|  |
| --- |
| C:\Users\or\Desktop\DominionGame\background\img_dominion_background.jpg |
| **Dominion Project** |
| **מגיש: אור בר צור**  **ת.ז: 213378706**  **כיתה: יב4**  **בית ספר: גימנסיה ריאלית ראשון לציון**  **מורה מנחה: גיא אלרם** |
|  |
|  |
|  |

|  |
| --- |
|  |

תוכן עניינים

[הסבר כללי על הפרוייקט 2](#_Toc39888251)

[יכולות ומסכים 3](#_Toc39888252)

[יכולות המערכת ומגבלות 3](#_Toc39888253)

[תרשים מסכים 3](#_Toc39888254)

[פירוט מסכים 4](#_Toc39888255)

[מדריך למשתמש 11](#_Toc39888256)

[תרשים UML מקוצר 18](#_Toc39888257)

[פירוט מחלקות עם הסברים 19](#_Toc39888258)

[אלגוריתמים מעניינים 31](#_Toc39888259)

[שימוש ב-Resources 34](#_Toc39888260)

[Broadcast Receiver 34](#_Toc39888261)

[Service 34](#_Toc39888262)

[Thread 35](#_Toc39888263)

[FireBase 35](#_Toc39888264)

[טבלת פונקציות בפייתון 37](#_Toc39888265)

[רפלקציה 41](#_Toc39888266)

[קוד מלא 42](#_Toc39888267)

# הסבר כללי על הפרוייקט

**נושא הפרויקט:** אפליקציה של המשחק "דומיניון" (הסבר כללי על המשחק [כאן](https://en.wikipedia.org/wiki/Dominion_(card_game)) ובמדריך למשתמש שמסביר את חוקי המשחק באופן מפורט).

**הסיבה שבחרתי שמשחק:** לפני כשנה קנינו את משחק הקופסה של "דומיניון". ממש התמכרנו אליו ושיחקתי בו עם כל החברים, כך שזה הפך להיות גם בילוי כאשר כל אחד בביתו ומשחקים באתר שלהם אונליין. כיום, הטלפונים הסלולריים החלו להחליף את השימוש הרב במחשבים וחשבתי שיש צורך באפליקציה למשחק (לא קיימת אפליקציה כרגע). עקב צורך זה שלי ושל חבריי באפליקציה של המשחק, בחרתי לעשות פרוייקט זה.

**קהל יעד:** כל הרשומים באתר של המשחק דומיניון וגם כאלו שמשחקים רק במשחק הקופסה ורוצים לשחק דרך הטלפון.

**הסבר כללי:** המשחק מיועד ל 2 עד 4 שחקנים. בתור התחלה, יצרתי משחק כזה עבור שני שחקנים בלבד. כל אחד מהשחקנים יראה מסך שונה בזמן המשחק, הכולל את הקלפים שבידיו, את הקלפים שבמשחק, אותם יכול לקנות, וכל מיני מדדים ואופציות נוספות, למשל autoplay treasure. בכל מהלך של שחקן, לשני השחקנים יוצגו הקלפים שהפעיל וכמות הנקודות, הקניות והפעולות שיש לו באותו התור.

כאשר יפתחו את האפליקציה, יוצג מסך התחברות או הרשמה (או שכבר מחוברים). לאחר ההתחברות, יעברו למסך אופציות, בהן ניתן לחפש משחק או ליצור, לראות הישגים, ללמוד כיצד לשחק, הגדרות וכו'. למשתמש יוצגו רשימת המשחקים אליהם יכול להיכנס ולשחק וכאשר נכנס לאחד, יתחיל המשחק. בסוף כל משחק יוצג מסך של מספר נקודות לכל אחד ומסך מתאים לניצחון ולהפסד.

השרת הוא שרת פייתון אשר מקבל הודעות מהלקוח (אנדרואיד) דרך flask שזו דרך לניתוב בקשות http לפעולות בפייתון לפי הסיומת של הבקשה (לדוגמה, /get\_tables). השרת מקושר לבסיס נתונים של פיירבייס ויכול להעלות נתונים לשרת, לשלוף מידע, לבדוק ולמחוק, ולבסוף להחזיר תשובה ללקוח שביקש את הבקשה. למשל, הוא יכול לשלוף את הנתונים על המשחק ממי שיצר את המשחק (host) עבור מי שנכנס אליו, על מנת ששני השחקנים ישחקו באותו המשחק. השרת מעביר גם הודעות "real time" שמועברות משחקן לשחקן במהלך המשחק על מנת שהמסך של כל שחקן יהיה מעודכן לפי התור האחרון והפעולה האחרונה שנעשתה במשחק.

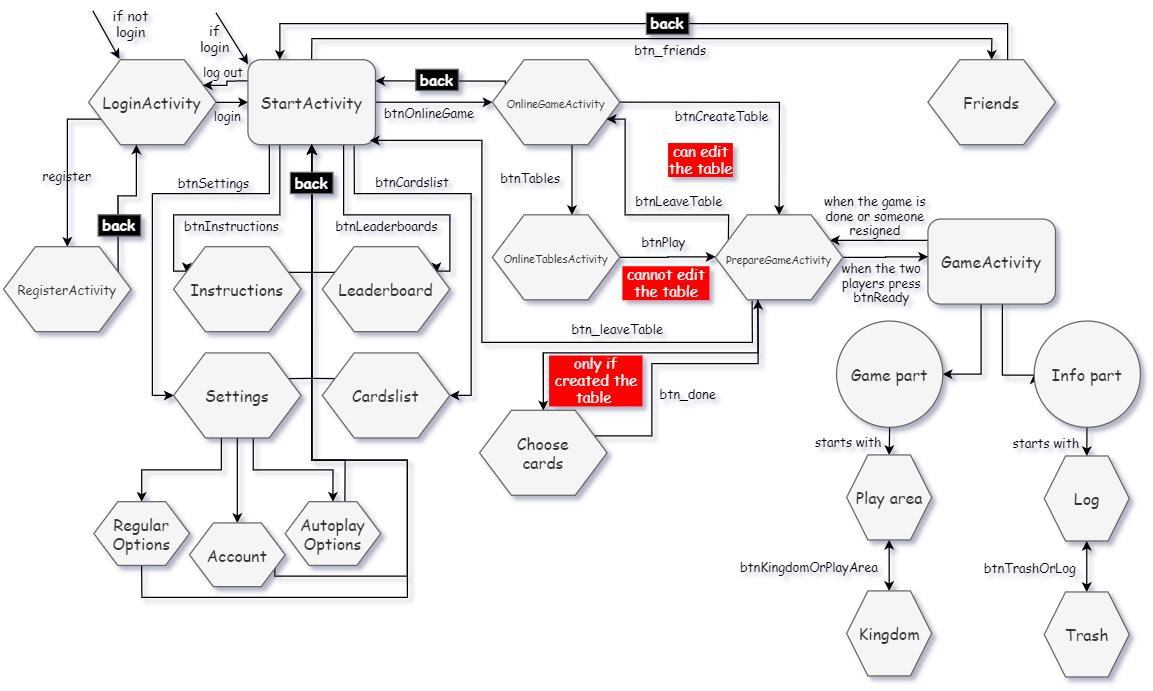
**מגבלות הפרויקט:** הפרויקט הבסיסי לא יכלול ככל הנראה את הדברים הבאים: הרחבות של המשחק, אופציה לשלושה ולארבעה שחקנים אונליין, אופציה למשחק נגד בוט כדי להתאמן, התכתבות בין שני השחקנים תוך כדי המשחק. בנוסף, מכיוון שאני עושה שימוש בשרת בבסיס נתונים של פיירבייס חינמי, ייתכן שההודעות אשר יועברו יהיו טיפה איטיות, אך ניתן לסדר זאת עם תקציב יחסית קטן לפרוייקט. הפרוייקט ירוץ על פייתון 3.0

# יכולות ומסכים

יכולות המערכת ומגבלות

* משחק דומיניון של שני שחקנים ברשת.
* אפשרות להתחבר למשחק קיים או ליצור משחק.
* אפשרות לשחק עם חברים.
* עדכון מסך המשחק במהלך התור של השחקן ובמהלך התור של היריב.
* מסך leaderboard המעדכן את המתחרים בעלי הכי הרבה ניצחונות ביחס להפסדים.
* קלפי פעולה של גרסת הבסיס עם אופציה להרחבה.
* שליחת הודעות בין השחקנים במהלך המשחק (בגדר הרחבה).

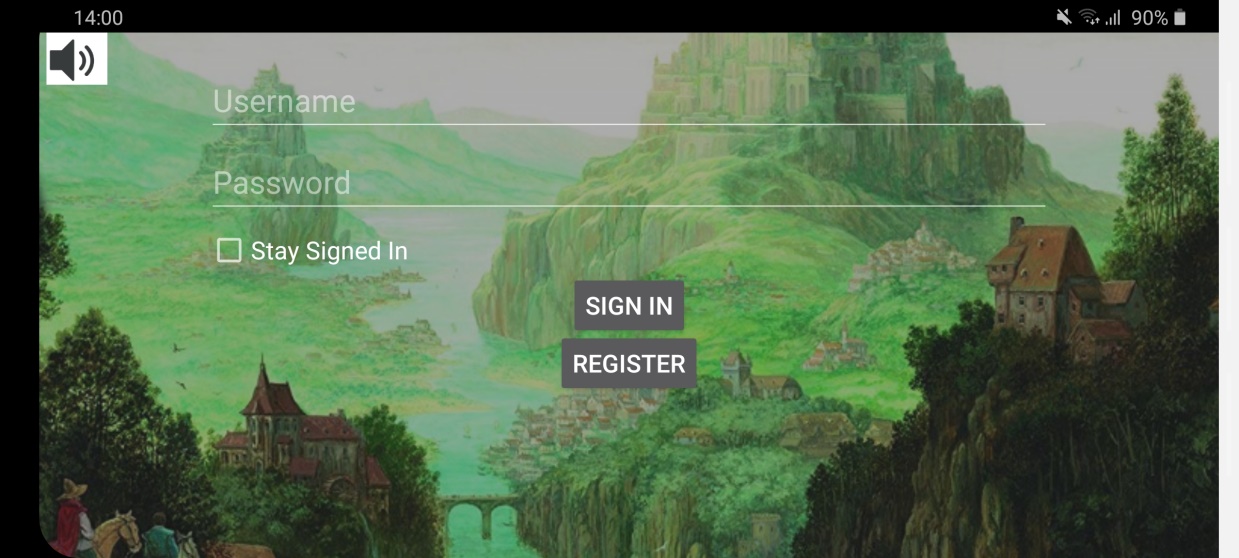
תרשים מסכים



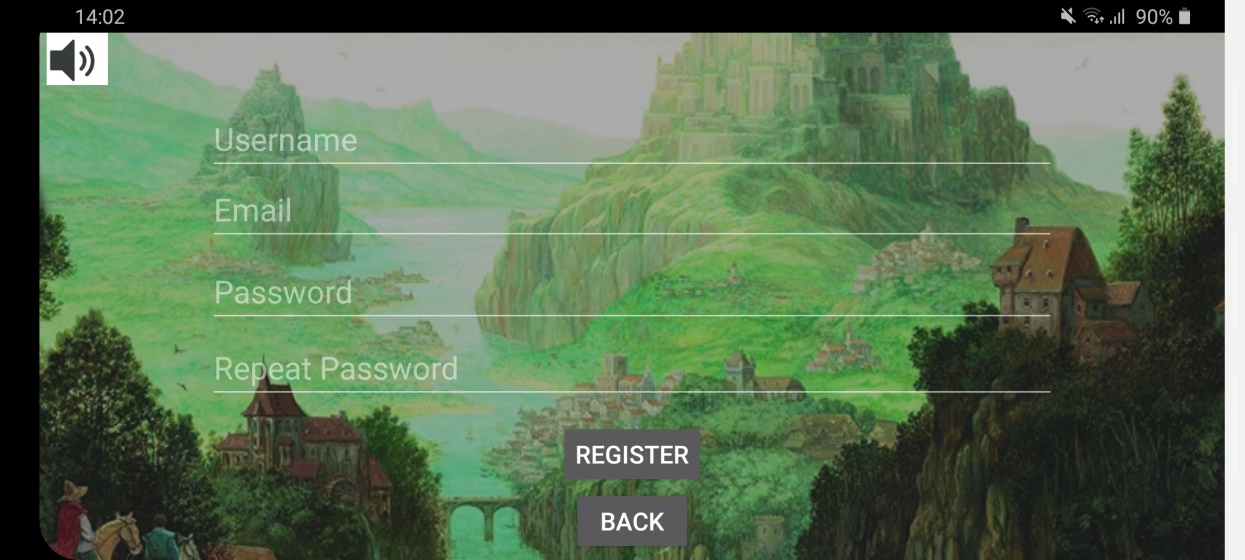
# פירוט מסכים

**\***בכל המסכים מלבד המסך GameActivity יש אפשרות להשתיק את המוסיקה ולהפעיל אותה. זה קורה על ידי SharedPreferences אשר שומר בכל לחיצה את המצב הקיים ומפעיל או מכבה את ה service בהתאם ללחיצה.

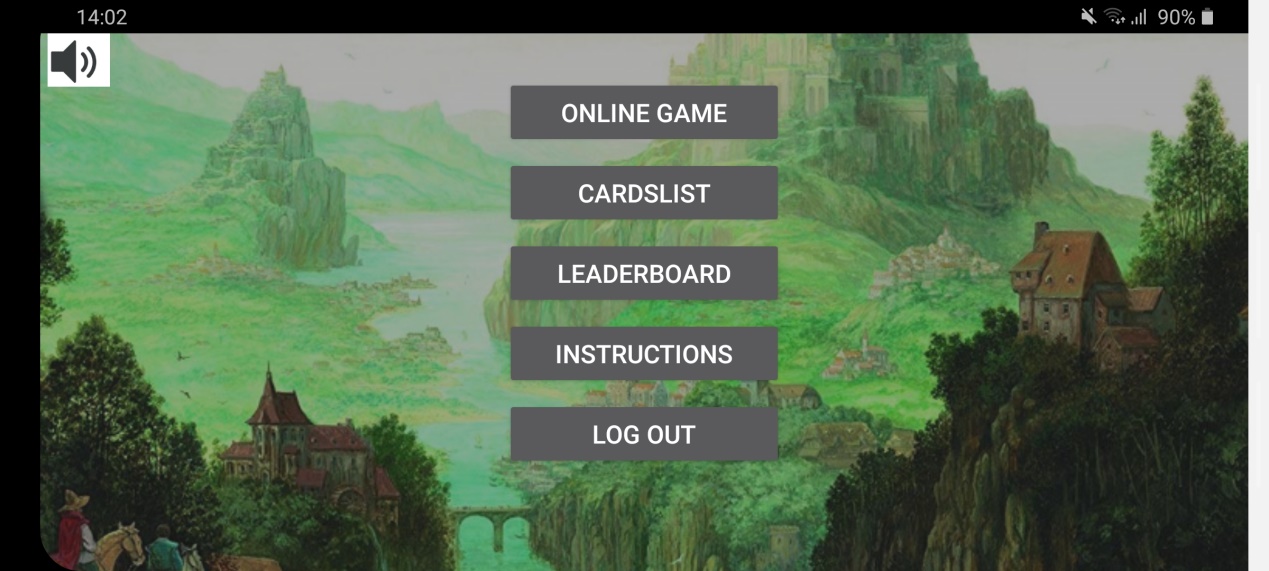
LoginActivity - מסך התחברות לאפליקציה שניתן להתחבר על ידי הזנת שם משתמש וסיסמה או לעבור למסך הירשמות לאפליקציה. אם מסמנים את ה Checkbox שכתוב עליו Stay signed In, אז בעת התחברות לאפליקציה בפעם הבאה יעברו אוטומטית למסך הפתיחה. לאחר הקשה על כפתור Sign In, מוצג progressDialog אשר מופסק כאשר מתקבלת תשובה מהשרת האם קיים שם משתמש כזה והאם הסיסמה נכונה, ולאחר מכן עוברים למסך הפתיחה אם השם משתמש והסיסמה נכונים, אחרת יוצג למשתמש Toast שהשם משתמש או הסיסמה אינם נכונים. המסך נראה כך:



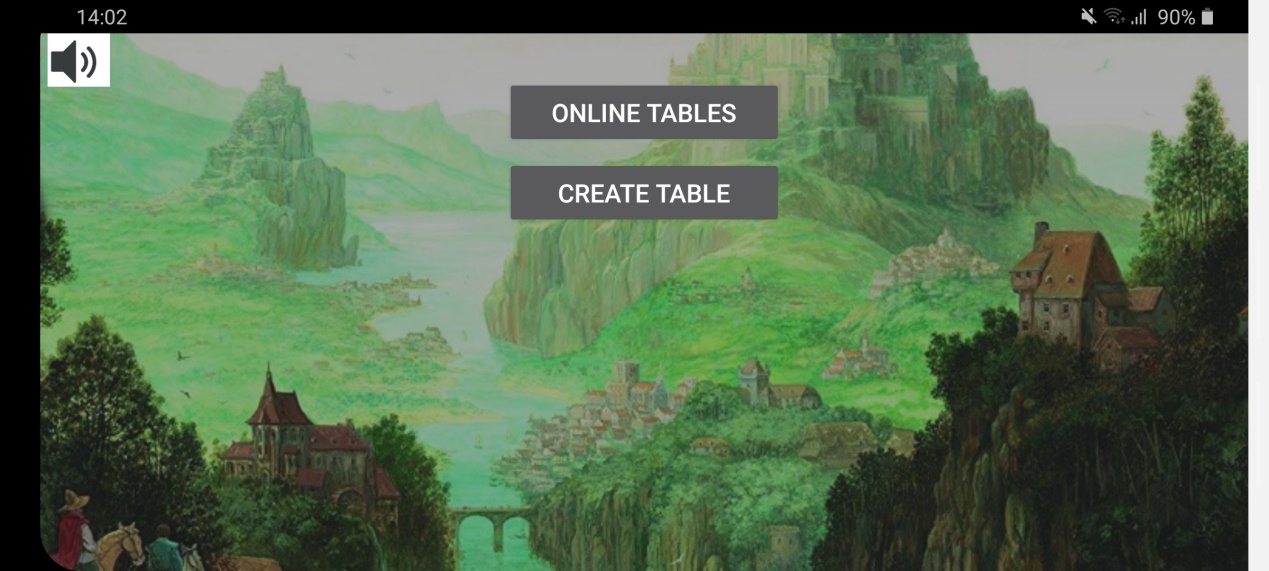
RegisterActivity - מסך הרשמה לאפליקציה אשר מאפשר הירשמות על ידי הזנת שם משתמש חדש, אימייל וסיסמה. לאחר הקשה על כפתור Register, מתבצעות בדיקות פנימיות האם המידע שהוזן חוקי ואם כן, אז מוצג progressDialog אשר מופסק כאשר מתקבלת תשובה מהשרת אם ניתן ליצור משתמש כזה ואם הוא יצר, אחרת יוצג למשתמש Toast שלא ניתן ליצור את המשתמש ופירוט עם הסיבה. אם כל המידע שהוזן תקין, השרת יוצר את המשתמש ועוברים חזרה למסך LoginActivity כאשר כבר כתוב אוטומטית בשם המשתמש את אותו שם משתמש שהמשתמש נרשם איתו. כל זה מתבצע באמצעות SharedPreferences אשר מאפשר לשמור בטלפון את פרטי החשבון של המשתמש. המסך נראה כך:



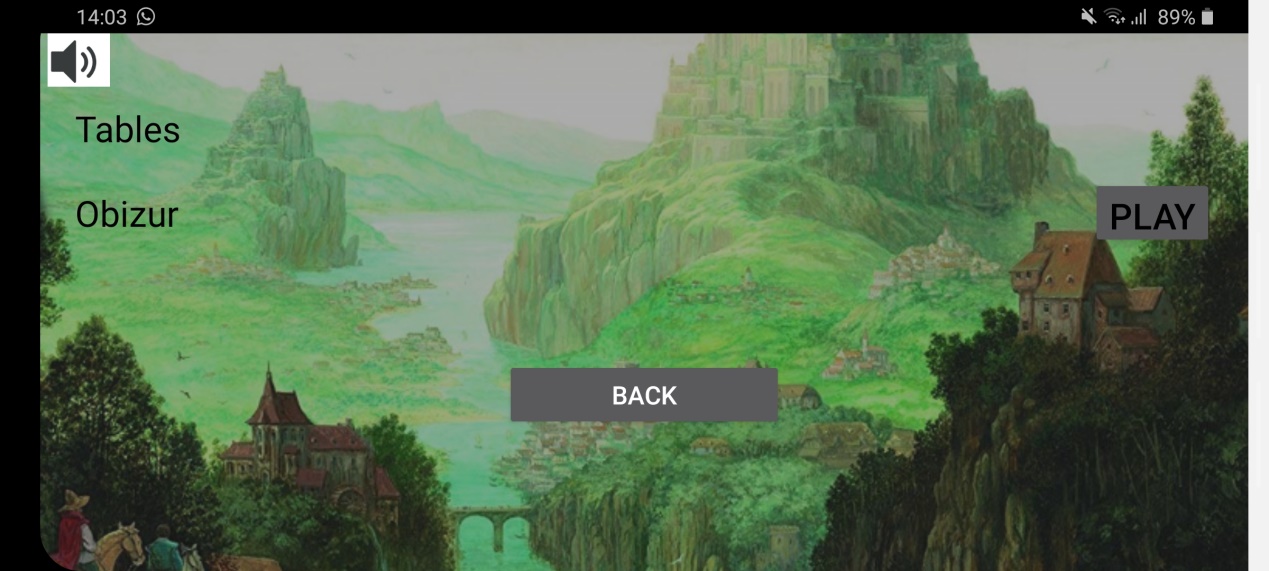
StartActivity - מסך הבית של האפליקציה. כאשר נכנסים כשמחוברים, אז מגיעים אוטומטית למסך הזה וממנו ניתן לעבור למסכים שונים ולעשות Log out. בלחיצה על הכפתור Online Game עוברים למסך OnlineGameActivity. המסך נראה כך:



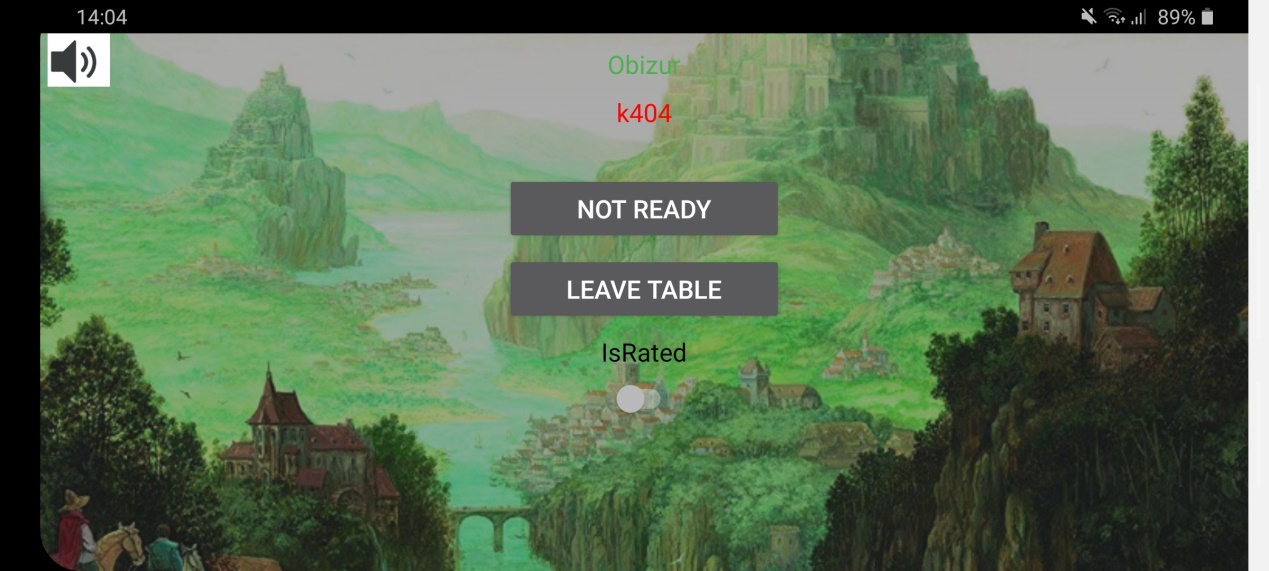
OnlineGameActivity - מסך שבו ניתן לבחור אם רוצים ליצור משחק או להצטרף למשחק קיים. בלחיצה על כפתור Online Tables עוברים למסך OnlineTablesActivity ובלחיצה על כפתור Create Table עוברים למסך ההכנה של המשחק – prepareGameActivity. המסך נראה כך:



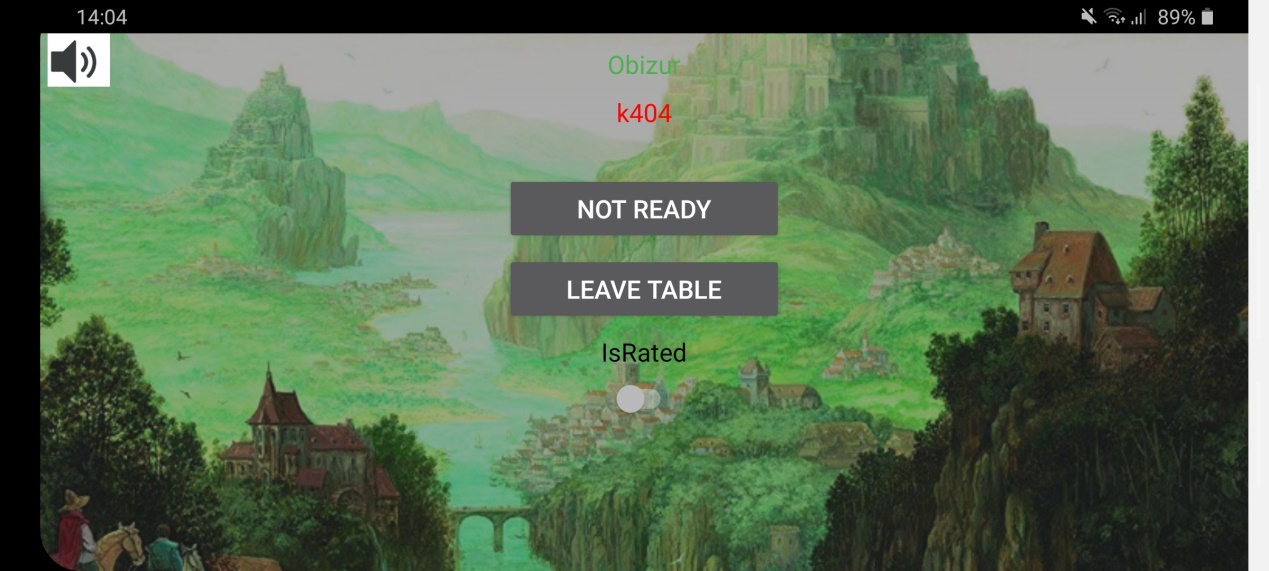
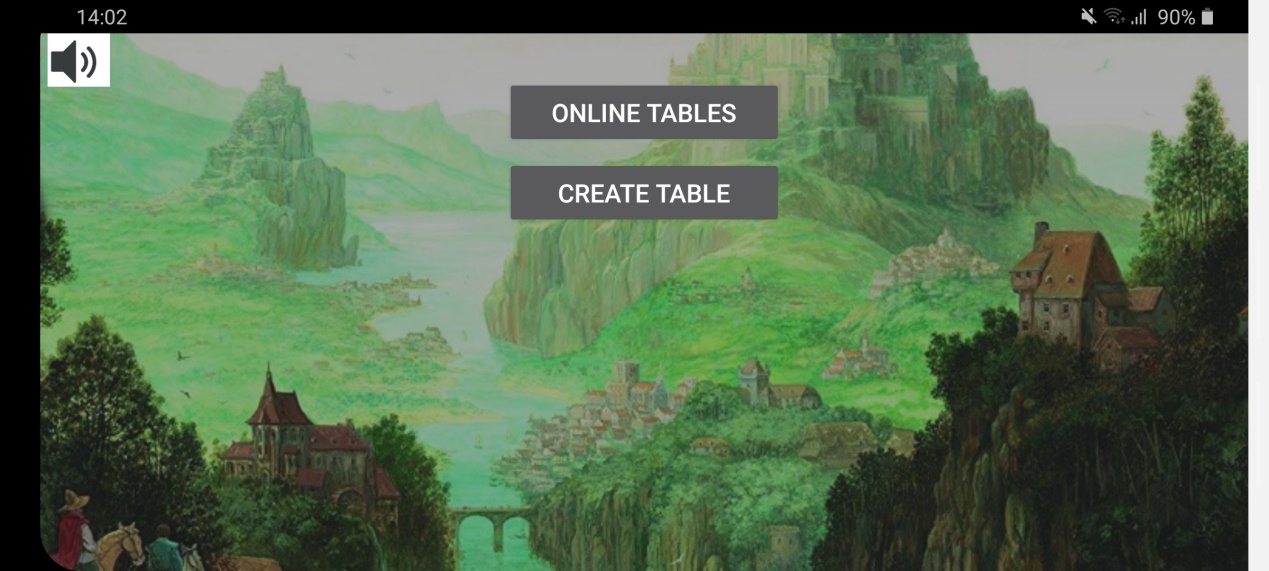
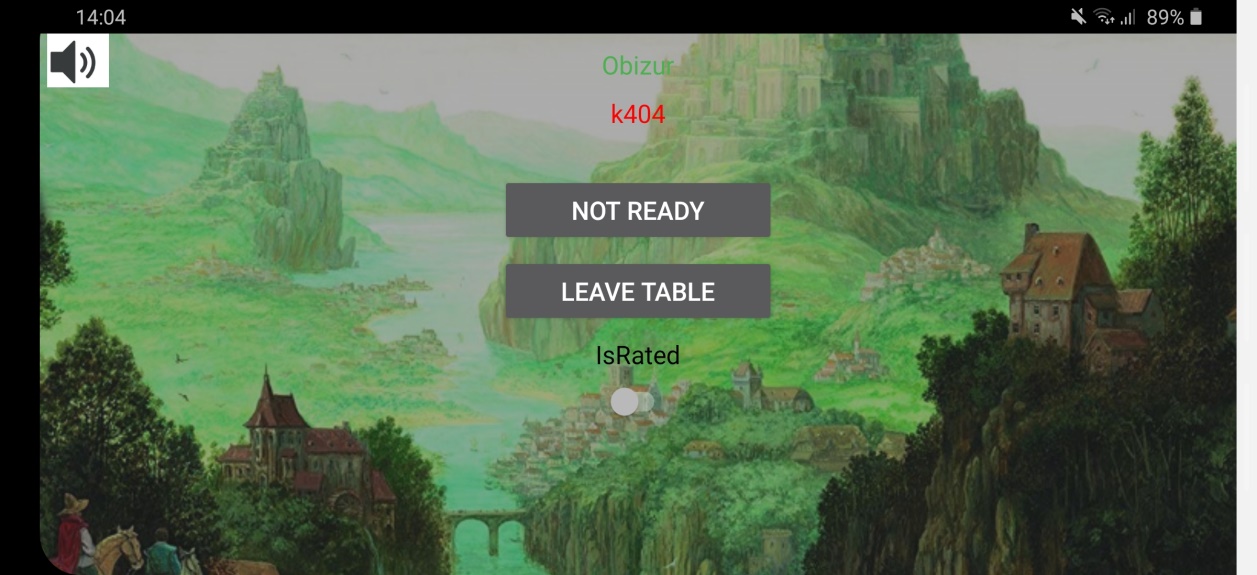
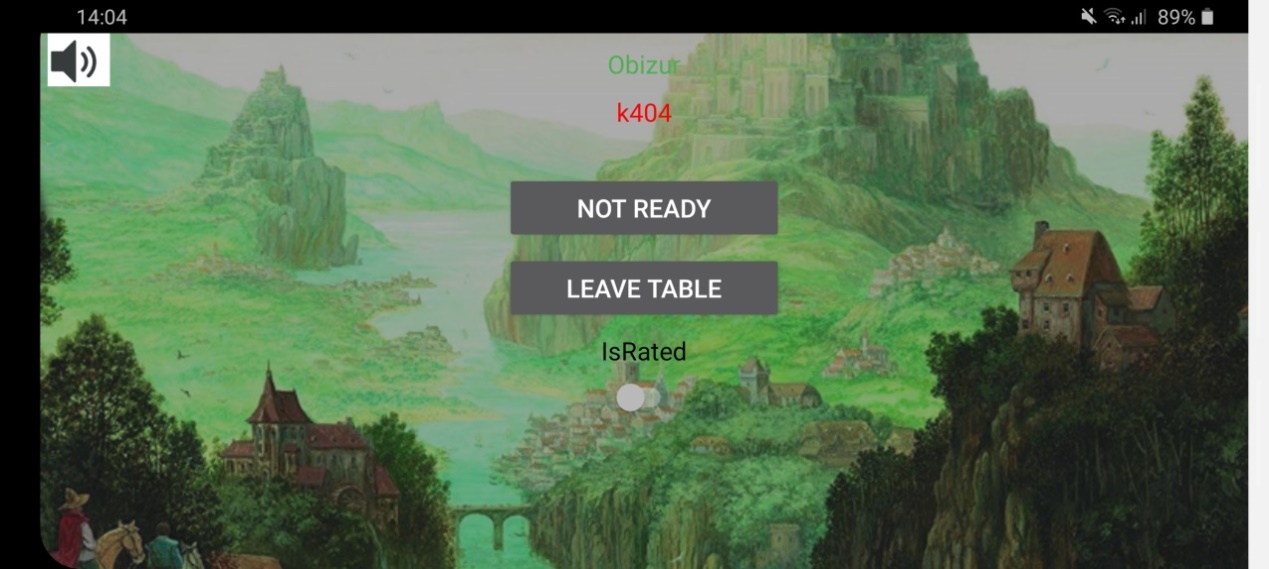
OnlineTablesActivity - מסך שבו רואים את כל המשחקים אונליין אשר מתקבלים מהשרת. כאשר מגיעים למסך הזה מופיע progressDialog שמופסק כאשר הגיעה תשובה מהשרת על המשחקים הקיימים. הטבלה מתעדכנת באמצעות TablesAdapter בלולאה שנפסקת רק עם היציאה מהמסך הזה. המסך נראה כך:



PrepareGameActivity - מסך ההכנה של המשחק. כאשר יוצרים את המשחק מופיע progressDialog אשר נגמר כאשר המשחק עלה לשרת. תחילה מופיע רק TextView עם השחקן שיצר את המשחק, אך כאשר מצטרף שחקן למשחק, הוא מופיע גם כן. ניתן ללחוץ על כפתור Ready אשר הופך את הטקסט לצבע ירוק ומעלה לשרת שהשחקן מוכן. אם השחקן הוא זה שיצר את המשחק, יש לו אופציות נוספות שבהן הוא יכול גם לשנות את ההגדרות של המשחק, למשל אם הוא יהיה מדורג או לא. כששני השחקנים מוכנים, עוברים למסך המשחק כדי להתחיל את המשחק. המסך נראה כך עבור מי שיצר את המשחק:



המסך נראה כך עבור מי שלא יצר את המשחק (מי שהצטרף):



GameActivity - מסך המשחק. כאשר עוברים למשחק הזה מופיע progressDialog אשר נגמר כאשר המידע ההתחלתי על המשחק עלה לשרת. המסך נראה בתחילת המשחק כך:

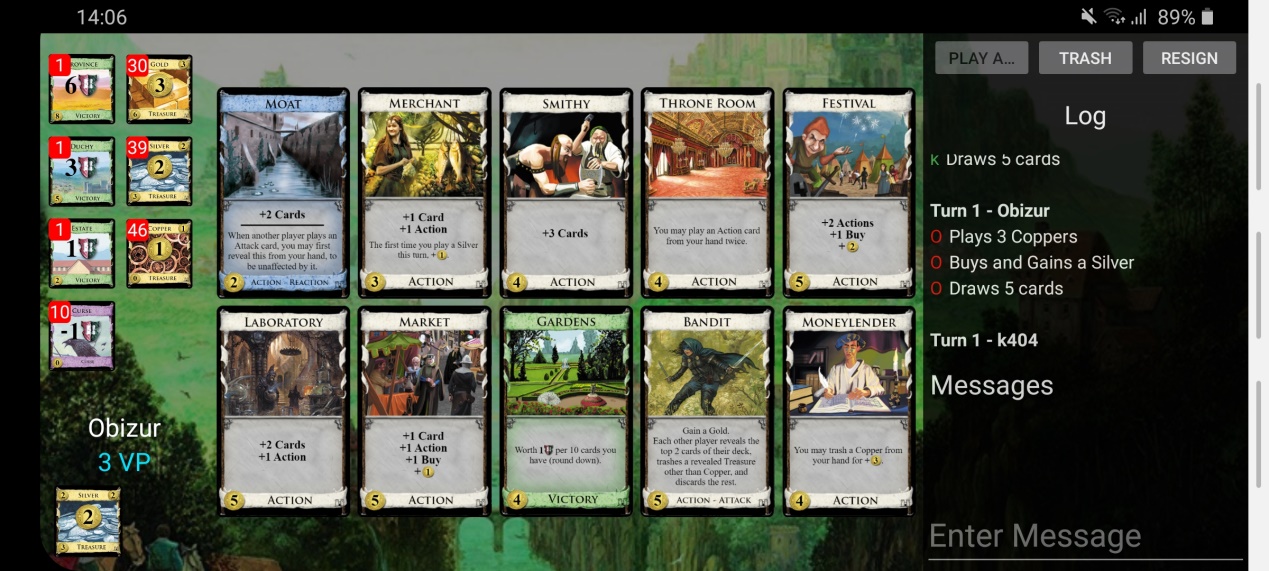
 כאשר שני השחקנים מסמנים שהם מוכנים כשלוחצים על הכפתור Start Game המשחק מתחיל ומגריל מי מתחיל. המסך נראה כך כאשר זהו התור של השחקן:



המסך נראה כך כאשר זהו אינו התור של השחקן (כשהוא מחכה שהיריב ישחק):  עבור כל קלף במשחק, ניתן ללחוץ לחיצה ארוכה אשר מפעילה המסך cardDialog של הקלף כשהוא מוגדל. ה dialog יכול להיות מבוטל בכל לחיצה על המסך. הוא נראה כך:

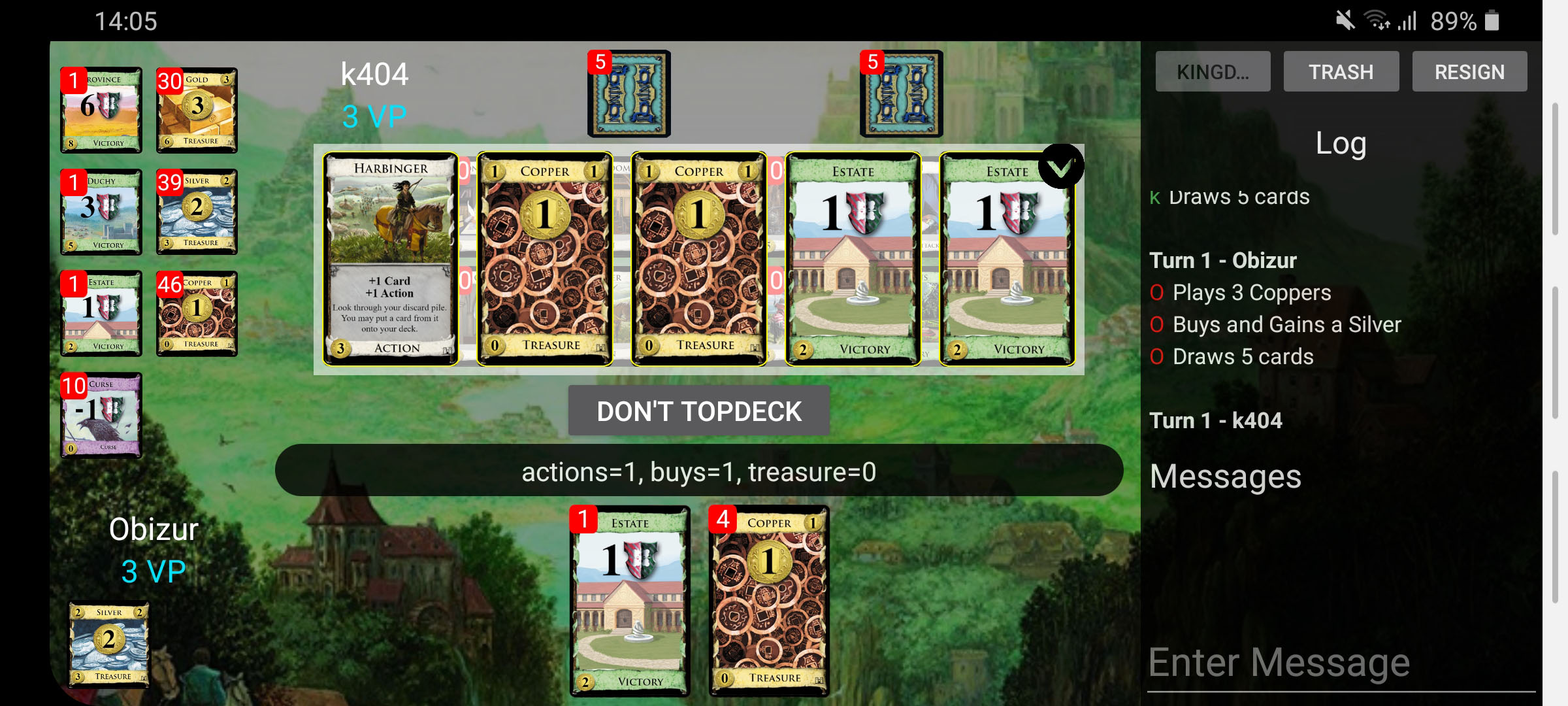


בלחיצה על הכפתור Kingdom האזור האמצעי מוחלף באזור אחר שבו מוצגים קלפי הפעולה של המשחק בגדול יותר. בלחיצה על הכפתור, מוחלף הטקסט שעליו ל Play Area שמאפשר לחזור לאזור הקודם וכך להחליף בין האזורים בלחיצה על אותו כפתור. האזור נראה כך (באמצע):

 בלחיצה על הכפתור Trash האזור הימני באמצע מוחלף באזור אחר שבו מוצגים קלפי ה Trash שבמשחק. בלחיצה על הכפתור, מוחלף הטקסט שעליו ל Log שמאפשר לחזור לאזור הקודם וכך להחליף בין האזורים בלחיצה על אותו כפתור. האזור נראה כך (ימין באמצע):



בהפעלת קלפים מסויימים עולה על המסך RecyclerView דומה למה שיש ביד. במסך הזה יש לבחור קלפים מסויימים, לסדר אותם וכדומה, בהתאם לקלף שהופעל. ניתן להזיז את האיזור גם למטה באמצעות החץ שבימין למטה על מנת לראות בו זמנית את הלוח ולא את היד. ה RecyclerView נראה כך (אמצע):



# מדריך למשתמש

\*המדריך למשתמש באנגלית כי חלק ההוראות מועתק ברובו מהאתר של דומיניון באינטרנט.

HOW TO PLAY

Dominion is a deckbuilding card game. Each player starts the game with a small deck of low-quality cards, which they can use to add more cards to their deck. The new cards they add can have a variety of effects, from drawing more cards into your hand, to being able to play more cards, to removing unwanted cards from your deck, to gaining more cards per turn than you would normally be able to, to hurling detrimental effects at your opponent. As your deck gets better, you will be able to acquire better and more expensive cards, including the very important Province cards, which are worth 6 points at the end of the game. However, the supply of cards is limited, and competition for certain cards will be fierce. The game ends when either the Province pile, or any 3 piles, empty out completely. At that point, the player with the most points wins.

On your side of the game board are a few different areas:

* Your Hand: These are the cards you’re currently holding, and can see. When you play a card, it will come from here. You normally start your turn with 5 cards in your hand.
* Your Play Area: When you play a card, it will go here.
* Your Deck: When you draw a card, it will come from here. You cannot see these cards, but you know how many cards are left in your deck.
* Your Discard Pile: When you discard a card, it will go here. At the end of your turn, all the cards you’ve played that turn will be put here. When there aren’t enough cards left in your deck in order to do something, such as drawing cards, your discard pile is shuffled, and put under your deck. You can only ever see the top card of your discard pile.

Your opponent has the same areas, just on his side of the board. You cannot see what cards are in his hand, but you can see all the cards he has played, and the top card of his discard pile. There is also a resource tracker in the center of the screen, that will say how many Actions, Buys, and Coins the current player has, as well as what the current player can do, or if the game is waiting for someone to do something.

Between the players is a communal area called the Supply. This is the set of cards that players can add to their deck this game. There is a set of seven Base Cards - Copper, Silver, Gold, Estate, Duchy, Province, and Curse. These cards are in every game. There are also 10 other cards called the Kingdom, which will change from game to game.

Whenever a player buys or otherwise gains a card, it will come from the Supply. Each pile in the Supply has a limited number of cards in it, and when that pile runs out, cards can no longer be gained from it.

**Card Types and Turn Phases**

There are 3 main types of cards in the game:

Victory cards - These cards have green borders, and are worth points at the end of the game, but don’t do anything else. They are essentially dead cards that take up space in your hand. If you have too many of them in your deck, you may find your hands clogged with these Victory (or “green”) cards, and unable to do anything of value. However, if you don’t get enough Victory cards, you may not have more points than your opponent, and will be in danger of losing. Thus a delicate balance must be struck, in deciding when to start acquiring Victory cards, which ones to get, and how many of them to add to your growing deck.

Treasure cards - These cards have yellow borders, and will be your main source of Coins, the resource used to buy cards. During your turn, you will be able to play any number of Treasures, each of which will produce some amount of Coins, typically shown on a large Coin symbol on the card. With the total amount of Coins you’ve produced, you will be able to buy one card from the Supply - each card has a cost in Coins shown in its lower left corner. After you buy a card, it is put into your discard pile. The next time you shuffle, it will be added to your deck, and you will be able to use it when it shows up in your hand.

Action cards - These cards usually have white borders, though a few have an extra type that gives them an additional or alternate color. You can only play one Action per turn. However, Actions have a wide range of abilities, and some can even let you play more Actions afterwards. Actions are the most common type of card in Dominion.

There is also one card with its own, eponymous type: The Curse card. Curses have purple borders, and are worth -1 points. You will usually not want to buy these, but a few Attack cards can force you to add a Curse to your deck.

Each turn is broken down into 3 phases:

Action phase - To start your turn, you may play one Action card. Specifically, you have 1 Action to use, which you can spend to play an Action card. Note that Actions, a resource you accumulate, are different from Action cards. You spend an Action to play an Action card. The resource counter tab in the center of the game screen tells you how many Actions you have remaining. Often when you play an Action card, it will yield an extra 1 or 2 Actions, allowing you to play more Actions afterwards. When you play an Action, it goes into your play area, and you follow the directions written on it. While most effects are written out, there are a few very common effects that are written in a shorthand:

+X Cards: Draw that number of cards from your personal deck, and add them to your hand.

+X Actions: Add that many Actions to your total, allowing you to play that many more Action cards this turn.

+X Buys: Add that many Buys to your total, allowing you to buy that many more cards this turn.

+X Coins (represented by a Coin symbol): Add that much Coin to your total, giving you that much more to spend on buying one or more cards this turn.

Buy phase - After you have finished playing Action cards (either because you’ve run out of Actions or Action cards, or because you decided you didn’t want to play any more), you may play any number of Treasures. You may then buy one card. Specifically, you have 1 Buy to use, which you can spend on a purchase. Some Action cards allow you to buy additional cards during this phase. The resource counter tab in the center of the game screen tells you how many Buys you have remaining. When you buy a card, the cost of the card is deducted from your current Coin total, and one of your Buys is spent. You may then buy another card, and so on, until you have no Buys remaining, or have decided you don’t want to buy anything else. Note that some cards cost 0 Coins - it still costs a Buy to buy one, though.

Clean-up phase - After you have finished buying cards, you then discard any cards you have played this turn, and any cards left in your hand. Discarding cards puts them into your discard pile. You then draw 5 cards from your deck, to be your hand for your next turn. If there are not enough cards left in your deck to do this, your discard pile is shuffled, and is put under what remains of your deck, and then your new hand is drawn. The same applies anytime you need to do something with the top of your deck and not enough cards remain.

Note that you are never forced to play an Action during your Action phase, and you are never forced to buy a card during your Buy phase, even if you have Actions or Buys remaining, respectively. You can always choose to end the phase early, and move on to the next one. After you’ve finished your Clean-up phase, play moves to the opponent, and they take their turn, in the same 3 phases. Once play returns to you, you will start again with a new Action phase, and have 1 Action and 1 Buy to use. If either the Province pile, or any 3 piles, are completely empty after a player’s turn, the game ends immediately and each player’s Victory points are counted.

**Secondary Types and Keywords**

Some cards have more than one type. The most common secondary type is Attack. This does not have any special ability associated with it, it just signifies that this card has a hostile effect on other players. Some other effects also reference Attack cards - several of these are Reaction cards, which have blue borders. A Reaction card has an effect that can be used at an unusual time, which is most often detailed on the lower half of its card text, beneath a dividing line. For example, the Moat card can be revealed from your hand when another player plays an Attack, to protect you from the Attack’s effects.

You will find that the text of most Dominion cards is fairly straightforward in detailing the effect of the card. However, there are a few keywords with a specific meaning:

Gain: Take a card from the Supply, and put it into your discard pile. The Supply is the set of cards that make up the large part of the game board, as described above. The most usual way to gain a card is to first buy it, but some effects instruct you to gain a card without buying it.

Discard: Put a card into your discard pile. If an effect doesn’t otherwise specify, cards are discarded from your hand. Some effects will tell you to discard a card from another location, though, such as your deck, and all the cards you have in play get discarded during your Clean-up phase. Note that when you play a card, it is not immediately put into your discard pile, but it stays in your play area until your Clean-up phase.

Trash: Put a card into the trash pile. The trash pile is used by all players as a receptacle for cards removed from their deck. Cards can only be trashed if an effect says so. Be careful: discarding and trashing are different things. If you discard a card, it will eventually be shuffled back into your deck and you will see it again in your hand. If you trash a card, it is removed from your deck entirely, and it will not show up in your hand again.

Reveal: Show this to all players. After a card is revealed, it is returned to where it came from, unless otherwise specified.

Look at: Only you get to see this. After you look at a card, it is returned to where it came from, unless otherwise specified.

Set aside: Put this card off to one side, not in your hand, play area, or discard pile, or on your deck. Cards are set aside face up by default, but some effects may tell you to set a card aside face down. Cards remain set aside until the effect that put them there says otherwise, or else until the end of the game.

USING THE DOMINION ONLINE INTERFACE

Each game begins by showing you the cards starting in yourdeck. To start playing, all players must click the "Start Game" button. Once they all have done so, play commences with the first player's turn.

To play a card from your hand, or to buy or otherwise gain a card from the Supply, simply press it. If you only want to view the card, long press it instead, and you will be shown an enlarged version of the card, including its full text.

During your Buy phase, there is a button “Autoplay Treasures” which will play all of your Treasures from your hand so you don’t have to click on them each individually.

Each pile in the Supply, and each player’s deck, has a red number in the upper left corner indicating how many cards are left in it.

As the game proceeds, everything that happens is recorded in the log, on the right side of the screen. The log is color-coded by card type, and can help you figure out what happened on a particularly complicated turn, or you can use it to check back on something if you forget what happened. A few abilities must be done through the log: specifically, the calling of Reserve cards and the ordering of simultaneous effects are prompted in blue text at the bottom of the log.

Above the log are some options:

Kingdom: This shows you all cards used in the current game (other than Basic cards), replacing the play area. This includes all Kingdom cards. These all have their text visible, but each can be long pressed to see a larger version. To switch back to the play area, simply click "Play Area", which replaces the "Kingdom" button when you are viewing the Kingdom.

Trash: This shows you the current contents of the trash, replacing the log. To switch back to the log, simply click “Log”, which replaces the “Trash” button when you are viewing the trash.

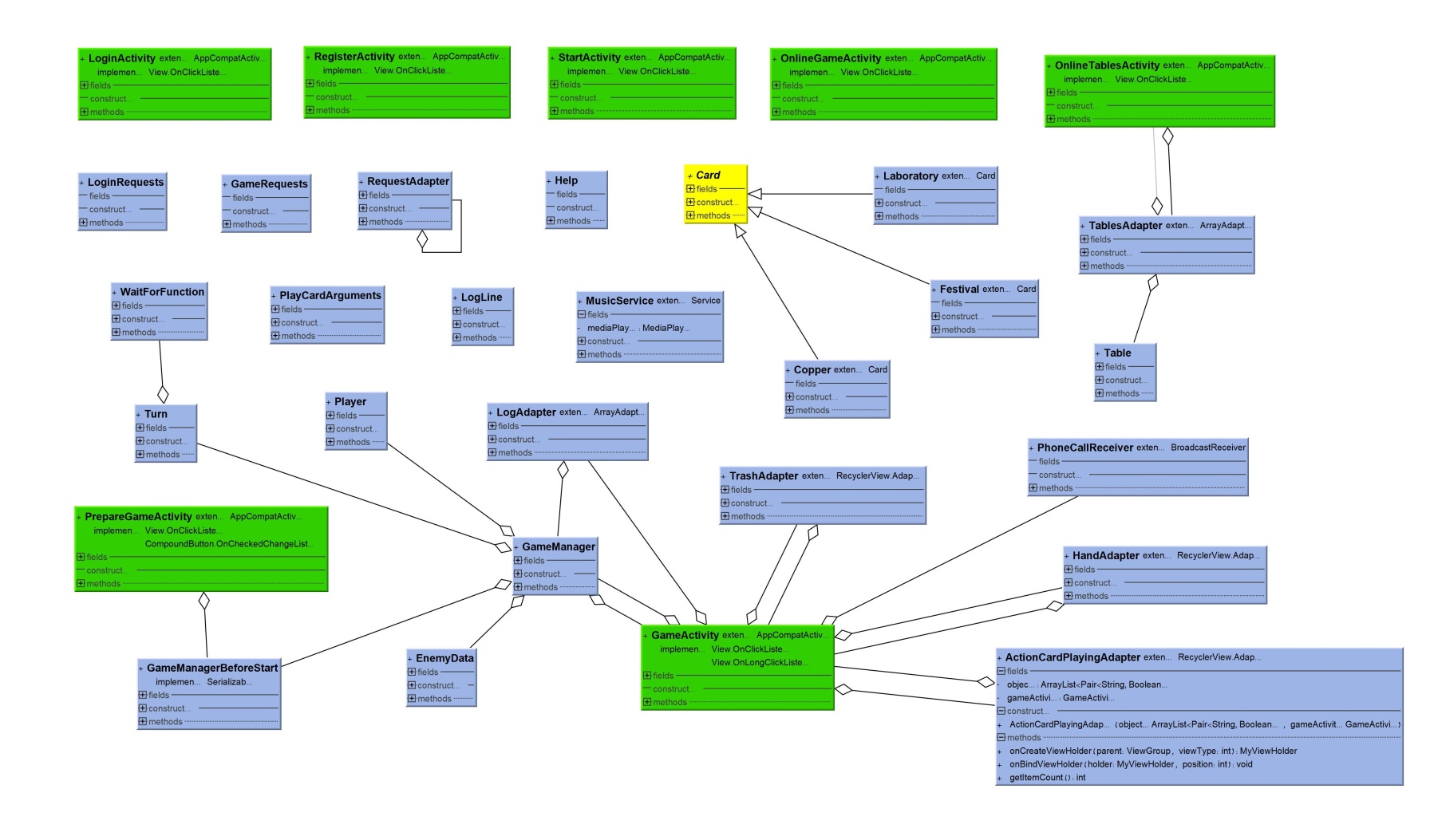
Undo: Request to undo the most recent thing that happened. Your opponent(s) must allow the undo in order for it to happen. While refusing may seem unsportsmanlike, it is every player’s right to refuse an undo request.

Resign: Concede the game, ending it. This counts as a loss for you if it is a rated game.

Next to each player’s deck are their username, and a point counter, which tracks how many Victory points that player would have if the game were to end at that moment.

After a game has ended, you are shown the game end screen. On the left are the options chosen for the Table. On the right are the contents of each player’s deck at the end of the game. In the center is a description of the outcome of the game, including the winner, and the score of each player, including a list of the sources of their Victory points. From here, you can choose to replay the Table by clicking Ready, or you can leave the Table or edit its settings. If all players click Ready, a new game starts with the same players and settings.

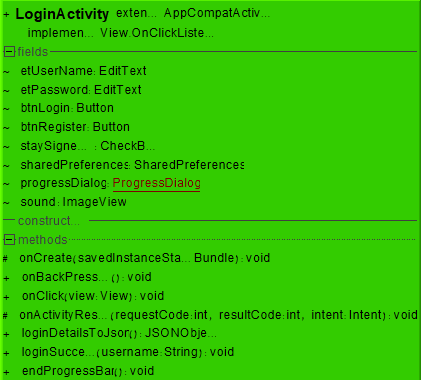
# תרשים UML מקוצר



# פירוט מחלקות עם הסברים

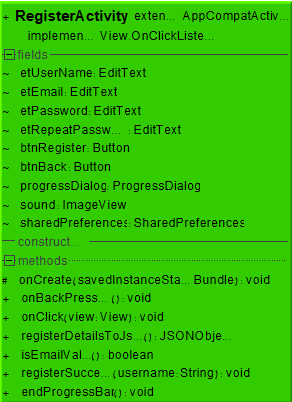
LoginActivity:

מסך התחברות אל האפליקציה שבו נדרש להכניס שם משתמש וסיסמה. אם שם המשתמש והסיסמה נמצאים במערכת, יעבור המשתמש אל מסך הפתיחה (יש אפשרות גם להישאר מחובר). למשתמש תהיה אפשרות לעבור למסך הירשמות אם אינו נמצא במערכת. ה layout של מסך זה הוא activity\_login.



RegisterActivity:

מסך הירשמות הכולל יצירת שם משתמש וסיסמה. בנוסף ייתן המשתמש את המייל (על מנת לפתוח את האפשרות להוסיף "שכחתי סיסמה"). שם המשתמש והסיסמה יתווספו במידה ששם המשתמש לא נמצא כבר במערכת. לאחר מכן יעבור המשתמש אל מסך ההתחברות. כולל כפתור חזרה אל Login Activity. ה layout של מסך זה הוא activity\_register.



מחלקה סטטית שדואגת להעביר לשרת בקשות login ו register, ומקבלת תשובה בהתאם האם ההרשמה או ההתחברות התבצעה בהצלחה או לא. הפעולות במחלקה נקראות מהמסכיםLogin Activity או Register Activity.

LoginRequests:



מסך הפתיחה של האפליקציה. כאשר נכנסים אל האפליקציה בתור מחוברים, מגיעים למסך זה וגם לאחר התחברות. המסך כולל:

- כפתור Online Game

- כפתור Friends

- כפתור Cardslist

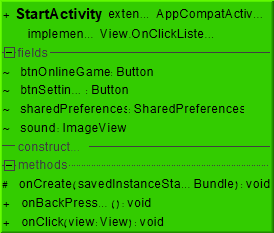
- כפתור Leaderboard

- כפתור Instructions

- כפתור Settings

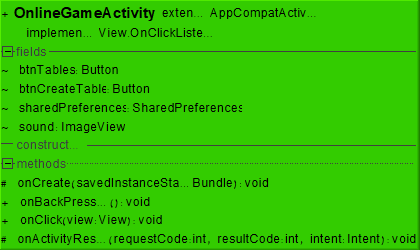
ה layout של מסך זה הוא activity\_start.

StartActivity:



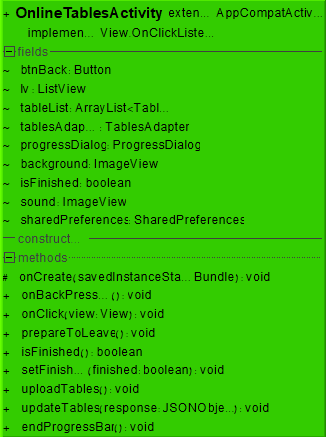
OnlineGameActivity:

מסך עבור מי שרוצה להתחיל משחק. יש לבחור אם רוצים ליצור משחק או להצטרף למשחק קיים. כולל כפתור חזרה אל Start Activity. ה layout של מסך זה הוא activity\_online\_game.



OnlineTablesActivity:

מסך המראה את טבלת המשחקים אונליין. ניתן להצטרף למשחק ולהגיע למסך ההכנה של המשחק, אך לא תהיה אפשרות לערוך את המשחק (מבחינת קלפים או הגדרות נוספות). כולל כפתור חזרה אל Online Game. ה layout של מסך זה הוא activity\_online\_tables.



מחלקה שמעדכנת ב listview שבמסך OnlineTablesActivity ומוכלת בתוכו. המחלקה מכילה את המשחקים שניתן לשחק בהם (נלקח מהשרת) ונותנת אפשרות להיכנס למשחק (יש כפתור play בכל שורה בטבלה). ה layout של כל table הוא table\_layout.

TablesAdapter:

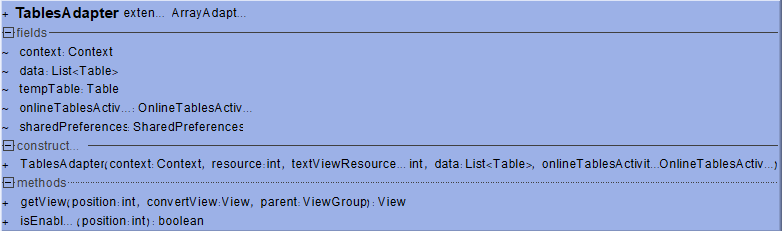
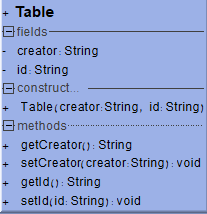


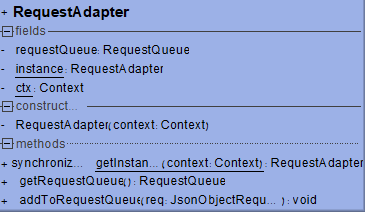
Table:

מחלקה ששומרת את כל התכונות של table (table זה משחק בדומיניון) שהם מי יצר את המשחק ואת ה-id של המשחק. המחלקה מוכלת ב TablesAdapter שבה נשמרים כל המשחקים.



RequestAdapter:

מחלקה שמנהלת את כל בקשות ה http של האפליקציה אשר נקראות מהמחלקות LoginRequests ו GameRequests. היא כוללת RequestQueue שזה תור של בקשות וככה באופן סינכרוני מעלה בקשות לתור ומפעילה בקשה אחר בקשה.



PrepareGameActivity:

מסך שבו רואים את השחקנים שמצטרפים למשחק. ניתן לערוך את המשחק, אם יש לך הרשאה, כלומר אם יצרת את המשחק. אם כל השחקנים המצטרפים מאשרים שהם מוכנים להתחלת המשחק, עוברים אל מסך הGame Activity. אותו מסך משמש גם בסוף המשחק שמסך שבו רואים את תוצאות המשחק. ניתן לראות את טבלת קלפי הניצחון, מי ניצח ומי הפסיד. יש גם אפשרות להתחיל את המשחק מחדש עם אותם אנשים. כולל כפתור חזרה אלOnline Game . ה layout של מסך זה הוא activity\_prepare\_game.



GameManagerBeforeStart:

מחלקה ששומרת בתוכה את ה-id של המשחק, את השמות של שני השחקנים שבמשחק, אם שני השחקנים מוכנים ותכונות נוספות על המשחק. המחלקה מוכלת במחלקותPrepare Game Activity ו GameManager. מחלקה זו אוספת את כל המידע החיוני על המשחק על מנת לתקשר עם השרת, כלומר ברוב בקשות המשחק, מביאים את GameManagerBeforeStart כ json על מנת שהשרת יידע לאן להכניס או מאיפה לשלוף את המידע. בנוסף המחלקה שומרת האם שני השחקנים מוכנים (כלומר אם לחצו על ready במסך Prepare Game Activity והאם לחצו על start במסך Game Activity.

GameActivity:

מסך המשחק שמחולק לחלקים רבים: כל קלפי המשחק, הקלפים שיש לי ביד, כמות הקלפים ב deck עבורי ועבור היריב, הקלף האחרון ב discard עבורי ועבור היריב, נקודות הניצחון עבורי ועבור היריב, log של המשחק שסוקר את כל הפעולות שנעשות, מסך הודעות שניתן לכתוב (עדיין לא פועל) וכפתורים שונים, למשל כפתור כדי להיכנע ולסיים את המשחק. המסך מכיל את GameManager, HandAdapter, TrashAdapter, LogAdapter, ActionCardPlayingAdapter ו PhoneCallReceiver.

ה layout של מסך זה הוא activity\_game שכולל בתוכו עוד כמה layouts: card\_sh (לקלפים שעל board) card\_kingdom (לקלפים שבאיזור ה kingdom), card\_dialog (לקלף המוגדל שמוצג בדיאלוג)

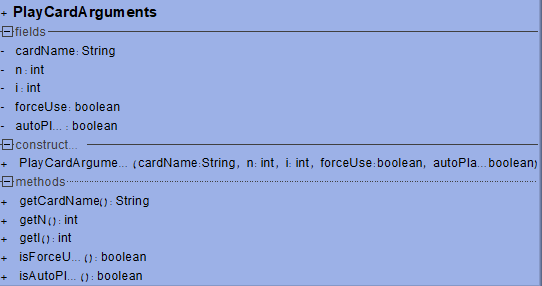
GameManager:

מחלקה ששומרת בתוכה את כל המידע על המשחק עבור כל שחקן ומכילה מחלקות רבות, כמו Player, Turn, GameManagerBeforeStart ועוד. המחלקה מכילה ומוכלת במסךGame Activity. המחלקה מנהלת ושומרת את כל המידע לכל שחקן במשחק ודואגת לעדכן את השרת על ידי קריאות למחלקה GameRequests ולשלוף מהשרת את המידע הרלוונטי. המחלקה דואגת לתחילת המשחק ובודקת את סיום המשחק, היא דואגת להחליף בין התורות ולעדכן את ה log של המשחק לכל שחקן. יש במחלקה זו גם קבועים רבים שקובעים את אופי המשחק וניתן לשנות אותם. בנוסף, המחלקה מגרילה בתחילת המשחק את קלפי הפעולה שישוחקו במשחק.



PlayCardArguments:

מחלקה ששומרת את המידע על קלף על מנת להפעילו בעתיד. הקלף יופעל על ידי קריאה לפעולה ב GameManager והפרמטרים שיישלחו לפעולה הם הפרמטרים שנשמרים כאן. זאת על מנת שקלפים יופעלו רק אחד אחרי השני ולא בו זמנית בטעות.



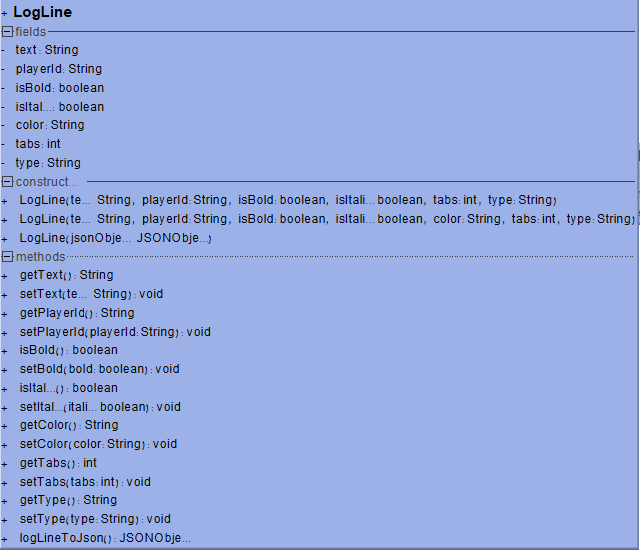
GameRequests:

מחלקה סטטית שמעלה בקשות בנוגע למשחק לשרת, למשל קבלת מידע ב"real time" והעלאת מידע. הפעולות במחלקה זו נקראות בעיקר מ GameManager והקישור בין השרת ללקוח, כלומר בין שני השחקנים במהלך המשחק, לפניו ואחריו.



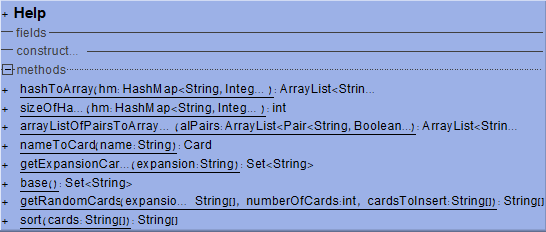
LogLine:

מחלקה שכוללת את כל המידע הנדרש על שורה ב log. במחלקה זו משתמשים ב GameManager וב LogAdapter כ ArrayList של LogLine על מנת לסנכרן ולהתאים את ה log עבור שני השחקנים. כל שורה כוללת את הטקסט שלה, מי השחקן שהפעיל אותה, באיזה צבע תופיע ועוד.



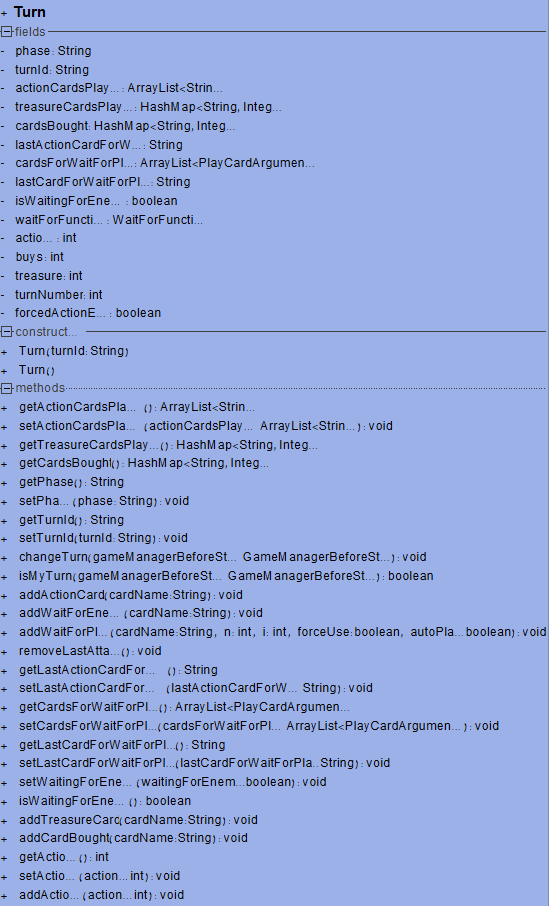
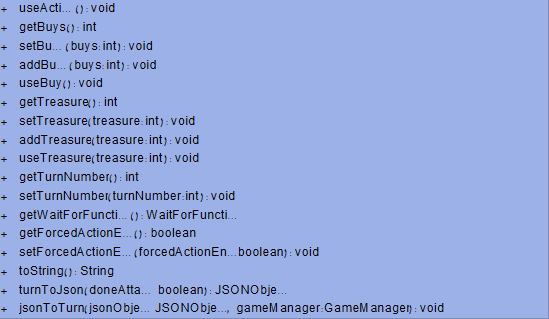
Help:

מחלקה סטטית שכוללת פעולות כלליות במשחק, ביניהן למשל פעולה שמחזירה כמות הקלפים שיש ב hashmap שהוא סכום של כל הערכים. בנוסף, במחלקה זו יש פעולה חשובה מאוד שממירה משם של קלף למחלקה של הקלף עצמו (על ידי switch case). בנוסף יש פעולה אשר יוצרת חבילת קלפים רנדומלית למשחק חדש.



Turn:

מחלקה שמוכלת במחלקה GameManager ושומרת את המידע הרלוונטי לתור. בתוכה יש פעולה changeTurn שמשנה את התור ומאפסת את הערכים ב Turn כדי שכל תור יהיה חדש. הערכים ב Turn עולים ב Game Activity ל text view שמראה את המידע הרלוונטי לשחקן.



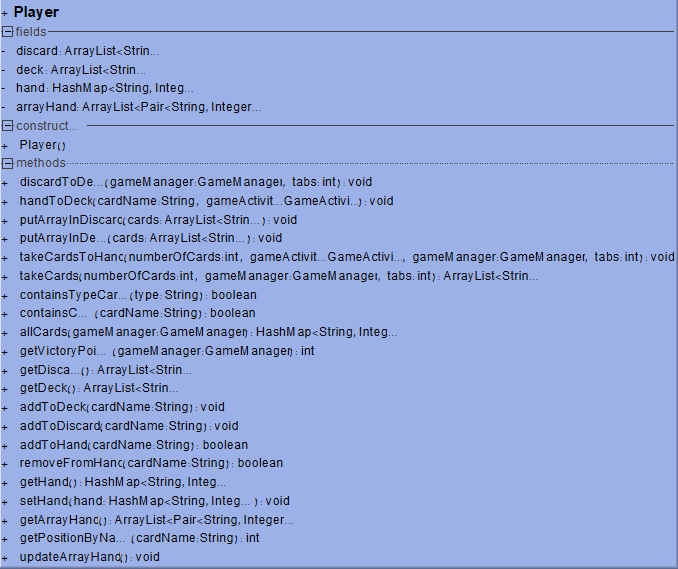
WaitForFunction:

מחלקה שמוכלת במחלקה Turn ושומרת את המידע הרלוונטי לכל הדברים שבהם יש לחכות במשחק, למשל לשחק היריב, ללחיצות מסויימות ועוד. המשתנים במחלקה זו מגבילים את המשחק מלהמשיכו בכך שיש בדיקה בכל פעולה האם השחקן מחכה למשהו ורק אם הוא לא מחכה, הוא יכול להמשיך לשחק.



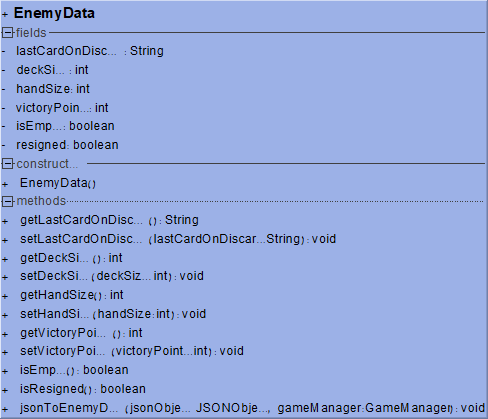
Player:

מחלקה שמוכלת במחלקה GameManager ושומרת את המידע הרלוונטי לכל שחקן לגבי הקלפים שיש לו – הקלפים ביד, ב discard וב deck. כאן יש מידע "סודי" שהשחקן השני לא אמור לדעת וחלק מהמידע גם אותו שחקן לא צריך לדעת ולכן לא מעלים אותו לשרת, אלא הוא נשמר במחלקה הזו. המחלקה דואגת גם לקחת קלפים ליד ולעוד פעולות שמתבצעות במהלך המשחק.



EnemyData:

מחלקה שמוכלת במחלקה GameManager ושומרת את המידע הרלוונטי על כל שחקן, אבל רק המידע שכל שחקן במשחק אמור לדעת, למשל כמות הקלפים שיש לשחקן ביד והקלף האחרון ב discard. מידע זה מועלה לשרת ונשלף מהשחקן השני על מנת לעדכן על המסך GameActivity בכל המידע הנדרש על היריב.

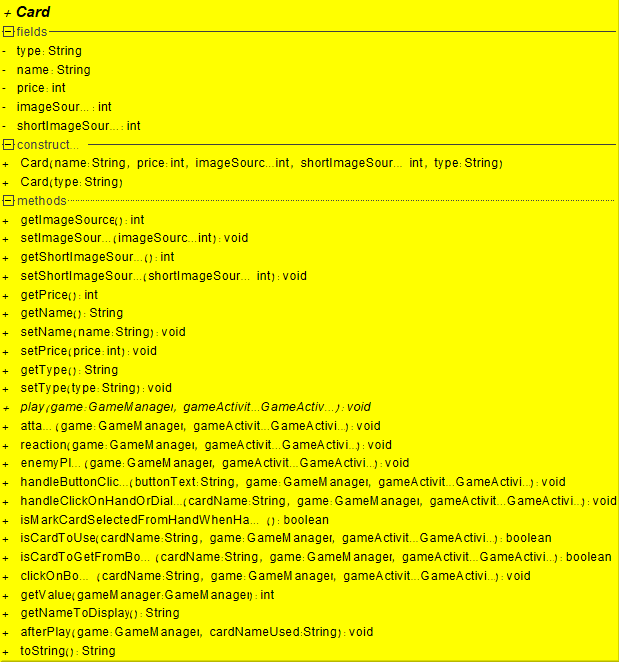


Card:

מחלקה אבסטרקטית של קלף שכוללת את הסוג שלו, השם שלו, המחיר שלו ועוד. במחלקה זו יש פעולה אבסטרקטית, play שממומשת עבור כל הקלפים שיורשים מהמחלקה. מהמחלקה יורש כל סוג קלף במשחק (יש כאן דוגמות לכמה קלפים).

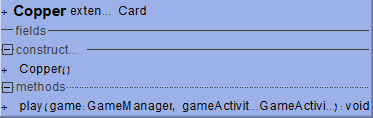
יש במחלקה פעולות רבות נוספות שניתנות לדריסה על ידי קלפים שיורשים מ Card.

ב GameManager ובמחלקות שהן תכונה שלו נשמר אך ורק שם הקלף. על מנת להפעיל פעולה במחלקה של קלף מסויים, קיימת הפעולה nameToCard שנמצאת ב Help והיא ממירה משם הקלף למחלקה של הקלף שיורש מ Card.



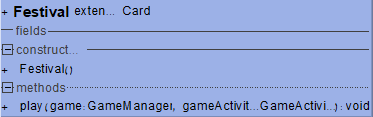
Copper:

מחלקה שיורשת מ Card ודורסת את הפעולה האבסטרקטית play. זו דוגמה לקלף פשוט במשחק.



Festival:

מחלקה שיורשת מ Card ודורסת את הפעולה האבסטרקטית play. זו דוגמה לקלף פשוט במשחק.



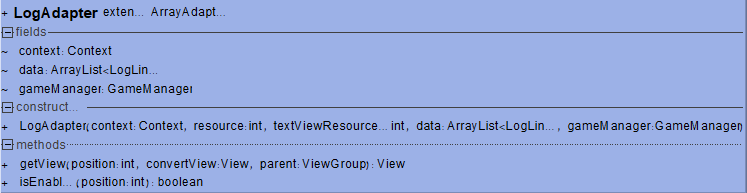
Laboratory:

מחלקה שיורשת מ Card ודורסת את הפעולה האבסטרקטית play. זו דוגמה לקלף פשוט במשחק.



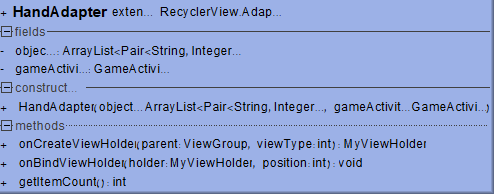
LogAdapter:

מחלקה שמעדכנת ב listview שבמסך Game Activity את כל שורות ה log שבמשחק ומעצבת כל שורה בהתאם לתכונות שלה ב LogLine. המחלקה מוכלת ב Game Activity ומכילה מצביע ל GameManager. ה layout של כל שורה הוא log\_layout.



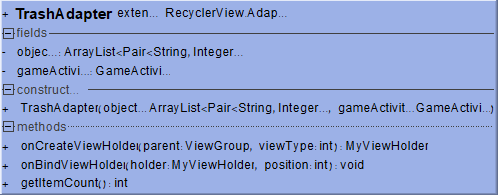
HandAdapter:

מחלקה שמעדכנת ב recyclerview שבמסך Game Activity את כל הקלפים שיש לשחקן ביד. המחלקה מוכלת ב Game Activity ומכילה מצביע לGame Activity. ה layout של כל סוג קלף ביד הוא card.



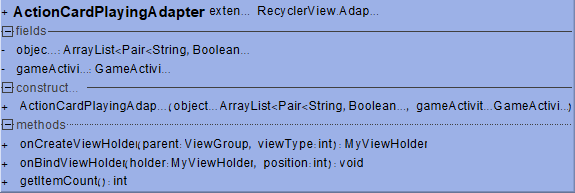
TrashAdapter:

מחלקה שמעדכנת ב - recyclerview שבמסךGame Activity את כל הקלפים שיש ב trash במשחק. המחלקה מוכלת ב Game Activity ומכילה מצביע ל Game Activity. ה layout של כל סוג קלף ב trash הוא card\_trash.



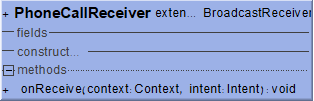
ActionCardPlayingAdapter:

מחלקה שמעדכנת ב recyclerview שמועלה על המסך Game Activity אם מופעל סוג מסוים של קלפי פעולה ומעדכן את הקלפים שם בהתאם לקלף שהופעל. המחלקה מוכלת ב Game Activity ומכילה מצביע ל Game Activity. ה layout של כל סוג קלף בדיאלוג זה הוא card\_for\_action\_cards\_playing.



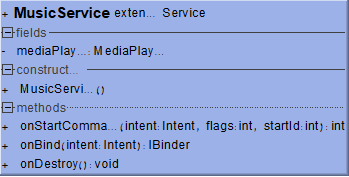
PhoneCallReceiver:

מחלקה שיורשת מ Broadcast Reciever ומאזינה לשינויים במצב של הטלפון. כאשר המצב הוא שמישהו מצלצל, אז הפעולה onReceive מופעלת, מנתקת אוטומטית את השיחה אם המסך הוא GameActivity ושולחת הודעה למי שהתקשר שהוא יחזור אליו אחר כך. המחלקה מוכלת ב Game Activity.



MusicService:

מחלקה שיורשת מService ומפעילה מוזיקת רקע. בכל Activity מלבד GameActivity ניתן להפעיל ולכבות את המוסיקה על ידי כפתור בצד המסך.



מסכים נוספים שיתווספו ועוד לא קיימים

מסך Friends - מסך שבו ניתן לראות את החברים שיש לך במשחק. ישנה אפשרות להוסיף חבר על פי שם משתמש ולמחוק חבר. כולל כפתור חזרה אל Start Activity.

מסך Cardslist - מסך שמראה את כל הקלפים במשחק, כולל תיאור של כל קלף. ניתן יהיה לסמן קלפים שאוהבים וקלפים שלא אוהבים על מנת שבמשחק אונליין ישחקו רק עם הקלפים שאוהבים. כולל כפתור חזרה אל Start Activity.

מסך Leaderboard - מסך שכולל טבלת מובילים לפי ניצחונות והפסדים. כולל כפתור חזרה אל Start Activity.

מסך Instructions - מסך ההוראות של המשחק. כולל כפתור חזרה אל Start Activity.

מסך Settings - מסך ההגדרות. כולל הגדרות כלליות, הגדרות של המשחק - autoplay (יש לדעת את החוקים כדי להבין) והגדרות החשבון. כולל כפתור חזרה אל Start Activity.

מסך Choose Cards - מסך שבו ניתן לבחור את הקלפים עבור המשחק (בדר"כ זה רנדומלי). יחד עם אופציות לשחק רק עם הקלפים האהובים וללא הקלפים שלא אוהבים. כולל כפתור אישור שמחזיר אלPrepare Table .

# אלגוריתמים מעניינים

1. takeCardsToHand - פעולה שנמצאת במחלקה Player ומטרתה לקחת קלפים בסוף כל תור מה-deck אל היד:

*/\*\*  
 \* A function that takes cards from deck to hand.  
 \** ***@param*** *numberOfCards An Integer with the number of cards to take  
 \** ***@param*** *gameActivity A reference to gameActivity  
 \** ***@param*** *gameManager A reference to gameManager  
 \** ***@param*** *tabs An Integer which is the count of tabs in log that  
 \* the line that would be added will have  
 \*/*public void takeCardsToHand(int numberOfCards, GameActivity gameActivity, GameManager gameManager, int tabs) {  
 for (int i = 0; i < numberOfCards && !(this.discard.isEmpty() && this.deck.isEmpty()); i++) {  
 if (this.deck.isEmpty())  
 discardToDeck(gameManager, tabs);  
 if (this.hand.containsKey(this.deck.get(this.deck.size() - 1)))  
 this.hand.put(this.deck.get(this.deck.size() - 1), this.hand.get(this.deck.get(this.deck.size() - 1)) + 1);  
 else  
 this.hand.put(this.deck.get(this.deck.size() - 1), 1);  
 // removes the last index which is the first card to take from deck  
 this.deck.remove(this.deck.size() - 1);  
 }  
 this.updateArrayHand();  
 gameActivity.getHandAdapter().notifyDataSetChanged();  
  
 gameManager.getLog().add(new LogLine("Draws " + numberOfCards + " card" + (numberOfCards == 1 ? "" : "s"), gameManager.getGameManagerBeforeStart().getMyId(), false, false, tabs, "take cards"));  
}

הפעולה עוברת בלולאת for ככמות הקלפים שצריך לקחת בתחילת כל תור (קבוע שנקרא numberOfCards ובדרך כלל שווה 5). הלולאה תעבור ככמות numberOfCards אלא אם גם ה-discard ריק וגם ה-deck והמשמעות היא שלא נותרו לו קלפים לקחת, ובמצב כזה יש לסיים את הלולאה. בתוך הלולאה בודקים אם ה-deck ריק ואם כין אז קוראים לפעולה discardToDeck שמעבירה את כל הקלפים מה-deck ל-discard כמו שצריך לעשות לפי החוקים. לאחר מכן הפעולה בודקת האם ביד של השחקן כבר קיים קלף מהסוג שנשלף מה-deck (הקלף האחרון שב-deck). במקרה שאכן הוא קיים, הפעולה פשוט מוסיפה אחד לערך ב hashmap של אותו קלף שנשלף, אחרת הפעולה יוצרת ערך חדש ב hashmap בשם של הקלף שעומד להישלף ומכניסה לערך אחד, כי יש רק קלף אחד כזה. כשהלולאה נגמרת ושנשלפו הקלפים, הפעולה מעדכנת את הקלפים שביד על מנת שיראו על המסך שנשלפו (notifyDataSetChanged()) ולבסוף מוסיפה ל log של המשחק שאותו שחקן לקח 5 קלפים.

1. playCard – פעולה שנמצאת ב GameManager ומפעילה את הקלף שלחץ עליו השחקן:

*/\*\*  
 \* A function that plays a card and adds to log.  
 \** ***@param*** *cardName A String which is the name of the card which should be used  
 \** ***@param*** *n An Integer which is the number of times that the card should be played  
 \** ***@param*** *i An Integer which is the place of the card in the wait queue  
 \** ***@param*** *forceUse A Boolean which is true if the use of the card was forced or not  
 \** ***@param*** *autoPlay A Boolean which is true if useCard was played with autoPlay and false if not  
 \*/*public void playCard(String cardName, int n, int i, boolean forceUse, boolean autoPlay) {  
 if (this.player.getHand().containsKey(cardName) && (autoPlay || n == 1 || i == 0)) {  
 this.player.removeFromHand(cardName);  
 this.player.updateArrayHand();  
 gameActivity.getHandAdapter().notifyDataSetChanged();  
 }  
  
 if (Help.*nameToCard*(cardName).getType().equals("action") && !forceUse)  
 this.log.add(new LogLine("Plays a " + Help.*nameToCard*(cardName).getNameToDisplay(), this.turn.getTurnId(), false, false, this.getTabs() - 1, "use action"));  
  
 else if (forceUse || (!autoPlay && n > 1 && i == 0)) // throne room first or vassal  
 this.log.add(new LogLine("Plays a " + Help.*nameToCard*(cardName).getNameToDisplay(), this.turn.getTurnId(), false, false, this.getLastLineFromLog().getTabs() + 1, "use action"));  
  
 else if ((!autoPlay && n > 1)) // not first played in throne room  
 this.log.add(new LogLine("Plays a " + Help.*nameToCard*(cardName).getNameToDisplay(), this.turn.getTurnId(), false, false, this.getLastLineFromLog().getTabs() - 1, "use action"));  
  
 if (Help.*nameToCard*(cardName).getType().equals("action") && n == 1 && !forceUse)  
 this.getTurn().useAction();  
  
 if (Help.*nameToCard*(cardName).getType().equals("action") && i == 0) {  
 this.turn.addActionCard(cardName);  
 }  
 else if (Help.*nameToCard*(cardName).getType().equals("treasure")) {  
 this.turn.addTreasureCard(cardName);  
 this.addHashToLog(this.turn.getTreasureCardsPlayed(), "use treasure", "Plays");  
 }  
  
 Help.*nameToCard*(cardName).play(this, gameActivity);  
}

הפעולה מורידה לאותו שחקן שהפעיל את הקלף את אותו קלף מהיד, מוסיפה אל ה log על כך שהשחקן השתמש בקלף הזה (יש פיצול לשלושה תנאים שלא אפרט כי הם קשורים לחוקי המשחק עצמו, אבל באופן כללי דואגים למספר הטאבים שצריך ב log, כלומר אם זו הפעלה של קלף שנבעה מהפעלה קודמת או הפעלה לאחר ששמו throne room). הפעולה מורידה לשחקן פעולה אחת כיוון שהשתמש בקלף בתנאי שהקלף לא הופעל ב"הכרחה", כלומר שלא צריך להוריד על כך פעולה. לבסוף, הפעולה מוסיפה אל הקלפים ששוחקו את אותו קלף בהתאם לסוגו (treasure, action...). לבסוף הפעולה מפעילה את הקלף על ידי play שפועלת על ה class המתאים לשם של הקלף אשר יורש מ Card. מכיוון שכל המחלקות של הקלפים יורשות מ Card, ניתן פשוט להפעיל על כל קלף את הפעולה play והיא תבצע את הפעולה שנדרסה במחלקה הספציפית של הקלף.

1. get\_and\_upload\_player\_data – פעולה מהשרת שמעלה מידע על השחקן כדי להעביר אל היריב ומקבלת מידע על השחקן השני:

'''  
A function that uploads the data about the player that sent the request  
and returns the data about the other player.  
'''  
@app.route('/get\_and\_upload\_player\_data', methods=['POST'])  
def get\_and\_upload\_player\_data():  
 data = request.json  
 game\_id = get\_games\_ref().child(data.get("gameManagerBeforeStart").get("gameId"))  
 my\_id = data.get("gameManagerBeforeStart").get("idP1")  
 enemy\_id = data.get("gameManagerBeforeStart").get("idP2")  
 if not data.get("gameManagerBeforeStart").get("isCreator"):  
 my\_id, enemy\_id = enemy\_id, my\_id  
 game\_id.child(my\_id).set(data.get("myData"))  
  
 enemy\_data = game\_id.child(enemy\_id).get()  
 if enemy\_data is None:  
 return {"success": False}  
  
 return {\*\*enemy\_data, \*\*{"success": True}}

הפעולה מתקבלת בהעלאת בקשת http מסוג post מהלקוח אבל בפועל מתרחשת לולאה שחוזרת על עצמה מתחילת המשחק ונגמרת רק בסופו. הפעולה מקבלת json שכולל את gameManagerBeforeStart ששם יש את ה-id של המשחק בבסיס הנתונים ואת השם של כל שחקן. בנוסף קיים המידע על השחקן שהעלה את הבקשה והוא מועלה לבסיס הנתונים במקום שמור שהמפתח שלו הוא שמו של השחקן. לאחר מכן הפעולה מקבלת מבסיס הנתונים את המידע על השחקן השני (רק המידע שמותר לו לדעת מן הסתם). אם הוא מצא את המידע, הוא מחזיר json של המידע יחד עם success עם ערך true, כלומר שהוא הצליח להביא את המידע ואם הוא לא מצליח, אז מוחזר json עם ערך ב success של false.

# שימוש ב-Resources

## Broadcast Receiver

המחלקה PhoneCallReceiver יורשת מ - BroadcastReceiver ויש בה את הפעולה onReceive שהיא פעולה שחובה לדרוס כאשר יורשים מ BroadcastReceiver. המחלקה מאזינה לשינויים במצב השיחות של הטלפון, כלומר אם מישהו מתקשר למשל, תופעל הפעולה onReceive. ברגע שמופעלת הפעולה, קולטים מהטלפון את מצב השיחה. אם המצב הנוכחי הוא CALL\_STATE\_RINGING שהוא קבוע שאומר שהמצב הוא שמישהו מתקשר, אז עושים את הפעולות הבאות:

ראשית, אם הגרסת SDK של הטלפון גדולה או שווה לזו של LOLLIPOP, אז ניתן להפעיל פעולה שנקראת endCall() שמופעלת על TelecomManager שנשלף מהטלפון ומנתקת את השיחה באופן אוטומטי (גרסאות קודמות לא מאפשרות את זה, לכן יש בדיקה של הגרסה). בנוסף, הפעולה שולחת הודעת SMS לאותו אדם שהתקשר "Sorry, I will call you back later because I am playing Dominion right now." ומפעילה Toast אצל המשתמש באפליקציה שנשלחה הודעה.

מטרת ה Broadcast היא שבמהלך המשחק לא יפריעו למשתמש עם שיחות נכנסות, מה שעלול להקשות עליו ולהוציא אותו מריכוז. כך נוח יותר לשחק בעיני, כאשר מיידעים את מי שהתקשר שהוא עסוק כרגע ויחזור אליו אחר כך, ולכן יצרתי את אותו Broadcast Receiver.

המחלקה מופעלת ב GameActivity כאשר מייצרים מופע שלה ומפעילים את הפעולה registerReceiver. אך לפני כן, נדרשות מהמשתמש הרשאות על מנת לעשות זאת. הרשאות אלו מתבקשות גם ב manifest אך המשתמש צריך לאשר אותן כדי להפעיל את ה Broadcast. ההרשאות הנדרשות הן: READ\_PHONE\_STATE, SEND\_SMS, READ\_CALL\_LOG.

## Service

המחלקה MusicService היא מחלקה שיורשת מ Service ומפעילה את מוסיקת הרקע כ -Service, כלומר היא תפעל גם אם לא נכנסים לאפליקציה כל עוד היא עדיין פתוחה. במחלקה יש את הפעולה onStartCommand שפועלת כאשר מפעילים את הService , את הפעולה onBind(Intent intent) שהיא פעולה שחובה לממש אך אצלי לא עושה כלום ואת הפעולה onDestroy() שפועלת כשמכבים את ה Service.

בכל מסך, מלבד מסך המשחק, יש תמונה של מוסיקה שניתן ללחוץ עליה ולהדליק ולכבות את ה Service, כך ניתן להפעיל ולבטל את מוסיקת הרקע כרצונו של המשתמש. בנוסף, המצב של המוסיקה (כבוי או דלוק) נשמר ב SharedPreferences, לכן, כאשר יסגרו את האפליקציה ויפתחו מחדש, המצב יהיה כמו שנקבע בפעם האחרונה שהשתמשו באפליקציה.

## Thread

הפעולות שפונות אל השרת כבקשות http מופעלות באופן אוטומטי על thread אחר, כאשר היא מחכה לתשובה מהשרת. לאחר יצירת הבקשה, מכניסים את הבקשה לתור של בקשות http שנקרא מהמחלקה RequestAdapter. לאחר שליחת הבקשה, מחכים לתשובה מהשרת שקורא לפעולה onResponse, אך הקוד ממשיך לרוץ ואינו מחכה לחזרת התשובה, לכן הוא פועל אוטומטית עלthread אחר.

## FireBase

השרת בפרוייקט נעזר בבסיס נתונים שנקרא firebase ומעלה אליו את כל המידע ושולף ממנו. המבנה של בסיס הנתונים הוא מבנה של json, כלומר לפי key ו value (NoSQL). המבנה של השרת דינמי ולא חייב להיות קבוע מראש אך בכל זאת יש לשרת שלי תבנית מרכזית שעל פיה הוא פועל, יחד עם שמות קוד שהשרת יודע כדי לשלוף את המידע המתאים בכל פעם.

השרת מעלה את כל המידע לשירות של firebase שנקרא realtime database שמאפשר העלאת מידע על ידי הלקוחות בזמן אמת ובאופן סינכרוני והמידע נשאר זמין גם כאשר האפליקציה כבויה. הנתונים ב firebase מחולקים למשחקים (games) ולמשתמשים (users), כאשר במשחקים נמצאים כל המשחקים שמשוחקים כרגע ובמשתמשים יש את כל המשתמשים שנרשמו אי פעם יחד עם הסיסמות והמידע עליהם.

על מנת לממש את העלאת המידע ושליפתו מה firebase דרך הפייתון, השתמשתי בסיפרייה של פייתון אשר נקראת firebase\_admin שמאפשרת בצד השרת לשלב את firebase. בסיפרייה יש פעולות רבות שמאפשרות העלאה, שליפה, מחיקה ועדכון מידע בבסיס הנתונים בקלות ובמהירות.

מבנה הנתונים ב firebase נראה כך (עמוד הבא):



כל המשתמשים שנרשמו אי פעם

כל המשחקים שפועלים כרגע

# טבלת פונקציות בפייתון

|  |  |
| --- | --- |
| הפונקציה | הסבר |
| @app.route('/creator\_start', methods=['GET']) def creator\_start() | פעולה שמקבלת בקשת HTTP מסוג GET שיוצרת משחק בשרת עם הפעולה push שיוצרת מפתח רנדומלי וייחודי (אין עוד אחד כמוהו). הפעולה מחזירה json שיש בו את מפתח המשחק והודעה בהתאם להצלחה או כישלון. |
| @app.route('/non\_creator\_start', methods=['POST']) def non\_creator\_start() | פעולה שמקבלת בקשת HTTP מסוג POST שבה json עם gameManagerBeforeStart שבו יש את המפתח הייחודי של המשחק, את שמות השחקנים ועוד. הפעולה מוסיפה את השחקן למשחק אם אין שני שחקנים כבר במשחק. הפעולה מחזירה json שיש בו הודעה בהתאם להצלחה או כישלון. |
| @app.route('/upload\_all\_data', methods=['POST']) def upload\_all\_data() | פעולה שמקבלת בקשת HTTP מסוג POST שבה json עם gameManagerBeforeStart שבו יש את המפתח הייחודי של המשחק, את שמות השחקנים ועוד ועם המידע הרלוונטי על המשחק לפני תחילת המשחק: קלפי פעולה, מי מתחיל ועוד מידע על מנת שהשחקן שלא יצר את המשחק יוכל לקבל אותו. הפעולה מעלה את כל המידע הזה לשרת. הפעולה מחזירה json שיש בו הודעה בהתאם להצלחה או כישלון. |
| @app.route('/upload\_real\_time', methods=['POST']) def upload\_real\_time() | פעולה שמקבלת בקשת HTTP מסוג POST שבה json עם gameManagerBeforeStart שבו יש את המפתח הייחודי של המשחק, את שמות השחקנים ועוד ועם המידע הרלוונטי על המשחק אחרי תחילת המשחק. הפעולה מעלה את כל המידע הזה לשרת. מי שמפעיל את הפעולה הוא מי שהתור שלו. הפעולה מחזירה json שיש בו הודעה בהתאם להצלחה או כישלון. |
| @app.route('/get\_all\_data', methods=['POST']) def get\_all\_data() | פעולה שמקבלת בקשת HTTP מסוג POST שבה json עם gameManagerBeforeStart שבו יש את המפתח הייחודי של המשחק, את שמות השחקנים ועוד. הפעולה מקבלת את כל המידע הרלוונטי על המשחק לפני תחילת המשחק. הפעולה מחזירה json שיש בו את המידע הזה והודעה בהתאם להצלחה או כישלון. |
| @app.route('/get\_real\_time', methods=['POST']) def get\_real\_time() | פעולה שמקבלת בקשת HTTP מסוג POST שבה json עם gameManagerBeforeStart שבו יש את המפתח הייחודי של המשחק, את שמות השחקנים ועוד. הפעולה הפעולה מקבלת את כל המידע הרלוונטי על המשחק אחרי תחילת המשחק. מי שמפעיל את הפעולה הוא מי שהתור לא שלו כדי לקבל את המידע על מה שהיריב עושה. הפעולה מחזירה json שיש בו את המידע הזה והודעה בהתאם להצלחה או כישלון. |
| @app.route('/get\_and\_upload\_player\_data', methods=['POST']) def get\_and\_upload\_player\_data() | פעולה שמקבלת בקשת HTTP מסוג POST שבה json עם gameManagerBeforeStart שבו יש את המפתח הייחודי של המשחק, את שמות השחקנים ועוד ועם המידע הרלוונטי על השחקן שהביא את הבקשה שהשחקן השני צריך לדעת. הפעולה מעלה את המידע הזה לשרת ומקבלת את אותו המידע הרלוונטי על השחקן השני. הפעולה מחזירה json שיש בו את המידע הזה והודעה בהתאם להצלחה או כישלון. |
| @app.route('/get\_game\_manager\_before\_start', methods=['POST']) def get\_game\_manager\_before\_start() | פעולה שמקבלת בקשת HTTP מסוג POST שבה json עם gameManagerBeforeStart שבו יש את המפתח הייחודי של המשחק, את שמות השחקנים ועוד. הפעולה מקבלת מידע מעודכן על gameManagerBeforeStart. הפעולה מחזירה json שיש בו את המידע הזה והודעה בהתאם להצלחה או כישלון. |
| @app.route('/upload\_game\_manager\_before\_start', methods=['POST']) def upload\_game\_manager\_before\_start() | פעולה שמקבלת בקשת HTTP מסוג POST שבה json עם gameManagerBeforeStart שבו יש את המפתח הייחודי של המשחק, את שמות השחקנים ועוד. הפעולה מעלה את המידע על gameManagerBeforeStart אל השרת. הפעולה מחזירה json שיש בו הודעה בהתאם להצלחה או כישלון. |
| @app.route('/update\_ready', methods=['POST']) def update\_ready() | פעולה שמקבלת בקשת HTTP מסוג POST שבה json עם gameManagerBeforeStart שבו יש את המפתח הייחודי של המשחק, את שמות השחקנים ועוד. הפעולה מעדכנת את השרת במידע על אותו שחקן שהעלה את הבקשה – אם הוא מוכן לעבור למסך המשחק או לא. הפעולה מחזירה json שיש בו הודעה בהתאם להצלחה או כישלון. |
| @app.route('/update\_ready\_to\_start', methods=['POST'])  def update\_ready\_to\_start() | פעולה שמקבלת בקשת HTTP מסוג POST שבה json עם gameManagerBeforeStart שבו יש את המפתח הייחודי של המשחק, את שמות השחקנים ועוד. הפעולה מעדכנת את השרת במידע על אותו שחקן שהעלה את הבקשה – אם הוא מוכן להתחיל את המשחק או לא. הפעולה מחזירה json שיש בו הודעה בהתאם להצלחה או כישלון. |
| @app.route('/delete\_game\_manager\_before\_start', methods=['POST']) def delete\_game\_manager\_before\_start() | פעולה שמקבלת בקשת HTTP מסוג POST שבה json עם gameManagerBeforeStart שבו יש את המפתח הייחודי של המשחק, את שמות השחקנים ועוד. הפעולה מוחקת את אותו משחק מהשרת. הפעולה מחזירה json שיש בו הודעה בהתאם להצלחה או כישלון. |
| @app.route('/delete\_p2', methods=['POST']) def delete\_p2() | פעולה שמקבלת בקשת HTTP מסוג POST שבה json עם gameManagerBeforeStart שבו יש את המפתח הייחודי של המשחק, את שמות השחקנים ועוד. הפעולה מוחקת את אותו שחקן מהמשחק. הפעולה מחזירה json שיש בו הודעה בהתאם להצלחה או כישלון. |
| @app.route('/get\_tables', methods=['GET']) def get\_tables() | פעולה שמקבלת בקשת HTTP מסוג GET ששולפת מהשרת את כל המשחקים שמשוחקים כרגע. הפעולה מחזירה json שיש בו את כל המשחקים עם המפתחות הייחודיים שלהם, יחד עם יוצר המשחק והודעה בהתאם להצלחה או כישלון. |
| @app.route('/end\_game', methods=['POST']) def end\_game() | פעולה שמקבלת בקשת HTTP מסוג POST שבה json עם gameManagerBeforeStart שבו יש את המפתח הייחודי של המשחק, את שמות השחקנים ועוד. הפעולה מוחקת את כל המידע על המשחק ובכך מוכנה למשחק חדש. הפעולה מחזירה json שיש בו הודעה בהתאם להצלחה או כישלון. |
| @app.route('/register', methods=['POST']) def register() | פעולה שמקבלת בקשת HTTP מסוג POST שבה json עם המידע על המשתמש שניסה להירשם. הפעולה בודקת אם השם משתמש כבר קיים במערכת או לא ורושמת את השחקן בשרת אם הוא אינו קיים. הפעולה מחזירה json שיש בו הודעה בהתאם להצלחה או כישלון ואת השם משתמש של השחקן בתנאי שההרשמה הצליחה. |
| @app.route('/login', methods=['POST']) def login() | פעולה שמקבלת בקשת HTTP מסוג POST שבה json עם המידע על המשתמש שניסה להתחבר. הפעולה בודקת אם השם משתמש קיים במערכת או לא ואם כן, בודקת שהסיסמה מתאימה. הפעולה מחזירה json שיש בו הודעה בהתאם להצלחה או כישלון ואת השם משתמש של השחקן בתנאי שההתחברות הצליחה. |
| def get\_games\_ref() | פעולה שמחזירה את ה reference למקום ב -database שבו נשמר המידע על המשחקים. |
| def get\_users\_ref() | פעולה שמחזירה את ה reference למקום ב -database שבו נשמר המידע על המשתמשים. |

# רפלקציה

הפרוייקט היה לי מאוד מעניין ותרם לי המון לידע, הן מבחינה תכנותית והן מבחינת ניהול פרוייקטים גדולים. מכיוון שזו האפליקציה הרצינית הראשונה שהכנתי, התמודדתי עם קשיים רבים אשר הקשו עליי בהמשך הפרוייקט. נעזרתי המון באינטרנט ובסוף הצלחתי למצוא תשובות ופתרונות לבעיות שהיו לי. אני חושב שהקשיים האלו יעזרו לי בעתיד להכין פרוייקטים טובים יותר עם פחות קשיים ממה שהיו לי בפרוייקט הזה. למרות זאת, לדעתי ניהלתי את הזמן כמו שצריך ולא מיהרתי ישר להתחיל לכתוב קוד, כי קודם חשוב לארגן ולתכנן את האפליקציה.

בתחילת העבודה, היה לי קצת קשה. לא היה לי כוח, ולא ידעתי כיצד להתחיל לכתוב את הקוד, אבל מהרגע שהתחלתי לעבוד ולתכנת, נהיה לי קל מאוד ונהניתי מאוד מכל מה שעשיתי, ואפילו ישבתי בלילות לתכנת במקום לישון לפעמים.

הרגשתי לעיתים במהלך הכנת הפרוייקט שלקחתי על עצמי אולי משימה גדולה מדי ולפעמים רציתי להקל על עצמי, אבל בכל מה שיכולתי עשיתי את המיטב והשתדלתי לעשות הכל בצורה שתיראה הכי טוב ותהיה הכי נוחה ויעילה.

בארבעה – חמישה חודשים האחרונים השקעתי המון זמן בהכנת הפרוייקט ואני מרגיש כעת סיפוק רב שהצלחתי להגיע לתוצאה מרשימה בעיני שאני גאה בה ושמח לעמוד מאחוריה, זאת אחרי קשיים ומאמצים רבים שהובילו אותי לסיום.

למרות זאת, אני מתכוון להמשיך ולשפר את הפרוייקט שלי על מנת בסופו של דבר לפנות לחברה Dominion כדי לנסות לעשות איתם שיתוף פעולה או לקבל מהם את האפשרות להעלות את האפליקציה לחנות האפליקציות תמורת כסף, כי לדעתי האפליקציה יכולה להיות שמישה מאוד עבור אוהבי המשחק דומיניון ויש לא מעט כאלו.

# קוד מלא

**Java - Client**

**Activities:**

LoginActivity

*/\*\*  
 \* LoginActivity is the activity of the login which handles  
 \* login to the application.  
 \*/*package com.example.dominion\_game.activities;  
  
import android.app.ProgressDialog;  
import android.content.Context;  
import android.content.Intent;  
import android.content.SharedPreferences;  
import android.os.Bundle;  
import android.util.Log;  
import android.view.View;  
import android.widget.Button;  
import android.widget.CheckBox;  
import android.widget.EditText;  
import android.widget.ImageView;  
import android.widget.TextView;  
import android.widget.Toast;  
  
import androidx.appcompat.app.AppCompatActivity;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.classes.LoginRequests;  
import com.example.dominion\_game.classes.MusicService;  
  
import org.json.JSONException;  
import org.json.JSONObject;  
  
public class LoginActivity extends AppCompatActivity implements View.OnClickListener {  
 EditText etUserName, etPassword;  
 Button btnLogin, btnRegister;  
 CheckBox staySignedIn;  
 SharedPreferences sharedPreferences;  
 ProgressDialog progressDialog;  
 ImageView sound;  
  
 */\*\*  
 \* A function that is called at the start of the activity  
 \* and handles all references.  
 \** ***@param*** *savedInstanceState  
 \*/* @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_login*);  
 sound = findViewById(R.id.*sound*);  
  
 sharedPreferences = getSharedPreferences("account", Context.*MODE\_PRIVATE*);  
 if (sharedPreferences.getString("sound", "").equals("on") || sharedPreferences.getString("sound", "").equals("")) {  
 startService(new Intent(this, MusicService.class)); // plays music  
 if (sharedPreferences.getString("sound", "").equals("")) {  
 SharedPreferences.Editor editor = sharedPreferences.edit();  
 editor.putString("sound", "on");  
 editor.apply();  
 }  
 sound.setImageResource(R.mipmap.*sound\_on*);  
 }  
 else  
 sound.setImageResource(R.mipmap.*sound\_off*);  
  
 if (sharedPreferences.getBoolean("staySignedIn", false) && sharedPreferences.getString("username", null) != null) {  
 Intent intent = new Intent(this, StartActivity.class);  
 startActivityForResult(intent, 1);  
 }  
  
 progressDialog = new ProgressDialog(this);  
  
 etUserName = findViewById(R.id.*username*);  
 etPassword = findViewById(R.id.*password*);  
 etPassword.setText("");  
  
 String username = sharedPreferences.getString("username", null);  
 if (username != null)  
 etUserName.setText(username);  
 else  
 etUserName.setText("");  
  
 staySignedIn = findViewById(R.id.*staySignedIn*);  
  
 btnLogin = findViewById(R.id.*login*);  
 btnRegister = findViewById(R.id.*register*);  
 btnLogin.setClickable(true);  
 btnRegister.setClickable(true);  
  
 btnLogin.setOnClickListener(this);  
 btnRegister.setOnClickListener(this);  
 sound.setOnClickListener(this);  
 }  
  
 */\*\*  
 \* A function which handles back press to do nothing.  
 \*/* @Override  
 public void onBackPressed() {  
  
 }  
  
 */\*\*  
 \* A function that handles all button presses.  
 \** ***@param*** *view A View which is the view that was pressed  
 \*/* @Override  
 public void onClick(View view) {  
 if (view == sound) {  
 if (sharedPreferences.getString("sound", "").equals("on")) {  
 stopService(new Intent(this, MusicService.class)); // stops music  
 SharedPreferences.Editor editor = sharedPreferences.edit();  
 editor.putString("sound", "off");  
 editor.apply();  
 sound.setImageResource(R.mipmap.*sound\_off*);  
 }  
 else if (sharedPreferences.getString("sound", "").equals("off")) {  
 startService(new Intent(this, MusicService.class)); // plays music  
 SharedPreferences.Editor editor = sharedPreferences.edit();  
 editor.putString("sound", "on");  
 editor.apply();  
 sound.setImageResource(R.mipmap.*sound\_on*);  
 }  
 }  
 else if (view == btnLogin) {  
 if (etUserName.getText().length() == 0 || etPassword.getText().length() == 0) {  
 Toast.*makeText*(this, "One or more of the fields are empty", Toast.*LENGTH\_SHORT*).show();  
 return;  
 }  
 progressDialog.setCancelable(false);  
 progressDialog.show();  
 LoginRequests.*login*(this, loginDetailsToJson());  
 }  
 else if (view == btnRegister) {  
 btnRegister.setClickable(false);  
 Intent intent = new Intent(this, RegisterActivity.class);  
 startActivityForResult(intent, 0);  
 }  
 }  
  
 */\*\*  
 \* A function which is called when the user logged out.  
 \** ***@param*** *requestCode  
 \** ***@param*** *resultCode An Integer which is the resultCode of the intent  
 \* - RESULT\_OK or RESULT\_CANCELED  
 \** ***@param*** *intent An Intent to get the extras  
 \*/* @Override  
 protected void onActivityResult(int requestCode, int resultCode, Intent intent) {  
 super.onActivityResult(requestCode, resultCode, intent);  
 if (requestCode == 0 && resultCode == *RESULT\_OK*) {  
 btnRegister.setClickable(true);  
 etUserName.setText(intent.getExtras().getString("username"));  
 etPassword.setText("");  
 }  
 else if (requestCode == 0 && resultCode == *RESULT\_CANCELED*) {  
 btnRegister.setClickable(true);  
 etUserName.setText("");  
 etPassword.setText("");  
 }  
  
 else if (requestCode == 1 && resultCode == *RESULT\_OK*) {  
 btnLogin.setClickable(true);  
 etUserName.setText("");  
 etPassword.setText("");  
 staySignedIn.setChecked(false);  
 }  
 }  
  
 */\*\*  
 \* A function that creates a JSONObject from the data that the user entered.  
 \** ***@return*** *A JSONObject of the login details  
 \*/* public JSONObject loginDetailsToJson() {  
 JSONObject jsonObject = new JSONObject();  
 try {  
 jsonObject.put("username", etUserName.getText().toString());  
 jsonObject.put("password", etPassword.getText().toString());  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
  
 return jsonObject;  
 }  
  
 */\*\*  
 \* A function that is called when the username and the password are correct.  
 \* The function saves them to the shared preferences and starts an intent to StartActivity.  
 \** ***@param*** *username A String with the username of the user  
 \*/* public void loginSuccess(String username) {  
 SharedPreferences.Editor editor = sharedPreferences.edit();  
 editor.putString("username", username);  
 editor.putBoolean("staySignedIn", staySignedIn.isChecked());  
 editor.apply();  
  
 Intent intent = new Intent(this, StartActivity.class);  
 startActivityForResult(intent, 1);  
 }  
  
 */\*\*  
 \* A function that ends the progress bar.  
 \*/* public void endProgressBar() {  
 if (progressDialog != null && progressDialog.isShowing()) {  
 progressDialog.dismiss();  
 }  
 }  
}

RegisterActivity

*/\*\*  
 \* RegisterActivity is the activity of the register which handles  
 \* register to the application.  
 \*/*package com.example.dominion\_game.activities;  
  
import android.app.ProgressDialog;  
import android.content.Context;  
import android.content.Intent;  
import android.content.SharedPreferences;  
import android.os.Bundle;  
import android.view.View;  
import android.widget.Button;  
import android.widget.EditText;  
import android.widget.ImageView;  
import android.widget.Toast;  
  
import androidx.appcompat.app.AppCompatActivity;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.classes.LoginRequests;  
import com.example.dominion\_game.classes.MusicService;  
  
import org.json.JSONException;  
import org.json.JSONObject;  
  
import java.util.regex.Pattern;  
  
public class RegisterActivity extends AppCompatActivity implements View.OnClickListener {  
 EditText etUserName, etEmail, etPassword, etRepeatPassword;  
 Button btnRegister, btnBack;  
 ProgressDialog progressDialog;  
 ImageView sound;  
 SharedPreferences sharedPreferences;  
  
 */\*\*  
 \* A function that is called at the start of the activity  
 \* and handles all references.  
 \** ***@param*** *savedInstanceState  
 \*/* @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_register*);  
 sharedPreferences = getSharedPreferences("account", Context.*MODE\_PRIVATE*);  
 sound = findViewById(R.id.*sound*);  
 if (sharedPreferences.getString("sound", "").equals("on"))  
 sound.setImageResource(R.mipmap.*sound\_on*);  
 else  
 sound.setImageResource(R.mipmap.*sound\_off*);  
 etUserName = findViewById(R.id.*username*);  
 etEmail = findViewById(R.id.*email*);  
 etPassword = findViewById(R.id.*password*);  
 etRepeatPassword = findViewById(R.id.*repeat\_password*);  
  
 progressDialog = new ProgressDialog(this);  
  
 btnRegister = findViewById(R.id.*register*);  
 btnBack = findViewById(R.id.*back*);  
  
 btnRegister.setOnClickListener(this);  
 btnBack.setOnClickListener(this);  
 sound.setOnClickListener(this);  
 }  
  
 */\*\*  
 \* A function which handles back press to return to LoginActivity.  
 \*/* @Override  
 public void onBackPressed() {  
 Intent intent = new Intent();  
 setResult(*RESULT\_CANCELED*, intent);  
 finish();  
 }  
  
 */\*\*  
 \* A function that handles all button presses.  
 \** ***@param*** *view A View which is the view that was pressed  
 \*/* @Override  
 public void onClick(View view) {  
 if (view == sound) {  
 if (sharedPreferences.getString("sound", "").equals("on")) {  
 stopService(new Intent(this, MusicService.class)); // stops music  
 SharedPreferences.Editor editor = sharedPreferences.edit();  
 editor.putString("sound", "off");  
 editor.apply();  
 sound.setImageResource(R.mipmap.*sound\_off*);  
 }  
 else if (sharedPreferences.getString("sound", "").equals("off")) {  
 startService(new Intent(this, MusicService.class)); // plays music  
 SharedPreferences.Editor editor = sharedPreferences.edit();  
 editor.putString("sound", "on");  
 editor.apply();  
 sound.setImageResource(R.mipmap.*sound\_on*);  
 }  
 }  
 else if (view == btnRegister) {  
 // checks if register is valid in some parameters and if valid, register in the server  
 if (etUserName.getText().length() == 0 || etEmail.getText().length() == 0  
 || etPassword.getText().length() == 0 || etRepeatPassword.getText().length() == 0) {  
 Toast.*makeText*(this, "One or more of the fields are empty", Toast.*LENGTH\_SHORT*).show();  
 return;  
 }  
 if (etUserName.getText().length() > 10) {  
 Toast.*makeText*(this, "Username must be maximum 10 characters", Toast.*LENGTH\_SHORT*).show();  
 return;  
 }  
  
 if (!isEmailValid()) {  
 Toast.*makeText*(this, "Email is not valid", Toast.*LENGTH\_SHORT*).show();  
 return;  
 }  
  
 if (!etPassword.getText().toString().equals(etRepeatPassword.getText().toString())) {  
 Toast.*makeText*(this, "Repeated password is different", Toast.*LENGTH\_SHORT*).show();  
 return;  
 }  
 progressDialog.setCancelable(false);  
 progressDialog.show();  
 LoginRequests.*register*(this, registerDetailsToJson());  
 }  
 else if (view == btnBack) {  
 Intent intent = new Intent();  
 setResult(*RESULT\_CANCELED*, intent);  
 finish();  
 }  
 }  
  
 */\*\*  
 \* A function that creates a JSONObject from the data that the user entered.  
 \** ***@return*** *A JSONObject of the register details  
 \*/* public JSONObject registerDetailsToJson() {  
 JSONObject jsonObject = new JSONObject();  
 try {  
 jsonObject.put("username", etUserName.getText().toString());  
 jsonObject.put("email", etEmail.getText().toString());  
 jsonObject.put("password", etPassword.getText().toString());  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
  
 return jsonObject;  
 }  
  
 */\*\*  
 \* A function that checks if the email entered is valid.  
 \** ***@return*** *A Boolean which is true if the email is valid and false if not  
 \*/* public boolean isEmailValid()  
 {  
 String emailRegex = "^[a-zA-Z0-9\_+&\*-]+(?:\\." +  
 "[a-zA-Z0-9\_+&\*-]+)\*@" +  
 "(?:[a-zA-Z0-9-]+\\.)+[a-z" +  
 "A-Z]{2,7}$";  
  
 Pattern pat = Pattern.*compile*(emailRegex);  
 return pat.matcher(etEmail.getText().toString()).matches();  
 }  
  
 */\*\*  
 \* A function that is called when the register happened.  
 \* The function finishes the activity by returning to LoginActivity.  
 \** ***@param*** *username A String with the username of the user  
 \*/* public void registerSuccess(String username) {  
 Intent intent = new Intent();  
 setResult(*RESULT\_OK*, intent);  
 intent.putExtra("username", username);  
 finish();  
 }  
  
 */\*\*  
 \* A function that ends the progress bar.  
 \*/* public void endProgressBar() {  
 if (progressDialog != null && progressDialog.isShowing()) {  
 progressDialog.dismiss();  
 }  
 }  
}

StartActivity

*/\*\*  
 \* StartActivity is the main activity when the user can choose what to do.  
 \*/*package com.example.dominion\_game.activities;  
  
import android.content.Context;  
import android.content.Intent;  
import android.content.SharedPreferences;  
import android.os.Bundle;  
import android.view.View;  
import android.widget.Button;  
import android.widget.ImageView;  
  
import androidx.appcompat.app.AppCompatActivity;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.classes.MusicService;  
  
public class StartActivity extends AppCompatActivity implements View.OnClickListener {  
 Button btnOnlineGame, btnSettings;  
 SharedPreferences sharedPreferences;  
 ImageView sound;  
  
 */\*\*  
 \* A function that is called at the start of the activity  
 \* and handles all references.  
 \** ***@param*** *savedInstanceState  
 \*/* @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_start*);  
  
 sound = findViewById(R.id.*sound*);  
 sharedPreferences = getSharedPreferences("account", Context.*MODE\_PRIVATE*);  
 if (sharedPreferences.getString("sound", "").equals("on"))  
 sound.setImageResource(R.mipmap.*sound\_on*);  
 else  
 sound.setImageResource(R.mipmap.*sound\_off*);  
  
 btnOnlineGame = findViewById(R.id.*btnOnlineGame*);  
 btnSettings = findViewById(R.id.*btnSettings*);  
 btnOnlineGame.setOnClickListener(this);  
 btnSettings.setOnClickListener(this);  
 sound.setOnClickListener(this);  
 }  
  
 */\*\*  
 \* A function which handles back press to do nothing.  
 \*/* @Override  
 public void onBackPressed() {  
  
 }  
  
 */\*\*  
 \* A function that handles all button presses.  
 \** ***@param*** *view A View which is the view that was pressed  
 \*/* @Override  
 public void onClick(View view) {  
 if (view == sound) {  
 if (sharedPreferences.getString("sound", "").equals("on")) {  
 stopService(new Intent(this, MusicService.class)); // stops music  
 SharedPreferences.Editor editor = sharedPreferences.edit();  
 editor.putString("sound", "off");  
 editor.apply();  
 sound.setImageResource(R.mipmap.*sound\_off*);  
 }  
 else if (sharedPreferences.getString("sound", "").equals("off")) {  
 startService(new Intent(this, MusicService.class)); // plays music  
 SharedPreferences.Editor editor = sharedPreferences.edit();  
 editor.putString("sound", "on");  
 editor.apply();  
 sound.setImageResource(R.mipmap.*sound\_on*);  
 }  
 }  
 else if (view == btnOnlineGame) {  
 Intent intent = new Intent(this, OnlineGameActivity.class);  
 startActivity(intent);  
 }  
 else if (view == btnSettings) {  
 // logs out and delete the data in the sharedPreferences  
 SharedPreferences.Editor editor = sharedPreferences.edit();  
 editor.remove("username");  
 editor.remove("staySignedIn");  
 editor.apply();  
  
 Intent intent = new Intent();  
 setResult(*RESULT\_OK*, intent);  
 finish();  
 }  
 }  
}

OnlineGameActivity

*/\*\*  
 \* OnlineGameActivity is the activity when the user can choose  
 \* between creating a table and joining a table.  
 \*/*package com.example.dominion\_game.activities;  
  
import androidx.appcompat.app.AppCompatActivity;  
  
import android.content.Context;  
import android.content.Intent;  
import android.content.SharedPreferences;  
import android.os.Bundle;  
import android.view.View;  
import android.widget.Button;  
import android.widget.ImageView;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.classes.GameManagerBeforeStart;  
import com.example.dominion\_game.classes.MusicService;  
  
public class OnlineGameActivity extends AppCompatActivity implements View.OnClickListener {  
 Button btnTables, btnCreateTable;  
 SharedPreferences sharedPreferences;  
 ImageView sound;  
  
 */\*\*  
 \* A function that is called at the start of the activity  
 \* and handles all references.  
 \** ***@param*** *savedInstanceState  
 \*/* @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_online\_game*);  
 sound = findViewById(R.id.*sound*);  
 sharedPreferences = getSharedPreferences("account", Context.*MODE\_PRIVATE*);  
 if (sharedPreferences.getString("sound", "").equals("on"))  
 sound.setImageResource(R.mipmap.*sound\_on*);  
 else  
 sound.setImageResource(R.mipmap.*sound\_off*);  
 btnTables = findViewById(R.id.*btnTables*);  
 btnCreateTable = findViewById(R.id.*btnCreateTable*);  
 btnTables.setOnClickListener(this);  
 btnCreateTable.setOnClickListener(this);  
 btnTables.setClickable(true);  
 btnCreateTable.setClickable(true);  
 sound.setOnClickListener(this);  
 }  
  
 */\*\*  
 \* A function which handles back press to return to StartActivity.  
 \*/* @Override  
 public void onBackPressed() {  
 finish();  
 }  
  
 */\*\*  
 \* A function that handles all button presses.  
 \** ***@param*** *view A View which is the view that was pressed  
 \*/* @Override  
 public void onClick(View view) {  
 if (view == sound) {  
 if (sharedPreferences.getString("sound", "").equals("on")) {  
 stopService(new Intent(this, MusicService.class)); // stops music  
 SharedPreferences.Editor editor = sharedPreferences.edit();  
 editor.putString("sound", "off");  
 editor.apply();  
 sound.setImageResource(R.mipmap.*sound\_off*);  
 }  
 else if (sharedPreferences.getString("sound", "").equals("off")) {  
 startService(new Intent(this, MusicService.class)); // plays music  
 SharedPreferences.Editor editor = sharedPreferences.edit();  
 editor.putString("sound", "on");  
 editor.apply();  
 sound.setImageResource(R.mipmap.*sound\_on*);  
 }  
 }  
 else if (btnTables == view)  
 {  
 btnTables.setClickable(false);  
 Intent intent = new Intent(this, OnlineTablesActivity.class);  
 startActivity(intent);  
 btnTables.setClickable(true);  
 }  
 else if (btnCreateTable == view)  
 {  
 btnCreateTable.setClickable(false);  
 Intent intent = new Intent(this, PrepareGameActivity.class);  
 intent.putExtra("gameManagerBeforeStart", new GameManagerBeforeStart(sharedPreferences.getString("username", "")));  
 this.startActivityForResult(intent, 0);  
 }  
 }  
  
 */\*\*  
 \* A function which is called when the user returned from  
 \* OnlineTablesActivity or from PrepareGameActivity.  
 \** ***@param*** *requestCode  
 \** ***@param*** *resultCode An Integer which is the resultCode of the intent  
 \* - RESULT\_OK or RESULT\_CANCELED  
 \** ***@param*** *intent An Intent to get the extras  
 \*/* @Override  
 protected void onActivityResult(int requestCode, int resultCode, Intent intent) {  
 super.onActivityResult(requestCode, resultCode, intent);  
 if (requestCode == 0 && resultCode == *RESULT\_OK*)  
 btnCreateTable.setClickable(true);  
 }  
}

OnlineTablesActivity

*/\*\*  
 \* OnlineTablesActivity is the activity when the user  
 \* see a list view of all online tables that he can join and play.  
 \*/*package com.example.dominion\_game.activities;  
  
import androidx.appcompat.app.AppCompatActivity;  
  
import android.app.ProgressDialog;  
import android.content.Context;  
import android.content.Intent;  
import android.content.SharedPreferences;  
import android.os.Bundle;  
import android.view.View;  
import android.widget.Button;  
import android.widget.ImageView;  
import android.widget.ListView;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.classes.GameRequests;  
import com.example.dominion\_game.classes.MusicService;  
import com.example.dominion\_game.classes.Table;  
  
import org.json.JSONArray;  
import org.json.JSONException;  
import org.json.JSONObject;  
  
import java.util.ArrayList;  
  
public class OnlineTablesActivity extends AppCompatActivity implements View.OnClickListener {  
 Button btnBack;  
 ListView lv;  
 ArrayList<Table> tableList;  
 TablesAdapter tablesAdapter;  
 ProgressDialog progressDialog;  
 ImageView background;  
 boolean isFinished;  
 ImageView sound;  
 SharedPreferences sharedPreferences;  
  
 */\*\*  
 \* A function that is called at the start of the activity  
 \* and handles all references.  
 \** ***@param*** *savedInstanceState  
 \*/* @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_online\_tables*);  
 sharedPreferences = getSharedPreferences("account", Context.*MODE\_PRIVATE*);  
 sound = findViewById(R.id.*sound*);  
 if (sharedPreferences.getString("sound", "").equals("on"))  
 sound.setImageResource(R.mipmap.*sound\_on*);  
 else  
 sound.setImageResource(R.mipmap.*sound\_off*);  
 sound.setOnClickListener(this);  
 btnBack = findViewById(R.id.*btnBack*);  
 btnBack.setOnClickListener(this);  
 btnBack.setClickable(true);  
  
 background = findViewById(R.id.*background*);  
 tableList = new ArrayList<>();  
 tableList.add(new Table("Tables", "")); // this item is the title  
 background.setVisibility(View.*VISIBLE*);  
 progressDialog = new ProgressDialog(this);  
 progressDialog.setMessage("Loading Tables");  
 progressDialog.setCancelable(false);  
 progressDialog.show();  
 lv = findViewById(R.id.*lvTables*);  
 isFinished = false;  
 uploadTables();  
 GameRequests.*getTables*(this);  
 endProgressBar();  
  
 }  
  
 */\*\*  
 \* A function which handles back press to set isFinished to true  
 \* and return to OnlineGameActivity.  
 \*/* @Override  
 public void onBackPressed() {  
 isFinished = true;  
 btnBack.setClickable(false);  
 }  
  
 */\*\*  
 \* A function that handles all button presses.  
 \** ***@param*** *view A View which is the view that was pressed  
 \*/* @Override  
 public void onClick(View view) {  
 if (view == sound) {  
 if (sharedPreferences.getString("sound", "").equals("on")) {  
 stopService(new Intent(this, MusicService.class));  
 SharedPreferences.Editor editor = sharedPreferences.edit();  
 editor.putString("sound", "off");  
 editor.apply();  
 sound.setImageResource(R.mipmap.*sound\_off*);  
 }  
 else if (sharedPreferences.getString("sound", "").equals("off")) {  
 startService(new Intent(this, MusicService.class));  
 SharedPreferences.Editor editor = sharedPreferences.edit();  
 editor.putString("sound", "on");  
 editor.apply();  
 sound.setImageResource(R.mipmap.*sound\_on*);  
 }  
 }  
 else if (btnBack == view) {  
 isFinished = true;  
 btnBack.setClickable(false);  
 }  
 }  
  
 */\*\*  
 \* A function that finishes the intent and returns to OnlineGameActivity  
 \*/* public void prepareToLeave() {  
 finish();  
 }  
  
 public boolean isFinished() {  
 return isFinished;  
 }  
  
 public void setFinished(boolean finished) {  
 isFinished = finished;  
 }  
  
 */\*\*  
 \* A function that creates the tables adapter and sets this as the ListView adapter  
 \*/* public void uploadTables() {  
 tablesAdapter = new TablesAdapter(this,0,0, tableList, this);  
 lv.setAdapter(tablesAdapter);  
 }  
  
 */\*\*  
 \* A function that updates the tables on real time.  
 \** ***@param*** *response A JSONObject that is given from the server and keeps all tables online  
 \*/* public void updateTables(JSONObject response) {  
 while (tableList.size() > 1)  
 tableList.remove(1);  
 JSONArray keys = response.names();  
 if (keys != null)  
 for (int i = 0; i < keys.length(); i++) {  
 try {  
 String creator = response.getJSONObject(keys.getString(i)).getString("idP1");  
 tableList.add(new Table(creator, keys.getString(i)));  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
 }  
 tablesAdapter.notifyDataSetChanged();  
 }  
  
 */\*\*  
 \* A function that ends the progress bar.  
 \*/* public void endProgressBar() {  
 if (progressDialog != null && progressDialog.isShowing()) {  
 progressDialog.dismiss();  
 background.setVisibility(View.*INVISIBLE*);  
 }  
 }  
}

PrepareGameActivity

*/\*\*  
 \* PrepareGameActivity is the activity before starting the game which handles  
 \* the players connected to the game. The creator of the game have also options  
 \* for the game (for example, if the game is rated or not), and it is also the  
 \* activity of the end of the game when the winner of the game is shown.  
 \*/*package com.example.dominion\_game.activities;  
  
import androidx.appcompat.app.AppCompatActivity;  
  
import android.app.ProgressDialog;  
import android.content.Context;  
import android.content.Intent;  
import android.content.SharedPreferences;  
import android.os.Bundle;  
import android.view.View;  
import android.widget.Button;  
import android.widget.CompoundButton;  
import android.widget.ImageView;  
import android.widget.ProgressBar;  
import android.widget.Switch;  
import android.widget.TextView;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.classes.GameManagerBeforeStart;  
import com.example.dominion\_game.classes.GameRequests;  
import com.example.dominion\_game.classes.Help;  
import com.example.dominion\_game.classes.MusicService;  
  
public class PrepareGameActivity extends AppCompatActivity implements View.OnClickListener, CompoundButton.OnCheckedChangeListener {  
 GameManagerBeforeStart gameManagerBeforeStart;  
 Button btnReady, btnLeaveTable;  
 TextView tvP1, tvP2;  
 TextView tvIsRated;  
 TextView tvWinner, tvP1VP, tvP2VP;  
 Switch sw;  
 ImageView background;  
 ProgressDialog progressDialog;  
 ImageView sound;  
 SharedPreferences sharedPreferences;  
  
 */\*\*  
 \* A function that is called at the start of the activity  
 \* and handles all references.  
 \** ***@param*** *savedInstanceState  
 \*/* @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_prepare\_game*);  
 sound = findViewById(R.id.*sound*);  
 sharedPreferences = getSharedPreferences("account", Context.*MODE\_PRIVATE*);  
 if (sharedPreferences.getString("sound", "").equals("on"))  
 sound.setImageResource(R.mipmap.*sound\_on*);  
 else  
 sound.setImageResource(R.mipmap.*sound\_off*);  
 tvIsRated = findViewById(R.id.*tvIsRated*);  
 sw = findViewById(R.id.*sw*);  
 btnReady = findViewById(R.id.*btnReady*);  
 btnReady.setVisibility(View.*VISIBLE*);  
 btnLeaveTable = findViewById(R.id.*btnLeaveTable*);  
 tvP1 = findViewById(R.id.*tvP1*);  
 tvP2 = findViewById(R.id.*tvP2*);  
 tvP1.setTextColor(getResources().getColor(R.color.*red*));  
 tvP2.setTextColor(getResources().getColor(R.color.*red*));  
  
 tvWinner = findViewById(R.id.*tvWinner*);  
 tvP1VP = findViewById(R.id.*tvP1VP*);  
 tvP2VP = findViewById(R.id.*tvP2VP*);  
  
 background = findViewById(R.id.*background*);  
 Intent intent = getIntent();  
 gameManagerBeforeStart = (GameManagerBeforeStart)intent.getExtras().getSerializable("gameManagerBeforeStart");  
 tvP1.setText(String.*valueOf*(gameManagerBeforeStart.getIdP1()));  
 tvP2.setText(String.*valueOf*(gameManagerBeforeStart.getIdP2()));  
 tvWinner.setVisibility(View.*INVISIBLE*);  
 tvP1VP.setVisibility(View.*INVISIBLE*);  
 tvP2VP.setVisibility(View.*INVISIBLE*);  
 if (gameManagerBeforeStart.isCreator()) {  
 tvIsRated.setVisibility(View.*VISIBLE*);  
 sw.setVisibility(View.*VISIBLE*);  
 sw.setOnCheckedChangeListener(this);  
 background.setVisibility(View.*VISIBLE*);  
 progressDialog = new ProgressDialog(this);  
 progressDialog.setMessage("Creating Table");  
 progressDialog.setCancelable(false);  
 progressDialog.show();  
 GameRequests.*creator\_start*(this);  
 }  
 else {  
 tvIsRated.setVisibility(View.*GONE*);  
 sw.setVisibility(View.*GONE*);  
 background.setVisibility(View.*VISIBLE*);  
 progressDialog = new ProgressDialog(this);  
 progressDialog.setMessage("Joining Table");  
 progressDialog.setCancelable(false);  
 progressDialog.show();  
 GameRequests.*non\_creator\_start*(this);  
 }  
  
 btnReady.setOnClickListener(this);  
 btnLeaveTable.setOnClickListener(this);  
 btnLeaveTable.setClickable(true);  
 sound.setOnClickListener(this);  
 }  
  
 */\*\*  
 \* A function which handles back press to leave the table.  
 \*/* @Override  
 public void onBackPressed() {  
 btnLeaveTable.setClickable(false);  
 prepareToLeave();  
 }  
  
 */\*\*  
 \* A function that handles all button presses.  
 \** ***@param*** *view A View which is the view that was pressed  
 \*/* @Override  
 public void onClick(View view) {  
 if (view == sound) {  
 if (sharedPreferences.getString("sound", "").equals("on")) {  
 stopService(new Intent(this, MusicService.class)); // stops music  
 SharedPreferences.Editor editor = sharedPreferences.edit();  
 editor.putString("sound", "off");  
 editor.apply();  
 sound.setImageResource(R.mipmap.*sound\_off*);  
 }  
 else if (sharedPreferences.getString("sound", "").equals("off")) {  
 startService(new Intent(this, MusicService.class)); // plays music  
 SharedPreferences.Editor editor = sharedPreferences.edit();  
 editor.putString("sound", "on");  
 editor.apply();  
 sound.setImageResource(R.mipmap.*sound\_on*);  
 }  
 }  
 else if (btnReady == view)  
 {  
 boolean setTo = true;  
 if (btnReady.getText().equals("Ready"))  
 btnReady.setText("Not Ready");  
 else if (btnReady.getText().equals("Not Ready")) {  
 setTo = false;  
 btnReady.setText("Ready");  
 }  
 if (this.gameManagerBeforeStart.isCreator())  
 this.gameManagerBeforeStart.setReady1(setTo);  
 else  
 this.gameManagerBeforeStart.setReady2(setTo);  
 updateUI(true);  
 GameRequests.*updateReady*(this);  
 }  
 else if (btnLeaveTable == view) {  
 btnLeaveTable.setClickable(false);  
 prepareToLeave();  
 }  
 }  
  
 */\*\*  
 \* A function that updates on screen the data from gameManagerBeforeStart.  
 \** ***@param*** *onlyMe A Boolean which is true if the function should update if the other player  
 \* is ready or only for the player  
 \*/* public void updateUI(boolean onlyMe) {  
 tvP1.setText(String.*valueOf*(gameManagerBeforeStart.getIdP1()));  
 tvP2.setText(String.valueOf(gameManagerBeforeStart.getIdP2()));  
 if (this.gameManagerBeforeStart.isCreator() || !onlyMe) {  
 if (this.gameManagerBeforeStart.isReady1())  
 this.tvP1.setTextColor(getResources().getColor(R.color.green));  
 else  
 this.tvP1.setTextColor(getResources().getColor(R.color.red));  
 }  
  
 if (!this.gameManagerBeforeStart.isCreator() || !onlyMe) {  
 if (this.gameManagerBeforeStart.isReady2())  
 this.tvP2.setTextColor(getResources().getColor(R.color.green));  
 else  
 this.tvP2.setTextColor(getResources().getColor(R.color.red));  
 }  
  
 if (this.gameManagerBeforeStart.isReady1() && this.gameManagerBeforeStart.isReady2())  
 btnReady.setVisibility(View.INVISIBLE);  
 else  
 btnReady.setVisibility(View.VISIBLE);  
 }  
  
 public GameManagerBeforeStart getGameManagerBeforeStart() {  
 return this.gameManagerBeforeStart;  
 }  
  
 */\*\*  
 \* A function that finishes the intent and returns to OnlineGameActivity  
 \*/* public void leaveTable() {  
 Intent intent = new Intent();  
 setResult(RESULT\_OK, intent);  
 finish();  
 }  
  
 */\*\*  
 \* A function that deletes the table if the player is the creator  
 \* and delete the player from the table if the player is not the creator.  
 \*/* public void prepareToLeave() {  
 if(gameManagerBeforeStart.isCreator()) {  
 progressDialog = new ProgressDialog(this);  
 progressDialog.setMessage("Deleting Table");  
 progressDialog.setCancelable(false);  
 progressDialog.show();  
 GameRequests.delete\_game\_manager\_before\_start(this);  
 }  
 else {  
 progressDialog = new ProgressDialog(this);  
 progressDialog.setMessage("Exit Table");  
 progressDialog.setCancelable(false);  
 progressDialog.show();  
 GameRequests.deleteP2(this);  
 }  
 }  
  
 */\*\*  
 \* A function that starts the game by starting GameActivity.  
 \*/* public void startGame() {  
 Intent intent = new Intent(this, GameActivity.class);  
 intent.putExtra("gameManagerBeforeStart", gameManagerBeforeStart);  
 startActivityForResult(intent, 0);  
 }  
  
 */\*\*  
 \* A function which is called when the game is ended and updates the result of the game.  
 \** ***@param*** *requestCode  
 \** ***@param*** *resultCode An Integer which is the resultCode of the intent  
 \* - RESULT\_OK or RESULT\_CANCELED  
 \** ***@param*** *intent An Intent to get the extras  
 \*/* @Override  
 protected void onActivityResult(int requestCode, int resultCode, Intent intent) {  
 super.onActivityResult(requestCode, resultCode, intent);  
 if(requestCode == 0 && resultCode == RESULT\_OK) {  
 btnLeaveTable.setClickable(true);  
 btnReady.setVisibility(View.VISIBLE);  
 tvWinner.setVisibility(View.VISIBLE);  
  
 btnReady.setText("Ready");  
 if (intent.getExtras().getString("result").equals("resign")) {  
 tvP1VP.setVisibility(View.INVISIBLE);  
 tvP2VP.setVisibility(View.INVISIBLE);  
 tvWinner.setText(intent.getExtras().getString("player").concat(" resigned"));  
 }  
 else if (intent.getExtras().getString("result").equals("win") || intent.getExtras().getString("result").equals("draw")) {  
 tvP1VP.setVisibility(View.VISIBLE);  
 tvP2VP.setVisibility(View.VISIBLE);  
 tvP1VP.setText(gameManagerBeforeStart.getIdP1() + ": " + intent.getExtras().getInt("P1VP"));  
 tvP2VP.setText(gameManagerBeforeStart.getIdP2() + ": " + intent.getExtras().getInt("P2VP"));  
 if (intent.getExtras().getString("result").equals("win"))  
 tvWinner.setText("The winner is ".concat(intent.getExtras().getString("player")));  
 else if (intent.getExtras().getString("result").equals("draw"))  
 tvWinner.setText("Draw");  
 }  
  
 gameManagerBeforeStart.restartReady();  
 updateUI(false);  
 GameRequests.waitForStartGame(gameManagerBeforeStart.isCreator(), this, 0);  
 }  
 }  
  
 */\*\*  
 \* A function that is called when the switch has changed  
 \* and updates it on gameManagerBeforeStart.  
 \** ***@param*** *compoundButton  
 \** ***@param*** *isRated A Boolean which is true if the state is checked and false if not  
 \*/* @Override  
 public void onCheckedChanged(CompoundButton compoundButton, boolean isRated) {  
 gameManagerBeforeStart.setRated(isRated);  
 }  
  
 */\*\*  
 \* A function that ends the progress bar.  
 \*/* public void endProgressBar() {  
 if (progressDialog != null && progressDialog.isShowing()) {  
 progressDialog.dismiss();  
 background.setVisibility(View.INVISIBLE);  
 }  
 }  
}

GameActivity

*/\*\*  
 \* GameActivity is the activity of the game which handles  
 \* all game plays and clicks until the end.  
 \*/*package com.example.dominion\_game.activities;  
  
import android.Manifest;  
import android.app.AlertDialog;  
import android.app.Dialog;  
import android.app.ProgressDialog;  
import android.content.DialogInterface;  
import android.content.Intent;  
import android.content.IntentFilter;  
import android.content.pm.PackageManager;  
import android.graphics.drawable.ColorDrawable;  
import android.os.Bundle;  
import android.view.View;  
import android.view.Window;  
import android.widget.AbsListView;  
import android.widget.Button;  
import android.widget.ImageView;  
import android.widget.ListView;  
import android.widget.RelativeLayout;  
import android.widget.TextView;  
  
import androidx.appcompat.app.AppCompatActivity;  
import androidx.constraintlayout.widget.ConstraintLayout;  
import androidx.constraintlayout.widget.ConstraintSet;  
import androidx.core.app.ActivityCompat;  
import androidx.core.content.ContextCompat;  
import androidx.recyclerview.widget.GridLayoutManager;  
import androidx.recyclerview.widget.LinearLayoutManager;  
import androidx.recyclerview.widget.RecyclerView;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.classes.\*;  
  
import java.util.ArrayList;  
import java.util.HashMap;  
  
public class GameActivity extends AppCompatActivity implements View.OnClickListener, View.OnLongClickListener {  
 GameManager game;  
 Button btnStart, btnEnd, btnAutoplay;  
 ArrayList<Button> btnActions;  
 TextView tvTurn;  
 ConstraintLayout clTurn;  
 HashMap<String, View> victoryCards;  
 HashMap<String, View> treasureCards;  
 HashMap<String, View> actionCards;  
 HashMap<String, View> actionBigCards;  
 Dialog cardDialog; // The dialog which is shown in long click to show the card bigger  
 View myDeck, enemyHand, enemyDeck;  
 ImageView ivEnemyDiscard, ivMyDiscard;  
 TextView tvMyVP, tvEnemyVP, tvMyName, tvEnemyName;  
  
 ListView lvLog;  
 LogAdapter logAdapter;  
 androidx.recyclerview.widget.RecyclerView rvTrash;  
 TrashAdapter trashAdapter;  
 TextView tvInfoTitle;  
  
 androidx.recyclerview.widget.RecyclerView rvHand;  
 HandAdapter handAdapter;  
  
 androidx.recyclerview.widget.RecyclerView rvActionCardsPlaying;  
 ActionCardPlayingAdapter actionCardsPlayingAdapter;  
 ConstraintLayout clActionCardsPlaying;  
 ImageView ivArrowActionCardsPlaying;  
 String positionOfRVActionCardsPlaying;  
  
 ImageView background;  
 ProgressDialog progressDialog;  
  
 Button btnKingdomOrPlayArea, btnTrashOrLog, btnResign;  
 ConstraintLayout clPlayArea, clKingdom;  
 RelativeLayout rlButtonsInPlay;  
  
 PhoneCallReceiver phoneCallReceiver; // broadcast receiver  
  
 */\*\*  
 \* A function that is called at the start of the activity  
 \* and handles all references and creates the gameManager.  
 \** ***@param*** *savedInstanceState  
 \*/* @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_game\_designed*);  
 phoneCallReceiver = new PhoneCallReceiver();  
 this.checkAndRequestPermissions();  
  
 btnStart = findViewById(R.id.*btnStart*);  
 btnEnd = findViewById(R.id.*btnEnd*);  
 btnAutoplay = findViewById(R.id.*btnAutoplay*);  
  
 btnActions = new ArrayList<>();  
 for(int i = 0; i < 3; i++) {  
 int resID = getResources().getIdentifier(("btnAction" + (i+1)), "id", getPackageName());  
 btnActions.add((Button)findViewById(resID));  
 btnActions.get(i).setVisibility(View.*GONE*);  
 }  
  
 tvTurn = findViewById(R.id.*tvTurn*);  
 clTurn = findViewById(R.id.*clTurn*);  
  
 victoryCards = new HashMap<>();  
 treasureCards = new HashMap<>();  
 actionCards = new HashMap<>();  
 actionBigCards = new HashMap<>();  
  
 // Creates the card dialog to be cancelable and with no title  
 cardDialog = new Dialog(this);  
 cardDialog.requestWindowFeature(Window.*FEATURE\_NO\_TITLE*);  
 cardDialog.setContentView(R.layout.*card\_dialog*);  
 cardDialog.getWindow().setBackgroundDrawable(new ColorDrawable(android.graphics.Color.*TRANSPARENT*));  
 cardDialog.getWindow().setLayout((int)(getResources().getDisplayMetrics().heightPixels\*0.6), (int)(getResources().getDisplayMetrics().heightPixels\*0.9));  
 cardDialog.setCancelable(true);  
  
 ivMyDiscard = findViewById(R.id.*ivMyDiscard*);  
 myDeck = findViewById(R.id.*myDeck*);  
 enemyHand = findViewById(R.id.*enemyHand*);  
 enemyDeck = findViewById(R.id.*enemyDeck*);  
 ivEnemyDiscard = findViewById(R.id.*ivEnemyDiscard*);  
 tvMyName = findViewById(R.id.*tvMyName*);  
 tvMyVP = findViewById(R.id.*tvMyVP*);  
 tvEnemyName = findViewById(R.id.*tvEnemyName*);  
 tvEnemyVP = findViewById(R.id.*tvEnemyVP*);  
  
 enemyHand.setVisibility(View.*VISIBLE*);  
 tvMyName.setVisibility(View.*VISIBLE*);  
 tvEnemyName.setVisibility(View.*VISIBLE*);  
 myDeck.setVisibility(View.*INVISIBLE*);  
 enemyDeck.setVisibility(View.*INVISIBLE*);  
 ivEnemyDiscard.setVisibility(View.*INVISIBLE*);  
 ivMyDiscard.setVisibility(View.*INVISIBLE*);  
 tvMyVP.setVisibility(View.*INVISIBLE*);  
 tvEnemyVP.setVisibility(View.*INVISIBLE*);  
 tvTurn.setVisibility(View.*INVISIBLE*);  
 clTurn.setVisibility(View.*INVISIBLE*);  
  
 lvLog = findViewById(R.id.*lvLog*);  
 rvTrash = findViewById(R.id.*rvTrash*);  
 rvHand = findViewById(R.id.*rvHand*);  
 rvActionCardsPlaying = findViewById(R.id.*rvActionCardsPlaying*);  
 clActionCardsPlaying = findViewById(R.id.*clActionCardsPlaying*);  
 clActionCardsPlaying.setVisibility(View.*INVISIBLE*);  
 ivArrowActionCardsPlaying = findViewById(R.id.*ivArrowActionCardsPlaying*);  
 this.positionOfRVActionCardsPlaying = "up";  
 ivArrowActionCardsPlaying.setOnClickListener(this);  
 tvInfoTitle = findViewById(R.id.*tvInfoTitle*);  
  
 // Gets the GameManagerBeforeStart from the PrepareGameActivity and creates the gameManager  
 Intent intent = getIntent();  
 GameManagerBeforeStart gameManagerBeforeStart = (GameManagerBeforeStart)intent.getExtras().getSerializable("gameManagerBeforeStart");  
 game = new GameManager(gameManagerBeforeStart, this);  
  
 btnKingdomOrPlayArea = findViewById(R.id.*btnKingdomOrPlayArea*);  
 btnTrashOrLog = findViewById(R.id.*btnTrashOrLog*);  
 btnResign = findViewById(R.id.*btnResign*);  
  
 clPlayArea = findViewById(R.id.*clPlayArea*);  
 clKingdom = findViewById(R.id.*clKingdom*);  
 clPlayArea.setVisibility(View.*VISIBLE*);  
 clKingdom.setVisibility(View.*GONE*);  
 btnKingdomOrPlayArea.setText("Kingdom");  
 btnTrashOrLog.setText("Trash");  
 rlButtonsInPlay = findViewById(R.id.*rlButtonsInPlay*);  
  
 background = findViewById(R.id.*background*);  
 background.setVisibility(View.*VISIBLE*);  
 // Shows a progress dialog for start game until starting to get or upload the game data  
 progressDialog = new ProgressDialog(this);  
 progressDialog.setMessage("Starting Game");  
 progressDialog.setCancelable(false);  
 progressDialog.show();  
 game.startUploadAndGet();  
 }  
  
 */\*\*  
 \* A function that checks the permissions: READ\_PHONE\_STATE, SEND\_SMS, READ\_CALL\_LOG  
 \* for the broadcast phoneCallReceiver and asks the permissions that haven't been  
 \* granted already.  
 \* If all permissions are granted already it registers phoneCallReceiver.  
 \*/* public void checkAndRequestPermissions() {  
 if (ContextCompat.*checkSelfPermission*(this, Manifest.permission.*READ\_PHONE\_STATE*)  
 == PackageManager.*PERMISSION\_GRANTED* &&  
 ContextCompat.*checkSelfPermission*(this, Manifest.permission.*SEND\_SMS*)  
 == PackageManager.*PERMISSION\_GRANTED* &&  
 ContextCompat.*checkSelfPermission*(this, Manifest.permission.*READ\_CALL\_LOG*)  
 == PackageManager.*PERMISSION\_GRANTED*) {  
 IntentFilter filter = new IntentFilter();  
 filter.addAction("android.intent.action.PHONE\_STATE");  
 registerReceiver(phoneCallReceiver, filter);  
 return;  
 }  
  
 if (ContextCompat.*checkSelfPermission*(this, Manifest.permission.*READ\_PHONE\_STATE*)  
 != PackageManager.*PERMISSION\_GRANTED*) {  
 // asks the permission  
 ActivityCompat.*requestPermissions*(this, new String[]{Manifest.permission.*READ\_PHONE\_STATE*}, 1);  
 }  
  
 if (ContextCompat.*checkSelfPermission*(this, Manifest.permission.*SEND\_SMS*)  
 != PackageManager.*PERMISSION\_GRANTED*) {  
 // asks the permission  
 ActivityCompat.*requestPermissions*(this, new String[]{Manifest.permission.*SEND\_SMS*}, 1);  
 }  
 if (ContextCompat.*checkSelfPermission*(this, Manifest.permission.*READ\_CALL\_LOG*)  
 != PackageManager.*PERMISSION\_GRANTED*) {  
 // asks the permission  
 ActivityCompat.*requestPermissions*(this, new String[]{Manifest.permission.*READ\_CALL\_LOG*}, 1);  
 }  
 }  
  
 */\*\*  
 \* A function that is called when all permissions were denied or granted.  
 \** ***@param*** *requestCode An Integer with the request code passed in requestPermissions  
 \** ***@param*** *permissions A String array with the requested permissions  
 \** ***@param*** *grantResults An Integer Array with the grant results for the corresponding  
 \* permissions which is either PERMISSION\_GRANTED or PERMISSION\_DENIED  
 \*/* @Override  
 public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults) {  
 for (int i = 0; i < permissions.length; i++) {  
 if (grantResults[i] != PackageManager.*PERMISSION\_GRANTED*)  
 return;  
 }  
 // all permissions were granted  
 IntentFilter filter = new IntentFilter();  
 filter.addAction("android.intent.action.PHONE\_STATE");  
 registerReceiver(phoneCallReceiver, filter);  
 }  
  
 */\*\*  
 \* A function that is called when this intent is destroyed and unregisters phoneCallReceiver.  
 \*/* @Override  
 protected void onDestroy() {  
 super.onDestroy();  
 unregisterReceiver(phoneCallReceiver);  
 }  
  
 */\*\*  
 \* A function that handles back press to show a dialog.  
 \*/* @Override  
 public void onBackPressed() {  
 // Shows an alert dialog to ask if he is he wants sure to resign  
 AlertDialog.Builder builder = new AlertDialog.Builder(this);  
 builder.setMessage("Do you really want to resign?");  
 builder.setCancelable(false);  
 builder.setPositiveButton("Yes", new DialogInterface.OnClickListener() {  
 */\*\*  
 \* Handles yes answer and ends the dialog and the game.  
 \** ***@param*** *dialogInterface  
 \** ***@param*** *i  
 \*/* @Override  
 public void onClick(DialogInterface dialogInterface, int i) {  
 GameActivity.this.endGame();  
 game.setResigned(true);  
 dialogInterface.dismiss();  
 }  
 });  
 builder.setNegativeButton("No", new DialogInterface.OnClickListener() {  
 */\*\*  
 \* Handles no answer and ends the dialog.  
 \** ***@param*** *dialogInterface  
 \** ***@param*** *i  
 \*/* @Override  
 public void onClick(DialogInterface dialogInterface, int i) {  
 dialogInterface.dismiss();  
 }  
 });  
 AlertDialog dialog = builder.create();  
 dialog.show();  
 }  
  
 */\*\*  
 \* A function that handles the long click of a card.  
 \* The function is called when a card when any card with a card image is pressed long.  
 \* The function shows a dialog of the card pressed in big.  
 \** ***@param*** *v A View which is the view that was pressed long  
 \** ***@return*** *A Boolean which is true always  
 \*/* @Override  
 public boolean onLongClick(View v) {  
 ImageView resourceIv;  
 // v is an imageView only when ivMyDiscard or ivEnemyDiscard is pressed  
 if (v instanceof ImageView)  
 resourceIv = (ImageView) v;  
 else  
 resourceIv = v.findViewById(R.id.*ivAction*);  
 ImageView iv = cardDialog.findViewById(R.id.*ivAction*);  
 if (resourceIv.getContentDescription() != null) {  
 Card card = Help.*nameToCard*(resourceIv.getContentDescription().toString());  
 if (card != null) {  
 iv.setImageResource(card.getImageSource());  
 cardDialog.show();  
 }  
 }  
 return true;  
 }  
  
 */\*\*  
 \* A function that is called when a card from hand is pressed and can be played.  
 \* The function calls a function useCard from gameManager.  
 \** ***@param*** *cardName A String which is the name of the card which should be used  
 \*/* public void useCard(String cardName) {  
 if (game.getTimes().peek() != 1) // when the card should be played more than once  
 game.useCard(cardName, game.getTimes().peek(), false, false);  
 else  
 game.useCard(cardName, 1, false, false);  
 }  
  
 */\*\*  
 \* A function that is called when a card from board is pressed and can be bought.  
 \* The function calls a function buyCard from gameManager and automatically changes  
 \* the turn if the player doesn't have more buys in turn.  
 \** ***@param*** *cardName A string which is the name of the card which the player wants to buy  
 \*/* public void buyCard(String cardName) {  
 game.buyCard(cardName);  
 if (game.getTurn().getBuys() == 0) {  
 btnEnd.setVisibility(View.*GONE*);  
 progressDialog = new ProgressDialog(this);  
 progressDialog.setCancelable(false);  
 progressDialog.show();  
 game.changeTurn();  
 }  
  
 turnUI();  
 updateCards(true);  
 }  
  
 */\*\*  
 \* A function that handles all presses on board image views or buttons.  
 \** ***@param*** *view A View which is the view that was pressed  
 \*/* @Override  
 public void onClick(View view) {  
 if (view == btnStart) {  
 btnStart.setVisibility(View.*GONE*);  
 if (game.getGameManagerBeforeStart().isCreator())  
 game.getGameManagerBeforeStart().setStart1(true);  
 else  
 game.getGameManagerBeforeStart().setStart2(true);  
 GameRequests.*updateReadyToStart*(game);  
 }  
 else if (view == btnAutoplay) {  
 game.autoPlayTreasures();  
 turnUI();  
 updateCards(true);  
 btnAutoplay.setVisibility(View.*GONE*);  
 }  
 else if (view == btnEnd) {  
 // force end of actions or buys  
 if (btnEnd.getText().toString().equals("End Actions")) {  
 game.getTurn().setForcedActionEnd(true);  
 turnUI();  
 this.handAdapter.notifyDataSetChanged();  
 }  
 else if (btnEnd.getText().toString().equals("End Buys")) {  
 btnEnd.setVisibility(View.*GONE*);  
 progressDialog = new ProgressDialog(this);  
 progressDialog.setCancelable(false);  
 progressDialog.show();  
 game.changeTurn();  
 }  
 }  
  
 else if (this.victoryCards.containsValue(view) || this.treasureCards.containsValue(view) || this.actionCards.containsValue(view)) {  
 if ((!game.getTurn().getPhase().contains("buy") && !game.getTurn().getWaitForFunction().isWaitingForBoard())  
 || game.getTurn().isWaitingForEnemy()  
 || game.getTurn().getWaitForFunction().isWaitingForHand()  
 || game.getTurn().getWaitForFunction().isWaitingForButtonsOnly()  
 || game.getTurn().getWaitForFunction().isWaitingForActionCardsDialog())  
 return;  
  
 ImageView ivAction = view.findViewById(R.id.*ivAction*);  
 if (game.getTurn().getWaitForFunction().isWaitingForBoard()) {  
 if (game.getBoard().get(ivAction.getContentDescription().toString()) == 0  
 || !Help.*nameToCard*(game.getTurn().getWaitForFunction().getCardName())  
 .isCardToGetFromBoard(ivAction.getContentDescription().toString(), game, this))  
 return;  
  
 game.getTurn().getWaitForFunction().insertCardSelectedInHand(ivAction.getContentDescription().toString());  
 Help.*nameToCard*(game.getTurn().getWaitForFunction().getCardName()).clickOnBoard(ivAction.getContentDescription().toString(), game, this);  
 }  
 else if (game.getTurn().isMyTurn(game.getGameManagerBeforeStart())){  
 game.getTurn().setPhase("buy");  
 this.buyCard(ivAction.getContentDescription().toString());  
 }  
 }  
  
 else if (view == btnKingdomOrPlayArea) {  
 // Switches the middle layout between kingdom to play area  
 if (btnKingdomOrPlayArea.getText().toString().equals("Kingdom")) {  
 btnKingdomOrPlayArea.setText("Play Area");  
 clPlayArea.setVisibility(View.*GONE*);  
 clKingdom.setVisibility(View.*VISIBLE*);  
 }  
 else if (btnKingdomOrPlayArea.getText().toString().equals("Play Area")) {  
 btnKingdomOrPlayArea.setText("Kingdom");  
 clPlayArea.setVisibility(View.*VISIBLE*);  
 clKingdom.setVisibility(View.*GONE*);  
 }  
 }  
 else if (view == btnTrashOrLog) {  
 // Switches the right middle layout between trash to log  
 if (btnTrashOrLog.getText().toString().equals("Trash")) {  
 btnTrashOrLog.setText("Log");  
 rvTrash.setVisibility(View.*VISIBLE*);  
 lvLog.setVisibility(View.*GONE*);  
 tvInfoTitle.setText("Trash");  
 }  
 else if (btnTrashOrLog.getText().toString().equals("Log")) {  
 btnTrashOrLog.setText("Trash");  
 rvTrash.setVisibility(View.*GONE*);  
 lvLog.setVisibility(View.*VISIBLE*);  
 tvInfoTitle.setText("Log");  
 }  
 }  
  
 else if (view == btnResign) {  
 // Shows an alert dialog to ask if he is he wants sure to resign  
 AlertDialog.Builder builder = new AlertDialog.Builder(this);  
 builder.setMessage("Do you really want to resign?");  
 builder.setCancelable(false);  
 builder.setPositiveButton("Yes", new DialogInterface.OnClickListener() {  
 */\*\*  
 \* Handles yes answer and ends the dialog and the game.  
 \** ***@param*** *dialogInterface  
 \** ***@param*** *i  
 \*/* @Override  
 public void onClick(DialogInterface dialogInterface, int i) {  
 GameActivity.this.endGame();  
 game.setResigned(true);  
 dialogInterface.dismiss();  
 }  
 });  
 builder.setNegativeButton("No", new DialogInterface.OnClickListener() {  
 */\*\*  
 \* Handles no answer and ends the dialog.  
 \** ***@param*** *dialogInterface  
 \** ***@param*** *i  
 \*/* @Override  
 public void onClick(DialogInterface dialogInterface, int i) {  
 dialogInterface.dismiss();  
 }  
 });  
 AlertDialog dialog = builder.create();  
 dialog.show();  
 }  
  
 else if (this.btnActions.contains(view)) {  
 Help.*nameToCard*(game.getTurn().getWaitForFunction().getCardName()).handleButtonClicks(((Button)view).getText().toString(), game, this);  
 }  
  
 else if (view == ivArrowActionCardsPlaying) {  
 ConstraintSet constraintSetPlayArea = new ConstraintSet();  
 constraintSetPlayArea.clone(clPlayArea);  
 if (this.positionOfRVActionCardsPlaying.equals("up")) {  
 this.positionOfRVActionCardsPlaying = "down";  
 ivArrowActionCardsPlaying.setImageResource(R.mipmap.*arrow\_up*);  
 constraintSetPlayArea.connect(clActionCardsPlaying.getId(), ConstraintSet.*TOP*, rlButtonsInPlay.getId(), ConstraintSet.*BOTTOM*);  
 constraintSetPlayArea.clear(clActionCardsPlaying.getId(), ConstraintSet.*BOTTOM*);  
 }  
 else {  
 this.positionOfRVActionCardsPlaying = "up";  
 ivArrowActionCardsPlaying.setImageResource(R.mipmap.*arrow\_down*);  
 constraintSetPlayArea.connect(clActionCardsPlaying.getId(), ConstraintSet.*BOTTOM*, rlButtonsInPlay.getId(), ConstraintSet.*TOP*);  
 constraintSetPlayArea.clear(clActionCardsPlaying.getId(), ConstraintSet.*TOP*);  
 }  
 constraintSetPlayArea.applyTo(clPlayArea);  
 }  
 }  
  
 */\*\*  
 \* A function that is called after creating gameManage object.  
 \* The function handles all views before the player pressed "start game".  
 \*/* public void beforeStart() {  
 game.beforeGame();  
 if (game.getGameManagerBeforeStart().isCreator()) {  
 tvMyName.setText(String.*valueOf*(game.getGameManagerBeforeStart().getIdP1()));  
 tvEnemyName.setText(game.getGameManagerBeforeStart().getIdP2());  
 }  
 else {  
 tvMyName.setText(String.*valueOf*(game.getGameManagerBeforeStart().getIdP2()));  
 tvEnemyName.setText(game.getGameManagerBeforeStart().getIdP1());  
 }  
  
 rvHand.setLayoutManager(new LinearLayoutManager(getApplicationContext(), LinearLayoutManager.*HORIZONTAL*, false));  
 this.game.getPlayer().updateArrayHand();  
 handAdapter = new HandAdapter(this.game.getPlayer().getArrayHand(), this);  
 rvHand.setAdapter(handAdapter);  
  
 rvActionCardsPlaying.setLayoutManager(new LinearLayoutManager(getApplicationContext(), LinearLayoutManager.*HORIZONTAL*, false));  
 actionCardsPlayingAdapter = new ActionCardPlayingAdapter(game.getTurn().getWaitForFunction().getCardsForDialog(), this);  
 rvActionCardsPlaying.setAdapter(actionCardsPlayingAdapter);  
  
 logAdapter = new LogAdapter(this,0,0, game.getLog(), game);  
 lvLog.setAdapter(logAdapter);  
 lvLog.setDivider(null);  
 lvLog.setDividerHeight(0);  
 lvLog.setTranscriptMode(AbsListView.*TRANSCRIPT\_MODE\_NORMAL*);  
  
 rvTrash.setLayoutManager(new GridLayoutManager(this, 2));  
 this.game.updateArrayTrash();  
 trashAdapter = new TrashAdapter(this.game.getArrayTrash(), this);  
 rvTrash.setAdapter(trashAdapter);  
  
 this.insertDataToHashMaps();  
 this.uploadBoard();  
  
 for (String cardName : actionBigCards.keySet()) {  
 ImageView iv = actionBigCards.get(cardName).findViewById(R.id.*ivAction*);  
 iv.setContentDescription(cardName);  
 iv.setImageResource(Help.*nameToCard*(cardName).getImageSource());  
 }  
  
 this.updateCountOfBoard();  
  
 btnStart.setOnClickListener(this);  
 btnEnd.setOnClickListener(this);  
 btnAutoplay.setOnClickListener(this);  
 btnKingdomOrPlayArea.setOnClickListener(this);  
 btnTrashOrLog.setOnClickListener(this);  
 btnResign.setOnClickListener(this);  
  
 for (Button button : btnActions) {  
 button.setOnClickListener(this);  
 }  
  
 for (String cardName : victoryCards.keySet()) {  
 victoryCards.get(cardName).setOnLongClickListener(this);  
 victoryCards.get(cardName).setOnClickListener(this);  
 }  
  
 for (String cardName : treasureCards.keySet()) {  
 treasureCards.get(cardName).setOnLongClickListener(this);  
 treasureCards.get(cardName).setOnClickListener(this);  
 }  
  
 for (String cardName : actionCards.keySet()) {  
 actionCards.get(cardName).setOnLongClickListener(this);  
 actionCards.get(cardName).setOnClickListener(this);  
 }  
  
 for (String cardName : actionBigCards.keySet()) {  
 actionBigCards.get(cardName).setOnLongClickListener(this);  
 }  
  
 ivMyDiscard.setOnLongClickListener(this);  
 ivEnemyDiscard.setOnLongClickListener(this);  
 }  
  
 */\*\*  
 \* A function that is called when both players pressed "start game" and starts the game.  
 \*/* public void startGame() {  
 game.setStarted(true);  
 tvTurn.setVisibility(View.*VISIBLE*);  
 clTurn.setVisibility(View.*VISIBLE*);  
 tvMyVP.setVisibility(View.*VISIBLE*);  
 tvEnemyVP.setVisibility(View.*VISIBLE*);  
 game.startGame();  
 if (!this.game.getTurn().isMyTurn(game.getGameManagerBeforeStart())) {  
 // when the player doesn't starts  
 btnEnd.setVisibility(View.*GONE*);  
 btnAutoplay.setVisibility(View.*GONE*);  
 GameRequests.*getDataInGame*(false, game);  
 }  
 else {  
 // when the player starts  
 GameRequests.*uploadDataInGame*(false, game);  
 turnUI();  
 updateCards(true);  
 game.addTurnNumberToLog();  
 }  
  
 myDeck.setOnLongClickListener(this);  
 enemyDeck.setOnLongClickListener(this);  
 }  
  
 */\*\*  
 \* A function that inserts all board cards to hash maps by their name reference.  
 \*/* public void insertDataToHashMaps() {  
 victoryCards.put("Province", findViewById(R.id.*province*));  
 victoryCards.put("Duchy", findViewById(R.id.*duchy*));  
 victoryCards.put("Estate", findViewById(R.id.*estate*));  
 victoryCards.put("Curse", findViewById(R.id.*curse*));  
  
 treasureCards.put("Gold", findViewById(R.id.*gold*));  
 treasureCards.put("Silver", findViewById(R.id.*silver*));  
 treasureCards.put("Copper", findViewById(R.id.*copper*));  
  
 for(int i = 0; i < 10; i++) {  
 int resID = getResources().getIdentifier(("action" + (i+1)), "id", getPackageName());  
 actionCards.put(game.getActionCards()[i], findViewById(resID));  
 }  
  
 for(int i = 0; i < 10; i++) {  
 int resID = getResources().getIdentifier(("actionBig" + (i+1)), "id", getPackageName());  
 actionBigCards.put(game.getActionCards()[i], findViewById(resID));  
 }  
 }  
  
 */\*\*  
 \* A function that is called when the game is ended,  
 \* either if someone resigned or the game was ended according to the rules.  
 \*/* public void endGame() {  
 btnEnd.setVisibility(View.*GONE*);  
 btnAutoplay.setVisibility(View.*GONE*);  
 progressDialog = new ProgressDialog(this);  
 if (game.getEnemyData().isResigned())  
 progressDialog.setMessage(game.getGameManagerBeforeStart().getEnemyId() + " resigned...");  
 else if (game.isResigned())  
 progressDialog.setMessage("You resigned...");  
 else  
 progressDialog.setMessage("Game is ended...");  
 progressDialog.setCancelable(false);  
 progressDialog.show();  
 }  
  
 */\*\*  
 \* A function that is called when the game is ended  
 \* and finishes the intent to prepareGameActivity with the results of the game.  
 \*/* public void startPrepareActivity() {  
 endProgressBar(false);  
 game.getGameManagerBeforeStart().setReady1(false);  
 game.getGameManagerBeforeStart().setReady2(false);  
 game.getGameManagerBeforeStart().setStart1(false);  
 game.getGameManagerBeforeStart().setStart2(false);  
 Intent intent = new Intent();  
 setResult(*RESULT\_OK*, intent);  
  
 if (game.getEnemyData().isResigned()) {  
 intent.putExtra("gameManagerBeforeStart", game.getGameManagerBeforeStart());  
 intent.putExtra("player", game.getGameManagerBeforeStart().getEnemyId());  
 intent.putExtra("result", "resign");  
 finish();  
 return;  
 }  
 else if (game.isResigned()) {  
 intent.putExtra("gameManagerBeforeStart", game.getGameManagerBeforeStart());  
 intent.putExtra("player", game.getGameManagerBeforeStart().getMyId());  
 intent.putExtra("result", "resign");  
 finish();  
 return;  
 }  
 int finalVP1 = game.getPlayer().getVictoryPoints(game);  
 int finalVP2 = game.getEnemyData().getVictoryPoints();  
 if (!game.getGameManagerBeforeStart().isCreator()) {  
 int temp = finalVP1;  
 finalVP1 = finalVP2;  
 finalVP2 = temp;  
 }  
 intent.putExtra("gameManagerBeforeStart", game.getGameManagerBeforeStart());  
  
 intent.putExtra("P1VP", finalVP1);  
 intent.putExtra("P2VP", finalVP2);  
 if (finalVP1 > finalVP2) {  
 intent.putExtra("player", game.getGameManagerBeforeStart().getIdP1());  
 intent.putExtra("result", "win");  
 }  
 else if (finalVP1 < finalVP2) {  
 intent.putExtra("player", game.getGameManagerBeforeStart().getIdP2());  
 intent.putExtra("result", "win");  
 }  
 else  
 intent.putExtra("result", "draw");  
  
 finish();  
 }  
  
 */\*\*  
 \* A function that uploads the board images by type.  
 \*/* public void uploadBoard() {  
 uploadByType(this.victoryCards);  
 uploadByType(this.treasureCards);  
 uploadByType(this.actionCards);  
 }  
  
 */\*\*  
 \* A function that puts the image to the image views of the board.  
 \** ***@param*** *hashMap A HashMap with card names and their references  
 \*/* public void uploadByType(HashMap<String, View> hashMap) {  
 for (String cardName : hashMap.keySet()) {  
 ImageView iv = hashMap.get(cardName).findViewById(R.id.*ivAction*);  
 iv.setContentDescription(cardName);  
 iv.setImageResource(Help.*nameToCard*(cardName).getShortImageSource());  
 }  
 }  
  
 */\*\*  
 \* A function that is called after any function which is doing changes in gameManager  
 \* and updates the changes on screen.  
 \** ***@param*** *updateHand A Boolean which is true when if update hand is needed and false if not  
 \*/* public void updateCards(boolean updateHand) {  
 updateCountOfBoard();  
 logAdapter.notifyDataSetChanged();  
 if (updateHand) {  
 this.game.getPlayer().updateArrayHand();  
 this.handAdapter.notifyDataSetChanged();  
 }  
  
 if (game.isStarted()) {  
 tvTurn.setText(game.getTurn().toString());  
  
 // My Data  
 tvMyVP.setText(String.*valueOf*(game.getPlayer().getVictoryPoints(game)).concat(" VP"));  
 int sizeDeck = game.getPlayer().getDeck().size();  
 if (sizeDeck > 0) {  
 myDeck.setVisibility(View.*VISIBLE*);  
 TextView tv = myDeck.findViewById(R.id.*countAction*);  
 tv.setText(String.*valueOf*(sizeDeck));  
 } else  
 myDeck.setVisibility(View.*INVISIBLE*);  
  
 if (!game.getPlayer().getDiscard().isEmpty()) {  
 ivMyDiscard.setVisibility(View.*VISIBLE*);  
 String cardName = game.getPlayer().getDiscard().get(game.getPlayer().getDiscard().size() - 1);  
 ivMyDiscard.setImageResource(Help.*nameToCard*(cardName).getShortImageSource());  
 ivMyDiscard.setContentDescription(cardName);  
 } else  
 ivMyDiscard.setVisibility(View.*INVISIBLE*);  
  
 // Enemy Data  
 tvEnemyVP.setText(String.*valueOf*(game.getEnemyData().getVictoryPoints()).concat(" VP"));  
 int sizeEnemyDeck = game.getEnemyData().getDeckSize();  
 if (sizeEnemyDeck > 0) {  
 enemyDeck.setVisibility(View.*VISIBLE*);  
 TextView tv = enemyDeck.findViewById(R.id.*countAction*);  
 tv.setText(String.*valueOf*(sizeEnemyDeck));  
 } else  
 enemyDeck.setVisibility(View.*INVISIBLE*);  
  
 if (!game.getEnemyData().getLastCardOnDiscard().equals("")) {  
 ivEnemyDiscard.setVisibility(View.*VISIBLE*);  
 String cardName = game.getEnemyData().getLastCardOnDiscard();  
 ivEnemyDiscard.setImageResource(Help.*nameToCard*(cardName).getShortImageSource());  
 ivEnemyDiscard.setContentDescription(cardName);  
 } else  
 ivEnemyDiscard.setVisibility(View.*INVISIBLE*);  
  
 }  
 int sizeEnemyHand = game.getEnemyData().getHandSize();  
 enemyHand.setVisibility(View.*VISIBLE*);  
 TextView tv = enemyHand.findViewById(R.id.*countAction*);  
 tv.setText(String.*valueOf*(sizeEnemyHand));  
 }  
  
 */\*\*  
 \* A function that updates on screen the board if any changes done in gameManager.  
 \*/* public void updateCountOfBoard() {  
 updateCountByType(this.victoryCards);  
 updateCountByType(this.treasureCards);  
 updateCountByType(this.actionCards);  
 }  
  
 */\*\*  
 \* A function that updates the count of any image view on the board.  
 \** ***@param*** *hashMap A HashMap with card names and their references  
 \*/* public void updateCountByType(HashMap<String, View> hashMap) {  
 for (String cardName : hashMap.keySet()) {  
 TextView tv = hashMap.get(cardName).findViewById(R.id.*countAction*);  
 tv.setText(String.*valueOf*(game.getBoard().get(cardName)));  
 }  
 }  
  
 */\*\*  
 \* A function that is called when changing turn.  
 \*/* public void turnActions() {  
 if (!this.game.getTurn().isMyTurn(game.getGameManagerBeforeStart())) {  
 btnEnd.setVisibility(View.*GONE*);  
 btnAutoplay.setVisibility(View.*GONE*);  
 }  
 else {  
 turnUI();  
 updateCards(true);  
 }  
  
 if (game.gameEnded()) {  
 game.setGameEnded(true);  
 this.endGame();  
 GameRequests.*endGame*(game);  
 }  
 }  
  
 */\*\*  
 \* A function that updates the turn buttons according to the turn phase.  
 \*/* public void turnUI() {  
 if (game.getTurn().isMyTurn(game.getGameManagerBeforeStart())) {  
 if (game.getTurn().isWaitingForEnemy()  
 || game.getTurn().getWaitForFunction().isWaitingForHand()  
 || game.getTurn().getWaitForFunction().isWaitingForBoard()  
 || game.getTurn().getWaitForFunction().isWaitingForButtonsOnly()  
 || game.getTurn().getWaitForFunction().isWaitingForActionCardsDialog()) {  
 btnEnd.setVisibility(View.*GONE*);  
 btnAutoplay.setVisibility(View.*GONE*);  
 return;  
 }  
 if (game.getPlayer().containsTypeCards("action") && (game.getTurn().getActions() > 0 || game.getTimes().peek() != 1) && !game.getTurn().getForcedActionEnd()) {  
 this.game.getTurn().setPhase("play-action");  
 actionsUI();  
 }  
 else {  
 this.game.getTurn().setPhase("play-treasure-buy");  
 buysUI();  
 }  
 }  
 }  
  
 */\*\*  
 \* A function that is called when the phase is play-action and updates game buttons.  
 \*/* public void actionsUI() {  
 btnEnd.setVisibility(View.*VISIBLE*);  
 btnEnd.setText("End Actions");  
 btnAutoplay.setVisibility(View.*GONE*);  
 }  
  
 */\*\*  
 \* A function that is called when the phase is play-treasure-buy and updates game buttons.  
 \*/* public void buysUI() {  
 btnEnd.setVisibility(View.*VISIBLE*);  
 btnEnd.setText("End Buys");  
 if (game.getPlayer().containsTypeCards("treasure"))  
 btnAutoplay.setVisibility(View.*VISIBLE*);  
 else  
 btnAutoplay.setVisibility(View.*GONE*);  
 }  
  
 */\*\*  
 \* A function that handles waiting for pressing on hand after playing some special cards.  
 \** ***@param*** *cardName A String which is the name of the card that is waiting for clicking on hand  
 \** ***@param*** *minAmount An Integer with the minimum of cards that should be selected from hand  
 \** ***@param*** *maxAmount An Integer with the maximum of cards that should be selected from hand  
 \** ***@param*** *typeOfAction A String with the type of action that will be  
 \* done with the selected cards (trash, discard...)  
 \** ***@param*** *handleClickOnCard A Boolean which is true if the card should handle every  
 \* click on card in hand and false if not  
 \*/* public void waitForHand(String cardName, int minAmount, int maxAmount, String typeOfAction, boolean handleClickOnCard) {  
 game.getTurn().getWaitForFunction().handleWaitingForHand(cardName, minAmount, maxAmount, typeOfAction, handleClickOnCard);  
 this.handAdapter.notifyDataSetChanged();  
 }  
  
 */\*\*  
 \* A function that handles waiting for pressing on board after playing some special cards.  
 \** ***@param*** *cardName A String which is the name of the card that is waiting for clicking on board  
 \** ***@param*** *minAmount An Integer with the minimum of cards that should be selected from board  
 \** ***@param*** *maxAmount An Integer with the maximum of cards that should be selected from board  
 \*/* public void waitForBoard(String cardName, int minAmount, int maxAmount) {  
 game.getTurn().getWaitForFunction().handleWaitingForBoard(cardName, minAmount, maxAmount);  
 }  
  
 */\*\*  
 \* A function that sets the visibility to the special buttons that appears for some cards.  
 \** ***@param*** *i An Integer with the place in btnActions of the specific button  
 \** ***@param*** *visibility An Integer with the visibility the should be set to this button  
 \*/* public void setVisibilityForAction(int i, int visibility) {  
 btnActions.get(i).setVisibility(visibility);  
 }  
  
 */\*\*  
 \* A function that makes the amount of special buttons invisible  
 \** ***@param*** *n An Integer with the amount of buttons that should be invisible  
 \*/* public void invisibleButtons(int n) {  
 for (int i = 0; i < n; i++)  
 btnActions.get(i).setVisibility(View.*GONE*);  
 }  
  
 */\*\*  
 \* A function that uploads the text of some buttons that are used for a card.  
 \** ***@param*** *textOnButtons A String Array with the text on the buttons  
 \** ***@param*** *hasUndoAndConfirm A Boolean which is true if there is undo and confirm  
 \* button and false if not  
 \*/* public void uploadActionButtons(String[] textOnButtons, boolean hasUndoAndConfirm) {  
 for (int i = 0; i < textOnButtons.length; i++)  
 btnActions.get(i).setText(textOnButtons[i]);  
  
 game.getTurn().getWaitForFunction().setHasUndoAndConfirm(hasUndoAndConfirm);  
 if (hasUndoAndConfirm)  
 updateActionButtons();  
 else  
 this.turnUI();  
 }  
  
 */\*\*  
 \* A function that updates the undo and confirm buttons visibility,  
 \* which are common buttons for special cards.  
 \*/* public void updateActionButtons() {  
 // 0 is always undo  
 if ((game.getTurn().getWaitForFunction().isWaitingForHand() || game.getTurn().getWaitForFunction().isWaitingForActionCardsDialog())  
 && Help.*sizeOfHash*(game.getTurn().getWaitForFunction().getCardsForActionCardPlay()) > 0)  
 btnActions.get(0).setVisibility(View.*VISIBLE*);  
 else  
 btnActions.get(0).setVisibility(View.*GONE*);  
  
 // 1 is always confirm  
 if ((game.getTurn().getWaitForFunction().isWaitingForHand() || game.getTurn().getWaitForFunction().isWaitingForActionCardsDialog())  
 && (Help.*sizeOfHash*(game.getTurn().getWaitForFunction().getCardsForActionCardPlay()) <= game.getTurn().getWaitForFunction().getMaxAmount()  
 || game.getTurn().getWaitForFunction().getMaxAmount() == -1)  
 && Help.*sizeOfHash*(game.getTurn().getWaitForFunction().getCardsForActionCardPlay()) >= game.getTurn().getWaitForFunction().getMinAmount())  
 btnActions.get(1).setVisibility(View.*VISIBLE*);  
 else  
 btnActions.get(1).setVisibility(View.*GONE*);  
  
 this.turnUI();  
 }  
  
 public RecyclerView.Adapter getHandAdapter() {  
 return this.handAdapter;  
 }  
  
 public RecyclerView.Adapter getActionCardsPlayingAdapter() {  
 return this.actionCardsPlayingAdapter;  
 }  
  
 */\*\*  
 \* A function that sets the visibility of the Action Cards Playing RecyclerView  
 \* and sets the place of the RecyclerView to be up.  
 \** ***@param*** *visibility  
 \*/* public void setVisibilityForRVActionCardsPlaying(int visibility) {  
 if (visibility == View.*INVISIBLE*) {  
 this.clActionCardsPlaying.setVisibility(View.*INVISIBLE*);  
 return;  
 }  
  
 ConstraintSet constraintSetPlayArea = new ConstraintSet();  
 constraintSetPlayArea.clone(clPlayArea);  
  
 constraintSetPlayArea.connect(clActionCardsPlaying.getId(), ConstraintSet.*BOTTOM*, rlButtonsInPlay.getId(), ConstraintSet.*TOP*);  
 constraintSetPlayArea.clear(clActionCardsPlaying.getId(), ConstraintSet.*TOP*);  
 this.positionOfRVActionCardsPlaying = "up";  
 ivArrowActionCardsPlaying.setImageResource(R.mipmap.*arrow\_down*);  
 constraintSetPlayArea.applyTo(clPlayArea);  
  
 this.clActionCardsPlaying.setVisibility(View.*VISIBLE*);  
 }  
  
 public RecyclerView.Adapter getTrashAdapter() {  
 return this.trashAdapter;  
 }  
  
 /\*  
 public float getDefaultSize() {  
 return tvTurn.getTextSize();  
 }  
 \*/  
  
 */\*\*  
 \* A function that ends the progress bar  
 \** ***@param*** *isHandleBackground A Boolean which is true only when  
 \* the progress bar was shown at the start of the game.  
 \*/* public void endProgressBar(boolean isHandleBackground) {  
 if (progressDialog != null && progressDialog.isShowing()) {  
 progressDialog.dismiss();  
 if (isHandleBackground)  
 background.setVisibility(View.*INVISIBLE*);  
 }  
 }  
}

**Adapters:**

TablesAdapter

*/\*\*  
 \* TablesAdapter is the adapter of the ListView of all the online tables.  
 \* The adapter updates the tables shown in real time.  
 \*/*package com.example.dominion\_game.activities;  
  
import android.app.Activity;  
import android.content.Context;  
import android.content.Intent;  
import android.content.SharedPreferences;  
import android.view.View;  
import android.view.ViewGroup;  
import android.widget.ArrayAdapter;  
import android.widget.Button;  
import android.widget.TextView;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.classes.GameManagerBeforeStart;  
import com.example.dominion\_game.classes.Table;  
  
import java.util.List;  
  
  
public class TablesAdapter extends ArrayAdapter<Table> {  
  
 Context context;  
 List<Table> data;  
 Table tempTable;  
 OnlineTablesActivity onlineTablesActivity;  
 SharedPreferences sharedPreferences;  
  
 */\*\*  
 \* The constructor  
 \** ***@param*** *context  
 \** ***@param*** *resource  
 \** ***@param*** *textViewResourceId  
 \** ***@param*** *data An ArrayList of LogLine with all tables online  
 \** ***@param*** *onlineTablesActivity A reference to onlineTablesActivity  
 \*/* public TablesAdapter(Context context, int resource, int textViewResourceId, List<Table> data, OnlineTablesActivity onlineTablesActivity) {  
 super(context, resource, textViewResourceId, data);  
  
 this.context = context;  
 this.data = data;  
 this.onlineTablesActivity = onlineTablesActivity;  
 this.sharedPreferences = onlineTablesActivity.getSharedPreferences("account", Context.*MODE\_PRIVATE*);  
 }  
  
 */\*\*  
 \* A function that creates the layout of every item in ListView  
 \* and updates the data inside it according to the data ArrayList.  
 \** ***@param*** *position An Integer of the position of the view in the ArrayList  
 \** ***@param*** *convertView  
 \** ***@param*** *parent A ViewGroup of the views in ListView according to the size of data  
 \** ***@return*** *A View which is the view of the item in ListView according to the data ArrayList  
 \*/* @Override  
 public View getView(int position, View convertView, ViewGroup parent) {  
 View view = ((Activity) context).getLayoutInflater().inflate(R.layout.*table\_layout*, parent, false);  
 TextView tvName = view.findViewById(R.id.*tvName*);  
 Button btnJoin = view.findViewById(R.id.*btnJoin*);  
 tempTable = data.get(position);  
 tvName.setText(tempTable.getCreator());  
 if (position == 0) {  
 btnJoin.setVisibility(View.*INVISIBLE*);  
 return view;  
 }  
  
 btnJoin.setOnClickListener(new View.OnClickListener() {  
 */\*\*  
 \* A function that handles a click on the "play" button on every item in the ListView.  
 \** ***@param*** *view A View which is the item that was pressed  
 \*/* @Override  
 public void onClick(View view) {  
 onlineTablesActivity.setFinished(true);  
 Intent intent = new Intent(context, PrepareGameActivity.class);  
 intent.putExtra("gameManagerBeforeStart", new GameManagerBeforeStart(tempTable.getId(), sharedPreferences.getString("username", "")));  
 context.startActivity(intent);  
 }  
 });  
 return view;  
 }  
  
 */\*\*  
 \* A function that disables the option to click on any item in the ListView.  
 \** ***@param*** *position An Integer of the position of the view in the ArrayList  
 \** ***@return*** *A Boolean which is always false  
 \*/* @Override  
 public boolean isEnabled(int position) {  
 return false;  
 }  
}

HandAdapter

*/\*\*  
 \* HandAdapter is the adapter of the RecyclerView of hand of the player.  
 \* The adapter updates the cards shown according to his hand.  
 \*/*package com.example.dominion\_game.activities;  
  
import android.util.Pair;  
import android.view.LayoutInflater;  
import android.view.View;  
import android.view.ViewGroup;  
import android.widget.ImageView;  
import android.widget.TextView;  
  
import androidx.recyclerview.widget.RecyclerView;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.classes.Help;  
  
import java.util.ArrayList;  
  
public class HandAdapter extends RecyclerView.Adapter<HandAdapter.MyViewHolder> {  
  
 private ArrayList<Pair<String, Integer>> objects;  
 private GameActivity gameActivity;  
  
 */\*\*  
 \* The constructor  
 \** ***@param*** *objects An ArrayList of Pairs of the card name with the count of this card in hand.  
 \** ***@param*** *gameActivity A reference to GameActivity  
 \*/* public HandAdapter(ArrayList<Pair<String, Integer>> objects, GameActivity gameActivity) {  
 this.objects = objects;  
 this.gameActivity = gameActivity;  
 }  
  
 */\*\*  
 \* A view holder for every item in RecyclerView  
 \*/* public class MyViewHolder extends RecyclerView.ViewHolder implements View.OnLongClickListener, View.OnClickListener {  
 ImageView ivAction, ivGreenMargin, ivRedMargin, ivYellowMargin, ivGreenX, ivRedX, ivYellowX;  
 TextView countAction, countActionForPlayAction;  
  
 */\*\*  
 \* The constructor  
 \** ***@param*** *view A View of one item in RecyclerView  
 \*/* public MyViewHolder(View view) {  
 super(view);  
 ivAction = view.findViewById(R.id.*ivAction*);  
 ivGreenMargin = view.findViewById(R.id.*ivGreenMargin*);  
 ivRedMargin = view.findViewById(R.id.*ivRedMargin*);  
 ivYellowMargin = view.findViewById(R.id.*ivYellowMargin*);  
 ivGreenX = view.findViewById(R.id.*ivGreenX*);  
 ivRedX = view.findViewById(R.id.*ivRedX*);  
 ivYellowX = view.findViewById(R.id.*ivYellowX*);  
 countActionForPlayAction = view.findViewById(R.id.*countActionForPlayAction*);  
 countAction = view.findViewById(R.id.*countAction*);  
 view.setOnLongClickListener(this);  
 view.setOnClickListener(this);  
 }  
  
 */\*\*  
 \* A function that handles long press on an item in the RecyclerView  
 \* The function shows a dialog of the card pressed in big.  
 \** ***@param*** *view A View of an item in RecyclerView that was pressed long  
 \** ***@return*** *A Boolean which is always true  
 \*/* @Override  
 public boolean onLongClick(View view) {  
 ImageView iv = gameActivity.cardDialog.findViewById(R.id.*ivAction*);  
 ImageView resourceIv = view.findViewById(R.id.*ivAction*);  
 iv.setImageResource(Help.*nameToCard*(resourceIv.getContentDescription().toString()).getImageSource());  
 gameActivity.cardDialog.show();  
 return true;  
 }  
  
 */\*\*  
 \* A function that handles a press on an item in the RecyclerView  
 \* The function checks if the card can be played  
 \* and calls a function in gameActivity to play the card.  
 \** ***@param*** *view A View of an item in RecyclerView that was pressed  
 \*/* @Override  
 public void onClick(View view) {  
 if ((!gameActivity.game.getTurn().getPhase().contains("play") && !gameActivity.game.getTurn().getWaitForFunction().isWaitingForHand())  
 || gameActivity.game.getTurn().isWaitingForEnemy()  
 || gameActivity.game.getTurn().getWaitForFunction().isWaitingForBoard()  
 || gameActivity.game.getTurn().getWaitForFunction().isWaitingForButtonsOnly()  
 || gameActivity.game.getTurn().getWaitForFunction().isWaitingForActionCardsDialog())  
 return;  
  
 if (gameActivity.game.getTurn().getWaitForFunction().isWaitingForHand()) {  
 ImageView resourceIv = view.findViewById(R.id.*ivAction*);  
 if (!Help.*nameToCard*(gameActivity.game.getTurn().getWaitForFunction().getCardName())  
 .isCardToUse(resourceIv.getContentDescription().toString(), gameActivity.game, gameActivity)  
 || Help.*sizeOfHash*(gameActivity.game.getTurn().getWaitForFunction().getCardsForActionCardPlay()) == gameActivity.game.getTurn().getWaitForFunction().getMaxAmount())  
 return;  
  
 if (gameActivity.game.getTurn().getWaitForFunction().isHandleClickOnCard()  
 && !Help.*nameToCard*(gameActivity.game.getTurn().getWaitForFunction().getCardName()).isMarkCardSelectedFromHandWhenHandle()) {  
 gameActivity.game.getTurn().getWaitForFunction().insertCardSelectedInHand(resourceIv.getContentDescription().toString());  
 Help.*nameToCard*(gameActivity.game.getTurn().getWaitForFunction().getCardName())  
 .handleClickOnHandOrDialog(resourceIv.getContentDescription().toString(), gameActivity.game, gameActivity);  
 return;  
 }  
  
 TextView tvCount = view.findViewById(R.id.*countAction*);  
 int count = Integer.*valueOf*(tvCount.getText().toString());  
 TextView tvCountSelected = view.findViewById(R.id.*countActionForPlayAction*);  
 int countSelected = gameActivity.game.getTurn().getWaitForFunction().getCardAmountInCardsInHand(resourceIv.getContentDescription().toString());  
  
 if (countSelected == count)  
 return;  
  
 if (gameActivity.game.getTurn().getWaitForFunction().getTypeOfAction().equals("trash"))  
 view.findViewById(R.id.*ivRedX*).setVisibility(View.*VISIBLE*);  
 else  
 view.findViewById(R.id.*ivRedX*).setVisibility(View.*INVISIBLE*);  
  
 if (!gameActivity.game.getTurn().getWaitForFunction().getTypeOfAction().equals("trash"))  
 view.findViewById(R.id.*ivYellowX*).setVisibility(View.*VISIBLE*);  
 else  
 view.findViewById(R.id.*ivYellowX*).setVisibility(View.*INVISIBLE*);  
  
 tvCountSelected.setText(String.*valueOf*(countSelected + 1));  
 if (count > 1)  
 tvCountSelected.setVisibility(View.*VISIBLE*);  
 else  
 tvCountSelected.setVisibility(View.*INVISIBLE*);  
  
 gameActivity.game.getTurn().getWaitForFunction().insertCardSelectedInHand(resourceIv.getContentDescription().toString());  
 if (gameActivity.game.getTurn().getWaitForFunction().isHandleClickOnCard()  
 && Help.*nameToCard*(gameActivity.game.getTurn().getWaitForFunction().getCardName()).isMarkCardSelectedFromHandWhenHandle()) {  
 Help.*nameToCard*(gameActivity.game.getTurn().getWaitForFunction().getCardName())  
 .handleClickOnHandOrDialog(resourceIv.getContentDescription().toString(), gameActivity.game, gameActivity);  
 return;  
 }  
 if (gameActivity.game.getTurn().getWaitForFunction().isHasUndoAndConfirm())  
 gameActivity.updateActionButtons();  
  
 }  
 else if (gameActivity.game.getTurn().getPhase().equals("play-action")  
 && gameActivity.game.getTurn().isMyTurn(gameActivity.game.getGameManagerBeforeStart())) {  
 ImageView resourceIv = view.findViewById(R.id.*ivAction*);  
 if (!Help.*nameToCard*(resourceIv.getContentDescription().toString()).getType().equals("action"))  
 return;  
  
 gameActivity.useCard(resourceIv.getContentDescription().toString());  
 }  
 else if (gameActivity.game.getTurn().getPhase().equals("play-treasure-buy")  
 && gameActivity.game.getTurn().isMyTurn(gameActivity.game.getGameManagerBeforeStart())) {  
 ImageView resourceIv = view.findViewById(R.id.*ivAction*);  
 if (!Help.*nameToCard*(resourceIv.getContentDescription().toString()).getType().equals("treasure"))  
 return;  
  
 gameActivity.useCard(resourceIv.getContentDescription().toString());  
 }  
 }  
 }  
  
 */\*\*  
 \* A function that creates the layout for every item in RecyclerView.  
 \** ***@param*** *parent  
 \** ***@param*** *viewType  
 \** ***@return*** *A view for each item in RecyclerView  
 \*/* @Override  
 public HandAdapter.MyViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {  
 View itemView = LayoutInflater.*from*(parent.getContext())  
 .inflate(R.layout.*card*, parent, false);  
  
 return new MyViewHolder(itemView);  
 }  
  
 */\*\*  
 \* // A function that replaces the contents of every item in RecyclerView.  
 \** ***@param*** *holder A view for each item in RecyclerView  
 \** ***@param*** *position An Integer of the position of the view in the ArrayList  
 \*/* @Override  
 public void onBindViewHolder(HandAdapter.MyViewHolder holder, int position) {  
 holder.ivAction.setImageResource(Help.*nameToCard*(this.objects.get(position).first).getImageSource());  
 holder.countAction.setText(String.*valueOf*(this.objects.get(position).second));  
 if (this.objects.get(position).second > 1)  
 holder.countAction.setVisibility(View.*VISIBLE*);  
 else  
 holder.countAction.setVisibility(View.*INVISIBLE*);  
  
 holder.ivAction.setContentDescription(this.objects.get(position).first);  
 if (gameActivity.game.getTurn().getWaitForFunction().isWaitingForHand()  
 && Help.*nameToCard*(gameActivity.game.getTurn().getWaitForFunction().getCardName())  
 .isCardToUse(this.objects.get(position).first, gameActivity.game, gameActivity)) {  
 if (gameActivity.game.getTurn().getWaitForFunction().getTypeOfAction().equals("trash")) {  
 holder.ivRedMargin.setVisibility(View.*VISIBLE*);  
 if (gameActivity.game.getTurn().getWaitForFunction().getCardsForActionCardPlay().get(this.objects.get(position).first) == null) {  
 holder.ivRedX.setVisibility(View.*INVISIBLE*);  
 holder.countActionForPlayAction.setVisibility(View.*INVISIBLE*);  
 }  
 else {  
 holder.ivRedX.setVisibility(View.*VISIBLE*);  
 if (gameActivity.game.getTurn().getWaitForFunction().getCardsForActionCardPlay().get(this.objects.get(position).first) > 1) {  
 holder.countActionForPlayAction.setText(String.*valueOf*(gameActivity.game.getTurn().getWaitForFunction().getCardsForActionCardPlay().get(this.objects.get(position).first)));  
 holder.countActionForPlayAction.setVisibility(View.*VISIBLE*);  
 }  
 else  
 holder.countActionForPlayAction.setVisibility(View.*INVISIBLE*);  
 }  
 }  
 else {  
 holder.ivRedMargin.setVisibility(View.*INVISIBLE*);  
 holder.ivRedX.setVisibility(View.*INVISIBLE*);  
 holder.countActionForPlayAction.setVisibility(View.*INVISIBLE*);  
 }  
  
 if (gameActivity.game.getTurn().getWaitForFunction().getTypeOfAction().equals("use"))  
 holder.ivGreenMargin.setVisibility(View.*VISIBLE*);  
 else  
 holder.ivGreenMargin.setVisibility(View.*INVISIBLE*);  
  
 if (!gameActivity.game.getTurn().getWaitForFunction().getTypeOfAction().equals("trash")  
 && !gameActivity.game.getTurn().getWaitForFunction().getTypeOfAction().equals("use")) {  
 holder.ivYellowMargin.setVisibility(View.*VISIBLE*);  
 if (gameActivity.game.getTurn().getWaitForFunction().getCardsForActionCardPlay().get(this.objects.get(position).first) == null) {  
 holder.ivYellowX.setVisibility(View.*INVISIBLE*);  
 holder.countActionForPlayAction.setVisibility(View.*INVISIBLE*);  
 }  
 else {  
 holder.ivYellowX.setVisibility(View.*VISIBLE*);  
 if (gameActivity.game.getTurn().getWaitForFunction().getCardsForActionCardPlay().get(this.objects.get(position).first) > 1) {  
 holder.countActionForPlayAction.setText(String.*valueOf*(gameActivity.game.getTurn().getWaitForFunction().getCardsForActionCardPlay().get(this.objects.get(position).first)));  
 holder.countActionForPlayAction.setVisibility(View.*VISIBLE*);  
 }  
 else  
 holder.countActionForPlayAction.setVisibility(View.*INVISIBLE*);  
 }  
 }  
 else {  
 holder.ivYellowMargin.setVisibility(View.*INVISIBLE*);  
 holder.ivYellowX.setVisibility(View.*INVISIBLE*);  
 holder.countActionForPlayAction.setVisibility(View.*INVISIBLE*);  
 }  
 }  
 else {  
 holder.ivGreenMargin.setVisibility(View.*INVISIBLE*);  
 holder.ivRedMargin.setVisibility(View.*INVISIBLE*);  
 holder.ivRedX.setVisibility(View.*INVISIBLE*);  
 holder.ivYellowMargin.setVisibility(View.*INVISIBLE*);  
 holder.ivYellowX.setVisibility(View.*INVISIBLE*);  
 holder.countActionForPlayAction.setVisibility(View.*INVISIBLE*);  
 }  
  
 if (!gameActivity.game.getTurn().getWaitForFunction().isWaitingForHand()  
 && !gameActivity.game.getTurn().isWaitingForEnemy()  
 && !gameActivity.game.getTurn().getWaitForFunction().isWaitingForBoard()  
 && !gameActivity.game.getTurn().getWaitForFunction().isWaitingForButtonsOnly()  
 && !gameActivity.game.getTurn().getWaitForFunction().isWaitingForActionCardsDialog()  
 && gameActivity.game.getTurn().isMyTurn(gameActivity.game.getGameManagerBeforeStart())  
 && ((Help.*nameToCard*(this.objects.get(position).first).getType().equals("action")  
 && gameActivity.game.getTurn().getPhase().equals("play-action"))  
 || (Help.*nameToCard*(this.objects.get(position).first).getType().equals("treasure")  
 && gameActivity.game.getTurn().getPhase().equals("play-treasure-buy"))))  
 holder.ivGreenMargin.setVisibility(View.*VISIBLE*);  
 else {  
 if (!gameActivity.game.getTurn().getWaitForFunction().getTypeOfAction().equals("use"))  
 holder.ivGreenMargin.setVisibility(View.*INVISIBLE*);  
 }  
 }  
  
 */\*\*  
 \** ***@return*** *An Integer of the size of the ArrayList of objects  
 \*/* @Override  
 public int getItemCount() {  
 return this.objects.size();  
 }  
}

TrashAdapter

*/\*\*  
 \* TrashAdapter is the adapter of the RecyclerView of trash of the game.  
 \* The adapter updates the cards shown according to the trash of the game.  
 \* \*/*package com.example.dominion\_game.activities;  
  
import android.util.Pair;  
import android.view.LayoutInflater;  
import android.view.View;  
import android.view.ViewGroup;  
import android.widget.ImageView;  
import android.widget.TextView;  
  
import androidx.recyclerview.widget.RecyclerView;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.classes.Help;  
  
import java.util.ArrayList;  
  
public class TrashAdapter extends RecyclerView.Adapter<TrashAdapter.MyViewHolder> {  
  
 private ArrayList<Pair<String, Integer>> objects;  
 private GameActivity gameActivity;  
  
 */\*\*  
 \* The constructor  
 \** ***@param*** *objects An ArrayList of Pairs of the card name with the count of this card in trash.  
 \** ***@param*** *gameActivity A reference to GameActivity  
 \*/* public TrashAdapter(ArrayList<Pair<String, Integer>> objects, GameActivity gameActivity) {  
 this.objects = objects;  
 this.gameActivity = gameActivity;  
 }  
  
 */\*\*  
 \* A view holder for every item in RecyclerView  
 \*/* public class MyViewHolder extends RecyclerView.ViewHolder implements View.OnLongClickListener {  
 ImageView ivAction;  
 TextView countAction;  
  
 */\*\*  
 \* The constructor  
 \** ***@param*** *view A View of one item in RecyclerView  
 \*/* public MyViewHolder(View view) {  
 super(view);  
 view.setOnLongClickListener(this);  
 ivAction = view.findViewById(R.id.*ivAction*);  
 countAction = view.findViewById(R.id.*countAction*);  
 }  
  
 */\*\*  
 \* A function that handles long press on an item in the RecyclerView  
 \* The function shows a dialog of the card pressed in big.  
 \** ***@param*** *view A View of an item in RecyclerView that was pressed long  
 \** ***@return*** *A Boolean which is always true  
 \*/* @Override  
 public boolean onLongClick(View view) {  
 ImageView iv = gameActivity.cardDialog.findViewById(R.id.*ivAction*);  
 ImageView resourceIv = view.findViewById(R.id.*ivAction*);  
 iv.setImageResource(Help.*nameToCard*(resourceIv.getContentDescription().toString()).getImageSource());  
 gameActivity.cardDialog.show();  
 return true;  
 }  
 }  
  
 */\*\*  
 \* A function that creates the layout for every item in RecyclerView.  
 \** ***@param*** *parent  
 \** ***@param*** *viewType  
 \** ***@return*** *A view for each item in RecyclerView  
 \*/* @Override  
 public TrashAdapter.MyViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {  
 View itemView = LayoutInflater.*from*(parent.getContext())  
 .inflate(R.layout.*card\_trash*, parent, false);  
 itemView.setPadding(20, 20, 20, 20);  
  
 return new MyViewHolder(itemView);  
 }  
  
 */\*\*  
 \* // A function that replaces the contents of every item in RecyclerView.  
 \** ***@param*** *holder A view for each item in RecyclerView  
 \** ***@param*** *position An Integer of the position of the view in the ArrayList  
 \*/* @Override  
 public void onBindViewHolder(TrashAdapter.MyViewHolder holder, int position) {  
 holder.ivAction.setImageResource(Help.*nameToCard*(this.objects.get(position).first).getShortImageSource());  
 holder.countAction.setText(String.*valueOf*(this.objects.get(position).second));  
 holder.ivAction.setContentDescription(this.objects.get(position).first);  
 }  
  
 */\*\*  
 \** ***@return*** *An Integer of the size of the ArrayList  
 \*/* @Override  
 public int getItemCount() {  
 return this.objects.size();  
 }  
}

LogAdapter

*/\*\*  
 \* LogAdapter is the adapter of the ListView of the log of the game.  
 \* The adapter updates the log shown according to the actions of the players.  
 \*/*package com.example.dominion\_game.activities;  
  
import android.app.Activity;  
import android.content.Context;  
import android.graphics.Typeface;  
import android.view.View;  
import android.view.ViewGroup;  
import android.widget.ArrayAdapter;  
import android.widget.RelativeLayout;  
import android.widget.TextView;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.classes.GameManager;  
import com.example.dominion\_game.classes.LogLine;  
  
import java.util.ArrayList;  
  
  
public class LogAdapter extends ArrayAdapter<LogLine> {  
  
 Context context;  
 ArrayList<LogLine> data;  
 GameManager gameManager;  
  
 */\*\*  
 \* The constructor  
 \** ***@param*** *context  
 \** ***@param*** *resource  
 \** ***@param*** *textViewResourceId  
 \** ***@param*** *data An ArrayList of LogLine with all lines in log  
 \** ***@param*** *gameManager A reference to gameManager  
 \*/* public LogAdapter(Context context, int resource, int textViewResourceId, ArrayList<LogLine> data, GameManager gameManager) {  
 super(context, resource, textViewResourceId, data);  
  
 this.context = context;  
 this.data = data;  
 this.gameManager = gameManager;  
 }  
  
 */\*\*  
 \* A function that creates the layout of every item in ListView  
 \* and updates the data inside it according to the Log Line attributes.  
 \** ***@param*** *position An Integer of the position of the view in the ArrayList  
 \** ***@param*** *convertView  
 \** ***@param*** *parent A ViewGroup of the views in ListView according to the size of data  
 \** ***@return*** *A View which is the view of the item in ListView according to the Log Line attributes  
 \*/* @Override  
 public View getView(int position, View convertView, ViewGroup parent) {  
 View view = ((Activity) context).getLayoutInflater().inflate(R.layout.*log\_layout*, parent, false);  
  
 TextView tvLine = view.findViewById(R.id.*tvLine*);  
 TextView tvPlayer = view.findViewById(R.id.*tvPlayer*);  
 RelativeLayout rl = view.findViewById(R.id.*rl*);  
 LogLine logLine = data.get(position);  
  
 tvLine.setTextSize(10);  
 tvLine.setText(logLine.getText());  
 tvLine.setTextColor(context.getResources().getColor(context.getResources().getIdentifier(logLine.getColor(), "color", context.getPackageName())));  
  
 if (logLine.getPlayerId().equals(""))  
 tvPlayer.setVisibility(View.*GONE*);  
 else {  
 tvPlayer.setVisibility(View.*VISIBLE*);  
 tvPlayer.setTextSize(10);  
 tvPlayer.setText(String.*valueOf*(logLine.getPlayerId().charAt(0)));  
 if (logLine.getPlayerId().equals(this.gameManager.getGameManagerBeforeStart().getMyId()))  
 tvPlayer.setTextColor(context.getResources().getColor(R.color.*red*));  
 else  
 tvPlayer.setTextColor(context.getResources().getColor(R.color.*green*));  
 }  
  
 if (logLine.isBold() && logLine.isItalic())  
 tvLine.setTypeface(null, Typeface.*BOLD\_ITALIC*);  
  
 else if (logLine.isBold())  
 tvLine.setTypeface(null, Typeface.*BOLD*);  
  
 else if (logLine.isItalic())  
 tvLine.setTypeface(null, Typeface.*ITALIC*);  
  
 rl.setPadding(logLine.getTabs()\*40, 1, 1, 1);  
 /\*  
 RelativeLayout.LayoutParams params = (RelativeLayout.LayoutParams)tvPlayer.getLayoutParams();  
 // params.setMargins(logLine.getTabs()\*20, params.topMargin, params.rightMargin, params.bottomMargin);  
 tvPlayer.setLayoutParams(params);  
 \*/  
  
 return view;  
 }  
  
 */\*\*  
 \* A function that disables the option to click on any item in the ListView.  
 \** ***@param*** *position An Integer of the position of the view in the ArrayList  
 \** ***@return*** *A Boolean which is always false  
 \*/* @Override  
 public boolean isEnabled(int position) {  
 return false;  
 }  
  
}

ActionCardPlayingAdapter

*/\*\*  
 \* ActionCardPlayingAdapter is the adapter of the RecyclerView in which some  
 \* cards are appeared as a dialog when some special cards are played.  
 \*/*package com.example.dominion\_game.activities;  
  
import android.util.Pair;  
import android.view.LayoutInflater;  
import android.view.View;  
import android.view.ViewGroup;  
import android.widget.ImageView;  
import android.widget.TextView;  
  
import androidx.constraintlayout.widget.ConstraintLayout;  
import androidx.recyclerview.widget.RecyclerView;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.classes.Help;  
  
import java.util.ArrayList;  
  
public class ActionCardPlayingAdapter extends RecyclerView.Adapter<ActionCardPlayingAdapter.MyViewHolder> {  
  
 private ArrayList<Pair<String, Boolean>> objects;  
 private GameActivity gameActivity;  
  
 */\*\*  
 \* The constructor  
 \** ***@param*** *objects An ArrayList of Pairs of the card name with a boolean which is  
 \* true if the card is selected and false if not.  
 \** ***@param*** *gameActivity A reference to GameActivity  
 \*/* public ActionCardPlayingAdapter(ArrayList<Pair<String, Boolean>> objects, GameActivity gameActivity) {  
 this.objects = objects;  
 this.gameActivity = gameActivity;  
 }  
  
 */\*\*  
 \* A view holder for every item in RecyclerView  
 \*/* public class MyViewHolder extends RecyclerView.ViewHolder implements View.OnLongClickListener, View.OnClickListener {  
 ImageView ivAction, ivGreenMargin, ivRedMargin, ivYellowMargin, ivGreenX, ivRedX, ivYellowX;  
 TextView tvAction;  
 ConstraintLayout constraintLayout;  
  
 */\*\*  
 \* The constructor  
 \** ***@param*** *view A View of one item in RecyclerView  
 \*/* public MyViewHolder(View view) {  
 super(view);  
 ivAction = view.findViewById(R.id.*ivAction*);  
 ivGreenMargin = view.findViewById(R.id.*ivGreenMargin*);  
 ivRedMargin = view.findViewById(R.id.*ivRedMargin*);  
 ivYellowMargin = view.findViewById(R.id.*ivYellowMargin*);  
 ivGreenX = view.findViewById(R.id.*ivGreenX*);  
 ivRedX = view.findViewById(R.id.*ivRedX*);  
 ivYellowX = view.findViewById(R.id.*ivYellowX*);  
 tvAction = view.findViewById(R.id.*tvAction*);  
 constraintLayout = view.findViewById(R.id.*constraintLayout*);  
 view.setOnLongClickListener(this);  
 view.setOnClickListener(this);  
 }  
  
 */\*\*  
 \* A function that handles long press on an item in the RecyclerView  
 \* The function shows a dialog of the card pressed in big.  
 \** ***@param*** *view A View of an item in RecyclerView that was pressed long  
 \** ***@return*** *A Boolean which is always true  
 \*/* @Override  
 public boolean onLongClick(View view) {  
 ImageView iv = gameActivity.cardDialog.findViewById(R.id.*ivAction*);  
 ImageView resourceIv = view.findViewById(R.id.*ivAction*);  
 iv.setImageResource(Help.*nameToCard*(resourceIv.getContentDescription().toString()).getImageSource());  
 gameActivity.cardDialog.show();  
 return true;  
 }  
  
 */\*\*  
 \* A function that handles a press on an item in the RecyclerView  
 \* The function checks if the card can be selected.  
 \** ***@param*** *view A View of an item in RecyclerView that was pressed  
 \*/* @Override  
 public void onClick(View view) {  
 if (!gameActivity.game.getTurn().getWaitForFunction().isWaitingForActionCardsDialog())  
 return;  
  
 ImageView resourceIv = view.findViewById(R.id.*ivAction*);  
 if (!Help.*nameToCard*(gameActivity.game.getTurn().getWaitForFunction().getCardName())  
 .isCardToUse(resourceIv.getContentDescription().toString(), gameActivity.game, gameActivity)  
 || Help.*sizeOfHash*(gameActivity.game.getTurn().getWaitForFunction().getCardsForActionCardPlay()) == gameActivity.game.getTurn().getWaitForFunction().getMaxAmount())  
 return;  
  
 if (gameActivity.game.getTurn().getWaitForFunction().isHandleClickOnCard()  
 && !Help.*nameToCard*(gameActivity.game.getTurn().getWaitForFunction().getCardName()).isMarkCardSelectedFromHandWhenHandle()) {  
 gameActivity.game.getTurn().getWaitForFunction().updateCardsForDialogByPosition(getAdapterPosition(), true);  
 Help.*nameToCard*(gameActivity.game.getTurn().getWaitForFunction().getCardName())  
 .handleClickOnHandOrDialog(resourceIv.getContentDescription().toString(), gameActivity.game, gameActivity);  
 return;  
 }  
  
 if (gameActivity.game.getTurn().getWaitForFunction().getTypeOfAction().equals("trash"))  
 view.findViewById(R.id.*ivRedX*).setVisibility(View.*VISIBLE*);  
  
 if (!gameActivity.game.getTurn().getWaitForFunction().getTypeOfAction().equals("trash")  
 && !gameActivity.game.getTurn().getWaitForFunction().getTypeOfAction().equals("order"))  
 view.findViewById(R.id.*ivYellowX*).setVisibility(View.*VISIBLE*);  
  
 if (gameActivity.game.getTurn().getWaitForFunction().getTypeOfAction().equals("order")) {  
 if (gameActivity.game.getTurn().getWaitForFunction().getCardsForDialog().get(getAdapterPosition()).second) {  
 view.findViewById(R.id.*ivYellowMargin*).setVisibility(View.*INVISIBLE*);  
 gameActivity.game.getTurn().getWaitForFunction().updateCardsForDialogByPosition(getAdapterPosition(), false);  
 }  
 else if (Help.*sizeOfHash*(gameActivity.game.getTurn().getWaitForFunction().getCardsForActionCardPlay()) == 1) {  
 gameActivity.game.getTurn().getWaitForFunction().updateCardsForDialogByPosition(getAdapterPosition(), true);  
 gameActivity.game.getTurn().getWaitForFunction().order();  
 gameActivity.getActionCardsPlayingAdapter().notifyDataSetChanged();  
 }  
 else {  
 gameActivity.game.getTurn().getWaitForFunction().updateCardsForDialogByPosition(getAdapterPosition(), true);  
 view.findViewById(R.id.*ivYellowMargin*).setVisibility(View.*VISIBLE*);  
 }  
 }  
 else  
 gameActivity.game.getTurn().getWaitForFunction().updateCardsForDialogByPosition(getAdapterPosition(), true);  
  
 if (gameActivity.game.getTurn().getWaitForFunction().isHandleClickOnCard()  
 && Help.*nameToCard*(gameActivity.game.getTurn().getWaitForFunction().getCardName()).isMarkCardSelectedFromHandWhenHandle()) {  
 Help.*nameToCard*(gameActivity.game.getTurn().getWaitForFunction().getCardName())  
 .handleClickOnHandOrDialog(resourceIv.getContentDescription().toString(), gameActivity.game, gameActivity);  
 return;  
 }  
 if (gameActivity.game.getTurn().getWaitForFunction().isHasUndoAndConfirm())  
 gameActivity.updateActionButtons();  
 }  
 }  
  
 */\*\*  
 \* A function that creates the layout for every item in RecyclerView.  
 \** ***@param*** *parent  
 \** ***@param*** *viewType  
 \** ***@return*** *A view for each item in RecyclerView  
 \*/* @Override  
 public ActionCardPlayingAdapter.MyViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {  
 View itemView = LayoutInflater.*from*(parent.getContext())  
 .inflate(R.layout.*card\_for\_action\_cards\_playing*, parent, false);  
  
 return new MyViewHolder(itemView);  
 }  
  
 */\*\*  
 \* // A function that replaces the contents of every item in RecyclerView.  
 \** ***@param*** *holder A view for each item in RecyclerView  
 \** ***@param*** *position An Integer of the position of the view in the ArrayList  
 \*/* @Override  
 public void onBindViewHolder(ActionCardPlayingAdapter.MyViewHolder holder, int position) {  
 holder.ivAction.setImageResource(Help.*nameToCard*(this.objects.get(position).first).getImageSource());  
 holder.ivAction.setContentDescription(this.objects.get(position).first);  
 holder.constraintLayout.setBackgroundColor(gameActivity.getResources().getColor(R.color.*white*));  
 holder.constraintLayout.getBackground().setAlpha(150);  
  
 if (!gameActivity.game.getTurn().getWaitForFunction().isWaitingForActionCardsDialog()  
 || !Help.*nameToCard*(gameActivity.game.getTurn().getWaitForFunction().getCardName())  
 .isCardToUse(this.objects.get(position).first, gameActivity.game, gameActivity)  
 || gameActivity.game.getTurn().getWaitForFunction().getTypeOfAction().equals("")) {  
 holder.ivRedMargin.setVisibility(View.*INVISIBLE*);  
 holder.ivRedX.setVisibility(View.*INVISIBLE*);  
 holder.ivYellowMargin.setVisibility(View.*INVISIBLE*);  
 holder.ivYellowX.setVisibility(View.*INVISIBLE*);  
 holder.ivGreenMargin.setVisibility(View.*INVISIBLE*);  
 return;  
 }  
 if (gameActivity.game.getTurn().getWaitForFunction().getTypeOfAction().equals("trash")) {  
 holder.ivRedMargin.setVisibility(View.*VISIBLE*);  
 if (this.objects.get(position).second)  
 holder.ivRedX.setVisibility(View.*VISIBLE*);  
 else  
 holder.ivRedX.setVisibility(View.*INVISIBLE*);  
 }  
 else {  
 holder.ivRedMargin.setVisibility(View.*INVISIBLE*);  
 holder.ivRedX.setVisibility(View.*INVISIBLE*);  
 }  
  
 if (gameActivity.game.getTurn().getWaitForFunction().getTypeOfAction().equals("use"))  
 holder.ivGreenMargin.setVisibility(View.*VISIBLE*);  
 else  
 holder.ivGreenMargin.setVisibility(View.*INVISIBLE*);  
  
 if (gameActivity.game.getTurn().getWaitForFunction().getTypeOfAction().equals("order")) {  
 if (position == 0)  
 holder.tvAction.setText("Bottom");  
 else if (position == this.getItemCount() - 1)  
 holder.tvAction.setText("Top");  
 else  
 holder.tvAction.setText(String.*valueOf*(this.getItemCount() - position));  
 holder.tvAction.setVisibility(View.*VISIBLE*);  
 // holder.tvAction.setTypeface(null, Typeface.BOLD);  
 }  
 else  
 holder.tvAction.setVisibility(View.*GONE*);  
  
 if (!gameActivity.game.getTurn().getWaitForFunction().getTypeOfAction().equals("trash")  
 && !gameActivity.game.getTurn().getWaitForFunction().getTypeOfAction().equals("use")) {  
 if (!gameActivity.game.getTurn().getWaitForFunction().getTypeOfAction().equals("order")  
 || this.objects.get(position).second)  
 holder.ivYellowMargin.setVisibility(View.*VISIBLE*);  
 else  
 holder.ivYellowMargin.setVisibility(View.*INVISIBLE*);  
  
 if (!gameActivity.game.getTurn().getWaitForFunction().getTypeOfAction().equals("order"))  
 if (this.objects.get(position).second)  
 holder.ivYellowX.setVisibility(View.*VISIBLE*);  
 else  
 holder.ivYellowX.setVisibility(View.*INVISIBLE*);  
 }  
 else {  
 holder.ivYellowMargin.setVisibility(View.*INVISIBLE*);  
 holder.ivYellowX.setVisibility(View.*INVISIBLE*);  
 }  
 }  
  
 */\*\*  
 \** ***@return*** *An Integer of the size of the ArrayList of objects  
 \*/* @Override  
 public int getItemCount() {  
 return this.objects.size();  
 }  
}

**Classes:**

GameManager

*/\*\*  
 \* GameManager is a class that keeps all the data about the game for the player.  
 \*/*package com.example.dominion\_game.classes;  
  
import android.util.Pair;  
  
import com.example.dominion\_game.activities.GameActivity;  
import com.google.gson.Gson;  
import com.google.gson.reflect.TypeToken;  
  
import org.json.JSONException;  
import org.json.JSONObject;  
  
import java.lang.reflect.Type;  
import java.util.ArrayList;  
import java.util.HashMap;  
import java.util.Random;  
import java.util.Stack;  
  
public class GameManager {  
 private static final int *numberOfCards* = 5;  
 private static final int *victoryAmount* = 8;  
 private static final int *curseAmount* = 10;  
 private static final int *actionAmount* = 10;  
 private static final int *copperAmount* = 46;  
 private static final int *silverAmount* = 40;  
 private static final int *goldAmount* = 30;  
 private static final int *decksToEndGame* = 3;  
  
 private GameManagerBeforeStart gameManagerBeforeStart;  
 private GameActivity gameActivity;  
 private String[] actionCards;  
 private Player player;  
 private EnemyData enemyData;  
 private HashMap <String, Integer> board;  
 private HashMap <String, Integer> trash;  
 private ArrayList<Pair<String, Integer>> arrayTrash;  
  
 private Turn turn;  
 private ArrayList<LogLine> log;  
 private boolean isStarted;  
 private boolean isGameEnded;  
 private boolean resigned;  
  
 private boolean playsAttack;  
 private boolean doneAttack;  
  
 private Stack <Integer> times; // times playing an action card (for card throne room)  
  
 */\*\*  
 \* The constructor with default values for all attributes but gameManagerBeforeStart  
 \* and gameActivity that were created before  
 \*/* public GameManager(GameManagerBeforeStart gameManagerBeforeStart, GameActivity gameActivity) {  
 this.gameActivity = gameActivity;  
 this.player = new Player();  
 this.enemyData = new EnemyData();  
 this.gameManagerBeforeStart = gameManagerBeforeStart;  
  
 this.board = new HashMap<>();  
 this.board.put("Estate", *victoryAmount*);  
 this.board.put("Duchy", *victoryAmount*);  
 this.board.put("Province", *victoryAmount*);  
 this.board.put("Curse", *curseAmount*);  
 this.board.put("Copper", *copperAmount*);  
 this.board.put("Silver", *silverAmount*);  
 this.board.put("Gold", *goldAmount*);  
 this.trash = new HashMap<>();  
 this.arrayTrash = new ArrayList<>();  
 this.log = new ArrayList<>();  
 this.isStarted = false;  
 this.isGameEnded = false;  
 this.resigned = false;  
 this.playsAttack = false;  
 this.doneAttack = false;  
 times = new Stack<>();  
 times.push(1);  
 }  
  
 */\*\*  
 \* A function that sends the creator to a function for uploading data  
 \* and the non creator for a function for getting data.  
 \* This function also generates for the creator  
 \* the action cards and chooses randomly who starts.  
 \*/* public void startUploadAndGet() {  
 if (gameManagerBeforeStart.isCreator()) {  
 this.actionCards = Help.*getRandomCards*(new String[]{"Base"}, 10, new String[]{});  
  
 Random randomStart = new Random();  
 this.turn = new Turn(randomStart.nextInt(2) == 0 ? gameManagerBeforeStart.getIdP1() : gameManagerBeforeStart.getIdP2());  
  
 this.gameActivity.beforeStart();  
 GameRequests.*uploadDataInGame*(true, this);  
 }  
 else {  
 this.actionCards = new String[10];  
 this.turn = new Turn();  
 GameRequests.*getDataInGame*(true, this);  
 }  
 }  
  
 */\*\*  
 \* A function that generates before start by put in hand the start cards  
 \* and put in board the starting cards.  
 \*/* public void beforeGame() {  
 for (String nameCard : actionCards) {  
 if (Help.*nameToCard*(nameCard).getType().equals("victory"))  
 this.board.put(nameCard, *victoryAmount*);  
 else  
 this.board.put(nameCard, *actionAmount*);  
 }  
  
 this.player.getHand().put("Copper", 7);  
 this.player.getHand().put("Estate", 3);  
  
 if (this.gameManagerBeforeStart.isCreator()) {  
 this.log.add(new LogLine("Starts with 7 Coppers", this.gameManagerBeforeStart.getMyId(), false, false, 0, "start"));  
 this.log.add(new LogLine("Starts with 3 Estates", this.gameManagerBeforeStart.getMyId(), false, false, 0, "start"));  
 this.log.add(new LogLine("Starts with 7 Coppers", this.gameManagerBeforeStart.getEnemyId(), false, false, 0, "start"));  
 this.log.add(new LogLine("Starts with 3 Estates", this.gameManagerBeforeStart.getEnemyId(), false, false, 0, "start"));  
 }  
 }  
  
 */\*\*  
 \* A function that generates start after both players were ready.  
 \*/* public void startGame() {  
 this.cleanUpPhase();  
  
 this.player.discardToDeck(this, 0);  
 this.player.takeCardsToHand(*numberOfCards*, this.gameActivity, this, 0);  
  
 if (this.turn.isMyTurn(this.gameManagerBeforeStart)) {  
 this.log.add(new LogLine("Shuffles his deck", this.gameManagerBeforeStart.getEnemyId(), false, true, "pink", 0, "start"));  
 this.log.add(new LogLine("Draws " + *numberOfCards* + " card" + (*numberOfCards* == 1 ? "" : "s"), this.gameManagerBeforeStart.getEnemyId(), false, false, 0, "start"));  
 }  
 else {  
 this.deleteLastLineFromLog();  
 this.deleteLastLineFromLog();  
 }  
 }  
  
 */\*\*  
 \* A function that adds to log the turn number.  
 \* This function is called at the start of every turn.  
 \*/* public void addTurnNumberToLog() {  
 this.log.add(new LogLine("", "", false, false, 0, "change turn"));  
 this.log.add(new LogLine("Turn " + this.turn.getTurnNumber() + " - " + this.turn.getTurnId(), "", true, false, 0, "change turn"));  
 }  
  
 */\*\*  
 \* A function that uses a card n times if the card can be used.  
 \** ***@param*** *cardName A String which is the name of the card which should be used  
 \** ***@param*** *n An Integer which is the number of times that the card should be played  
 \** ***@param*** *forceUse A Boolean which is true if the use of the card was forced or not  
 \** ***@param*** *autoPlay A Boolean which is true if useCard was played with autoPlay and false if not  
 \*/* public void useCard(String cardName, int n, boolean forceUse, boolean autoPlay) {  
 // n > 1 when throne room or autoPlay  
 if (!this.turn.isMyTurn(this.gameManagerBeforeStart) || Help.*nameToCard*(cardName).getType().equals("victory") ||  
 (Help.*nameToCard*(cardName).getType().equals("action") && this.getTurn().getActions() == 0 && n == 1))  
 return;  
  
 for (int i = 0; i < n; i++)  
 this.playCardOrAddToWaitList(cardName, n, i, forceUse, autoPlay);  
 }  
  
 */\*\*  
 \* A function that plays a card or adding it to the wait list of cards.  
 \** ***@param*** *cardName A String which is the name of the card which should be used  
 \** ***@param*** *n An Integer which is the number of times that the card should be played  
 \** ***@param*** *i An Integer which is the place of the card in the wait queue  
 \** ***@param*** *forceUse A Boolean which is true if the use of the card was forced or not  
 \** ***@param*** *autoPlay A Boolean which is true if useCard was played with autoPlay and false if not  
 \*/* public void playCardOrAddToWaitList(String cardName, int n, int i, boolean forceUse, boolean autoPlay) {  
 if (this.turn.getLastCardForWaitForPlay().equals("")) {  
 this.turn.setLastCardForWaitForPlay(cardName);  
 this.playCard(cardName, n, i, forceUse, autoPlay);  
 }  
 else  
 this.turn.addWaitForPlay(cardName, n, i, forceUse, autoPlay);  
 }  
 */\*\*  
 \* A function that plays a card and adds to log.  
 \** ***@param*** *cardName A String which is the name of the card which should be used  
 \** ***@param*** *n An Integer which is the number of times that the card should be played  
 \** ***@param*** *i An Integer which is the place of the card in the wait queue  
 \** ***@param*** *forceUse A Boolean which is true if the use of the card was forced or not  
 \** ***@param*** *autoPlay A Boolean which is true if useCard was played with autoPlay and false if not  
 \*/* public void playCard(String cardName, int n, int i, boolean forceUse, boolean autoPlay) {  
 if (this.player.getHand().containsKey(cardName) && (autoPlay || n == 1 || i == 0)) {  
 this.player.removeFromHand(cardName);  
 this.player.updateArrayHand();  
 gameActivity.getHandAdapter().notifyDataSetChanged();  
 }  
  
 if (Help.*nameToCard*(cardName).getType().equals("action") && !forceUse)  
 this.log.add(new LogLine("Plays a " + Help.*nameToCard*(cardName).getNameToDisplay(), this.turn.getTurnId(), false, false, this.getTabs() - 1, "use action"));  
  
 else if (forceUse || (!autoPlay && n > 1 && i == 0)) // throne room first or vassal  
 this.log.add(new LogLine("Plays a " + Help.*nameToCard*(cardName).getNameToDisplay(), this.turn.getTurnId(), false, false, this.getLastLineFromLog().getTabs() + 1, "use action"));  
  
 else if ((!autoPlay && n > 1)) // not first played in throne room  
 this.log.add(new LogLine("Plays a " + Help.*nameToCard*(cardName).getNameToDisplay(), this.turn.getTurnId(), false, false, this.getLastLineFromLog().getTabs() - 1, "use action"));  
  
 if (Help.*nameToCard*(cardName).getType().equals("action") && n == 1 && !forceUse)  
 this.getTurn().useAction();  
  
 if (Help.*nameToCard*(cardName).getType().equals("action") && i == 0) {  
 this.turn.addActionCard(cardName);  
 }  
 else if (Help.*nameToCard*(cardName).getType().equals("treasure")) {  
 this.turn.addTreasureCard(cardName);  
 this.addHashToLog(this.turn.getTreasureCardsPlayed(), "use treasure", "Plays");  
 }  
  
 Help.*nameToCard*(cardName).play(this, gameActivity);  
 }  
  
 */\*\*  
 \* A function that calls the function afterPlay  
 \* for every card used before in this turn.  
 \** ***@param*** *cardNameUsed A String with the name of card used  
 \*/* public void useAfterPlay(String cardNameUsed, boolean addWaitForEnemy) {  
 this.turn.setLastCardForWaitForPlay("");  
 for (String cardName : this.getTurn().getActionCardsPlayed())  
 Help.*nameToCard*(cardName).afterPlay(this, cardNameUsed);  
  
 for (String cardName : this.turn.getTreasureCardsPlayed().keySet())  
 for (int i = 0; i < this.turn.getTreasureCardsPlayed().get(cardName); i++)  
 Help.*nameToCard*(cardName).afterPlay(this, cardNameUsed);  
  
 if (addWaitForEnemy)  
 this.turn.addWaitForEnemy(cardNameUsed);  
 else  
 this.endCard();  
 }  
  
 */\*\*  
 \* A function that is called after the card ended and plays future cards if has.  
 \*/* public void endCard() {  
 if (!this.turn.getCardsForWaitForPlay().isEmpty()) {  
 PlayCardArguments playCardArguments = this.turn.getCardsForWaitForPlay().remove(0);  
 this.turn.setLastCardForWaitForPlay(playCardArguments.getCardName());  
 this.playCard(playCardArguments.getCardName(), playCardArguments.getN(), playCardArguments.getI(), playCardArguments.isForceUse(), playCardArguments.isAutoPlay());  
 }  
 else {  
 if (this.times.peek() != 1)  
 this.times.pop();  
 }  
 gameActivity.turnUI();  
 gameActivity.updateCards(true);  
 }  
  
 */\*\*  
 \* A function that buys a card times if the card can be bought.  
 \** ***@param*** *cardName A String which is the name of the card which should be used  
 \*/* public void buyCard(String cardName) {  
 if ((this.board.get(cardName) == 0 || this.turn.getBuys() == 0 || this.turn.getTreasure() < Help.*nameToCard*(cardName).getPrice()))  
 return;  
  
 this.turn.useBuy();  
 this.turn.useTreasure(Help.*nameToCard*(cardName).getPrice());  
 // remove from board cards  
 this.board.put(cardName, this.board.get(cardName) - 1);  
 this.turn.addCardBought(cardName);  
 this.addHashToLog(this.turn.getCardsBought(), "buy and gain", "Buys and Gains");  
 }  
  
 */\*\*  
 \* A function that removes a card from board and returns it.  
 \** ***@param*** *cardName A String with the card name  
 \** ***@return*** *A String with the card name  
 \*/* public String getCard(String cardName) {  
 if (this.board.get(cardName) == 0)  
 return "";  
 // remove from board cards  
 this.board.put(cardName, this.board.get(cardName) - 1);  
 return cardName;  
 }  
  
 */\*\*  
 \* A function that handles cleanUp and add all cards that  
 \* were used to the discard and clear the hand.  
 \*/* public void cleanUpPhase() {  
 this.addToDiscardByType("action");  
 this.addToDiscardByType("treasure");  
 this.addToDiscardByType("victory");  
 this.player.getDiscard().addAll(this.turn.getActionCardsPlayed());  
 this.player.getDiscard().addAll(Help.*hashToArray*(this.turn.getTreasureCardsPlayed()));  
 this.player.getDiscard().addAll(Help.*hashToArray*(this.turn.getCardsBought()));  
  
 this.player.getHand().clear();  
 this.player.updateArrayHand();  
 gameActivity.getHandAdapter().notifyDataSetChanged();  
 times.clear();  
 times.push(1);  
 }  
  
 */\*\*  
 \* A function that adds the cards of this type from hand to discard.  
 \** ***@param*** *type A String with the type of the card (action, treasure or victory)  
 \*/* public void addToDiscardByType(String type) {  
 for (String cardName : this.player.getHand().keySet())  
 if (Help.*nameToCard*(cardName).getType().equals(type))  
 for (int i = 0; i < this.player.getHand().get(cardName); i++)  
 this.player.getDiscard().add(cardName);  
 }  
  
 */\*\*  
 \* A function that autoPlay all treasures in hand by using each treasure card.  
 \*/* public void autoPlayTreasures() {  
 String[] arrayKeys = this.player.getHand().keySet().toArray  
 (new String[this.player.getHand().keySet().size()]);  
 for (int i = 0; i < arrayKeys.length; i++) {  
 if (Help.*nameToCard*(arrayKeys[i]).getType().equals("treasure"))  
 useCard(arrayKeys[i], this.player.getHand().get(arrayKeys[i]), false, true);  
 }  
  
 this.addHashToLog(this.turn.getTreasureCardsPlayed(), "use treasure", "Plays");  
 }  
  
 */\*\*  
 \* A function that adds to log the use or buy of HashMap with cards.  
 \** ***@param*** *hm A HashMap of card names with the amount of it  
 \** ***@param*** *action A String with the action in the log  
 \*/* public void addHashToLog(HashMap<String, Integer> hm, String action, String textBefore) {  
 if ((action.equals("buy and gain") && this.getLastLineFromLog().getType().equals("buy and gain"))  
 || (action.equals("use treasure") && this.getLastLineFromLog().getTabs() == 0 && this.getLastLineFromLog().getType().equals("use treasure")))  
 this.deleteLastLineFromLog(); // deletes the last line because it will add an updated line  
  
 String[] arrayKeys = hm.keySet().toArray  
 (new String[hm.keySet().size()]);  
 for (int i = 0; i < arrayKeys.length; i++) {  
 if (i == 0)  
 this.log.add(new LogLine(textBefore + " " + (hm.get(arrayKeys[0]) == 1  
 ? "a " : hm.get(arrayKeys[0]) + " ") + Help.*nameToCard*(arrayKeys[0]).getNameToDisplay() + (hm.get(arrayKeys[i]) == 1 ? "" : "s"),  
 this.turn.getTurnId(), false, false, 0, action));  
  
 else if (i < arrayKeys.length - 1)  
 this.getLastLineFromLog().setText(this.getLastLineFromLog().getText() + ", " + (hm.get(arrayKeys[i]) == 1  
 ? "a " : hm.get(arrayKeys[i]) + " ") + Help.*nameToCard*(arrayKeys[i]).getNameToDisplay() + (hm.get(arrayKeys[i]) == 1 ? "" : "s"));  
  
 else  
 this.getLastLineFromLog().setText(this.getLastLineFromLog().getText() + " and " + (hm.get(arrayKeys[i]) == 1  
 ? "a " : hm.get(arrayKeys[i]) + " ") + Help.*nameToCard*(arrayKeys[i]).getNameToDisplay() + (hm.get(arrayKeys[i]) == 1 ? "" : "s"));  
 }  
 }  
  
 public String[] getActionCards() {  
 return this.actionCards;  
 }  
  
 public void setActionCards(String[] actionCards) {  
 this.actionCards = actionCards;  
 }  
  
 public HashMap<String, Integer> getBoard() {  
 return this.board;  
 }  
  
 public void setBoard(HashMap<String, Integer> board) {  
 this.board = board;  
 }  
  
 public HashMap<String, Integer> getTrash() {  
 return this.trash;  
 }  
  
 public void setTrash(HashMap<String, Integer> trash) {  
 this.trash = trash;  
 }  
  
 public ArrayList<Pair<String, Integer>> getArrayTrash() {  
 return this.arrayTrash;  
 }  
  
 */\*\*  
 \* A function that adds a card to trash  
 \** ***@param*** *cardName A String with the name of card to be added to trash  
 \*/* public void addToTrash(String cardName, int n) {  
 if (this.trash.containsKey(cardName))  
 this.trash.put(cardName, this.trash.get(cardName) + n);  
 else  
 this.trash.put(cardName, n);  
 this.updateArrayTrash();  
 gameActivity.getTrashAdapter().notifyDataSetChanged();  
 }  
  
 */\*\*  
 \* A function that updates the arrayTrash to be same as in trash.  
 \*/* public void updateArrayTrash() {  
 this.arrayTrash.clear();  
 for (String cardName : this.trash.keySet())  
 this.arrayTrash.add(new Pair<>(cardName, this.trash.get(cardName)));  
 }  
  
 public Player getPlayer() {  
 return this.player;  
 }  
  
 public void setPlayer(Player player) {  
 this.player = player;  
 }  
  
 */\*\*  
 \* A function that returns whether the game is ended according to the rules.  
 \** ***@return*** *A Boolean which is true if the game is ended according to the rules and false if not  
 \*/* public boolean gameEnded() {  
 if (this.board.get("Province") == 0)  
 return true;  
 int count = 0;  
 for (String cardName : this.board.keySet()) {  
 if (this.board.get(cardName) == 0)  
 count++;  
 if (count >= *decksToEndGame*)  
 return true;  
 }  
 return false;  
 }  
  
 */\*\*  
 \* A function that changes the turn by calling clean up, restarting turn and take new cards.  
 \*/* public void changeTurn() {  
 this.turn.setPhase("cleanUp");  
 this.cleanUpPhase();  
 this.turn.changeTurn(this.gameManagerBeforeStart);  
 this.player.takeCardsToHand(*numberOfCards*, this.gameActivity, this, 0);  
 }  
  
 public Turn getTurn() {  
 return this.turn;  
 }  
  
 public void setTurn(Turn turn) {  
 this.turn = turn;  
 }  
  
 public ArrayList<LogLine> getLog() {  
 return this.log;  
 }  
  
 */\*\*  
 \* A function that deletes the last line in the log.  
 \*/* public void deleteLastLineFromLog() {  
 if (this.log.size() > 0)  
 this.log.remove(this.log.size() - 1);  
 }  
  
 */\*\*  
 \* A function that returns the last line in the log.  
 \** ***@return*** *A LogLine which is the last line in the log  
 \*/* public LogLine getLastLineFromLog() {  
 if (this.log.size() > 0)  
 return this.log.get(this.log.size() - 1);  
 return null;  
 }  
  
 public Stack<Integer> getTimes() {  
 return this.times;  
 }  
  
 public int getTabs() {  
 return this.times.size();  
 }  
  
 public GameActivity getGameActivity() {  
 return this.gameActivity;  
 }  
  
 public void setGameActivity(GameActivity gameActivity) {  
 this.gameActivity = gameActivity;  
 }  
  
 */\*\*  
 \* A function that creates a JSONObject from the data in  
 \* gameManager to upload to server before starting game.  
 \** ***@return*** *A JSONObject of the gameManager attributes  
 \*/* public JSONObject gameManagerToJsonStart() {  
 JSONObject jsonObject = new JSONObject();  
 Gson gson = new Gson();  
 try {  
 jsonObject.put("gameManagerBeforeStart", this.gameManagerBeforeStart.gameManagerBeforeStartToJson());  
 jsonObject.put("board", gson.toJson(this.board));  
 jsonObject.put("turn", this.turn.turnToJson(true));  
 jsonObject.put("trash", gson.toJson(this.trash));  
 jsonObject.put("log", gson.toJson(this.logToJson()));  
 jsonObject.put("actionCards", gson.toJson(this.actionCards));  
 jsonObject.put("isGameEnded", this.isGameEnded);  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
  
 return jsonObject;  
 }  
  
 */\*\*  
 \* A function that creates a JSONObject from the data in  
 \* gameManager to upload to server after starting game.  
 \** ***@return*** *A JSONObject of the gameManager attributes  
 \*/* public JSONObject gameManagerToJsonRealTime() {  
 JSONObject jsonObject = new JSONObject();  
 Gson gson = new Gson();  
 try {  
 jsonObject.put("gameManagerBeforeStart", this.gameManagerBeforeStart.gameManagerBeforeStartToJson());  
 jsonObject.put("isGameEnded", this.isGameEnded);  
 if (this.turn.isWaitingForEnemy() && this.turn.getLastActionCardForWait().equals("") && this.turn.isMyTurn(this.gameManagerBeforeStart))  
 return jsonObject;  
  
 jsonObject.put("board", gson.toJson(this.board));  
 jsonObject.put("trash", gson.toJson(this.trash));  
 jsonObject.put("log", gson.toJson(this.logToJson()));  
 jsonObject.put("turn", this.turn.turnToJson(this.doneAttack || this.turn.isMyTurn(this.gameManagerBeforeStart)));  
  
 if (this.doneAttack) {  
 this.doneAttack = false;  
 this.playsAttack = false;  
 }  
  
 if (!this.turn.getLastActionCardForWait().equals(""))  
 this.turn.removeLastAttack();  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
  
 return jsonObject;  
 }  
  
 */\*\*  
 \* A function that updates the values on the attributes  
 \** ***@param*** *jsonObject A JSONObject that is given from the server  
 \*/* public void jsonToGameManagerStart(JSONObject jsonObject) {  
 Gson gson = new Gson();  
 try {  
 this.gameManagerBeforeStart.jsonToGameManagerBeforeStart(jsonObject, true);  
 Type type = new TypeToken<HashMap<String, Integer>>(){}.getType();  
 this.board = gson.fromJson(jsonObject.getString("board"), type);  
 this.turn.jsonToTurn(jsonObject.getJSONObject("turn"), this);  
 type = new TypeToken<ArrayList<JSONObject>>(){}.getType();  
 this.jsonToLog((ArrayList<JSONObject>)gson.fromJson(jsonObject.getString("log"), type));  
 this.actionCards = gson.fromJson(jsonObject.getString("actionCards"), String[].class);  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* A function that updates the values on the attributes  
 \** ***@param*** *jsonObject A JSONObject that is given from the server  
 \*/* public void jsonToGameManagerRealTime(JSONObject jsonObject) {  
 if (!this.turn.isMyTurn(this.gameManagerBeforeStart) && !jsonObject.has("turn"))  
 return;  
 Gson gson = new Gson();  
 try {  
 Type type = new TypeToken<HashMap<String, Integer>>(){}.getType();  
 this.board = gson.fromJson(jsonObject.getString("board"), type);  
 if (!this.trash.equals(gson.fromJson(jsonObject.getString("trash"), type))) {  
 this.trash = gson.fromJson(jsonObject.getString("trash"), type);  
 this.updateArrayTrash();  
 gameActivity.getTrashAdapter().notifyDataSetChanged();  
 }  
 type = new TypeToken<ArrayList<JSONObject>>(){}.getType();  
 this.jsonToLog((ArrayList<JSONObject>)gson.fromJson(jsonObject.getString("log"), type));  
 this.turn.jsonToTurn(jsonObject.getJSONObject("turn"), this);  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* A function that creates a JSONObject from the relevant data  
 \* for uploading to server about the size of deck and hand and more.  
 \** ***@return*** *A JSONObject relevant attributes for the other player  
 \*/* public JSONObject myDataToJson() {  
 JSONObject myData = new JSONObject();  
 try {  
 if (!this.player.getDiscard().isEmpty())  
 myData.put("lastCardOnDiscard", this.player.getDiscard().get(this.player.getDiscard().size() - 1));  
 else  
 myData.put("lastCardOnDiscard", "");  
 myData.put("deckSize", this.player.getDeck().size());  
 myData.put("handSize", Help.*sizeOfHash*(this.player.getHand()));  
 myData.put("victoryPoints", this.player.getVictoryPoints(this));  
 myData.put("resigned", this.resigned);  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
 JSONObject jsonObject = new JSONObject();  
 try {  
 JSONObject gameManagerBeforeStartJson = this.gameManagerBeforeStart.gameManagerBeforeStartToJson();  
 jsonObject.put("gameManagerBeforeStart", gameManagerBeforeStartJson);  
 jsonObject.put("myData", myData);  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
  
 return jsonObject;  
 }  
  
 */\*\*  
 \* A function that returns an ArrayList of JSONObject with all the log lines.  
 \** ***@return*** *An ArrayList of JSONObject with all the log lines  
 \*/* public ArrayList<JSONObject> logToJson() {  
 ArrayList<JSONObject> logLines = new ArrayList<>();  
 for (LogLine logLine : this.log)  
 logLines.add(logLine.logLineToJson());  
  
 return logLines;  
 }  
  
 */\*\*  
 \* A function that updates new log lines in log.  
 \** ***@param*** *jsonLogLines A JSONObject that is given from the server with the lines of log  
 \*/* public void jsonToLog(ArrayList<JSONObject> jsonLogLines) {  
 ArrayList<LogLine> logLines = new ArrayList<>();  
 for (JSONObject logLine : jsonLogLines)  
 logLines.add(new LogLine(logLine));  
  
 if (logLines.size() > this.log.size())  
 this.log.addAll(logLines.subList(this.log.size(), logLines.size()));  
 }  
  
 public GameManagerBeforeStart getGameManagerBeforeStart() {  
 return this.gameManagerBeforeStart;  
 }  
  
 public EnemyData getEnemyData() {  
 return this.enemyData;  
 }  
  
 public boolean isStarted() {  
 return this.isStarted;  
 }  
  
 public void setStarted(boolean started) {  
 this.isStarted = started;  
 }  
  
 public boolean isGameEnded() {  
 return this.isGameEnded;  
 }  
  
 public void setGameEnded(boolean gameEnded) {  
 this.isGameEnded = gameEnded;  
 }  
  
 public boolean isResigned() {  
 return this.resigned;  
 }  
  
 public void setResigned(boolean resigned) {  
 this.resigned = resigned;  
 }  
  
 public boolean isPlaysAttack() {  
 return this.playsAttack;  
 }  
  
 public void setPlaysAttack(boolean playsAttack) {  
 this.playsAttack = playsAttack;  
 }  
  
 public boolean isDoneAttack() {  
 return this.doneAttack;  
 }  
  
 public void setDoneAttack(boolean doneAttack) {  
 this.doneAttack = doneAttack;  
 }  
}

Player

*/\*\*  
 \* Player is a class that keeps the player cards and handles in-game functions.  
 \*/*package com.example.dominion\_game.classes;  
  
import android.util.Pair;  
import com.example.dominion\_game.activities.GameActivity;  
  
import java.util.ArrayList;  
import java.util.HashMap;  
import java.util.Random;  
  
public class Player {  
 private ArrayList<String> discard;  
 private ArrayList<String> deck;  
 private HashMap<String, Integer> hand;  
 private ArrayList<Pair<String, Integer>> arrayHand;  
  
 */\*\*  
 \* The constructor with default values for the attributes  
 \*/* public Player() {  
 this.discard = new ArrayList<>();  
 this.deck = new ArrayList<>();  
 this.hand = new HashMap<>();  
 this.arrayHand = new ArrayList<>();  
 }  
  
 */\*\*  
 \* A function that transfers all cards from discard to deck and shuffles the deck.  
 \** ***@param*** *gameManager A reference to gameManager  
 \** ***@param*** *tabs An Integer which is the count of tabs in log that  
 \* the line that would be added will have  
 \*/* public void discardToDeck(GameManager gameManager, int tabs) {  
 gameManager.getLog().add(new LogLine("Shuffles his deck", gameManager.getGameManagerBeforeStart().getMyId(), false, true, "pink", tabs, "shuffle"));  
 this.deck.addAll(discard);  
 Random rand = new Random(); // creating Random object  
 for (int i = 0; i < this.deck.size(); i++) {  
 // switches between value in i with the value in a random index  
 int randomIndexToSwap = rand.nextInt(this.deck.size());  
 String temp = this.deck.get(randomIndexToSwap);  
 this.deck.set(randomIndexToSwap, this.deck.get(i));  
 this.deck.set(i, temp);  
 }  
 this.discard.clear();  
 }  
  
 */\*\*  
 \* A function that topdecks a card by removing it from hand and adding it to deck.  
 \** ***@param*** *cardName A String with the card that should move from hand to deck  
 \** ***@param*** *gameActivity A reference to gameActivity  
 \*/* public void handToDeck(String cardName, GameActivity gameActivity) {  
 if (this.hand.get(cardName) == 1) {  
 this.hand.remove(cardName);  
 gameActivity.getHandAdapter().notifyItemRemoved(this.getPositionByName(cardName));  
 this.updateArrayHand();  
 this.deck.add(cardName);  
 }  
 else if (this.hand.get(cardName) > 1) {  
 this.hand.put(cardName, this.hand.get(cardName) - 1);  
 this.updateArrayHand();  
 gameActivity.getHandAdapter().notifyItemChanged(this.getPositionByName(cardName));  
 this.deck.add(cardName);  
 }  
 }  
  
 */\*\*  
 \* A function that puts an array to discard.  
 \** ***@param*** *cards An ArrayList of Strings with the cards that should be added to discard  
 \*/* public void putArrayInDiscard(ArrayList<String> cards) {  
 this.discard.addAll(cards);  
 }  
  
 */\*\*  
 \* A function that puts an array to deck.  
 \** ***@param*** *cards An ArrayList of Strings with the cards that should be added to deck  
 \*/* public void putArrayInDeck(ArrayList<String> cards) {  
 this.deck.addAll(cards);  
 }  
  
 */\*\*  
 \* A function that takes cards from deck to hand.  
 \** ***@param*** *numberOfCards An Integer with the number of cards to take  
 \** ***@param*** *gameActivity A reference to gameActivity  
 \** ***@param*** *gameManager A reference to gameManager  
 \** ***@param*** *tabs An Integer which is the count of tabs in log that  
 \* the line that would be added will have  
 \*/* public void takeCardsToHand(int numberOfCards, GameActivity gameActivity, GameManager gameManager, int tabs) {  
 for (int i = 0; i < numberOfCards && !(this.discard.isEmpty() && this.deck.isEmpty()); i++) {  
 if (this.deck.isEmpty())  
 discardToDeck(gameManager, tabs);  
 if (this.hand.containsKey(this.deck.get(this.deck.size() - 1)))  
 this.hand.put(this.deck.get(this.deck.size() - 1), this.hand.get(this.deck.get(this.deck.size() - 1)) + 1);  
 else  
 this.hand.put(this.deck.get(this.deck.size() - 1), 1);  
 // removes the last index which is the first card to take from deck  
 this.deck.remove(this.deck.size() - 1);  
 }  
 this.updateArrayHand();  
 gameActivity.getHandAdapter().notifyDataSetChanged();  
  
 gameManager.getLog().add(new LogLine("Draws " + numberOfCards + " card" + (numberOfCards == 1 ? "" : "s"), gameManager.getGameManagerBeforeStart().getMyId(), false, false, tabs, "take cards"));  
 }  
  
 */\*\*  
 \* A function that takes cards from deck and returns them in ArrayList.  
 \** ***@param*** *numberOfCards An Integer with the number of cards to take  
 \** ***@param*** *gameManager A reference to gameManager  
 \** ***@param*** *tabs An Integer which is the count of tabs in log that  
 \* the line that would be added will have  
 \** ***@return*** *An ArrayList with the cards removed from deck  
 \*/* public ArrayList<String> takeCards(int numberOfCards, GameManager gameManager, int tabs) {  
 ArrayList<String> returnCards = new ArrayList<>();  
 for (int i = 0; i < numberOfCards && !(this.discard.isEmpty() && this.deck.isEmpty()); i++) {  
 if (this.deck.isEmpty())  
 discardToDeck(gameManager, tabs);  
 returnCards.add(this.deck.remove(this.deck.size() - 1));  
 }  
 return returnCards;  
 }  
  
 */\*\*  
 \* A function that checks whether there are cards of this type in hand.  
 \** ***@param*** *type A String with the type of the card (action, treasure or victory)  
 \** ***@return*** *A Boolean which is true if there is any card in hand from this type and false if not  
 \*/* public boolean containsTypeCards(String type) {  
 for (String cardName : this.hand.keySet())  
 if (Help.*nameToCard*(cardName).getType().equals(type))  
 return true;  
 return false;  
 }  
  
 */\*\*  
 \* A function that check if a card is in hand or not.  
 \** ***@param*** *cardName A String with the card name  
 \** ***@return*** *A Boolean which is true if the card is in hand and false if not  
 \*/* public boolean containsCard(String cardName) {  
 return this.hand.containsKey(cardName);  
 }  
  
 */\*\*  
 \* A function that returns all the cards that the player has in game as a HashMap.  
 \** ***@return*** *A HashMap with all cards and count of them that the player has in game  
 \*/* public HashMap<String, Integer> allCards(GameManager gameManager) {  
 HashMap <String, Integer> cards = new HashMap<>(this.hand);  
 for (String cardName : this.deck)  
 if (cards.containsKey(cardName))  
 cards.put(cardName, cards.get(cardName) + 1);  
 else  
 cards.put(cardName, 1);  
  
 for (String cardName : this.discard)  
 if (cards.containsKey(cardName))  
 cards.put(cardName, cards.get(cardName) + 1);  
 else  
 cards.put(cardName, 1);  
  
 if (gameManager.getTurn().isMyTurn(gameManager.getGameManagerBeforeStart())) {  
 for (String cardName : gameManager.getTurn().getActionCardsPlayed())  
 if (cards.containsKey(cardName))  
 cards.put(cardName, cards.get(cardName) + 1);  
 else  
 cards.put(cardName, 1);  
  
 for (String cardName : gameManager.getTurn().getTreasureCardsPlayed().keySet())  
 for (int i = 0; i < gameManager.getTurn().getTreasureCardsPlayed().get(cardName); i++)  
 if (cards.containsKey(cardName))  
 cards.put(cardName, cards.get(cardName) + 1);  
 else  
 cards.put(cardName, 1);  
 }  
 return cards;  
 }  
  
 */\*\*  
 \* A function that returns the count of victory points that the player has.  
 \** ***@return*** *An Integer which is the number of victory points that the player has  
 \*/* public int getVictoryPoints(GameManager gameManager) {  
 int victoryPoints = 0;  
 HashMap<String, Integer> cards = this.allCards(gameManager);  
 for (String cardName : cards.keySet())  
 if (Help.*nameToCard*(cardName).getType().equals("victory"))  
 victoryPoints += Help.*nameToCard*(cardName).getValue(gameManager)\*cards.get(cardName);  
  
 return victoryPoints;  
 }  
  
 public ArrayList<String> getDiscard() {  
 return this.discard;  
 }  
  
 public ArrayList<String> getDeck() {  
 return deck;  
 }  
  
 */\*\*  
 \* A function that adds a card to deck.  
 \** ***@param*** *cardName A String with the card name  
 \*/* public void addToDeck(String cardName) {  
 if (!cardName.equals(""))  
 this.deck.add(cardName);  
 }  
  
 */\*\*  
 \* A function that adds a card to discard.  
 \** ***@param*** *cardName A String with the card name  
 \*/* public void addToDiscard(String cardName) {  
 if (!cardName.equals(""))  
 this.discard.add(cardName);  
 }  
  
 */\*\*  
 \* A function that adds a card to hand.  
 \** ***@param*** *cardName A String with the card name  
 \** ***@return*** *A Boolean of success or not  
 \*/* public boolean addToHand(String cardName) {  
 if (cardName.equals(""))  
 return false;  
  
 if (this.hand.containsKey(cardName))  
 this.hand.put(cardName, this.hand.get(cardName) + 1);  
 else  
 this.hand.put(cardName, 1);  
 return true;  
 }  
  
 */\*\*  
 \* A function that removes a card to hand.  
 \** ***@param*** *cardName A String with the card name  
 \** ***@return*** *A Boolean of success or not  
 \*/* public boolean removeFromHand(String cardName) {  
 if (!this.hand.containsKey(cardName))  
 return false;  
 if (this.hand.get(cardName) == 1) {  
 this.hand.remove(cardName);  
 return true;  
 }  
 if (this.hand.get(cardName) > 1) {  
 this.hand.put(cardName, this.hand.get(cardName) - 1);  
 return true;  
 }  
 return false;  
 }  
  
 public HashMap<String, Integer> getHand() {  
 return this.hand;  
 }  
  
 public void setHand(HashMap<String, Integer> hand) {  
 this.hand = hand;  
 }  
  
 public ArrayList<Pair<String,Integer>> getArrayHand() {  
 return this.arrayHand;  
 }  
  
 */\*\*  
 \* A function that returns the position of the card in arrayHand.  
 \** ***@param*** *cardName A String with the name of card  
 \** ***@return*** *An Integer which is the index of the card in arrayHand  
 \*/* public int getPositionByName(String cardName) {  
 for (int i = 0; i < this.arrayHand.size(); i++) {  
 if (this.arrayHand.get(i).first.equals(cardName))  
 return i;  
 }  
 return -1;  
 }  
  
 */\*\*  
 \* A function that updates the arrayHand to be same as in hand.  
 \*/* public void updateArrayHand() {  
 this.arrayHand.clear();  
 for (String cardName : this.hand.keySet()) {  
 this.arrayHand.add(new Pair<>(cardName, this.hand.get(cardName)));  
 }  
 }  
}

Turn

*/\*\*  
 \* Turn is a class that keeps all the relevant data about the turn  
 \*/*package com.example.dominion\_game.classes;  
  
import org.json.JSONException;  
import org.json.JSONObject;  
  
import java.util.ArrayList;  
import java.util.HashMap;  
  
public class Turn {  
  
 private String phase;  
 private String turnId;  
 private ArrayList<String> actionCardsPlayed;  
 private HashMap<String, Integer> treasureCardsPlayed;  
 private HashMap<String, Integer> cardsBought;  
 private String lastActionCardForWait;  
 private ArrayList<PlayCardArguments> cardsForWaitForPlay;  
 private String lastCardForWaitForPlay;  
 private boolean isWaitingForEnemy;  
 private WaitForFunction waitForFunction;  
 private int actions;  
 private int buys;  
 private int treasure;  
 private int turnNumber;  
 private boolean forcedActionEnd;  
  
 */\*\*  
 \* A constructor with default values for all attributes but turnId  
 \*/* public Turn(String turnId) {  
 this.turnId = turnId;  
 this.phase = "";  
 this.actionCardsPlayed = new ArrayList<>();  
 this.treasureCardsPlayed = new HashMap<>();  
 this.cardsBought = new HashMap<>();  
 this.lastActionCardForWait = "";  
 this.cardsForWaitForPlay = new ArrayList<>();  
 this.lastCardForWaitForPlay = "";  
 this.actions = 1;  
 this.buys = 1;  
 this.treasure = 0;  
 this.turnNumber = 1;  
 this.isWaitingForEnemy = false;  
 this.waitForFunction = new WaitForFunction();  
 this.forcedActionEnd = false;  
 }  
  
 */\*\*  
 \* A constructor with default values for all attributes  
 \*/* public Turn() {  
 this.turnId = "";  
 this.phase = "";  
 this.actionCardsPlayed = new ArrayList<>();  
 this.treasureCardsPlayed = new HashMap<>();  
 this.cardsBought = new HashMap<>();  
 this.lastActionCardForWait = "";  
 this.cardsForWaitForPlay = new ArrayList<>();  
 this.lastCardForWaitForPlay = "";  
 this.actions = 1;  
 this.buys = 1;  
 this.treasure = 0;  
 this.turnNumber = 1;  
 this.isWaitingForEnemy = false;  
 this.waitForFunction = new WaitForFunction();  
 this.forcedActionEnd = false;  
 }  
  
 public ArrayList<String> getActionCardsPlayed() {  
 return this.actionCardsPlayed;  
 }  
  
 public void setActionCardsPlayed(ArrayList<String> actionCardsPlayed) {  
 this.actionCardsPlayed = actionCardsPlayed;  
 }  
  
 public HashMap<String, Integer> getTreasureCardsPlayed() {  
 return this.treasureCardsPlayed;  
 }  
  
 public HashMap<String, Integer> getCardsBought() {  
 return this.cardsBought;  
 }  
  
 public String getPhase() {  
 return this.phase;  
 }  
  
 public void setPhase(String phase) {  
 this.phase = phase;  
 }  
  
 public String getTurnId() {  
 return this.turnId;  
 }  
  
 public void setTurnId(String turnId) {  
 this.turnId = turnId;  
 }  
  
 */\*\*  
 \* A function that restart values in turn and changes the turnId.  
 \** ***@param*** *gameManagerBeforeStart A reference to gameManagerBeforeStart  
 \*/* public void changeTurn(GameManagerBeforeStart gameManagerBeforeStart) {  
 this.actionCardsPlayed.clear();  
 this.treasureCardsPlayed.clear();  
 this.cardsBought.clear();  
 this.lastActionCardForWait = "";  
 this.cardsForWaitForPlay.clear();  
 this.lastCardForWaitForPlay = "";  
 this.actions = 1;  
 this.buys = 1;  
 this.treasure = 0;  
 this.turnNumber++;  
 this.phase = "";  
 if (this.turnId.equals(gameManagerBeforeStart.getIdP1()))  
 this.turnId = gameManagerBeforeStart.getIdP2();  
 else  
 this.turnId = gameManagerBeforeStart.getIdP1();  
 this.isWaitingForEnemy = false;  
 this.waitForFunction.clear();  
 this.forcedActionEnd = false;  
 }  
  
 */\*\*  
 \* A function that returns if it is my turn or not.  
 \** ***@param*** *gameManagerBeforeStart A reference to gameManagerBeforeStart  
 \** ***@return*** *A Boolean which is true if it is my turn and false if not  
 \*/* public boolean isMyTurn(GameManagerBeforeStart gameManagerBeforeStart) {  
 if (gameManagerBeforeStart.isCreator())  
 return this.turnId.equals(gameManagerBeforeStart.getIdP1());  
 else  
 return this.turnId.equals(gameManagerBeforeStart.getIdP2());  
 }  
  
 public void addActionCard(String cardName) {  
 this.actionCardsPlayed.add(cardName);  
 }  
  
 public void addWaitForEnemy(String cardName) {  
 this.lastActionCardForWait = cardName;  
 this.isWaitingForEnemy = true;  
 }  
  
 public void addWaitForPlay(String cardName, int n, int i, boolean forceUse, boolean autoPlay) {  
 this.cardsForWaitForPlay.add(new PlayCardArguments(cardName, n, i, forceUse, autoPlay));  
 }  
  
 public void removeLastAttack() {  
 this.lastActionCardForWait = "";  
 }  
  
 public String getLastActionCardForWait() {  
 return this.lastActionCardForWait;  
 }  
  
 public void setLastActionCardForWait(String lastActionCardForWait) {  
 this.lastActionCardForWait = lastActionCardForWait;  
 }  
  
 public ArrayList<PlayCardArguments> getCardsForWaitForPlay() {  
 return this.cardsForWaitForPlay;  
 }  
  
 public void setCardsForWaitForPlay(ArrayList<PlayCardArguments> cardsForWaitForPlay) {  
 this.cardsForWaitForPlay = cardsForWaitForPlay;  
 }  
  
 public String getLastCardForWaitForPlay() {  
 return this.lastCardForWaitForPlay;  
 }  
  
 public void setLastCardForWaitForPlay(String lastCardForWaitForPlay) {  
 this.lastCardForWaitForPlay = lastCardForWaitForPlay;  
 }  
  
 public void setWaitingForEnemy(boolean waitingForEnemy) {  
 this.isWaitingForEnemy = waitingForEnemy;  
 }  
  
 public boolean isWaitingForEnemy() {  
 return this.isWaitingForEnemy;  
 }  
  
 public void addTreasureCard(String cardName) {  
 if (this.treasureCardsPlayed.containsKey(cardName))  
 this.treasureCardsPlayed.put(cardName, this.treasureCardsPlayed.get(cardName) + 1);  
 else  
 this.treasureCardsPlayed.put(cardName, 1);  
 }  
  
 public void addCardBought(String cardName) {  
 if (this.cardsBought.containsKey(cardName))  
 this.cardsBought.put(cardName, this.cardsBought.get(cardName) + 1);  
 else  
 this.cardsBought.put(cardName, 1);  
 }  
  
 public int getActions() {  
 return this.actions;  
 }  
  
 public void setActions(int actions) {  
 this.actions = actions;  
 }  
  
 public void addActions(int actions) {  
 this.actions += actions;  
 }  
  
 public void useAction() {  
 this.actions -= 1;  
 }  
  
 public int getBuys() {  
 return this.buys;  
 }  
  
 public void setBuys(int buys) {  
 this.buys = buys;  
 }  
  
 public void addBuys(int buys) {  
 this.buys += buys;  
 }  
  
 public void useBuy() {  
 this.buys -= 1;  
 }  
  
 public int getTreasure() {  
 return this.treasure;  
 }  
  
 public void setTreasure(int treasure) {  
 this.treasure = treasure;  
 }  
  
 public void addTreasure(int treasure) {  
 this.treasure += treasure;  
 }  
  
 public void useTreasure(int treasure) {  
 this.treasure -= treasure;  
 }  
  
 public int getTurnNumber() {  
 return this.turnNumber;  
 }  
  
 public void setTurnNumber(int turnNumber) {  
 this.turnNumber = turnNumber;  
 }  
  
 public WaitForFunction getWaitForFunction() {  
 return this.waitForFunction;  
 }  
  
 public boolean getForcedActionEnd() {  
 return this.forcedActionEnd;  
 }  
  
 public void setForcedActionEnd(boolean forcedActionEnd) {  
 this.forcedActionEnd = forcedActionEnd;  
 }  
  
 @Override  
 public String toString() {  
 return "actions=" + this.actions + ", buys=" + this.buys + ", treasure=" + this.treasure;  
 }  
  
 */\*\*  
 \* A function that creates a JSONObject from the data in turn to upload to server.  
 \** ***@return*** *A JSONObject of some attributes from Turn  
 \*/* public JSONObject turnToJson(boolean doneAttack) {  
 JSONObject jsonObject = new JSONObject();  
 try {  
 jsonObject.put("actions", this.actions);  
 jsonObject.put("buys", this.buys);  
 jsonObject.put("treasure", this.treasure);  
 jsonObject.put("turnId", this.turnId);  
 jsonObject.put("phase", this.phase);  
 if (doneAttack)  
 jsonObject.put("lastActionCardForWait", this.lastActionCardForWait);  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
  
 return jsonObject;  
 }  
  
 */\*\*  
 \* A function that updates the values on some attributes  
 \** ***@param*** *jsonObject A JSONObject that is given from the server  
 \*/* public void jsonToTurn(JSONObject jsonObject, GameManager gameManager) {  
 try {  
 this.actions = jsonObject.getInt("actions");  
 this.buys = jsonObject.getInt("buys");  
 this.treasure = jsonObject.getInt("treasure");  
 this.turnId = jsonObject.getString("turnId");  
 this.phase = jsonObject.getString("phase");  
 if (!this.isMyTurn(gameManager.getGameManagerBeforeStart()))  
 this.lastActionCardForWait = jsonObject.getString("lastActionCardForWait");  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
 }  
}

WaitForFunction

*/\*\*  
 \* WaitForFunction is a class that keeps all data about waiting for  
 \* things in game (enemy, clicks...).  
 \*/*package com.example.dominion\_game.classes;  
  
import android.util.Pair;  
  
import java.util.ArrayList;  
import java.util.HashMap;  
  
public class WaitForFunction {  
 private boolean isWaitingForActionCardsDialog;  
 private boolean isWaitingForButtonsOnly;  
 private boolean isWaitingForBoard;  
 private boolean isWaitingForHand;  
 private String cardName;  
 private int minAmount;  
 private int maxAmount;  
 private int minPriceForGain;  
 private int maxPriceForGain;  
 private String typeOfAction;  
 private HashMap<String, Integer> cardsForActionCardPlay;  
 private ArrayList<Pair<String, Boolean>> cardsForDialog;  
 private boolean handleClickOnCard;  
 private boolean hasUndoAndConfirm;  
  
 */\*\*  
 \* The constructor with default values.  
 \*/* public WaitForFunction() {  
 this.isWaitingForActionCardsDialog = false;  
 this.isWaitingForButtonsOnly = false;  
 this.isWaitingForBoard = false;  
 this.isWaitingForHand = false;  
 this.cardName = "";  
 this.minAmount = 0;  
 this.maxAmount = 0;  
 this.minPriceForGain = 0;  
 this.maxPriceForGain = 0;  
 this.typeOfAction = "";  
 this.cardsForActionCardPlay = new HashMap<>();  
 this.cardsForDialog = new ArrayList<>();  
 this.handleClickOnCard = false;  
 this.hasUndoAndConfirm = false;  
 }  
  
 */\*\*  
 \* A function that handles waiting for hand.  
 \** ***@param*** *cardName A String which is the name of the card that is waiting for clicking on hand  
 \** ***@param*** *minAmount An Integer with the minimum of cards that should be selected from hand  
 \** ***@param*** *maxAmount An Integer with the maximum of cards that should be selected from hand  
 \** ***@param*** *typeOfAction A String with the type of action that will be  
 \* done with the selected cards (trash, discard...)  
 \** ***@param*** *handleClickOnCard A Boolean which is true if the card should handle every  
 \* click on card in hand and false if not  
 \*/* public void handleWaitingForHand(String cardName, int minAmount, int maxAmount, String typeOfAction, boolean handleClickOnCard) {  
 this.isWaitingForHand = true;  
 this.cardName = cardName;  
 this.minAmount = minAmount;  
 this.maxAmount = maxAmount;  
 this.typeOfAction = typeOfAction;  
 this.handleClickOnCard = handleClickOnCard;  
 }  
  
 */\*\*  
 \* A function that handles waiting for dialog.  
 \** ***@param*** *cardName A String which is the name of the card that is waiting for clicking on hand  
 \** ***@param*** *minAmount An Integer with the minimum of cards that should be selected from hand  
 \** ***@param*** *maxAmount An Integer with the maximum of cards that should be selected from hand  
 \** ***@param*** *typeOfAction A String with the type of action that will be  
 \* done with the selected cards (trash, discard...)  
 \** ***@param*** *handleClickOnCard A Boolean which is true if the card should handle every  
 \* click on card in hand and false if not  
 \** ***@param*** *cards An ArrayList of String with the cards should be displayed on the dialog  
 \*/* public void handleWaitingForActionCardsDialog(String cardName, int minAmount, int maxAmount, String typeOfAction, boolean handleClickOnCard, ArrayList<String> cards) {  
 this.isWaitingForActionCardsDialog = true;  
 this.cardName = cardName;  
 this.minAmount = minAmount;  
 this.maxAmount = maxAmount;  
 this.typeOfAction = typeOfAction;  
 this.handleClickOnCard = handleClickOnCard;  
 for(String card : cards)  
 this.addToCardsForDialog(card);  
 }  
  
 */\*\*  
 \* A function that adds a card to the dialog.  
 \** ***@param*** *card A String with the name of the card to be added  
 \*/* public void addToCardsForDialog(String card) {  
 this.cardsForDialog.add(new Pair<>(card, false));  
 }  
  
 */\*\*  
 \* A function that handles waiting for board.  
 \** ***@param*** *cardName A String which is the name of the card that is waiting for clicking on board  
 \** ***@param*** *minAmount An Integer with the minimum of cards that should be selected from board  
 \** ***@param*** *maxAmount An Integer with the maximum of cards that should be selected from board  
 \*/* public void handleWaitingForBoard(String cardName, int minAmount, int maxAmount) {  
 this.isWaitingForBoard = true;  
 this.cardName = cardName;  
 this.minAmount = minAmount;  
 this.maxAmount = maxAmount;  
 }  
  
 */\*\*  
 \* A function that handles waiting only for buttons.  
 \** ***@param*** *cardName A String with the name of card that is waiting for buttons  
 \*/* public void handleWaitingForButtonsOnly(String cardName) {  
 this.isWaitingForButtonsOnly = true;  
 this.cardName = cardName;  
 }  
  
 */\*\*  
 \* A function that puts the default values for all arguments.  
 \*/* public void clear() {  
 this.isWaitingForActionCardsDialog = false;  
 this.isWaitingForButtonsOnly = false;  
 this.isWaitingForBoard = false;  
 this.isWaitingForHand = false;  
 this.cardName = "";  
 this.minAmount = 0;  
 this.maxAmount = 0;  
 this.minPriceForGain = 0;  
 this.maxPriceForGain = 0;  
 this.typeOfAction = "";  
 this.cardsForActionCardPlay.clear();  
 this.cardsForDialog.clear();  
 this.handleClickOnCard = false;  
 this.hasUndoAndConfirm = false;  
 }  
  
 */\*\*  
 \* A function that handles undo and clears the cards that were selected.  
 \*/* public void undo() {  
 this.cardsForActionCardPlay.clear();  
 if (isWaitingForActionCardsDialog)  
 for(int i = 0; i < this.cardsForDialog.size(); i++)  
 this.cardsForDialog.set(i, new Pair<>(this.cardsForDialog.get(i).first, false));  
 }  
  
 */\*\*  
 \* A function that returns all the selected cards in dialog.  
 \** ***@return*** *An ArrayList of String with all the card selected in dialog  
 \*/* public ArrayList<String> cardsSelectedForDialog() {  
 ArrayList<String> al = new ArrayList<>();  
 for (int i = 0; i < this.cardsForDialog.size(); i++) {  
 if (this.cardsForDialog.get(i).second)  
 al.add(this.cardsForDialog.get(i).first);  
 }  
 return al;  
 }  
  
 */\*\*  
 \* A function that returns all the non-selected cards in dialog.  
 \** ***@return*** *An ArrayList of String with all the card non-selected in dialog  
 \*/* public ArrayList<String> cardsLeftForDialog() {  
 ArrayList<String> al = new ArrayList<>();  
 for (int i = 0; i < this.cardsForDialog.size(); i++) {  
 if (!this.cardsForDialog.get(i).second)  
 al.add(this.cardsForDialog.get(i).first);  
 }  
 return al;  
 }  
  
 */\*\*  
 \* A function that updates the cards by position after selecting or unselecting in dialog.  
 \** ***@param*** *position An Integer with the position of the card in cardsForDialog  
 \** ***@param*** *isSelected A Boolean which is true if the card was selected and false if unselected.  
 \*/* public void updateCardsForDialogByPosition(int position, boolean isSelected) {  
 this.cardsForDialog.set(position, new Pair<>(this.cardsForDialog.get(position).first, isSelected));  
 String cardName = this.cardsForDialog.get(position).first;  
 if (isSelected) {  
 if (this.cardsForActionCardPlay.containsKey(cardName))  
 this.cardsForActionCardPlay.put(cardName, this.cardsForActionCardPlay.get(cardName) + 1);  
 else  
 this.cardsForActionCardPlay.put(cardName, 1);  
 }  
 else {  
 if (this.cardsForActionCardPlay.get(cardName) > 1)  
 this.cardsForActionCardPlay.put(cardName, this.cardsForActionCardPlay.get(cardName) - 1);  
 else if (this.cardsForActionCardPlay.get(cardName) == 1)  
 this.cardsForActionCardPlay.remove(cardName);  
 }  
 }  
  
 */\*\*  
 \* A function that reorders and replaces between the two cards that should be reordered.  
 \*/* public void order() {  
 this.cardsForActionCardPlay.clear();  
 ArrayList<Integer> al = new ArrayList<>();  
 for (int i = 0; i < this.cardsForDialog.size(); i++) {  
 if (this.cardsForDialog.get(i).second) {  
 al.add(i);  
 }  
 }  
 if (al.size() != 2)  
 return;  
  
 String temp = this.cardsForDialog.get(0).first;  
 this.cardsForDialog.set(0, new Pair<>(this.cardsForDialog.get(1).first, false));  
 this.cardsForDialog.set(1, new Pair<>(temp, false));  
 }  
  
 */\*\*  
 \* A function that inserts the card that selected to cardsForActionCardPlay.  
 \** ***@param*** *cardName A String with the name of the card  
 \*/* public void insertCardSelectedInHand(String cardName) {  
 if (this.cardsForActionCardPlay.containsKey(cardName))  
 this.cardsForActionCardPlay.put(cardName, this.cardsForActionCardPlay.get(cardName) + 1);  
 else  
 this.cardsForActionCardPlay.put(cardName, 1);  
 }  
  
 */\*\*  
 \* A function that returns the amount of a specific cards that selected.  
 \** ***@param*** *cardName A String with the card name  
 \** ***@return*** *An Integer with the amount of a specific cards that selected  
 \*/* public int getCardAmountInCardsInHand(String cardName) {  
 if (this.cardsForActionCardPlay.containsKey(cardName))  
 return this.cardsForActionCardPlay.get(cardName);  
 return 0;  
 }  
  
 public boolean isHandleClickOnCard() {  
 return this.handleClickOnCard;  
 }  
  
 public void setHandleClickOnCard(boolean handleClickOnCard) {  
 this.handleClickOnCard = handleClickOnCard;  
 }  
  
 public int getMinPriceForGain() {  
 return this.minPriceForGain;  
 }  
  
 public void setMinPriceForGain(int minPriceForGain) {  
 this.minPriceForGain = minPriceForGain;  
 }  
  
 public int getMaxPriceForGain() {  
 return this.maxPriceForGain;  
 }  
  
 public void setMaxPriceForGain(int maxPriceForGain) {  
 this.maxPriceForGain = maxPriceForGain;  
 }  
  
 public HashMap<String, Integer> getCardsForActionCardPlay() {  
 return this.cardsForActionCardPlay;  
 }  
  
 public void setCardsForActionCardPlay(HashMap<String, Integer> cardsForActionCardPlay) {  
 this.cardsForActionCardPlay = cardsForActionCardPlay;  
 }  
  
 public ArrayList<Pair<String, Boolean>> getCardsForDialog() {  
 return this.cardsForDialog;  
 }  
  
 public void setCardsForDialog(ArrayList<Pair<String, Boolean>> cardsForDialog) {  
 this.cardsForDialog = cardsForDialog;  
 }  
  
 public boolean isWaitingForActionCardsDialog() {  
 return this.isWaitingForActionCardsDialog;  
 }  
  
 public void setWaitingForActionCardsDialog(boolean waitingForActionCardsDialog) {  
 this.isWaitingForActionCardsDialog = waitingForActionCardsDialog;  
 }  
  
 public boolean isWaitingForButtonsOnly() {  
 return this.isWaitingForButtonsOnly;  
 }  
  
 public void setWaitingForButtonsOnly(boolean waitingForButtonsOnly) {  
 this.isWaitingForButtonsOnly = waitingForButtonsOnly;  
 }  
  
 public boolean isWaitingForBoard() {  
 return this.isWaitingForBoard;  
 }  
  
 public void setWaitingForBoard(boolean waitingForBoard) {  
 this.isWaitingForBoard = waitingForBoard;  
 }  
  
 public boolean isWaitingForHand() {  
 return this.isWaitingForHand;  
 }  
  
 public void setWaitingForHand(boolean waitingForHand) {  
 this.isWaitingForHand = waitingForHand;  
 }  
  
 public String getCardName() {  
 return this.cardName;  
 }  
  
 public void setCardName(String cardName) {  
 this.cardName = cardName;  
 }  
  
 public int getMinAmount() {  
 return this.minAmount;  
 }  
  
 public void setMinAmount(int minAmount) {  
 this.minAmount = minAmount;  
 }  
  
 public int getMaxAmount() {  
 return this.maxAmount;  
 }  
  
 public void setMaxAmount(int maxAmount) {  
 this.maxAmount = maxAmount;  
 }  
  
 public String getTypeOfAction() {  
 return this.typeOfAction;  
 }  
  
 public void setTypeOfAction(String typeOfAction) {  
 this.typeOfAction = typeOfAction;  
 }  
  
 public boolean isHasUndoAndConfirm() {  
 return this.hasUndoAndConfirm;  
 }  
  
 public void setHasUndoAndConfirm(boolean hasUndoAndConfirm) {  
 this.hasUndoAndConfirm = hasUndoAndConfirm;  
 }  
}

EnemyData

*/\*\*  
 \* EnemyData is a class that keeps all the relevant data about the enemy  
 \* that is given from the server.  
 \*/*package com.example.dominion\_game.classes;  
  
import org.json.JSONException;  
import org.json.JSONObject;  
  
public class EnemyData {  
 private String lastCardOnDiscard;  
 private int deckSize;  
 private int handSize;  
 private int victoryPoints;  
 private boolean isEmpty;  
 private boolean resigned;  
  
 */\*\*  
 \* The constructor with default values for the attributes  
 \*/* public EnemyData() {  
 this.isEmpty = true;  
 this.lastCardOnDiscard = "";  
 this.deckSize = 0;  
 this.handSize = 0;  
 this.victoryPoints = 0;  
 this.resigned = false;  
 }  
  
 public String getLastCardOnDiscard() {  
 return this.lastCardOnDiscard;  
 }  
  
 public void setLastCardOnDiscard(String lastCardOnDiscard) {  
 this.lastCardOnDiscard = lastCardOnDiscard;  
 }  
  
 public int getDeckSize() {  
 return this.deckSize;  
 }  
  
 public void setDeckSize(int deckSize) {  
 this.deckSize = deckSize;  
 }  
  
 public int getHandSize() {  
 return this.handSize;  
 }  
  
 public void setHandSize(int handSize) {  
 this.handSize = handSize;  
 }  
  
 public int getVictoryPoints() {  
 return this.victoryPoints;  
 }  
  
 public void setVictoryPoints(int victoryPoints) {  
 this.victoryPoints = victoryPoints;  
 }  
  
 public boolean isEmpty() {  
 return this.isEmpty;  
 }  
  
 public boolean isResigned() {  
 return this.resigned;  
 }  
  
 */\*\*  
 \* A function that updates the values on the attributes  
 \** ***@param*** *jsonObject A JSONObject that is given from the server  
 \** ***@param*** *gameManager A reference to gameManager  
 \*/* public void jsonToEnemyData(JSONObject jsonObject, GameManager gameManager) {  
 try {  
 this.isEmpty = false;  
 if (this.handSize != jsonObject.getInt("handSize")  
 || !this.lastCardOnDiscard.equals(jsonObject.getString("lastCardOnDiscard"))  
 || this.deckSize != jsonObject.getInt("deckSize")  
 || this.victoryPoints != jsonObject.getInt("victoryPoints")) {  
 this.handSize = jsonObject.getInt("handSize");  
 if (gameManager.isStarted()) {  
 this.lastCardOnDiscard = jsonObject.getString("lastCardOnDiscard");  
 this.deckSize = jsonObject.getInt("deckSize");  
 this.victoryPoints = jsonObject.getInt("victoryPoints");  
 }  
 }  
 this.resigned = jsonObject.getBoolean("resigned");  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
 }  
}

PlayCardArguments

*/\*\*  
 \* PlayCardArguments is a class that contains all parameter of playing a future card.  
 \*/*package com.example.dominion\_game.classes;  
  
public class PlayCardArguments {  
 private String cardName;  
 private int n;  
 private int i;  
 private boolean forceUse;  
 private boolean autoPlay;  
  
 */\*\*  
 \* The constructor  
 \** ***@param*** *cardName A String which is the name of the card which should be used  
 \** ***@param*** *n An Integer which is the number of times that the card should be played  
 \** ***@param*** *i An Integer which is the place of the card in the wait queue  
 \** ***@param*** *forceUse A Boolean which is true if the use of the card was forced or not  
 \** ***@param*** *autoPlay A Boolean which is true if useCard was played with autoPlay and false if not  
 \*/* public PlayCardArguments(String cardName, int n, int i, boolean forceUse, boolean autoPlay) {  
 this.cardName = cardName;  
 this.n = n;  
 this.i = i;  
 this.forceUse = forceUse;  
 this.autoPlay = autoPlay;  
 }  
  
 public String getCardName() {  
 return this.cardName;  
 }  
  
 public int getN() {  
 return this.n;  
 }  
  
 public int getI() {  
 return this.i;  
 }  
  
 public boolean isForceUse() {  
 return this.forceUse;  
 }  
  
 public boolean isAutoPlay() {  
 return this.autoPlay;  
 }  
}

LogLine

*/\*\*  
 \* LogLine is a class that keeps all the relevant data of every line in the log.  
 \*/*package com.example.dominion\_game.classes;  
  
import org.json.JSONException;  
import org.json.JSONObject;  
  
public class LogLine {  
 private String text;  
 private String playerId;  
 private boolean isBold;  
 private boolean isItalic;  
 private String color;  
 private int tabs;  
 private String type;  
  
 */\*\*  
 \* A constructor with all attributes but color  
 \** ***@param*** *text A String with the text for the line  
 \** ***@param*** *playerId A String with the playerId that this line belongs to  
 \** ***@param*** *isBold A Boolean which is true if the line should be bold  
 \** ***@param*** *isItalic A Boolean which is true if the line should be italic  
 \** ***@param*** *tabs An Integer with the count of tabs for this line  
 \** ***@param*** *type A String with type of the line  
 \*/* public LogLine(String text, String playerId, boolean isBold, boolean isItalic, int tabs, String type) {  
 this.text = text;  
 this.playerId = playerId;  
 this.isBold = isBold;  
 this.isItalic = isItalic;  
 this.color = "white";  
 this.tabs = tabs;  
 this.type = type;  
 }  
  
 */\*\*  
 \* A constructor with all attributes  
 \** ***@param*** *text A String with the text for the line  
 \** ***@param*** *playerId A String with the playerId that this line belongs to  
 \** ***@param*** *isBold A Boolean which is true if the line should be bold  
 \** ***@param*** *isItalic A Boolean which is true if the line should be italic  
 \** ***@param*** *tabs An Integer with the count of tabs for this line  
 \** ***@param*** *type A String with type of the line  
 \** ***@param*** *color A String with the color for the line  
 \*/* public LogLine(String text, String playerId, boolean isBold, boolean isItalic, String color, int tabs, String type) {  
 this.text = text;  
 this.playerId = playerId;  
 this.isBold = isBold;  
 this.isItalic = isItalic;  
 this.color = color;  
 this.tabs = tabs;  
 this.type = type;  
 }  
  
 */\*\*  
 \* A constructor with all attributes as JSONObject  
 \** ***@param*** *jsonObject A JSONObject that is given from the server  
 \*/* public LogLine(JSONObject jsonObject) {  
 try {  
 this.text = jsonObject.getString("text");  
 this.playerId = jsonObject.getString("playerId");  
 this.isBold = jsonObject.getBoolean("isBold");  
 this.isItalic = jsonObject.getBoolean("isItalic");  
 this.color = jsonObject.getString("color");  
 this.tabs = jsonObject.getInt("tabs");  
 this.type = jsonObject.getString("type");  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public String getText() {  
 return this.text;  
 }  
  
 public void setText(String text) {  
 this.text = text;  
 }  
  
 public String getPlayerId() {  
 return this.playerId;  
 }  
  
 public void setPlayerId(String playerId) {  
 this.playerId = playerId;  
 }  
  
 public boolean isBold() {  
 return this.isBold;  
 }  
  
 public void setBold(boolean bold) {  
 this.isBold = bold;  
 }  
  
 public boolean isItalic() {  
 return this.isItalic;  
 }  
  
 public void setItalic(boolean italic) {  
 this.isItalic = italic;  
 }  
  
 public String getColor() {  
 return this.color;  
 }  
  
 public void setColor(String color) {  
 this.color = color;  
 }  
  
 public int getTabs() {  
 return this.tabs;  
 }  
  
 public void setTabs(int tabs) {  
 this.tabs = tabs;  
 }  
  
 public String getType() {  
 return this.type;  
 }  
  
 public void setType(String type) {  
 this.type = type;  
 }  
  
 */\*\*  
 \* A function that creates a JSONObject from the data in  
 \* logLine to upload to server.  
 \** ***@return*** *A JSONObject of the login details  
 \*/* public JSONObject logLineToJson() {  
 JSONObject jsonObject = new JSONObject();  
 try {  
 jsonObject.put("text", this.text);  
 jsonObject.put("playerId", this.playerId);  
 jsonObject.put("isBold", this.isBold);  
 jsonObject.put("isItalic", this.isItalic);  
 jsonObject.put("color", this.color);  
 jsonObject.put("tabs", this.tabs);  
 jsonObject.put("type", this.type);  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
  
 return jsonObject;  
 }  
}

GameManagerBeforeStart

*/\*\*  
 \* GameManagerBeforeStart is a class that keeps all the relevant data  
 \* before starting the game.  
 \*/*package com.example.dominion\_game.classes;  
  
import org.json.JSONException;  
import org.json.JSONObject;  
  
import java.io.Serializable;  
  
public class GameManagerBeforeStart implements Serializable {  
 private String gameId;  
 private String idP1;  
 private String idP2;  
 private boolean isCreator;  
 private boolean isReady1;  
 private boolean isReady2;  
 private boolean isStart1;  
 private boolean isStart2;  
 private boolean isRated;  
  
 */\*\*  
 \* The constructor for the creator  
 \** ***@param*** *creatorId A String with the id of the creator  
 \*/* public GameManagerBeforeStart(String creatorId) {  
 this.gameId = "";  
 this.idP1 = creatorId;  
 this.idP2 = "";  
 this.isCreator = true;  
 restartReady();  
 this.isRated = false;  
 }  
  
 */\*\*  
 \* The constructor for the non creator  
 \** ***@param*** *tableId A String with the table id  
 \** ***@param*** *nonCreatorId A String with the id of the non creator  
 \*/* public GameManagerBeforeStart(String tableId, String nonCreatorId) {  
 this.gameId = tableId;  
 this.idP1 = "";  
 this.idP2 = nonCreatorId;  
 this.isCreator = false;  
 restartReady();  
 this.isRated = false;  
 }  
  
 */\*\*  
 \* A function that restart the ready and start attributes for both players to be false.  
 \* The function is called before the game starts.  
 \*/* public void restartReady() {  
 this.isReady1 = false;  
 this.isReady2 = false;  
 this.isStart1 = false;  
 this.isStart2 = false;  
 }  
  
 public String getGameId() {  
 return this.gameId;  
 }  
  
 public void setGameId(String gameId) {  
 this.gameId = gameId;  
 }  
  
 public String getIdP1() {  
 return this.idP1;  
 }  
  
 public void setIdP1(String idP1) {  
 this.idP1 = idP1;  
 }  
  
 public String getIdP2() {  
 return this.idP2;  
 }  
  
 public void setIdP2(String idP2) {  
 this.idP2 = idP2;  
 }  
  
 public boolean isCreator() {  
 return this.isCreator;  
 }  
  
 public void setCreator(boolean creator) {  
 isCreator = creator;  
 }  
  
 public boolean isReady1() {  
 return this.isReady1;  
 }  
  
 public void setReady1(boolean ready1) {  
 isReady1 = ready1;  
 }  
  
 public boolean isReady2() {  
 return this.isReady2;  
 }  
  
 public void setReady2(boolean ready2) {  
 isReady2 = ready2;  
 }  
  
 public boolean isStart1() {  
 return this.isStart1;  
 }  
  
 public void setStart1(boolean start1) {  
 this.isStart1 = start1;  
 }  
  
 public boolean isStart2() {  
 return this.isStart2;  
 }  
  
 public void setStart2(boolean start2) {  
 this.isStart2 = start2;  
 }  
  
 public boolean isRated() {  
 return this.isRated;  
 }  
  
 public void setRated(boolean rated) {  
 isRated = rated;  
 }  
  
 */\*\*  
 \* A function that gives my id.  
 \** ***@return*** *A String with my id  
 \*/* public String getMyId() {  
 if (this.isCreator)  
 return this.idP1;  
 return this.idP2;  
 }  
  
 */\*\*  
 \* A function that gives the enemy id.  
 \** ***@return*** *A String with the enemy id  
 \*/* public String getEnemyId() {  
 if (this.isCreator)  
 return this.idP2;  
 return this.idP1;  
 }  
  
 */\*\*  
 \* A function that creates a JSONObject from the data in  
 \* gameManagerBeforeStart to upload to server.  
 \** ***@return*** *A JSONObject of the gameManagerBeforeStart attributes  
 \*/* public JSONObject gameManagerBeforeStartToJson() {  
 JSONObject jsonObject = new JSONObject();  
 try {  
 jsonObject.put("gameId", this.gameId);  
 jsonObject.put("idP1", this.idP1);  
 jsonObject.put("idP2", this.idP2);  
 jsonObject.put("isCreator", this.isCreator);  
 jsonObject.put("isReady1", this.isReady1);  
 jsonObject.put("isReady2", this.isReady2);  
 jsonObject.put("isStart1", this.isStart1);  
 jsonObject.put("isStart2", this.isStart2);  
 jsonObject.put("isRated", this.isRated);  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
 return jsonObject;  
 }  
  
 */\*\*  
 \* A function that updates the values on the attributes  
 \** ***@param*** *jsonObject A JSONObject that is given from the server  
 \** ***@param*** *afterReady A Boolean which is true if the game started and false if not  
 \*/* public void jsonToGameManagerBeforeStart(JSONObject jsonObject, boolean afterReady) {  
 try {  
 if (!isCreator) {  
 this.idP1 = jsonObject.getString("idP1");  
 this.isRated = jsonObject.getBoolean("isRated");  
 if (!afterReady)  
 this.isReady1 = jsonObject.getBoolean("isReady1");  
 else  
 this.isStart1 = jsonObject.getBoolean("isStart1");  
 }  
 else {  
 this.idP2 = jsonObject.getString("idP2");  
 if (!afterReady)  
 this.isReady2 = jsonObject.getBoolean("isReady2");  
 else  
 this.isStart2 = jsonObject.getBoolean("isStart2");  
 }  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
 }  
  
}

RequestsAdapter

*/\*\*  
 \* RequestAdapter is a class that handles all http requests  
 \* and adds them to a requestQueue until they are executed.  
 \*/*package com.example.dominion\_game.classes;  
  
import android.content.Context;  
  
import com.android.volley.RequestQueue;  
import com.android.volley.toolbox.JsonObjectRequest;  
import com.android.volley.toolbox.Volley;  
  
public class RequestAdapter {  
 private RequestQueue requestQueue;  
 private static RequestAdapter *instance*;  
 private static Context *ctx*;  
  
 */\*\*  
 \* The constructor  
 \** ***@param*** *context A Context which is the application context  
 \*/* private RequestAdapter(Context context) {  
 *ctx* = context;  
 requestQueue = getRequestQueue();  
 }  
  
 */\*\*  
 \* A function that returns the RequestAdapter that handles the RequestQueue  
 \* and is synchronized because it can be called from different threads at the same time.  
 \** ***@param*** *context A Context which is the application context  
 \** ***@return*** *A RequestAdapter which is instance  
 \*/* public static synchronized RequestAdapter getInstance(Context context) {  
 if (*instance* == null) {  
 *instance* = new RequestAdapter(context);  
 }  
 return *instance*;  
 }  
  
 */\*\*  
 \* A function that creates the requestQueue if null and returns it.  
 \** ***@return*** *A RequestQueue which is requestQueue  
 \*/* public RequestQueue getRequestQueue() {  
 if (requestQueue == null) {  
 requestQueue = Volley.*newRequestQueue*(*ctx*.getApplicationContext());  
 }  
 return requestQueue;  
 }  
  
 */\*\*  
 \* A function that adds the specified request to the request queue.  
 \** ***@param*** *req  
 \*/* public void addToRequestQueue(JsonObjectRequest req) {  
 requestQueue.add(req);  
 }  
}

GameRequests

*/\*\*  
 \* GameRequests is a class that handles all game requests from the server.  
 \*/*package com.example.dominion\_game.classes;  
  
import android.util.Log;  
  
import com.android.volley.Request;  
import com.android.volley.Response;  
import com.android.volley.VolleyError;  
import com.android.volley.toolbox.JsonObjectRequest;  
import com.example.dominion\_game.activities.OnlineTablesActivity;  
import com.example.dominion\_game.activities.PrepareGameActivity;  
  
import org.json.JSONException;  
import org.json.JSONObject;  
  
public class GameRequests {  
  
 public static String getServerIP() {  
 return "192.168.1.181";  
 }  
  
 public static String getPort() {  
 return "8888";  
 }  
  
 */\*\*  
 \* A function that sends GET http request to create a table.  
 \** ***@param*** *prepareGameActivity A reference to prepareGameActivity  
 \*/* public static void creator\_start(final PrepareGameActivity prepareGameActivity)/\* throws IOException\*/ {  
 JsonObjectRequest jsonObjectRequest = new JsonObjectRequest(Request.Method.*GET*,  
 "http://" + *getServerIP*() + ":" + *getPort*() + "/creator\_start",  
 null,  
 new Response.Listener<JSONObject>() {  
 */\*\*  
 \* A function that handles the response from the server for the request.  
 \** ***@param*** *response A JSONObject with the key success which is true or false  
 \* and the unique key of the game  
 \*/* @Override  
 public void onResponse(JSONObject response) {  
 try {  
 if (response.getBoolean("success")) {  
 prepareGameActivity.getGameManagerBeforeStart().setGameId(response.getString("game\_id"));  
 *uploadGameManagerBeforeStart*(prepareGameActivity);  
 }  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
 }  
 }, new com.android.volley.Response.ErrorListener() {  
  
 @Override  
 public void onErrorResponse(VolleyError error) {  
  
 }  
 });  
 RequestAdapter.*getInstance*(prepareGameActivity).addToRequestQueue(jsonObjectRequest);  
 }  
  
 */\*\*  
 \* A function that sends POST http request with gameManagerBeforeStart to the server.  
 \* This function add the non creator player to the table.  
 \** ***@param*** *prepareGameActivity A reference to prepareGameActivity  
 \*/* public static void non\_creator\_start(final PrepareGameActivity prepareGameActivity) {  
 JsonObjectRequest jsonObjectRequest = new JsonObjectRequest(Request.Method.*POST*,  
 "http://" + *getServerIP*() + ":" + *getPort*() + "/non\_creator\_start",  
 prepareGameActivity.getGameManagerBeforeStart().gameManagerBeforeStartToJson(),  
 new Response.Listener<JSONObject>() {  
 */\*\*  
 \* A function that handles the response from the server for the request.  
 \** ***@param*** *response A JSONObject with the key success which is true or false  
 \*/* @Override  
 public void onResponse(JSONObject response) {  
 try {  
 if (!response.getBoolean("success"))  
 prepareGameActivity.prepareToLeave();  
 else  
 GameRequests.*waitForStartGame*(false, prepareGameActivity, 0);  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
 }  
 }, new com.android.volley.Response.ErrorListener() {  
 @Override  
 public void onErrorResponse(VolleyError error) {  
  
 }  
 });  
 RequestAdapter.*getInstance*(prepareGameActivity).addToRequestQueue(jsonObjectRequest);  
 }  
  
 */\*\*  
 \* A function that sends POST http request with my relevant data in game to the server  
 \* and gets the relevant data about the enemy.  
 \** ***@param*** *gameManager A reference to gameManager  
 \*/* public static void uploadAndGetPlayerData(final GameManager gameManager) {  
 JsonObjectRequest jsonObjectRequest = new JsonObjectRequest(Request.Method.*POST*,  
 "http://" + *getServerIP*() + ":" + *getPort*() + "/get\_and\_upload\_player\_data",  
 gameManager.myDataToJson(),  
 new Response.Listener<JSONObject>() {  
 */\*\*  
 \* A function that handles the response from the server for the request.  
 \** ***@param*** *response A JSONObject with the key success which is true or false  
 \* and the enemy data  
 \*/* @Override  
 public void onResponse(JSONObject response) {  
 try {  
 if (response.getBoolean("success")) {  
 gameManager.getEnemyData().jsonToEnemyData(response, gameManager);  
 gameManager.getGameActivity().updateCards(false);  
 }  
  
 if (gameManager.isGameEnded())  
 return;  
  
 if (gameManager.getEnemyData().isResigned()) {  
 gameManager.setGameEnded(true);  
 gameManager.getGameActivity().endGame();  
 *endGame*(gameManager);  
 return;  
 }  
  
 if (gameManager.isResigned()) {  
 gameManager.setGameEnded(true);  
 gameManager.getGameActivity().startPrepareActivity();  
 }  
 GameRequests.*uploadAndGetPlayerData*(gameManager);  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
 }  
 }, new com.android.volley.Response.ErrorListener() {  
 @Override  
 public void onErrorResponse(VolleyError error) {  
  
 }  
 });  
 RequestAdapter.*getInstance*(gameManager.getGameActivity()).addToRequestQueue(jsonObjectRequest);  
 }  
  
 */\*\*  
 \* A function that sends POST http request with the game data to the server.  
 \* When the game already started, it makes recursion calls while this is my turn  
 \* and uploads the game data real time.  
 \** ***@param*** *isStart A Boolean which is true if the game already started and false if not  
 \** ***@param*** *gameManager A reference to gameManager  
 \*/* public static void uploadDataInGame(final boolean isStart, final GameManager gameManager) {  
 JsonObjectRequest jsonObjectRequest = new JsonObjectRequest(Request.Method.*POST*,  
 "http://" + getServerIP() + ":" + getPort() + (isStart ? "/upload\_all\_data" : "/upload\_real\_time"),  
 isStart ? gameManager.gameManagerToJsonStart() : gameManager.gameManagerToJsonRealTime(),  
 new Response.Listener<JSONObject>() {  
 */\*\*  
 \* A function that handles the response from the server for the request.  
 \** ***@param*** *response A JSONObject with the key success which is true or false  
 \*/* @Override  
 public void onResponse(JSONObject response) {  
 try {  
 if (isStart) {  
 if (response.getBoolean("success")) {  
 GameRequests.waitForStartGame(gameManager);  
 GameRequests.uploadAndGetPlayerData(gameManager);  
 }  
 }  
 else {  
 if (!response.getBoolean("success")) {  
 if (gameManager.isGameEnded())  
 return;  
 GameRequests.uploadDataInGame(false, gameManager);  
 }  
 if (response.has("turn")) {  
 gameManager.jsonToGameManagerRealTime(response);  
 if (response.getJSONObject("turn").getString("lastActionCardForWait").equals("")) {  
 gameManager.getTurn().setWaitingForEnemy(false);  
 gameManager.getGameActivity().turnUI();  
 gameManager.endCard();  
 }  
  
 gameManager.getGameActivity().updateCards(true);  
 GameRequests.uploadDataInGame(false, gameManager);  
 return;  
 }  
 if (!gameManager.getTurn().isMyTurn(gameManager.getGameManagerBeforeStart())  
 && response.getString("turnId").equals(gameManager.getTurn().getTurnId())) {  
 gameManager.getGameActivity().endProgressBar(false);  
 if (gameManager.isGameEnded())  
 return;  
 gameManager.getGameActivity().turnActions();  
 GameRequests.getDataInGame(false, gameManager);  
 }  
 else {  
 if (gameManager.isGameEnded())  
 return;  
 gameManager.getGameActivity().updateCards(false);  
 GameRequests.uploadDataInGame(false, gameManager);  
 }  
 }  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
 }  
 }, new com.android.volley.Response.ErrorListener() {  
 @Override  
 public void onErrorResponse(VolleyError error) {  
  
 }  
 });  
 RequestAdapter.getInstance(gameManager.getGameActivity()).addToRequestQueue(jsonObjectRequest);  
 }  
  
 */\*\*  
 \* A function that sends POST http request with gameManagerBeforeStart to the server.  
 \* When the game already started, it makes recursion calls while this is not my turn  
 \* and gets the game data real time.  
 \** ***@param*** *isStart A Boolean which is true if the game already started and false if not  
 \** ***@param*** *gameManager A reference to gameManager  
 \*/* public static void getDataInGame(final boolean isStart, final GameManager gameManager)/\* throws IOException\*/ {  
 JsonObjectRequest jsonObjectRequest = new JsonObjectRequest(Request.Method.POST,  
 "http://" + getServerIP() + ":" + getPort() + (isStart ? "/get\_all\_data" : "/get\_real\_time"),  
 gameManager.isPlaysAttack() ? gameManager.gameManagerToJsonRealTime() :  
 gameManager.getGameManagerBeforeStart().gameManagerBeforeStartToJson(),  
 new Response.Listener<JSONObject>() {  
 */\*\*  
 \* A function that handles the response from the server for the request.  
 \** ***@param*** *response A JSONObject with the key success which is true or false  
 \* and the game data  
 \*/* @Override  
 public void onResponse(JSONObject response) {  
 try {  
 if (isStart) {  
 if (!response.getBoolean("success")) {  
 if (gameManager.isGameEnded())  
 return;  
 GameRequests.getDataInGame(true, gameManager);  
 }  
 else {  
 gameManager.jsonToGameManagerStart(response);  
 gameManager.getGameActivity().beforeStart();  
 GameRequests.waitForStartGame(gameManager);  
 GameRequests.uploadAndGetPlayerData(gameManager);  
 }  
 }  
 else {  
 if (!response.getBoolean("success")) {  
 if (gameManager.isGameEnded())  
 return;  
 GameRequests.getDataInGame(false, gameManager);  
 }  
 else {  
 gameManager.jsonToGameManagerRealTime(response);  
  
 if (!gameManager.getTurn().getLastActionCardForWait().equals("")) {  
 gameManager.setPlaysAttack(true);  
 Help.nameToCard(gameManager.getTurn().getLastActionCardForWait()).enemyPlay(gameManager, gameManager.getGameActivity());  
 if (gameManager.getPlayer().containsCard("Moat"))  
 Help.nameToCard("Moat").reaction(gameManager, gameManager.getGameActivity());  
 else {  
 Help.nameToCard(gameManager.getTurn().getLastActionCardForWait()).attack(gameManager, gameManager.getGameActivity());  
 gameManager.getTurn().removeLastAttack();  
 }  
 }  
  
 if (gameManager.getTurn().isMyTurn(gameManager.getGameManagerBeforeStart())) {  
 gameManager.setGameEnded(response.getBoolean("isGameEnded"));  
 gameManager.getGameActivity().endProgressBar(false);  
 if (gameManager.isGameEnded()) {  
 gameManager.getGameActivity().endGame();  
 endGame(gameManager);  
 return;  
 }  
 GameRequests.uploadDataInGame(false, gameManager);  
 gameManager.getGameActivity().turnActions();  
 gameManager.addTurnNumberToLog();  
 }  
 else {  
 if (gameManager.isGameEnded())  
 return;  
 gameManager.getGameActivity().updateCards(false);  
 GameRequests.getDataInGame(false, gameManager);  
 }  
 }  
 }  
 }  
 catch (JSONException e) {  
 e.printStackTrace();  
 }  
 }  
 }, new com.android.volley.Response.ErrorListener() {  
  
 @Override  
 public void onErrorResponse(VolleyError error) {  
  
 }  
 });  
 RequestAdapter.getInstance(gameManager.getGameActivity()).addToRequestQueue(jsonObjectRequest);  
 }  
  
 */\*\*  
 \* A function that sends GET http request to get all tables.  
 \* The function makes recursion calls while on onlineTablesActivity.  
 \** ***@param*** *onlineTablesActivity A reference to onlineTablesActivity  
 \*/* public static void getTables(final OnlineTablesActivity onlineTablesActivity) {  
 JsonObjectRequest jsonObjectRequest = new JsonObjectRequest(Request.Method.GET,  
 "http://" + getServerIP() + ":" + getPort() + "/get\_tables",  
 null,  
 new Response.Listener<JSONObject>() {  
 */\*\*  
 \* A function that handles the response from the server for the request.  
 \** ***@param*** *jsonObject A JSONObject with the key success which is true or false  
 \* and all tables with their ids  
 \*/* @Override  
 public void onResponse(JSONObject jsonObject) {  
 if (onlineTablesActivity.isFinished()) {  
 onlineTablesActivity.prepareToLeave();  
 return;  
 }  
 jsonObject.remove("success");  
 onlineTablesActivity.updateTables(jsonObject);  
 getTables(onlineTablesActivity);  
 }  
 },  
 new Response.ErrorListener() {  
 @Override  
 public void onErrorResponse(VolleyError error) {  
  
 }  
 });  
  
 RequestAdapter.getInstance(onlineTablesActivity).addToRequestQueue(jsonObjectRequest);  
 }  
  
 */\*\*  
 \* A function that sends POST http request with gameManagerBeforeStart to the server.  
 \** ***@param*** *prepareGameActivity A reference to prepareGameActivity  
 \*/* public static void uploadGameManagerBeforeStart(final PrepareGameActivity prepareGameActivity) {  
 JsonObjectRequest jsonObjectRequest = new JsonObjectRequest(Request.Method.POST,  
 "http://" + getServerIP() + ":" + getPort() + "/upload\_game\_manager\_before\_start",  
 prepareGameActivity.getGameManagerBeforeStart().gameManagerBeforeStartToJson(),  
 new Response.Listener<JSONObject>() {  
 */\*\*  
 \* A function that handles the response from the server for the request.  
 \** ***@param*** *response A JSONObject with the key success which is true or false  
 \*/* @Override  
 public void onResponse(JSONObject response) {  
 prepareGameActivity.endProgressBar();  
 waitForPlayerToEnterTable(prepareGameActivity);  
 }  
 }, new com.android.volley.Response.ErrorListener() {  
 @Override  
 public void onErrorResponse(VolleyError error) {  
  
 }  
 });  
 RequestAdapter.getInstance(prepareGameActivity).addToRequestQueue(jsonObjectRequest);  
 }  
  
 */\*\*  
 \* A function that sends POST http request with gameManagerBeforeStart to the server.  
 \* This function updates on server that the player is ready.  
 \** ***@param*** *prepareGameActivity A reference to prepareGameActivity  
 \*/* public static void updateReady(final PrepareGameActivity prepareGameActivity) {  
 JsonObjectRequest jsonObjectRequest = new JsonObjectRequest(Request.Method.POST,  
 "http://" + getServerIP() + ":" + getPort() + "/update\_ready",  
 prepareGameActivity.getGameManagerBeforeStart().gameManagerBeforeStartToJson(),  
 new Response.Listener<JSONObject>() {  
 */\*\*  
 \* A function that handles the response from the server for the request.  
 \** ***@param*** *response A JSONObject with the key success which is true or false  
 \*/* @Override  
 public void onResponse(JSONObject response) {  
 try {  
 if(!response.getBoolean("success"))  
 prepareGameActivity.prepareToLeave();  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
 }  
 }, new com.android.volley.Response.ErrorListener() {  
 @Override  
 public void onErrorResponse(VolleyError error) {  
  
 }  
 });  
 RequestAdapter.getInstance(prepareGameActivity).addToRequestQueue(jsonObjectRequest);  
 }  
  
 */\*\*  
 \* A function that sends POST http request with gameManagerBeforeStart to the server.  
 \* This function updates on server that the player is ready to start.  
 \** ***@param*** *gameManager A reference to gameManager  
 \*/* public static void updateReadyToStart(final GameManager gameManager) {  
 JsonObjectRequest jsonObjectRequest = new JsonObjectRequest(Request.Method.POST,  
 "http://" + getServerIP() + ":" + getPort() + "/update\_ready\_to\_start",  
 gameManager.getGameManagerBeforeStart().gameManagerBeforeStartToJson(),  
 new Response.Listener<JSONObject>() {  
 @Override  
 public void onResponse(JSONObject response) {  
 }  
 }, new com.android.volley.Response.ErrorListener() {  
 @Override  
 public void onErrorResponse(VolleyError error) {  
  
 }  
 });  
 RequestAdapter.getInstance(gameManager.getGameActivity()).addToRequestQueue(jsonObjectRequest);  
 }  
  
 */\*\*  
 \* A function that sends POST http request with gameManagerBeforeStart to the server.  
 \* The function makes recursion calls while a second player didn't enter the table.  
 \** ***@param*** *prepareGameActivity A reference to prepareGameActivity  
 \*/* public static void waitForPlayerToEnterTable(final PrepareGameActivity prepareGameActivity) {  
 JsonObjectRequest jsonObjectRequest = new JsonObjectRequest(Request.Method.POST,  
 "http://" + getServerIP() + ":" + getPort() + "/get\_game\_manager\_before\_start",  
 prepareGameActivity.getGameManagerBeforeStart().gameManagerBeforeStartToJson(),  
 new Response.Listener<JSONObject>() {  
 */\*\*  
 \* A function that handles the response from the server for the request.  
 \** ***@param*** *jsonObject A JSONObject with the key success which is true or false  
 \* and with gameManagerBeforeStart updated  
 \*/* @Override  
 public void onResponse(JSONObject jsonObject) {  
 try {  
 if (!jsonObject.getBoolean("success"))  
 prepareGameActivity.leaveTable();  
 else {  
 prepareGameActivity.getGameManagerBeforeStart().jsonToGameManagerBeforeStart(jsonObject, false);  
 prepareGameActivity.updateUI(true);  
 if (prepareGameActivity.getGameManagerBeforeStart().getIdP2().equals(""))  
 waitForPlayerToEnterTable(prepareGameActivity);  
 else {  
 waitForStartGame(true, prepareGameActivity, 0);  
 }  
 }  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
 }  
 },  
 new Response.ErrorListener() {  
 @Override  
 public void onErrorResponse(VolleyError error) {  
  
 }  
 });  
  
 RequestAdapter.getInstance(prepareGameActivity).addToRequestQueue(jsonObjectRequest);  
 }  
  
 */\*\*  
 \* A function that sends POST http request with gameManagerBeforeStart to the server.  
 \* The function makes recursion calls while there is a player that is not ready to start.  
 \** ***@param*** *gameManager A reference to gameManager  
 \*/* public static void waitForStartGame(final GameManager gameManager) {  
 JsonObjectRequest jsonObjectRequest = new JsonObjectRequest(Request.Method.POST,  
 "http://" + getServerIP() + ":" + getPort() + "/get\_game\_manager\_before\_start",  
 gameManager.getGameManagerBeforeStart().gameManagerBeforeStartToJson(),  
 new Response.Listener<JSONObject>() {  
 */\*\*  
 \* A function that handles the response from the server for the request.  
 \** ***@param*** *jsonObject A JSONObject with the key success which is true or false  
 \* and with gameManagerBeforeStart updated  
 \*/* @Override  
 public void onResponse(JSONObject jsonObject) {  
 gameManager.getGameManagerBeforeStart().jsonToGameManagerBeforeStart(jsonObject, true);  
 if (!gameManager.getEnemyData().isEmpty()) {  
 gameManager.getGameActivity().endProgressBar(true);  
 }  
 if (gameManager.getGameManagerBeforeStart().isStart1() &&  
 gameManager.getGameManagerBeforeStart().isStart2()) {  
 gameManager.getGameActivity().startGame();  
 }  
 else  
 GameRequests.waitForStartGame(gameManager);  
 }  
 },  
 new Response.ErrorListener() {  
 @Override  
 public void onErrorResponse(VolleyError error) {  
  
 }  
 });  
  
 RequestAdapter.getInstance(gameManager.getGameActivity()).addToRequestQueue(jsonObjectRequest);  
 }  
  
 */\*\*  
 \* A function that sends POST http request with gameManagerBeforeStart to the server.  
 \* The function makes recursion calls while there is a player that is not ready.  
 \** ***@param*** *isCreator A Boolean which is true if the player is the creator and false if not  
 \** ***@param*** *prepareGameActivity A reference to prepareGameActivity  
 \** ***@param*** *countCheck An Integer that ensures that both players are ready and count  
 \* to 5 until the game starts  
 \*/* public static void waitForStartGame(final boolean isCreator, final PrepareGameActivity prepareGameActivity, final int countCheck) {  
 JsonObjectRequest jsonObjectRequest = new JsonObjectRequest(Request.Method.POST,  
 "http://" + getServerIP() + ":" + getPort() + "/get\_game\_manager\_before\_start",  
 prepareGameActivity.getGameManagerBeforeStart().gameManagerBeforeStartToJson(),  
 new Response.Listener<JSONObject>() {  
 */\*\*  
 \* A function that handles the response from the server for the request.  
 \** ***@param*** *jsonObject A JSONObject with the key success which is true or false  
 \* and with gameManagerBeforeStart updated  
 \*/* @Override  
 public void onResponse(JSONObject jsonObject) {  
 try {  
 if (!jsonObject.getBoolean("success"))  
 prepareGameActivity.prepareToLeave();  
 else {  
 prepareGameActivity.endProgressBar();  
 prepareGameActivity.getGameManagerBeforeStart().jsonToGameManagerBeforeStart(jsonObject, false);  
 prepareGameActivity.updateUI(false);  
 if (isCreator && jsonObject.getString("idP2").equals(""))  
 waitForPlayerToEnterTable(prepareGameActivity);  
 else {  
 if (!prepareGameActivity.getGameManagerBeforeStart().getIdP1().equals("") &&  
 !prepareGameActivity.getGameManagerBeforeStart().getIdP2().equals("") &&  
 prepareGameActivity.getGameManagerBeforeStart().isReady1() &&  
 prepareGameActivity.getGameManagerBeforeStart().isReady2()) // checks if both players are ready  
 if (countCheck == 5)  
 prepareGameActivity.startGame();  
 else if (countCheck < 5)  
 waitForStartGame(isCreator, prepareGameActivity, countCheck + 1);  
 else  
 Log.d("error", String.valueOf(countCheck));  
 else  
 waitForStartGame(isCreator, prepareGameActivity, 0);  
 }  
 }  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
 }  
 },  
 new Response.ErrorListener() {  
 @Override  
 public void onErrorResponse(VolleyError error) {  
  
 }  
 });  
  
 RequestAdapter.getInstance(prepareGameActivity).addToRequestQueue(jsonObjectRequest);  
 }  
  
 */\*\*  
 \* A function that sends POST http request with gameManagerBeforeStart  
 \* to the server and deletes the table.  
 \* This function is for the creator of the game.  
 \** ***@param*** *prepareGameActivity A reference to prepareGameActivity  
 \*/* public static void delete\_game\_manager\_before\_start(final PrepareGameActivity prepareGameActivity) {  
 JsonObjectRequest jsonObjectRequest = new JsonObjectRequest(Request.Method.POST,  
 "http://" + getServerIP() + ":" + getPort() + "/delete\_game\_manager\_before\_start",  
 prepareGameActivity.getGameManagerBeforeStart().gameManagerBeforeStartToJson(),  
 new Response.Listener<JSONObject>() {  
 */\*\*  
 \* A function that handles the response from the server for the request.  
 \** ***@param*** *response A JSONObject with the key success which is true or false  
 \*/* @Override  
 public void onResponse(JSONObject response) {  
 try {  
 if (response.getBoolean("success")) {  
 prepareGameActivity.endProgressBar();  
 prepareGameActivity.leaveTable();  
 }  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
 }  
 }, new com.android.volley.Response.ErrorListener() {  
 @Override  
 public void onErrorResponse(VolleyError error) {  
  
 }  
 });  
 RequestAdapter.getInstance(prepareGameActivity).addToRequestQueue(jsonObjectRequest);  
 }  
  
 */\*\*  
 \* A function that sends POST http request with gameManagerBeforeStart  
 \* to the server and deletes the player from the table.  
 \* This function is for the non creator of the game.  
 \** ***@param*** *prepareGameActivity A reference to prepareGameActivity  
 \*/* public static void deleteP2(final PrepareGameActivity prepareGameActivity) {  
 JsonObjectRequest jsonObjectRequest = new JsonObjectRequest(Request.Method.POST,  
 "http://" + getServerIP() + ":" + getPort() + "/delete\_p2",  
 prepareGameActivity.getGameManagerBeforeStart().gameManagerBeforeStartToJson(),  
 new Response.Listener<JSONObject>() {  
 */\*\*  
 \* A function that handles the response from the server for the request.  
 \** ***@param*** *response A JSONObject with the key success which is true or false  
 \*/* @Override  
 public void onResponse(JSONObject response) {  
 try {  
 if(response.getBoolean("success")) {  
 prepareGameActivity.endProgressBar();  
 prepareGameActivity.leaveTable();  
 }  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
 }  
 }, new com.android.volley.Response.ErrorListener() {  
 @Override  
 public void onErrorResponse(VolleyError error) {  
  
 }  
 });  
 RequestAdapter.getInstance(prepareGameActivity).addToRequestQueue(jsonObjectRequest);  
 }  
  
 */\*\*  
 \* A function that sends POST http request with gameManagerBeforeStart to the server.  
 \** ***@param*** *gameManager A reference to gameManager  
 \*/* public static void endGame(final GameManager gameManager) {  
 JsonObjectRequest jsonObjectRequest = new JsonObjectRequest(Request.Method.POST,  
 "http://" + getServerIP() + ":" + getPort() + "/end\_game",  
 gameManager.getGameManagerBeforeStart().gameManagerBeforeStartToJson(),  
 new Response.Listener<JSONObject>() {  
 */\*\*  
 \* A function that handles the response from the server for the request.  
 \** ***@param*** *response A JSONObject with the key success which is true or false  
 \*/* @Override  
 public void onResponse(JSONObject response) {  
 try {  
 if(response.getBoolean("success")) {  
 gameManager.getGameActivity().startPrepareActivity();  
 }  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
 }  
 }, new com.android.volley.Response.ErrorListener() {  
 @Override  
 public void onErrorResponse(VolleyError error) {  
  
 }  
 });  
 RequestAdapter.getInstance(gameManager.getGameActivity()).addToRequestQueue(jsonObjectRequest);  
 }  
}

LoginRequests

*/\*\*  
 \* LoginRequests is a class that handles all login and register requests from the server.  
 \*/*package com.example.dominion\_game.classes;  
  
import android.widget.Toast;  
  
import com.android.volley.Request;  
import com.android.volley.Response;  
import com.android.volley.VolleyError;  
import com.android.volley.toolbox.JsonObjectRequest;  
import com.example.dominion\_game.activities.LoginActivity;  
import com.example.dominion\_game.activities.RegisterActivity;  
  
import org.json.JSONException;  
import org.json.JSONObject;  
  
public class LoginRequests {  
 public static String getServerIP() {  
 return "192.168.1.181";  
 }  
  
 public static String getPort() {  
 return "8888";  
 }  
  
 */\*\*  
 \* A function that sends POST http request with registerRequest to the server.  
 \** ***@param*** *registerActivity A reference to registerActivity  
 \** ***@param*** *registerRequest A JSONObject of the register details  
 \*/* public static void register(final RegisterActivity registerActivity, JSONObject registerRequest) {  
 JsonObjectRequest jsonObjectRequest = new JsonObjectRequest(Request.Method.*POST*,  
 "http://" + *getServerIP*() + ":" + *getPort*() + "/register",  
 registerRequest,  
 new Response.Listener<JSONObject>() {  
 */\*\*  
 \* A function that handles the response from the server for the registerRequest.  
 \** ***@param*** *response A JSONObject with the key success which is true or false  
 \*/* @Override  
 public void onResponse(JSONObject response) {  
 registerActivity.endProgressBar();  
 try {  
 if (!response.getBoolean("success"))  
 Toast.*makeText*(registerActivity, "There is already a user with this name", Toast.*LENGTH\_SHORT*).show();  
 else  
 registerActivity.registerSuccess(response.getString("username"));  
  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
 }  
 }, new com.android.volley.Response.ErrorListener() {  
 @Override  
 public void onErrorResponse(VolleyError error) {  
  
 }  
 });  
 RequestAdapter.*getInstance*(registerActivity).addToRequestQueue(jsonObjectRequest);  
 }  
  
 */\*\*  
 \* A function that sends POST http request with loginRequest to the server.  
 \** ***@param*** *loginActivity A reference to loginActivity  
 \** ***@param*** *loginRequest A JSONObject of the login details  
 \*/* public static void login(final LoginActivity loginActivity, JSONObject loginRequest) {  
 JsonObjectRequest jsonObjectRequest = new JsonObjectRequest(Request.Method.*POST*,  
 "http://" + *getServerIP*() + ":" + *getPort*() + "/login",  
 loginRequest,  
 new Response.Listener<JSONObject>() {  
 */\*\*  
 \* A function that handles the response from the server for the loginRequest.  
 \** ***@param*** *response A JSONObject with the key success which is true or false  
 \*/* @Override  
 public void onResponse(JSONObject response) {  
 loginActivity.endProgressBar();  
 try {  
 if (!response.getBoolean("success"))  
 Toast.*makeText*(loginActivity, "Wrong username or password", Toast.*LENGTH\_SHORT*).show();  
 else  
 loginActivity.loginSuccess(response.getString("username"));  
  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
 }  
 }, new com.android.volley.Response.ErrorListener() {  
 @Override  
 public void onErrorResponse(VolleyError error) {  
  
 }  
 });  
 RequestAdapter.*getInstance*(loginActivity).addToRequestQueue(jsonObjectRequest);  
 }  
}

Help

*/\*\*  
 \* Help is a class with static function in it that are general  
 \* and could be used in any class as a help.  
 \*/*package com.example.dominion\_game.classes;  
  
import android.util.Pair;  
  
import com.example.dominion\_game.cards\_classes.\*;  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.Collections;  
import java.util.Comparator;  
import java.util.HashMap;  
import java.util.HashSet;  
import java.util.Random;  
import java.util.Set;  
  
public class Help {  
  
 */\*\*  
 \* A function that convert HashMap to an ArrayList.  
 \** ***@param*** *hm A HashMap of card names with the amount of it  
 \** ***@return*** *An ArrayList of all cards in hm  
 \*/* public static ArrayList<String> hashToArray(HashMap<String, Integer> hm) {  
 ArrayList<String> al = new ArrayList<>();  
 for (String cardName : hm.keySet())  
 for (int i = 0; i < hm.get(cardName); i++)  
 al.add(cardName);  
  
 return al;  
 }  
  
 */\*\*  
 \* A function that returns the size of HashMap according to the values.  
 \** ***@param*** *hm A HashMap of card names with the amount of it  
 \** ***@return*** *An Integer with the amount of cards in hm  
 \*/* public static int sizeOfHash(HashMap<String, Integer> hm) {  
 int count = 0;  
 for (String cardName : hm.keySet())  
 count += hm.get(cardName);  
  
 return count;  
 }  
  
 public static ArrayList<String> arrayListOfPairsToArrayList(ArrayList<Pair<String, Boolean>> alPairs) {  
 ArrayList<String> al = new ArrayList<>();  
 for (Pair<String, Boolean> pair : alPairs)  
 al.add(pair.first);  
  
 return al;  
 }  
  
 */\*\*  
 \* A function that takes the name of a card and returns the class of this specific card.  
 \* This function is used for calling functions in this class, for example: play.  
 \** ***@param*** *name A String with the name of a card  
 \** ***@return*** *A Class that extends from Card - a specific card  
 \*/* public static Card nameToCard(String name) {  
 switch (name) {  
 case "Artisan":  
 return new Artisan();  
 case "Bandit":  
 return new Bandit();  
 case "Bureaucrat":  
 return new Bureaucrat();  
 case "Cellar":  
 return new Cellar();  
 case "Chapel":  
 return new Chapel();  
 case "Copper":  
 return new Copper();  
 case "CouncilRoom":  
 return new CouncilRoom();  
 case "Festival":  
 return new Festival();  
 case "Gardens":  
 return new Gardens();  
 case "Gold":  
 return new Gold();  
 case "Harbinger":  
 return new Harbinger();  
 case "Laboratory":  
 return new Laboratory();  
 case "Library":  
 return new Library();  
 case "Market":  
 return new Market();  
 case "Merchant":  
 return new Merchant();  
 case "Militia":  
 return new Militia();  
 case "Mine":  
 return new Mine();  
 case "Moat":  
 return new Moat();  
 case "Moneylender":  
 return new Moneylender();  
 case "Poacher":  
 return new Poacher();  
 case "Remodel":  
 return new Remodel();  
 case "Sentry":  
 return new Sentry();  
 case "Silver":  
 return new Silver();  
 case "Smithy":  
 return new Smithy();  
 case "ThroneRoom":  
 return new ThroneRoom();  
 case "Vassal":  
 return new Vassal();  
 case "Estate":  
 return new Victory("Estate");  
 case "Duchy":  
 return new Victory("Duchy");  
 case "Province":  
 return new Victory("Province");  
 case "Curse":  
 return new Victory("Curse");  
 case "Village":  
 return new Village();  
 case "Witch":  
 return new Witch();  
 case "Workshop":  
 return new Workshop();  
 }  
 return null;  
 }  
  
 */\*\*  
 \* A function that returns all the cards of the expansions.  
 \** ***@param*** *expansion A String with the specific expansion  
 \** ***@return*** *A Set of all cards of the specific expansion  
 \*/* public static Set<String> getExpansionCards(String expansion) {  
 switch (expansion) {  
 case "Base":  
 return *base*();  
 }  
 return new HashSet<>();  
 }  
  
 */\*\*  
 \* A function that returns all the cards of the base action cards game.  
 \** ***@return*** *A set of all the cards of the base action cards game  
 \*/* public static Set<String> base() {  
 return new HashSet<>(Arrays.*asList*(  
 "Artisan",  
 "Bandit",  
 "Bureaucrat",  
 "Cellar",  
 "Chapel",  
 "CouncilRoom",  
 "Festival",  
 "Gardens",  
 "Harbinger",  
 "Laboratory",  
 "Library",  
 "Market",  
 "Merchant",  
 "Militia",  
 "Mine",  
 "Moat",  
 "Moneylender",  
 "Poacher",  
 "Remodel",  
 "Sentry",  
 "Smithy",  
 "ThroneRoom",  
 "Vassal",  
 "Village",  
 "Witch",  
 "Workshop"));  
 }  
  
 */\*\*  
 \* A function that randomizes all cards of the specific expansions  
 \* and returns the set of cards sorted for a new game.  
 \** ***@param*** *expansions A String Array with the expansions that can be included in tis game  
 \** ***@param*** *numberOfCards An Integer with the number of cards that should be in this set  
 \** ***@param*** *cardsToInsert A String Array with specific cards that should be in this game  
 \** ***@return*** *A String Array with randomized sorted cards for a new game  
 \*/* public static String[] getRandomCards(String[] expansions, int numberOfCards, String[] cardsToInsert) {  
 Set<String> allCards = new HashSet<>();  
 for (String expansion : expansions)  
 allCards.addAll(*getExpansionCards*(expansion));  
  
 allCards.removeAll(Arrays.*asList*(cardsToInsert));  
 ArrayList<String> allCardsArray = new ArrayList<>(Arrays.*asList*(allCards.toArray(new String[0])));  
  
 String[] cards = new String[numberOfCards];  
 Random random = new Random();  
 for (int i = 0; i < cards.length; i++)  
 if (i < cardsToInsert.length)  
 cards[i] = cardsToInsert[i];  
 else  
 cards[i] = allCardsArray.remove(random.nextInt(allCardsArray.size()));  
  
 return Help.*sort*(cards);  
 }  
  
 */\*\*  
 \* A function that sorts the cards by price and by name.  
 \** ***@param*** *cards A String Array with the cards that are in this game  
 \** ***@return*** *A String Array with the cards that are in this game sorted  
 \*/* public static String[] sort(String[] cards) {  
 ArrayList<String> arrayCards = new ArrayList<>(Arrays.*asList*(cards));  
 Collections.*sort*(arrayCards, new Comparator<String>() {  
 */\*\*  
 \* A function that compares between two values in the array of cards  
 \* and returns which is first by price and name.  
 \** ***@param*** *s1 A String 1  
 \** ***@param*** *s2 A String 2  
 \** ***@return*** *An Integer that is zero if the objects are equal,  
 \* a positive value if s1 is greater than s2  
 \* and a negative value otherwise.  
 \*/* @Override  
 public int compare(String s1, String s2) {  
 int first = Integer.*valueOf*(Help.*nameToCard*(s1).getPrice()).compareTo(Help.*nameToCard*(s2).getPrice()); // compared by price  
  
 if (first != 0) // if the cards don't have the same price  
 return first;  
  
 return s1.compareTo(s2); // compared by name  
 }  
 });  
 return arrayCards.toArray(new String[0]);  
 }  
}

MusicService

*/\*\*  
 \* MusicService is a class extends Service which plays and stops music.  
 \*/*package com.example.dominion\_game.classes;  
  
import android.app.Service;  
import android.content.Intent;  
import android.media.MediaPlayer;  
import android.os.IBinder;  
  
import com.example.dominion\_game.R;  
  
  
public class MusicService extends Service {  
  
 private MediaPlayer mediaPlayer;  
 public MusicService() {  
  
 }  
  
 */\*\*  
 \* A function that starts the service.  
 \** ***@param*** *intent  
 \** ***@param*** *flags  
 \** ***@param*** *startId  
 \** ***@return*** *START\_STICKY  
 \*/* @Override  
 public int onStartCommand(Intent intent, int flags, int startId) {  
 //getting systems default ringtone  
 mediaPlayer = MediaPlayer.*create*(this, R.raw.*super\_smash\_bros\_music*);  
 //setting loop play to true  
 //this will make the ringtone continuously playing  
 mediaPlayer.setLooping(true);  
  
 //staring the player  
 mediaPlayer.start();  
  
 //we have some options for service  
 //start sticky means service will be explicity started and stopped  
 return *START\_STICKY*;  
 }  
  
 @Override  
 public IBinder onBind(Intent intent) {  
 return null;  
 }  
  
 */\*\*  
 \* A function that stops the player.  
 \*/* @Override  
 public void onDestroy() {  
 super.onDestroy();  
 //stopping the player when service is destroyed  
 mediaPlayer.stop();  
 mediaPlayer.release();  
 }  
}

PhoneCallReceiver

*/\*\*  
 \* PhoneCallReceiver is a broadcast receiver that listens to call state changes.  
 \* When the phone is ringing, it automatically hangs up and sends SMS to this number phone.  
 \*/*package com.example.dominion\_game.classes;  
  
import android.content.BroadcastReceiver;  
import android.content.Context;  
import android.content.Intent;  
import android.telecom.TelecomManager;  
import android.telephony.PhoneStateListener;  
import android.telephony.SmsManager;  
import android.telephony.TelephonyManager;  
import android.widget.Toast;  
  
import static android.content.Context.*TELEPHONY\_SERVICE*;  
  
public class PhoneCallReceiver extends BroadcastReceiver {  
  
 */\*\*  
 \* A function that is called when the BroadcastReceiver is receiving an Intent broadcast.  
 \** ***@param*** *context The Context in which the receiver is running (GameActivity)  
 \** ***@param*** *intent The Intent being received.  
 \*/* @Override  
 public void onReceive(final Context context, Intent intent) {  
 TelephonyManager manager = (TelephonyManager) context.getSystemService(*TELEPHONY\_SERVICE*);  
 manager.listen(new PhoneStateListener() {  
 */\*\*  
 \* A function that is called when there is a change in the call state.  
 \** ***@param*** *state The state code that changed to, for example 1 is CALL\_STATE\_RINGING  
 \** ***@param*** *incomingNumber A String with the phone number  
 \*/* @Override  
 public void onCallStateChanged(int state, String incomingNumber) {  
 super.onCallStateChanged(state, incomingNumber);  
 if (state == TelephonyManager.*CALL\_STATE\_RINGING*) {  
 // ends call only if the the SDK version is LOLLIPOP or above  
 if (android.os.Build.VERSION.*SDK\_INT* >= android.os.Build.VERSION\_CODES.*LOLLIPOP*) {  
 TelecomManager tm = (TelecomManager) context.getSystemService(Context.*TELECOM\_SERVICE*);  
 if (tm != null)  
 tm.endCall();  
 }  
 // creates a SmsManager that sends the message  
 SmsManager smsManager = SmsManager.*getDefault*();  
 smsManager.sendTextMessage(incomingNumber, null,  
 "Sorry, I will call you back later because I am playing Dominion right now.", null, null);  
 Toast.*makeText*(context, "Message Sent to " + incomingNumber, Toast.*LENGTH\_LONG*).show();  
 }  
  
 }  
 }, PhoneStateListener.*LISTEN\_CALL\_STATE*);  
 }  
}

Table

*/\*\*  
 \* Table is a class that keeps all the relevant data for every table in OnlineTablesActivity  
 \*/*package com.example.dominion\_game.classes;  
  
public class Table {  
 private String creator;  
 private String id;  
  
 */\*\*  
 \* The constructor  
 \*/* public Table(String creator, String id) {  
 this.creator = creator;  
 this.id = id;  
 }  
  
 public String getCreator() {  
 return creator;  
 }  
  
 public void setCreator(String creator) {  
 this.creator = creator;  
 }  
  
 public String getId() {  
 return this.id;  
 }  
  
 public void setId(String id) {  
 this.id = id;  
 }  
  
}

Card

*/\*\*  
 \* Card is an abstract class which is the base of all card classes.  
 \* This class has an abstract function: "play" that has to be overrided for all cards.  
 \* This class has also many functions that are not abstract but are also  
 \* overrided for some special cards, and here these functions are empty.  
 \*/*package com.example.dominion\_game.classes;  
  
import com.example.dominion\_game.activities.GameActivity;  
  
public abstract class Card {  
 private String type;  
 private String name;  
 private int price;  
 private int imageSource;  
 private int shortImageSource;  
  
 */\*\*  
 \* The constructor  
 \** ***@param*** *name A String with the name of the card  
 \** ***@param*** *price An Integer with the price of the card  
 \** ***@param*** *imageSource An Integer with the image resource of the card  
 \** ***@param*** *shortImageSource An Integer with the short image resource of the card  
 \** ***@param*** *type A String with the type of the card (action, treasure or victory)  
 \*/* public Card(String name, int price, int imageSource, int shortImageSource, String type) {  
 this.name = name;  
 this.price = price;  
 this.imageSource = imageSource;  
 this.shortImageSource = shortImageSource;  
 this.type = type;  
 }  
  
 */\*\*  
 \* The constructor with default values  
 \** ***@param*** *type  
 \*/* public Card(String type) {  
 this.name = "";  
 this.price = 0;  
 this.imageSource = 0;  
 this.shortImageSource = 0;  
 this.type = type;  
 }  
  
 public int getImageSource() {  
 return this.imageSource;  
 }  
  
 public void setImageSource(int imageSource) {  
 this.imageSource = imageSource;  
 }  
  
 public int getShortImageSource() {  
 return this.shortImageSource;  
 }  
  
 public void setShortImageSource(int shortImageSource) {  
 this.shortImageSource = shortImageSource;  
 }  
  
 public int getPrice() {  
 return this.price;  
 }  
  
 public String getName() {  
 return this.name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public void setPrice(int price) {  
 this.price = price;  
 }  
  
 public String getType() {  
 return this.type;  
 }  
  
 public void setType(String type) {  
 this.type = type;  
 }  
  
 */\*\*  
 \* An abstract function that plays the card.  
 \** ***@param*** *game A reference to gameManager  
 \** ***@param*** *gameActivity A reference to gameActivity  
 \*/* public abstract void play(GameManager game, GameActivity gameActivity);  
  
 */\*\*  
 \* A function that plays the attack of the card for the enemy.  
 \** ***@param*** *game A reference to gameManager  
 \** ***@param*** *gameActivity A reference to gameActivity  
 \*/* public void attack(GameManager game, GameActivity gameActivity) {  
 game.setDoneAttack(true);  
 }  
  
 */\*\*  
 \* A function that plays the reaction of the card for the enemy.  
 \** ***@param*** *game A reference to gameManager  
 \** ***@param*** *gameActivity A reference to gameActivity  
 \*/* public void reaction(GameManager game, GameActivity gameActivity) {  
  
 }  
  
 */\*\*  
 \* A function that plays an action of the card for the enemy.  
 \** ***@param*** *game A reference to gameManager  
 \** ***@param*** *gameActivity A reference to gameActivity  
 \*/* public void enemyPlay(GameManager game, GameActivity gameActivity) {  
 }  
  
 */\*\*  
 \* A function that handles clicking on a button that was enabled after playing this card.  
 \** ***@param*** *buttonText A String with the text on the button  
 \** ***@param*** *game A reference to gameManager  
 \** ***@param*** *gameActivity A reference to gameActivity  
 \*/* public void handleButtonClicks(String buttonText, GameManager game, GameActivity gameActivity) {  
 }  
  
 */\*\*  
 \* A function that handles clicks on hand or on dialog after playing a card  
 \* that should wait for clicking on hand or on dialog.  
 \** ***@param*** *cardName A String which is the name of the card which  
 \* was clicked on hand or on dialog  
 \** ***@param*** *game A reference to gameManager  
 \** ***@param*** *gameActivity A reference to gameActivity  
 \*/* public void handleClickOnHandOrDialog(String cardName, GameManager game, GameActivity gameActivity) {  
 }  
  
 */\*\*  
 \** ***@return*** *A Boolean which is true if cards selected from hand should be  
 \* marked or not after playing that card and false if not  
 \*/* public boolean isMarkCardSelectedFromHandWhenHandle() {  
 return false;  
 }  
  
 */\*\*  
 \* A function that returns whether the card can be used after playing this card.  
 \** ***@param*** *cardName A String which is the name of the card which was pressed  
 \** ***@param*** *game A reference to gameManager  
 \** ***@param*** *gameActivity A reference to gameActivity  
 \** ***@return*** *A Boolean which is true if the card can be used after playing  
 \* this card and false if not  
 \*/* public boolean isCardToUse(String cardName, GameManager game, GameActivity gameActivity) {  
 return false;  
 }  
  
 */\*\*  
 \* A function that returns whether the card can be bought after playing this card.  
 \** ***@param*** *cardName A String which is the name of the card which was pressed  
 \** ***@param*** *game A reference to gameManager  
 \** ***@param*** *gameActivity A reference to gameActivity  
 \** ***@return*** *A Boolean which is true if the card can be bought after playing  
 \* this card and false if not  
 \*/* public boolean isCardToGetFromBoard(String cardName, GameManager game, GameActivity gameActivity) {  
 return false;  
 }  
  
 */\*\*  
 \* A function that that handles click on a card from board after playing this card.  
 \** ***@param*** *cardName A String which is the name of the card which was clicked  
 \** ***@param*** *game A reference to gameManager  
 \** ***@param*** *gameActivity A reference to gameActivity  
 \*/* public void clickOnBoard(String cardName, GameManager game, GameActivity gameActivity) {  
 }  
  
 */\*\*  
 \* A function that returns the value of the card.  
 \** ***@param*** *gameManager A reference to gameManager  
 \** ***@return*** *An Integer with the value of the card  
 \*/* public int getValue(GameManager gameManager) {  
 return 0;  
 }  
  
 */\*\*  
 \* A function that is overrided if the name of the card is more than one word.  
 \** ***@return*** *A String with the the card name for display  
 \*/* public String getNameToDisplay() {  
 return this.name;  
 }  
  
 */\*\*  
 \* A function that plays the after play for the card (when a card used after played this card).  
 \** ***@param*** *game A reference to gameManager  
 \** ***@param*** *cardNameUsed A String with the name of card used  
 \*/* public void afterPlay(GameManager game, String cardNameUsed) {}  
  
 @Override  
 public String toString() {  
 return this.getNameToDisplay();  
 }  
}

**Card Classes:**

Artisan

package com.example.dominion\_game.cards\_classes;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.activities.GameActivity;  
import com.example.dominion\_game.classes.Card;  
import com.example.dominion\_game.classes.GameManager;  
import com.example.dominion\_game.classes.Help;  
  
public class Artisan extends Card {  
 public Artisan() {  
 super("Artisan", 6, R.mipmap.*artisan*, R.mipmap.*artisan\_sh*, "action");  
 }  
  
 @Override  
 public void play(GameManager game, GameActivity gameActivity) {  
 gameActivity.waitForBoard(this.getName(), 1, 1);  
 }  
  
 @Override  
 public void clickOnBoard(String cardName, GameManager game, GameActivity gameActivity) {  
 game.getPlayer().addToHand(game.getCard(cardName));  
 game.getTurn().getWaitForFunction().clear();  
 gameActivity.updateCards(false);  
  
 gameActivity.waitForHand(this.getName(), 1, 1, "topdeck", false);  
 gameActivity.uploadActionButtons(new String[]{gameActivity.getString(R.string.*undo*), gameActivity.getString(R.string.*confirm\_top\_deck*)}, true);  
 gameActivity.turnUI();  
 game.getPlayer().updateArrayHand();  
 gameActivity.getHandAdapter().notifyDataSetChanged();  
 }  
  
 @Override  
 public boolean isCardToGetFromBoard(String cardName, GameManager game, GameActivity gameActivity) {  
 return Help.*nameToCard*(cardName).getPrice() <= 5;  
 }  
  
 @Override  
 public void handleButtonClicks(String buttonText, GameManager game, GameActivity gameActivity) {  
 if (buttonText.equals(gameActivity.getString(R.string.*undo*))) {  
 game.getTurn().getWaitForFunction().undo();  
 gameActivity.updateActionButtons();  
 gameActivity.getHandAdapter().notifyDataSetChanged();  
 }  
 else if (buttonText.equals(gameActivity.getString(R.string.*confirm\_top\_deck*))) {  
 String cardName = String.*valueOf*(game.getTurn().getWaitForFunction().getCardsForActionCardPlay().keySet().toArray()[0]);  
 game.getPlayer().handToDeck(cardName, gameActivity);  
 game.getTurn().getWaitForFunction().clear();  
 gameActivity.updateActionButtons();  
 game.getPlayer().updateArrayHand();  
 gameActivity.getHandAdapter().notifyDataSetChanged();  
 game.useAfterPlay(this.getName(), false);  
 }  
 }  
  
 @Override  
 public boolean isCardToUse(String cardName, GameManager game, GameActivity gameActivity) {  
 return true;  
 }  
}

Bandit

package com.example.dominion\_game.cards\_classes;  
  
import android.view.View;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.activities.GameActivity;  
import com.example.dominion\_game.classes.Card;  
import com.example.dominion\_game.classes.GameManager;  
import com.example.dominion\_game.classes.Help;  
  
import java.util.ArrayList;  
  
public class Bandit extends Card {  
 public Bandit() {  
 super("Bandit", 5, R.mipmap.*bandit*, R.mipmap.*bandit\_sh*, "action");  
 }  
 @Override  
 public void play(GameManager game, GameActivity gameActivity) {  
 game.getPlayer().addToDiscard(game.getCard("Gold"));  
 game.useAfterPlay(this.getName(), true);  
 }  
  
 @Override  
 public void attack(GameManager game, GameActivity gameActivity) {  
 ArrayList<String> cards = game.getPlayer().takeCards(2, game, game.getLastLineFromLog().getTabs());  
 if (cards.size() == 0) {  
 game.setDoneAttack(true);  
 return;  
 }  
 game.getTurn().getWaitForFunction().handleWaitingForActionCardsDialog(this.getName(), 1, 1, "", false, cards);  
 gameActivity.getActionCardsPlayingAdapter().notifyDataSetChanged();  
 gameActivity.setVisibilityForRVActionCardsPlaying(View.*VISIBLE*);  
 boolean canRemove1 = Help.*nameToCard*(cards.get(0)).getType().equals("treasure") && !cards.get(0).equals("Copper");  
 boolean canRemove2 = false;  
 if (cards.size() > 1)  
 canRemove2 = Help.*nameToCard*(cards.get(1)).getType().equals("treasure") && !cards.get(1).equals("Copper");  
  
 if (!(canRemove1 && canRemove2) || cards.get(0).equals(cards.get(1))) {  
 if (canRemove1 || canRemove2) {  
 game.addToTrash(cards.get(canRemove1 ? 0 : 1), 1);  
 cards.remove(canRemove1 ? 0 : 1);  
 game.getPlayer().putArrayInDiscard(cards);  
 }  
 game.getTurn().getWaitForFunction().clear();  
 gameActivity.turnUI();  
 gameActivity.getActionCardsPlayingAdapter().notifyDataSetChanged();  
 gameActivity.setVisibilityForRVActionCardsPlaying(View.*INVISIBLE*);  
 game.getPlayer().putArrayInDiscard(cards);  
 gameActivity.updateCards(true);  
 game.setDoneAttack(true);  
 return;  
 }  
  
 game.getTurn().getWaitForFunction().setTypeOfAction("trash");  
 gameActivity.getActionCardsPlayingAdapter().notifyDataSetChanged();  
 gameActivity.uploadActionButtons(new String[]{gameActivity.getString(R.string.*undo*), gameActivity.getString(R.string.*confirm\_trash*)}, true);  
 }  
  
 @Override  
 public void handleButtonClicks(String buttonText, GameManager game, GameActivity gameActivity) {  
 if (buttonText.equals(gameActivity.getString(R.string.*undo*))) {  
 game.getTurn().getWaitForFunction().undo();  
 gameActivity.updateActionButtons();  
 gameActivity.getActionCardsPlayingAdapter().notifyDataSetChanged();  
 }  
 else if (buttonText.equals(gameActivity.getString(R.string.*confirm\_trash*))) {  
 ArrayList<String> cardSelected = game.getTurn().getWaitForFunction().cardsSelectedForDialog();  
 game.addToTrash(cardSelected.get(0), 1);  
 game.getPlayer().putArrayInDiscard(game.getTurn().getWaitForFunction().cardsLeftForDialog());  
 game.getTurn().getWaitForFunction().clear();  
 gameActivity.updateActionButtons();  
 game.getPlayer().updateArrayHand();  
 gameActivity.getHandAdapter().notifyDataSetChanged();  
 gameActivity.getActionCardsPlayingAdapter().notifyDataSetChanged();  
 gameActivity.setVisibilityForRVActionCardsPlaying(View.*INVISIBLE*);  
 game.setDoneAttack(true);  
 }  
 }  
  
 @Override  
 public boolean isCardToUse(String cardName, GameManager game, GameActivity gameActivity) {  
 return true;  
 }  
}

Bureaucrat

package com.example.dominion\_game.cards\_classes;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.activities.GameActivity;  
import com.example.dominion\_game.classes.Card;  
import com.example.dominion\_game.classes.GameManager;  
import com.example.dominion\_game.classes.Help;  
  
public class Bureaucrat extends Card {  
 public Bureaucrat() {  
 super("Bureaucrat", 4, R.mipmap.*bureaucrat*, R.mipmap.*bureaucrat\_sh*, "action");  
 }  
 @Override  
 public void play(GameManager game, GameActivity gameActivity) {  
 game.getPlayer().addToDeck(game.getCard("Silver"));  
 game.useAfterPlay(this.getName(), true);  
 }  
  
 @Override  
 public void attack(GameManager game, GameActivity gameActivity) {  
 if (game.getPlayer().containsTypeCards("victory")) {  
 gameActivity.waitForHand(this.getName(), 1, 1, "topdeck", false);  
 gameActivity.uploadActionButtons(new String[]{gameActivity.getString(R.string.*undo*), gameActivity.getString(R.string.*confirm\_top\_deck*)}, true);  
 }  
 else {  
 game.addHashToLog(game.getPlayer().getHand(), "in action", "Reveals hand:");  
 game.setDoneAttack(true);  
 }  
 }  
  
 @Override  
 public void handleButtonClicks(String buttonText, GameManager game, GameActivity gameActivity) {  
 if (buttonText.equals(gameActivity.getString(R.string.*undo*))) {  
 game.getTurn().getWaitForFunction().undo();  
 gameActivity.updateActionButtons();  
 gameActivity.getHandAdapter().notifyDataSetChanged();  
 }  
 else if (buttonText.equals(gameActivity.getString(R.string.*confirm\_top\_deck*))) {  
 String cardName = String.*valueOf*(game.getTurn().getWaitForFunction().getCardsForActionCardPlay().keySet().toArray()[0]);  
 game.getPlayer().handToDeck(cardName, gameActivity);  
 game.getTurn().getWaitForFunction().clear();  
 gameActivity.updateActionButtons();  
 game.getPlayer().updateArrayHand();  
 gameActivity.getHandAdapter().notifyDataSetChanged();  
 game.setDoneAttack(true);  
 }  
 }  
  
 @Override  
 public boolean isCardToUse(String cardName, GameManager game, GameActivity gameActivity) {  
 return Help.*nameToCard*(cardName).getType().equals("victory");  
 }  
}

Cellar

package com.example.dominion\_game.cards\_classes;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.activities.GameActivity;  
import com.example.dominion\_game.classes.Card;  
import com.example.dominion\_game.classes.GameManager;  
import com.example.dominion\_game.classes.Help;  
  
public class Cellar extends Card {  
 public Cellar() {  
 super("Cellar", 2, R.mipmap.*cellar*, R.mipmap.*cellar\_sh*, "action");  
 }  
  
 @Override  
 public void play(GameManager game, GameActivity gameActivity) {  
 game.getTurn().addActions(1);  
 if (Help.*sizeOfHash*(game.getPlayer().getHand()) > 0) {  
 gameActivity.waitForHand(this.getName(), 0, -1, "discard", false);  
 gameActivity.uploadActionButtons(new String[]{gameActivity.getString(R.string.*undo*), gameActivity.getString(R.string.*confirm\_discard*)}, true);  
 }  
 else  
 game.useAfterPlay(this.getName(), false);  
 }  
  
 @Override  
 public void handleButtonClicks(String buttonText, GameManager game, GameActivity gameActivity) {  
 if (buttonText.equals(gameActivity.getString(R.string.*undo*))) {  
 game.getTurn().getWaitForFunction().undo();  
 gameActivity.updateActionButtons();  
 gameActivity.getHandAdapter().notifyDataSetChanged();  
 }  
 else if (buttonText.equals(gameActivity.getString(R.string.*confirm\_discard*))) {  
 for (String cardName : game.getTurn().getWaitForFunction().getCardsForActionCardPlay().keySet()) {  
 if (game.getPlayer().getHand().get(cardName).equals(game.getTurn().getWaitForFunction().getCardsForActionCardPlay().get(cardName)))  
 game.getPlayer().getHand().remove(cardName);  
 else if (game.getPlayer().getHand().get(cardName) > game.getTurn().getWaitForFunction().getCardsForActionCardPlay().get(cardName))  
 game.getPlayer().getHand().put(cardName, game.getPlayer().getHand().get(cardName) - game.getTurn().getWaitForFunction().getCardsForActionCardPlay().get(cardName));  
 for (int i = 0; i < game.getTurn().getWaitForFunction().getCardsForActionCardPlay().get(cardName); i++)  
 game.getPlayer().getDiscard().add(cardName);  
 }  
 game.getPlayer().takeCardsToHand(Help.*sizeOfHash*(game.getTurn().getWaitForFunction().getCardsForActionCardPlay()), gameActivity, game, game.getTabs());  
 game.getTurn().getWaitForFunction().clear();  
 gameActivity.updateActionButtons();  
 game.getPlayer().updateArrayHand();  
 gameActivity.getHandAdapter().notifyDataSetChanged();  
 game.useAfterPlay(this.getName(), false);  
 }  
 }  
  
 @Override  
 public boolean isCardToUse(String cardName, GameManager game, GameActivity gameActivity) {  
 return true;  
 }  
}

Chapel

package com.example.dominion\_game.cards\_classes;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.activities.GameActivity;  
import com.example.dominion\_game.classes.Card;  
import com.example.dominion\_game.classes.GameManager;  
import com.example.dominion\_game.classes.Help;  
  
public class Chapel extends Card {  
 public Chapel() {  
 super("Chapel", 2, R.mipmap.*chapel*, R.mipmap.*chapel\_sh*, "action");  
 }  
  
 @Override  
 public void play(GameManager game, GameActivity gameActivity) {  
 if (Help.*sizeOfHash*(game.getPlayer().getHand()) > 0) {  
 gameActivity.waitForHand(this.getName(), 0, 4, "trash", false);  
 gameActivity.uploadActionButtons(new String[]{gameActivity.getString(R.string.*undo*), gameActivity.getString(R.string.*confirm\_trash*)}, true);  
 }  
 else  
 game.useAfterPlay(this.getName(), false);  
 }  
  
 @Override  
 public void handleButtonClicks(String buttonText, GameManager game, GameActivity gameActivity) {  
 if (buttonText.equals(gameActivity.getString(R.string.*undo*))) {  
 game.getTurn().getWaitForFunction().undo();  
 gameActivity.updateActionButtons();  
 gameActivity.getHandAdapter().notifyDataSetChanged();  
 }  
 else if (buttonText.equals(gameActivity.getString(R.string.*confirm\_trash*))) {  
 for (String cardName : game.getTurn().getWaitForFunction().getCardsForActionCardPlay().keySet()) {  
 if (game.getPlayer().getHand().get(cardName).equals(game.getTurn().getWaitForFunction().getCardsForActionCardPlay().get(cardName)))  
 game.getPlayer().getHand().remove(cardName);  
 else if (game.getPlayer().getHand().get(cardName) > game.getTurn().getWaitForFunction().getCardsForActionCardPlay().get(cardName))  
 game.getPlayer().getHand().put(cardName, game.getPlayer().getHand().get(cardName) - game.getTurn().getWaitForFunction().getCardsForActionCardPlay().get(cardName));  
 game.addToTrash(cardName, game.getTurn().getWaitForFunction().getCardsForActionCardPlay().get(cardName));  
 }  
 game.getTurn().getWaitForFunction().clear();  
 gameActivity.updateActionButtons();  
 game.getPlayer().updateArrayHand();  
 gameActivity.getHandAdapter().notifyDataSetChanged();  
 game.useAfterPlay(this.getName(), false);  
 }  
 }  
  
 @Override  
 public boolean isCardToUse(String cardName, GameManager game, GameActivity gameActivity) {  
 return true;  
 }  
}

Copper

*/\*\*  
 \* Copper is an example to a card in game that extends Card  
 \* This class overrides an abstract function from Card: play  
 \*/*package com.example.dominion\_game.cards\_classes;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.activities.GameActivity;  
import com.example.dominion\_game.classes.Card;  
import com.example.dominion\_game.classes.GameManager;  
  
public class Copper extends Card {  
 */\*\*  
 \* The Constructor with the Copper Card attributes  
 \*/* public Copper() {  
 super("Copper", 0, R.mipmap.*copper*, R.mipmap.*copper\_sh*, "treasure");  
 }  
 @Override  
 public void play(GameManager game, GameActivity gameActivity) {  
 game.getTurn().addTreasure(1);  
 game.useAfterPlay(this.getName(), false);  
 }  
  
}

CouncilRoom

*/\*\*  
 \* Copper is an example to a card in game that extends Card  
 \* This class overrides an abstract function from Card: play  
 \*/*package com.example.dominion\_game.cards\_classes;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.activities.GameActivity;  
import com.example.dominion\_game.classes.Card;  
import com.example.dominion\_game.classes.GameManager;  
  
public class Copper extends Card {  
 */\*\*  
 \* The Constructor with the Copper Card attributes  
 \*/* public Copper() {  
 super("Copper", 0, R.mipmap.*copper*, R.mipmap.*copper\_sh*, "treasure");  
 }  
 @Override  
 public void play(GameManager game, GameActivity gameActivity) {  
 game.getTurn().addTreasure(1);  
 game.useAfterPlay(this.getName(), false);  
 }  
  
}

Festival

*/\*\*  
 \* Festival is an example to a card in game that extends Card  
 \* This class overrides an abstract function from Card: play  
 \*/*package com.example.dominion\_game.cards\_classes;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.activities.GameActivity;  
import com.example.dominion\_game.classes.Card;  
import com.example.dominion\_game.classes.GameManager;  
  
public class Festival extends Card {  
 */\*\*  
 \* The Constructor with the Festival Card attributes  
 \*/* public Festival() {  
 super("Festival", 5, R.mipmap.*festival*, R.mipmap.*festival\_sh*, "action");  
 }  
 @Override  
 public void play(GameManager game, GameActivity gameActivity) {  
 game.getTurn().addActions(2);  
 game.getTurn().addBuys(1);  
 game.getTurn().addTreasure(2);  
 game.useAfterPlay(this.getName(), false);  
 }  
}

Gardens

package com.example.dominion\_game.cards\_classes;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.activities.GameActivity;  
import com.example.dominion\_game.classes.Card;  
import com.example.dominion\_game.classes.GameManager;  
import com.example.dominion\_game.classes.Help;  
import com.example.dominion\_game.classes.Player;  
  
import java.util.HashMap;  
  
public class Gardens extends Card {  
 public Gardens() {  
 super("Gardens", 4, R.mipmap.*gardens*, R.mipmap.*gardens\_sh*, "victory");  
 }  
 @Override  
 public void play(GameManager game, GameActivity gameActivity) {  
 }  
  
 @Override  
 public int getValue(GameManager gameManager) {  
 return Help.*sizeOfHash*(gameManager.getPlayer().allCards(gameManager))/10;  
 }  
}

Gold

package com.example.dominion\_game.cards\_classes;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.activities.GameActivity;  
import com.example.dominion\_game.classes.Card;  
import com.example.dominion\_game.classes.GameManager;  
  
public class Gold extends Card {  
 public Gold() {  
 super("Gold", 6, R.mipmap.*gold*, R.mipmap.*gold\_sh*, "treasure");  
 }  
 @Override  
 public void play(GameManager game, GameActivity gameActivity) {  
 game.getTurn().addTreasure(3);  
 game.useAfterPlay(this.getName(), false);  
 }  
  
}

Harbinger

package com.example.dominion\_game.cards\_classes;  
  
import android.view.View;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.activities.GameActivity;  
import com.example.dominion\_game.classes.Card;  
import com.example.dominion\_game.classes.GameManager;  
import com.example.dominion\_game.classes.Help;  
  
import java.util.ArrayList;  
  
public class Harbinger extends Card {  
 public Harbinger() {  
 super("Harbinger", 3, R.mipmap.*harbinger*, R.mipmap.*harbinger\_sh*, "action");  
 }  
 @Override  
 public void play(GameManager game, GameActivity gameActivity) {  
 game.getPlayer().takeCardsToHand(1, gameActivity, game, game.getTabs());  
 game.getTurn().addActions(1);  
 if (game.getPlayer().getDiscard().size() > 0) {  
 game.getTurn().getWaitForFunction().handleWaitingForActionCardsDialog(this.getName(), 1, 1, "topdeck", true, game.getPlayer().getDiscard());  
 gameActivity.getActionCardsPlayingAdapter().notifyDataSetChanged();  
 gameActivity.setVisibilityForRVActionCardsPlaying(View.*VISIBLE*);  
 gameActivity.uploadActionButtons(new String[]{gameActivity.getString(R.string.*undo*), "Don't Topdeck", gameActivity.getString(R.string.*confirm\_top\_deck*)}, false);  
 gameActivity.setVisibilityForAction(1, View.*VISIBLE*);  
 }  
 else  
 game.useAfterPlay(this.getName(), false);  
 }  
  
 @Override  
 public boolean isMarkCardSelectedFromHandWhenHandle() {  
 return true;  
 }  
  
 @Override  
 public void handleClickOnHandOrDialog(String cardName, GameManager game, GameActivity gameActivity) {  
 gameActivity.setVisibilityForAction(0, View.*VISIBLE*);  
 gameActivity.setVisibilityForAction(2, View.*VISIBLE*);  
 }  
  
 @Override  
 public void handleButtonClicks(String buttonText, GameManager game, GameActivity gameActivity) {  
 if (buttonText.equals(gameActivity.getString(R.string.*undo*))) {  
 game.getTurn().getWaitForFunction().undo();  
 gameActivity.setVisibilityForAction(0, View.*GONE*);  
 gameActivity.setVisibilityForAction(2, View.*GONE*);  
 gameActivity.getActionCardsPlayingAdapter().notifyDataSetChanged();  
 }  
 else if (buttonText.equals("Don't Topdeck")) {  
 game.getTurn().getWaitForFunction().clear();  
 gameActivity.invisibleButtons(3);  
 gameActivity.turnUI();game.getPlayer().updateArrayHand();  
 gameActivity.getHandAdapter().notifyDataSetChanged();  
 gameActivity.getActionCardsPlayingAdapter().notifyDataSetChanged();  
 gameActivity.setVisibilityForRVActionCardsPlaying(View.INVISIBLE);  
 game.useAfterPlay(this.getName(), false);  
 }  
 else if (buttonText.equals(gameActivity.getString(R.string.confirm\_top\_deck))) {  
 ArrayList<String> cardSelected = Help.arrayListOfPairsToArrayList(game.getTurn().getWaitForFunction().getCardsForDialog());  
 if (cardSelected.size() > 0) {  
 game.getPlayer().getDiscard().remove(cardSelected.get(0));  
 game.getPlayer().getDeck().add(cardSelected.get(0));  
 gameActivity.getActionCardsPlayingAdapter().notifyDataSetChanged();  
 gameActivity.updateCards(false);  
 }  
 game.getTurn().getWaitForFunction().clear();  
 gameActivity.invisibleButtons(3);  
 gameActivity.turnUI();  
 game.getPlayer().updateArrayHand();  
 gameActivity.getHandAdapter().notifyDataSetChanged();  
 gameActivity.getActionCardsPlayingAdapter().notifyDataSetChanged();  
 gameActivity.setVisibilityForRVActionCardsPlaying(View.INVISIBLE);  
 game.useAfterPlay(this.getName(), false);  
 }  
 }  
  
 @Override  
 public boolean isCardToUse(String cardName, GameManager game, GameActivity gameActivity) {  
 return true;  
 }  
}

Laboratory

*/\*\*  
 \* Laboratory is an example to a card in game that extends Card  
 \* This class overrides an abstract function from Card: play  
 \*/*package com.example.dominion\_game.cards\_classes;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.activities.GameActivity;  
import com.example.dominion\_game.classes.Card;  
import com.example.dominion\_game.classes.GameManager;  
  
public class Laboratory extends Card {  
 */\*\*  
 \* The Constructor with the Laboratory Card attributes  
 \*/* public Laboratory() {  
 super("Laboratory", 5, R.mipmap.*laboratory*, R.mipmap.*laboratory\_sh*, "action");  
 }  
 @Override  
 public void play(GameManager game, GameActivity gameActivity) {  
 game.getPlayer().takeCardsToHand(2, gameActivity, game, game.getTabs());  
 game.getTurn().addActions(1);  
 game.useAfterPlay(this.getName(), false);  
 }  
}

Library

package com.example.dominion\_game.cards\_classes;  
  
import android.view.View;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.activities.GameActivity;  
import com.example.dominion\_game.classes.Card;  
import com.example.dominion\_game.classes.GameManager;  
import com.example.dominion\_game.classes.Help;  
  
public class Library extends Card {  
 public Library() {  
 super("Library", 5, R.mipmap.*library*, R.mipmap.*library\_sh*, "action");  
 }  
 @Override  
 public void play(GameManager game, GameActivity gameActivity) {  
 if (Help.*sizeOfHash*(game.getPlayer().getHand()) >= 7) {  
 game.useAfterPlay(this.getName(), false);  
 return;  
 }  
  
 game.getTurn().getWaitForFunction().handleWaitingForButtonsOnly(this.getName());  
 this.takeCardsWhileNotAction(game, gameActivity);  
 }  
  
 public void takeCardsWhileNotAction(GameManager game, GameActivity gameActivity) {  
 while (Help.*sizeOfHash*(game.getPlayer().getHand()) < 7 && !(game.getPlayer().getDiscard().isEmpty() && game.getPlayer().getDeck().isEmpty())) {  
 if (game.getPlayer().getDeck().isEmpty())  
 game.getPlayer().discardToDeck(game, game.getTabs());  
 if (game.getPlayer().getHand().containsKey(game.getPlayer().getDeck().get(game.getPlayer().getDeck().size() - 1)))  
 game.getPlayer().getHand().put(game.getPlayer().getDeck().get(game.getPlayer().getDeck().size() - 1),  
 game.getPlayer().getHand().get(game.getPlayer().getDeck().get(game.getPlayer().getDeck().size() - 1)) + 1);  
 else  
 game.getPlayer().getHand().put(game.getPlayer().getDeck().get(game.getPlayer().getDeck().size() - 1), 1);  
 // removes the last index which is the first card to take from deck  
 String cardName = game.getPlayer().getDeck().remove(game.getPlayer().getDeck().size() - 1);  
 game.getPlayer().updateArrayHand();  
 gameActivity.getHandAdapter().notifyDataSetChanged();  
 if (Help.*nameToCard*(cardName).getType().equals("action")) {  
 gameActivity.uploadActionButtons(new String[]{"Keep " + Help.*nameToCard*(cardName).getNameToDisplay(), "Skip It"}, false);  
 gameActivity.setVisibilityForAction(0, View.*VISIBLE*);  
 gameActivity.setVisibilityForAction(1, View.*VISIBLE*);  
 game.getTurn().getWaitForFunction().insertCardSelectedInHand(cardName);  
 return;  
 }  
 }  
 game.getPlayer().putArrayInDiscard(Help.*arrayListOfPairsToArrayList*(game.getTurn().getWaitForFunction().getCardsForDialog()));  
 game.getTurn().getWaitForFunction().clear();  
 gameActivity.invisibleButtons(2);  
 gameActivity.turnUI();  
 game.getPlayer().updateArrayHand();  
 gameActivity.getHandAdapter().notifyDataSetChanged();  
 gameActivity.getActionCardsPlayingAdapter().notifyDataSetChanged();  
 gameActivity.setVisibilityForRVActionCardsPlaying(View.*INVISIBLE*);  
 game.useAfterPlay(this.getName(), false);  
 }  
  
 @Override  
 public void handleButtonClicks(String buttonText, GameManager game, GameActivity gameActivity) {  
 if (buttonText.equals("Skip It")) {  
 String cardName = String.*valueOf*(game.getTurn().getWaitForFunction().getCardsForActionCardPlay().keySet().toArray()[0]);  
 if (!game.getPlayer().removeFromHand(cardName)) {  
 game.getTurn().getWaitForFunction().undo();  
 this.takeCardsWhileNotAction(game, gameActivity);  
 return;  
 }  
  
 game.getTurn().getWaitForFunction().undo();  
 game.getTurn().getWaitForFunction().addToCardsForDialog(cardName);  
 gameActivity.getActionCardsPlayingAdapter().notifyDataSetChanged();  
 gameActivity.setVisibilityForRVActionCardsPlaying(View.*VISIBLE*);  
 }  
 this.takeCardsWhileNotAction(game, gameActivity);  
 }  
}

Market

package com.example.dominion\_game.cards\_classes;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.activities.GameActivity;  
import com.example.dominion\_game.classes.Card;  
import com.example.dominion\_game.classes.GameManager;  
  
public class Market extends Card {  
 public Market() {  
 super("Market", 5, R.mipmap.*market*, R.mipmap.*market\_sh*, "action");  
 }  
 @Override  
 public void play(GameManager game, GameActivity gameActivity) {  
 game.getPlayer().takeCardsToHand(1, gameActivity, game, game.getTabs());  
 game.getTurn().addActions(1);  
 game.getTurn().addBuys(1);  
 game.getTurn().addTreasure(1);  
 game.useAfterPlay(this.getName(), false);  
 }  
}

Merchant

package com.example.dominion\_game.cards\_classes;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.activities.GameActivity;  
import com.example.dominion\_game.classes.Card;  
import com.example.dominion\_game.classes.GameManager;  
  
public class Merchant extends Card {  
 public Merchant() {  
 super("Merchant", 3, R.mipmap.*merchant*, R.mipmap.*merchant\_sh*, "action");  
 }  
 @Override  
 public void play(GameManager game, GameActivity gameActivity) {  
 game.getPlayer().takeCardsToHand(1, gameActivity, game, game.getTabs());  
 game.getTurn().addActions(1);  
 game.useAfterPlay(this.getName(), false);  
 }  
  
 @Override  
 public void afterPlay(GameManager game, String cardNameUsed) {  
 if (game.getTurn().getTreasureCardsPlayed().get("Silver") != null  
 && game.getTurn().getTreasureCardsPlayed().get("Silver") == 1  
 && cardNameUsed.equals("Silver"))  
 game.getTurn().addTreasure(1);  
 }  
}

Militia

package com.example.dominion\_game.cards\_classes;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.activities.GameActivity;  
import com.example.dominion\_game.classes.Card;  
import com.example.dominion\_game.classes.GameManager;  
import com.example.dominion\_game.classes.Help;  
  
public class Militia extends Card {  
 public Militia() {  
 super("Militia", 4, R.mipmap.*militia*, R.mipmap.*militia\_sh*, "action");  
 }  
 @Override  
 public void play(GameManager game, GameActivity gameActivity) {  
 game.getTurn().addTreasure(2);  
 game.useAfterPlay(this.getName(), true);  
 }  
  
 @Override  
 public void attack(GameManager game, GameActivity gameActivity) {  
 if (Help.*sizeOfHash*(game.getPlayer().getHand()) > 3) {  
 gameActivity.waitForHand(this.getName(), Help.*sizeOfHash*(game.getPlayer().getHand()) - 3,  
 Help.*sizeOfHash*(game.getPlayer().getHand()) - 3, "discard", false);  
 gameActivity.uploadActionButtons(new String[]{gameActivity.getString(R.string.*undo*), gameActivity.getString(R.string.*confirm\_discard*)}, true);  
 }  
 else  
 game.setDoneAttack(true);  
 }  
  
 @Override  
 public void handleButtonClicks(String buttonText, GameManager game, GameActivity gameActivity) {  
 if (buttonText.equals(gameActivity.getString(R.string.*undo*))) {  
 game.getTurn().getWaitForFunction().undo();  
 gameActivity.updateActionButtons();  
 gameActivity.getHandAdapter().notifyDataSetChanged();  
 }  
 else if (buttonText.equals(gameActivity.getString(R.string.*confirm\_discard*))) {  
 for (String cardName : game.getTurn().getWaitForFunction().getCardsForActionCardPlay().keySet()) {  
 if (game.getPlayer().getHand().get(cardName).equals(game.getTurn().getWaitForFunction().getCardsForActionCardPlay().get(cardName)))  
 game.getPlayer().getHand().remove(cardName);  
 else if (game.getPlayer().getHand().get(cardName) > game.getTurn().getWaitForFunction().getCardsForActionCardPlay().get(cardName))  
 game.getPlayer().getHand().put(cardName, game.getPlayer().getHand().get(cardName) - game.getTurn().getWaitForFunction().getCardsForActionCardPlay().get(cardName));  
 for (int i = 0; i < game.getTurn().getWaitForFunction().getCardsForActionCardPlay().get(cardName); i++)  
 game.getPlayer().getDiscard().add(cardName);  
 }  
 game.getTurn().getWaitForFunction().clear();  
 gameActivity.updateActionButtons();  
 game.getPlayer().updateArrayHand();  
 gameActivity.getHandAdapter().notifyDataSetChanged();  
 game.setDoneAttack(true);  
 }  
 }  
  
 @Override  
 public boolean isCardToUse(String cardName, GameManager game, GameActivity gameActivity) {  
 return true;  
 }  
}

Mine

package com.example.dominion\_game.cards\_classes;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.activities.GameActivity;  
import com.example.dominion\_game.classes.Card;  
import com.example.dominion\_game.classes.GameManager;  
import com.example.dominion\_game.classes.Help;  
  
public class Mine extends Card {  
 public Mine() {  
 super("Mine", 5, R.mipmap.*mine*, R.mipmap.*mine\_sh*, "action");  
 }  
 @Override  
  
 public void play(GameManager game, GameActivity gameActivity) {  
 if (game.getPlayer().containsTypeCards("treasure")) {  
 gameActivity.waitForHand(this.getName(), 0, 1, "trash", false);  
 gameActivity.uploadActionButtons(new String[]{gameActivity.getString(R.string.*undo*), gameActivity.getString(R.string.*confirm\_trash*)}, true);  
 }  
 else  
 game.useAfterPlay(this.getName(), false);  
 }  
  
 @Override  
 public void handleButtonClicks(String buttonText, GameManager game, GameActivity gameActivity) {  
 if (buttonText.equals(gameActivity.getString(R.string.*undo*))) {  
 game.getTurn().getWaitForFunction().undo();  
 gameActivity.updateActionButtons();  
 gameActivity.getHandAdapter().notifyDataSetChanged();  
 }  
 else if (buttonText.equals(gameActivity.getString(R.string.*confirm\_trash*))) {  
 if (game.getTurn().getWaitForFunction().getCardsForActionCardPlay().size() == 0) {  
 game.getTurn().getWaitForFunction().clear();  
 gameActivity.updateActionButtons();  
 game.getPlayer().updateArrayHand();  
 gameActivity.getHandAdapter().notifyDataSetChanged();  
 game.useAfterPlay(this.getName(), false);  
 return;  
 }  
 String cardName = String.*valueOf*(game.getTurn().getWaitForFunction().getCardsForActionCardPlay().keySet().toArray()[0]);  
 if (game.getPlayer().getHand().get(cardName).equals(game.getTurn().getWaitForFunction().getCardsForActionCardPlay().get(cardName)))  
 game.getPlayer().getHand().remove(cardName);  
 else if (game.getPlayer().getHand().get(cardName) > game.getTurn().getWaitForFunction().getCardsForActionCardPlay().get(cardName))  
 game.getPlayer().getHand().put(cardName, game.getPlayer().getHand().get(cardName) - game.getTurn().getWaitForFunction().getCardsForActionCardPlay().get(cardName));  
 game.addToTrash(cardName, game.getTurn().getWaitForFunction().getCardsForActionCardPlay().get(cardName));  
 game.getTurn().getWaitForFunction().clear();  
  
 gameActivity.waitForBoard(this.getName(), 1, 1);  
 game.getTurn().getWaitForFunction().setMaxPriceForGain(Help.*nameToCard*(cardName).getPrice() + 3);  
 gameActivity.updateActionButtons();  
 game.getPlayer().updateArrayHand();  
 gameActivity.getHandAdapter().notifyDataSetChanged();  
 game.useAfterPlay(this.getName(), false);  
 }  
 }  
  
 @Override  
 public boolean isCardToUse(String cardName, GameManager game, GameActivity gameActivity) {  
 return Help.*nameToCard*(cardName).getType().equals("treasure");  
 }  
  
 @Override  
 public void clickOnBoard(String cardName, GameManager game, GameActivity gameActivity) {  
 if (!game.getPlayer().addToHand(game.getCard(cardName))) {  
 game.getTurn().getWaitForFunction().getCardsForActionCardPlay().clear();  
 return;  
 }  
 game.getTurn().getWaitForFunction().clear();  
 gameActivity.turnUI();  
 game.getPlayer().updateArrayHand();  
 gameActivity.getHandAdapter().notifyDataSetChanged();  
 }  
  
 @Override  
 public boolean isCardToGetFromBoard(String cardName, GameManager game, GameActivity gameActivity) {  
 return Help.*nameToCard*(cardName).getPrice() <= game.getTurn().getWaitForFunction().getMaxPriceForGain()  
 && Help.*nameToCard*(cardName).getType().equals("treasure");  
 }  
}

Moat

package com.example.dominion\_game.cards\_classes;  
  
import android.view.View;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.activities.GameActivity;  
import com.example.dominion\_game.classes.Card;  
import com.example.dominion\_game.classes.GameManager;  
import com.example.dominion\_game.classes.Help;  
  
public class Moat extends Card {  
 public Moat() {  
 super("Moat", 2, R.mipmap.*moat*, R.mipmap.*moat\_sh*, "action");  
 }  
 @Override  
 public void play(GameManager game, GameActivity gameActivity) {  
 game.getPlayer().takeCardsToHand(2, gameActivity, game, game.getTabs());  
 game.useAfterPlay(this.getName(), false);  
 }  
  
 @Override  
 public void reaction(GameManager game, GameActivity gameActivity) {  
 game.getTurn().getWaitForFunction().handleWaitingForButtonsOnly(this.getName());  
 gameActivity.uploadActionButtons(new String[]{"Reveal Moat", "Don't Reveal"}, false);  
 gameActivity.setVisibilityForAction(0, View.*VISIBLE*);  
 gameActivity.setVisibilityForAction(1, View.*VISIBLE*);  
 game.getTurn().getWaitForFunction().insertCardSelectedInHand(game.getTurn().getLastActionCardForWait());  
 game.getTurn().removeLastAttack();  
  
 }  
  
 @Override  
 public void handleButtonClicks(String buttonText, GameManager game, GameActivity gameActivity) {  
 String cardName = String.*valueOf*(game.getTurn().getWaitForFunction().getCardsForActionCardPlay().keySet().toArray()[0]);  
 game.getTurn().getWaitForFunction().clear();  
 gameActivity.invisibleButtons(2);  
  
 if (buttonText.startsWith("Reveal"))  
 game.setDoneAttack(true);  
 else  
 Help.*nameToCard*(cardName).attack(game, gameActivity);  
 }  
}

Moneylender

package com.example.dominion\_game.cards\_classes;  
  
import android.view.View;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.activities.GameActivity;  
import com.example.dominion\_game.classes.Card;  
import com.example.dominion\_game.classes.GameManager;  
import com.example.dominion\_game.classes.Help;  
  
public class Moneylender extends Card {  
 public Moneylender() {  
 super("Moneylender", 4, R.mipmap.*moneylender*, R.mipmap.*moneylender\_sh*, "action");  
 }  
  
 @Override  
 public void play(GameManager game, GameActivity gameActivity) {  
 if (!game.getPlayer().containsCard("Copper")) {  
 game.useAfterPlay(this.getName(), false);  
 return;  
 }  
  
 gameActivity.waitForHand(this.getName(), 1, 1, "trash", true);  
 gameActivity.uploadActionButtons(new String[]{gameActivity.getString(R.string.*undo*), "Don't Trash", gameActivity.getString(R.string.*confirm\_trash*)}, false);  
 gameActivity.setVisibilityForAction(1, View.*VISIBLE*);  
 }  
  
 @Override  
 public void handleClickOnHandOrDialog(String cardName, GameManager game, GameActivity gameActivity) {  
 gameActivity.setVisibilityForAction(0, View.*VISIBLE*);  
 gameActivity.setVisibilityForAction(2, View.*VISIBLE*);  
 }  
  
 @Override  
 public boolean isMarkCardSelectedFromHandWhenHandle() {  
 return true;  
 }  
  
 @Override  
 public void handleButtonClicks(String buttonText, GameManager game, GameActivity gameActivity) {  
 if (buttonText.equals(gameActivity.getString(R.string.*undo*))) {  
 game.getTurn().getWaitForFunction().undo();  
 gameActivity.setVisibilityForAction(0, View.*GONE*);  
 gameActivity.setVisibilityForAction(2, View.*GONE*);  
 }  
 else if (buttonText.equals("Don't Trash")) {  
 game.getTurn().getWaitForFunction().clear();  
 gameActivity.invisibleButtons(3);  
 gameActivity.turnUI();  
 game.useAfterPlay(this.getName(), false);  
 }  
 else if (buttonText.equals(gameActivity.getString(R.string.*confirm\_trash*))) {  
 if (game.getPlayer().getHand().keySet().contains("Copper")) {  
 if (!game.getPlayer().removeFromHand("Copper")) {  
 game.useAfterPlay(this.getName(), false);  
 return;  
 }  
 game.addToTrash("Copper", 1);  
 game.getTurn().addTreasure(3);  
 }  
 game.getTurn().getWaitForFunction().clear();  
 gameActivity.invisibleButtons(3);  
 gameActivity.turnUI();  
 game.useAfterPlay(this.getName(), false);  
 }  
 game.getPlayer().updateArrayHand();  
 gameActivity.getHandAdapter().notifyDataSetChanged();  
 }  
  
 @Override  
 public boolean isCardToUse(String cardName, GameManager game, GameActivity gameActivity) {  
 return Help.*nameToCard*(cardName).getName().equals("Copper");  
 }  
}

Poacher

package com.example.dominion\_game.cards\_classes;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.activities.GameActivity;  
import com.example.dominion\_game.classes.Card;  
import com.example.dominion\_game.classes.GameManager;  
  
public class Poacher extends Card {  
 public Poacher() {  
 super("Poacher", 4, R.mipmap.*poacher*, R.mipmap.*poacher\_sh*, "action");  
 }  
 @Override  
 public void play(GameManager game, GameActivity gameActivity) {  
 game.getPlayer().takeCardsToHand(1, gameActivity, game, game.getTabs());  
 game.getTurn().addActions(1);  
 game.getTurn().addTreasure(1);  
 int count = 0;  
 for (String cardName : game.getBoard().keySet()) {  
 if (game.getBoard().get(cardName) == 0)  
 count++;  
 }  
 if (count == 0) {  
 game.useAfterPlay(this.getName(), false);  
 return;  
 }  
 gameActivity.waitForHand(this.getName(), count, count, "discard", false);  
 gameActivity.uploadActionButtons(new String[]{gameActivity.getString(R.string.*undo*), gameActivity.getString(R.string.*confirm\_discard*)}, true);  
 }  
  
 @Override  
 public void handleButtonClicks(String buttonText, GameManager game, GameActivity gameActivity) {  
 if (buttonText.equals(gameActivity.getString(R.string.*undo*))) {  
 game.getTurn().getWaitForFunction().undo();  
 gameActivity.updateActionButtons();  
 gameActivity.getHandAdapter().notifyDataSetChanged();  
 }  
 else if (buttonText.equals(gameActivity.getString(R.string.*confirm\_discard*))) {  
 for (String cardName : game.getTurn().getWaitForFunction().getCardsForActionCardPlay().keySet()) {  
 if (game.getPlayer().getHand().get(cardName).equals(game.getTurn().getWaitForFunction().getCardsForActionCardPlay().get(cardName)))  
 game.getPlayer().getHand().remove(cardName);  
 else if (game.getPlayer().getHand().get(cardName) > game.getTurn().getWaitForFunction().getCardsForActionCardPlay().get(cardName))  
 game.getPlayer().getHand().put(cardName, game.getPlayer().getHand().get(cardName) - game.getTurn().getWaitForFunction().getCardsForActionCardPlay().get(cardName));  
 for (int i = 0; i < game.getTurn().getWaitForFunction().getCardsForActionCardPlay().get(cardName); i++)  
 game.getPlayer().getDiscard().add(cardName);  
 }  
 game.getTurn().getWaitForFunction().clear();  
 gameActivity.updateActionButtons();  
 game.getPlayer().updateArrayHand();  
 gameActivity.getHandAdapter().notifyDataSetChanged();  
 game.useAfterPlay(this.getName(), false);  
 }  
 }  
  
 @Override  
 public boolean isCardToUse(String cardName, GameManager game, GameActivity gameActivity) {  
 return true;  
 }  
}

Remodel

package com.example.dominion\_game.cards\_classes;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.activities.GameActivity;  
import com.example.dominion\_game.classes.Card;  
import com.example.dominion\_game.classes.GameManager;  
import com.example.dominion\_game.classes.Help;  
  
public class Remodel extends Card {  
 public Remodel() {  
 super("Remodel", 4, R.mipmap.remodel, R.mipmap.remodel\_sh, "action");  
 }  
 @Override  
 public void play(GameManager game, GameActivity gameActivity) {  
 gameActivity.waitForHand(this.getName(), 1, 1, "trash", false);  
 gameActivity.uploadActionButtons(new String[]{gameActivity.getString(R.string.undo), gameActivity.getString(R.string.confirm\_trash)}, true);  
 }  
  
 @Override  
 public void handleButtonClicks(String buttonText, GameManager game, GameActivity gameActivity) {  
 if (buttonText.equals(gameActivity.getString(R.string.undo))) {  
 game.getTurn().getWaitForFunction().undo();  
 gameActivity.updateActionButtons();  
 gameActivity.getHandAdapter().notifyDataSetChanged();  
 }  
 else if (buttonText.equals(gameActivity.getString(R.string.confirm\_trash))) {  
 String cardName = String.valueOf(game.getTurn().getWaitForFunction().getCardsForActionCardPlay().keySet().toArray()[0]);  
 if (game.getPlayer().getHand().get(cardName).equals(game.getTurn().getWaitForFunction().getCardsForActionCardPlay().get(cardName)))  
 game.getPlayer().getHand().remove(cardName);  
 else if (game.getPlayer().getHand().get(cardName) > game.getTurn().getWaitForFunction().getCardsForActionCardPlay().get(cardName))  
 game.getPlayer().getHand().put(cardName, game.getPlayer().getHand().get(cardName) - game.getTurn().getWaitForFunction().getCardsForActionCardPlay().get(cardName));  
 game.addToTrash(cardName, game.getTurn().getWaitForFunction().getCardsForActionCardPlay().get(cardName));  
 game.getTurn().getWaitForFunction().clear();  
  
 gameActivity.waitForBoard(this.getName(), 1, 1);  
 game.getTurn().getWaitForFunction().setMaxPriceForGain(Help.nameToCard(cardName).getPrice() + 2);  
 gameActivity.updateActionButtons();  
 game.getPlayer().updateArrayHand();  
 gameActivity.getHandAdapter().notifyDataSetChanged();  
 game.useAfterPlay(this.getName(), false);  
 }  
 }  
  
 @Override  
 public boolean isCardToUse(String cardName, GameManager game, GameActivity gameActivity) {  
 return true;  
 }  
  
 @Override  
 public void clickOnBoard(String cardName, GameManager game, GameActivity gameActivity) {  
 game.getPlayer().addToDiscard(game.getCard(cardName));  
 game.getTurn().getWaitForFunction().clear();  
 gameActivity.turnUI();  
 game.getPlayer().updateArrayHand();  
 gameActivity.getHandAdapter().notifyDataSetChanged();  
 }  
  
 @Override  
 public boolean isCardToGetFromBoard(String cardName, GameManager game, GameActivity gameActivity) {  
 return Help.nameToCard(cardName).getPrice() <= game.getTurn().getWaitForFunction().getMaxPriceForGain();  
 }  
}

Sentry

package com.example.dominion\_game.cards\_classes;  
  
import android.view.View;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.activities.GameActivity;  
import com.example.dominion\_game.classes.Card;  
import com.example.dominion\_game.classes.GameManager;  
import com.example.dominion\_game.classes.Help;  
  
import java.util.ArrayList;  
  
public class Sentry extends Card {  
 public Sentry() {  
 super("Sentry", 5, R.mipmap.*sentry*, R.mipmap.*sentry\_sh*, "action");  
 }  
 @Override  
 public void play(GameManager game, GameActivity gameActivity) {  
 game.getPlayer().takeCardsToHand(1, gameActivity, game, game.getTabs());  
 game.getTurn().addActions(1);  
 ArrayList<String> cards = game.getPlayer().takeCards(2, game, game.getLastLineFromLog().getTabs());  
 if (cards.size() == 0) {  
 game.useAfterPlay(this.getName(), false);  
 return;  
 }  
  
 game.getTurn().getWaitForFunction().handleWaitingForActionCardsDialog(this.getName(), 0, 2, "trash", false, cards);  
 gameActivity.getActionCardsPlayingAdapter().notifyDataSetChanged();  
 gameActivity.setVisibilityForRVActionCardsPlaying(View.*VISIBLE*);  
 gameActivity.uploadActionButtons(new String[]{gameActivity.getString(R.string.*undo*), gameActivity.getString(R.string.*confirm\_trash*)}, true);  
 }  
  
 @Override  
 public void handleButtonClicks(String buttonText, GameManager game, GameActivity gameActivity) {  
 if (buttonText.equals(gameActivity.getString(R.string.undo))) {  
 game.getTurn().getWaitForFunction().undo();  
 gameActivity.updateActionButtons();  
 gameActivity.getActionCardsPlayingAdapter().notifyDataSetChanged();  
 }  
 else if (buttonText.equals(gameActivity.getString(R.string.confirm\_trash))) {  
 ArrayList<String> cardSelected = game.getTurn().getWaitForFunction().cardsSelectedForDialog();  
 for (String cardName : cardSelected)  
 game.addToTrash(cardName, 1);  
  
 ArrayList<String> cardsLeft = game.getTurn().getWaitForFunction().cardsLeftForDialog();  
 game.getTurn().getWaitForFunction().clear();  
 gameActivity.updateActionButtons();  
 if (cardsLeft.size() == 0) {  
 game.getPlayer().updateArrayHand();  
 gameActivity.getHandAdapter().notifyDataSetChanged();  
 gameActivity.getActionCardsPlayingAdapter().notifyDataSetChanged();  
 gameActivity.setVisibilityForRVActionCardsPlaying(View.INVISIBLE);  
 game.useAfterPlay(this.getName(), false);  
 return;  
 }  
 game.getTurn().getWaitForFunction().handleWaitingForActionCardsDialog(this.getName(), 0, cardsLeft.size(), "discard", false, cardsLeft);  
 gameActivity.getActionCardsPlayingAdapter().notifyDataSetChanged();  
 gameActivity.uploadActionButtons(new String[]{gameActivity.getString(R.string.undo), gameActivity.getString(R.string.confirm\_discard)}, true);  
 }  
  
 else if (buttonText.equals(gameActivity.getString(R.string.confirm\_discard))) {  
 ArrayList<String> cardSelected = game.getTurn().getWaitForFunction().cardsSelectedForDialog();  
 for (String cardName : cardSelected)  
 game.getPlayer().addToDiscard(cardName);  
  
 ArrayList<String> cardsLeft = game.getTurn().getWaitForFunction().cardsLeftForDialog();  
 game.getTurn().getWaitForFunction().clear();  
 gameActivity.updateActionButtons();  
 if (cardsLeft.size() != 2 || cardsLeft.get(0).equals(cardsLeft.get(1))) {  
 game.getPlayer().putArrayInDeck(Help.arrayListOfPairsToArrayList(game.getTurn().getWaitForFunction().getCardsForDialog()));  
 game.getPlayer().updateArrayHand();  
 gameActivity.getHandAdapter().notifyDataSetChanged();  
 gameActivity.getActionCardsPlayingAdapter().notifyDataSetChanged();  
 gameActivity.setVisibilityForRVActionCardsPlaying(View.INVISIBLE);  
 game.useAfterPlay(this.getName(), false);  
 return;  
 }  
 game.getTurn().getWaitForFunction().handleWaitingForActionCardsDialog(this.getName(), 0, 2, "order", false, cardsLeft);  
 gameActivity.getActionCardsPlayingAdapter().notifyDataSetChanged();  
 gameActivity.uploadActionButtons(new String[]{gameActivity.getString(R.string.order)}, false);  
 gameActivity.setVisibilityForAction(0, View.VISIBLE);  
 }  
 else if (buttonText.equals(gameActivity.getString(R.string.order))) {  
 game.getPlayer().putArrayInDeck(Help.arrayListOfPairsToArrayList(game.getTurn().getWaitForFunction().getCardsForDialog()));  
 game.getTurn().getWaitForFunction().clear();  
 gameActivity.invisibleButtons(1);  
 gameActivity.turnUI();  
 game.getPlayer().updateArrayHand();  
 gameActivity.getHandAdapter().notifyDataSetChanged();  
 gameActivity.getActionCardsPlayingAdapter().notifyDataSetChanged();  
 gameActivity.setVisibilityForRVActionCardsPlaying(View.INVISIBLE);  
 game.useAfterPlay(this.getName(), false);  
 }  
 }  
  
 @Override  
 public boolean isCardToUse(String cardName, GameManager game, GameActivity gameActivity) {  
 return true;  
 }  
}

Silver

package com.example.dominion\_game.cards\_classes;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.activities.GameActivity;  
import com.example.dominion\_game.classes.Card;  
import com.example.dominion\_game.classes.GameManager;  
  
public class Silver extends Card {  
 public Silver() {  
 super("Silver", 3, R.mipmap.*silver*, R.mipmap.*silver\_sh*, "treasure");  
 }  
 @Override  
 public void play(GameManager game, GameActivity gameActivity) {  
 game.getTurn().addTreasure(2);  
 game.useAfterPlay(this.getName(), false);  
 }  
}

Smithy

package com.example.dominion\_game.cards\_classes;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.activities.GameActivity;  
import com.example.dominion\_game.classes.Card;  
import com.example.dominion\_game.classes.GameManager;  
  
public class Smithy extends Card {  
 public Smithy() {  
 super("Smithy", 4, R.mipmap.*smithy*, R.mipmap.*smithy\_sh*, "action");  
 }  
 @Override  
 public void play(GameManager game, GameActivity gameActivity) {  
 game.getPlayer().takeCardsToHand(3, gameActivity, game, game.getTabs());  
 game.useAfterPlay(this.getName(), false);  
 }  
}

ThroneRoom

package com.example.dominion\_game.cards\_classes;  
  
import android.view.View;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.activities.GameActivity;  
import com.example.dominion\_game.classes.Card;  
import com.example.dominion\_game.classes.GameManager;  
import com.example.dominion\_game.classes.Help;  
  
public class ThroneRoom extends Card {  
 public ThroneRoom() {  
 super("ThroneRoom", 4, R.mipmap.*throneroom*, R.mipmap.*throneroom\_sh*, "action");  
 }  
  
 @Override  
 public void play(GameManager game, GameActivity gameActivity) {  
 if (!game.getPlayer().containsTypeCards("action")) {  
 game.useAfterPlay(this.getName(), false);  
 return;  
 }  
  
 game.getTimes().push(2);  
 gameActivity.waitForHand(this.getName(), 1, 1, "use", true);  
 gameActivity.uploadActionButtons(new String[]{"Don't Throne"}, false);  
 gameActivity.setVisibilityForAction(0, View.*VISIBLE*);  
 }  
  
 @Override  
 public void handleClickOnHandOrDialog(String cardName, GameManager game, GameActivity gameActivity) {  
 game.getTurn().getWaitForFunction().clear();  
 gameActivity.invisibleButtons(1);  
 game.getTurn().addWaitForPlay(cardName, game.getTimes().peek(), 0, false, false);  
 game.getTurn().addWaitForPlay(cardName, game.getTimes().peek(), 1, false, false);  
 game.useAfterPlay(this.getName(), false);  
 }  
  
 @Override  
 public void handleButtonClicks(String buttonText, GameManager game, GameActivity gameActivity) {  
 if (buttonText.equals("Don't Throne")) {  
 game.getTurn().getWaitForFunction().clear();  
 gameActivity.invisibleButtons(1);  
 gameActivity.turnUI();  
 game.useAfterPlay(this.getName(), false);  
 }  
 }  
  
 @Override  
 public boolean isCardToUse(String cardName, GameManager game, GameActivity gameActivity) {  
 return Help.*nameToCard*(cardName).getType().equals("action");  
 }  
  
 @Override  
 public String getNameToDisplay() {  
 return "Throne Room";  
 }  
}

Vassal

package com.example.dominion\_game.cards\_classes;  
  
import android.view.View;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.activities.GameActivity;  
import com.example.dominion\_game.classes.Card;  
import com.example.dominion\_game.classes.GameManager;  
import com.example.dominion\_game.classes.Help;  
  
import java.util.ArrayList;  
  
public class Vassal extends Card {  
 public Vassal() {  
 super("Vassal", 3, R.mipmap.*vassal*, R.mipmap.*vassal\_sh*, "action");  
 }  
 @Override  
 public void play(GameManager game, GameActivity gameActivity) {  
 game.getTurn().addTreasure(2);  
 ArrayList<String> cards = game.getPlayer().takeCards(1, game, game.getTabs());  
 if (cards.size() == 0) {  
 game.useAfterPlay(this.getName(), false);  
 return;  
 }  
 String cardName = cards.get(0);  
  
 if (!Help.*nameToCard*(cardName).getType().equals("action")) {  
 ArrayList<String> cardArrayList = new ArrayList<>();  
 cardArrayList.add(cardName);  
 game.getPlayer().getDiscard().addAll(cardArrayList);  
 game.useAfterPlay(this.getName(), false);  
 return;  
 }  
 game.getTurn().getWaitForFunction().handleWaitingForButtonsOnly(this.getName());  
 gameActivity.uploadActionButtons(new String[]{"Play " + Help.*nameToCard*(cardName).getNameToDisplay(), "Don't Play"}, false);  
 gameActivity.setVisibilityForAction(0, View.*VISIBLE*);  
 gameActivity.setVisibilityForAction(1, View.*VISIBLE*);  
 game.getTurn().getWaitForFunction().insertCardSelectedInHand(cardName);  
 }  
  
 @Override  
 public void handleButtonClicks(String buttonText, GameManager game, GameActivity gameActivity) {  
 String cardName = String.*valueOf*(game.getTurn().getWaitForFunction().getCardsForActionCardPlay().keySet().toArray()[0]);  
 if (buttonText.startsWith("Play"))  
 game.getTurn().addWaitForPlay(cardName, 1, 0, true, false);  
 else  
 game.getPlayer().addToDiscard(cardName);  
 game.getTurn().getWaitForFunction().clear();  
 gameActivity.invisibleButtons(2);  
 gameActivity.turnUI();  
 game.getPlayer().updateArrayHand();  
 gameActivity.getHandAdapter().notifyDataSetChanged();  
 game.useAfterPlay(this.getName(), false);  
 }  
}

Victory

package com.example.dominion\_game.cards\_classes;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.activities.GameActivity;  
import com.example.dominion\_game.classes.Card;  
import com.example.dominion\_game.classes.GameManager;  
import com.example.dominion\_game.classes.Player;  
  
public class Victory extends Card {  
 private int value;  
 public Victory(String name) {  
 super("victory");  
 switch (name) {  
 case "Estate":  
 setName("Estate");  
 setPrice(2);  
 setImageSource(R.mipmap.*estate*);  
 setShortImageSource(R.mipmap.*estate\_sh*);  
 this.value = 1;  
 break;  
 case "Duchy":  
 setName("Duchy");  
 setPrice(5);  
 setImageSource(R.mipmap.*duchy*);  
 setShortImageSource(R.mipmap.*duchy\_sh*);  
 this.value = 3;  
 break;  
 case "Province":  
 setName("Province");  
 setPrice(8);  
 setImageSource(R.mipmap.*province*);  
 setShortImageSource(R.mipmap.*province\_sh*);  
 this.value = 6;  
 break;  
 case "Curse":  
 setName("Curse");  
 setPrice(0);  
 setImageSource(R.mipmap.*curse*);  
 setShortImageSource(R.mipmap.*curse\_sh*);  
 this.value = -1;  
 break;  
 }  
 }  
 @Override  
 public void play(GameManager game, GameActivity gameActivity) {  
  
 }  
  
 @Override  
 public int getValue(GameManager gameManager) {  
 return this.value;  
 }  
}

Village

package com.example.dominion\_game.cards\_classes;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.activities.GameActivity;  
import com.example.dominion\_game.classes.Card;  
import com.example.dominion\_game.classes.GameManager;  
  
public class Village extends Card {  
 public Village() {  
 super("Village", 3, R.mipmap.*village*, R.mipmap.*village\_sh*, "action");  
 }  
 @Override  
 public void play(GameManager game, GameActivity gameActivity) {  
 game.getPlayer().takeCardsToHand(1, gameActivity, game, game.getTabs());  
 game.getTurn().addActions(2);  
 game.useAfterPlay(this.getName(), false);  
 }  
}

Witch

package com.example.dominion\_game.cards\_classes;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.activities.GameActivity;  
import com.example.dominion\_game.classes.Card;  
import com.example.dominion\_game.classes.GameManager;  
import com.example.dominion\_game.classes.LogLine;  
  
public class Witch extends Card {  
 public Witch() {  
 super("Witch", 5, R.mipmap.*witch*, R.mipmap.*witch\_sh*, "action");  
 }  
 @Override  
 public void play(GameManager game, GameActivity gameActivity) {  
 game.getPlayer().takeCardsToHand(2, gameActivity, game, game.getTabs());  
 game.useAfterPlay(this.getName(), true);  
 }  
  
 @Override  
 public void attack(GameManager game, GameActivity gameActivity) {  
 game.getPlayer().addToDiscard(game.getCard("Curse"));  
 game.getLog().add(new LogLine("Gains a Curse", game.getGameManagerBeforeStart().getMyId(), false, false,  
 game.getTabs(), "in action"));  
 game.setDoneAttack(true);  
 }  
}

Workshop

package com.example.dominion\_game.cards\_classes;  
  
import com.example.dominion\_game.R;  
import com.example.dominion\_game.activities.GameActivity;  
import com.example.dominion\_game.classes.Card;  
import com.example.dominion\_game.classes.GameManager;  
import com.example.dominion\_game.classes.Help;  
  
public class Workshop extends Card {  
 public Workshop() {  
 super("Workshop", 3, R.mipmap.*workshop*, R.mipmap.*workshop\_sh*, "action");  
 }  
  
 @Override  
 public void play(GameManager game, GameActivity gameActivity) {  
 gameActivity.waitForBoard(this.getName(), 1, 1);  
 }  
  
 @Override  
 public void clickOnBoard(String cardName, GameManager game, GameActivity gameActivity) {  
 game.getPlayer().addToDiscard(game.getCard(cardName));  
 game.getTurn().getWaitForFunction().clear();  
 gameActivity.turnUI();  
 game.getPlayer().updateArrayHand();  
 gameActivity.getHandAdapter().notifyDataSetChanged();  
 game.useAfterPlay(this.getName(), false);  
 }  
  
 @Override  
 public boolean isCardToGetFromBoard(String cardName, GameManager game, GameActivity gameActivity) {  
 return Help.*nameToCard*(cardName).getPrice() <= 4;  
 }  
}

**XMLs:**

activity\_login

<?xml version="1.0" encoding="utf-8"?>  
<androidx.constraintlayout.widget.ConstraintLayout 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:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 tools:context=".activities.LoginActivity">  
  
 <ImageView  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:scaleType="centerCrop"  
 android:src="@mipmap/img\_dominion\_background\_cut"  
 android:alpha="0.6"  
 android:contentDescription="background" />  
  
 <androidx.constraintlayout.widget.ConstraintLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent">  
  
 <EditText  
 android:id="@+id/username"  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginStart="124dp"  
 android:layout\_marginTop="24dp"  
 android:layout\_marginEnd="124dp"  
  
 android:hint="Username"  
 android:selectAllOnFocus="true"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent" />  
  
 <EditText  
 android:id="@+id/password"  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginStart="124dp"  
 android:layout\_marginTop="8dp"  
 android:layout\_marginEnd="124dp"  
  
 android:hint="Password"  
 android:imeActionLabel="Sign in"  
 android:imeOptions="actionDone"  
 android:inputType="textPassword"  
 android:selectAllOnFocus="true"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@+id/username" />  
  
 <CheckBox  
 android:id="@+id/staySignedIn"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginStart="124dp"  
 android:layout\_marginTop="8dp"  
  
 android:text="Stay Signed In"  
 android:selectAllOnFocus="true"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@+id/password"/>  
  
 <Button  
 android:id="@+id/login"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_gravity="start"  
 android:layout\_marginStart="48dp"  
 android:layout\_marginEnd="48dp"  
 android:text="Sign In"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@+id/password"  
 app:layout\_constraintVertical\_bias="0.2" />  
  
  
 <Button  
 android:id="@+id/register"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_gravity="start"  
 android:layout\_marginStart="48dp"  
 android:layout\_marginEnd="48dp"  
 android:layout\_marginBottom="64dp"  
 android:text="Register"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@+id/password"  
 app:layout\_constraintVertical\_bias="0.6" />  
  
  
 </androidx.constraintlayout.widget.ConstraintLayout>  
  
 <androidx.constraintlayout.widget.Guideline  
 android:id="@+id/guideline"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:orientation="horizontal"  
 app:layout\_constraintGuide\_percent="0.1"/>  
  
 <ImageView  
 android:id="@+id/sound"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="0dp"  
 android:src="@mipmap/sound\_on"  
 app:layout\_constraintVertical\_weight="0.5"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent"  
 app:layout\_constraintBottom\_toTopOf="@id/guideline"/>  
</androidx.constraintlayout.widget.ConstraintLayout>

activity\_register

<?xml version="1.0" encoding="utf-8"?>  
<androidx.constraintlayout.widget.ConstraintLayout 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:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 tools:context=".activities.RegisterActivity">  
  
 <ImageView  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:scaleType="centerCrop"  
 android:src="@mipmap/img\_dominion\_background\_cut"  
 android:alpha="0.6"  
 android:contentDescription="background" />  
  
 <androidx.constraintlayout.widget.ConstraintLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent">  
  
 <EditText  
 android:id="@+id/email"  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginStart="124dp"  
 android:layout\_marginTop="104dp"  
 android:layout\_marginEnd="124dp"  
  
 android:hint="Email"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintHorizontal\_bias="1.0"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent" />  
  
 <EditText  
 android:id="@+id/username"  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginStart="124dp"  
 android:layout\_marginTop="52dp"  
 android:layout\_marginEnd="124dp"  
  
 android:hint="Username"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintHorizontal\_bias="1.0"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent" />  
  
 <EditText  
 android:id="@+id/password"  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginStart="124dp"  
 android:layout\_marginTop="8dp"  
 android:layout\_marginEnd="124dp"  
  
 android:hint="Password"  
 android:inputType="textPassword"  
 android:selectAllOnFocus="true"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@+id/email" />  
  
 <EditText  
 android:id="@+id/repeat\_password"  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginStart="124dp"  
 android:layout\_marginTop="64dp"  
 android:layout\_marginEnd="124dp"  
  
 android:hint="Repeat Password"  
 android:imeOptions="actionDone"  
 android:inputType="textPassword"  
 android:selectAllOnFocus="true"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@+id/email" />  
  
 <Button  
 android:id="@+id/register"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_gravity="start"  
 android:layout\_marginStart="48dp"  
 android:layout\_marginEnd="48dp"  
 android:text="Register"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@+id/repeat\_password"  
 app:layout\_constraintVertical\_bias="0.2" />  
  
 <Button  
 android:id="@+id/back"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_gravity="start"  
 android:layout\_marginStart="48dp"  
 android:layout\_marginEnd="48dp"  
 android:text="Back"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@+id/register"  
 app:layout\_constraintVertical\_bias="0.2" />  
  
  
 </androidx.constraintlayout.widget.ConstraintLayout>  
  
 <androidx.constraintlayout.widget.Guideline  
 android:id="@+id/guideline"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:orientation="horizontal"  
 app:layout\_constraintGuide\_percent="0.1"/>  
  
 <ImageView  
 android:id="@+id/sound"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="0dp"  
 android:src="@mipmap/sound\_on"  
 app:layout\_constraintVertical\_weight="0.5"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent"  
 app:layout\_constraintBottom\_toTopOf="@id/guideline"/>  
</androidx.constraintlayout.widget.ConstraintLayout>

activity\_start

<?xml version="1.0" encoding="utf-8"?>  
<androidx.constraintlayout.widget.ConstraintLayout 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:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 tools:context=".activities.StartActivity"  
 android:orientation="vertical">  
  
 <ImageView  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:scaleType="centerCrop"  
 android:src="@mipmap/img\_dominion\_background\_cut"  
 android:alpha="0.6"  
 android:contentDescription="background" />  
  
 <LinearLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:orientation="vertical">  
 <Button  
 android:id="@+id/btnOnlineGame"  
 android:layout\_width="200dp"  
 android:layout\_height="50dp"  
 android:text="Online Game"  
 android:layout\_gravity="center"  
 android:layout\_marginTop="32dp">  
 </Button>  
  
 <Button  
 android:id="@+id/btnCardslist"  
 android:layout\_width="200dp"  
 android:layout\_height="50dp"  
 android:text="Cardslist"  
 android:layout\_gravity="center"  
 android:layout\_marginTop="8dp">  
 </Button>  
  
 <Button  
 android:id="@+id/btnLeaderboard"  
 android:layout\_width="200dp"  
 android:layout\_height="50dp"  
 android:text="Leaderboard"  
 android:layout\_gravity="center"  
 android:layout\_marginTop="8dp">  
 </Button>  
  
 <Button  
 android:id="@+id/btnInstructions"  
 android:layout\_width="200dp"  
 android:layout\_height="50dp"  
 android:text="Instructions"  
 android:layout\_gravity="center"  
 android:layout\_marginTop="8dp">  
 </Button>  
  
 <Button  
 android:id="@+id/btnSettings"  
 android:layout\_width="200dp"  
 android:layout\_height="50dp"  
 android:text="Log Out"  
 android:layout\_gravity="center"  
 android:layout\_marginTop="8dp">  
 </Button>  
 </LinearLayout>  
  
 <androidx.constraintlayout.widget.Guideline  
 android:id="@+id/guideline"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:orientation="horizontal"  
 app:layout\_constraintGuide\_percent="0.1"/>  
  
 <ImageView  
 android:id="@+id/sound"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="0dp"  
 android:src="@mipmap/sound\_on"  
 app:layout\_constraintVertical\_weight="0.5"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent"  
 app:layout\_constraintBottom\_toTopOf="@id/guideline"/>  
</androidx.constraintlayout.widget.ConstraintLayout>

activity\_online\_game

<?xml version="1.0" encoding="utf-8"?>  
<androidx.constraintlayout.widget.ConstraintLayout 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:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 tools:context="com.example.dominion\_game.activities.OnlineGameActivity"  
 android:orientation="vertical">  
  
 <ImageView  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:scaleType="centerCrop"  
 android:src="@mipmap/img\_dominion\_background\_cut"  
 android:alpha="0.6"  
 android:contentDescription="background" />  
  
 <LinearLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:orientation="vertical">  
 <Button  
 android:id="@+id/btnTables"  
 android:layout\_width="200dp"  
 android:layout\_height="50dp"  
 android:text="Online Tables"  
 android:layout\_gravity="center"  
 android:layout\_marginTop="32dp">  
 </Button>  
  
 <Button  
 android:id="@+id/btnCreateTable"  
 android:layout\_width="200dp"  
 android:layout\_height="50dp"  
 android:text="Create Table"  
 android:layout\_gravity="center"  
 android:layout\_marginTop="8dp">  
 </Button>  
 </LinearLayout>  
  
 <ImageView  
 android:id="@+id/background"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:scaleType="centerCrop"  
 android:visibility="invisible"  
 android:src="@mipmap/img\_dominion\_background\_cut"  
 android:contentDescription="background" />  
  
 <androidx.constraintlayout.widget.Guideline  
 android:id="@+id/guideline"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:orientation="horizontal"  
 app:layout\_constraintGuide\_percent="0.1"/>  
  
 <ImageView  
 android:id="@+id/sound"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="0dp"  
 android:src="@mipmap/sound\_on"  
 app:layout\_constraintVertical\_weight="0.5"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent"  
 app:layout\_constraintBottom\_toTopOf="@id/guideline"/>  
  
</androidx.constraintlayout.widget.ConstraintLayout>

activity\_online\_tables

<?xml version="1.0" encoding="utf-8"?>  
<androidx.constraintlayout.widget.ConstraintLayout 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:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 tools:context="com.example.dominion\_game.activities.OnlineTablesActivity"  
 android:orientation="vertical">  
  
 <ImageView  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:scaleType="centerCrop"  
 android:src="@mipmap/img\_dominion\_background\_cut"  
 android:alpha="0.6"  
 android:contentDescription="background" />  
  
 <LinearLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="0dp"  
 android:orientation="vertical"  
 android:weightSum="10"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@id/guideline">  
  
 <ListView  
 android:id="@+id/lvTables"  
 android:layout\_width="match\_parent"  
 android:layout\_height="0dp"  
 android:layout\_weight="7">  
  
 </ListView>  
  
 <Button  
 android:id="@+id/btnBack"  
 android:layout\_width="200dp"  
 android:layout\_height="50dp"  
 android:layout\_gravity="center"  
 android:layout\_margin="30dp"  
 android:text="Back"/>  
  
 </LinearLayout>  
  
 <ImageView  
 android:id="@+id/background"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:scaleType="centerCrop"  
 android:visibility="invisible"  
 android:src="@mipmap/img\_dominion\_background\_cut"  
 android:contentDescription="background" />  
  
 <androidx.constraintlayout.widget.Guideline  
 android:id="@+id/guideline"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:orientation="horizontal"  
 app:layout\_constraintGuide\_percent="0.1"/>  
  
 <ImageView  
 android:id="@+id/sound"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="0dp"  
 android:src="@mipmap/sound\_on"  
 app:layout\_constraintVertical\_weight="0.5"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent"  
 app:layout\_constraintBottom\_toTopOf="@id/guideline"/>  
  
</androidx.constraintlayout.widget.ConstraintLayout>

activity\_prepare\_game

<?xml version="1.0" encoding="utf-8"?>  
<androidx.constraintlayout.widget.ConstraintLayout 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:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 tools:context="com.example.dominion\_game.activities.PrepareGameActivity">  
  
 <ImageView  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:scaleType="centerCrop"  
 android:src="@mipmap/img\_dominion\_background\_cut"  
 android:alpha="0.6"  
 android:contentDescription="background" />  
   
 <LinearLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:orientation="vertical">  
 <TextView  
 android:id="@+id/tvP1"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:layout\_gravity="center"  
 android:gravity="center"  
 android:text="P1"  
 android:textColor="@color/black"  
 android:layout\_marginTop="10dp">  
  
 </TextView>  
  
 <TextView  
 android:id="@+id/tvP2"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:layout\_gravity="center"  
 android:gravity="center"  
 android:text="P2"  
 android:textColor="@color/black"  
 android:layout\_marginTop="10dp">  
  
 </TextView>  
  
 <Button  
 android:id="@+id/btnReady"  
 android:layout\_width="200dp"  
 android:layout\_height="50dp"  
 android:text="Ready"  
 android:layout\_gravity="center"  
 android:layout\_marginTop="32dp">  
 </Button>  
  
 <Button  
 android:id="@+id/btnLeaveTable"  
 android:layout\_width="200dp"  
 android:layout\_height="50dp"  
 android:text="Leave Table"  
 android:layout\_gravity="center"  
 android:layout\_marginTop="8dp">  
 </Button>  
  
 <TextView  
 android:id="@+id/tvIsRated"  
 android:layout\_width="200dp"  
 android:layout\_height="wrap\_content"  
 android:text="IsRated"  
 android:layout\_gravity="center"  
 android:gravity="center"  
 android:textColor="@color/black"  
 android:layout\_marginTop="8dp">  
 </TextView>  
  
 <Switch  
 android:id="@+id/sw"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:checked="false"  
 android:layout\_gravity="center"  
 android:layout\_marginTop="8dp">  
  
 </Switch>  
  
 <TextView  
 android:id="@+id/tvWinner"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:layout\_gravity="center"  
 android:gravity="center"  
 android:text="Winner"  
 android:textColor="@color/black"  
 android:layout\_marginTop="10dp">  
  
 </TextView>  
  
 <TextView  
 android:id="@+id/tvP1VP"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:layout\_gravity="center"  
 android:gravity="center"  
 android:text="P1 VP"  
 android:textColor="@color/black"  
 android:layout\_marginTop="10dp">  
  
 </TextView>  
  
 <TextView  
 android:id="@+id/tvP2VP"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:layout\_gravity="center"  
 android:gravity="center"  
 android:text="P2 VP"  
 android:textColor="@color/black"  
 android:layout\_marginTop="10dp">  
  
 </TextView>  
 </LinearLayout>  
  
 <ImageView  
 android:id="@+id/background"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:scaleType="centerCrop"  
 android:visibility="invisible"  
 android:src="@mipmap/img\_dominion\_background\_cut"  
 android:contentDescription="background" />  
  
 <androidx.constraintlayout.widget.Guideline  
 android:id="@+id/guideline"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:orientation="horizontal"  
 app:layout\_constraintGuide\_percent="0.1"/>  
  
 <ImageView  
 android:id="@+id/sound"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="0dp"  
 android:src="@mipmap/sound\_on"  
 app:layout\_constraintVertical\_weight="0.5"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent"  
 app:layout\_constraintBottom\_toTopOf="@id/guideline"/>  
   
</androidx.constraintlayout.widget.ConstraintLayout>

activity\_game\_designed

<?xml version="1.0" encoding="utf-8"?>  
<RelativeLayout 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:orientation="horizontal" android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:background="@android:color/black">  
  
 <ImageView  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:scaleType="centerCrop"  
 android:src="@mipmap/img\_dominion\_background\_cut"  
 android:alpha="0.6"  
 android:layout\_alignParentTop="true"  
 android:contentDescription="background" />  
  
 <LinearLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:id="@+id/llGamePanel"  
 android:orientation="horizontal"  
 android:baselineAligned="false">  
  
 <androidx.constraintlayout.widget.ConstraintLayout  
 android:layout\_width="0dp"  
 android:layout\_height="match\_parent"  
 android:id="@+id/clLeft"  
 android:orientation="vertical"  
 android:layout\_weight="13"  
 >  
  
 <androidx.constraintlayout.widget.ConstraintLayout  
 android:id="@+id/clBasicCards"  
 android:layout\_width="match\_parent"  
 android:layout\_height="0dp"  
 android:layout\_gravity="center"  
 android:orientation="horizontal"  
 android:padding="5dp"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent"  
 app:layout\_constraintBottom\_toTopOf="@+id/clMyDeckAndDiscard"  
 app:layout\_constraintVertical\_weight="4">  
  
 <androidx.constraintlayout.widget.ConstraintLayout  
 android:id="@+id/clVictoryCards"  
 android:layout\_width="0dp"  
 android:layout\_height="match\_parent"  
 android:orientation="vertical"  
 android:padding="1dp"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toStartOf="@+id/clTreasureCards"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent">  
  
 <include  
 android:id="@+id/province"  
 layout="@layout/card\_sh"  
 android:layout\_width="match\_parent"  
 android:layout\_height="0dp"  
 app:layout\_constraintBottom\_toTopOf="@+id/duchy"  
 app:layout\_constraintDimensionRatio="w,436:459"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent" />  
  
 <include  
 android:id="@+id/duchy"  
 layout="@layout/card\_sh"  
 android:layout\_width="match\_parent"  
 android:layout\_height="0dp"  
 app:layout\_constraintBottom\_toTopOf="@+id/estate"  
 app:layout\_constraintDimensionRatio="w,436:459"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@+id/province" />  
  
 <include  
 android:id="@+id/estate"  
 layout="@layout/card\_sh"  
 android:layout\_width="match\_parent"  
 android:layout\_height="0dp"  
 app:layout\_constraintBottom\_toTopOf="@+id/curse"  
 app:layout\_constraintDimensionRatio="w,436:459"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@+id/duchy" />  
  
 <include  
 android:id="@+id/curse"  
 layout="@layout/card\_sh"  
 android:layout\_width="match\_parent"  
 android:layout\_height="0dp"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintDimensionRatio="w,436:459"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@+id/estate" />  
  
 </androidx.constraintlayout.widget.ConstraintLayout>  
  
 <androidx.constraintlayout.widget.ConstraintLayout  
 android:id="@+id/clTreasureCards"  
 android:layout\_width="0dp"  
 android:layout\_height="match\_parent"  
 android:orientation="vertical"  
 android:padding="1dp"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintStart\_toEndOf="@+id/clVictoryCards"  
 app:layout\_constraintTop\_toTopOf="parent">  
  
 <include  
 android:id="@+id/gold"  
 layout="@layout/card\_sh"  
 android:layout\_width="match\_parent"  
 android:layout\_height="0dp"  
 app:layout\_constraintBottom\_toTopOf="@+id/silver"  
 app:layout\_constraintDimensionRatio="w,436:459"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent" />  
  
 <include  
 android:id="@+id/silver"  
 layout="@layout/card\_sh"  
 android:layout\_width="match\_parent"  
 android:layout\_height="0dp"  
 app:layout\_constraintBottom\_toTopOf="@+id/copper"  
 app:layout\_constraintDimensionRatio="w,436:459"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@+id/gold" />  
  
 <include  
 android:id="@+id/copper"  
 layout="@layout/card\_sh"  
 android:layout\_width="match\_parent"  
 android:layout\_height="0dp"  
 app:layout\_constraintBottom\_toTopOf="@+id/include"  
 app:layout\_constraintDimensionRatio="w,436:459"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@+id/silver" />  
  
 <include  
 android:id="@+id/include"  
 layout="@layout/card\_sh"  
 android:layout\_width="match\_parent"  
 android:layout\_height="0dp"  
 android:visibility="invisible"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintDimensionRatio="w,436:459"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@+id/copper" />  
  
 </androidx.constraintlayout.widget.ConstraintLayout>  
  
 </androidx.constraintlayout.widget.ConstraintLayout>  
  
 <androidx.constraintlayout.widget.ConstraintLayout  
 android:id="@+id/clMyDeckAndDiscard"  
 android:layout\_width="match\_parent"  
 android:layout\_height="0dp"  
 android:orientation="vertical"  
 android:padding="5dp"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@id/clBasicCards"  
 app:layout\_constraintVertical\_weight="2">  
  
 <TextView  
 android:id="@+id/tvMyName"  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content"  
 android:gravity="center"  
 android:maxLength="10"  
 android:maxLines="1"  
 android:text="My Name"  
 android:textColor="@color/white"  
 app:autoSizeTextType="uniform"  
 app:layout\_constraintBottom\_toTopOf="@+id/tvMyVP"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent" />  
  
 <TextView  
 android:id="@+id/tvMyVP"  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content"  
 android:layout\_alignParentBottom="true"  
 android:gravity="center"  
 android:maxLines="1"  
 android:text="0 VP"  
 android:textColor="@color/colorVP"  
 app:autoSizeTextType="uniform"  
 app:layout\_constraintBottom\_toTopOf="@+id/clMyInfo"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent" />  
  
 <androidx.constraintlayout.widget.ConstraintLayout  
 android:id="@+id/clMyInfo"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:orientation="horizontal"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 android:padding="5dp">  
  
 <ImageView  
 android:id="@+id/ivMyDiscard"  
 android:layout\_width="0dp"  
 android:layout\_height="0dp"  
 android:src="@mipmap/cardback\_sh"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintDimensionRatio="w,436:459"  
 app:layout\_constraintEnd\_toStartOf="@+id/myDeck"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent" />  
  
 <include  
 android:id="@+id/myDeck"  
 layout="@layout/card\_sh"  
 android:layout\_width="0dp"  
 android:layout\_height="0dp"  
 android:layout\_marginStart="3dp"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintDimensionRatio="w,436:459"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintStart\_toEndOf="@+id/ivMyDiscard"  
 app:layout\_constraintTop\_toTopOf="parent" />  
 </androidx.constraintlayout.widget.ConstraintLayout>  
  
 </androidx.constraintlayout.widget.ConstraintLayout>  
  
 </androidx.constraintlayout.widget.ConstraintLayout>  
  
 <androidx.constraintlayout.widget.ConstraintLayout  
 android:id="@+id/clPlayArea"  
 android:layout\_width="0dp"  
 android:layout\_height="match\_parent"  
 android:layout\_weight="55"  
 android:orientation="vertical"  
 android:visibility="visible">  
  
 <RelativeLayout  
 android:id="@+id/rlButtonsInPlay"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="0dp"  
 android:layout\_gravity="center"  
 app:layout\_constraintBottom\_toTopOf="@+id/clTurn"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@+id/clActionCards"  
 app:layout\_constraintVertical\_weight="1">  
  
 <Button  
 android:id="@+id/btnStart"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Start Game"  
 app:autoSizeTextType="uniform"  
 android:autoSizeMinTextSize="1sp"  
 android:autoSizeMaxTextSize="40sp"  
 android:autoSizeStepGranularity="1sp"/>  
  
 <Button  
 android:id="@+id/btnEnd"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_toEndOf="@+id/btnStart"  
 android:text="End Actions"  
 android:visibility="gone"  
 app:autoSizeTextType="uniform"  
 android:autoSizeMinTextSize="1sp"  
 android:autoSizeMaxTextSize="40sp"  
 android:autoSizeStepGranularity="1sp"/>  
  
 <Button  
 android:id="@+id/btnAutoplay"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_toEndOf="@+id/btnEnd"  
 android:text="Autoplay Treasure"  
 android:visibility="gone"  
 app:autoSizeTextType="uniform"  
 android:autoSizeMinTextSize="1sp"  
 android:autoSizeMaxTextSize="40sp"  
 android:autoSizeStepGranularity="1sp"/>  
  
 <Button  
 android:id="@+id/btnAction1"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Action 1"  
 android:visibility="gone"  
 app:autoSizeTextType="uniform"  
 android:autoSizeMinTextSize="1sp"  
 android:autoSizeMaxTextSize="40sp"  
 android:autoSizeStepGranularity="1sp"/>  
  
 <Button  
 android:id="@+id/btnAction2"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_toEndOf="@+id/btnAction1"  
 android:text="Action 2"  
 android:visibility="gone"  
 app:autoSizeTextType="uniform"  
 android:autoSizeMinTextSize="1sp"  
 android:autoSizeMaxTextSize="40sp"  
 android:autoSizeStepGranularity="1sp"/>  
  
 <Button  
 android:id="@+id/btnAction3"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_toEndOf="@+id/btnAction2"  
 android:text="Action 3"  
 android:visibility="gone"  
 app:autoSizeTextType="uniform"  
 android:autoSizeMinTextSize="1sp"  
 android:autoSizeMaxTextSize="40sp"  
 android:autoSizeStepGranularity="1sp"/>  
  
 </RelativeLayout>  
  
 <androidx.constraintlayout.widget.ConstraintLayout  
 android:id="@+id/clActionCards"  
 android:layout\_width="match\_parent"  
 android:layout\_height="0dp"  
 android:layout\_gravity="center"  
 android:orientation="vertical"  
 android:padding="5dp"  
 app:layout\_constraintBottom\_toTopOf="@+id/rlButtonsInPlay"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@+id/clEnemyArea"  
 app:layout\_constraintVertical\_weight="3.25">  
  
 <androidx.constraintlayout.widget.ConstraintLayout  
 android:id="@+id/constraintLayout2"  
 android:layout\_width="match\_parent"  
 android:layout\_height="0dp"  
 android:layout\_marginBottom="5dp"  
 android:orientation="horizontal"  
 app:layout\_constraintBottom\_toTopOf="@+id/constraintLayout"  
 app:layout\_constraintTop\_toTopOf="parent"  
 app:layout\_constraintVertical\_chainStyle="packed"  
 app:layout\_constraintVertical\_weight="1">  
  
 <include  
 android:id="@+id/action6"  
 layout="@layout/card\_sh"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="match\_parent"  
 app:layout\_constraintEnd\_toStartOf="@+id/action7"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintHorizontal\_chainStyle="packed"  
 app:layout\_constraintStart\_toStartOf="parent"  
 tools:layout\_editor\_absoluteY="1dp" />  
  
 <include  
 android:id="@+id/action7"  
 layout="@layout/card\_sh"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="match\_parent"  
 android:layout\_marginStart="5dp"  
 app:layout\_constraintEnd\_toStartOf="@+id/action8"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintStart\_toEndOf="@+id/action6"  
 tools:layout\_editor\_absoluteY="1dp" />  
  
 <include  
 android:id="@+id/action8"  
 layout="@layout/card\_sh"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="match\_parent"  
 android:layout\_marginStart="5dp"  
 app:layout\_constraintEnd\_toStartOf="@+id/action9"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintStart\_toEndOf="@+id/action7"  
 tools:layout\_editor\_absoluteY="1dp" />  
  
 <include  
 android:id="@+id/action9"  
 layout="@layout/card\_sh"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="match\_parent"  
 android:layout\_marginStart="5dp"  
 app:layout\_constraintEnd\_toStartOf="@+id/action10"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintStart\_toEndOf="@+id/action8"  
 tools:layout\_editor\_absoluteY="1dp" />  
  
 <include  
 android:id="@+id/action10"  
 layout="@layout/card\_sh"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="match\_parent"  
 android:layout\_marginStart="5dp"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintStart\_toEndOf="@+id/action9"  
 tools:layout\_editor\_absoluteY="1dp" />  
  
 </androidx.constraintlayout.widget.ConstraintLayout>  
  
 <androidx.constraintlayout.widget.ConstraintLayout  
 android:id="@+id/constraintLayout"  
 android:layout\_width="match\_parent"  
 android:layout\_height="0dp"  
 android:orientation="horizontal"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@+id/constraintLayout2"  
 app:layout\_constraintVertical\_weight="1">  
  
 <include  
 android:id="@+id/action1"  
 layout="@layout/card\_sh"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="match\_parent"  
 app:layout\_constraintEnd\_toStartOf="@+id/action2"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintHorizontal\_chainStyle="packed"  
 app:layout\_constraintStart\_toStartOf="parent"  
 tools:layout\_editor\_absoluteY="1dp" />  
  
 <include  
 android:id="@+id/action2"  
 layout="@layout/card\_sh"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="match\_parent"  
 android:layout\_marginStart="5dp"  
 app:layout\_constraintEnd\_toStartOf="@+id/action3"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintStart\_toEndOf="@+id/action1"  
 tools:layout\_editor\_absoluteY="1dp" />  
  
 <include  
 android:id="@+id/action3"  
 layout="@layout/card\_sh"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="match\_parent"  
 android:layout\_marginStart="5dp"  
 app:layout\_constraintEnd\_toStartOf="@+id/action4"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintStart\_toEndOf="@+id/action2"  
 tools:layout\_editor\_absoluteY="1dp" />  
  
 <include  
 android:id="@+id/action4"  
 layout="@layout/card\_sh"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="match\_parent"  
 android:layout\_marginStart="5dp"  
 app:layout\_constraintEnd\_toStartOf="@+id/action5"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintStart\_toEndOf="@+id/action3"  
 tools:layout\_editor\_absoluteY="1dp" />  
  
 <include  
 android:id="@+id/action5"  
 layout="@layout/card\_sh"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="match\_parent"  
 android:layout\_marginStart="5dp"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintStart\_toEndOf="@+id/action4"  
 tools:layout\_editor\_absoluteY="1dp" />  
  
  
 </androidx.constraintlayout.widget.ConstraintLayout>  
  
 </androidx.constraintlayout.widget.ConstraintLayout>  
  
 <androidx.constraintlayout.widget.ConstraintLayout  
 android:id="@+id/clEnemyArea"  
 android:layout\_width="match\_parent"  
 android:layout\_height="0dp"  
 android:padding="5dp"  
 app:layout\_constraintBottom\_toTopOf="@+id/clActionCards"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent"  
 app:layout\_constraintVertical\_chainStyle="packed"  
 app:layout\_constraintVertical\_weight="1.5">  
  
 <ImageView  
 android:id="@+id/ivEnemyDiscard"  
 android:layout\_width="0dp"  
 android:layout\_height="match\_parent"  
 android:contentDescription="ivEnemyDiscard"  
 android:src="@mipmap/cardback\_sh"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintDimensionRatio="w,436:459"  
 app:layout\_constraintEnd\_toStartOf="@+id/enemyDeck"  
 app:layout\_constraintHorizontal\_chainStyle="packed"  
 app:layout\_constraintStart\_toEndOf="@id/guidelineDeckAndDiscard"  
 app:layout\_constraintTop\_toTopOf="parent" />  
  
 <LinearLayout  
 android:id="@+id/clEnemyInfo"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:orientation="vertical"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toStartOf="@+id/guidelineInfo"  
 app:layout\_constraintHorizontal\_weight="0"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent">  
  
 <TextView  
 android:id="@+id/tvEnemyName"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:layout\_gravity="center"  
 android:gravity="center"  
 android:maxLength="10"  
 android:maxLines="1"  
 android:text="Enemy Name"  
 android:textColor="@color/white"  
 app:autoSizeTextType="uniform" />  
  
 <TextView  
 android:id="@+id/tvEnemyVP"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:layout\_gravity="center"  
 android:gravity="center"  
 android:maxLines="1"  
 android:text="0 VP"  
 android:textColor="@color/colorVP"  
 app:autoSizeTextType="uniform" />  
 </LinearLayout>  
  
  
 <androidx.constraintlayout.widget.Guideline  
 android:id="@+id/guidelineInfo"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:orientation="vertical"  
 app:layout\_constraintGuide\_percent="0.25" />  
  
 <androidx.constraintlayout.widget.Guideline  
 android:id="@+id/guidelineDeckAndDiscard"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:orientation="vertical"  
 app:layout\_constraintGuide\_percent="0.27" />  
  
 <include  
 android:id="@+id/enemyDeck"  
 layout="@layout/card\_sh"  
 android:layout\_width="0dp"  
 android:layout\_height="match\_parent"  
 android:layout\_marginStart="2dp"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintDimensionRatio="w,436:459"  
 app:layout\_constraintStart\_toEndOf="@+id/ivEnemyDiscard"  
 app:layout\_constraintTop\_toTopOf="parent" />  
  
 <include  
 android:id="@+id/enemyHand"  
 layout="@layout/card\_sh"  
 android:layout\_width="0dp"  
 android:layout\_height="match\_parent"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintDimensionRatio="w,436:459"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toEndOf="@+id/enemyDeck"  
 app:layout\_constraintTop\_toTopOf="parent" />  
  
 </androidx.constraintlayout.widget.ConstraintLayout>  
  
 <androidx.constraintlayout.widget.ConstraintLayout  
 android:id="@+id/clTurn"  
 android:layout\_width="match\_parent"  
 android:layout\_height="0dp"  
 android:layout\_gravity="center\_vertical"  
 android:alpha="0.8"  
 android:background="@drawable/rounded\_corners"  
 android:orientation="vertical"  
 android:padding="5dp"  
 android:layout\_marginEnd="10dp"  
 android:layout\_marginStart="10dp"  
 app:layout\_constraintBottom\_toTopOf="@+id/rvHand"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@+id/rlButtonsInPlay"  
 app:layout\_constraintVertical\_weight="0.75">  
  
 <TextView  
 android:id="@+id/tvTurn"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:gravity="center"  
 android:singleLine="true"  
 android:text="Turn"  
 android:textColor="@color/white"  
 app:autoSizeTextType="uniform"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent" />  
 </androidx.constraintlayout.widget.ConstraintLayout>  
  
 <androidx.recyclerview.widget.RecyclerView  
 android:id="@+id/rvHand"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="0dp"  
 android:layout\_gravity="center"  
 android:scrollbars="horizontal"  
 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/clTurn"  
 app:layout\_constraintVertical\_weight="3" />  
  
 <androidx.constraintlayout.widget.ConstraintLayout  
 android:id="@+id/clActionCardsPlaying"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="0dp"  
 android:layout\_gravity="center"  
 android:src="@color/white"  
 android:visibility="invisible"  
 app:layout\_constraintBottom\_toTopOf="@id/rlButtonsInPlay"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintHeight\_percent="0.35"  
 app:layout\_constraintStart\_toStartOf="parent">  
  
 <androidx.recyclerview.widget.RecyclerView  
 android:id="@+id/rvActionCardsPlaying"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:scrollbars="horizontal"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent" />  
  
 <ImageView  
 android:id="@+id/ivArrowActionCardsPlaying"  
 android:layout\_width="0dp"  
 android:layout\_height="0dp"  
 android:src="@mipmap/arrow\_down"  
 app:layout\_constraintTop\_toTopOf="parent"  
 app:layout\_constraintDimensionRatio="h,1:1"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintHeight\_percent="0.2" />  
  
 </androidx.constraintlayout.widget.ConstraintLayout>  
  
  
 </androidx.constraintlayout.widget.ConstraintLayout>  
  
 <androidx.constraintlayout.widget.ConstraintLayout  
 android:id="@+id/clKingdom"  
 android:layout\_width="0dp"  
 android:layout\_height="match\_parent"  
 android:layout\_weight="55"  
 android:orientation="vertical"  
 android:visibility="gone">  
  
 <androidx.constraintlayout.widget.ConstraintLayout  
 android:id="@+id/constraintLayout3"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginBottom="5dp"  
 android:orientation="horizontal"  
 app:layout\_constraintBottom\_toTopOf="@+id/constraintLayout4"  
 app:layout\_constraintTop\_toTopOf="parent"  
 app:layout\_constraintVertical\_chainStyle="packed">  
  
 <include  
 android:id="@+id/actionBig6"  
 layout="@layout/card\_kingdom"  
 android:layout\_width="0dp"  
 android:layout\_height="0dp"  
 android:layout\_marginStart="5dp"  
 app:layout\_constraintDimensionRatio="h,436:685"  
 app:layout\_constraintEnd\_toStartOf="@+id/actionBig7"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintStart\_toStartOf="parent" />  
  
 <include  
 android:id="@+id/actionBig7"  
 layout="@layout/card\_kingdom"  
 android:layout\_width="0dp"  
 android:layout\_height="0dp"  
 android:layout\_marginStart="5dp"  
 app:layout\_constraintDimensionRatio="h,436:685"  
 app:layout\_constraintEnd\_toStartOf="@+id/actionBig8"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintStart\_toEndOf="@+id/actionBig6" />  
  
 <include  
 android:id="@+id/actionBig8"  
 layout="@layout/card\_kingdom"  
 android:layout\_width="0dp"  
 android:layout\_height="0dp"  
 android:layout\_marginStart="5dp"  
 app:layout\_constraintDimensionRatio="h,436:685"  
 app:layout\_constraintEnd\_toStartOf="@+id/actionBig9"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintStart\_toEndOf="@+id/actionBig7" />  
  
 <include  
 android:id="@+id/actionBig9"  
 layout="@layout/card\_kingdom"  
 android:layout\_width="0dp"  
 android:layout\_height="0dp"  
 android:layout\_marginStart="5dp"  
 app:layout\_constraintDimensionRatio="h,436:685"  
 app:layout\_constraintEnd\_toStartOf="@+id/actionBig10"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintStart\_toEndOf="@+id/actionBig8" />  
  
 <include  
 android:id="@+id/actionBig10"  
 layout="@layout/card\_kingdom"  
 android:layout\_width="0dp"  
 android:layout\_height="0dp"  
 android:layout\_marginStart="5dp"  
 android:layout\_marginEnd="5dp"  
 app:layout\_constraintDimensionRatio="h,436:685"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintStart\_toEndOf="@+id/actionBig9" />  
  
 </androidx.constraintlayout.widget.ConstraintLayout>  
  
 <androidx.constraintlayout.widget.ConstraintLayout  
 android:id="@+id/constraintLayout4"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:orientation="horizontal"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@+id/constraintLayout3">  
  
 <include  
 android:id="@+id/actionBig1"  
 layout="@layout/card\_kingdom"  
 android:layout\_width="0dp"  
 android:layout\_height="0dp"  
 android:layout\_marginStart="5dp"  
 app:layout\_constraintDimensionRatio="h,436:685"  
 app:layout\_constraintEnd\_toStartOf="@+id/actionBig2"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintStart\_toStartOf="parent" />  
  
 <include  
 android:id="@+id/actionBig2"  
 layout="@layout/card\_kingdom"  
 android:layout\_width="0dp"  
 android:layout\_height="0dp"  
 android:layout\_marginStart="5dp"  
 app:layout\_constraintDimensionRatio="h,436:685"  
 app:layout\_constraintEnd\_toStartOf="@+id/actionBig3"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintStart\_toEndOf="@+id/actionBig1" />  
  
 <include  
 android:id="@+id/actionBig3"  
 layout="@layout/card\_kingdom"  
 android:layout\_width="0dp"  
 android:layout\_height="0dp"  
 android:layout\_marginStart="5dp"  
 app:layout\_constraintDimensionRatio="h,436:685"  
 app:layout\_constraintEnd\_toStartOf="@+id/actionBig4"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintStart\_toEndOf="@+id/actionBig2" />  
  
 <include  
 android:id="@+id/actionBig4"  
 layout="@layout/card\_kingdom"  
 android:layout\_width="0dp"  
 android:layout\_height="0dp"  
 android:layout\_marginStart="5dp"  
 app:layout\_constraintDimensionRatio="h,436:685"  
 app:layout\_constraintEnd\_toStartOf="@+id/actionBig5"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintStart\_toEndOf="@+id/actionBig3" />  
  
 <include  
 android:id="@+id/actionBig5"  
 layout="@layout/card\_kingdom"  
 android:layout\_width="0dp"  
 android:layout\_height="0dp"  
 android:layout\_marginStart="5dp"  
 android:layout\_marginEnd="5dp"  
 app:layout\_constraintDimensionRatio="h,436:685"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintStart\_toEndOf="@+id/actionBig4" />  
  
  
 </androidx.constraintlayout.widget.ConstraintLayout>  
  
 </androidx.constraintlayout.widget.ConstraintLayout>  
  
 <androidx.constraintlayout.widget.ConstraintLayout  
 android:id="@+id/clRight"  
 android:layout\_width="0dp"  
 android:layout\_height="match\_parent"  
 android:layout\_weight="25"  
 android:alpha="0.8"  
 android:background="@android:color/black"  
 android:orientation="vertical">  
  
 <RelativeLayout  
 android:id="@+id/relativeLayout"  
 android:layout\_width="match\_parent"  
 android:layout\_height="0dp"  
 android:orientation="vertical"  
 android:padding="5dp"  
 android:visibility="visible"  
 app:layout\_constraintBottom\_toTopOf="@+id/llMessages"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@+id/tvInfoTitle"  
 app:layout\_constraintVertical\_weight="3.25">  
  
 <ListView  
 android:id="@+id/lvLog"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:orientation="vertical"  
 android:visibility="visible">  
  
 </ListView>  
  
 <androidx.recyclerview.widget.RecyclerView  
 android:id="@+id/rvTrash"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:orientation="vertical"  
 android:visibility="gone">  
  
 </androidx.recyclerview.widget.RecyclerView>  
  
 </RelativeLayout>  
  
 <TextView  
 android:id="@+id/tvInfoTitle"  
 android:layout\_width="match\_parent"  
 android:layout\_height="0dp"  
 android:gravity="center"  
 android:padding="5dp"  
 android:text="Log"  
 android:textColor="@color/white"  
 app:layout\_constraintBottom\_toTopOf="@+id/relativeLayout"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@+id/clButtons"  
 app:layout\_constraintVertical\_weight="1" />  
  
 <androidx.constraintlayout.widget.ConstraintLayout  
 android:id="@+id/llMessages"  
 android:layout\_width="match\_parent"  
 android:layout\_height="0dp"  
 android:orientation="vertical"  
 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/relativeLayout"  
 app:layout\_constraintVertical\_weight="3.25">  
  
 <EditText  
 android:id="@+id/etWriteMessage"  
 android:layout\_width="match\_parent"  
 android:layout\_height="0dp"  
 android:hint="Enter Message"  
 style="@style/Widget.AppCompat.EditText"  
 android:inputType="text"  
 android:textColor="@color/white"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@+id/scrollView"  
 app:layout\_constraintVertical\_weight="3" />  
  
 <ScrollView  
 android:id="@+id/scrollView"  
 android:layout\_width="match\_parent"  
 android:layout\_height="0dp"  
 android:orientation="vertical"  
 android:padding="5dp"  
 app:layout\_constraintBottom\_toTopOf="@+id/etWriteMessage"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent"  
 app:layout\_constraintVertical\_weight="7">  
  
 <TextView  
 android:id="@+id/tvMessages"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:text="Messages"  
 android:textColor="@color/white"  
 android:textSize="15sp" />  
 </ScrollView>  
  
 </androidx.constraintlayout.widget.ConstraintLayout>  
  
 <androidx.constraintlayout.widget.ConstraintLayout  
 android:id="@+id/clButtons"  
 android:layout\_width="match\_parent"  
 android:layout\_height="0dp"  
 android:layout\_gravity="center"  
 android:orientation="horizontal"  
 android:paddingLeft="5dp"  
 android:paddingRight="5dp"  
 app:layout\_constraintBottom\_toTopOf="@+id/tvInfoTitle"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent"  
 app:layout\_constraintVertical\_weight="0.75">  
  
 <Button  
 android:id="@+id/btnKingdomOrPlayArea"  
 android:layout\_width="0dp"  
 android:layout\_height="match\_parent"  
 app:layout\_constraintHorizontal\_weight="1"  
 android:gravity="center"  
 android:singleLine="true"  
 android:text="Kingdom"  
 android:textAlignment="center"  
 android:textColor="@color/black"  
 android:autoSizeTextType="uniform"  
 android:autoSizeMinTextSize="1sp"  
 android:autoSizeMaxTextSize="40sp"  
 android:autoSizeStepGranularity="1sp"  
 app:layout\_constraintEnd\_toStartOf="@+id/btnTrashOrLog"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintStart\_toStartOf="parent" />  
  
 <Button  
 android:id="@+id/btnTrashOrLog"  
 android:layout\_width="0dp"  
 android:layout\_height="match\_parent"  
 app:layout\_constraintHorizontal\_weight="1"  
 android:gravity="center"  
 android:singleLine="true"  
 android:text="Trash"  
 android:autoSizeTextType="uniform"  
 android:autoSizeMinTextSize="1sp"  
 android:autoSizeMaxTextSize="40sp"  
 android:autoSizeStepGranularity="1sp"  
 app:layout\_constraintEnd\_toStartOf="@+id/btnResign"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintStart\_toEndOf="@+id/btnKingdomOrPlayArea" />  
  
 <Button  
 android:id="@+id/btnResign"  
 android:layout\_width="0dp"  
 android:layout\_height="match\_parent"  
 app:layout\_constraintHorizontal\_weight="1"  
 android:gravity="center"  
 android:singleLine="true"  
 android:text="resign"  
 android:autoSizeTextType="uniform"  
 android:autoSizeMinTextSize="1sp"  
 android:autoSizeMaxTextSize="40sp"  
 android:autoSizeStepGranularity="1sp"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintStart\_toEndOf="@+id/btnTrashOrLog" />  
  
 </androidx.constraintlayout.widget.ConstraintLayout>  
  
 </androidx.constraintlayout.widget.ConstraintLayout>  
  
 </LinearLayout>  
  
 <ImageView  
 android:id="@+id/background"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:scaleType="centerCrop"  
 android:visibility="invisible"  
 android:src="@mipmap/img\_dominion\_background\_cut"  
 android:layout\_alignParentTop="true"  
 android:contentDescription="background" />  
  
</RelativeLayout>

card

<?xml version="1.0" encoding="utf-8"?>  
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 android:orientation="vertical"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="match\_parent"  
 android:padding="5dp">  
  
 <ImageView  
 android:id="@+id/ivAction"  
 android:layout\_width="0dp"  
 android:layout\_height="match\_parent"  
 android:src="@mipmap/cardback"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintDimensionRatio="w,436:685"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent" />  
  
 <TextView  
 android:id="@+id/countAction"  
 android:layout\_width="0dp"  
 android:layout\_height="0dp"  
 android:background="@mipmap/counter"  
 android:gravity="center"  
 android:maxLength="2"  
 android:singleLine="true"  
 android:text="0"  
 android:autoSizeTextType="uniform"  
 android:autoSizeMinTextSize="1sp"  
 android:autoSizeMaxTextSize="40sp"  
 android:autoSizeStepGranularity="1sp"  
 android:includeFontPadding="false"  
 android:textColor="@color/white"  
 app:layout\_constraintDimensionRatio="w, 1:1"  
 app:layout\_constraintBottom\_toTopOf="@+id/guidelineActionsHorizontal"  
 app:layout\_constraintStart\_toStartOf="@+id/ivAction"  
 app:layout\_constraintTop\_toTopOf="@+id/ivAction" />  
  
 <ImageView  
 android:id="@+id/ivGreenMargin"  
 android:layout\_width="0dp"  
 android:layout\_height="match\_parent"  
 android:src="@mipmap/green\_margin"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintDimensionRatio="w,436:685"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent"  
 android:visibility="invisible"/>  
  
 <ImageView  
 android:id="@+id/ivRedMargin"  
 android:layout\_width="0dp"  
 android:layout\_height="match\_parent"  
 android:src="@mipmap/red\_margin\_for\_trash"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintDimensionRatio="w,436:685"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent"  
 android:visibility="invisible"/>  
  
 <ImageView  
 android:id="@+id/ivYellowMargin"  
 android:layout\_width="0dp"  
 android:layout\_height="match\_parent"  
 android:src="@mipmap/yellow\_margin\_for\_discard"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintDimensionRatio="w,436:685"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent"  
 android:visibility="invisible"/>  
  
 <ImageView  
 android:id="@+id/ivGreenX"  
 android:layout\_width="0dp"  
 android:layout\_height="match\_parent"  
 android:src="@mipmap/green\_x"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintDimensionRatio="w,436:685"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent"  
 android:visibility="invisible"/>  
  
 <ImageView  
 android:id="@+id/ivRedX"  
 android:layout\_width="0dp"  
 android:layout\_height="match\_parent"  
 android:src="@mipmap/red\_x\_for\_trash"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintDimensionRatio="w,436:685"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent"  
 android:visibility="invisible"/>  
  
 <ImageView  
 android:id="@+id/ivYellowX"  
 android:layout\_width="0dp"  
 android:layout\_height="match\_parent"  
 android:src="@mipmap/yellow\_x\_for\_discard"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintDimensionRatio="w,436:685"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent"  
 android:visibility="invisible"/>  
  
 <TextView  
 android:id="@+id/countActionForPlayAction"  
 android:layout\_width="0dp"  
 android:layout\_height="0dp"  
 android:background="@mipmap/red\_counter\_for\_action"  
 android:gravity="center"  
 android:maxLength="2"  
 android:singleLine="true"  
 android:text="0"  
 android:autoSizeTextType="uniform"  
 android:autoSizeMinTextSize="1sp"  
 android:autoSizeMaxTextSize="40sp"  
 android:autoSizeStepGranularity="1sp"  
 android:includeFontPadding="false"  
 android:textColor="@color/white"  
 app:layout\_constraintBottom\_toTopOf="@+id/guidelineXHorizontalEnd"  
 app:layout\_constraintTop\_toBottomOf="@+id/guidelineXHorizontalStart"  
 app:layout\_constraintStart\_toStartOf="@+id/guidelineXVerticalStart"  
 app:layout\_constraintEnd\_toEndOf="@+id/guidelineXVerticalEnd"  
 android:visibility="invisible"/>  
  
 <androidx.constraintlayout.widget.Guideline  
 android:id="@+id/guidelineActionsHorizontal"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:orientation="horizontal"  
 app:layout\_constraintGuide\_percent="0.15" />  
  
 <androidx.constraintlayout.widget.Guideline  
 android:id="@+id/guidelineXHorizontalStart"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:orientation="horizontal"  
 app:layout\_constraintGuide\_percent="0.41" />  
  
 <androidx.constraintlayout.widget.Guideline  
 android:id="@+id/guidelineXHorizontalEnd"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:orientation="horizontal"  
 app:layout\_constraintGuide\_percent="0.56" />  
  
 <androidx.constraintlayout.widget.Guideline  
 android:id="@+id/guidelineXVerticalStart"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:orientation="vertical"  
 app:layout\_constraintGuide\_percent="0.41" />  
  
 <androidx.constraintlayout.widget.Guideline  
 android:id="@+id/guidelineXVerticalEnd"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:orientation="vertical"  
 app:layout\_constraintGuide\_percent="0.6" />  
  
</androidx.constraintlayout.widget.ConstraintLayout>

card\_dialog

<?xml version="1.0" encoding="utf-8"?>  
<RelativeLayout 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:orientation="vertical"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent" >  
  
 <ImageView  
 android:id="@+id/ivAction"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:src="@mipmap/cardback" />  
  
</RelativeLayout>

card\_for\_action\_cards\_playing

<?xml version="1.0" encoding="utf-8"?>  
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 android:orientation="vertical"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="match\_parent"  
 android:padding="5dp"  
 android:id="@+id/constraintLayout">  
  
 <ImageView  
 android:id="@+id/ivAction"  
 android:layout\_width="0dp"  
 android:layout\_height="0dp"  
 android:src="@mipmap/cardback"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintDimensionRatio="h,436:685"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@+id/tvAction" />  
  
 <ImageView  
 android:id="@+id/ivGreenMargin"  
 android:layout\_width="0dp"  
 android:layout\_height="0dp"  
 android:src="@mipmap/green\_margin"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintDimensionRatio="h,436:685"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@+id/tvAction"  
 android:visibility="invisible"/>  
  
 <ImageView  
 android:id="@+id/ivRedMargin"  
 android:layout\_width="0dp"  
 android:layout\_height="0dp"  
 android:src="@mipmap/red\_margin\_for\_trash"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintDimensionRatio="h,436:685"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@+id/tvAction"  
 android:visibility="invisible"/>  
  
 <ImageView  
 android:id="@+id/ivYellowMargin"  
 android:layout\_width="0dp"  
 android:layout\_height="0dp"  
 android:src="@mipmap/yellow\_margin\_for\_discard"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintDimensionRatio="h,436:685"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@+id/tvAction"  
 android:visibility="invisible"/>  
  
 <ImageView  
 android:id="@+id/ivGreenX"  
 android:layout\_width="0dp"  
 android:layout\_height="0dp"  
 android:src="@mipmap/green\_x"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintDimensionRatio="h,436:685"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@+id/tvAction"  
 android:visibility="invisible"/>  
  
 <ImageView  
 android:id="@+id/ivRedX"  
 android:layout\_width="0dp"  
 android:layout\_height="0dp"  
 android:src="@mipmap/red\_x\_for\_trash"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintDimensionRatio="h,436:685"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@+id/tvAction"  
 android:visibility="invisible"/>  
  
 <ImageView  
 android:id="@+id/ivYellowX"  
 android:layout\_width="0dp"  
 android:layout\_height="0dp"  
 android:src="@mipmap/yellow\_x\_for\_discard"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintDimensionRatio="h,436:685"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@+id/tvAction"  
 android:visibility="invisible"/>  
  
 <TextView  
 android:id="@+id/tvAction"  
 android:layout\_width="0dp"  
 android:layout\_height="0dp"  
 android:gravity="center"  
 android:singleLine="true"  
 android:text="Text"  
 android:autoSizeTextType="uniform"  
 android:autoSizeMinTextSize="1sp"  
 android:autoSizeMaxTextSize="100sp"  
 android:autoSizeStepGranularity="1sp"  
 android:textColor="@color/black"  
 app:layout\_constraintTop\_toTopOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintHeight\_percent="0.15"  
 android:visibility="gone"/>  
  
</androidx.constraintlayout.widget.ConstraintLayout>

card\_kingdom

<?xml version="1.0" encoding="utf-8"?>  
<androidx.constraintlayout.widget.ConstraintLayout 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:orientation="vertical"  
android:layout\_width="match\_parent"  
android:layout\_height="match\_parent">  
  
<ImageView  
 android:id="@+id/ivAction"  
 android:layout\_width="0dp"  
 android:layout\_height="match\_parent"  
 android:src="@mipmap/cardback"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintDimensionRatio="w,436:685"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent" />  
  
</androidx.constraintlayout.widget.ConstraintLayout>

card\_sh

<?xml version="1.0" encoding="utf-8"?>  
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 android:orientation="vertical"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content">  
  
 <ImageView  
 android:id="@+id/ivAction"  
 android:layout\_width="0dp"  
 android:layout\_height="match\_parent"  
 android:src="@mipmap/cardback\_sh"  
 app:layout\_constraintDimensionRatio="w,436:459"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent" />  
  
 <TextView  
 android:id="@+id/countAction"  
 android:layout\_width="0dp"  
 android:layout\_height="0dp"  
 android:background="@mipmap/counter"  
 android:gravity="center"  
 android:maxLength="2"  
 android:singleLine="true"  
 android:text="0"  
 android:autoSizeTextType="uniform"  
 android:autoSizeMinTextSize="1sp"  
 android:autoSizeMaxTextSize="40sp"  
 android:autoSizeStepGranularity="1sp"  
 android:includeFontPadding="false"  
 android:textColor="@color/white"  
 app:layout\_constraintDimensionRatio="w, 1:1"  
 app:layout\_constraintEnd\_toStartOf="@+id/guidelineActionsVertical"  
 app:layout\_constraintStart\_toStartOf="@+id/ivAction"  
 app:layout\_constraintTop\_toTopOf="@+id/ivAction" />  
  
 <androidx.constraintlayout.widget.Guideline  
 android:id="@+id/guidelineActionsVertical"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:orientation="vertical"  
 app:layout\_constraintGuide\_percent="0.3" />  
  
  
</androidx.constraintlayout.widget.ConstraintLayout>

card\_trash

<?xml version="1.0" encoding="utf-8"?>  
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 android:orientation="vertical"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content">  
  
 <ImageView  
 android:id="@+id/ivAction"  
 android:layout\_width="match\_parent"  
 android:layout\_height="0dp"  
 android:src="@mipmap/cardback\_sh"  
 app:layout\_constraintDimensionRatio="w,436:459"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent" />  
  
 <TextView  
 android:id="@+id/countAction"  
 android:layout\_width="0dp"  
 android:layout\_height="0dp"  
 android:background="@mipmap/counter"  
 android:gravity="center"  
 android:maxLength="2"  
 android:singleLine="true"  
 android:text="0"  
 android:autoSizeTextType="uniform"  
 android:autoSizeMinTextSize="1sp"  
 android:autoSizeMaxTextSize="40sp"  
 android:autoSizeStepGranularity="1sp"  
 android:includeFontPadding="false"  
 android:textColor="@color/white"  
 app:layout\_constraintDimensionRatio="w, 1:1"  
 app:layout\_constraintEnd\_toStartOf="@+id/guidelineActionsVertical"  
 app:layout\_constraintStart\_toStartOf="@+id/ivAction"  
 app:layout\_constraintTop\_toTopOf="@+id/ivAction" />  
  
 <androidx.constraintlayout.widget.Guideline  
 android:id="@+id/guidelineActionsVertical"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:orientation="vertical"  
 app:layout\_constraintGuide\_percent="0.3" />  
  
  
</androidx.constraintlayout.widget.ConstraintLayout>

log\_layout

<?xml version="1.0" encoding="utf-8"?>  
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 android:orientation="horizontal"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:id="@+id/rl">  
  
 <TextView  
 android:id="@+id/tvPlayer"  
 android:layout\_alignParentStart="true"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_gravity="center"  
 android:gravity="start"  
 android:text="Player"  
 android:layout\_marginEnd="5dp"/>  
  
 <TextView  
 android:id="@+id/tvLine"  
 android:layout\_alignParentEnd="true"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:layout\_gravity="center"  
 android:gravity="start"  
 android:text="Name"  
 android:layout\_toEndOf="@id/tvPlayer"/>  
  
</RelativeLayout>

table\_layout

<?xml version="1.0" encoding="utf-8"?>  
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 android:orientation="horizontal"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:padding="5dp">  
  
 <TextView  
 android:id="@+id/tvName"  
 android:layout\_alignParentStart="true"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="50dp"  
 android:layout\_gravity="center\_vertical"  
 android:layout\_marginStart="20dp"  
 android:layout\_centerVertical="true"  
 android:gravity="center"  
 android:text="Name"  
 android:textColor="@color/colorPrimary"  
 android:textSize="20sp" />  
  
 <Button  
 android:id="@+id/btnJoin"  
 android:layout\_marginEnd="20dp"  
 android:layout\_alignParentEnd="true"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="50dp"  
 android:layout\_gravity="center\_vertical"  
 android:layout\_centerVertical="true"  
 android:gravity="center"  
 android:text="Play"  
 android:textColor="@color/colorPrimary"  
 android:textSize="20sp" />  
  
</RelativeLayout>

rounded\_corners

<?xml version="1.0" encoding="utf-8"?>  
<shape xmlns:android="http://schemas.android.com/apk/res/android">  
 <solid android:color="@android:color/black"/>  
 <corners android:radius="20dp"/>  
 <padding android:left="0dp" android:top="0dp" android:right="0dp" android:bottom="0dp" />  
</shape>

**Manifest:**

<?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.dominion\_game"  
 android:minSdkVersion="7">  
  
 <uses-permission android:name="android.permission.READ\_PHONE\_STATE" />  
 <uses-permission android:name="android.permission.SEND\_SMS" />  
 <uses-permission android:name="android.permission.READ\_CALL\_LOG" />  
 <uses-permission android:name="android.permission.ANSWER\_PHONE\_CALLS" />  
 <uses-permission android:name="android.permission.INTERNET" />  
  
 <application  
 android:allowBackup="true"  
 android:icon="@mipmap/img\_dominion\_background"  
 android:label="@string/app\_name"  
 android:roundIcon="@mipmap/img\_dominion\_background"  
 android:supportsRtl="false"  
 android:theme="@style/Theme.AppCompat.NoActionBar"  
 android:usesCleartextTraffic="true"  
 tools:ignore="GoogleAppIndexingWarning">  
 <activity  
 android:name=".activities.LoginActivity"  
 android:configChanges="orientation|screenSize"  
 android:screenOrientation="landscape">  
 <intent-filter>  
 <action android:name="android.intent.action.MAIN" />  
  
 <category android:name="android.intent.category.LAUNCHER" />  
 </intent-filter>  
 </activity>  
  
 <activity  
 android:name=".activities.RegisterActivity"  
 android:configChanges="orientation|screenSize"  
 android:screenOrientation="landscape" />  
 <activity  
 android:name=".activities.StartActivity"  
 android:configChanges="orientation|screenSize"  
 android:screenOrientation="landscape" />  
 <activity  
 android:name=".activities.OnlineGameActivity"  
 android:configChanges="orientation|screenSize"  
 android:screenOrientation="landscape" />  
 <activity  
 android:name=".activities.PrepareGameActivity"  
 android:configChanges="orientation|screenSize"  
 android:screenOrientation="landscape" />  
 <activity  
 android:name=".activities.OnlineTablesActivity"  
 android:configChanges="orientation|screenSize"  
 android:screenOrientation="landscape" />  
 <activity  
 android:name=".activities.GameActivity"  
 android:configChanges="orientation|screenSize"  
 android:screenOrientation="landscape" />  
 <receiver  
 android:name=".classes.PhoneCallReceiver"  
 android:enabled="true"  
 android:exported="false" />  
  
 <service  
 android:name=".classes.MusicService"  
 android:exported="false" />  
 </application>  
  
  
</manifest>

**Colors:**

<?xml version="1.0" encoding="utf-8"?>  
<resources>  
 <color name="colorPrimary">#000000</color>  
 <color name="colorPrimaryDark">#000000</color>  
 <color name="colorAccent">#000000</color>  
 <color name="colorVP">#00E2FF</color>  
 <color name="black">#000000</color>  
 <color name="red">#FF0000</color>  
 <color name="pink">#FF4E4E</color>  
 <color name="green">#4CAF50</color>  
 <color name="white">#FFFFFF</color>  
 <color name="player\_red">#FF0000</color>  
</resources>

**Strings:**

<resources>  
 <string name="app\_name">Dominion\_game</string>  
 <string name="undo">Undo</string>  
 <string name="confirm\_trash">Confirm Trashing</string>  
 <string name="confirm\_discard">Confirm Discarding</string>  
 <string name="confirm\_top\_deck">Confirm Top Deck</string>  
 <string name="order">Done Ordering</string>  
  
</resources>

**Styles:**

<resources>  
  
 <!-- Base application theme. -->  
 <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">  
 <!-- Customize your theme here. -->  
 <item name="colorPrimary">@color/colorPrimary</item>  
 <item name="colorPrimaryDark">@color/colorPrimaryDark</item>  
 <item name="colorAccent">@color/colorAccent</item>  
 </style>  
  
</resources>

**Python - Server:**

from flask import Flask, request  
import firebase\_admin  
from firebase\_admin import credentials  
from firebase\_admin import db  
import json  
  
app = Flask(\_\_name\_\_)  
  
  
'''  
A function that creates a game with the function push that creates  
a random and unique string as a key to the game and returns this key to the creator.  
'''  
@app.route('/creator\_start', methods=['GET'])  
def creator\_start():  
 game\_id = get\_games\_ref().push().key  
  
 return {\*\*{"game\_id": game\_id}, \*\*{"success": True}}  
  
  
'''  
A function that adds the second player to the table if there is only one player there.  
'''  
@app.route('/non\_creator\_start', methods=['POST'])  
def non\_creator\_start():  
 data = request.json  
 game\_id = get\_games\_ref().child(data.get("gameId"))  
 if game\_id.child("before-game-data").child("idP2").get() != "":  
 return {"success": False}  
 game\_id.child("before-game-data").update({"idP2": data.get("idP2")})  
  
 return {"success": True}  
  
  
'''  
A function that uploads all data while game didn't start.  
'''  
@app.route('/upload\_all\_data', methods=['POST'])  
def upload\_all\_data():  
 data = request.json  
 game\_id = get\_games\_ref().child(data.get("gameManagerBeforeStart").get("gameId"))  
 game\_id.child("before-game-data").update({"idP1": data.get("gameManagerBeforeStart").get("idP1"),  
 "idP2": data.get("gameManagerBeforeStart").get("idP2"),  
 "isRated": data.get("gameManagerBeforeStart").get("isRated"),  
 "isReady1": data.get("gameManagerBeforeStart").get("isReady1"),  
 "isReady2": data.get("gameManagerBeforeStart").get("isReady2"),  
 "isStart1": data.get("gameManagerBeforeStart").get("isStart1"),  
 "isStart2": data.get("gameManagerBeforeStart").get("isStart2")  
 })  
 game\_id.child("general-data").set({"actionCards": data.get("actionCards")})  
 game\_id.child("game-data").set({"board": json.loads(data.get("board")), "trash": data.get("trash"),  
 "log": data.get("log"), "turn": data.get("turn"), "isGameEnded": data.get("isGameEnded")})  
  
 return {"success": True}  
  
  
'''  
A function that uploads game data while game started.  
'''  
@app.route('/upload\_real\_time', methods=['POST'])  
def upload\_real\_time():  
 data = request.json  
 game\_id = get\_games\_ref().child(data.get("gameManagerBeforeStart").get("gameId"))  
 game\_id.child("game-data").update({"isGameEnded": data.get("isGameEnded")})  
 if "board" in data and "trash" in data and "log" in data and "turn" in data:  
 game\_id.child("game-data").update({"board": json.loads(data.get("board")), "trash": data.get("trash"),  
 "log": data.get("log")})  
 game\_id.child("game-data").child("turn").update(data.get("turn"))  
 return {\*\*data.get("turn"), \*\*{"success": True}}  
  
 data\_response\_game = game\_id.child("game-data").get()  
 data\_response\_game["board"] = json.dumps(data\_response\_game["board"])  
 print("upload\_real\_time:" + " getting data rn")  
 return {\*\*data\_response\_game, \*\*{"success": True}}  
  
  
'''  
A function that returns all data while game didn't start.  
'''  
@app.route('/get\_all\_data', methods=['POST'])  
def get\_all\_data():  
 data = request.json  
 game\_id = get\_games\_ref().child(data.get("gameId"))  
 data\_response\_before\_game = game\_id.child("before-game-data").get()  
 data\_response\_game\_general = game\_id.child("general-data").get()  
 data\_response\_game = game\_id.child("game-data").get()  
  
 if data\_response\_before\_game is None or data\_response\_game\_general is None or data\_response\_game is None:  
 return {"success": False}  
  
 data\_response\_game["board"] = json.dumps(data\_response\_game["board"])  
 return {\*\*data\_response\_before\_game, \*\*data\_response\_game\_general, \*\*data\_response\_game, \*\*{"success": True}}  
  
  
'''  
A function that returns game data while game started.  
'''  
@app.route('/get\_real\_time', methods=['POST'])  
def get\_real\_time():  
 data = request.json  
 if "gameManagerBeforeStart" not in data:  
 game\_id = get\_games\_ref().child(data.get("gameId"))  
 data\_response\_game = game\_id.child("game-data").get()  
 if data\_response\_game is None:  
 return {"success": False}  
  
 data\_response\_game["board"] = json.dumps(data\_response\_game["board"])  
 return {\*\*data\_response\_game, \*\*{"success": True}}  
  
 game\_id = get\_games\_ref().child(data.get("gameManagerBeforeStart").get("gameId"))  
 game\_id.child("game-data").update({"board": json.loads(data.get("board")), "trash": data.get("trash"), "log": data.get("log")})  
  
 game\_id.child("game-data").child("turn").update(data.get("turn"))  
 print("get\_real\_time:" + " uploading data rn")  
 return {"success": True}  
  
  
'''  
A function that uploads the data about the player that sent the request  
and returns the data about the other player.  
'''  
@app.route('/get\_and\_upload\_player\_data', methods=['POST'])  
def get\_and\_upload\_player\_data():  
 data = request.json  
 game\_id = get\_games\_ref().child(data.get("gameManagerBeforeStart").get("gameId"))  
 my\_id = data.get("gameManagerBeforeStart").get("idP1")  
 enemy\_id = data.get("gameManagerBeforeStart").get("idP2")  
 if not data.get("gameManagerBeforeStart").get("isCreator"):  
 my\_id, enemy\_id = enemy\_id, my\_id  
 game\_id.child(my\_id).set(data.get("myData"))  
  
 enemy\_data = game\_id.child(enemy\_id).get()  
 if enemy\_data is None:  
 return {"success": False}  
  
 return {\*\*enemy\_data, \*\*{"success": True}}  
  
  
'''  
A function that returns before-game-data from the database.  
'''  
@app.route('/get\_game\_manager\_before\_start', methods=['POST'])  
def get\_game\_manager\_before\_start():  
 data = request.json  
 game\_id = get\_games\_ref().child(data.get("gameId"))  
 before\_game\_data = game\_id.child("before-game-data").get()  
 if before\_game\_data is None:  
 return {"success": False}  
  
 return {\*\*before\_game\_data, \*\*{"success": True}}  
  
  
'''  
A function that uploads before-game-data to the database.  
'''  
@app.route('/upload\_game\_manager\_before\_start', methods=['POST'])  
def upload\_game\_manager\_before\_start():  
 data = request.json  
 game\_id = get\_games\_ref().child(data.get("gameId"))  
 game\_id.child("before-game-data").set({"idP1": data.get("idP1"),  
 "idP2": data.get("idP2"),  
 "isReady1": data.get("isReady1"),  
 "isReady2": data.get("isReady2"),  
 "isStart1": data.get("isStart1"),  
 "isStart2": data.get("isStart2"),  
 "isRated": data.get("isRated")})  
  
 return {"success": True}  
  
  
'''  
A function that updates ready to the player sent the request.  
'''  
@app.route('/update\_ready', methods=['POST'])  
def update\_ready():  
 data = request.json  
 game\_id = get\_games\_ref().child(data.get("gameId"))  
 game\_id.child("before-game-data").update({"isReady1": data.get("isReady1")}  
 if data.get("isCreator")  
 else {"isReady2": data.get("isReady2")})  
  
 return {"success": True}  
  
  
'''  
A function that updates ready to start to the player sent the request.  
'''  
@app.route('/update\_ready\_to\_start', methods=['POST'])  
def update\_ready\_to\_start():  
 data = request.json  
 game\_id = get\_games\_ref().child(data.get("gameId"))  
 game\_id.child("before-game-data").update({"isStart1": data.get("isStart1")}  
 if data.get("isCreator")  
 else {"isStart2": data.get("isStart2")})  
  
 return {"success": True}  
  
  
'''  
A function that deletes the table.  
'''  
@app.route('/delete\_game\_manager\_before\_start', methods=['POST'])  
def delete\_game\_manager\_before\_start():  
 data = request.json  
 game\_id = get\_games\_ref().child(data.get("gameId"))  
 game\_id.delete()  
  
 return {"success": True}  
  
  
'''  
A function that deletes the second player.  
'''  
@app.route('/delete\_p2', methods=['POST'])  
def delete\_p2():  
 data = request.json  
 game\_id = get\_games\_ref().child(data.get("gameId"))  
 game\_id.child("before-game-data").update({"idP2": "", "isReady2": False})  
  
 return {"success": True}  
  
  
'''  
A function that returns all the tables in database.  
'''  
@app.route('/get\_tables', methods=['GET'])  
def get\_tables():  
 data\_request = get\_games\_ref().get()  
 if data\_request is None:  
 return {"success": False}  
 data\_edited = {}  
 for game in data\_request:  
 if type(data\_request.get(game)) != str:  
 data\_edited[game] = data\_request.get(game).get("before-game-data")  
  
 return {\*\*data\_edited, \*\*{"success": True}}  
  
  
'''  
A function that deletes all the last game data to be ready for the next game.  
'''  
@app.route('/end\_game', methods=['POST'])  
def end\_game():  
 data = request.json  
 game\_id = get\_games\_ref().child(data.get("gameId"))  
 game\_id.child("game-data").delete()  
 game\_id.child("general-data").delete()  
 game\_id.child(data.get("idP1")).delete()  
 game\_id.child(data.get("idP2")).delete()  
 game\_id.child("before-game-data").update({"idP1": data.get("idP1"),  
 "idP2": data.get("idP2"),  
 "isReady1": False,  
 "isReady2": False,  
 "isStart1": False,  
 "isStart2": False,  
 "isRated": data.get("isRated")})  
  
 return {"success": True}  
  
  
'''  
A function that gets the details that the player entered to register,  
gets all users from database and checks if there is the username.  
if the username exists, the function returns success false,  
otherwise it creates the user and returns success true.  
'''  
@app.route('/register', methods=['POST'])  
def register():  
 data\_register = request.json  
 data\_request = get\_users\_ref().get(shallow=True)  
 if data\_request is not None and data\_register.get("username") in data\_request.keys():  
 return {"success": False}  
  
 username = get\_users\_ref().child(data\_register.get("username"))  
 username.set({"email": data\_register.get("email"), "password": data\_register.get("password")})  
  
 return {"username": data\_register.get("username"), "success": True}  
  
  
'''  
A function that gets the details that the player entered to login,  
gets all users from database and checks if there is the username.  
if the username doesn't exist or the password doesn't match the username, the function returns success false,  
otherwise it returns success true and returns the username.  
'''  
@app.route('/login', methods=['POST'])  
def login():  
 data\_register = request.json  
 data\_request = get\_users\_ref().get(shallow=True)  
 if data\_request is None or data\_register.get("username") not in data\_request.keys()\  
 or get\_users\_ref().child(data\_register.get("username")).get().get("password") != data\_register.get("password"):  
 return {"success": False}  
  
 return {"username": data\_register.get("username"), "success": True}  
  
  
def get\_games\_ref():  
 return db.reference("Games")  
  
  
def get\_users\_ref():  
 return db.reference("Users")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # initializes a new App instance and runs the flask  
 firebase\_admin.initialize\_app(credentials.Certificate(  
 r"C:\Users\or\Desktop\DominionGame\dominion-project-firebase-adminsdk-wzw45-052b5aeeea.json"),  
 {'databaseURL': 'https://dominion-project.firebaseio.com'})  
 app.run(host='0.0.0.0', port=8888)