישה ל-API) של אף מחלקה אחרת. הלוח מכיר רק את המכונית. המשחק מכיר את שניהם. כלומר המשחק הוא היחיד שמכיר את כל חוקי המשחק, הלוח והמכונית מכירים רק דברים שקשורים אליהם ישירות. היררכיה זו אמורה גם להגיד לכם איפה מבצעים פעולות שונות, כולל איפה שמים קבועים למיניהם.

באופן כללי הצורה שבה התרגיל יבדק הוא שהמכונית, שלא תלויה במחלקה אחרת תיבדק בפני עצמה. הלוח יבדק עם מחלקת מכונית שלכם ומחלקת מכונית שלנו. והמשחק ייבדק עם המחלקות שלכם, עם מכונית שלנו ולוח שלכם ולהיפך. בכל מקום שיש מחלקה שהיא לא שלכם, אנחנו יכולים לספק מחלקה חלופי שעונה על דרישות ה-API אבל מתפקדת קצת אחר (כמו לדוגמא שינוי של משבצת המטרה).

דבר אחרון, שמנו לב שיש הרבה שאלות שנשאלו בפורום שיש עליהן כבר תשובות בתיאור התרגיל, שגם אם לא כתובות במפורש, ניתן להסיק אותם בקריאה נוספת, אחרי שמבינים יותר טוב מי נגד מי. אנו ממליצים מאוד לקרוא שוב את הוראות התרגיל ולנסות למצוא שם רמזים ותשובות נוספות ובנוסף, לראות שלא הפרתם שום כלל.