יוני 2020 – שנה"ל תש"ף

# Atomic Towers

עבודת גמר

## תכנון ותכנות מערכות 883599


**מגיש:** עידו סבן (ת.ז. 212619787)

**מורה:** ישראל אברמוביץ'

**כיתה:** י"ב 6

**בית ספר:** תיכון קוגל, חולון

# תוכן עניינים

# מבוא

הפרויקט Atomic Towers הוא משחק עבור מערכת ההפעלה Android תחת ז׳אנר המשחקים Tower Defense, שקיבל השראה ממסדרת משחקי Bloons TD. הפרויקט למעשה משתמש במרכיבי סביבת ה-Android באופן בלעדי, ללא שימוש במנוע משחקים חיצוני, כמו Unity למשל.

ספר פרויקט זה מכיל בעיקר תיאור של כל מרכיבי הפרויקט, ממערכת המשחק שבניתי ועד למרכיבי ה-Android וספריות שהשתמשתי בהן. כמו כן, ישנו גם מדריך למשתמש המתאר את מסכי האפליקציה, האינטראקציה ביניהם, את הוראות המשחק, וגם את הדרישות להתקנתו. הספר מכיל כמו כן את כל קובצי הפרויקט החשובים, ביניהם קובצי source של Java, וכן קובצי נתונים/משאבים (resources) של XML ו-JSON, והן קובצי תמונות מסוג PNG. החסר בספר הפרויקט הוא רק קובץ ה-MP4 של מוזיקת הרקע במשחק.

ישנם מספר אתגרים מרכזיים שעמדו בפני בזמן בניית הפרויקט:

האתגר הראשון היה החוסר במנוע משחקים, כפי שציינתי לעיל. ללא שימוש במנוע משחקים, הייתי למעשה צריך לבנות מערכת מצומצמת עליה יפעל המשחק, דבר שלקח זמן רק לתכנן. הרבה מהרעיונות לארכיטקטורת הפרויקט לקחתי גם ממנועי משחקים כמו Unity, על מנת לקבל רעיונות. למעשה הייתי צריך להתחיל הכל כמעט מאפס: תחילה היה צריך ליצור לולאת משחק יציבה, אחרי כן יצרתי את מערכת ה-Components שבפרויקט, עליה מבוססים חלקי המשחק השונים (אטומים, מגדלים, וכו׳), ולבסוף הייתי צריך גם לקשר בין מרכיבי ה-Android למרכיבי המשחק, בדרכים שאסביר במהלך הפרויקט.

אתגר נוסף היה מציאת הדרך הפשוטה והטובה ביותר לשמור את כל נתוני המשחק. היות שלמשחק נותנים רבים, הייתי צריך למצוא פתרון לאופן שבו אני מרכז אותם וכן שומר אותם. לבסוף בחרתי לשמור את נתוני הפרויקט בקובצי JSON, כפי שאסביר בהמשך.

אתגר שלישי שעלה היה שמירה ע ביצועים סבירים של המשחק על מכשיר נייד. היות שלא עשיתי שימוש בהאצה גרפית עקב הסיבוך שבכך, הייתי צריך למצוא דרכים ופתרונות שישמרו על ביצועי המשחק, כך שירוץ בצורה חלקה על מגוון מכשירי טלפון.

אתגר רביעי שהיה בפרויקט הוא הלמידה ושהימוש בספריות רבות, ביניהם ספריות עיקריות לפרויקט כמו RxJava ו-Gson. הדבר דרש זמן למידה רב, שנחלק לאורך השנה האחרונה, ולאור השימוש המרובה בספריות היה גם צורך לחשוב על ארכיטקטורה מתאימה לאפליקציה על מנת לשמור עליה עד כמה שיותר מסודרת.

אתגר חמישי ואחרון שהתמודדתי איתו בפרויקט היה הלמידה על ארכיטקטורת אפליקציות Android, למשל, תבנית הצופה, MVVM. למרות שדבר נועד לשמור על הסדר באפליקציה ולהקל על הרחבתה ומציאת בעיות בעתיד, הוא גם הופך אותה לפרויקט גדול יותר, ולכן יש לשמור על רצף עבודה ברור ומסודר.

המשחק משמש למטרות הנאה בלבד עבור המשתמש, ולהעברת שעות הפנאי – כרוב המשחקים לטלפון הנייד. את הרעיון לפרויקט קיבלתי מאבי, שכן אהב לשחק במשחקים כאלו בעבר. לאור הרצון לגוון להביא רעיון מקורי, חשבתי עם אבי על נושאים מעניינים ומקוריים למשחק, עד שעלה הרעיון למשחק שעוסק בתחום הכמיה. בניגוד למשחקים אחרים, בפרויקט יש שימוש ברעיון שונה – הגנה נגד "פלישת אטומים".

רציתי לבנות פרויקט שמעמיד את הידע שלי ואת יכולותיי במבחן, ואכן בניית משחק שכזה היא אתגר לא פשוט, במיוחד עקב החוסר במנוע משחקים או סביבה ייעודית.

# מבנה וארכיטקטורה

הפרויקט למעשה מציג רעיון שעומדות מאחוריו צורות פתרון רבות. על מנת לשמור על הסדר בפרויקט ולשמור על הרעיון Separation of Concerns, בחרתי בארכיטקטורה שהפכה לטבעית ב-Android כיום והיא ארכיטקטורת MVVM (ראה הסבר תחת "עקרונות ומושגים").

למעשה, אפשר לחלק את הפרויקט לארבעה חלקים עיקריים על פי ארכיטקטורת MVVM:

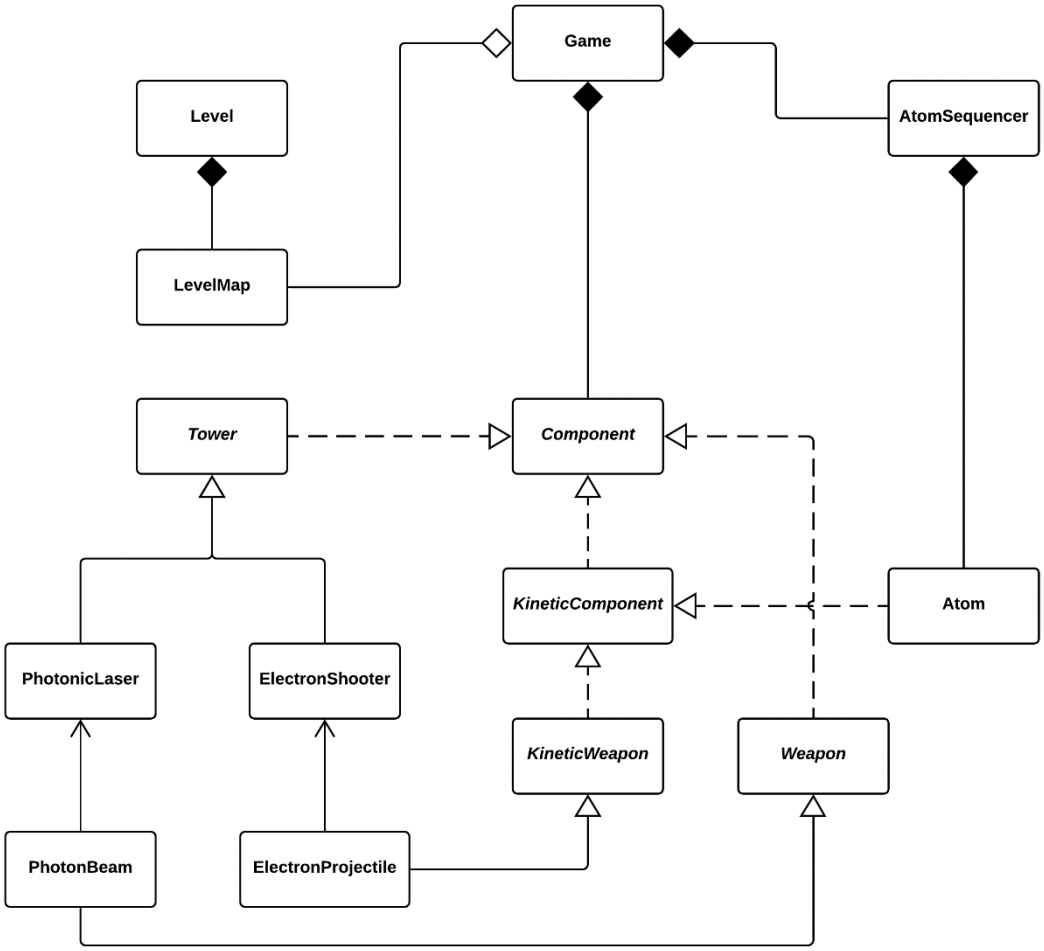
1. **Model** – כולל את כל מרכיבי הפרויקט שאינם תלויים למעשה בסביבת העבודה של Android. המרכיבים שנכללים הם למעשה כל המחלקות תחת תת-החבילה (package) בשם components, שזה כולל את מחלקת המשחק Game, ואת כל המחלקות שיורשות מ-Component, נוסף למספר מחלקות עזר שהמחלקות לעיל עושות בהן שימוש ספציפי.
2. **View** – שכבת התצוגה של האפליקציה. למעשה היא כוללת את כל ה-Fragments ואת MainActivity של הפרויקט, את המחלקה GameView המשמשת לציור המשחק על המסך, וכן את קובצי ה-XML תחת res/layout, res/navigation, res/menu, וכדומה.
3. **ViewModel** – מהווה את השכבה המחברת בין ה-Model ל-View, ומכאן שמה. למעשה נכללת כאן רק המחלקות GameViewModel ו-SettingsViewModel, שכן רק ה-GameFragment וה- SettingsFragment עושות שימוש בקישוריות כזו (שאר המסכים יכולים להיחשב כסטטיים).
4. **Repository** – מהווה את שכבת הנתונים באפליקציה. למעשה באפליקציה קיימת מחלה אחת שנקראת GameRepository, והיא אחראית לספק את כל נתוני המשחק לאפליקציה על ידי חשיפת API פשוט, שקורא וכותב לזיכרון קובצי JSON, אם כי שאר האפליקציה לא צריכה להיות מודעה לעובדה זו, כי באותה המידה היה אפשר להשתמש באופן אחר לאכסן את הנתונים. למעשה, GameRepository עושה שימוש במספר רב של מחלקות נתונים, מהן היא יוצרת אובייקטים המכילים נתונים המועברים לאפליקציה, כך שהיא שומרת על האבסטרקציה של האופן בו שמור הנתונים מפני הרכיבים המשתמשים בה. אפשר לומר בקירוב כי כל המחלקות המדוברות לעיל נמצאות תחת החבילה data.game.

השימוש בארכיטקטורה שתוארה לעיל אפשרה לא רק סדר בפרויקט כי שכבר צוין, אלא גם מודולריות, שכן היה קל ליצור שינויים במהלך בניית הפרויקט. במהלך הפרק אתייחס לשכבות המתוארות לעיל בפרקים נפרדים תואמים.

# שכבת ה-Model

זוהי השכבה הרחבה ביותר בפרויקט – היא למעשה מכילה את כל הלוגיקה שבין רכיבי המשחק, וממשת את מערכת המשחק שבניתי. עבור הקוד של כל המחלקות ראה את הפרק "מדריך למפתח".

## תרשים קשרים ומחלקות (UML)




ישנן מספר מחלקות שלא נמצאות בתרשים לעיל על מנת לשמור על מהותו. מחלקות אלו כוללות: מחלקות של מבני נתונים, מחלקות ששייכות לשכבה אחרת (כמו GameRepository), מחלקות עזר לשמירת נתונים כמו Vector2, וכן מחלקות שבאות מספריות בהן אני עושה שימוש בפרויקט, כשהבולטות בשימוש הן מחלקות מהספרייה RxJava.

## פירוט המחלקות והקשרים ביניהן

חשוב לציין שלא אתייחס באופן יחידני לכל המחלקות שבתרשים, אלא אתייחס לעיקריות, ואתן הסבר לשאר כחלק מההסבר על המחלקות העיקריות.

### Game

זוהי מחלקה שמאחדת בין כל חלקי הפרויקט, ובצורה הפשוטה ביותר, מייצגת משחק יחיד – כשרוצים להתחיל משחק חדש, יוצרים אובייקט חדש של Game. המחלקה מכילה נתונים ופעולות רבים, אך אתאר רק את הבולטים והחשובים להבנה.

תחילה, המחלקה מכילה את כל הנתונים על אותו השלב, והחשוב ביניהם הוא המפה של השלב, LevelMap, הממדים של המשחק (בפיקסלים), ששמורים באובייקט Vector2 תחת השדה mDimensions, וגודל של אריח במשחק תחת השדה mTileSize מסוג float. את הנתונים האלו מאתחל המשחק ביזמן יצירתו, על ידי שימוש בפרמטרים שהוא מקבל בפעולה הבונה ובנתונים שהוא משיג דרך GameRepository. הנתונים הללו משמשים את המשחק לצייר את מפת המשחק של המסך, אך משמשים גם חלק גדול מרכיבי המשחק השונים שהוא מכיל.

כמו כן, המחלקה מכילה גם אובייקט מסוג SparseArray<Component>, שמשמש מעקב אחר כל ה-Components שעל המסך (ראה תרשים כדי להתרשם מהקשר שבין המחלקות). זהו למעשה מבנה נתונים של מפה, שכל המפתחות שלה הם מסוג int. יש רק דרך אחד למעשה לשנות את ערכי המפה הזו, והיא דרך הפעולות addComponent() ו-removeComponent().
למעשה, בעת יצירת Component חדש על ידי הפעולה addComponent(), נוצר לאותו Component מספר ID ייחודי (על ידי הפעולה generateComponentId()) והוא מתווסף למפה. מרגע זה, כל עוד לא מסירים אותו מהמפה על ידי קריאה לפעולה removeComponent() עם מספר ה-ID שלו, בכל פריים של המשחק נקרא על אותו ה-Component הפעולות update() ו-draw(), בפעולות בעלות שמות זהים במחלקה Game. במילים אחרות, מהרגע שנוסף Component למפה, המשחק מעדכן ומצייר אותו עד הסרתו.

שדה בולט נוסף שקיים במחלקה הוא אובייקט של AtomSequencer. למעשה, עבור כל אובייקט משחק חדש, נוצר אובייקט יחיד של AtomSequencer, שלמעשה אחראי ליצור אטומים חדשים במרווחי זמן קבועים, הנתונים על פי המידע השמור ב-JSON, אותו הוא משיג דרך GameRepository. אפשר לומר שזהו האובייקט שמניע את השלב קדימה.

אלו הם החלקים החשובים לדעתי במחלקה Game, אך לא אמשיך לפרט על הכל שכן מדובר במחלקה די ארוכה, שמכילה גם הרבה פעולות עזר.

## Component

זוהי מחלקה אבסטרקטית המהווה בסיס לכל ה-Components במשחק. למעשה כל מה שהיא מכילה הם שני שדות: ייחוס לאובייקט של המשחק (שמטרתו דומה ל-Context בסביבת העבודה של Android ורכיביו), וכן את ה-ID שקיבל בעת יצירתו.

מלבד אלו, למחלקה מסר פעולות אבסטרקטיות חשובות:

1. getPosition() – מימושה צריך להחזיר אובייקט של Vector2 שמתאר את מיקומו על מסך המשחק.
2. update() – מימושה צריך לעדכן את ערכי האובייקט על פי הפרש זמן שהוא מקבל. פעולה זו נקראת בפעולה Game#update().
3. draw() – מימושה צריך לצייר את עצמו על המסך באמצעות הפניה לאובייקט Canvas שהוא מקבל. פעולה זו נקראת בפעולה Game#draw().

ישנה גם הפעולה (שאינה אבסטרקטית), destroy() – פעולה זו נקראת כאשר יש רצון בהריסת ה-Compoent, ולמעשה מימושה במחלקה Component קורא לפעולה Game#removeComponent() עם ה-ID של עצמו.

## KineticComponent

זוהי מחלקה אבסטרקטית שיורשת את Component, ולמעשה מרחיבה את Component כך שיוכל לנוע על המסך, ולכן קינטי. המחלקה מכילה שני שדות נוספים: אחד הוא Vector2 של וקטור מהירות ה-Component, ושני גם הוא Vector2 שמציין את מיקום "המטרה" שלו. למעשה, KineticComponents משנים בעקביות את מיקומם על פי המהירות שלהם, שמשתנה בצורה שהם ינועו לכיוון מטרתם. לשם כך, קיימות פעולות כמו setTartget(), שמאפשרת לקבוע מטרה, ו-setVelocity(), שמאפשרות לקבוע את גודל המהירות.

## Atom

זוהי מחלקה שיורשת מ-KineticComponent ומתארת אטום יחיד. מדובר במחלקה שמכילה יחסית הרבה פעולות וערכים, ולכן אתמקד בעיקר כמו לפני כן. בעצם, לכל אטום (גם במשחק וגם בטבע) יש מספר אטומי. לכן, במשחק פירשתי את המספר האטומי כ"חוזק" (Strength), שלמעשה מייצג את ה"חיים" של אותו אטום במשחק. בכל פעם שאטום חווה פגיעה, כלומר חוזקו קטן, אם החוזק נמוך מכדי להיות מתאים למספר האטומי שהוא נמצא בו כרגע, הוא ירד במספר האטומי שלו בהתאם, ויתעדכן בהתאם סוגו (הרי שלכל מספר אטומי יש שם וסוג מסויים).

חלק גדול מהפעולות של המחלקה עוסקות בעיקר בחישובים טכניים עבור ציורו על המסך, לכן לא אתייחס אליהן. הפעולות החשובות הן בעיקר הדריסה של הפעולה update(), שנוסף למימוש שבמחלקת הבסיס, דואגת לעדכן את סוגו על ידי בדיקת ערכי החוזק שלו ומספרו האטומי, תוך שימוש בפעולה changeElement(), שמורידה את מספרו האטומי באחד ומעדכנת את כל הדרוש. כמו כן, הפעולה update() גם דואגת לעדכן את "מטרת" האטום, שהוא למעשה אריח המשחק הבא, כך שהוא ינוע על השביל של מפת המשחק.

האטום גם שומר ייחוס לאובייקט של AtomDrawable, שהוא אחראי לצייר אותו על המסך ב-Canvas שהוא מקבל. האטום קורא בפעולת ה-draw() שלו לפעולה AtomDrawable#draw().

## Tower

זוהי מחלקה אבסטרקטית שיורשת מ-Component ומייצגת מגדל במשחק, שהוא זה שתוקף את האטומים. המחלקה בעיקר מכילה תכונות כלליות של מגדל: מיקומו על מפת המשחק; טווח הירי שלו (נמדד בגודל אריחי משחק); אינטרוול הירי שלו (כל כמה זמן הוא יורה – אם לא קיים, הערך הוא 0); אטום מטרה שהוא מסמן (משתנה על פי מימוש של כל מחלקה יורשת); אובייקט WeaponType, שמתאר את תכונות הנשק שלו, באמצעותו הוא יוצר Weapons, עליהם אסביר בהמשך; וגם אובייקט של Drawbale אותו הוא מצייר בפעולת ה-draw() שלו.

כרגע קיימות שתי מחלקות שיורשות את Tower במשחק: ElectronShooter, שהוא מגדל שיורה אלקטרונים על אטומים; PhotonicLaser, שהוא מגדל שיורה אלומת פוטונים רציפה על אטומים, למעשה לייזר.

## Weapon

זוהי מחלקה אבסטרקטית שיורשת מ-KineticComponent ומייצגת את הנשק בו המגדלים במשחק משמשים. למעשה, המגדלים, Towers, יוצרים אובייקטים שיורשים את Weapon, כל מגדל יוצר את הנשק המתאים לו, כשה-Weapons עצמאיים באופן יחסי מהמגדל. המחלקה מכילה מספר תכונות עיקריות: ייחוס לאטום המטרה, אליו הם יכוונו ויפגעו בו; המיקום שלהם על מסך המשחק (Vector2); ואת הנזק שהם עושים, אם כי כל מגדל יכול להשתמש בערך זה בצורה שונה.

בהתאם למגדלים שקיימים במשחק, יש שתי מחלקות שיורשות את Weapon:
ElectronProjectile, שמייצגת אלקטרון אחד שיוצר המגדל ElectronShooter באינטרוול הנתון, שכשאר פוגע באטום הוא נהרס ופוגע בחוזק באטום; PhotonBeam, שמייצגת את קרן הלייזר שנורית מהמגדל PhotonicLaser, ופוגעת באטום באופן רציף ובקצב קבוע עד שנהרס.

# שכבת ה-View

עבור שכבה זו, שמייצגת את ממשק המשתמש, אין צורך בתרשים מחלקות, שכן מדובר בעיקר במסכים השונים באפליקציה. את הקשרים שבין מסכי האפליקציה ראה תחת הפרק "מדריך למשתמש". אתייחס כאן לקובצי ה-Java ולקובצי ה-layout יחדיו.

## פירוט המחלקות והקשרים ביניהן

### MainActivity (activity_main.xml)

לאור השימוש בספרייה Navigation Architecture Component (ראה הסבר תחת התת-פרק "ספריות"), ה-MainActivity משמשת רק מעין מכולה, או container, לכל מסכי האפליקציה. לפיכך, כמעט שהיא לא מכילה כלום מלבד כמה דברים טכניים, וכן אובייקט של MediaPlayer, באמצעותו מתנגנת מוזיקת הקרע של האפליקציה. בעצם כל התוכן של המחלקה עוסק בניגון המוזיקה ועצירתה כשצריך, וכן הנגשתה לשאר לחלקי האפליקציה (על מנת שיוכלו לשלוט בה).

### MainFragment (fragment_main.xml)

מחלקה זו מייצגת את המסך הראשי, ממנו מגיעים שלשאר חלקי האפליקציה. המחלקה מכילה שתי פעולות עיקריות:

1. showPopupMenu() – יוצרת PopupMenu חדש כשלוחצים על כפתור התפריט שעל המסך הראשי, ממנו היא מנווטת למסכי ההגדרות, הוראות ואודות.
2. showResumeLastGameDialog() – יוצרת AlertDialog חדש כשלוחצים על הכפתור "Start", אם קיים משחק קודם שנשמר בזיכרון שלא נגמר. הדיאלוג שואל את השחקן אם ברצונו להתחיל משחק חדש או להמשיך את המשחק הקודם שעצר. משם, השחקן מנווט למסך המשחק.

### GameFragment (fragment_game.xml)

מחלקה זו מייצגת את מסך המשחק, שמהווה את המסך העיקרי בפרויקט. GameFragment מקושרת ל-GameViewModel, ולמעשה המסך שלה מכיל את ה-GameView ואת הסרגל שבצד, בו ניתן לראות את נתוני המשחק השונים. עיקר התוכן של המחלקה הוא:

1. המחלקה יוצרת BroadcastReceiver שמאזין לניתוק אוזניות מהמכשיר. ברגע שהוא מקבל את ה-broadcast אליו הוא מאזין, הוא משהה את המשחק וקורא לפעולה showGamePauseDialog().
2. המחלקה יוצאת מאזין לכפתור ההשהיה של המשחק – אם נלחץ, המשחק מושהה והפעולה showGamePauseDialog() נקראת.

3. הפעולה showGamePauseDialog() – יוצרת AlertDialog חדש ששואל את השחקן אם ברצונו לחזור למסך הראשי (אם נבחר, אז הוא מנווט למסך הראשי) או להמשיך לשחק (אם נבחר, ממשיך את המשחק שהושהה).
4. הפעולה showGameEndedDialog() – יוצרת AlertDialog חדש שמציג האם השחקן ניצח או הפסיד, בהתאם להודעה שהפעולה מקבלת. ה-Dialog מכיל אפשרות אחת והיא לחזור למסך הראשי.

**GameView**

מחלקה זו יורשת מ-SurfaceView ומייצגת את מסך המשחק עליו מצטיירת מפת המשחק וכל רכיבי המשחק. הסיבה שהמחלקה יורשת מ-SurfaceView היא על מנת לאפשר ציור ועדכון המשחק על Thread אחר, שהוא לא ה-Main Thread (או ה-UI Thread). בעצם המחלקה גם מממשת את הממשק של Runnable (גם את LifecycleOwner, אבל זה לא רלוונטי – זה רק מאפשר להשתמש באנוטציות של @OnLifecycleEvent), וממשת את הפעולה run(). הפעולה run() מכילה את לולאת המשחק, ואפשר לומר שהמטרה העיקרית של המחלקה היא לנהל את לולאת המשחק.

אתן הסבר פשוט על הפעולה run() שמממשת המחלקה: כשהפעולה נקראת, היא מתחילה בלולאת while שרצה עד שהשדה mRunning הוא false. בתוך הלולאה, יש בקשה ל-SurfaceHolder של ה-GameView לנעול את ה-Canvas, על מנת שיהיה אפשר לעדכן את המשחק ולצייר אותו עליו, ולבסוף לשחרר אותו. על מנת לשמור על לולאת משחק חלקה, יש מדידה של הזמן שלוקח לכל פריים לעדכן את המשחק ולצייר אותו – אם הזמן לקח פחות ממה שאמור לקחת כפי שמוגדר בקבוע FRAME_PERIOD של המחלקה, אז מרדימים את ה-Thread להפרש הזמן שנשאר. כמו כן, את הזמן שלקח לעדכן ולצייר מעבירים גם בקריאה לפעולה Game#update(), כך שהמשחק יעדכן את עצמו בהתאם (הדבר מונע קפיצות או תנועה לא עקבית של רכיבים המשחק על המסך).

הפעולה run() נקראת בתוך הפעולה resume(), שיוצרת Thread חדש ומעבירה לו את ה-GameView כ-Runnable, על מנת שהפעולה run() תרוץ על ה-Thread החדש. ישנה גם פעולה pause(), שקוראת לפעולה Thread#join() על ה-Thread עליו רצה לולאת המשחק, שבמילים אחרות עוצרות את המשחק. שתי הפעולות resume() ו-pause() נקראות כאשר ה-GameFragment עוברת למצבים של onCreate() ו-onPause(), בהתאמה, וישנם חלקים שונים בדוק שקוראים לפעולות אלו בנוסף.

המחלקה גם מכילה פעולה שנקראת saveGameState(), שיוצרת GameStateService חדש, ששומר את המצב הנוכחי של המשחק לזיכרון המכשיר, ומשמש לקריאה מאוחרת יותר אם המשחק נעצר בזמן הריצה (למשל, האפליקציה נסגרה או שהשחקן השהה את המשחק וחזר למסך הבית). הפעולה pause() קוראת לפעולה זו.

כמו כן, המחלקה גם דורסת את הפעולה onTouchEvent() ומעבירה את המיקום עליו לחץ המשתמש על המסך לאובייקט המשחק, כך שיחליט אם אפשר לשים שם מגדל או לא, ואם כן, לשים מגדל חדש.

**SettingsFragment (fragment_settings.xml)**

מחלקה זו מייצגת את מסך ההגדרות של האפליקציה. ההגדרה היחידה שקיימת במסך זה היא שליטה בעוצמת מוזיקת הרקע. מחלקה זו מקושרת למחלקה SettingsViewModel, שביחד הן דואגות לשמור את עוצמת הווליום שנבחרה ב-Shared Preferences ולעדכן את ממשק המשתמש בהתאם לנתונים שנשמרו.

**InstructionsFragment (fragment_instructions.xml)**

מחלקה זו מייצגת את מסך ההוראות של האפליקציה. מסך ההוראות סטטי, ומכיל רק את ההוראות למשחק.

**AboutFragment (fragment_about.xml)**

מחלקה זו מייצגת את מסך האודות של האפליקציה. מסך האודות סטטי, ומכיל מידע על קרדיטים באפליקציה, כמו קרדיט למוזיקת הרקע תחת הרשיון.

# שכבת ה-ViewModel

שכבה זו משמשת כחלק המקשר שבין ה-View ל-ViewModel. אין צורך בתרשים מחלקות, כיוון שה-ViewModels לא קשורים האחד לשני.

## פירוט המחלקות והקשרים ביניהן

### GameViewModel

זוהי המחלקה המקשרת (ומגשרת) בין מערכת המשחק לממשק המשתמש במסך המשחק. למעשה, עיקר תכולתה הוא אובייקט ה-Game היחיד שהיא מכילה, אותו היא יוצרת בפעולה הבונה שלה, כלומר עם יצירתה. היות ש-Game מכילה את כל נתוני המשחק ומקשרת בין רכיביו, כל מטרתו של GameViewModel היא לשמור על נתוני המשחק (במיוחד בגלל שהיא לא תלויה במחזור החיים של GameFragment), ולקשר חלק מנתוני המשחק לממשק המשתמש באמצעות מספר אובייקטי LiveData (למידע נוסף, ראה תת-פרק "ספריות"), שמתעדכנים בהתאם לנתוני המשחק הרלוונטיים השמורים באובייקט של Game.

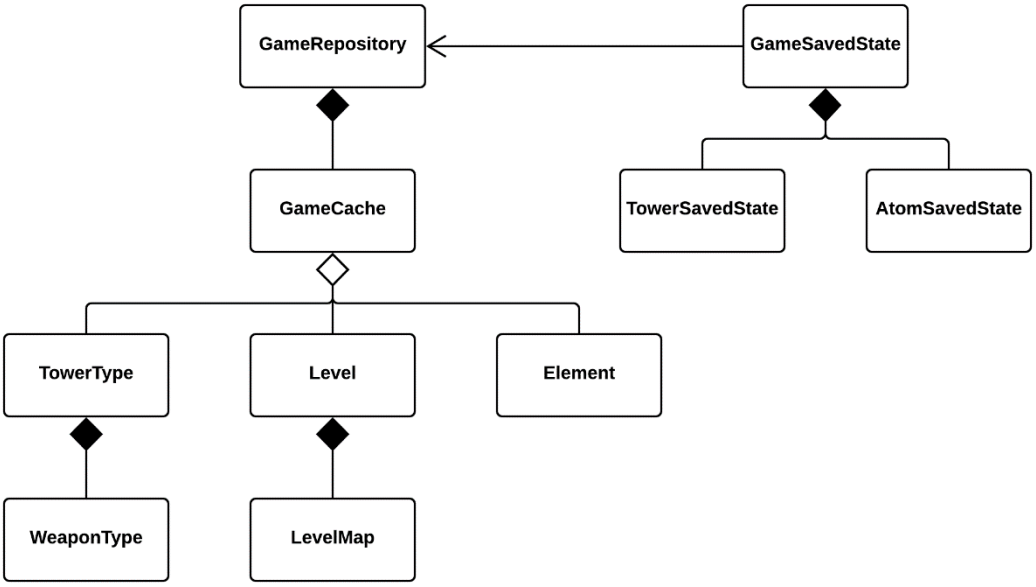
### SettingsViewModel

זוהי המחלקה המקשרת (ומגשרת) בין ממש המשתמש לנתונים במסך ההגדרות. המחלקה מכילה מספר אובייקטי LiveData למטרה זו.

# שכבת ה-Repository

שכבה זו בעצם מציגה API ציבורי לשאר האפליקציה, ומהווה אבסטרקציה מעל לנתונים השמורים על המכשיר של האפליקציה. חשוב לציין, שיש שיחשיבו שכבה זו כחלק משכבת ה-Model, אבל אני בחרתי להפריד בין השניים.

## תרשים קשרים ומחלקות (UML)




## פירוט המחלקות והקשרים ביניהן

### GameRepository

זוהי המחלקה שאחראית למימוש ה-API לגישה לנתוני המשחק. המחלקה היא בעצם סינגלטון, שעל מנת לגשת לאובייקט שלה יש להשתמש בפעולה הסטטית GameRepository#getInstance(). המחלקה מכילה פעולות שונות המשמשות לקריאה ודסריאליזציה של קובצי JSON השמורים הן כמשאבים והן בזיכרון המכשיר ( internal storage), וכן השגת אובייקט Drawable של משאב. עבור קבצים שאינם משתנים, כלומר קבצים שהם משאבים של האפליקציה, GameRepository קוראת ומפרשת אותם פעם אחת, ושומרת את התוצאה (או "מטמינה") אותה באובייקט של GameCache שהיא מכילה, לשימוש מאוחר יותר. עבור קובצי JSON יש שימוש גם בספרית הסריאליזציה Gson של גוגל.

**GameSavedState**

זוהי מחלקת נתונים שמייצגת את מצב המשחק ברגע נתון. השדות העיקריים הם: החיים שיש באותו משחק; האנרגיה שיש באותו משחק; רשימה מסוג AtomSavedState, שמייצגת את המצבים של כל אטום על מסך המשחק (מקום, חוזקה, מספר אטומי, וכו'); רשימה מסוג TowerSavedState, שמייצגת את המצבים של כל מגדל אחד על מסך המשחק (מקום, וכו').

למעשה אובייקט מסוג GameSavedState עובר סריאליזציה ל-JSON שנשמר בזיכרון הפנימי של המכשיר (internal storage) בתוך ה-GameStateService, שמשמש מאוחר יותר לקריאת הנתונים חזרה ואחזור המצב הקודם של המשחק.

**Level**

מתאר נתונים של שלב במשחק (מספר השלב, השם שלו, רצף האטומים שלו עבור AtomSequencer, וכולי), ומכיל בתוכו גם אובייקט של GameLevel, שמייצג את מפת המשחק, ומשמש ליצירת אובייקט Game וכן את שאר רכיבי המשחק בזמן הרצתו. כל ה-Levels של המשחק שמורים בקובץ levels.json במשאבי האפליקציה.

**Element**

מתאר נתונים של אטום עם מספר אטומי מסוים (שם, סימן כימי, מספר פרוטונים ונויטרונים, וכולי), שמשמש ליצירת אובייקט חדש של Atom. כל ה-Elements של המשחק שמורים בקובץ elements.json במשאבי האפליקציה.

**TowerType**

מתאר נתונים של מגדל (שם, טווח ירי, אינטרוול ירייה, וכולי), שמשמש ליצירת אובייקט חדש של Tower, וכן כולל בתוכו גם אובייקט מסוג WeaponType, שמתאר את נתוני הנשק בו משתמש המגדל (נזק, מהירות, וכו'). כל ה-TowerTypes של המשחק שמורים בקובץ towers.json במשאבי האפליקציה.

# מדריך למשתמש

## הוראות המשחק

המשחק Atomic Towers עוסק בהגנה של הבסיס מפני פלישת אטומים. קיימים במשחק שני מגדלים, שכל מטרתם היא לפגוע באטומים במטרה להרוס אותם לפני שהם מגיעים לבסיס. לכל מגדל יש את היכולות שלו ואת המחיר שלו, ולכן יש לשים לב היכן מציבים כל מגדל ובאיזה סוג של מגדל בוחרים.

לאטומים יש מהירויות שונות שתלויות במספר האטומי שלהם, אותו ניתן לזהות לפי הסימן הכימי שעל האטומים במשחק – ככל שהמספר האטומי נמוך יותר, כך הם מהירים יותר.
לאטומים יש חוזק, וכשעוברים רף מסוים של נזק, האטום מאבד פרוטון ויורד במספרו האטומי.
כך כל מגדל למעשה פוגע בהדרגה באטומים עד שהם מאבדים את כל הפרוטונים שלהם ונהרסים.

במהלך המשחק אפשר לשים לב לשני מדדים: אנרגיה וחיים. בכל פעם שאטום מגיע לבסיס, יורדים לבסיס חיים. אם החיים של הבסיס מגיעים לאפס, השחקן מפסיד. כמו כן, בכל פעם שאלקטרון מאבד פרוטון או נהרס, השחקן מקבל נקודת אנרגיה אחת, שנמדדת בג'ולים.
באמצעות נקודות האנרגיה ניתן לקנות מגדלים נוספים.

למעשה, השחקן מנצח את המשחק כאשר השלב נגמר, כל האטומים נהרסו, ולא נגמרו החיים של הבסיס.

# דרישות מיוחדות ומגבלות

אין דרישות או הרשאות מיוחדות.

# בניית הפרויקט והרצתו

גרסת ה-Android המינימלית לפרויקט היא API 19 (KitKat).

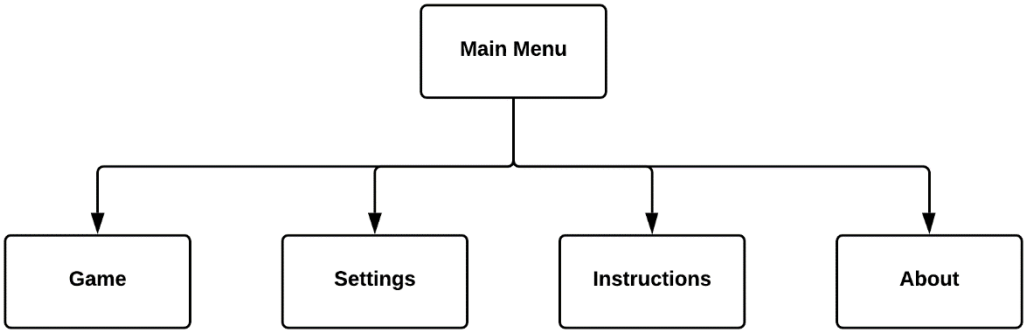
הפרויקט משתמש בתכונות השפה Java 8, לכן על מנת לבנותו יש לוודא כי סביבת העבודה תומכת ב-Java בגרסה 8 או יותר.

# מכשירים עליהם נבדק הפרויקט

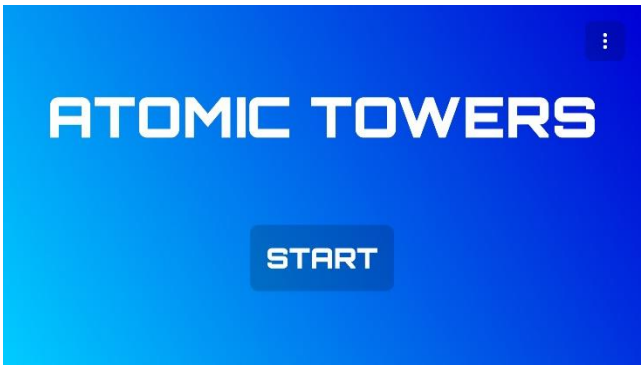
- מכשיר פיזי (אישי): Meizu MX6, עליו רץ Android API 25.

# תפעול מערכת הפרויקט (המשחק)

**תרשים זרימה בין המסכים**

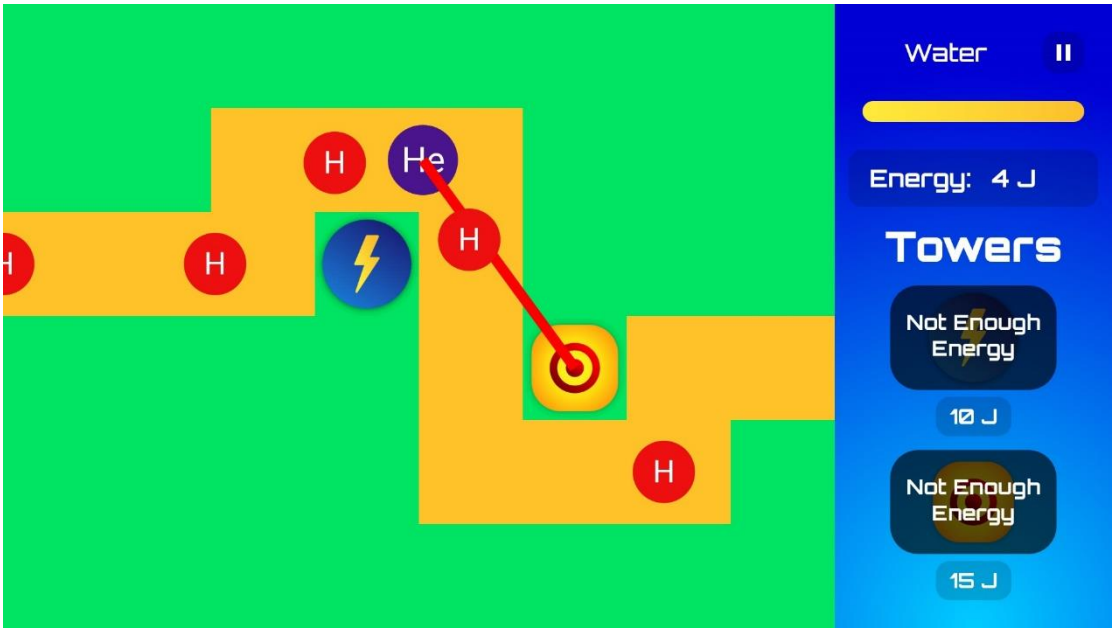


**מסך הפתיחה (Main Menu)**

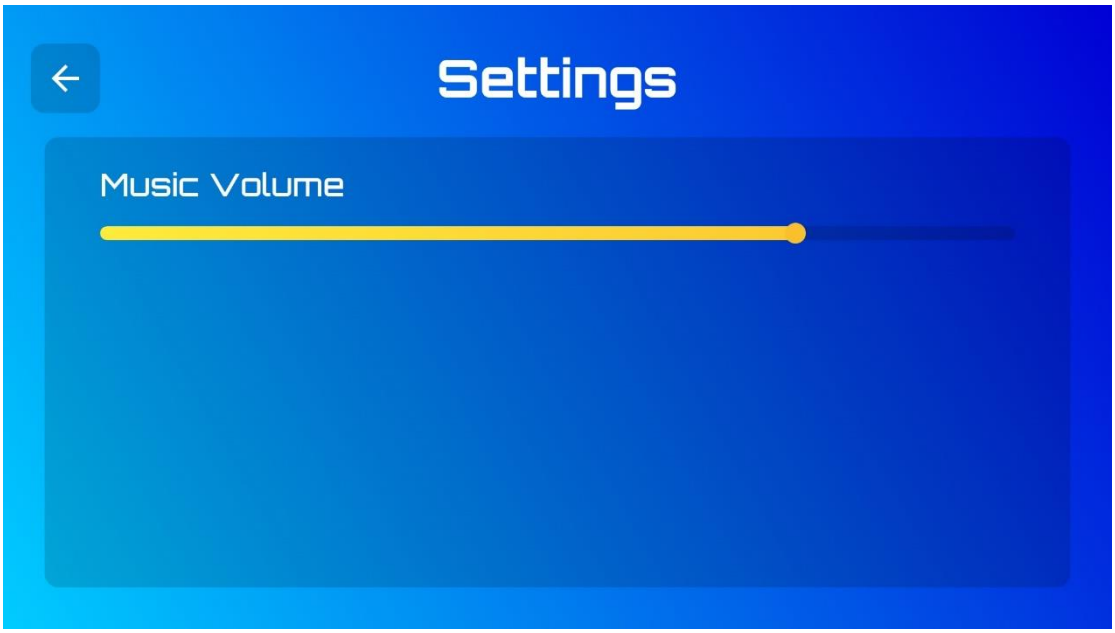

זהו המסך הראשון אלו מגיע השחקן, ומכיל את התפריט הראשי. המסך מציג שני כפתורים: זה שכתוב עליו "Start" מתחיל משחק חדש, ואילו כפתור התפריט בחלק הימני-עליון של המסך פותח תפריט, שמאפשר לנווט למסכי ההגדרות, ההוראות והאודות. מסך זה משמש למעשה נקודת מוצא לשאר המשחק, כלומר ממנו מנווטים אל שאר המסכים.

**מסך המשחק (Game)**



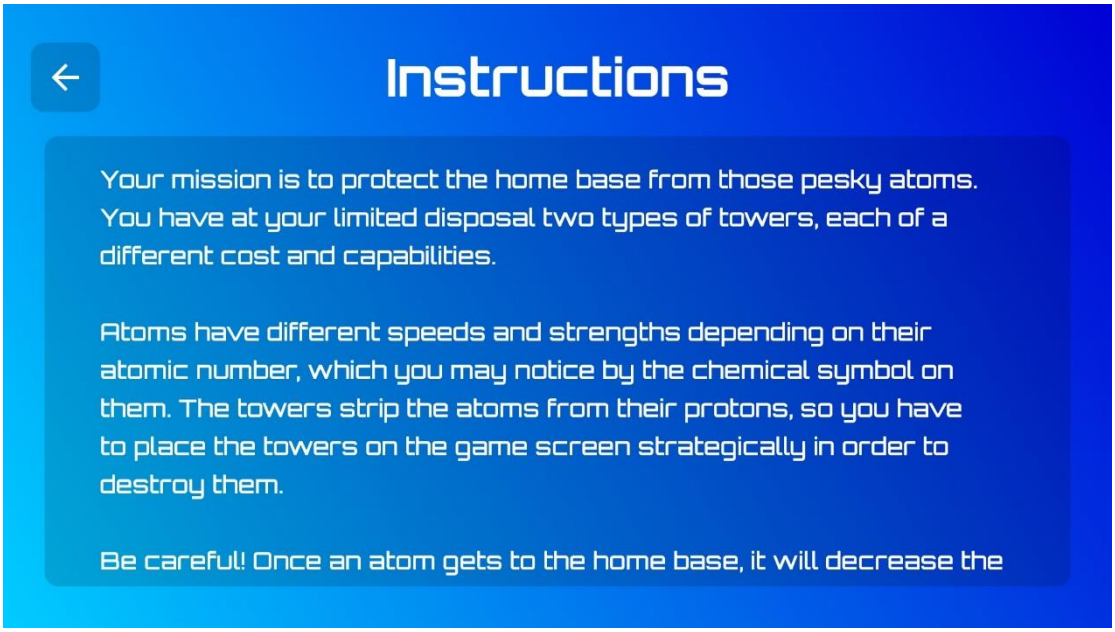

מסך זה הוא עיקר הפרויקט. במסך זה מתנהל המשחק עצמו. בסוף השלב, יופיע לשחקן דיאלוג, שיציג האם השחקן ניצח או לא. במסך זה קיימי מספר כפתורים: הראשון הוא הכפתור בפינה העליונה הימנית, שמשהה את המשחק אם לוחצים עליו ומעלה דיאלוג, ששואל אם ברצון השחקן לחזור למסך הבית או להמשיך לשחק. שני הכפתורים הנוספים הם כפתורים לבחירת מגדל וקנייתו, וכל אחד מייצג מגדל אחר שניתן לשים על מסך המשחק, כפי שניתן לראות בדוגמא לעיל.

**מסך ההגדרות (Settings)**



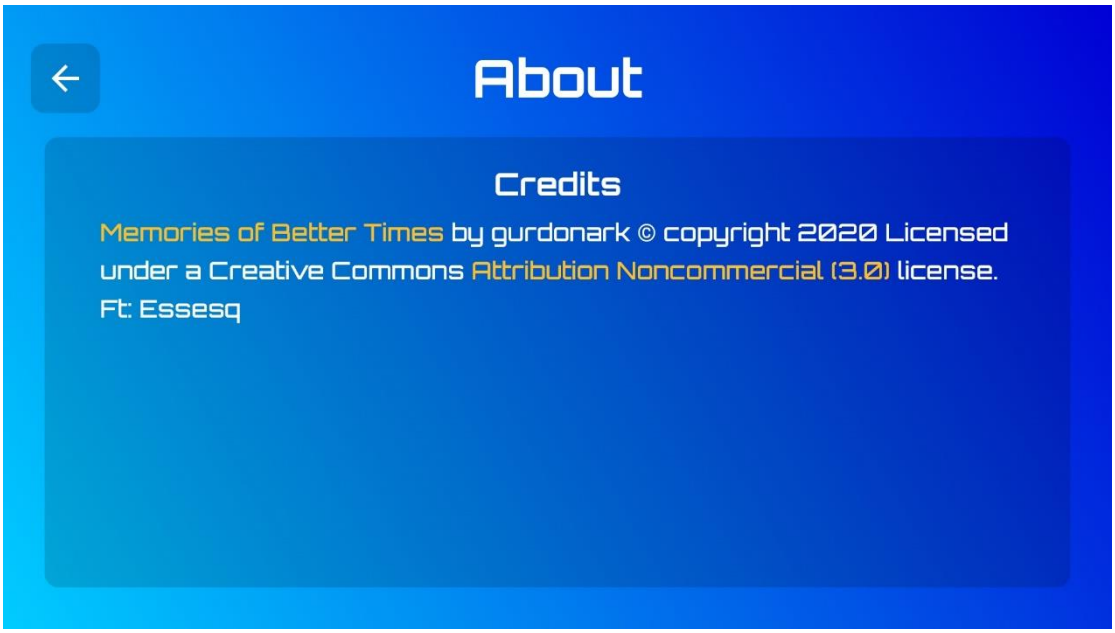

במסך זה תהיינה רשימת הגדרות עליהן יוכל השחקן להחליט. טרם בחרתי בסוג ההגדרות שיהיו במסך.

**מסך ההוראות (Instructions)**




מזך זה משמש להצגת ההוראות עבור השחקן במשחק.

**מסך האודות (About)**




מסך זה משמש לציון קרדיטים (שמירה על זכויות יוצרים) עבור מוזיקת הרקע של האפליקציה.

# בסיס הנתונים

בפרויקט עשיתי שימוש בקובצי JSON לשמירת נתוני האפליקציה, הן במשאבי האפליקציה והן באיחסון הפנימי של האפליקציה (internal storage).

# פירוט הקבצים

## קבצים שנמצאים תחת res/raw (משאבי האפליקציה)

### levels.json

מכיל את כל הנתונים עבור כל שלבי המשחק. למעשה הקובץ מכיל רשימת JSON שמכילה אובייקטים, כאשר כל אחד מהם מתאר למעשה שלב במשחק, אותם ממירים לרשימת אובייקטים מסוג Level באמצעות דסירליזציה שמספקת ספריית Gson. בתוך כל שלב בקובץ יש שדה "map", שמכיל למעשה את נתוני מפת השלב שמומרים אחר כך לאובייקט LevelMap, שמהווה חלק מאובייקט Level. כמו כן, ישנה גם רשימה תחת השדה "atomSequence" שמכיל למעשה את רצף האטומים של השלב, באמצעותו AtomSequencer מתזמן את יצירת האטומים.

### elements.json

מכיל רשימת נתונים של כל סוגי האטומים הנתמכים (כל אובייקט JSON ברשימה מייצג נתונין עבור אטום עם מספר אטומי מסויים). הקובץ למעשה משמש בזמן ריצת האפליקציה לדליית נתוני כל האטומים, על ידי יצירת רשימה מסוג Element (שוב, על ידי שימוש ב-Gson).

### towers.json

מכיל אובייקט שכל אחד משדותיו מייצגים מגדל אחר במשחק, ושמם על פי שם המשחק (למשל, עבור ElectronShooter שם השדה יהיה "electronShooter"). כל אובייקט של מגדל מכיל בתוכו גם אובייקט של הנשק שלו תחת השדה "weapon", כאשר בזמן ריצת האפליקציה נתוני המגדלים מומרים למפה שסוג ערכיה הוא TowerType, כאשר כל אובייקט שכזה מכיל בתוכו גם אובייקט של WeaponType, כפי שנראה בקובץ (כמובן, על ידי שימוש ב-Gson).

## קבצים שנוצרים בזמן ריצה בזיכרון המכשיר (Internal Storage)

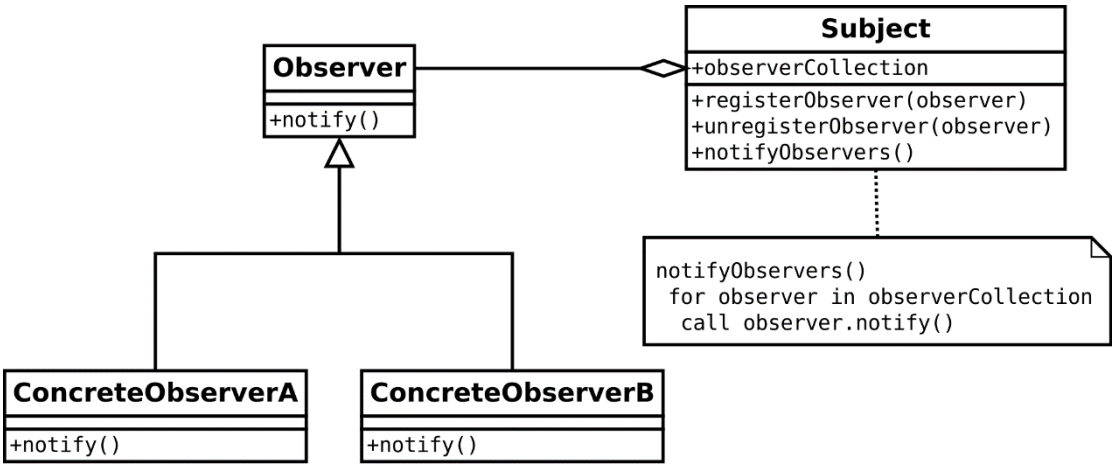
**saved_game_state.json**

זהו קובץ שנוצר בזמן ריצת האפליקציה ונשמר בזיכרון המכשיר הפנימי (internal storage), ומכיל למעשה את המצב בו נעצר המשחק, כאשר שומר ה-GameStateService את ערכי SavedGameState באמצעות סריאליזציה של הספריה Gson. למעשה, ה-GameRepository יוצרת את הקובץ וקוראת אותו בפעולות setSavedGameState() ו-getSavedGameState(), תוך שימוש בפעולות עזר המוגדרות במחלקה Util לקריאה וכתיבה לזיכרון הפנימי של המכשיר.
יש לציין, כי שם הקובץ כתוב בקבוע
GameRepository#SAVED_GAME_STATE_FILENAME, ולכן עלול להשתנות.

# מדריך למפתח

## עקרונות ומושגים

### תבנית הצופה (Observer Design Pattern)




תבנית הצופה (Observer Design Pattern) היא תבנית עיצוב, בה אובייקט "נצפה", הנקרא Subject, מחזיק ברשימה של אובייקטים שתלויים בו (מקושרים אליו), כאשר כל אחד מן האובייקטים המקושרים אליו נקרא "צופה" שלו, Observer.

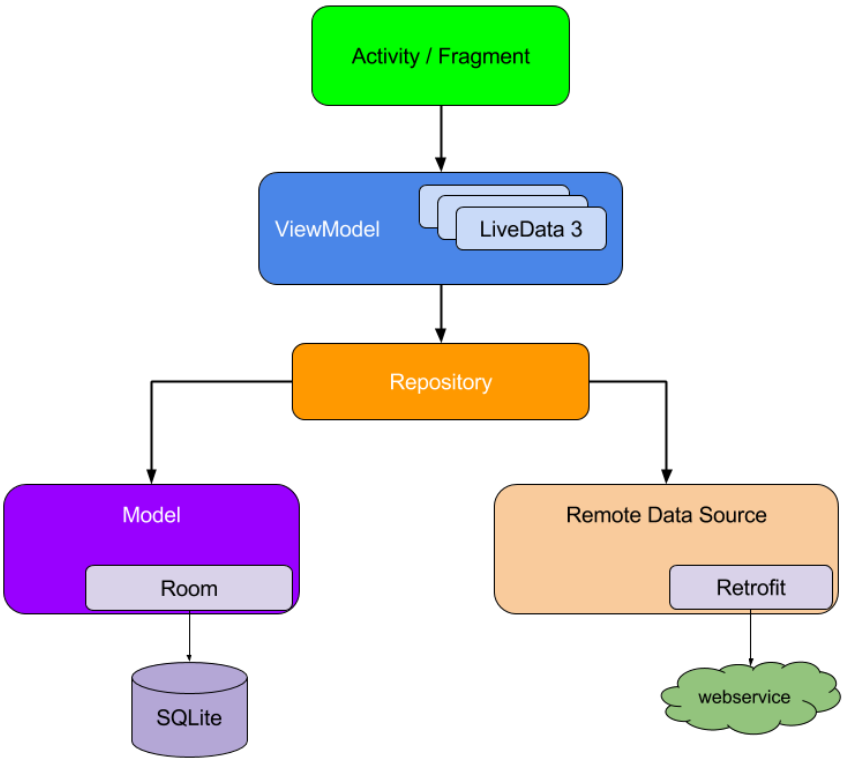
האובייקט "הנצפה" מודיע לצופים עליו על שינויים המתרחשים בו, כמו שינוי ערך פנימי שלו, בדרך כלל על ידי קריאה לאחת מהפעולות שלהם. למשל, בדיאגרמה לעיל ניתן לראות כי לנצפה יש רשימה של אובייקטים, אליה הוא יכול לצרף צופים או להסיר. בכל פעם שמתרחש באובייקט הנצפה שינוי, הוא מודיע לצופיו על השינוי באמצעות הפעולה notifyObservers(), שקוראת לפעולה notify() של כל אחד מהצופים בו.

בתבנית זו ישנו שימוש רחב בספריות וסביבות עבודה רבות, בהן הסביבה של Android, והספרייה RxJava.

### ארכיטקטורת MVVM (על פי הנחיות Android Developers)

ארכיטקטורת MVVM היא ארכיטקטורת האפליקציות המומלצת על פי צוות ה-Android וכן, Google. כתוצאה מכך, Android, במסגרת הפרויקט Android Jetpack, החלו לספק מספר ספריות תחת השם Android Architecture Components שתומכות בארכיטקטורה ומעודדות אותה, ועוזרות לפתח אפליקציות Android קלות להרחבה ופיתוח בעתיד, וכן לשמור על הסדר והמודולריזציה של האפליקציה.

בהתאם לספריות המוצעות בסביבת Android, לארכיטקטורת MVVM באפליקציות Android כיום יש מבנה מאוד מסוים:




- **חשוב!** התמונה לא משקפת באופן מדויק את ארכיטקטורת האפליקציה כרגע, אעדכן אותה בעתיד. זהו רק הסבר כללי על הארכיטקטורה על פי Google.

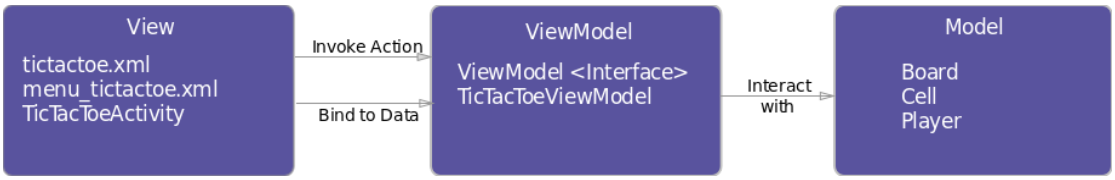
<u>רכיבי הארכיטקטורה</u>

בדומה לארכיטקטורות אפליקציות אחרות, כמו MVC או MVP, ישנם שלושה רכיבים עיקריים בארכיטקטורה הזו:

- **View** – את רכיב זה מייצגים ה-Activity/Fragment. רכיב זה אחראי להציג את הנתונים שמקבל בלבד. כלומר, ב-Activity/Fragment לא תהיה לוגיקה מרובה, ותהיה בעיקר אחראית לממשק המשתמש.
- **Model** – מודל לוגי לנתונים ולקשרים באפליקציה. את רכיב זה מייצגות לרוב מחלקות רגילות, בהן כל התוכן והלוגיקה של האפליקציה, וכן אחראיות על אספקת נתוני האפליקציה. כפי שניתן לראות לעיל, קיים רכיב שנקרא Repository, שאחראי לספק למעשה את כל הנתונים שנשמרים במקור חיצוני, למשל על ידי קריאת קבצים, גישה למסד נתונים, או גישה לאינטרנט. רכיב זה אינו תמיד קיים, ובמקרה של הפרויקט שלי, הוא יתקיים לצד שאר המחלקות הלוגיות.

- **ViewModel** – זהו הרכיב המקשר בין ה-View ל-Model באפליקציה, ומכאן שמו. רכיב זה מכין את כל הנתונים שה-Model מספק להצגה על ממשק המשתמש, ובמילים אחרות, הוא מודל הנתונים של ה-View.

<u>הקשרים שבין רכיבי הארכיטקטורה</u>




- **חשוב!** בעתיד אוסיף דיאגרמה מתאימה יותר למחלקות בפרויקט.

כפי שניתן לראות בדיאגרמה לעיל, ה-ViewModel עוטף בצורה מסוימת ה-View ומקיים אינטראקציה עם ה-Model (קורא לפעולות, קורא נתונים, וכולי). למעשה, ה-View וה-ViewModel מקושרים זה לזה בצורה חד כיוונית – ל-View יש גישה ל-ViewModel, אולם ל-ViewModel אין גישה ל-View. הדבר מונע בעיות הקשורות במחזור החיים של ה-Activity/Fragment, עליהן אדבר בהמשך.

כמו כן, ה-View מודיע ל-ViewModel על אינטראקציות משתמש, שינויים למיניהם, וכדומה, ובכך מאפשר ל-ViewModel לפעול בהתאם (ראה חצים בדיאגרמה לעיל). למשל, על פי תבנית הצופה, כאשר משתנה ערך מסויים ב-Model, ה-ViewModel יתריע על כך לצופה בו, ה-View, על מנת שיעדכן את ממשק המשתמש.

## <u>ארכיטקטורת Single Activity</u>

בשלוש השנים האחרונות, Google וצוות ה-Android החלו לקדם את ארכיטקטורת Single Activity באפליקציות Android. על פי ארכיטקטורה זו, האפליקציה תורכב מ-Activity אחת, ומסכיה השונים ייוצגו על ידי שימוש ב-Fragments. לבחירה זו מספר סיבות, עליהן אדבר בהמשך.

מאחר שהשימוש ב-Fragments יכול להרתיע, ולעיתים מסובך יותר משימוש ב-Activities, צוות ה-Android פיתחו ספריה כחלק מהפרויקט Android Jetpack בשם Navigation Architecture Component, שמקלה על הניווט באפליקציה בין המסכים השונים, מאפשרת חופשיות גדולה יותר במעברים ביניהם, וכן מציגה מספר הבטחות עליהן אדבר בהמשך.

# ספריות

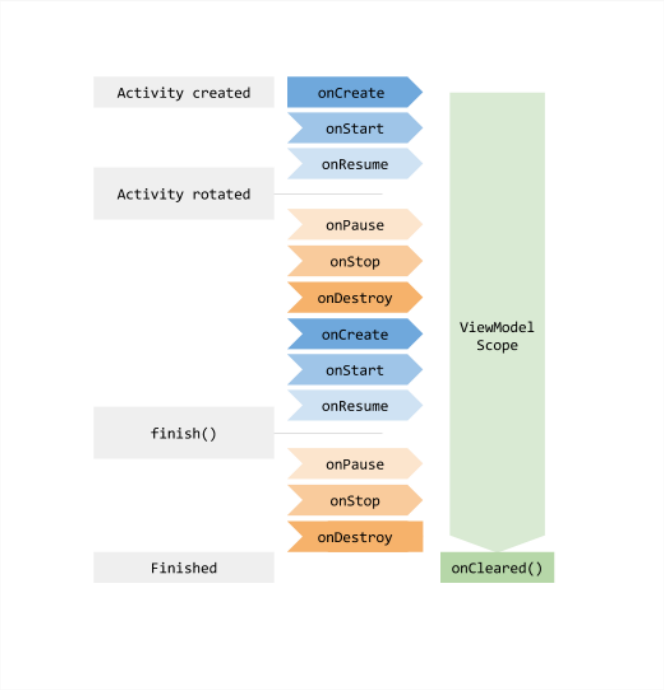
## Lifecycle-Aware Components (Android Jetpack)

ספריה מרכזית מאוד בארכיטקטורת MVVM המוצעת על ידי Google וצוות ה-Android. בספריה זו יש שימוש רחב בתבנית הצופה, תבנית שהחלה להשפיע על חלקים רבים בסביבת העבודה של Android בשנים האחרונות. הספרייה למעשה מכילה רכיבים שונים המודעים ומותאמים למחזור החיים של ה-Activity/Fragment, ומכאן השם Lifecycle-Aware Components.

הספרייה כוללת מספר מחלקות להן שימוש עיקרי בפרויקט, וכן מספר אנוטציות המקלות על ההתאמה למחזור החיים של האפליקציה:

### ViewModel

המחלקה הזו והמחלקות הנלוות אליה מקלות ומקדמות את מימוש ארכיטקטורת ה-MVVM באפליקציה. המחלקה מהווה מחלקת בסיס לכל ה-ViewModels באפליקציה, וכן היא מודעת למחזור החיים של ה-Activity/Fragment אליה היא שייכת. באופן זה היא יכולה למנוע למשל דליפות בזיכרון, על ידי פינוי משאבים לפני הריסתה. מאחר שה-ViewModel אינו תלוי ב-Activity/Fragment, הוא גם אינו תלוי במחזור החיים שלהם, כפי שניתן לראות בדיאגרמה להלן:




כפי שניתן לראות בדיאגרמה לעיל, ה-ViewModel נוצר ברגע שה-Activity/Fragment נוצרים, ואינו מושפע משינויים בתצורתם (Configuration Changes), כמו סיבוב המסך – למרות שה-Activity/Fragment נהרסים ונבנים מחדש כשמסובבים את המסך, ה-ViewModel מודע

למצב ונשאר "בחיים". רק כאשר ה-Activity/Fragment נהרסים לגמרי, למשל לאחר סגירת אפליקציה, גם ה-ViewModel הורס עצמו ומנקה את עצמו מהזיכרון, תוך כדי שקורא לפעולה onCleared(), המאפשרת למתכנת לנקות משאבים למשל. התנהגות זו למעשה פותרת בעיות רבות, שבעבר היו צריכות להיפתר על ידי שחזור ערכים קודמים של ה-Activity באמצעות הפעולה onSaveInstanceState().

LiveData

מתוך התיעוד באתר Android Developers:

" LiveData is an observable data holder class "

כפי שניתן לראות, עצמי LiveData משמשים לאחסון של נתונים, ערך, בו ניתן לצפות. כלומר, על פי תבנית הצופה, אם משתנה הערך שמכיל העצם, הוא מודיע לכך לצופים בו. הדבר מאפשר להגיב לשינויי בערכים בזמן אמת, ועל ידי שימוש בספריית ה-Data Binding, מקל מאוד על שמירת ממשק המשתמש עדכני.

LiveData עובדת יד ביד עם מחלקת ה-ViewModel, ובדומה לה, מודעת גם היא לשינויי התצורה בממש המשתמש. הדבר בא לידי ביטוי שאם למשל מסתובב המסך, ה-LiveData דואג לעדכן את הצופים בו כי יש לעדכן את ערכי המשתנה – הדבר מאוד נוח, אם בונים טיימר למשל.

האנוטציה OnLifecycleEvent

אנוטציה זו משמשת לסימון פעולות שיגיבו (יקראו) בהתאם כאשר ה-Activity/Fragment עוברות אירוע מסוים במחזור החיים שלהן. למשל, ב-GameView, כאשר GameFragment עוברת את האירועים onPause ו-onResume, הפעולות pause() ו-resume() יקראו, בהתאמה. בפעולות אלו ניתן לדאוג לעצור ולהתחיל מחדש את הציור למסך למשל, וכולי.

## Data Binding (Android Jetpack)

ספריה מרכזית נוספת לארכיטקטורת ה-MVVM, אולם היא לא מחייבת ארכיטקטורה מסוימת. כפי ששמה מציע, ספריה זו מאפשרת לקשר בין רכיבי ממשק המשתמש לבין הרכיבים הלוגיים בדרך דקלרטיבית. הדבר נעשה על ידי תוסף (plug-in) שבא יחד עם הספרייה, ומאפשר לקשר בצורה דקלרטיבית בין קובצי ה-layout/*.xml לנתונים ומחלקות באפליקציה. מאחר הקישור נעשה בזמן בניית הפרויקט, אין צורך במציאת Views בזמן ריצה, דבר שמאיץ מעט את האפליקציה. נוסף לכך, הדבר מאפשר גם בדיקה סטטית, לפני ריצת האפליקציה, של ה-Views אליהם ניגשים ולמנוע שגיאות מיותרות.

הדבר המיוחד ביותר בספריה הוא האפשרות לקישור חד/דו כיווני בין רכיבי ממשק המשתמש לחלקים הלוגיים באפליקציה, באופן דקלרטיבי. הדבר תורם מאוד לשמירה על הסדר באפליקציה

ולמניעת קוד Boilerplate, הרי שהרכיבים מגיבים לשינויים ומעדכנים עצמם. הספרייה במיוחד תורמת למימוש ארכיטקטורת ה-MVVM שמקדמת קישוריות בין ה-View ל-ViewModel.

## Navigation Architecture Component (Android Jetpack)

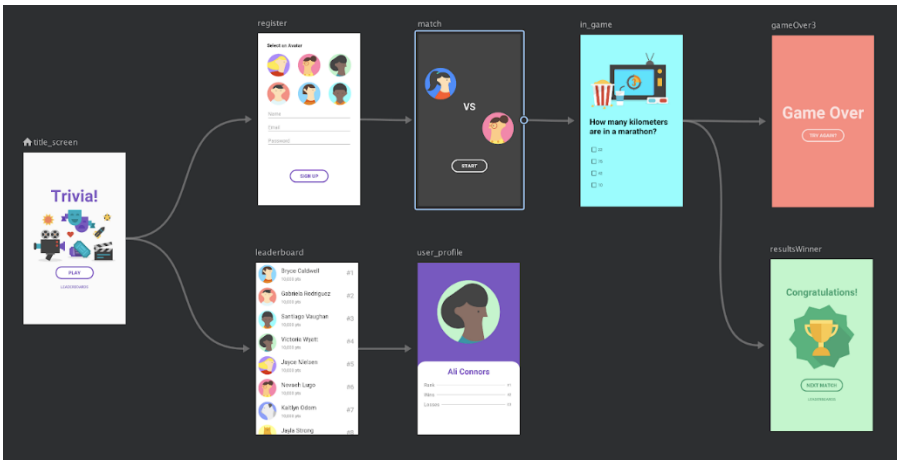
ספריה זו נבנתה עבור ארכיטקטורת ה-Single Activity, ומתייחסת לאינטראקציות שבין חלקי תוכן שונים באפליקציה. במילים אחרות, הספרייה להפשיט את הניווט שבין המסכין השונים באפליקציה, תוך שימוש בארכיטקטורת ה-Single Activity.

הספרייה יכולה לטפל במקרים הכי פשוטים והכי בעיתיים, ולמרות שהיא רחבה, לא נדרש ידע רב על מנת ליצור איתה אפליקציה מרובת מסכים במהירות. אופן הפעולה הכללי של הספרייה מתחלק לשלושה חלקים:

### Navigation Graph

משאב XML שמכיל את כל המידע הקשור לניווט שבין מסכי האפליקציה במקום אחד.

Android Studio מספק גם עורך גרפי, שמקל על התהליך והוויזואליזציה של המסכים השונים והקשרים שביניהם. להלן דוגמא לצילום מסך של העורך הגרפי:




כפי שניתן לראות, המסכים השונים, שנקראים יעדים (destinations), מיוצגים כתרשים זרימה של הניווט באפליקציה, כאשר החצים שביניהם, שנקראים actions, מתארים את כיוון הזרימה שביניהם.

### NavHost

משמש כ"מכל" (container) ריק, המציג את היעדים, המסכים, של ה-navigation graph.

NavHost הוא למעשה Fragment, ולאור השימוש החוזר והזהה בה בין האפליקציות השונות, הספרייה מספקת NavHost ברירת מחדל, בו ניתן להשתמש מיד, מבלי לממש שום מחלקה.

NavController

מחלקה פנימית של Navhost, באמצעותה הוא מנהל את הניווט וההחלפה בין מסכי האפליקציה. הדבר כולל למשל את מימוש התנהגות כפתור החזרה של המכשיר, ומימוש ניווט אחורה ב- Back Stack, על מנת ליצור התנהגות זהה לניווט בין Activities ב-Android.

נוסף לכל, הספרייה מאפשרת העברת פרמטרים בין המסכים השונים, תכונה שנקראת Safe Args, ומבטיחה העברה תקינה של כל הפרמטרים הנדרשים, על ידי בדיקת מספר הפרמטרים שעברו ובדיקת הסוג שלהם בזמן בניית האפליקציה. הדבר מנוגד לשימוש ב-Intents, שעוצרות בתוכן הזדמנויות רבות יותר לשגיאות, אותן פותרת הספרייה באמצעות Safe Args למשל, כפי שצוין קודם לכן.

## RxJava (RxAndroid)

אחת מתוך מספר הספריות שמציעים ReactiveX (קיצור ל-Reactive Extensions) עבור שפות התכנות השונות. הספרייה עבור שפת Java היא RxJava, ומהווה את אחת הספריות הפופולריות ביותר בקרב מפתחי Java. נוסף לספריה זו, ישנה גם הספרייה RxAndroid, המהווה הרחבה והתאמה טובה יותר עבור סביבת העבודה של Android.

הספרייה מאפשרת לשפות תכנות אימפרטיביות, כמו Java, לתפעל על זרמים של נתונים, בין אם הם סינכרוניים או אסינכרוניים. הספרייה למעשה משלבת מספר רעיונות חשובים, בהם תבנית הצופה, תכנות פונקציונלי ותבנית האיטרטור (עליה לא ארחיב במסגרת חוברת הפרויקט), והיא פועלת על בסיס ארעי (event-driven). מתוך האתר של ReactiveX:

" ReactiveX is a combination of the best ideas from the Observer pattern, the Iterator pattern, and functional programming "

מושגים בסיסיים

- Publisher/Observable – מייצג זרם אסינכרוני (Asynchronous Stream) של נתונים, ומשמש כאובייקט הנצפה בתבנית הצופה.
- Observer/Subscriber – מייצג את הצופה באובייקט Observable בתבנית הצופה, ביכולתו לקבל נתונים מן זרם הנתונים שלו.
- Operator (אופרטור) – פעולות ומניפולציות שניתן לבצע על Observables, על מנת לשנות ולמפות את ערכיהם למשל. כל אופרטור מחזיר Observable חדש עם הערכים המתאימים, ולכן ניתן לשרשר אופרטורים אחד אחר השני.

הסבר כללי

באופן הכי בסיסי, Observable פולט ערכים באופן סינכרוני או אסינכרוני. כאשר קוראים לאופרטור (לפעולה) ()subscribe על ה-Observable, נוצר Subscriber חדש, המכיל בתוכו פעולות שונות הקשורות לאופן הגבתו לאירועים שונים של ה-Observable, למשל מה יקרה כאשר נפלט ערך, או מה קורה כשקוראת שגיאה.

באמצעות אופרטורים שונים ניתן לבצע מניפולציות לזרמי הנתונים, ובכך להתאים אותם בקלות יחסית לנתונים הרצויים. כמו כן, ישנה האפשרות להגדיר על איזה Thread יצפה הצופה או על איזה Thread יפלוט ה-Observable את ערכיו ויחשב אותם. הדבר נעשה באמצעות שימוש ב-Scheduler, וזוהי אחת הסיבות לשימוש בהרחבה של RxAndroid, המותאמת ל-Threads הייעודיים לממשק המשתמש (Main Thread/UI Thread, לתהליכי קלט/פלט (IO Thread), וכדומה.

ReactiveX, ובפרט RxJava, מהווים ספריה עשירה מאוד, ולכן קשה לעבור על כל דבר, ואפילו על הבסיס. לכן, אבקש לפנות לאתר של ReactiveX למידע נוסף.

## Gson

ספריית קוד פתוח מבית Google, המשמשת לסריאליזציה/דסריאליזציה (Serialization/Deserialization) של עצמי Java לקבציי JSON, וההפך. סריאליזציה היא תהליך תרגום של מבני נתונים או מצב של אובייקטים, לפורמט שניתן לאחסן אותו בזיכרון הקבוע או להעבירו בין מכשירים, ולשחזר אותו חזרה בשלב מאוחר יותר. באמצעות שימוש באנוטציות ו-Reflection, הספרייה מאפשרת לתרגם עצמי Java בפשטות לטקסט בפורמט JSON, ולשחזר אותו חזרה לאחר מכן.

# תכולת הפרויקט

## קובצי קונפיגורציה

**קובץ ה-Manifest (AndroidManifest.xml)**

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.example.atomictowers">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme"
        tools:ignore="GoogleAppIndexingWarning">

        <activity
            android:name=".MainActivity"
            android:screenOrientation="sensorLandscape"
            tools:ignore="LockedOrientationActivity">
            <intent-filter>
                <action
android:name="android.intent.action.MAIN" />
                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <service

android:name=".data.game.game_state.GameStateService"

android:description="@string/game_state_service_description"
            android:exported="false" />

    </application>

</manifest>
```

## קובצי Gradle

build.gradle של הפרויקט

```
// Top-level build file where you can add configuration
options common to all sub-projects/modules.

buildscript {
    ext {
        lifecycles_version = "2.2.0"
        nav_version = "2.2.1"
        rxandroid_version = "2.1.1"
        rxjava_version = "2.2.16"
        gson_version = "2.8.6"
    }
    repositories {
        google()
        jcenter()

    }
    dependencies {
        classpath 'com.android.tools.build:gradle:4.0.0'
        classpath "androidx.navigation:navigation-safe-args-
gradle-plugin:$nav_version"

        // NOTE: Do not place your application dependencies
here; they belong
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        google()
        jcenter()

    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

build.gradle של מודול האפליקציה (app)

```
apply plugin: 'com.android.application'
apply plugin: "androidx.navigation.safeargs"

android {
    compileSdkVersion 29
    buildToolsVersion "29.0.2"
    defaultConfig {
        applicationId "com.example.atomictowers"
        minSdkVersion 19
        targetSdkVersion 29
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner
"androidx.test.runner.AndroidJUnitRunner"
        vectorDrawables.useSupportLibrary = true
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-
android-optimize.txt'), 'proguard-rules.pro'
        }
    }
    buildFeatures {
        dataBinding = true
    }
    compileOptions {
        sourceCompatibility = 1.8
        targetCompatibility = 1.8
    }
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])

    // AppCompat
    implementation 'androidx.appcompat:appcompat:1.1.0'

    // AndroidX Legacy Support
    implementation 'androidx.legacy:legacy-support-v4:1.0.0'

    // ConstraintLayout
    implementation
'androidx.constraintlayout:constraintlayout:1.1.3'

    // Lifecycles
    implementation "androidx.lifecycle:lifecycle-
extensions:$lifecycles_version"
```

```
    // Navigation Component
    implementation "androidx.navigation:navigation-
fragment:$nav_version"
    implementation "androidx.navigation:navigation-
ui:$nav_version"

    // RxAndroid
    implementation
"io.reactivex.rxjava2:rxandroid:$rxandroid_version"

    // RxJava (RxAndroid releases are few and far between, so
it is recommended to also
    // explicitly depend on RxJava's latest version for bug
fixes and new features).
    implementation
"io.reactivex.rxjava2:rxjava:$rxjava_version"

    // RxLint (RxJava lint)
    lintChecks 'nl.littlerobots.rxlint:detector:1.7.5'

    // Gson
    implementation "com.google.code.gson:gson:$gson_version"

    // Testing (JUnit and Espresso)
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit:1.1.1'
    androidTestImplementation
'androidx.test.espresso:espresso-core:3.2.0'
}
```

## קובצי Java (מחלקות) תחת החבילה (Package) של האפליקציה

**הערה:** לתת-חבילות שנמצאות תחת החבילה של האפליקציה אתייחס בשמן הקצר (אשמיט את שם החבילה של האפליקציה, com.example.atomictowers.

MainActivity.java

```
package com.example.atomictowers;

import android.content.Context;
import android.content.SharedPreferences;
import android.media.MediaPlayer;
import android.os.Bundle;

import androidx.appcompat.app.AppCompatActivity;

/**
 * Serves as a container for a single-activity app, using the
AndroidX Navigation Component.
 */
public class MainActivity extends AppCompatActivity {
    private static final String TAG =
MainActivity.class.getSimpleName();
    public static final float DEFAULT_MUSIC_VOLUME = 0.75f;

    public static MediaPlayer mediaPlayer;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mediaPlayer = MediaPlayer.create(this,
R.raw.memories_of_better_times__gurdonark);
        mediaPlayer.setLooping(true);
        setMusicVolume();
    }

    @Override
    protected void onResume() {
        super.onResume();
        mediaPlayer.start();
    }

    @Override
    protected void onPause() {
        super.onPause();
        mediaPlayer.pause();
    }
```

```
    @Override
    protected void onDestroy() {
        super.onDestroy();
        mediaPlayer.release();
    }

    private void setMusicVolume() {
        SharedPreferences sharedPref =
getPreferences(Context.MODE_PRIVATE);
        float musicVolume = sharedPref.getFloat(
            getString(R.string.key_music_volume),
DEFAULT_MUSIC_VOLUME);
        mediaPlayer.setVolume(musicVolume, musicVolume);
    }
}
```

**components Package**

Game.java

```
package com.example.atomictowers.components;

import android.graphics.Canvas;
import android.graphics.drawable.Drawable;
import android.util.Log;
import android.util.Pair;
import android.util.SparseArray;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.annotation.StringRes;

import com.example.atomictowers.R;
import com.example.atomictowers.components.atoms.Atom;
import
com.example.atomictowers.components.atoms.AtomSequencer;
import
com.example.atomictowers.components.towers.ElectronShooter;
import
com.example.atomictowers.components.towers.PhotonicLaser;
import com.example.atomictowers.components.towers.Tower;
import com.example.atomictowers.data.game.GameRepository;
import com.example.atomictowers.data.game.Level;
import com.example.atomictowers.data.game.LevelMap;
import com.example.atomictowers.data.game.TowerType;
import
com.example.atomictowers.data.game.game_state.AtomSavedState;
import
com.example.atomictowers.data.game.game_state.SavedGameState;
import
com.example.atomictowers.data.game.game_state.TowerSavedState;
import com.example.atomictowers.drawables.LevelMapDrawable;
import com.example.atomictowers.util.Vector2;

import java.lang.reflect.InvocationTargetException;

import io.reactivex.Observable;
import io.reactivex.disposables.CompositeDisposable;
import io.reactivex.subjects.BehaviorSubject;
import io.reactivex.subjects.PublishSubject;

public class Game {
    private static final String TAG =
Game.class.getSimpleName();

    private static final int HORIZONTAL_TILE_COUNT = 8;
```

```
    private static final int VERTICAL_TILE_COUNT = 6;

    /**
     * A multiplier used to scale the damage of the towers,
and the strength of the atoms accordingly.
     */
    public static final int DAMAGE_MULTIPLIER = 100;

    public static final int MAX_HEALTH = 10 *
DAMAGE_MULTIPLIER;
    public static final int INITIAL_ENERGY = 15;

    public static final int ELECTRON_SHOOTER_PRICE = 10;
    public static final int PHOTONIC_LASER_PRICE = 15;

    public final GameRepository gameRepository;
    private final CompositeDisposable mCompositeDisposable =
new CompositeDisposable();

    private BehaviorSubject<Boolean> mGamePausedSubject =
BehaviorSubject.createDefault(false);

    private float mTileSize;
    private Vector2 mDimensions;

    private LevelMap mLevelMap;
    private Drawable mLevelMapDrawable;

    /**
     * Used as a counter for {@link Component} ID's.
     * Should only be used by {@link #generateComponentId()}!
     */
    private volatile int mIdCounter = 0;

    /**
     * Used to index all {@link Component}s on screen.
     * {@link SparseArray} is used instead of {@link
java.util.HashMap} because of IntelliJ warning.
     *
     * @see <a
href="https://stackoverflow.com/questions/25560629/sparsearray
-vs-hashmap">
     * SparseArray vs HashMap (Stack Overflow)</a>
     */
    private final SparseArray<Component> mComponents = new
SparseArray<>();

    private final PublishSubject<Pair<Atom, Vector2>>
mAtomPositions = PublishSubject.create();
```

```
    private AtomSequencer mAtomSequencer;

    private String mSelectedTowerTypeKey;


    private final BehaviorSubject<Integer> mHealth =
BehaviorSubject.createDefault(MAX_HEALTH);

    public int getHealth() {
        return mHealth.getValue();
    }

    public Observable<Integer> getHealthObservable() {
        Log.i(TAG, "energy: " + mHealth.getValue());
        return mHealth.hide();
    }


    private final BehaviorSubject<Integer> mEnergy =
BehaviorSubject.createDefault(INITIAL_ENERGY);

    public int getEnergy() {
        return mEnergy.getValue();
    }

    public Observable<Integer> getEnergyObservable() {
        return mEnergy.hide();
    }


    private final BehaviorSubject<Integer> mGameEnded =
BehaviorSubject.create();

    public Observable<Integer> getGameEndedObservable() {
        return mGameEnded.hide();
    }


    /**
     * Creates and initializes a new {@link Game} object.
     * This constructor should <i>only</i> be used for the
initialization of the {@link Game} object.
     *
     * @param gameRepository A {@link GameRepository} instance
for the game to retrieve game data from.
     * @param dimensions     The dimensions of the game window
     *                       (the dimensions of {@linkplain
com.example.atomictowers.screens.game.GameView GameView}).
```

```
     */
    public Game(@NonNull GameRepository gameRepository,
@NonNull Vector2 dimensions,
                @NonNull SavedGameState savedGameState) {
        this.gameRepository = gameRepository;
        Log.d(TAG, "New Game created");

        mHealth.onNext(savedGameState.health);
        mEnergy.onNext(savedGameState.energy);


mCompositeDisposable.add(gameRepository.getLevel(savedGameStat
e.levelNumber).subscribe(level -> {
            updateDimensions(dimensions);
            mLevelMap = level.map;
            mLevelMapDrawable = new
LevelMapDrawable(mLevelMap);
            mLevelMapDrawable.setBounds(0, 0, (int)
mDimensions.x, (int) mDimensions.y);

            mAtomSequencer = new AtomSequencer(
                this,
                level.elementsAtomicNumbers,
                level.atomSequence,
                savedGameState.numberOfCreatedAtoms);

            initComponents(savedGameState);

            Log.i(TAG, "Game is initialized");
            start();
        }, Throwable::printStackTrace));
    }

    private void initComponents(@NonNull SavedGameState
savedGameState) {
        if (savedGameState.atomSavedStates != null) {
            for (AtomSavedState savedState :
savedGameState.atomSavedStates) {
                addComponent(savedState);
            }
        }
        if (savedGameState.towerSavedStates != null) {
            for (TowerSavedState towerState :
savedGameState.towerSavedStates) {
                addComponent(towerState);
            }
        }
    }
```

```
    /**
     * Called <i>Only</i> when the {@link Game} object is
fully initialized.
     * This is the <i>only</i> starting point of the game.
     */
    private void start() {
        mAtomSequencer.start();
    }

    public void pause() {
        mGamePausedSubject.onNext(true);
        mAtomSequencer.pause();
    }

    public void resume() {
        mGamePausedSubject.onNext(false);
        start();
    }

    public boolean isGamePaused() {
        return mGamePausedSubject.getValue();
    }

    public BehaviorSubject<Boolean> getGamePausedSubject() {
        return mGamePausedSubject;
    }

    /**
     * Updates all game components with a given change in
time.
     *
     * @param timeDiff The change in time in seconds.
     */
    public void update(float timeDiff) {
        for (int i = 0; i < mComponents.size(); i++) {
            int key = mComponents.keyAt(i);
            Component component = mComponents.get(key);
            if (component != null) {
                component.update(timeDiff);
            }
        }
    }

    public void draw(@NonNull Canvas canvas) {
        if (mLevelMapDrawable != null) {
            mLevelMapDrawable.draw(canvas);
        }

        for (int i = 0; i < mComponents.size(); i++) {
```

```
            int key = mComponents.keyAt(i);
            Component component = mComponents.get(key);
            if (component != null) {
                component.draw(canvas);
            }
        }
    }

    public float getTileSize() {
        return mTileSize;
    }

    @NonNull
    public Vector2 getDimensions() {
        return mDimensions;
    }

    @NonNull
    public static Vector2 getDimensionsByScreenHeight(int
height) {
        float tileSize = (float) height / VERTICAL_TILE_COUNT;
        return new Vector2(tileSize * HORIZONTAL_TILE_COUNT,
tileSize * VERTICAL_TILE_COUNT);
    }

    public void updateDimensions(@NonNull Vector2 dimensions)
{
        // The game must have square tiles
        mTileSize = dimensions.y / VERTICAL_TILE_COUNT;
        mDimensions = new Vector2(mTileSize *
HORIZONTAL_TILE_COUNT, mTileSize * VERTICAL_TILE_COUNT);
    }

    public int getLevelNumber() {
        // TODO: Update this method
        return Level.LEVEL_ONE;
    }

    @NonNull
    public LevelMap getMap() {
        return mLevelMap;
    }

    public void selectTowerType(@Nullable String
selectedTowerTypeKey) {
        mSelectedTowerTypeKey = selectedTowerTypeKey;
    }

    public void putTowerOnMap(@NonNull Vector2 tileIndex) {
```

```
        if (mSelectedTowerTypeKey != null &&
mLevelMap.getAtIndex(tileIndex) == LevelMap.TILE_EMPTY) {
            String selectedTowerTypeKey =
mSelectedTowerTypeKey;

            if
(mSelectedTowerTypeKey.equals(TowerType.ELECTRON_SHOOTER_TYPE_
KEY)) {
                decreaseEnergy(ELECTRON_SHOOTER_PRICE);

            } else if
(mSelectedTowerTypeKey.equals(TowerType.PHOTONIC_LASER_TYPE_KE
Y)) {
                decreaseEnergy(PHOTONIC_LASER_PRICE);
            }

            mLevelMap.setAtIndex(tileIndex,
LevelMap.TILE_TOWER);
            mCompositeDisposable.add(

gameRepository.getTowerType(selectedTowerTypeKey)
                    .subscribe(towerType -> {
                        towerType.setTileIndex(tileIndex);

addComponent(convertTowerTypeKeyToClass(selectedTowerTypeKey),
towerType);
                    }, Throwable::printStackTrace));
        }
    }

    private Class<? extends Tower>
convertTowerTypeKeyToClass(@NonNull String towerTypeKey) {
        switch (towerTypeKey) {
            case TowerType.ELECTRON_SHOOTER_TYPE_KEY:
                return ElectronShooter.class;
            case TowerType.PHOTONIC_LASER_TYPE_KEY:
                return PhotonicLaser.class;
            default:
                throw new
IllegalArgumentException("towerTypeKey `" + towerTypeKey + "`
is not a valid");
        }
    }

    @NonNull
    public SparseArray<Component> getComponentsMap() {
        return mComponents.clone();
    }
```

```
    public int addComponent(@NonNull Class<? extends
Component> type, @Nullable Object data) {
        int id = generateComponentId();
        try {
            Component component =
type.getConstructor(Game.class, int.class, Object.class)
                .newInstance(this, id, data);
            mComponents.append(id, component);
            Log.d(TAG, "Created new component with id: " +
id);
        } catch (IllegalAccessException
            | InstantiationException
            | InvocationTargetException
            | NoSuchMethodException e) {
            throw new IllegalArgumentException("Could not
create a new component", e);
        }
        return id;
    }

    public int addComponent(@NonNull AtomSavedState
savedState) {
        int id = generateComponentId();

mCompositeDisposable.add(gameRepository.getElement(savedState.
atomicNumber).subscribe(element -> {
            Component component = new Atom(this, id, element,
savedState);
            mComponents.append(id, component);
            Log.d(TAG, "Created new component with id: " +
id);
        }, e -> {
            throw new IllegalArgumentException("Could not
create a new component", e);
        }));
        return id;
    }

    public int addComponent(@NonNull TowerSavedState
savedState) {
        int id = generateComponentId();

mCompositeDisposable.add(gameRepository.getTowerType(savedStat
e.towerTypeKey).subscribe(towerType -> {
            Class<? extends Tower> type =
convertTowerTypeKeyToClass(savedState.towerTypeKey);
            Component component =
                type.getConstructor(Game.class, int.class,
TowerType.class, TowerSavedState.class)
```

```
                    .newInstance(this, id, towerType,
savedState);
            mComponents.append(id, component);

            mLevelMap.setAtPosition(savedState.position,
mTileSize, LevelMap.TILE_TOWER);
            Log.d(TAG, "Created new component with id: " +
id);
        }, e -> {
            throw new IllegalArgumentException("Could not
create a new component", e);
        }));
        return id;
    }

    /**
     * Generates a unique new {@link Component} ID.
     *
     * @return a unique {@link Component} ID
     */
    private synchronized int generateComponentId() {
        return ++mIdCounter;
    }

    @Nullable
    public Component getComponent(int componentId) {
        return mComponents.get(componentId);
    }

    public void removeComponent(int componentId) {
        mComponents.remove(componentId);
        Log.d(TAG, "removed component with id: " +
componentId);
    }

    public void postAtomPosition(Atom atom, Vector2 position)
{
        mAtomPositions.onNext(new Pair<>(atom, position));
    }

    public void increaseEnergy() {
        mEnergy.onNext(mEnergy.getValue() + 1);
    }

    public void decreaseEnergy(int energyDecrease) {
        mEnergy.onNext(mEnergy.getValue() - energyDecrease);
    }

    public void decreaseHealth(int atomStrength) {
```

```
        mHealth.onNext(mHealth.getValue() - atomStrength);

        if (mHealth.getValue() <= 0) {
            finish(R.string.game_lost_message);
        }
    }

    public Observable<Pair<Atom, Vector2>>
getAtomPositionObservable() {
        return mAtomPositions.hide();
    }

    public int getNumberOfCreatedAtoms() {
        return mAtomSequencer.getNumberOfCreatedAtoms();
    }

    public void finish(@StringRes int
gameEndedMessageStringId) {
        mGameEnded.onNext(gameEndedMessageStringId);
        mCompositeDisposable.dispose();
        mAtomSequencer.destroy();
    }

    public boolean hasFinished() {
        if (mGameEnded.getValue() == null) {
            return false;
        }
        return mGameEnded.getValue() ==
R.string.game_won_message
            || mGameEnded.getValue() ==
R.string.game_lost_message;
    }
}
```

Component.java

```
package com.example.atomictowers.components;

import android.graphics.Canvas;
import android.util.Log;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;

import com.example.atomictowers.util.Vector2;

public abstract class Component {
    private static final String TAG =
Component.class.getSimpleName();

    private final Game mGame;
    private final int mId;

    public Component(@NonNull Game game, int id, @Nullable
Object data) {
        mGame = game;
        mId = id;
    }

    @NonNull
    public Game getGame() {
        return mGame;
    }

    public int getId() {
        return mId;
    }

    @NonNull
    public abstract Vector2 getPosition();

    public abstract void update(float timeDiff);

    public abstract void draw(@NonNull Canvas canvas);

    public void destroy() {
        mGame.removeComponent(mId);
        Log.i(TAG, "destroyed: " + this.toString());
    }
}
```

KineticComponent.java

```
package com.example.atomictowers.components;

import androidx.annotation.NonNull;

import com.example.atomictowers.util.Vector2;

public abstract class KineticComponent extends Component {
    private static final String TAG =
KineticComponent.class.getSimpleName();

    private Vector2 mTarget = Vector2.ZERO;
    private Vector2 mVelocity = Vector2.ZERO;

    public KineticComponent(Game game, int id, Object data) {
        super(game, id, data);
    }

    @NonNull
    public Vector2 getTarget() {
        return mTarget;
    }

    public void setTarget(@NonNull Vector2 target) {
        mTarget = target;
    }

    public boolean isNearTarget(float radius) {
        return mTarget.distance(getPosition()) <= radius;
    }

    @NonNull
    public Vector2 getVelocity() {
        return mVelocity;
    }

    protected void setVelocity(float speed) {
        Vector2 direction =
mTarget.subtract(getPosition()).toUnit();
        mVelocity = direction.scale(speed);
    }

    @Override
    public void destroy() {
        mVelocity = Vector2.ZERO;
        super.destroy();
    }
}
```

**components.atoms Package**

Atom.java

```
package com.example.atomictowers.components.atoms;

import android.annotation.SuppressLint;
import android.graphics.Canvas;
import android.graphics.Color;
import android.util.Log;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;

import com.example.atomictowers.R;
import com.example.atomictowers.components.Game;
import com.example.atomictowers.components.KineticComponent;
import com.example.atomictowers.data.game.Element;
import com.example.atomictowers.data.game.LevelMap;
import
com.example.atomictowers.data.game.game_state.AtomSavedState;
import com.example.atomictowers.drawables.AtomDrawable;
import com.example.atomictowers.util.Vector2;

import io.reactivex.Observable;
import io.reactivex.disposables.CompositeDisposable;
import io.reactivex.subjects.BehaviorSubject;

public class Atom extends KineticComponent {
    private static final String TAG =
Atom.class.getSimpleName();
    private static final float MAX_SPEED = 2f;

    private final CompositeDisposable mCompositeDisposable =
new CompositeDisposable();

    private final BehaviorSubject<String> mName =
BehaviorSubject.createDefault("");
    private final BehaviorSubject<String> mSymbol =
BehaviorSubject.createDefault("");

    /**
     * Defined as the atom's number of protons + number of
neutrons,
     * multiplied by <code>Game.DAMAGE_MULTIPLIER</code>.
     */
    private int mStrength;

    /**
     * The number of protons the atom has, which describe
```

```
which element it is.
     */
    private int mAtomicNumber;

    private LevelMap mMap;
    private int mPathIndex = 0;

    private final BehaviorSubject<Vector2> mPosition =
BehaviorSubject.createDefault(Vector2.ZERO);
    private float mSpeed = 0;

    private AtomDrawable mDrawable;
    private final BehaviorSubject<Integer> mColor =
BehaviorSubject.createDefault(Color.BLACK);
    private final BehaviorSubject<Float> mRadius =
BehaviorSubject.createDefault(0f);

    private final boolean mIsLastAtom;

    public Atom(@NonNull Game game, int id, @Nullable Object
data) {
        super(game, id, data);
        if (!(data instanceof Element)) {
            throw new IllegalArgumentException("`data` is not
of type " + Element.class.getName());
        }
        Element element = (Element) data;

        mAtomicNumber = element.protons;
        mStrength = (element.protons + element.neutrons) *
Game.DAMAGE_MULTIPLIER;

        mMap = getGame().getMap();
        if (mMap.getPath().isEmpty()) {
            throw new IllegalStateException("Map path is
empty");
        }

        initAtomTypeFields(element);
        mIsLastAtom = element.isLastAtom;

        mDrawable = new AtomDrawable(this,
mCompositeDisposable);


mPosition.onNext(getGame().getMap().getStartingPosition(getGam
e()));
        mCompositeDisposable.add(mPosition.subscribe(
            position -> getGame().postAtomPosition(this,
```

```
position),
            Throwable::printStackTrace));

        setTarget(mMap.getPositionFromPath(getGame(),
mPathIndex));
        setVelocity(mSpeed);
    }

    public Atom(@NonNull Game game, int id, @NonNull Element
element, @NonNull AtomSavedState savedState) {
        super(game, id, element);

        mAtomicNumber = element.protons;

        mMap = getGame().getMap();
        if (mMap.getPath().isEmpty()) {
            throw new IllegalStateException("Map path is
empty");
        }

        initAtomTypeFields(element);

        mStrength = savedState.strength;
        mPathIndex = savedState.pathIndex;
        mPosition.onNext(savedState.position);
        mIsLastAtom = savedState.isLastAtom;

        mDrawable = new AtomDrawable(this,
mCompositeDisposable);

        mCompositeDisposable.add(mPosition.subscribe(
            position -> getGame().postAtomPosition(this,
position),
            Throwable::printStackTrace));

        setTarget(mMap.getPositionFromPath(getGame(),
mPathIndex));
        setVelocity(mSpeed);
    }

    private void initAtomTypeFields(@NonNull Element element)
{
        mName.onNext(element.name);
        mSymbol.onNext(element.symbol);
        mColor.onNext(Color.parseColor(element.colorString));
        mRadius.onNext(calculateRadius());

        mSpeed = (MAX_SPEED / mAtomicNumber) *
getGame().getTileSize();
```

```
    }

    private float calculateRadius() {
        float maxRadius = 0.45f * getGame().getTileSize();
        // Radius of the atom should be at most 2/3 of tile
width or height
        return maxRadius - maxRadius / (mAtomicNumber + 2);
    }

    public int getAtomicNumber() {
        return mAtomicNumber;
    }

    public int getStrength() {
        return mStrength;
    }

    public int getPathIndex() {
        return mPathIndex;
    }

    @NonNull
    public Observable<Vector2> getPositionObservable() {
        return mPosition.hide();
    }

    @NonNull
    @Override
    public Vector2 getPosition() {
        return mPosition.getValue();
    }

    @NonNull
    public Observable<Integer> getColorObservable() {
        return mColor.hide();
    }

    @NonNull
    public Observable<Float> getRadiusObservable() {
        return mRadius.hide();
    }

    public float getRadius() {
        return mRadius.getValue();
    }

    @NonNull
    public Observable<String> getSymbolObservable() {
        return mSymbol.hide();
```

```
    }

    @Override
    public void update(float timeDiff) {
        if (mAtomicNumber <= 0) {
            getGame().increaseEnergy();
            destroy();
        } else if (mPosition.getValue().x >
getGame().getDimensions().x) {
            getGame().decreaseHealth(mStrength);
            destroy();
        } else {
            if (isNearTarget(getVelocity().magnitude() * (0.1f
/ MAX_SPEED))) {
                mPathIndex++;

                if (mPathIndex < mMap.getPath().size()) {

setTarget(mMap.getPositionFromPath(getGame(), mPathIndex));
                } else if (mPathIndex ==
mMap.getPath().size()) {

setTarget(mMap.getEndingPosition(getGame()));
                } else {
                    destroy();
                }
            }

            setVelocity(mSpeed);

mPosition.onNext(mPosition.getValue().add(getVelocity().scale(
timeDiff)));
        }
    }

    @Override
    public void draw(@NonNull Canvas canvas) {
        mDrawable.draw(canvas);
    }

    public void applyDamage(float damage) {
        mStrength -= damage;

        if (mAtomicNumber <= 0) {
            destroy();
        } else if (mStrength <= (mAtomicNumber - 1) * 2 *
Game.DAMAGE_MULTIPLIER) {
            // If the strength is lower than or equal to the
minimum strength
```

```
            // for this atomic number, lower the atomic number
            changeElement();
        }
    }

    public boolean isLastAtom() {
        return mIsLastAtom;
    }

    @Override
    public void destroy() {
        mSpeed = 0;
        mCompositeDisposable.dispose();

        if (mIsLastAtom) {
            if (!getGame().hasFinished()) {
                getGame().finish(R.string.game_won_message);
            }
            Log.i(TAG, "Game ended - last atom destroyed!");
        }

        super.destroy();
    }

    public boolean isDestroyed() {
        return mAtomicNumber <= 0;
    }

    private void changeElement() {
        mAtomicNumber--;
        getGame().increaseEnergy();

mCompositeDisposable.add(getGame().gameRepository.getElement(m
AtomicNumber)
            .subscribe(this::initAtomTypeFields,
Throwable::printStackTrace));
    }

    @SuppressLint("DefaultLocale")
    @NonNull
    @Override
    public String toString() {
        return String.format(
            "%s atom { id: %d, strength: %d, atomicNumber: %d,
position: %s, velocity: %s }",
            mSymbol.getValue(),
            getId(),
            mStrength,
            mAtomicNumber,
```

```
            getPosition(),
            getVelocity());
    }
}
```

AtomSequencer.java

```
package com.example.atomictowers.components.atoms;

import android.util.Log;
import android.util.SparseArray;

import androidx.annotation.NonNull;

import com.example.atomictowers.components.Game;
import com.example.atomictowers.data.game.Element;
import com.google.gson.annotations.SerializedName;

import java.util.List;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.atomic.AtomicInteger;

import io.reactivex.Observable;
import io.reactivex.android.schedulers.AndroidSchedulers;
import io.reactivex.disposables.Disposable;
import io.reactivex.schedulers.Schedulers;

public class AtomSequencer {
    private static final String TAG =
AtomSequencer.class.getSimpleName();

    public static class AtomSequenceItem {
        @SerializedName("atomicNumber")
        public int atomicNumber;

        @SerializedName("delay")
        public int delay;
    }

    private final Game mGame;
    private final List<AtomSequenceItem> mSequence;
    private final SparseArray<Element> mElements = new
SparseArray<>();
    private final AtomicInteger mNumberOfCreatedAtoms = new
AtomicInteger(0);

    private Disposable mInitElementsDisposable;
    private Disposable mSequencerDisposable;

    public AtomSequencer(@NonNull Game game, @NonNull
List<Integer> elementsAtomicNumbers,
                         @NonNull List<AtomSequenceItem>
sequence, int numberOfCreatedAtoms) {
        mGame = game;
        mSequence = sequence;
```

```
        mInitElementsDisposable =
game.gameRepository.getElements().subscribe(elements -> {
            for (int atomicNumber : elementsAtomicNumbers) {
                mElements.append(atomicNumber,
elements.get(atomicNumber));
            }
        }, Throwable::printStackTrace);
        mNumberOfCreatedAtoms.set(numberOfCreatedAtoms);
    }

    public int getNumberOfCreatedAtoms() {
        return mNumberOfCreatedAtoms.get();
    }

    public void start() {
        mSequencerDisposable =
Observable.fromIterable(mSequence)
            .skip(mNumberOfCreatedAtoms.get())
            .concatMap(item ->
Observable.just(item).delay(item.delay,
TimeUnit.MILLISECONDS))
            .subscribeOn(Schedulers.single())
            .observeOn(AndroidSchedulers.mainThread())
            .subscribe(item -> {
                Element element =
mElements.get(item.atomicNumber);
                if (mNumberOfCreatedAtoms.get() + 1 >=
mSequence.size()) {
                    element.isLastAtom = true;
                }
                mGame.addComponent(Atom.class, element);
                mNumberOfCreatedAtoms.incrementAndGet();
            }, Throwable::printStackTrace, () -> Log.i(TAG,
"All atoms created"));
    }

    public void pause() {
        if (mSequencerDisposable != null &&
!mSequencerDisposable.isDisposed()) {
            mSequencerDisposable.dispose();
        }
    }

    public void destroy() {
        if (mInitElementsDisposable != null &&
!mInitElementsDisposable.isDisposed()) {
            mInitElementsDisposable.dispose();
        }
        if (mSequencerDisposable != null &&
```

```
!mSequencerDisposable.isDisposed()) {
            mSequencerDisposable.dispose();
        }
    }
}
```

**components.towers Package**

Tower.java

```
package com.example.atomictowers.components.towers;

import android.graphics.Canvas;
import android.graphics.drawable.Drawable;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;

import com.example.atomictowers.components.Component;
import com.example.atomictowers.components.Game;
import com.example.atomictowers.components.atoms.Atom;
import com.example.atomictowers.data.game.TowerType;
import com.example.atomictowers.data.game.WeaponType;
import
com.example.atomictowers.data.game.game_state.TowerSavedState;
import com.example.atomictowers.util.Vector2;

public abstract class Tower extends Component {
    private static final String TAG =
Tower.class.getSimpleName();

    /**
     * Top-left corner of the tile
     */
    protected Vector2 mPosition;
    protected float mRange;
    protected long mShootInterval;
    protected Atom mTarget;
    protected WeaponType mWeaponType;
    protected Drawable mDrawable;

    public Tower(@NonNull Game game, int id, @Nullable Object
data) {
        super(game, id, data);

        if (!(data instanceof TowerType)) {
            throw new IllegalArgumentException("`data` is not
of type " + Vector2.class.getName());
        }
        TowerType type = (TowerType) data;

        float tileSize = game.getTileSize();

        mRange = type.getRange(tileSize);
        mShootInterval = type.shootInterval;
        mWeaponType = type.weaponType;
```

```
        initPosition(type.getTileIndex());
        mWeaponType.setStartingPosition(
            mPosition.add(new Vector2(tileSize * 0.5f,
tileSize * 0.5f)));
        initDrawable();
    }

    private void initPosition(@NonNull Vector2 tileIndex) {
        float x = tileIndex.x * getGame().getTileSize();
        float y = tileIndex.y * getGame().getTileSize();
        mPosition = new Vector2(x, y);
    }

    public Tower(@NonNull Game game, int id, @NonNull
TowerType towerType, @NonNull TowerSavedState savedState) {
        super(game, id, towerType);

        float tileSize = game.getTileSize();

        mRange = towerType.getRange(tileSize);
        mShootInterval = towerType.shootInterval;
        mWeaponType = towerType.weaponType;

        mPosition = savedState.position;

        mWeaponType.setStartingPosition(
            mPosition.add(new Vector2(tileSize * 0.5f,
tileSize * 0.5f)));
        initDrawable();
    }

    protected abstract void initDrawable();

    @NonNull
    public abstract String getTowerTypeKey();

    @Override
    public void draw(@NonNull Canvas canvas) {
        mDrawable.draw(canvas);
    }

    @NonNull
    @Override
    public Vector2 getPosition() {
        return mPosition;
    }
}
```

ElectronShooter.java

```
package com.example.atomictowers.components.towers;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;

import com.example.atomictowers.R;
import com.example.atomictowers.components.Game;
import com.example.atomictowers.components.atoms.Atom;
import
com.example.atomictowers.components.towers.weapons.ElectronPro
jectile;
import com.example.atomictowers.data.game.TowerType;
import
com.example.atomictowers.data.game.game_state.TowerSavedState;
import com.example.atomictowers.util.Vector2;

import java.util.concurrent.TimeUnit;

import io.reactivex.Observable;
import io.reactivex.disposables.Disposable;
import io.reactivex.schedulers.Schedulers;

public class ElectronShooter extends Tower {
    private static final String TAG =
ElectronShooter.class.getSimpleName();

    private Disposable mAtomPositionObservableSubscription;
    private Disposable mShooterSubscription;
    private Disposable mGamePauseSubscription;

    public ElectronShooter(@NonNull Game game, int id,
@Nullable Object data) {
        super(game, id, data);
        setGamePauseListener();
        setAtomRadar();
    }

    public ElectronShooter(@NonNull Game game, int id,
@NonNull TowerType towerType,
                           @NonNull TowerSavedState
savedState) {
        super(game, id, towerType, savedState);
        setGamePauseListener();
        setAtomRadar();
    }

    @Override
    protected void initDrawable() {
```

```
        int tileSize = (int) getGame().getTileSize();
        int topLeftCornerX = (int) mPosition.x;
        int topLeftCornerY = (int) mPosition.y;

        mDrawable =
getGame().gameRepository.getDrawableFromResources(R.drawable.e
lectron_shooter);
        mDrawable.setBounds(topLeftCornerX, topLeftCornerY,
            topLeftCornerX + tileSize, topLeftCornerY +
tileSize);
    }

    @NonNull
    @Override
    public String getTowerTypeKey() {
        return TowerType.ELECTRON_SHOOTER_TYPE_KEY;
    }

    private void setGamePauseListener() {
        mGamePauseSubscription =
getGame().getGamePausedSubject()
            .filter(isPaused -> isPaused)
            .subscribe(isPaused -> {
                if (mShooterSubscription != null &&
!mShooterSubscription.isDisposed()) {
                    mShooterSubscription.dispose();
                }
            }, Throwable::printStackTrace);
    }

    private void setAtomRadar() {
        mAtomPositionObservableSubscription =
getGame().getAtomPositionObservable()
            .filter(pair -> {
                Vector2 atomPosition = pair.second;
                return mPosition.distance(atomPosition) <=
mRange;
            })
            .subscribeOn(Schedulers.computation())
            .subscribe(pair -> {
                Atom targetAtom = pair.first;
                if (mTarget == null) {
                    mTarget = targetAtom;
                    setShooter();
                    if
(!mAtomPositionObservableSubscription.isDisposed()) {

mAtomPositionObservableSubscription.dispose();
                    }
```

```
                }
            }, Throwable::printStackTrace);
    }

    private void setShooter() {
        mShooterSubscription = Observable.interval(
            mShootInterval, TimeUnit.MILLISECONDS,
Schedulers.computation())
            .subscribe(l -> {
                mWeaponType.setTargetAtom(mTarget);

getGame().addComponent(ElectronProjectile.class, mWeaponType);
            }, Throwable::printStackTrace);
    }

    @Override
    public void update(float timeDiff) {
        if (mTarget != null) {
            if (mTarget.isDestroyed() ||
mPosition.distance(mTarget.getPosition()) > mRange) {
                mTarget = null;
                if (!mShooterSubscription.isDisposed()) {
                    mShooterSubscription.dispose();
                }
                setAtomRadar();
            } else if (mShooterSubscription.isDisposed() &&
!getGame().getGamePausedSubject().getValue()) {
                setShooter();
            }
        }
    }

    @Override
    public void destroy() {
        if (!mGamePauseSubscription.isDisposed()) {
            mGamePauseSubscription.dispose();
        }
        if (!mShooterSubscription.isDisposed()) {
            mShooterSubscription.dispose();
        }
        if (!mAtomPositionObservableSubscription.isDisposed())
{
            mAtomPositionObservableSubscription.dispose();
        }
        super.destroy();
    }
}
```

PhotonicLaser.java

```
package com.example.atomictowers.components.towers;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;

import com.example.atomictowers.R;
import com.example.atomictowers.components.Game;
import com.example.atomictowers.components.atoms.Atom;
import
com.example.atomictowers.components.towers.weapons.PhotonBeam;
import com.example.atomictowers.data.game.TowerType;
import
com.example.atomictowers.data.game.game_state.TowerSavedState;
import com.example.atomictowers.util.Vector2;

import io.reactivex.disposables.Disposable;
import io.reactivex.schedulers.Schedulers;

public class PhotonicLaser extends Tower {
    private int mWeaponComponentId;
    private Disposable mAtomPositionObservableSubscription;

    public PhotonicLaser(@NonNull Game game, int id, @Nullable
Object data) {
        super(game, id, data);
        setAtomRadar();
    }

    public PhotonicLaser(@NonNull Game game, int id, @NonNull
TowerType towerType, @NonNull TowerSavedState savedState) {
        super(game, id, towerType, savedState);
        setAtomRadar();
    }

    @Override
    protected void initDrawable() {
        int tileSize = (int) getGame().getTileSize();
        int topLeftCornerX = (int) mPosition.x;
        int topLeftCornerY = (int) mPosition.y;

        mDrawable =
getGame().gameRepository.getDrawableFromResources(R.drawable.p
hotonic_laser);
        mDrawable.setBounds(topLeftCornerX, topLeftCornerY,
            topLeftCornerX + tileSize, topLeftCornerY +
tileSize);
    }
```

```
    private void setAtomRadar() {
        mAtomPositionObservableSubscription =
getGame().getAtomPositionObservable()
            .filter(pair -> {
                Vector2 atomPosition = pair.second;
                return mPosition.distance(atomPosition) <=
mRange;
            })
            .subscribeOn(Schedulers.computation())
            .subscribe(pair -> {
                Atom targetAtom = pair.first;
                if (mTarget == null) {
                    mTarget = targetAtom;
                    setBeam();
                    if
(!mAtomPositionObservableSubscription.isDisposed()) {

mAtomPositionObservableSubscription.dispose();
                    }
                }
            }, Throwable::printStackTrace);
    }

    private void setBeam() {
        mWeaponType.setTargetAtom(mTarget);
        mWeaponComponentId =
getGame().addComponent(PhotonBeam.class, mWeaponType);
    }

    @Override
    public void update(float timeDiff) {
        if (mTarget != null) {
            if (mTarget.isDestroyed() ||
mPosition.distance(mTarget.getPosition()) > mRange) {
                mTarget = null;
                getGame().removeComponent(mWeaponComponentId);
                mWeaponComponentId = 0;
                setAtomRadar();
            }
        }
    }

    @NonNull
    @Override
    public String getTowerTypeKey() {
        return TowerType.PHOTONIC_LASER_TYPE_KEY;
    }

    @Override
```

```
    public void destroy() {
        if (!mAtomPositionObservableSubscription.isDisposed())
{
            mAtomPositionObservableSubscription.dispose();
        }
        super.destroy();
    }
}
```

**components.towers.weapons Package**

Weapon.java

```
package com.example.atomictowers.components.towers.weapons;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;

import com.example.atomictowers.components.Component;
import com.example.atomictowers.components.Game;
import com.example.atomictowers.components.atoms.Atom;
import com.example.atomictowers.data.game.WeaponType;
import com.example.atomictowers.util.Vector2;

public abstract class Weapon extends Component {
    private static final String TAG =
KineticWeapon.class.getSimpleName();

    private Atom mTargetAtom;

    private Vector2 mPosition;

    /**
     * Damage per millisecond.
     */
    private final float mDamage;

    public Weapon(@NonNull Game game, int id, @Nullable Object
data) {
        super(game, id, data);

        if (!(data instanceof WeaponType)) {
            throw new IllegalArgumentException("`data` is not
of type " + WeaponType.class.getName());
        }
        WeaponType type = (WeaponType) data;

        mPosition = type.getStartingPosition();
        mDamage = type.getDamage();
        setTarget(type.getTargetAtom());
    }

    @NonNull
    @Override
    public Vector2 getPosition() {
        return mPosition;
    }

    public void setPosition(@NonNull Vector2 position) {
```

```
        mPosition = position;
    }

    protected float getDamage() {
        return mDamage;
    }

    @Override
    public void update(float timeDiff) {
        damage(mTargetAtom, timeDiff);
    }

    public void setTarget(@NonNull Atom atom) {
        mTargetAtom = atom;
    }

    @NonNull
    public Atom getTarget() {
        return mTargetAtom;
    }

    protected void damage(@NonNull Atom atom, float timeDiff)
{
        atom.applyDamage(mDamage * timeDiff);
    }
}
```

KineticWeapon.java

```
package com.example.atomictowers.components.towers.weapons;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;

import com.example.atomictowers.components.Game;
import com.example.atomictowers.components.KineticComponent;
import com.example.atomictowers.components.atoms.Atom;
import com.example.atomictowers.data.game.WeaponType;
import com.example.atomictowers.util.Vector2;

import io.reactivex.disposables.Disposable;

public abstract class KineticWeapon extends KineticComponent {
    private static final String TAG =
KineticWeapon.class.getSimpleName();

    private Disposable mTargetAtomSubscription;

    private Atom mTargetAtom;

    private Vector2 mPosition;
    private float mSpeed;

    /**
     * Damage per hit.
     */
    private final float mDamage;

    public KineticWeapon(@NonNull Game game, int id, @Nullable
Object data) {
        super(game, id, data);

        if (!(data instanceof WeaponType)) {
            throw new IllegalArgumentException("`data` is not
of type " + WeaponType.class.getName());
        }
        WeaponType type = (WeaponType) data;

        mPosition = type.getStartingPosition();
        mSpeed = type.speed * getGame().getTileSize();
        mDamage = type.getDamage();
        setTarget(type.getTargetAtom());
    }

    @NonNull
    @Override
    public Vector2 getPosition() {
```

```
        return mPosition;
    }

    public void setPosition(@NonNull Vector2 position) {
        mPosition = position;
    }

    protected float getDamage() {
        return mDamage;
    }

    @Override
    public void update(float timeDiff) {
        if (isNearTarget(mTargetAtom.getRadius())) {
            damage(mTargetAtom);
        }

        Vector2 gameDimensions = getGame().getDimensions();
        if (getPosition().x < 0 || getPosition().x >
gameDimensions.x ||
            getPosition().y < 0 || getPosition().y >
gameDimensions.y) {
            destroy();
        }

        setVelocity(mSpeed);
        mPosition =
mPosition.add(getVelocity().scale(timeDiff));
    }

    public void setTarget(@NonNull Atom atom) {
        mTargetAtom = atom;

        // Listen to the target atom's position, and update
the target position
        // every time it changes.
        mTargetAtomSubscription = atom.getPositionObservable()
            .subscribe(this::setTarget,
Throwable::printStackTrace);
    }

    protected void damage(@NonNull Atom atom) {
        atom.applyDamage(mDamage);
    }

    @Override
    public void destroy() {
        mSpeed = 0;
        mTargetAtomSubscription.dispose();
```

```
        super.destroy();
    }
}
```

ElectronProjectile.java

```
package com.example.atomictowers.components.towers.weapons;

import android.annotation.SuppressLint;
import android.graphics.Canvas;

import androidx.annotation.NonNull;

import com.example.atomictowers.components.Game;
import com.example.atomictowers.components.atoms.Atom;
import
com.example.atomictowers.drawables.weapons.ElectronProjectileD
rawable;

public class ElectronProjectile extends KineticWeapon {
    private static final String TAG =
ElectronProjectile.class.getSimpleName();

    private ElectronProjectileDrawable mDrawable;

    public ElectronProjectile(Game game, int id, Object data)
{
        super(game, id, data);
        mDrawable = new ElectronProjectileDrawable(this,
game.getTileSize());
    }

    @Override
    public void update(float timeDiff) {
        super.update(timeDiff);
        // In order to fix bug where electrons would get stuck
on the game screen because
        // of not hitting the target
        if (getVelocity().magnitude() < 0.01) {
            destroy();
        }
    }

    @Override
    public void draw(@NonNull Canvas canvas) {
        mDrawable.draw(canvas);
    }

    @Override
    protected void damage(@NonNull Atom atom) {
        super.damage(atom);
        destroy();
    }
```

```
    @SuppressLint("DefaultLocale")
    @NonNull
    @Override
    public String toString() {
        return String.format(
            "electron projectile { id: %d, damage: %.2f,
position: %s, velocity: %s, target: %s }",
            getId(),
            getDamage(),
            getPosition(),
            getVelocity(),
            getTarget());
    }
}
```

PhotonBeam.java

```
package com.example.atomictowers.components.towers.weapons;

import android.annotation.SuppressLint;
import android.graphics.Canvas;

import androidx.annotation.NonNull;

import com.example.atomictowers.components.Game;
import com.example.atomictowers.components.atoms.Atom;
import
com.example.atomictowers.drawables.weapons.ElectronProjectileD
rawable;

public class ElectronProjectile extends KineticWeapon {
    private static final String TAG =
ElectronProjectile.class.getSimpleName();

    private ElectronProjectileDrawable mDrawable;

    public ElectronProjectile(Game game, int id, Object data)
{
        super(game, id, data);
        mDrawable = new ElectronProjectileDrawable(this,
game.getTileSize());
    }

    @Override
    public void update(float timeDiff) {
        super.update(timeDiff);
        // In order to fix bug where electrons would get stuck
on the game screen because
        // of not hitting the target
        if (getVelocity().magnitude() < 0.01) {
            destroy();
        }
    }

    @Override
    public void draw(@NonNull Canvas canvas) {
        mDrawable.draw(canvas);
    }

    @Override
    protected void damage(@NonNull Atom atom) {
        super.damage(atom);
        destroy();
    }
```

```
    @SuppressLint("DefaultLocale")
    @NonNull
    @Override
    public String toString() {
        return String.format(
            "electron projectile { id: %d, damage: %.2f,
position: %s, velocity: %s, target: %s }",
            getId(),
            getDamage(),
            getPosition(),
            getVelocity(),
            getTarget());
    }
}
```

**screens.main Package**

MainFragment.java

```
package com.example.atomictowers.screens.main;

import android.app.AlertDialog;
import android.app.Dialog;
import android.graphics.drawable.ColorDrawable;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.MenuInflater;
import android.view.View;
import android.view.ViewGroup;
import android.view.Window;
import android.widget.PopupMenu;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.databinding.DataBindingUtil;
import androidx.fragment.app.Fragment;
import androidx.navigation.fragment.NavHostFragment;

import com.example.atomictowers.R;
import com.example.atomictowers.data.game.GameRepository;
import com.example.atomictowers.data.game.Level;
import
com.example.atomictowers.data.game.game_state.SavedGameState;
import
com.example.atomictowers.databinding.FragmentMainBinding;
import
com.example.atomictowers.screens.main.MainFragmentDirections.A
ctionMainFragmentToGameFragment;

import java.util.Objects;

import io.reactivex.disposables.CompositeDisposable;

/**
 * The game's main menu.
 */
public class MainFragment extends Fragment {
    private static final String TAG =
MainFragment.class.getSimpleName();
    private final CompositeDisposable mCompositeDisposable =
new CompositeDisposable();

    @Nullable
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater,
```

```
@Nullable ViewGroup container,
                                @Nullable Bundle
savedInstanceState) {
        FragmentMainBinding mBinding =
DataBindingUtil.inflate(
            inflater, R.layout.fragment_main, container,
false);
        mBinding.setLifecycleOwner(this);

        mBinding.startButton.setOnClickListener(view -> {
            GameRepository repository =
GameRepository.getInstance(

Objects.requireNonNull(getActivity()).getApplicationContext())
;


mCompositeDisposable.add(repository.hasSavedGameState().subscr
ibe(hasSavedGameState -> {
                if (hasSavedGameState) {

mCompositeDisposable.add(repository.getSavedGameState().subscr
ibe(
                        this::showResumeLastGameDialog,
                        Throwable::printStackTrace));
                } else {
                    ActionMainFragmentToGameFragment action =

MainFragmentDirections.actionMainFragmentToGameFragment(
                            new
SavedGameState(Level.LEVEL_ONE)); // TODO: Update this!

NavHostFragment.findNavController(this).navigate(action);
                }
            }, Throwable::printStackTrace));
        });


mBinding.menuButton.setOnClickListener(this::showPopupMenu);

        return mBinding.getRoot();
    }

    private void showPopupMenu(View view) {
        PopupMenu popup = new PopupMenu(getContext(), view);

        MenuInflater inflater = popup.getMenuInflater();
        inflater.inflate(R.menu.menu_main, popup.getMenu());
```

```
        popup.setOnMenuItemClickListener(menuItem -> {
            switch (menuItem.getItemId()) {
                case R.id.settings:
                    NavHostFragment.findNavController(this)

.navigate(R.id.action_mainFragment_to_settingsFragment);
                    return true;
                case R.id.instructions:
                    NavHostFragment.findNavController(this)

.navigate(R.id.action_mainFragment_to_instructionsFragment);
                    return true;
                case R.id.about:
                    NavHostFragment.findNavController(this)

.navigate(R.id.action_mainFragment_to_aboutFragment);
                    return true;
                default:
                    return false;
            }
        });

        popup.show();
    }

    private void showResumeLastGameDialog(@NonNull
SavedGameState savedGameState) {
        AlertDialog.Builder builder = new
AlertDialog.Builder(getContext());

        LayoutInflater inflater =
requireActivity().getLayoutInflater();
        View layout =
inflater.inflate(R.layout.dialog_resume_last_game, null);

        builder.setView(layout);

        Dialog dialog = builder.create();
        Window dialogWindow = dialog.getWindow();
        if (dialogWindow != null) {
            dialogWindow.setBackgroundDrawable(new
ColorDrawable(android.graphics.Color.TRANSPARENT));
        }


layout.findViewById(R.id.resume_button).setOnClickListener(vie
w -> {
            ActionMainFragmentToGameFragment action =
```

```
MainFragmentDirections.actionMainFragmentToGameFragment(savedG
ameState);

NavHostFragment.findNavController(this).navigate(action);
            dialog.dismiss();
        });


layout.findViewById(R.id.new_game_button).setOnClickListener(v
iew -> {
            ActionMainFragmentToGameFragment action =

MainFragmentDirections.actionMainFragmentToGameFragment(
                    new SavedGameState(Level.LEVEL_ONE)); //
TODO: Update this!

NavHostFragment.findNavController(this).navigate(action);
            dialog.dismiss();
        });

        dialog.show();
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        if (!mCompositeDisposable.isDisposed()) {
            mCompositeDisposable.dispose();
        }
    }
}
```

**screens.game Package**

GameFragment.java

```
package com.example.atomictowers.screens.game;

import android.app.AlertDialog;
import android.app.Dialog;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.graphics.drawable.ColorDrawable;
import android.media.AudioManager;
import android.os.Bundle;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.view.ViewTreeObserver;
import android.view.Window;
import android.widget.TextView;

import androidx.activity.OnBackPressedCallback;
import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.annotation.StringRes;
import androidx.databinding.DataBindingUtil;
import androidx.fragment.app.Fragment;
import androidx.lifecycle.ViewModelProvider;
import androidx.navigation.fragment.NavHostFragment;

import com.example.atomictowers.MainActivity;
import com.example.atomictowers.R;
import com.example.atomictowers.components.Game;
import com.example.atomictowers.data.game.GameRepository;
import
com.example.atomictowers.data.game.game_state.SavedGameState;
import
com.example.atomictowers.databinding.FragmentGameBinding;
import com.example.atomictowers.util.Vector2;

import java.util.Objects;

import io.reactivex.disposables.Disposable;

/**
 * Container for the game
 */
public class GameFragment extends Fragment {
```

```
    private static final String TAG =
GameFragment.class.getSimpleName();

    private GameViewModel mViewModel;
    private FragmentGameBinding mBinding;
    private Disposable mGamePausedSubscription;

    private BroadcastReceiver
mUnpluggedHeadsetBroadcastReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent)
{
            if
(AudioManager.ACTION_AUDIO_BECOMING_NOISY.equals(intent.getAct
ion())) {
                Log.i(TAG, "Received broadcast!");
                mBinding.gameView.pause();
            }
        }
    };
    private IntentFilter mUnpluggedHeadsetIntentFilter =
        new
IntentFilter(AudioManager.ACTION_AUDIO_BECOMING_NOISY);

    @Override
    public void onCreate(@Nullable Bundle savedInstanceState)
{
        super.onCreate(savedInstanceState);

        OnBackPressedCallback callback = new
OnBackPressedCallback(true) {
            @Override
            public void handleOnBackPressed() {
                mBinding.gameView.pause();
            }
        };

requireActivity().getOnBackPressedDispatcher().addCallback(thi
s, callback);
    }

    @Override
    public View onCreateView(@NonNull LayoutInflater inflater,
@Nullable ViewGroup container,
                             @Nullable Bundle
savedInstanceState) {
        mBinding = DataBindingUtil.inflate(inflater,
R.layout.fragment_game, container, false);
        mBinding.setLifecycleOwner(this);
```

```
        // Make GameView lifecycle aware

getViewLifecycleOwner().getLifecycle().addObserver(mBinding.ga
meView);

        Log.d(TAG, TAG + " created");

        return mBinding.getRoot();
    }

    @Override
    public void onViewCreated(@NonNull View view, @Nullable
Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);

        GameRepository repository =
GameRepository.getInstance(

Objects.requireNonNull(getActivity()).getApplicationContext())
;

        SavedGameState savedGameState;
        if (getArguments() != null) {
            savedGameState =
GameFragmentArgs.fromBundle(getArguments()).getSavedGameState(
);
        } else {
            throw new IllegalStateException("Saved game state
was not passes as argument to GameFragment");
        }

        // See why at:
        //
https://stackoverflow.com/questions/16925317/getwidth-and-
getheight-always-returning-0-custom-view/16925529

view.getViewTreeObserver().addOnGlobalLayoutListener(new
ViewTreeObserver.OnGlobalLayoutListener() {
            @Override
            public void onGlobalLayout() {

view.getViewTreeObserver().removeOnGlobalLayoutListener(this);

                GameViewModelFactory factory = new
GameViewModelFactory(
                    repository,
                    new Vector2(mBinding.gameView.getWidth(),
mBinding.gameView.getHeight()),
```

```
                    savedGameState);

                mViewModel = new
ViewModelProvider(GameFragment.this,
factory).get(GameViewModel.class);

                mBinding.setViewModel(mViewModel);
                mBinding.gameView.setGame(mViewModel.game);

                mBinding.electronShooterPriceText.setText(
                    getString(R.string.energy_in_joules,
Game.ELECTRON_SHOOTER_PRICE));
                mBinding.photonicLaserPriceText.setText(
                    getString(R.string.energy_in_joules,
Game.PHOTONIC_LASER_PRICE));

                setListeners();
            }
        });
    }

    private void setListeners() {
        mBinding.pauseButton.setOnClickListener(view ->
mBinding.gameView.pause());

        mGamePausedSubscription =
mViewModel.game.getGamePausedSubject()
            .subscribe(isPaused -> {
                if (isPaused && !mViewModel.hasGameEnded()) {
                    showGamePauseDialog();
                }
            }, Throwable::printStackTrace);

        mViewModel.gameEnded().observe(this, messageId -> {
            if (messageId == R.string.game_won_message ||
messageId == R.string.game_lost_message) {
                mBinding.gameView.pause();
                showGameEndedDialog(messageId);
            }
        });
    }

    private void showGamePauseDialog() {
        AlertDialog.Builder builder = new
AlertDialog.Builder(getContext());

        LayoutInflater inflater =
requireActivity().getLayoutInflater();
        View layout =
```

```
inflater.inflate(R.layout.dialog_game_pause, null);

        builder.setView(layout);

        Dialog dialog = builder.create();
        Window dialogWindow = dialog.getWindow();
        if (dialogWindow != null) {
            dialogWindow.setBackgroundDrawable(new
ColorDrawable(android.graphics.Color.TRANSPARENT));
        }
        dialog.setCancelable(false);


layout.findViewById(R.id.resume_button).setOnClickListener(vie
w -> {
            dialog.dismiss();
            mBinding.gameView.resume();
        });


layout.findViewById(R.id.home_button).setOnClickListener(view
-> {
            dialog.dismiss();


NavHostFragment.findNavController(this).popBackStack(R.id.main
Fragment, false);
        });

        dialog.show();
    }

    private void showGameEndedDialog(@StringRes int messageId)
{
        AlertDialog.Builder builder = new
AlertDialog.Builder(getContext());

        LayoutInflater inflater =
requireActivity().getLayoutInflater();
        View layout =
inflater.inflate(R.layout.dialog_game_ended, null);
        ((TextView)
layout.findViewById(R.id.game_ended_text)).setText(messageId);

        builder.setView(layout);

        Dialog dialog = builder.create();
        Window dialogWindow = dialog.getWindow();
        if (dialogWindow != null) {
```

```
            dialogWindow.setBackgroundDrawable(new
ColorDrawable(android.graphics.Color.TRANSPARENT));
        }
        dialog.setCancelable(false);


layout.findViewById(R.id.home_button).setOnClickListener(view
-> {
            dialog.dismiss();


NavHostFragment.findNavController(this).popBackStack(R.id.main
Fragment, false);
        });

        dialog.show();
    }

    @Override
    public void onResume() {
        super.onResume();
        Context context = getActivity();
        if (context != null) {
            context.registerReceiver(
                mUnpluggedHeadsetBroadcastReceiver,
mUnpluggedHeadsetIntentFilter);
        }
    }

    @Override
    public void onPause() {
        super.onPause();
        Context context = getActivity();
        if (context != null) {

context.unregisterReceiver(mUnpluggedHeadsetBroadcastReceiver)
;
        }
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        if (!mGamePausedSubscription.isDisposed()) {
            mGamePausedSubscription.dispose();
        }

        // Resume music
        if (!MainActivity.mediaPlayer.isPlaying()) {
```

```
            MainActivity.mediaPlayer.start();
        }
    }
}
```

GameView.java

```
package com.example.atomictowers.screens.game;

import android.annotation.SuppressLint;
import android.content.Context;
import android.content.Intent;
import android.graphics.Canvas;
import android.util.AttributeSet;
import android.util.Log;
import android.view.MotionEvent;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

import androidx.annotation.NonNull;
import androidx.lifecycle.Lifecycle;
import androidx.lifecycle.LifecycleObserver;
import androidx.lifecycle.OnLifecycleEvent;

import com.example.atomictowers.MainActivity;
import com.example.atomictowers.components.Game;
import
com.example.atomictowers.data.game.game_state.GameStateService
;
import
com.example.atomictowers.data.game.game_state.SavedGameState;
import com.example.atomictowers.util.Vector2;

public class GameView extends SurfaceView implements Runnable,
LifecycleObserver {

    private static final String TAG =
GameView.class.getSimpleName();

    private static final int MAX_FPS = 60;
    private static final int FRAME_PERIOD = 1000 / MAX_FPS;

    private final SurfaceHolder mSurfaceHolder;

    private Thread mGameThread;
    private boolean mRunning;

    private Game mGame;

    public GameView(Context context) {
        this(context, null, 0);
    }

    public GameView(Context context, AttributeSet attrs) {
        this(context, attrs, 0);
```

```
    }

    public GameView(Context context, AttributeSet attrs, int
defStyleAttr) {
        super(context, attrs, defStyleAttr);
        mSurfaceHolder = getHolder();
        Log.d(TAG, "GameView created");
    }

    @Override
    protected void onMeasure(int widthMeasureSpec, int
heightMeasureSpec) {
        int widthMode = MeasureSpec.getMode(widthMeasureSpec);
        int heightSize =
MeasureSpec.getSize(heightMeasureSpec);

        int width;
        int height;

        if (widthMode == MeasureSpec.EXACTLY) {
            throw new IllegalStateException("GameView must not
have exact width");
        } else {
            height = heightSize;
            width = (int)
Game.getDimensionsByScreenHeight(heightSize).x;
        }

        //MUST CALL THIS
        setMeasuredDimension(width, height);
    }

    @Override
    protected void onSizeChanged(int w, int h, int oldw, int
oldh) {
        super.onSizeChanged(w, h, oldw, oldh);

        Log.i(TAG, "onSizeChanged() called");

        if (mGame != null) {
            mGame.updateDimensions(new Vector2(w, h));
        }
    }

    /**
     * Contains the game loop, which updates and renders
(draws) the game to the GameView's canvas,
     * as well as keep the game at a constant game speed
through all devices, as defined with the
```

```
     * game's {@linkplain #MAX_FPS}.
     */
    @Override
    public void run() {
        if (mGame != null) {
            mGame.resume();
        }

        Canvas canvas;
        long previousFrameTime = System.currentTimeMillis();

        while (mRunning) {
            if (mSurfaceHolder.getSurface().isValid()) {
                try {
                    long beginTime =
System.currentTimeMillis();
                    canvas = mSurfaceHolder.lockCanvas();

                    // Measure the frame time (see:
https://stackoverflow.com/questions/24561596/smoothing-out-
android-game-loop)
                    long currentTime =
System.currentTimeMillis();
                    long frameTime = currentTime -
previousFrameTime;
                    previousFrameTime = currentTime;
                    mGame.update(frameTime / 1000.0f);

                    mGame.draw(canvas);

mSurfaceHolder.unlockCanvasAndPost(canvas);

                    long sleepTime = FRAME_PERIOD -
(System.currentTimeMillis() - beginTime);

                    // Delay the thread to maintain a constant
frame rate (`FRAME_PERIOD`)
                    // for each frame
                    if (sleepTime > 0) {
                        try {
                            Thread.sleep(sleepTime);
                        } catch (InterruptedException ignored)
{
                        }
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                    pause();
                }
```

```
            }
        }
    }

    @OnLifecycleEvent(Lifecycle.Event.ON_PAUSE)
    public void pause() {
        if (!mGame.isGamePaused()) {
            mGame.pause();
        }
        mRunning = false;
        try {
            // Stop the thread (rejoin the main thread)
            mGameThread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        // Pause music
        if (MainActivity.mediaPlayer.isPlaying()) {
            MainActivity.mediaPlayer.pause();
        }

        // Save the game state with GameStateService
        saveGameState();

        Log.d(TAG, "pause() called");
    }

    private void saveGameState() {
        Intent intent = new Intent(getContext(),
GameStateService.class);
        if (!mGame.hasFinished()) {
            SavedGameState savedGameState = new
SavedGameState(
                mGame.getLevelNumber(),
                mGame.getHealth(),
                mGame.getEnergy(),
                mGame.getNumberOfCreatedAtoms(),
                mGame.getComponentsMap());

intent.putExtra(GameStateService.GAME_STATE_INTENT_EXTRA_NAME,
savedGameState);
        }
        getContext().startService(intent);
    }

    @OnLifecycleEvent(Lifecycle.Event.ON_CREATE)
    public void resume() {
        mRunning = true;
```

```
        mGameThread = new Thread(this);
        mGameThread.start();

        // Resume music
        if (!MainActivity.mediaPlayer.isPlaying()) {
            MainActivity.mediaPlayer.start();
        }

        Log.d(TAG, "resume() called");
    }

    @SuppressLint("ClickableViewAccessibility")
    @Override
    public boolean onTouchEvent(MotionEvent event) {
        if (mGame != null && event.getAction() ==
MotionEvent.ACTION_DOWN) {

mGame.putTowerOnMap(convertTouchCoordinatesToTileIndex(event.g
etX(), event.getY()));
            return true;
        }
        return false;
    }

    @NonNull
    private Vector2 convertTouchCoordinatesToTileIndex(float
x, float y) {
        float tileSize = mGame.getTileSize();
        float xIndex = ((x - x % tileSize) / tileSize);
        float yIndex = ((y - y % tileSize) / tileSize);
        return new Vector2(xIndex, yIndex);
    }

    public void setGame(@NonNull Game game) {
        mGame = game;
    }
}
```

GameViewModel.java

```
package com.example.atomictowers.screens.game;

import android.util.Log;

import androidx.annotation.NonNull;
import androidx.lifecycle.LiveData;
import androidx.lifecycle.MutableLiveData;
import androidx.lifecycle.Transformations;
import androidx.lifecycle.ViewModel;

import com.example.atomictowers.R;
import com.example.atomictowers.components.Game;
import com.example.atomictowers.data.game.GameRepository;
import com.example.atomictowers.data.game.TowerType;
import
com.example.atomictowers.data.game.game_state.SavedGameState;
import com.example.atomictowers.util.Vector2;

import io.reactivex.android.schedulers.AndroidSchedulers;
import io.reactivex.disposables.CompositeDisposable;

public class GameViewModel extends ViewModel {
    private static final String TAG =
GameViewModel.class.getSimpleName();

    private CompositeDisposable mCompositeDisposable = new
CompositeDisposable();

    public final Game game;

    public GameViewModel(@NonNull GameRepository
gameRepository, @NonNull Vector2 gameDimensions,
                         @NonNull SavedGameState
savedGameState) {
        Log.d(TAG, TAG + " created");
        game = new Game(gameRepository, gameDimensions,
savedGameState);

        mCompositeDisposable.add(game.getHealthObservable()
            .subscribeOn(AndroidSchedulers.mainThread())
            .observeOn(AndroidSchedulers.mainThread())
            .subscribe(health -> mHealth.setValue(health),
Throwable::printStackTrace));

        mCompositeDisposable.add(game.getEnergyObservable()
            .subscribeOn(AndroidSchedulers.mainThread())
            .observeOn(AndroidSchedulers.mainThread())
            .subscribe(energy -> {
```

```
                // Check if energy decreased (due to buying a
tower)
                if (mEnergy.getValue() != null && energy <
mEnergy.getValue()) {
                    resetSelections();
                }
                mEnergy.setValue(energy);
            }, Throwable::printStackTrace));

        mCompositeDisposable.add(game.getGameEndedObservable()
            .subscribeOn(AndroidSchedulers.mainThread())
            .observeOn(AndroidSchedulers.mainThread())
            .subscribe(messageId ->
mGameEnded.setValue(messageId), Throwable::printStackTrace));
    }

    private MutableLiveData<Integer> mHealth = new
MutableLiveData<>(Game.MAX_HEALTH);

    public LiveData<Integer> getHealthPercent() {
        return Transformations.map(mHealth,
            health -> Math.round(((float) health /
Game.MAX_HEALTH) * 100));
    }


    private MutableLiveData<Integer> mEnergy = new
MutableLiveData<>(Game.INITIAL_ENERGY);

    public LiveData<Integer> getEnergy() {
        return mEnergy;
    }


    private MutableLiveData<Integer> mGameEnded = new
MutableLiveData<>();

    public LiveData<Integer> gameEnded() {
        return mGameEnded;
    }

    public boolean hasGameEnded() {
        if (mGameEnded.getValue() == null) {
            return false;
        }
        return mGameEnded.getValue() ==
R.string.game_won_message
            || mGameEnded.getValue() ==
R.string.game_lost_message;
```

```
    }


    private MutableLiveData<Boolean> mElectronShooterSelected
= new MutableLiveData<>(false);

    public LiveData<Boolean> isElectronShooterSelected() {
        return mElectronShooterSelected;
    }

    public void selectElectronShooter() {
        mElectronShooterSelected.setValue(true);
        mPhotonicLaserSelected.setValue(false);

game.selectTowerType(TowerType.ELECTRON_SHOOTER_TYPE_KEY);
    }


    private MutableLiveData<Boolean> mPhotonicLaserSelected =
new MutableLiveData<>(false);

    public LiveData<Boolean> isPhotonicLaserSelected() {
        return mPhotonicLaserSelected;
    }

    public void selectPhotonicLaser() {
        mPhotonicLaserSelected.setValue(true);
        mElectronShooterSelected.setValue(false);

game.selectTowerType(TowerType.PHOTONIC_LASER_TYPE_KEY);
    }


    public void resetSelections() {
        game.selectTowerType(null);
        mElectronShooterSelected.setValue(false);
        mPhotonicLaserSelected.setValue(false);
    }


    @Override
    protected void onCleared() {
        super.onCleared();
        if (!game.hasFinished()) {
            game.finish(0);
        }
        if (!mCompositeDisposable.isDisposed()) {
            mCompositeDisposable.dispose();
        }
```

```
        Log.d(TAG, TAG + "cleared");
    }
}
```

GameViewModelFactory.java

```
package com.example.atomictowers.screens.game;

import androidx.annotation.NonNull;
import androidx.lifecycle.ViewModel;
import androidx.lifecycle.ViewModelProvider;

import com.example.atomictowers.data.game.GameRepository;
import
com.example.atomictowers.data.game.game_state.SavedGameState;
import com.example.atomictowers.util.Vector2;

import java.lang.reflect.InvocationTargetException;

public class GameViewModelFactory implements
ViewModelProvider.Factory {

    private final GameRepository mGameRepository;
    private final Vector2 mGameDimensions;
    private final SavedGameState mSavedGameState;

    public GameViewModelFactory(@NonNull GameRepository
gameRepository, @NonNull Vector2 gameDimensions,
                                @NonNull SavedGameState
savedGameState) {
        mGameRepository = gameRepository;
        mGameDimensions = gameDimensions;
        mSavedGameState = savedGameState;
    }

    @NonNull
    @Override
    public <T extends ViewModel> T create(@NonNull Class<T>
modelClass) {
        try {
            return
modelClass.getConstructor(GameRepository.class, Vector2.class,
SavedGameState.class)
                    .newInstance(mGameRepository, mGameDimensions,
mSavedGameState);
        } catch (IllegalAccessException
                | InstantiationException
                | InvocationTargetException
                | NoSuchMethodException e) {
            throw new IllegalArgumentException("Cannot create
an instance of " + modelClass, e);
        }
    }
}
```

**screens.settings Package**

SettingsFragment.java

```
package com.example.atomictowers.screens.settings;

import android.content.Context;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.databinding.DataBindingUtil;
import androidx.fragment.app.Fragment;
import androidx.lifecycle.ViewModelProvider;
import androidx.navigation.fragment.NavHostFragment;

import com.example.atomictowers.MainActivity;
import com.example.atomictowers.R;
import
com.example.atomictowers.databinding.FragmentSettingsBinding;

import java.util.Objects;

public class SettingsFragment extends Fragment {
    private SettingsViewModel mViewModel;
    private FragmentSettingsBinding mBinding;

    @Nullable
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater,
@Nullable ViewGroup container,
                             @Nullable Bundle
savedInstanceState) {
        mBinding = DataBindingUtil.inflate(inflater,
R.layout.fragment_settings, container, false);
        mViewModel = new
ViewModelProvider(this).get(SettingsViewModel.class);

        mBinding.setViewModel(mViewModel);
        mBinding.setLifecycleOwner(this);

        mBinding.backButton.setOnClickListener(
            view ->
NavHostFragment.findNavController(this).popBackStack());

        initVolumeSettings();
```

```
        return mBinding.getRoot();
    }

    private void initVolumeSettings() {

mViewModel.getVolumePercentage().setValue(getVolumePercentageF
romPreferences());

        mViewModel.getVolumePercentage()
            .observe(getViewLifecycleOwner(), volumePercentage
-> {
                float volume = volumePercentage / 100f;
                MainActivity.mediaPlayer.setVolume(volume,
volume);
            });

        mViewModel.getSaveVolumePreferenceEvent()
            .observe(getViewLifecycleOwner(), shouldSave -> {
                if (shouldSave) {
                    saveVolumePreference();
                    mViewModel.saveVolumePreferenceComplete();
                }
            });
    }

    private int getVolumePercentageFromPreferences() {
        SharedPreferences sharedPref =
Objects.requireNonNull(getActivity()).getPreferences(Context.M
ODE_PRIVATE);
        float volume = sharedPref.getFloat(
            getString(R.string.key_music_volume),
MainActivity.DEFAULT_MUSIC_VOLUME);
        return (int) (volume * 100);
    }

    private void saveVolumePreference() {
        Integer volumePercentage =
mViewModel.getVolumePercentage().getValue();

        if (volumePercentage != null) {
            SharedPreferences sharedPref =
Objects.requireNonNull(getActivity())
                .getPreferences(Context.MODE_PRIVATE);
            SharedPreferences.Editor editor =
sharedPref.edit();


editor.putFloat(getString(R.string.key_music_volume),
```

```
volumePercentage / 100f);

            editor.apply();
        }
    }
}
```

SettingsViewModel.java

```
package com.example.atomictowers.screens.settings;

import androidx.lifecycle.LiveData;
import androidx.lifecycle.MutableLiveData;
import androidx.lifecycle.ViewModel;

public class SettingsViewModel extends ViewModel {

    private final MutableLiveData<Integer> mVolumePercentage =
new MutableLiveData<>();

    public MutableLiveData<Integer> getVolumePercentage() {
        return mVolumePercentage;
    }


    private final MutableLiveData<Boolean>
mSaveVolumePreferenceEvent = new MutableLiveData<>();

    public LiveData<Boolean> getSaveVolumePreferenceEvent() {
        return mSaveVolumePreferenceEvent;
    }

    public void saveVolumePreference() {
        mSaveVolumePreferenceEvent.setValue(true);
    }

    public void saveVolumePreferenceComplete() {
        mSaveVolumePreferenceEvent.setValue(false);
    }
}
```

**screens.instructions Package**

InstructionsFragment.java

```
package com.example.atomictowers.screens.instructions;

import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.databinding.DataBindingUtil;
import androidx.fragment.app.Fragment;
import androidx.navigation.fragment.NavHostFragment;

import com.example.atomictowers.R;
import
com.example.atomictowers.databinding.FragmentInstructionsBindi
ng;

public class InstructionsFragment extends Fragment {
    FragmentInstructionsBinding mBinding;

    @Nullable
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater,
@Nullable ViewGroup container,
                             @Nullable Bundle
savedInstanceState) {
        mBinding = DataBindingUtil.inflate(inflater,
R.layout.fragment_instructions, container, false);

        mBinding.backButton.setOnClickListener(
            view ->
NavHostFragment.findNavController(this).popBackStack());

        return mBinding.getRoot();
    }
}
```

**screens.about Package**

AboutFragment.java

```
package com.example.atomictowers.screens.about;

import android.os.Bundle;
import android.text.method.LinkMovementMethod;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.databinding.DataBindingUtil;
import androidx.fragment.app.Fragment;
import androidx.navigation.fragment.NavHostFragment;

import com.example.atomictowers.R;
import
com.example.atomictowers.databinding.FragmentAboutBinding;

public class AboutFragment extends Fragment {
    private FragmentAboutBinding mBinding;

    @Nullable
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater,
@Nullable ViewGroup container,
                             @Nullable Bundle
savedInstanceState) {
        mBinding = DataBindingUtil.inflate(inflater,
R.layout.fragment_about, container, false);

        mBinding.backButton.setOnClickListener(
            view ->
NavHostFragment.findNavController(this).popBackStack());

        // To allow for hyperlinks to redirect outside the app

mBinding.creditsContentText.setMovementMethod(LinkMovementMeth
od.getInstance());

        return mBinding.getRoot();
    }
}
```

**data.game Package**

GameRepository.java

```
package com.example.atomictowers.data.game;

import android.content.Context;
import android.graphics.drawable.Drawable;
import android.os.Build;

import androidx.annotation.DrawableRes;
import androidx.annotation.NonNull;
import androidx.annotation.Nullable;

import com.example.atomictowers.R;
import
com.example.atomictowers.data.game.game_state.SavedGameState;
import com.google.gson.Gson;
import com.google.gson.reflect.TypeToken;

import java.lang.reflect.Type;
import java.util.List;
import java.util.Map;

import io.reactivex.Completable;
import io.reactivex.Single;
import io.reactivex.android.schedulers.AndroidSchedulers;
import io.reactivex.schedulers.Schedulers;

import static
com.example.atomictowers.util.Util.internalStorageFileExists;
import static
com.example.atomictowers.util.Util.readInternalStorageFile;
import static
com.example.atomictowers.util.Util.readResourceFile;
import static
com.example.atomictowers.util.Util.writeInternalStorageFile;

public class GameRepository {
    private static final String TAG =
GameRepository.class.getSimpleName();
    public static final String SAVED_GAME_STATE_FILENAME =
"saved_game_state.json";

    private volatile static GameRepository INSTANCE;
    private final Context mApplicationContext;
    private final Gson mGson = new Gson();
    private final GameCache mGameCache;

    private GameRepository(@NonNull Context
```

```
applicationContext, @NonNull GameCache gameCache) {
        mApplicationContext = applicationContext;
        mGameCache = gameCache;
    }

    public static GameRepository getInstance(@NonNull Context
applicationContext) {
        if (INSTANCE == null) {
            synchronized (GameRepository.class) {
                if (INSTANCE == null) {
                    INSTANCE = new
GameRepository(applicationContext, new GameCache());
                }
            }
        }

        return INSTANCE;
    }

    @NonNull
    public Single<Level> getLevel(int levelNumber) {
        Level level = mGameCache.getLevel(levelNumber);
        if (level != null) {
            return Single.just(level);
        }

        return setLevelsInCache()
            .andThen(Single.defer(() -> {
                Level newLevel =
mGameCache.getLevel(levelNumber);
                if (newLevel == null) {
                    throw new RuntimeException(
                        "error retrieving Level number " +
levelNumber);
                }
                return Single.just(newLevel);
            }));
    }

    @NonNull
    private Completable setLevelsInCache() {
        return Single.fromCallable(() ->
readResourceFile(mApplicationContext, R.raw.levels))
            .flatMapCompletable(levelsJson -> {
                Type type = new TypeToken<List<Level>>() {
                }.getType();
                List<Level> levels =
mGson.fromJson(levelsJson, type);
```

```
                if (levels == null) {
                    throw new RuntimeException("error parsing
`levels.json`");
                }
                mGameCache.setLevels(levels);

                return Completable.complete();
            })
            .subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread());
    }

    @NonNull
    public Single<List<Element>> getElements() {
        if (mGameCache.getElements() != null) {
            return Single.just(mGameCache.getElements());
        }

        return setElementsInCache()
            .andThen(Single.defer(() ->
Single.just(mGameCache.getElements())));
    }

    @NonNull
    public Single<Element> getElement(int atomicNumber) {
        Element element = mGameCache.getElement(atomicNumber);
        if (element != null) {
            return Single.just(element);
        }

        return setElementsInCache()
            .andThen(Single.defer(() -> {
                Element newElement =
mGameCache.getElement(atomicNumber);
                if (newElement == null) {
                    throw new RuntimeException(
                        "error retrieving Element with atomic
number " + atomicNumber);
                }
                return Single.just(newElement);
            }));
    }

    @NonNull
    private Completable setElementsInCache() {
        return Single.fromCallable(() ->
readResourceFile(mApplicationContext, R.raw.elements))
            .flatMapCompletable(elementsJson -> {
                Type type = new TypeToken<List<Element>>() {
```

```
                }.getType();
                List<Element> elements =
mGson.fromJson(elementsJson, type);

                if (elements == null) {
                    throw new RuntimeException("error parsing
`elements.json`");
                }
                mGameCache.setElements(elements);

                return Completable.complete();
            })
            .subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread());
    }

    @NonNull
    public Single<TowerType> getTowerType(@NonNull String
towerTypeKey) {
        TowerType towerType =
mGameCache.getTowerType(towerTypeKey);
        if (towerType != null) {
            return Single.just(towerType);
        }

        return setTowerTypesInCache()
            .andThen(Single.defer(() -> {
                TowerType newTowerType =
mGameCache.getTowerType(towerTypeKey);
                if (newTowerType == null) {
                    throw new RuntimeException(
                        "error retrieving WeaponType with key
`" + towerTypeKey + "`");
                }
                return Single.just(newTowerType);
            }));
    }

    @NonNull
    private Completable setTowerTypesInCache() {
        return Single.fromCallable(() ->
readResourceFile(mApplicationContext, R.raw.towers))
            .flatMapCompletable(towerTypesJson -> {
                Type mapType = new TypeToken<Map<String,
TowerType>>() {
                }.getType();
                Map<String, TowerType> towerTypes =
mGson.fromJson(towerTypesJson, mapType);
```

```
                if (towerTypes == null) {
                    throw new RuntimeException("error parsing
`towers.json`");
                }
                mGameCache.setTowerTypes(towerTypes);

                return Completable.complete();
            })
            .subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread());
    }

    @NonNull
    public Drawable getDrawableFromResources(@DrawableRes int
resourceId) {
        if (Build.VERSION.SDK_INT >=
Build.VERSION_CODES.LOLLIPOP) {
            return
mApplicationContext.getResources().getDrawable(resourceId,
null);
        } else {
            //noinspection deprecation
            return
mApplicationContext.getResources().getDrawable(resourceId);
        }
    }

    @NonNull
    public Completable setSavedGameState(@Nullable
SavedGameState gameState) {
        if (gameState == null) {
            return Completable.fromCallable(() -> {
                writeInternalStorageFile(mApplicationContext,
SAVED_GAME_STATE_FILENAME, "");
                return Completable.complete();
            })
                .subscribeOn(Schedulers.io())
                .observeOn(AndroidSchedulers.mainThread());
        }

        return Completable.fromCallable(() -> {
            String gameStateJson = mGson.toJson(gameState);
            writeInternalStorageFile(mApplicationContext,
SAVED_GAME_STATE_FILENAME, gameStateJson);
            return Completable.complete();
        })
            .subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread());
    }
```

```
    @NonNull
    public Single<SavedGameState> getSavedGameState() {
        return Single.fromCallable(() -> {
            String savedStateJson =
readInternalStorageFile(mApplicationContext,
SAVED_GAME_STATE_FILENAME);
            return mGson.fromJson(savedStateJson,
SavedGameState.class);
        })
            .subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread());
    }

    @NonNull
    public Single<Boolean> hasSavedGameState() {
        if (!internalStorageFileExists(mApplicationContext,
SAVED_GAME_STATE_FILENAME)) {
            return Single.just(false);
        }

        return Single.fromCallable(() -> {
            String savedStateJson =
readInternalStorageFile(mApplicationContext,
SAVED_GAME_STATE_FILENAME);
            return !savedStateJson.isEmpty();
        })
            .subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread());
    }
}
```

GameCache.java

```
package com.example.atomictowers.data.game;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;

import java.util.List;
import java.util.Map;
import java.util.Objects;

/**
 * Used to cache JSON deserialization results for {@link
GameRepository} after retrieval.
 */
class GameCache {

    private List<Level> mLevels;
    private List<Element> mElements;
    private Map<String, TowerType> mTowerTypes;

    @Nullable
    Level getLevel(int level) {
        if (mLevels == null) {
            return null;
        }

        return new Level(mLevels.get(level));
    }

    void setLevels(@NonNull List<Level> levels) {
        mLevels = levels;
    }

    @Nullable
    List<Element> getElements() {
        return mElements;
    }

    @Nullable
    Element getElement(int atomicNumber) {
        if (mElements == null) {
            return null;
        }

        return new Element(mElements.get(atomicNumber));
    }

    void setElements(@NonNull List<Element> elements) {
        mElements = elements;
```

```
    }

    @Nullable
    TowerType getTowerType(@NonNull String towerTypeKey) {
        if (mTowerTypes == null) {
            return null;
        }
        return new
TowerType(Objects.requireNonNull(mTowerTypes.get(towerTypeKey)
));
    }

    void setTowerTypes(@NonNull Map<String, TowerType>
towerTypes) {
        mTowerTypes = towerTypes;
    }
}
```

Level.java

```
package com.example.atomictowers.data.game;

import android.annotation.SuppressLint;

import androidx.annotation.NonNull;

import
com.example.atomictowers.components.atoms.AtomSequencer.AtomSe
quenceItem;
import com.google.gson.annotations.SerializedName;

import java.util.ArrayList;
import java.util.List;

public class Level {
    public static final int LEVEL_ONE = 0;

    @SerializedName("level")
    public int number;

    @SerializedName("name")
    public String name;

    @SerializedName("elements")
    public List<Integer> elementsAtomicNumbers;

    @SerializedName("atomSequence")
    public List<AtomSequenceItem> atomSequence;

    @SerializedName("map")
    public LevelMap map;

    public Level(@NonNull Level level) {
        number = level.number;
        name = level.name;
        elementsAtomicNumbers = new
ArrayList<>(level.elementsAtomicNumbers);
        atomSequence = new ArrayList<>(level.atomSequence);
        map = new LevelMap(level.map);
    }

    @SuppressLint("DefaultLocale")
    @NonNull
    @Override
    public String toString() {
        return String.format("\n{\nlevel: %d, map:\n%s}",
number, map);
```

```
    }
}
```

LevelMap.java

```
package com.example.atomictowers.data.game;

import androidx.annotation.NonNull;

import com.example.atomictowers.components.Game;
import com.example.atomictowers.util.Vector2;
import com.google.gson.annotations.SerializedName;

import java.util.ArrayList;
import java.util.List;

public class LevelMap {
    private static final String TAG =
LevelMap.class.getSimpleName();

    public static final int TILE_EMPTY = 0;
    public static final int TILE_PATH = 1;
    public static final int TILE_TOWER = 2;

    @SerializedName("path")
    private List<Vector2> mPath;

    private transient int[] mMap;

    @SerializedName("cols")
    public int cols;

    @SerializedName("rows")
    public int rows;

    public LevelMap(@NonNull LevelMap map) {
        cols = map.cols;
        rows = map.rows;
        mPath = new ArrayList<>(map.mPath);
    }

    @NonNull
    public List<Vector2> getPath() {
        return mPath;
    }

    @NonNull
    public int[] getMap() {
        if (mMap != null) {
            return mMap;
        }

        mMap = new int[cols * rows];
```

```
        for (int col = 0; col < cols; col++) {
            for (int row = 0; row < rows; row++) {
                mMap[row * cols + col] = TILE_EMPTY;
            }
        }
        for (Vector2 index : mPath) {
            int col = (int) index.x;
            int row = (int) index.y;
            mMap[row * cols + col] = TILE_PATH;
        }
        return mMap;
    }

    public int getAtIndex(@NonNull Vector2 index) {
        int col = (int) index.x;
        int row = (int) index.y;
        return getAtIndex(col, row);
    }

    public int getAtIndex(int col, int row) {
        return getMap()[row * cols + col];
    }

    public void setAtPosition(@NonNull Vector2 position, float
tileSize, int value) {
        int col = (int) (position.x / tileSize);
        int row = (int) (position.y / tileSize);

        setAtIndex(col, row, value);
    }

    public void setAtIndex(@NonNull Vector2 index, int value)
{
        int col = (int) index.x;
        int row = (int) index.y;
        setAtIndex(col, row, value);
    }

    public void setAtIndex(int col, int row, int value) {
        getMap()[row * cols + col] = value;
    }

    @NonNull
    public Vector2 getStartingPosition(@NonNull Game game) {
        float startX = game.getTileSize() * (mPath.get(0).x -
0.5f);
        float startY = game.getTileSize() * (mPath.get(0).y +
0.5f);
```

```
        return new Vector2(startX, startY);
    }

    @NonNull
    public Vector2 getPositionFromPath(@NonNull Game game, int
pathIndex) {
        if (pathIndex >= mPath.size()) {
            pathIndex--;
        }

        float startX = game.getTileSize() *
(mPath.get(pathIndex).x + 0.5f);
        float startY = game.getTileSize() *
(mPath.get(pathIndex).y + 0.5f);

        return new Vector2(startX, startY);
    }

    @NonNull
    public Vector2 getEndingPosition(@NonNull Game game) {
        float startX = game.getTileSize() *
(mPath.get(mPath.size() - 1).x + 1.5f);
        float startY = game.getTileSize() *
(mPath.get(mPath.size() - 1).y + 0.5f);

        return new Vector2(startX, startY);
    }

    @NonNull
    @Override
    public String toString() {
        StringBuilder builder = new StringBuilder();

        for (int row = 0; row < rows; row++) {
            for (int col = 0; col < cols; col++) {
                builder.append(getAtIndex(col, row)).append(",
");
            }
            builder.append('\n');
        }

        return builder.toString();
    }
}
```

Element.java

```
package com.example.atomictowers.data.game;

import androidx.annotation.NonNull;

import com.google.gson.annotations.SerializedName;

public class Element {
    public static final int HYDROGEN = 1;
    public static final int HELIUM = 2;
    public static final int LITHIUM = 3;
    public static final int BERYLLIUM = 4;
    public static final int BORON = 5;
    public static final int CARBON = 6;
    public static final int NITROGEN = 7;
    public static final int OXYGEN = 8;

    @SerializedName("name")
    public String name;

    @SerializedName("symbol")
    public String symbol;

    @SerializedName("protons")
    public int protons;

    @SerializedName("neutrons")
    public int neutrons;

    @SerializedName("color")
    public String colorString;

    @SerializedName("summary")
    public String summary;

    public transient boolean isLastAtom = false;

    public Element(@NonNull Element element) {
        name = element.name;
        symbol = element.symbol;
        protons = element.protons;
        neutrons = element.neutrons;
        colorString = element.colorString;
        summary = element.summary;
    }
}
```

TowerType.java

```
package com.example.atomictowers.data.game;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;

import java.util.List;
import java.util.Map;
import java.util.Objects;

/**
 * Used to cache JSON deserialization results for {@link
GameRepository} after retrieval.
 */
class GameCache {

    private List<Level> mLevels;
    private List<Element> mElements;
    private Map<String, TowerType> mTowerTypes;

    @Nullable
    Level getLevel(int level) {
        if (mLevels == null) {
            return null;
        }

        return new Level(mLevels.get(level));
    }

    void setLevels(@NonNull List<Level> levels) {
        mLevels = levels;
    }

    @Nullable
    List<Element> getElements() {
        return mElements;
    }

    @Nullable
    Element getElement(int atomicNumber) {
        if (mElements == null) {
            return null;
        }

        return new Element(mElements.get(atomicNumber));
    }

    void setElements(@NonNull List<Element> elements) {
        mElements = elements;
```

```
    }

    @Nullable
    TowerType getTowerType(@NonNull String towerTypeKey) {
        if (mTowerTypes == null) {
            return null;
        }
        return new
TowerType(Objects.requireNonNull(mTowerTypes.get(towerTypeKey)
));
    }

    void setTowerTypes(@NonNull Map<String, TowerType>
towerTypes) {
        mTowerTypes = towerTypes;
    }
}
```

WeaponType.java

```
package com.example.atomictowers.data.game;

import androidx.annotation.NonNull;

import com.example.atomictowers.components.Game;
import com.example.atomictowers.components.atoms.Atom;
import
com.example.atomictowers.components.towers.weapons.KineticWeap
on;
import com.example.atomictowers.util.Vector2;
import com.google.gson.annotations.SerializedName;

/**
 * A data class that represents properties of a {@linkplain
com.example.atomictowers.components.towers}
 */
public class WeaponType {
    private transient Vector2 mStartingPosition;
    private transient Atom mTargetAtom;

    /**
     * The magnitude of the {@linkplain KineticWeapon
KineticWeapon}'s
     * velocity. If the weapon is a beam for example, the
speed will be 0.
     */
    @SerializedName("speed")
    public float speed;

    /**
     * A floating point number indicating the relative effect
of the damage in the atom's strength.
     * For example, a relative damage of 0.1 will decrease the
strength of an atom
     * by 0.1 * {@linkplain
com.example.atomictowers.components.Game#DAMAGE_MULTIPLIER
Game.DAMAGE_MULTIPLIER}.
     */
    @SerializedName("relativeDamage")
    public float relativeDamage;

    WeaponType(@NonNull WeaponType weaponType) {
        speed = weaponType.speed;
        relativeDamage = weaponType.relativeDamage;
    }

    public Atom getTargetAtom() {
        if (mTargetAtom == null) {
```

```
            throw new IllegalStateException(
                "target Atom was not set - to set target atom
use `setTargetAtom(targetAtom)`");
        }

        return mTargetAtom;
    }

    public void setTargetAtom(@NonNull Atom targetAtom) {
        mTargetAtom = targetAtom;
    }

    @NonNull
    public Vector2 getStartingPosition() {
        if (mStartingPosition == null) {
            throw new IllegalStateException("weapon's starting
position was not set" +
                " - to set the starting position use
`setStartingPosition()`");
        }

        return mStartingPosition;
    }

    public void setStartingPosition(@NonNull Vector2
startingPosition) {
        mStartingPosition = startingPosition;
    }

    public float getDamage() {
        return relativeDamage * Game.DAMAGE_MULTIPLIER;
    }
}
```

**data.game.game_state Package**

GameStateService.java

```
package com.example.atomictowers.data.game.game_state;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.util.Log;

import androidx.annotation.Nullable;

import com.example.atomictowers.data.game.GameRepository;

import java.util.Objects;

import io.reactivex.disposables.Disposable;

public class GameStateService extends Service {
    private static final String TAG =
GameStateService.class.getSimpleName();
    public static final String GAME_STATE_INTENT_EXTRA_NAME =
"game_state_extra_key";

    private Disposable saveStateDisposable;

    @Override
    public int onStartCommand(@Nullable Intent intent, int
flags, int startId) {
        Log.i(TAG, "GameStateService started");

        SavedGameState savedGameState = null;
        if (intent != null && intent.getExtras() != null) {
            savedGameState = (SavedGameState)
Objects.requireNonNull(intent.getExtras())

.getSerializable(GAME_STATE_INTENT_EXTRA_NAME);
        }

        saveStateDisposable =
GameRepository.getInstance(getApplicationContext())
            .setSavedGameState(savedGameState)
            .subscribe(this::stopSelf,
Throwable::printStackTrace);

        return START_REDELIVER_INTENT;
    }

    @Nullable
```

```
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }

    @Override
    public void onDestroy() {
        if (saveStateDisposable != null &&
!saveStateDisposable.isDisposed()) {
            saveStateDisposable.dispose();
        }
        Log.i(TAG, "GameStateService destroyed");
    }
}
```

SavedGameState.java

```
package com.example.atomictowers.data.game.game_state;

import android.util.SparseArray;

import androidx.annotation.NonNull;

import com.example.atomictowers.components.Component;
import com.example.atomictowers.components.Game;
import com.example.atomictowers.components.atoms.Atom;
import com.example.atomictowers.components.towers.Tower;
import
com.example.atomictowers.components.towers.weapons.KineticWeap
on;
import com.example.atomictowers.data.game.Level;
import com.google.gson.annotations.SerializedName;

import java.io.Serializable;
import java.util.ArrayList;

public class SavedGameState implements Serializable {
    public SavedGameState(int levelNumber) {
        this.levelNumber = levelNumber;
    }

    public SavedGameState(int levelNumber, int health, int
energy, int numberOfCreatedAtoms,
                          @NonNull SparseArray<Component>
components) {
        this.levelNumber = levelNumber;
        this.health = health;
        this.energy = energy;
        this.numberOfCreatedAtoms = numberOfCreatedAtoms;
        initSavedStates(components);
    }

    private void initSavedStates(@NonNull
SparseArray<Component> components) {
        for (int i = 0; i < components.size(); i++) {
            int key = components.keyAt(i);
            Component component = components.get(key);

            if (component == null || component instanceof
KineticWeapon) {
                break;
            } else if (component instanceof Atom) {
                Atom atom = (Atom) component;
                atomSavedStates.add(new AtomSavedState(
                    atom.getAtomicNumber(),
```

```
                    atom.getStrength(),
                    atom.getPathIndex(),
                    atom.getPosition(),
                    atom.isLastAtom()));
            } else if (component instanceof Tower) {
                Tower tower = (Tower) component;
                towerSavedStates.add(new TowerSavedState(
                    tower.getTowerTypeKey(),
                    tower.getPosition()));
            }
        }
    }

    @SerializedName("level")
    public int levelNumber = Level.LEVEL_ONE;

    @SerializedName("health")
    public int health = Game.MAX_HEALTH;

    @SerializedName("energy")
    public int energy = Game.INITIAL_ENERGY;

    @SerializedName("numberOfCreatedAtoms")
    public int numberOfCreatedAtoms = 0;

    @SerializedName("componentSavedStates")
    public ArrayList<AtomSavedState> atomSavedStates = new
ArrayList<>();

    @SerializedName("towerSavedState")
    public ArrayList<TowerSavedState> towerSavedStates = new
ArrayList<>();
}
```

AtomSavedState.java

```
package com.example.atomictowers.data.game.game_state;

import androidx.annotation.NonNull;

import com.example.atomictowers.util.Vector2;
import com.google.gson.annotations.SerializedName;

import java.io.Serializable;

public class AtomSavedState implements Serializable {
    public AtomSavedState() {
    }

    public AtomSavedState(int atomicNumber, int strength, int
pathIndex, @NonNull Vector2 position,
                          boolean isLastAtom) {
        this.atomicNumber = atomicNumber;
        this.strength = strength;
        this.pathIndex = pathIndex;
        this.position = position;
        this.isLastAtom = isLastAtom;
    }

    @SerializedName("atomicNumber")
    public int atomicNumber;

    @SerializedName("strength")
    public int strength;

    @SerializedName("pathIndex")
    public int pathIndex;

    @SerializedName("position")
    public Vector2 position;

    @SerializedName("isLastAtom")
    public boolean isLastAtom = false;
}
```

TowerSavedState.java

```
package com.example.atomictowers.data.game.game_state;

import androidx.annotation.NonNull;

import com.example.atomictowers.util.Vector2;
import com.google.gson.annotations.SerializedName;

import java.io.Serializable;

public class TowerSavedState implements Serializable {
    public TowerSavedState() {
    }

    public TowerSavedState(@NonNull String towerTypeKey,
@NonNull Vector2 position) {
        this.towerTypeKey = towerTypeKey;
        this.position = position;
    }

    @SerializedName("towerTypeKey")
    public String towerTypeKey;

    @SerializedName("position")
    public Vector2 position;
}
```

**drawables Package**

LevelMapDrawable.java

```
package com.example.atomictowers.drawables;

import android.graphics.Canvas;
import android.graphics.ColorFilter;
import android.graphics.Paint;
import android.graphics.PixelFormat;
import android.graphics.Rect;
import android.graphics.drawable.Drawable;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;

import com.example.atomictowers.data.game.LevelMap;

public class LevelMapDrawable extends Drawable {

    private final LevelMap mMap;

    private float mTileSize;

    private final Paint mEmptyTilePaint = new Paint();
    private final Paint mPathTilePaint = new Paint();

    public LevelMapDrawable(@NonNull LevelMap map) {
        mMap = map;

        mEmptyTilePaint.setColor(0xff00e266);
        mPathTilePaint.setColor(0xfffbc02d);
    }

    @Override
    protected void onBoundsChange(Rect bounds) {
        super.onBoundsChange(bounds);

        if (bounds != null) {
            mTileSize = (float) (bounds.right - bounds.left) /
mMap.cols;
        }
    }

    @Override
    public void draw(@NonNull Canvas canvas) {
        for (int col = 0; col < mMap.cols; col++) {
            for (int row = 0; row < mMap.rows; row++) {
                float x = col * mTileSize;
                float y = row * mTileSize;
```

```
                switch (mMap.getAtIndex(col, row)) {
                    case LevelMap.TILE_PATH:
                        canvas.drawRect(x, y,
                            x + mTileSize, y + mTileSize,
                            mPathTilePaint);
                        break;
                    default:
                        canvas.drawRect(x, y,
                            x + mTileSize, y + mTileSize,
                            mEmptyTilePaint);
                        break;
                }
            }
        }
    }

    @Override
    public void setAlpha(int i) {
        throw new IllegalArgumentException("GameView must have
an opaque background");
    }

    @Override
    public void setColorFilter(@Nullable ColorFilter
colorFilter) {
        throw new UnsupportedOperationException("Method
setColorFilter() is not implemented");
    }

    @Override
    public int getOpacity() {
        return PixelFormat.TRANSLUCENT;
    }
}
```

AtomDrawable.java

```
package com.example.atomictowers.drawables;

import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.ColorFilter;
import android.graphics.Paint;
import android.graphics.PixelFormat;
import android.graphics.drawable.Drawable;

import androidx.annotation.ColorInt;
import androidx.annotation.NonNull;
import androidx.annotation.Nullable;

import com.example.atomictowers.components.atoms.Atom;
import com.example.atomictowers.util.Vector2;

import io.reactivex.disposables.CompositeDisposable;

public class AtomDrawable extends Drawable {

    private static final String TAG =
AtomDrawable.class.getSimpleName();

    private Vector2 mPosition;

    private float mRadius;
    private String mSymbol;

    private final Paint mBackgroundPaint = new Paint();
    private final Paint mTextPaint = new Paint();

    public AtomDrawable(@NonNull Atom atom, @NonNull
CompositeDisposable compositeDisposable) {

        compositeDisposable.add(
            atom.getPositionObservable().subscribe(position ->
mPosition = position,
                Throwable::printStackTrace));

        compositeDisposable.add(
            atom.getColorObservable().subscribe(color -> {
                mBackgroundPaint.setColor(color);

mTextPaint.setColor(calculateTextColor(color));
            }, Throwable::printStackTrace));

        compositeDisposable.add(
            atom.getRadiusObservable().subscribe(radius -> {
```

```
                mRadius = radius;
                mTextPaint.setTextSize(radius);
            }, Throwable::printStackTrace));

        compositeDisposable.add(
            atom.getSymbolObservable().subscribe(symbol ->
mSymbol = symbol,
                Throwable::printStackTrace));

        mBackgroundPaint.setAntiAlias(true);

        mTextPaint.setTextAlign(Paint.Align.CENTER);
        mTextPaint.setAntiAlias(true);
    }

    @Override
    public void draw(@NonNull Canvas canvas) {
        canvas.drawCircle(mPosition.x, mPosition.y, mRadius,
mBackgroundPaint);

        // Draw text at the center of the circle.
        // Based on:
https://stackoverflow.com/questions/11120392/android-center-
text-on-canvas
        canvas.drawText(mSymbol,
            mPosition.x,
            mPosition.y - ((mTextPaint.descent() +
mTextPaint.ascent()) / 2),
            mTextPaint);
    }

    /**
     * Calculates the text color according to the background
color.
     * Source: <a
href="https://stackoverflow.com/questions/1855884/determine-
font-color-based-on-background-color">Determine font color
based on background color</a>.
     *
     * @param backgroundColor The background color.
     * @return The calculated color of the text.
     */
    @ColorInt
    private int calculateTextColor(@ColorInt int
backgroundColor) {
        if (backgroundColor == Color.TRANSPARENT) {
            return Color.TRANSPARENT;
        }
```

```
        // Calculate the perceptive luminance
        double luminance = (0.299 * Color.red(backgroundColor)
+
            0.587 * Color.green(backgroundColor) +
            0.114 * Color.blue(backgroundColor)) / 255;

        // Return black for bright colors, white for dark
colors
        return luminance > 0.5 ? Color.BLACK : Color.WHITE;
    }

    @Override
    public void setAlpha(int i) {
    }

    @Override
    public void setColorFilter(@Nullable ColorFilter
colorFilter) {
    }

    @Override
    public int getOpacity() {
        return PixelFormat.TRANSLUCENT;
    }
}
```

**drawables.weapons Package**

ElectronProjectileDrawable.java

```
package com.example.atomictowers.drawables.weapons;

import android.graphics.Canvas;
import android.graphics.ColorFilter;
import android.graphics.Paint;
import android.graphics.PixelFormat;
import android.graphics.drawable.Drawable;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;

import
com.example.atomictowers.components.towers.weapons.ElectronPro
jectile;
import com.example.atomictowers.drawables.AtomDrawable;

public class ElectronProjectileDrawable extends Drawable {
    private static final String TAG =
AtomDrawable.class.getSimpleName();

    /**
     * The size of the radius relative to the tile dimensions.
     */
    private static final float RELATIVE_RADIUS_SIZE = 0.1f;

    private final ElectronProjectile mElectronProjectile;
    private float mRadius;
    private final Paint mPaint = new Paint();

    public ElectronProjectileDrawable(@NonNull
ElectronProjectile electronProjectile,
                                      float tileSize) {
        mElectronProjectile = electronProjectile;

        mPaint.setColor(0xFF00E5FF); // Material color: Cyan A
400
        mPaint.setAntiAlias(true);

        mRadius = tileSize * RELATIVE_RADIUS_SIZE;
    }

    @Override
    public void draw(@NonNull Canvas canvas) {
        canvas.drawCircle(mElectronProjectile.getPosition().x,
            mElectronProjectile.getPosition().y,
            mRadius, mPaint);
```

```
    }

    @Override
    public void setAlpha(int i) {
    }

    @Override
    public void setColorFilter(@Nullable ColorFilter
colorFilter) {
    }

    @Override
    public int getOpacity() {
        return PixelFormat.TRANSLUCENT;
    }
}
```

PhotonBeamDrawable.java

```
package com.example.atomictowers.drawables.weapons;

import android.graphics.Canvas;
import android.graphics.ColorFilter;
import android.graphics.Paint;
import android.graphics.PixelFormat;
import android.graphics.drawable.Drawable;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;

import
com.example.atomictowers.components.towers.weapons.PhotonBeam;
import com.example.atomictowers.util.Vector2;

public class PhotonBeamDrawable extends Drawable {
    private static final String TAG =
PhotonBeamDrawable.class.getSimpleName();

    /**
     * The size of the radius relative to the tile dimensions.
     */
    private static final float RELATIVE_RADIUS_SIZE = 0.1f;

    private final PhotonBeam mPhotonBeam;
    private final Paint mPaint = new Paint();

    public PhotonBeamDrawable(@NonNull PhotonBeam photonBeam)
{
        mPhotonBeam = photonBeam;
        mPaint.setColor(0xFFFF0000);
        mPaint.setStrokeWidth(18);
        mPaint.setAntiAlias(true);
    }

    @Override
    public void draw(@NonNull Canvas canvas) {
        Vector2 start = mPhotonBeam.getPosition();
        Vector2 end = mPhotonBeam.getTarget().getPosition();
        canvas.drawLine(start.x, start.y, end.x, end.y,
mPaint);
    }

    @Override
    public void setAlpha(int i) {
    }

    @Override
```

```
    public void setColorFilter(@Nullable ColorFilter
colorFilter) {
    }

    @Override
    public int getOpacity() {
        return PixelFormat.TRANSLUCENT;
    }
}
```

**util Package**

Util.java

```
package com.example.atomictowers.util;

import android.content.Context;

import androidx.annotation.NonNull;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;

public final class Util {

    private Util() {
    }

    @NonNull
    public static String readResourceFile(Context
applicationContext, int resourceId) throws IOException {
        InputStream inputStream =
applicationContext.getResources().openRawResource(resourceId);
        StringBuilder stringBuilder = new StringBuilder();

        try (BufferedReader reader = new BufferedReader(new
InputStreamReader(inputStream))) {
            String line;
            while ((line = reader.readLine()) != null) {
                stringBuilder.append(line);
            }
        }

        return stringBuilder.toString();
    }

    @NonNull
    public static String readInternalStorageFile(@NonNull
Context context,
                                                 @NonNull
String filename) throws IOException {
        FileInputStream fileInputStream =
context.openFileInput(filename);
        StringBuilder stringBuilder = new StringBuilder();
```

```
        try (BufferedReader reader = new BufferedReader(new
InputStreamReader(fileInputStream))) {
            String line;
            while ((line = reader.readLine()) != null) {
                stringBuilder.append(line);
            }
        }

        return stringBuilder.toString();
    }

    public static boolean internalStorageFileExists(@NonNull
Context context, @NonNull String filename) {
        File file = context.getFileStreamPath(filename);
        return file.exists();
    }

    public static void writeInternalStorageFile(@NonNull
Context context, @NonNull String filename,
                                                @NonNull
String data) throws IOException {
        try (FileOutputStream fos =
context.openFileOutput(filename, Context.MODE_PRIVATE)) {
            fos.write(data.getBytes());
        }
    }
}
```

Vector2.java

```
package com.example.atomictowers.util;

import android.annotation.SuppressLint;

import androidx.annotation.NonNull;

import com.google.gson.annotations.SerializedName;

import java.io.Serializable;

public class Vector2 implements Serializable {
    public static final Vector2 ZERO = new Vector2(0, 0);
    public static final Vector2 UNIT_RIGHT = new Vector2(1,
0);
    public static final Vector2 UNIT_DOWN = new Vector2(0, 1);
    public static final Vector2 UNIT_LEFT = new Vector2(-1,
0);
    public static final Vector2 UNIT_UP = new Vector2(0, -1);

    @SerializedName("x")
    public float x;

    @SerializedName("y")
    public float y;

    public Vector2(float x, float y) {
        this.x = x;
        this.y = y;
    }

    /**
     * Creates a new vector from polar form.
     *
     * @param magnitude The magnitude of the vector
     * @param angle     The angle in radians the vector
creates with the positive x direction.
     * @return A new {@link Vector2} instance.
     */
    @NonNull
    public static Vector2 fromPolar(float magnitude, float
angle) {
        float x = magnitude * (float) Math.cos(angle);
        float y = magnitude * (float) Math.sin(angle);

        return new Vector2(x, y);
    }

    @NonNull
```

```
    public static Vector2 unit(float angle) {
        float x = (float) Math.cos(angle);
        float y = (float) Math.sin(angle);

        return new Vector2(x, y);
    }

    public float magnitude() {
        return (float) Math.sqrt(this.x * this.x + this.y *
this.y);
    }

    @NonNull
    public Vector2 toUnit() {
        return this.scale(1 / this.magnitude());
    }

    public float angle() {
        return (float) Math.atan2(this.y, this.x);
    }

    /**
     * Sets the vector's coordinates using vector polar form.
     *
     * @param length The length of the vector
     * @param angle  The angle in radians the vector creates
with the positive x direction.
     */
    public void setPolar(float length, float angle) {
        this.x = length * (float) Math.cos(angle);
        this.y = length * (float) Math.sin(angle);
    }

    @NonNull
    public Vector2 add(@NonNull Vector2 v) {
        return new Vector2(this.x + v.x, this.y + v.y);
    }

    @NonNull
    public Vector2 subtract(@NonNull Vector2 v) {
        return new Vector2(this.x - v.x, this.y - v.y);
    }

    /**
     * Vector-scalar multiplication. Scales the {@link
Vector2} by {@code scalar}.
     *
     * @param scalar The scalar to scale the {@link Vector2}
by.
```

```
     * @return A new scaled vector.
     */
    @NonNull
    public Vector2 scale(float scalar) {
        return new Vector2(scalar * this.x, scalar * this.y);
    }

    @NonNull
    public Vector2 negate() {
        return scale(-1);
    }

    /**
     * Rotates the vector according to a given angle.
     * Formula by <a href="https://matthew-
brett.github.io/teaching/rotation_2d.html">Matthew Brett</a>.
     *
     * @param angle The angle in radians to rotate the vector
with.
     * @return A new rotated vector.
     */
    @NonNull
    public Vector2 rotate(float angle) {
        float x = (float) (Math.cos(angle * this.x) -
Math.sin(angle * this.y));
        float y = (float) (Math.sin(angle * this.x) +
Math.cos(angle * this.y));

        return new Vector2(x, y);
    }

    public float distance(@NonNull Vector2 v) {
        return this.subtract(v).magnitude();
    }

    public boolean equals(@NonNull Vector2 other) {
        return this.x == other.x && this.y == other.y;
    }

    @SuppressLint("DefaultLocale")
    @NonNull
    @Override
    public String toString() {
        return String.format("(%.2f, %.2f)", this.x, this.y);
    }
}
```

## משאבים

אתייחס כאן רק למשאבים העיקריים.

### (res/layout) Layouts

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<merge
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <fragment
        android:id="@+id/nav_host_fragment"

android:name="androidx.navigation.fragment.NavHostFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:defaultNavHost="true"
        app:navGraph="@navigation/nav_graph" />
</merge>
```

fragment_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<layout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools">

    <androidx.constraintlayout.widget.ConstraintLayout
        android:id="@+id/main"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="@drawable/screen_background"
        tools:context=".screens.main.MainFragment">

        <TextView
            android:id="@+id/title_text"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="36dp"
            android:layout_marginBottom="16dp"
            android:fontFamily="@font/orbitron_black"
            android:text="@string/app_name"
            android:textAllCaps="true"
            android:textColor="@android:color/white"
            android:textSize="56sp"
            android:textStyle="bold"

app:layout_constraintBottom_toTopOf="@+id/start_button"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintHorizontal_bias="0.5"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

        <Button
            android:id="@+id/start_button"
            style="@style/GameButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="16dp"
            android:layout_marginBottom="32dp"
            android:padding="16dp"
            android:text="@string/start"
            android:textSize="28sp"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintHorizontal_bias="0.5"
            app:layout_constraintStart_toStartOf="parent"

app:layout_constraintTop_toBottomOf="@+id/title_text" />
```

```
        <ImageButton
            android:id="@+id/menu_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="16dp"
            android:layout_marginEnd="16dp"
            android:background="@drawable/button_background"
            android:contentDescription="@string/menu"
            android:padding="8dp"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintTop_toTopOf="parent"
            app:srcCompat="@drawable/ic_menu_24px" />

    </androidx.constraintlayout.widget.ConstraintLayout>

</layout>
```

fragment_game.xml

```
<?xml version="1.0" encoding="utf-8"?>
<layout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools">

    <data>

        <import type="android.view.View" />

        <import
type="com.example.atomictowers.components.Game" />

        <variable
            name="viewModel"

type="com.example.atomictowers.screens.game.GameViewModel" />

    </data>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="horizontal"
        tools:context=".screens.game.GameFragment">

        <com.example.atomictowers.screens.game.GameView
            android:id="@+id/game_view"
            android:layout_width="wrap_content"
            android:layout_height="match_parent" />

        <androidx.constraintlayout.widget.ConstraintLayout
            android:id="@+id/sidebar"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:background="@drawable/sidebar_background"
            android:padding="8dp">

            <ImageButton
                android:id="@+id/pause_button"
                android:layout_width="wrap_content"
                android:layout_height="0dp"
                android:layout_marginTop="8dp"
                android:layout_marginEnd="8dp"

android:background="@drawable/button_background"
                android:contentDescription="@string/pause"
                android:padding="4dp"
```

```
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintTop_toTopOf="parent"
                app:srcCompat="@drawable/ic_pause_24px" />

            <TextView
                android:id="@+id/level_name"
                style="@style/GameText"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/water"
                android:textSize="14sp"

app:layout_constraintBottom_toBottomOf="@+id/pause_button"

app:layout_constraintEnd_toStartOf="@+id/pause_button"
                app:layout_constraintStart_toStartOf="parent"

app:layout_constraintTop_toTopOf="@+id/pause_button" />

            <ProgressBar
                android:id="@+id/health_progress"
                style="@style/GameProgressBar"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:layout_marginStart="8dp"
                android:layout_marginTop="16dp"
                android:layout_marginEnd="8dp"
                android:max="100"
                android:progress="@{viewModel.healthPercent}"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintStart_toStartOf="parent"

app:layout_constraintTop_toBottomOf="@+id/pause_button"
                tools:progress="75" />

            <TextView
                android:id="@+id/energy_text"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginTop="16dp"

android:background="@drawable/button_background"
                android:fontFamily="@font/orbitron_bold"
                android:paddingStart="12dp"
                android:paddingTop="8dp"
                android:paddingEnd="12dp"
                android:paddingBottom="8dp"

android:text="@{@string/energy_sidebar_indicator(viewModel.ene
```

```
rgy)}"
                android:textColor="@android:color/white"
                android:textSize="14sp"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintStart_toStartOf="parent"

app:layout_constraintTop_toBottomOf="@+id/health_progress"
                tools:text="Energy:  15 J" />

            <TextView
                android:id="@+id/towers_title_text"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="center_horizontal"
                android:layout_margin="8dp"
                android:layout_marginTop="24dp"
                android:fontFamily="@font/orbitron_black"
                android:text="@string/towers"
                android:textAllCaps="false"
                android:textColor="@android:color/white"
                android:textSize="24sp"
                android:textStyle="bold"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintStart_toStartOf="parent"

app:layout_constraintTop_toBottomOf="@+id/energy_text" />

            <FrameLayout
                android:id="@+id/electron_shooter_frame"
                android:layout_width="0dp"
                android:layout_height="0dp"
                android:layout_marginStart="24dp"
                android:layout_marginTop="8dp"
                android:layout_marginEnd="24dp"
                android:layout_weight="1"

app:layout_constraintBottom_toTopOf="@+id/electron_shooter_pri
ce_text"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintHorizontal_bias="0.5"
                app:layout_constraintStart_toStartOf="parent"

app:layout_constraintTop_toBottomOf="@+id/towers_title_text">

                <ImageView
                    android:id="@+id/electron_shooter_sidebar"
                    android:layout_width="match_parent"
                    android:layout_height="match_parent"
```

```
android:background="@{viewModel.isElectronShooterSelected()
&amp;&amp; viewModel.energy >= Game.ELECTRON_SHOOTER_PRICE ?
@drawable/button_background :
@drawable/transparent_background}"
                    android:clickable="@{viewModel.energy &lt;
Game.ELECTRON_SHOOTER_PRICE ? false : true}"

android:contentDescription="@string/electron_shooter"
                    android:onClick="@{() ->
viewModel.selectElectronShooter()}"
                    android:src="@drawable/electron_shooter"
/>

                <TextView

android:id="@+id/electron_shooter_foreground"
                    style="@style/GameText.Bold"
                    android:layout_width="match_parent"
                    android:layout_height="match_parent"

android:background="@drawable/disabled_tower_foreground"
                    android:gravity="center"
                    android:lineSpacingMultiplier="1.2"
                    android:text="@string/not_enough_energy"
                    android:textSize="12sp"
                    android:visibility="@{viewModel.energy
&lt; Game.ELECTRON_SHOOTER_PRICE ? View.VISIBLE : View.GONE}"
                    tools:visibility="visible" />

            </FrameLayout>


            <TextView
                android:id="@+id/electron_shooter_price_text"
                style="@style/GameText.Bold"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginStart="16dp"
                android:layout_marginTop="4dp"
                android:layout_marginEnd="16dp"
                android:layout_marginBottom="8dp"

android:background="@drawable/button_background"
                android:gravity="center"
                android:paddingStart="8dp"
                android:paddingTop="4dp"
                android:paddingEnd="8dp"
                android:paddingBottom="4dp"
                android:textSize="12sp"
```

```
app:layout_constraintBottom_toTopOf="@+id/photonic_laser_frame
"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintHorizontal_bias="0.5"
                app:layout_constraintStart_toStartOf="parent"

app:layout_constraintTop_toBottomOf="@+id/electron_shooter_fra
me"
                tools:text="15 J" />

            <FrameLayout
                android:id="@+id/photonic_laser_frame"
                android:layout_width="0dp"
                android:layout_height="0dp"
                android:layout_marginStart="24dp"
                android:layout_marginEnd="24dp"
                android:layout_weight="1"

app:layout_constraintBottom_toTopOf="@+id/photonic_laser_price
_text"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintHorizontal_bias="0.5"
                app:layout_constraintStart_toStartOf="parent"

app:layout_constraintTop_toBottomOf="@+id/electron_shooter_pri
ce_text">

                <ImageView
                    android:id="@+id/photonic_laser_sidebar"
                    android:layout_width="match_parent"
                    android:layout_height="match_parent"

android:background="@{viewModel.isPhotonicLaserSelected()
&amp;&amp; viewModel.energy >= Game.PHOTONIC_LASER_PRICE ?
@drawable/button_background :
@drawable/transparent_background}"
                    android:clickable="@{viewModel.energy &lt;
Game.PHOTONIC_LASER_PRICE ? false : true}"

android:contentDescription="@string/photonic_laser"
                    android:onClick="@{() ->
viewModel.selectPhotonicLaser()}"
                    android:padding="2dp"
                    android:src="@drawable/photonic_laser"
                    tools:ignore="ContentDescription" />

                <TextView
```

```
android:id="@+id/photonic_laser_foreground"
                    style="@style/GameText.Bold"
                    android:layout_width="match_parent"
                    android:layout_height="match_parent"

android:background="@drawable/disabled_tower_foreground"
                    android:gravity="center"
                    android:lineSpacingMultiplier="1.2"
                    android:padding="8dp"
                    android:text="@string/not_enough_energy"
                    android:textSize="12sp"
                    android:visibility="@{viewModel.energy
&lt; Game.PHOTONIC_LASER_PRICE ? View.VISIBLE : View.GONE}" />

            </FrameLayout>

            <TextView
                android:id="@+id/photonic_laser_price_text"
                style="@style/GameText.Bold"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginStart="16dp"
                android:layout_marginTop="4dp"
                android:layout_marginEnd="16dp"
                android:layout_marginBottom="8dp"

android:background="@drawable/button_background"
                android:gravity="center"
                android:paddingStart="8dp"
                android:paddingTop="4dp"
                android:paddingEnd="8dp"
                android:paddingBottom="4dp"
                android:textSize="12sp"
                android:visibility="visible"

app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintHorizontal_bias="0.5"
                app:layout_constraintStart_toStartOf="parent"

app:layout_constraintTop_toBottomOf="@+id/photonic_laser_frame
"
                tools:text="20 J" />

        </androidx.constraintlayout.widget.ConstraintLayout>

    </LinearLayout>

</layout>
```

fragment_settings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<layout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools">

    <data>

        <variable
            name="viewModel"

type="com.example.atomictowers.screens.settings.SettingsViewMo
del" />

    </data>

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="@drawable/screen_background"
        android:padding="8dp"
        tools:context=".screens.settings.SettingsFragment">

        <ImageButton
            android:id="@+id/back_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginStart="8dp"
            android:background="@drawable/button_background"
            android:contentDescription="@string/back"
            android:padding="8dp"

app:layout_constraintBottom_toBottomOf="@+id/settings_text"
            app:layout_constraintStart_toStartOf="parent"

app:layout_constraintTop_toTopOf="@+id/settings_text"
            app:srcCompat="@drawable/ic_arrow_back_24px" />

        <TextView
            android:id="@+id/settings_text"
            style="@style/GameText.Bold"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginStart="8dp"
            android:layout_marginTop="16dp"
            android:layout_marginEnd="8dp"
            android:text="@string/settings"
            android:textSize="30sp"
```

```
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

        <ScrollView
            android:id="@+id/settings_scroll_view"
            android:layout_width="0dp"
            android:layout_height="0dp"
            android:layout_marginStart="16dp"
            android:layout_marginTop="16dp"
            android:layout_marginEnd="16dp"
            android:layout_marginBottom="16dp"
            android:background="@drawable/button_background"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintHorizontal_bias="1.0"
            app:layout_constraintStart_toStartOf="parent"

app:layout_constraintTop_toBottomOf="@+id/settings_text">

            <LinearLayout
                android:id="@+id/settings_layout"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:orientation="vertical">

                <TextView
                    android:id="@+id/music_volume_text"
                    style="@style/GameText"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:layout_marginStart="16dp"
                    android:layout_marginTop="16dp"
                    android:layout_marginEnd="16dp"
                    android:layout_marginBottom="4dp"
                    android:paddingStart="16dp"
                    android:paddingEnd="16dp"
                    android:text="@string/music_volume"
                    android:textSize="18sp" />

                <SeekBar
                    android:id="@+id/music_volume_seekbar"
                    style="@style/GameSeekBar"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:layout_marginStart="16dp"
                    android:layout_marginTop="4dp"
                    android:layout_marginEnd="16dp"
                    android:layout_marginBottom="16dp"
```

```
                    android:max="100"
                    android:onStopTrackingTouch="@{(seekbar) -
> viewModel.saveVolumePreference()}"

android:progress="@={viewModel.volumePercentage}"
                    tools:progress="75" />

            </LinearLayout>

        </ScrollView>

    </androidx.constraintlayout.widget.ConstraintLayout>

</layout>
```

fragment_instructions.xml

```
<?xml version="1.0" encoding="utf-8"?>
<layout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools">

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="@drawable/screen_background"
        android:padding="8dp"

tools:context=".screens.instructions.InstructionsFragment">

        <ImageButton
            android:id="@+id/back_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginStart="8dp"
            android:background="@drawable/button_background"
            android:contentDescription="@string/back"
            android:padding="8dp"

app:layout_constraintBottom_toBottomOf="@+id/instructions_text
"
            app:layout_constraintStart_toStartOf="parent"

app:layout_constraintTop_toTopOf="@+id/instructions_text"
            app:srcCompat="@drawable/ic_arrow_back_24px" />

        <TextView
            android:id="@+id/instructions_text"
            style="@style/GameText.Bold"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginStart="8dp"
            android:layout_marginTop="16dp"
            android:layout_marginEnd="8dp"
            android:text="@string/instructions"
            android:textSize="30sp"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

        <ScrollView
            android:id="@+id/content_scroll_view"
            android:layout_width="0dp"
            android:layout_height="0dp"
```

```
            android:layout_marginStart="16dp"
            android:layout_marginTop="16dp"
            android:layout_marginEnd="16dp"
            android:layout_marginBottom="16dp"
            android:background="@drawable/button_background"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintHorizontal_bias="1.0"
            app:layout_constraintStart_toStartOf="parent"

app:layout_constraintTop_toBottomOf="@+id/instructions_text">

            <LinearLayout
                android:id="@+id/content_layout"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:orientation="vertical"
                android:paddingStart="32dp"
                android:paddingTop="16dp"
                android:paddingEnd="32dp"
                android:paddingBottom="16dp">

                <TextView
                    android:id="@+id/credits_content_text"
                    style="@style/GameText"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:lineSpacingExtra="8sp"

android:text="@string/instructions_content" />

            </LinearLayout>

        </ScrollView>

    </androidx.constraintlayout.widget.ConstraintLayout>

</layout>
```

fragment_about.xml

```
<?xml version="1.0" encoding="utf-8"?>
<layout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools">

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="@drawable/screen_background"
        android:padding="8dp"
        tools:context=".screens.about.AboutFragment">

        <ImageButton
            android:id="@+id/back_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginStart="8dp"
            android:background="@drawable/button_background"
            android:contentDescription="@string/back"
            android:padding="8dp"

app:layout_constraintBottom_toBottomOf="@+id/about_text"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="@+id/about_text"
            app:srcCompat="@drawable/ic_arrow_back_24px" />

        <TextView
            android:id="@+id/about_text"
            style="@style/GameText.Bold"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginStart="8dp"
            android:layout_marginTop="16dp"
            android:layout_marginEnd="8dp"
            android:text="@string/about"
            android:textSize="30sp"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

        <ScrollView
            android:id="@+id/content_scroll_view"
            android:layout_width="0dp"
            android:layout_height="0dp"
            android:layout_marginStart="16dp"
            android:layout_marginTop="16dp"
            android:layout_marginEnd="16dp"
```

```
            android:layout_marginBottom="16dp"
            android:background="@drawable/button_background"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintHorizontal_bias="1.0"
            app:layout_constraintStart_toStartOf="parent"

app:layout_constraintTop_toBottomOf="@+id/about_text">

            <LinearLayout
                android:id="@+id/content_layout"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:orientation="vertical"
                android:paddingStart="32dp"
                android:paddingTop="16dp"
                android:paddingEnd="32dp"
                android:paddingBottom="16dp">

                <TextView
                    android:id="@+id/credits_tile_text"
                    style="@style/GameText.Bold"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:layout_marginBottom="8dp"
                    android:gravity="center"
                    android:text="@string/credits"
                    android:textSize="18sp" />

                <TextView
                    android:id="@+id/credits_content_text"
                    style="@style/GameText"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:lineSpacingExtra="8sp"
                    android:linksClickable="true"

android:text="@string/credits_content_text" />

            </LinearLayout>

        </ScrollView>

    </androidx.constraintlayout.widget.ConstraintLayout>

</layout>
```

**(res/navigation) Navigation**

nav_graph.xml

```
<?xml version="1.0" encoding="utf-8"?>
<navigation
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/nav_graph"
    app:startDestination="@id/mainFragment">
    <fragment
        android:id="@+id/mainFragment"

android:name="com.example.atomictowers.screens.main.MainFragme
nt"
        android:label="fragment_main"
        tools:layout="@layout/fragment_main">
        <action

android:id="@+id/action_mainFragment_to_gameFragment"
            app:destination="@id/gameFragment"
            app:enterAnim="@anim/nav_default_enter_anim"
            app:exitAnim="@anim/nav_default_exit_anim"

app:popEnterAnim="@anim/nav_default_pop_enter_anim"
            app:popExitAnim="@anim/nav_default_pop_exit_anim"
/>
        <action

android:id="@+id/action_mainFragment_to_instructionsFragment"
            app:destination="@id/instructionsFragment"
            app:enterAnim="@anim/nav_default_enter_anim"
            app:exitAnim="@anim/nav_default_exit_anim"

app:popEnterAnim="@anim/nav_default_pop_enter_anim"
            app:popExitAnim="@anim/nav_default_pop_exit_anim"
/>
        <action

android:id="@+id/action_mainFragment_to_settingsFragment"
            app:destination="@id/settingsFragment"
            app:enterAnim="@anim/nav_default_enter_anim"
            app:exitAnim="@anim/nav_default_exit_anim"

app:popEnterAnim="@anim/nav_default_pop_enter_anim"
            app:popExitAnim="@anim/nav_default_pop_exit_anim"
/>
        <action
```

```
android:id="@+id/action_mainFragment_to_aboutFragment"
            app:destination="@id/aboutFragment"
            app:enterAnim="@anim/nav_default_enter_anim"
            app:exitAnim="@anim/nav_default_exit_anim"

app:popEnterAnim="@anim/nav_default_pop_enter_anim"
            app:popExitAnim="@anim/nav_default_pop_exit_anim"
/>
    </fragment>
    <fragment
        android:id="@+id/gameFragment"

android:name="com.example.atomictowers.screens.game.GameFragme
nt"
        android:label="fragment_game"
        tools:layout="@layout/fragment_game">
        <argument
            android:name="savedGameState"

app:argType="com.example.atomictowers.data.game.game_state.Sav
edGameState" />
    </fragment>
    <fragment
        android:id="@+id/instructionsFragment"

android:name="com.example.atomictowers.screens.instructions.In
structionsFragment"
        android:label="InstructionsFragment"
        tools:layout="@layout/fragment_instructions" />
    <fragment
        android:id="@+id/settingsFragment"

android:name="com.example.atomictowers.screens.settings.Settin
gsFragment"
        android:label="SettingsFragment"
        tools:layout="@layout/fragment_settings" />
    <fragment
        android:id="@+id/aboutFragment"

android:name="com.example.atomictowers.screens.about.AboutFrag
ment"
        android:label="AboutFragment"
        tools:layout="@layout/fragment_about" />
</navigation>
```

**(res/raw) Raw**

levels.json

```
[
  {
    "level": 1,
    "name": "Water",
    "phase": "liquid",
    "summary": "Water is an inorganic, transparent, tasteless,
odorless, and nearly colorless chemical substance, which is
the main constituent of Earth's hydrosphere, and the fluids of
most living organisms. It is vital for all known forms of
life, even though it provides no calories or organic
nutrients. Its chemical formula is H<sub>2</sub>O, meaning
that each of its molecules contains one oxygen and two
hydrogen atoms, connected by covalent bonds.",
    "elements": [1, 8],
    "atomSequence": [
      {"delay": 3000, "atomicNumber": 1},
      {"delay": 2000, "atomicNumber": 1},
      {"delay": 2000, "atomicNumber": 1},
      {"delay": 1000, "atomicNumber": 1},
      {"delay": 4000, "atomicNumber": 8},
      {"delay": 10000, "atomicNumber": 1},
      {"delay": 2000, "atomicNumber": 1},
      {"delay": 1000, "atomicNumber": 1},
      {"delay": 1000, "atomicNumber": 1},
      {"delay": 1000, "atomicNumber": 1},
      {"delay": 4000, "atomicNumber": 8},
      {"delay": 2000, "atomicNumber": 1},
      {"delay": 4000, "atomicNumber": 8},
      {"delay": 5000, "atomicNumber": 8}
    ],
    "map": {
      "path": [
        {"x": 0, "y": 2},
        {"x": 1, "y": 2},
        {"x": 2, "y": 2},
        {"x": 2, "y": 1},
        {"x": 3, "y": 1},
        {"x": 4, "y": 1},
        {"x": 4, "y": 2},
        {"x": 4, "y": 3},
        {"x": 4, "y": 4},
        {"x": 5, "y": 4},
        {"x": 6, "y": 4},
        {"x": 6, "y": 3},
        {"x": 7, "y": 3}
      ],
```

```
      "cols": 8,
      "rows": 6
    }
  }
]
```

elements.json


```
[
  {
    "name": "N/A",
    "symbol": "N/A",
    "protons": 0,
    "neutrons": 0,
    "color": "#00000000",
    "phase": "N/A",
    "summary": "N/A"
  },
  {
    "name": "Hydrogen",
    "symbol": "H",
    "protons": 1,
    "neutrons": 0,
    "color": "#FFE81313",
    "phase": "Gas",
    "summary": "Hydrogen is a chemical element with chemical
symbol H and atomic number 1. With an atomic weight of 1.00794
u, hydrogen is the lightest element on the periodic table. Its
monatomic form (H) is the most abundant chemical substance in
the Universe, constituting roughly 75% of all baryonic mass."
  },
  {
    "name": "Helium",
    "symbol": "He",
    "protons": 2,
    "neutrons": 2,
    "color": "#FF4A148C",
    "phase": "Gas",
    "summary": "Helium is a chemical element with the symbol
He and atomic number 2. It is a colorless, odorless,
tasteless, non-toxic gas, the first in the noble gas group in
the periodic table. Its boiling point is the lowest among all
the elements. Helium is the second lightest and second most
abundant element in the observable universe."
  },
  {
    "name": "Lithium",
    "symbol": "Li",
    "protons": 3,
    "neutrons": 4,
    "color": "#FF424242",
    "phase": "Solid",
    "summary": "Lithium is a chemical element with the symbol
Li and atomic number 3. It is a soft, silvery-white alkali
metal. Under standard conditions, it is the lightest metal and
the lightest solid element. Like all alkali metals, lithium is
```

```
highly reactive and flammable, and must be stored in mineral
oil."
  },
  {
    "name": "Beryllium",
    "symbol": "Be",
    "protons": 4,
    "neutrons": 5,
    "color": "#FFFF6F00",
    "phase": "Solid",
    "summary": "Beryllium is a chemical element with the
symbol Be and atomic number 4. It is a relatively rare element
in the universe, usually occurring as a product of the
spallation of larger atomic nuclei that have collided with
cosmic rays. Within the cores of stars, beryllium is depleted
as it is fused into heavier elements."
  },
  {
    "name": "Boron",
    "symbol": "B",
    "protons": 5,
    "neutrons": 6,
    "color": "#FF4E342E",
    "phase": "Solid",
    "summary": "Boron is a chemical element with the symbol B
and atomic number 5. Produced entirely by cosmic ray
spallation and supernovae, it is a low-abundance element in
the Solar System and in the Earth's crust. Boron is
concentrated on Earth by the water-solubility of its more
common naturally occurring compounds, the borate minerals."
  },
  {
    "name": "Carbon",
    "symbol": "C",
    "protons": 6,
    "neutrons": 6,
    "color": "#FF212121",
    "phase": "Solid",
    "summary": "Carbon is a chemical element with the symbol C
and atomic number 6. Carbon is the 15th most abundant element
in the Earth's crust, and the fourth most abundant element in
the universe by mass after hydrogen, helium, and oxygen.
Carbon's abundance, its unique diversity of organic compounds,
and its unusual ability to form polymers at the temperatures
commonly encountered on Earth enables this element to serve as
a common element of all known life."
  },
  {
    "name": "Nitrogen",
```

```
    "symbol": "N",
    "protons": 7,
    "neutrons": 7,
    "color": "#FF0D47A1",
    "phase": "Gas",
    "summary": "Nitrogen is the chemical element with the
symbol N and atomic number 7. It is a common element in the
universe, estimated at about seventh in total abundance in the
Milky Way and the Solar System. Nitrogen occurs in all
organisms, primarily in amino acids (and thus proteins), in
the nucleic acids (DNA and RNA) and in the energy transfer
molecule adenosine triphosphate."
  },
  {
    "name": "Oxygen",
    "symbol": "O",
    "protons": 8,
    "neutrons": 8,
    "color": "#FFFFFFFF",
    "phase": "gas",
    "summary": "Oxygen is a chemical element with symbol O and
atomic number 8. It is a member of the chalcogen group on the
periodic table and is a highly reactive nonmetal and oxidizing
agent that readily forms compounds (notably oxides) with most
elements. By mass, oxygen is the third-most abundant element
in the universe, after hydrogen and helium."
  }
]
```

towers.json


```
[
  {
    "name": "N/A",
    "symbol": "N/A",
    "protons": 0,
    "neutrons": 0,
    "color": "#00000000",
    "phase": "N/A",
    "summary": "N/A"
  },
  {
    "name": "Hydrogen",
    "symbol": "H",
    "protons": 1,
    "neutrons": 0,
    "color": "#FFE81313",
    "phase": "Gas",
    "summary": "Hydrogen is a chemical element with chemical
symbol H and atomic number 1. With an atomic weight of 1.00794
u, hydrogen is the lightest element on the periodic table. Its
monatomic form (H) is the most abundant chemical substance in
the Universe, constituting roughly 75% of all baryonic mass."
  },
  {
    "name": "Helium",
    "symbol": "He",
    "protons": 2,
    "neutrons": 2,
    "color": "#FF4A148C",
    "phase": "Gas",
    "summary": "Helium is a chemical element with the symbol
He and atomic number 2. It is a colorless, odorless,
tasteless, non-toxic gas, the first in the noble gas group in
the periodic table. Its boiling point is the lowest among all
the elements. Helium is the second lightest and second most
abundant element in the observable universe."
  },
  {
    "name": "Lithium",
    "symbol": "Li",
    "protons": 3,
    "neutrons": 4,
    "color": "#FF424242",
    "phase": "Solid",
    "summary": "Lithium is a chemical element with the symbol
Li and atomic number 3. It is a soft, silvery-white alkali
metal. Under standard conditions, it is the lightest metal and
the lightest solid element. Like all alkali metals, lithium is
```

```
highly reactive and flammable, and must be stored in mineral
oil."
  },
  {
    "name": "Beryllium",
    "symbol": "Be",
    "protons": 4,
    "neutrons": 5,
    "color": "#FFFF6F00",
    "phase": "Solid",
    "summary": "Beryllium is a chemical element with the
symbol Be and atomic number 4. It is a relatively rare element
in the universe, usually occurring as a product of the
spallation of larger atomic nuclei that have collided with
cosmic rays. Within the cores of stars, beryllium is depleted
as it is fused into heavier elements."
  },
  {
    "name": "Boron",
    "symbol": "B",
    "protons": 5,
    "neutrons": 6,
    "color": "#FF4E342E",
    "phase": "Solid",
    "summary": "Boron is a chemical element with the symbol B
and atomic number 5. Produced entirely by cosmic ray
spallation and supernovae, it is a low-abundance element in
the Solar System and in the Earth's crust. Boron is
concentrated on Earth by the water-solubility of its more
common naturally occurring compounds, the borate minerals."
  },
  {
    "name": "Carbon",
    "symbol": "C",
    "protons": 6,
    "neutrons": 6,
    "color": "#FF212121",
    "phase": "Solid",
    "summary": "Carbon is a chemical element with the symbol C
and atomic number 6. Carbon is the 15th most abundant element
in the Earth's crust, and the fourth most abundant element in
the universe by mass after hydrogen, helium, and oxygen.
Carbon's abundance, its unique diversity of organic compounds,
and its unusual ability to form polymers at the temperatures
commonly encountered on Earth enables this element to serve as
a common element of all known life."
  },
  {
    "name": "Nitrogen",
```

```
    "symbol": "N",
    "protons": 7,
    "neutrons": 7,
    "color": "#FF0D47A1",
    "phase": "Gas",
    "summary": "Nitrogen is the chemical element with the
symbol N and atomic number 7. It is a common element in the
universe, estimated at about seventh in total abundance in the
Milky Way and the Solar System. Nitrogen occurs in all
organisms, primarily in amino acids (and thus proteins), in
the nucleic acids (DNA and RNA) and in the energy transfer
molecule adenosine triphosphate."
  },
  {
    "name": "Oxygen",
    "symbol": "O",
    "protons": 8,
    "neutrons": 8,
    "color": "#FFFFFFFF",
    "phase": "gas",
    "summary": "Oxygen is a chemical element with symbol O and
atomic number 8. It is a member of the chalcogen group on the
periodic table and is a highly reactive nonmetal and oxidizing
agent that readily forms compounds (notably oxides) with most
elements. By mass, oxygen is the third-most abundant element
in the universe, after hydrogen and helium."
  }
]
```

**(res/values) Values**

styles.xml

```
<resources>

    <!-- Base application theme -->
    <style name="AppTheme"
parent="Theme.AppCompat.Light.NoActionBar">
        <!-- Fullscreen activity -->
        <item name="android:windowFullscreen">true</item>
        <item name="android:windowContentOverlay">@null</item>

        <!-- Popup menu background -->
        <item
name="android:popupMenuStyle">@style/PopupMenu</item>
        <item name="android:textAppearanceLargePopupMenu">
            @style/AppTheme.TextAppearance.Popup.Large
        </item>
        <item name="android:textAppearanceSmallPopupMenu">
            @style/AppTheme.TextAppearance.Popup.Small
        </item>

        <!-- Colors -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item
name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

    <!-- Text -->
    <style name="GameText">
        <item
name="android:fontFamily">@font/orbitron_medium</item>
        <item
name="android:textColor">@android:color/white</item>
    </style>

    <style name="GameText.Bold">
        <item
name="android:fontFamily">@font/orbitron_bold</item>
    </style>

    <!-- Button -->
    <style name="GameButton">
        <item
name="android:background">@drawable/button_background</item>
        <item
name="android:textColor">@android:color/white</item>
        <item
```

```
name="android:fontFamily">@font/orbitron_black</item>
    </style>

    <!-- SeekBar -->
    <style name="GameSeekBar">
        <item name="android:maxHeight">8dp</item>
        <item
name="android:progressDrawable">@drawable/progress_drawable</i
tem>
    </style>

    <!-- Progress bar -->
    <style name="GameProgressBar"
parent="Widget.AppCompat.ProgressBar.Horizontal">
        <item name="android:minHeight">12dp</item>
        <item
name="android:progressDrawable">@drawable/progress_drawable</i
tem>
    </style>

    <!-- Popup Menu -->
    <style name="PopupMenu"
parent="@android:style/Widget.PopupMenu">
        <item
name="android:popupBackground">@drawable/game_pause_dialog_bac
kground</item>
    </style>

    <style name="AppTheme.TextAppearance.Popup.Large"
parent="TextAppearance.AppCompat.Light.Widget.PopupMenu.Large"
>
        <item name="android:textSize">18sp</item>
        <item
name="android:textColor">@android:color/white</item>
        <item
name="android:fontFamily">@font/orbitron_medium</item>
    </style>

    <style name="AppTheme.TextAppearance.Popup.Small"
parent="TextAppearance.AppCompat.Light.Widget.PopupMenu.Small"
>
        <item
name="android:textColor">@android:color/white</item>
        <item
name="android:fontFamily">@font/orbitron_medium</item>
    </style>

</resources>
```

strings.xml

```
<resources>
    <!-- String keys -->
    <string name="key_music_volume">music_volume_key</string>

    <!-- Content strings -->
    <string name="app_name">Atomic Towers</string>
    <string name="game_state_service_description">Saves
current paused Atomic Towers game state</string>
    <string name="start">Start</string>
    <string name="towers">Towers</string>
    <string name="energy_sidebar_indicator"><b>Energy:</b>
\u00A0 %1$d J</string>
    <string name="pause">Pause</string>
    <string name="electron_shooter">Electron Shooter</string>
    <string name="resume">Resume</string>
    <string name="home">Home</string>
    <string name="menu">Menu</string>
    <string name="settings">Settings</string>
    <string name="instructions">Instructions</string>
    <string name="back">Back</string>
    <string name="music_volume">Music Volume</string>
    <string name="about">About</string>
    <string name="credits">Credits</string>
    <string name="credits_content_text">&lt;p&gt;&lt;a
href="http://dig.ccmixter.org/files/gurdonark/61313">Memories
of Better Times</a> by gurdonark © copyright 2020 Licensed
under a Creative Commons <a
href="http://creativecommons.org/licenses/by-
nc/3.0/">Attribution Noncommercial (3.0)</a> license. Ft:
Essesq</p></string>
    <string name="resume_last_game_prompt">Resume last
game?</string>
    <string name="new_game">New Game</string>
    <string name="game_won_message">Victory!</string>
    <string name="game_lost_message">Game Over</string>
    <string name="not_enough_energy">Not Enough
Energy</string>
    <string name="energy_in_joules">%1$d J</string>
    <string name="photonic_laser">Photonic Laser</string>
    <string name="water">Water</string>
    <string name="instructions_content">Your mission is to
protect the home base from those pesky atoms. You have at your
limited disposal two types of towers, each of a different cost
and capabilities.\n\nAtoms have different speeds and strengths
depending on their atomic number, which you may notice by the
chemical symbol on them. The towers strip the atoms from their
protons, so you have to place the towers on the game screen
strategically in order to destroy them.\n\nBe careful! Once an
```

```
atom gets to the home base, it will decrease the base\'s
health. When the health gets to zero, the game is lost. You
win the game if you have succeeded destroying most atoms,
without losing all of your health.\n\nYou may also notice the
Energy indicator on the side panel - when an atom looses a
proton, and when it gets fully destroyed, you get one energy
point, measured in the energy unit Joule (J). You may use
these energy points to buy towers for placement on the screen
in order to win the game.\n\nHope you\'ll have fun, and good
luck!
    </string>
</resources>
```

colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">@color/lightBlue</color>
    <color name="colorPrimaryDark">@color/blue</color>
    <color name="colorAccent">@color/yellow</color>

    <color name="blue">#0000D4</color>
    <color name="lightBlue">#00CCFF</color>
    <color name="yellow">#FBC02D</color>
    <color name="lightYellow">#FFEB3B</color>
</resources>
```

# סיכום אישי

הפרויקט היווה אחד הפרויקטים הגדולים ביותר שעבדתי עליהם בחיי. הפרויקט הציג בפניי אתגרים ומשימות מגוונות, שעל מנת לפתור אותן למדתי המון דברים חדשים, ולא רק ב-Android. העבודה על הפרויקט הייתה מאוד חווייתית עבורי – נהנתי בזמן כתיבת הפרויקט, דבר שהיה גם מלווה בלחץ עקב זמני ההגשה הצפופים והרצון לעשות ולכלול הכל.

אני חשוב שקיבלתי מספר כלים מהפרויקט:

הראשון הוא סדר וארגון. דרך הפרויקט למדתי כיצד להשתמש ב-Git ו-GitHub על מנת לעקוב אחר פיתוח הפרויקט, ופיתחתי צורת עבודה מסודרת, דבר שחשוב במיוחד עבור פרויקטים גדולים שכאלו.

השני הוא כישורי התכנות. קיבלתי כלים נוספים להתמודד עם פרויקטים גדולים, כיצד לנהל אותם, ולמדתי כיצד לכתוב קוד ברור יותר ומתומצת יותר.

השלישי הוא העובדה שלמדתי להשתמש במספר ספריות עבור הפרויקט, שאני בטוח שיעזרו לי גם בעתיד, במיוחד ספריות כמו RxJava ו-Gson, שאינן ספציפיות לפיתוח Android. הפרויקט גם לימד אותי כיצד ללמוד את עיקר הספרייה במהירות ויעילות, דבר שהייתה לי התנסות מועטה איתו בעבר.

אני חושב שהקושי או האתגר הגדול יותר שעלה בזמן בניית הפרויקט היה העמידה בזמנים, במיוחד כשמדובר בעבודה יחידנית על פרויקט רחב יחסית. הייתי צריך לנהל את הזמן שלי בצורה שאספיק לכלול את הבסיס שרציתי למשחק עד מועד ההגשה, ואודה שזו לא הייתה משימה קלה; אבל ראו – זה קרה.

אחת המסקנות שהסקתי מהעבודה על הפרויקט הייתה שגם פרויקטים שנראים פשוטים יכולים להגיע לממדים שקשה לדמיין, ולכן אסור לשפוט מראש את כמות הזמן שייקח לסיימו – הזמן המתוכנן כמעט תמיד לא יספיק. לכן, חשוב לתכנן כמה שאפשר את העבודה מראש וכן לתכנן את הזמן בצורה נבונה.

אם הייתי מתחיל היום את הפרויקט מחדש, הייתי מנהל יותר טוב את הזמנים שלי, מה שהיווה עבורי מכשול גדול. כמו כן, הייתי אולי מקדים ולומד את כל הדברים הנחוצים לפני תחילת הפרויקט, תוך נסיון לתכנונו מראש. אם זה היה המצב, אני חושב שהייתי מספיק להוסיף שלבים נוספים ועוד אפשרויות שרציתי להוסיף, אך לא הספקתי.