



Dominion Project

מגיש: אור בר צור

ת.ז: 213378706

כתובת: יב⁴

בית ספר: גימנסיה ריאלית ראשון לציון

תוכן עניינים

2.....	הסבר כללי על הפרויקט
3.....	יכולות וMSCIMS
3.....	יכולות המערכת ומוגבלות
3.....	תרשימים MSCIMS
4.....	פירוט MSCIMS
11.....	מדריך למשתמש
18.....	תרשימים UML מקוצר
19.....	פירוט מחלקות עם הסברים
31.....	אלגוריתמים מעניינים
34.....	שימוש ב-Resources
34.....	Broadcast Receiver
34.....	Service
35.....	Thread
35.....	FireBase
37.....	טבלת פונקציות בפייתון
41.....	רפלקציה
42.....	קוד מלא

הסבר כללי על הפרויקט

נושא הפרויקט: אפליקציה של המשחק "דומיניון" (הסבר כללי על המשחק [כאן](#) ובמדריך למשתמש שסביר את חוקי המשחק באופן מפורט).

הסיבה שבחرتني לשחק: לפני השנה קנינו את משחק הקופסה של "דומיניון". ממש התמכרנו אליו ושיחקתי בו עם כל החברים, כך שזה הפך להיות גם בילוי כאשר כל אחד בabitו ומשחקים באתר שלהם אונליין. כיום, הטלפונים הסולריים החלו להחליף את השימוש הרוב במחשבים וחשבתי שיש צורך באפליקציה לשחק (לא קיימת אפליקציה כרגע). עקב לכך זה שלי ושל חבריי באפליקציה של המשחק, בחרתי לעשות פרויקט זה.

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

הסבר כללי: המשחק מיועד ל 2 עד 4 שחקנים. בטור התחלה, יוצרת משחק צזה עבור שני שחקנים בלבד. כל אחד מהשחקנים יראה מסך שונה בזמן המשחק, הכולל את הקלפים שבידיו, את הקלפים שבמשחק, אותם יכול לנחות, וכל מיני מדדים ואופציות נוספות, למשל autoplay treasure.

בכל מהלך של משחק, לשני השחקנים יוצגו הקלפים שהפעיל וכמות הנזודות, הנקודות והפעולות שיש לו באותו המורו.

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

השרת הוא שרת פיתון אשר מקבל הודעות מהלקוח (אנדרואיד) דרך flask שזו דרך לניטוב בקשות `http` לפעולות בפייתון לפי הסיומת של הבקשה (`dogsma, get_tables`). השרת מקשר לבסיס נתונים של פיירבייס ויכול להעלות נתונים לשרת, לשולף מידע, לבדוק ולמחוק, ולבסוף להחזיר תשובה ללקוח שבקש את הבקשה. למשל, הוא יכול לשולף את הנתונים על המשחק ממי שיצר את המשחק (`host`) עבור מי שנכנס אליו, על מנת שני השחקנים ישחקו באותו המשחק. השרת מעביר גם הודעות `"real time"` שימושיות לשחקן במהלך המשחק על מנת שהמספר של כל שחקן יהיה מעודכן לפי התור האחרון והפעולה الأخيرة שנעשתה במהלך המשחק.

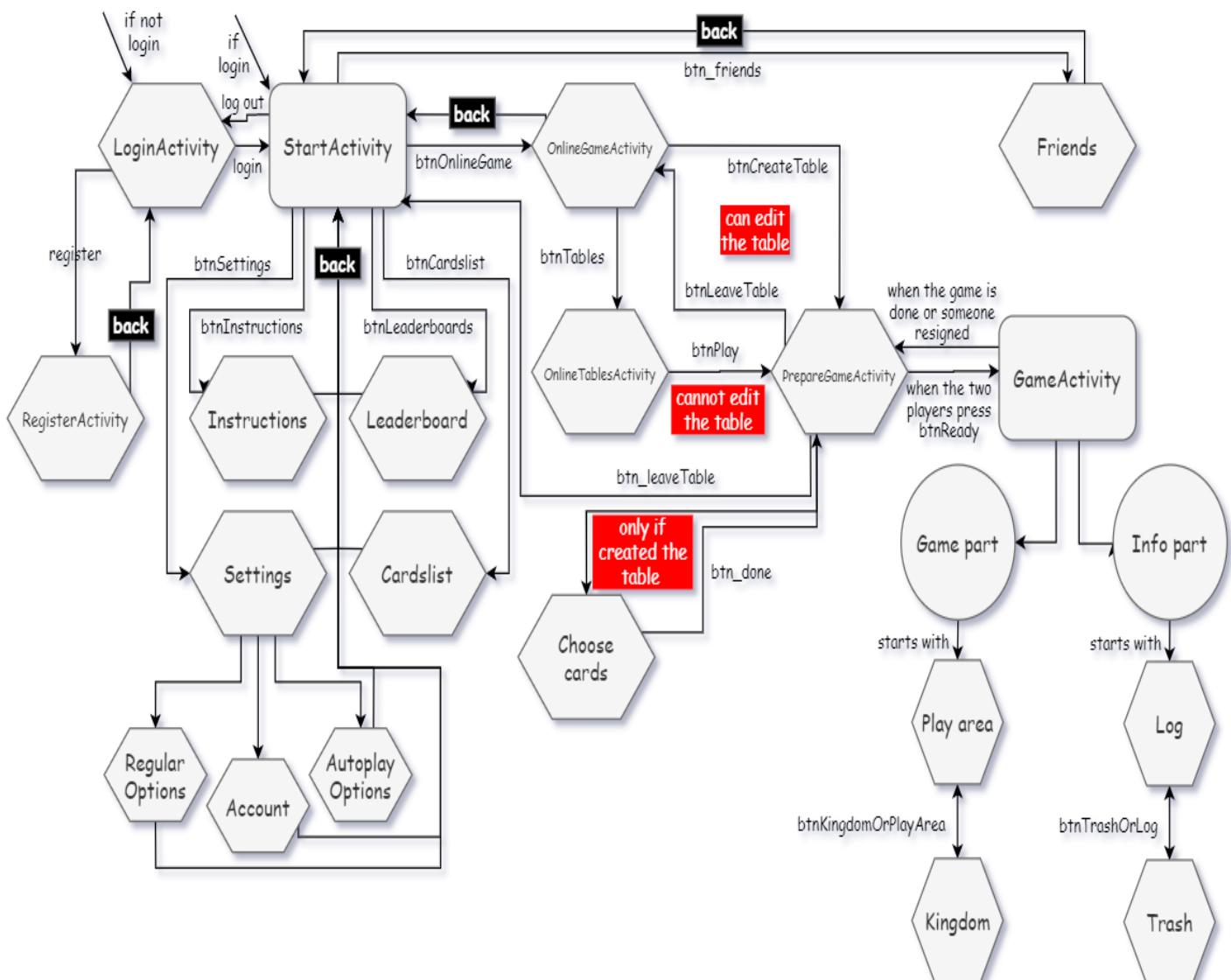
מגבילות הפרויקט: הפרויקט הבסיסי לא יכול לכל הנראה את הדברים הבאים: הרחבות של המשחק, אופציה לשולשה ולארבעה שחקנים אונליין, אופציה לשחק נגדبوت כדי להתאמן, התכתבות בין שני השחקנים תוך כדי המשחק. בנוסף, מכיוון שאינו עושים שימוש בשרת בבסיס נתונים של פיירבייס חינמי, יתכן שההודעות אשר יעברו יהיו טיפה איטיות, אך ניתן לסדר זאת עם תקציב יחסית קטן לפרויקט. הפרויקט יrown על פיתון 3.0

יכולות וMSCים

יכולות המערכת ומוגבלות

- ☒ מושך דומיניו של שני שחקנים בראשת.
- ☒ אפשרות להתחבר למשחק קיים או ליצור משחק.
- ☒ אפשרות לשחק עם חברים.
- ☒ עדכון מסך המשחק במהלך התוור של השחקן ובמהלך התוור של היריב.
- ☒ מסך leaderboard המעדכן את המתחרים בעלי הכירוב ניצחונות ביחס להפסדים.
- ☒ קלפי פעולה של גרסת הבסיס עם אופציה להרחבה.
- ☒ שליחת הודעות בין השחקנים במהלך המשחק (בגדר הרחבה).

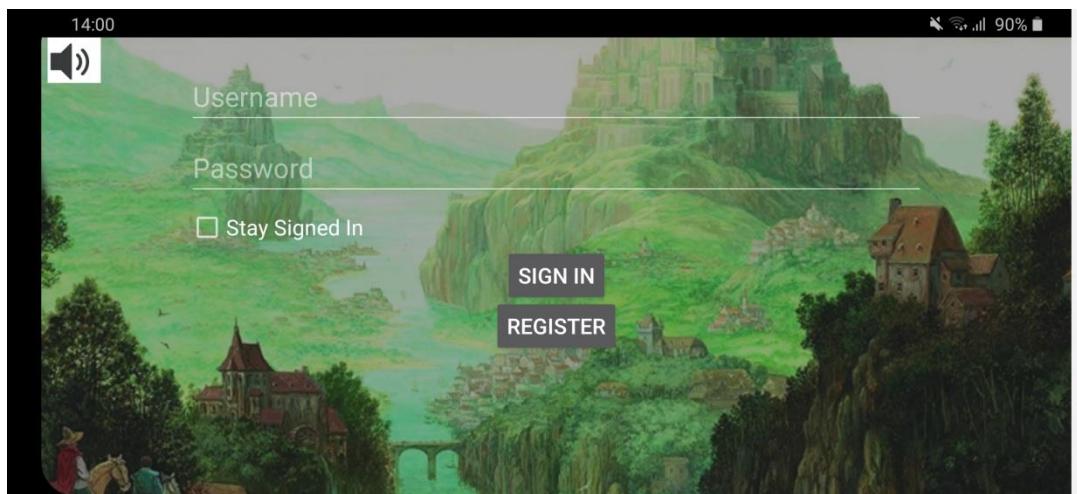
תרשים MSCים



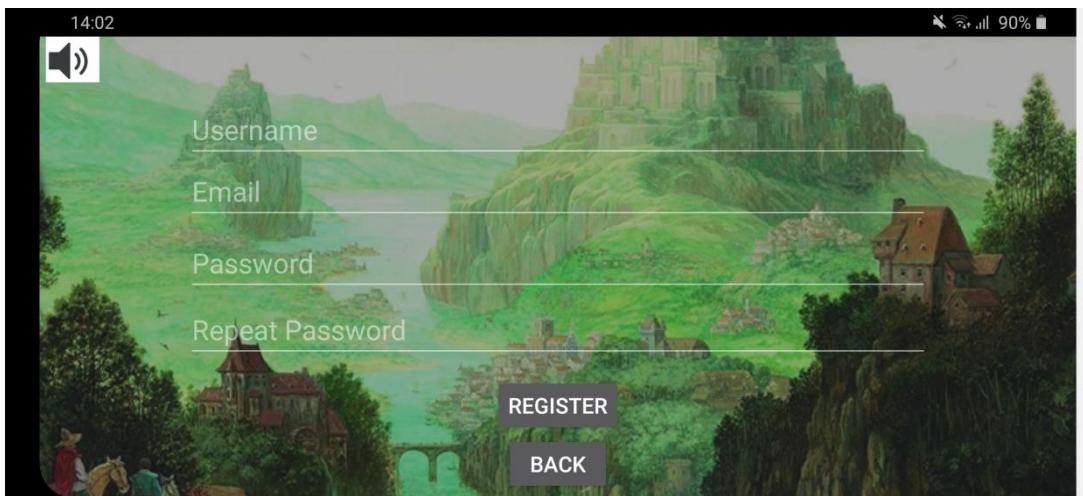
פירוט מסכימים

*בכל המ██ים מלבד המ██ GameActivity יש אפשרות להשתיק את המ██יקה ולהפעיל אותה. זה קורה על יד' SharedPreferences אשר שומר בכל ליחיצה את המצב הנוכחי ומפעיל או מכבה את ה service בהתאם ללחיצה.

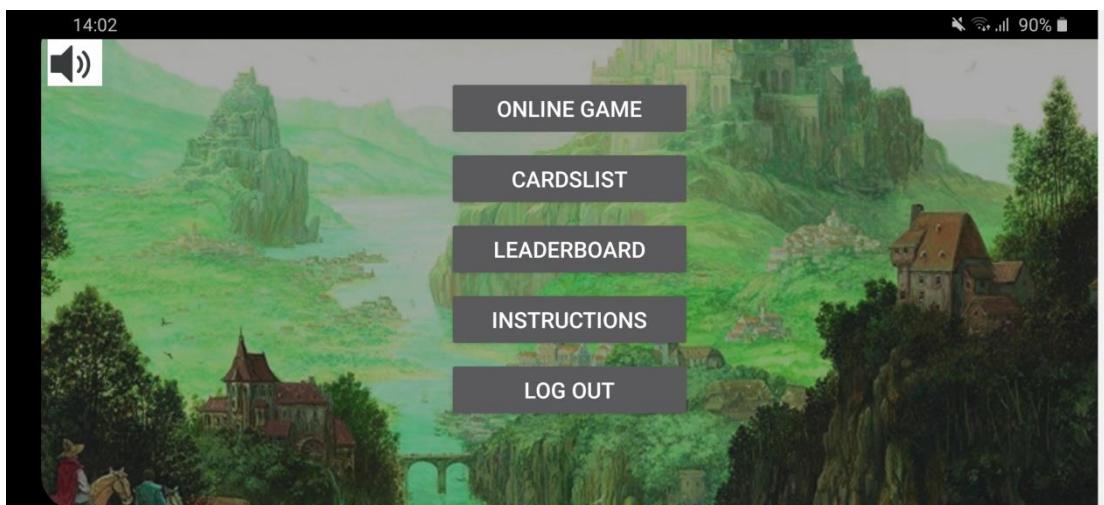
- מסך התחברות לאפליקציה שנייתן להתחבר על ידי הזנת שם משתמש וסיסמה או לעבור למסך הירשנות לאפליקציה. אם מסמנים את ה checkbox שכתוב עליו In signed, אז בעת התחברות לאפליקציה בפעם הבאה יעברו אוטומטית למסך הפתיחה. לאחר הקשה על כפתור In sign, מוצג progressDialog אשר מופסק כאשר מתקבלת תשובה מהשרת האם קיימם שם המשתמש צזה 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:



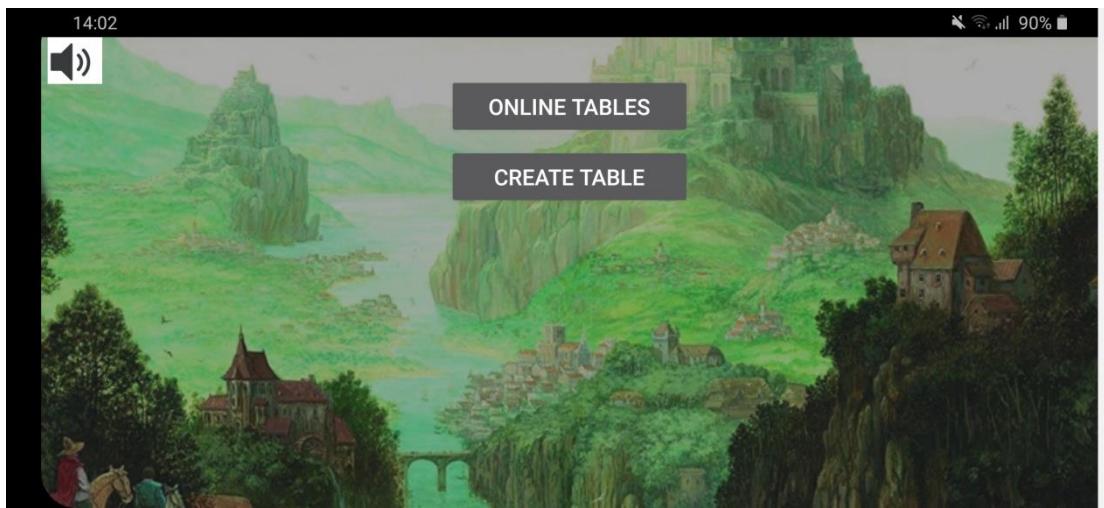
- מסך הרשמה לאפליקציה אשר מאפשר הירשנות על ידי הזנת שם משתמש חדש, אמייל וסיסמה. לאחר הקשה על כפתור Register, מתבצעות בדיקות פנימיות האם המידע שהוזן חוקי ואם כן, אז מוצג progressDialog אשר מופסק כאשר מתקבלת תשובה מהשרת אם ניתן ליצור משתמש חדש ואם הוא יצר, אחרת יוצג למשתמש Toast שלא ניתן ליצור את המשתמש 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 אליו. כל זה מתבצע באמצעות SharedPreferences אשר מאפשר לשמור בטלפון את פרטי החשבון של המשתמש. המסך נראה כך:



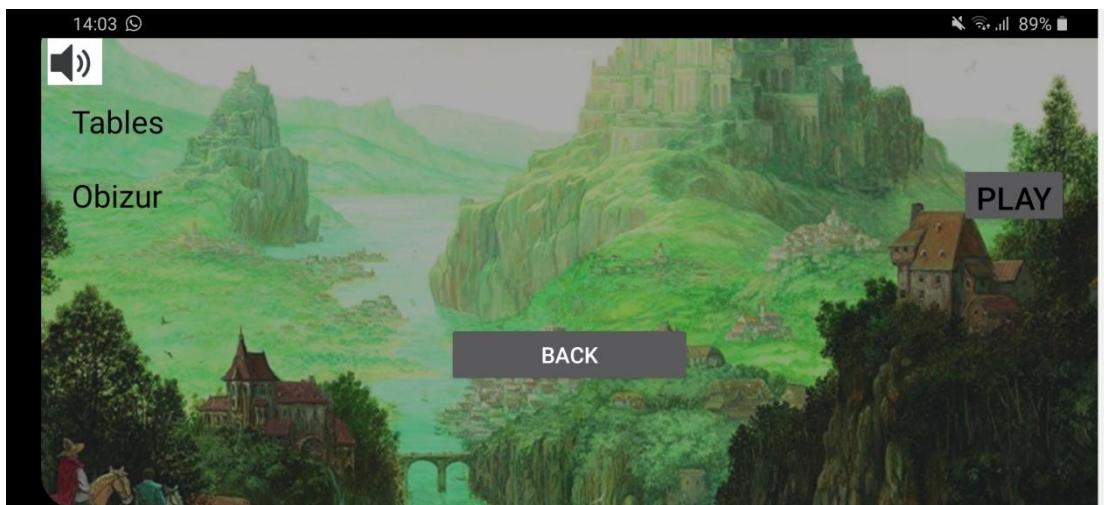
- מסך הבית של האפליקציה. כאשר נכנסים כশתוחבים, אז מגעים אוטומטית למסך זהה וממנו ניתן לעבור למסךים שונים ולעשות out Log. בלחיצה על הכפתור Online Game עוברים למסך OnlineGameActivity. המסך נראה כך:



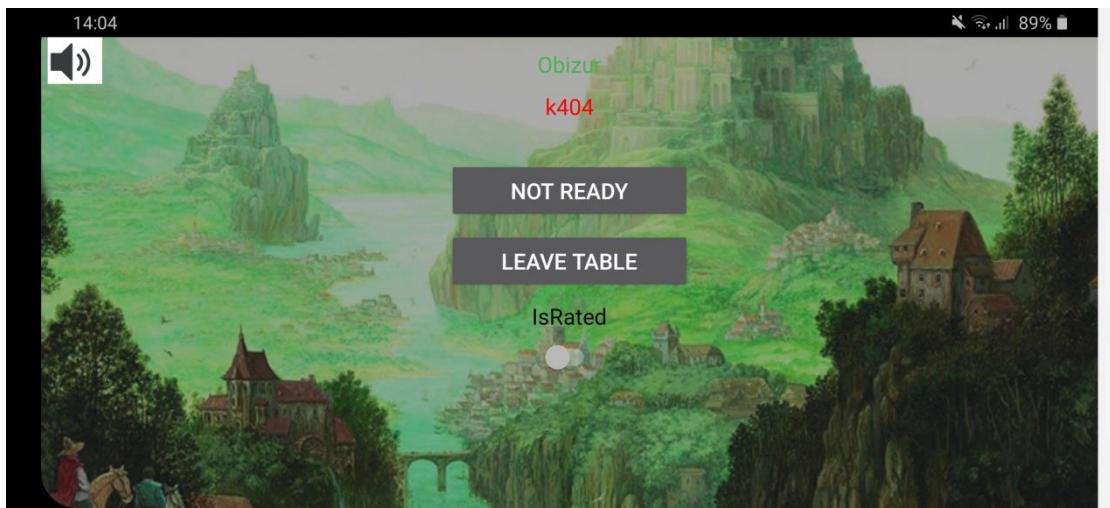
- מסך שבו ניתן לבחור אם רוצים ליצור משחק או להצטרף למשחק קיימ. בלחיצה על כפתור Online Tables עוברים למסך OnlineTablesActivity ובלחיצה על כפתור Create Table עוברים למסך ההכנה של המשחק – prepareGameActivity. המסך נראה כך:



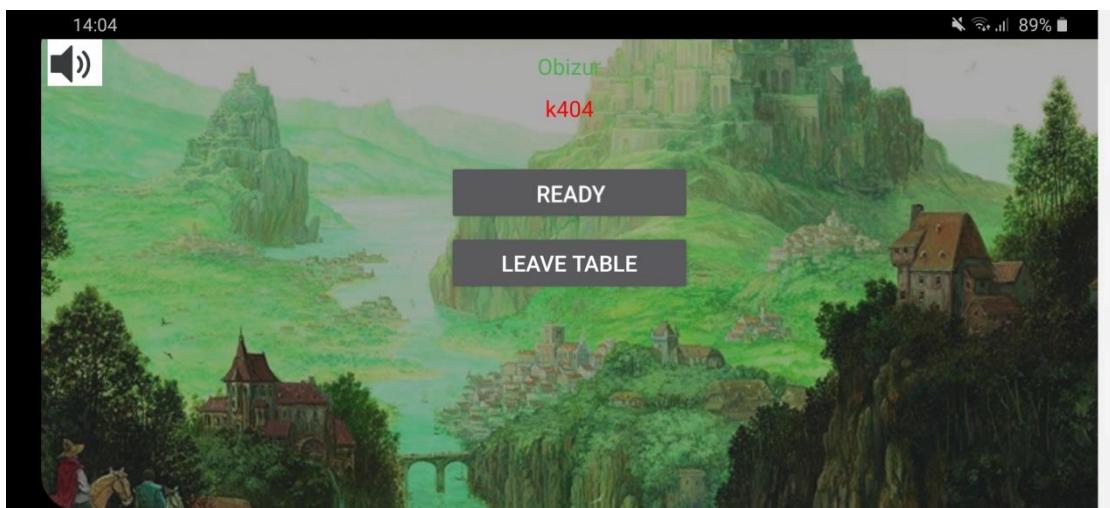
- מסך שבו רואים את כל המשחקים אונליין אשר מתקבלים מהשרת. כאשר מגיעים למסך זהה מופיע progressDialog שמוסך כאשר הגיעו תשובה מהשרת על המשחקים הקיימים. הטבלה מתעדכנת באמצעות TablesAdapter בולולאה שנפסקת רק עם היציאה מהמסך הזה. המסך נראה כך:



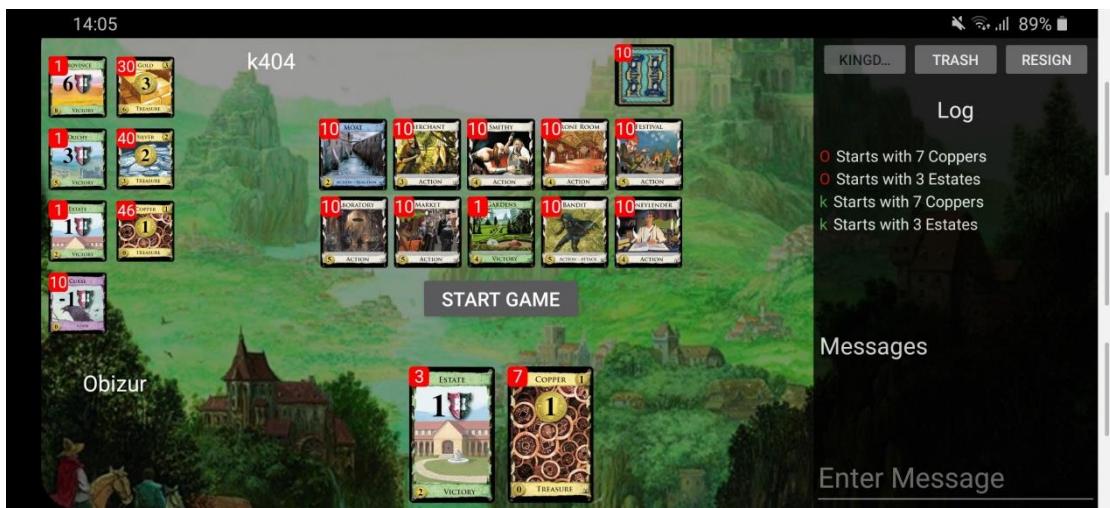
- מסך ההכנה של המשחק. כאשר יוצרים את המשחק מופיע progressDialog אשר נגמר כאשר המשחק עלה לשרת. תחילת מופיע רק TextView עם השחקן שייצור את המשחק, אך כאשר מצטרף שחקן למשחק, הוא מופיע גם כן. ניתן לחוץ על כפתור Ready אשר הופך את הטקסט לצבע ירוק ומעלה לשרת שהשחקן מוכן. אם השחקן זה שייצור את המשחק, יש לו אופציות נוספות שבנה 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 עברו מי שייצור את המשחק:



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



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



כאשר שני השחקנים מסמנים שהם מוכנים כשלוחצים על הכפתור Start Game המשחק מתחל וmagiel מי מתחל. המסר נראה כך כאשר זהו התור של השחקן:



המסר נראה כך כאשר זהו אינם התור של השחקן (כשהוא מחכה שהיריב ישבוק):



עבור כל קלף במשחק, ניתן ללחוץ לחיצה ארוכה אשר מפעילה המסר של cardDialog המופיע בחלון dialog יכול להיות מבוטל בכל לחיצה על המסר. הוא נראה כך:



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



בלחיצה על הכפתור Trash האזור ימני באנט עליון מוחלף באזור אחר שבו מוצגים קלפיי Trash שבמשחק. בלחיצה על הכפתור, מוחלף הטקסט שעליו ל Log שמאפשר לחזור לאזור הקודם וכך להחליף בין האזורים בלחיצה על אותו כפתור. האזור נראה כר (ימין) (באמצע):



בהפעלת קלפים מסוימים עולה על המסך RecyclerView דומה למה שיש ביד. בمسך זה יש לבחור קלפים מסוימים, לסדר אותם וcdcומה, בהתאם לקלף שהופעל. ניתן להציג 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 Dominion 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 your deck. 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.

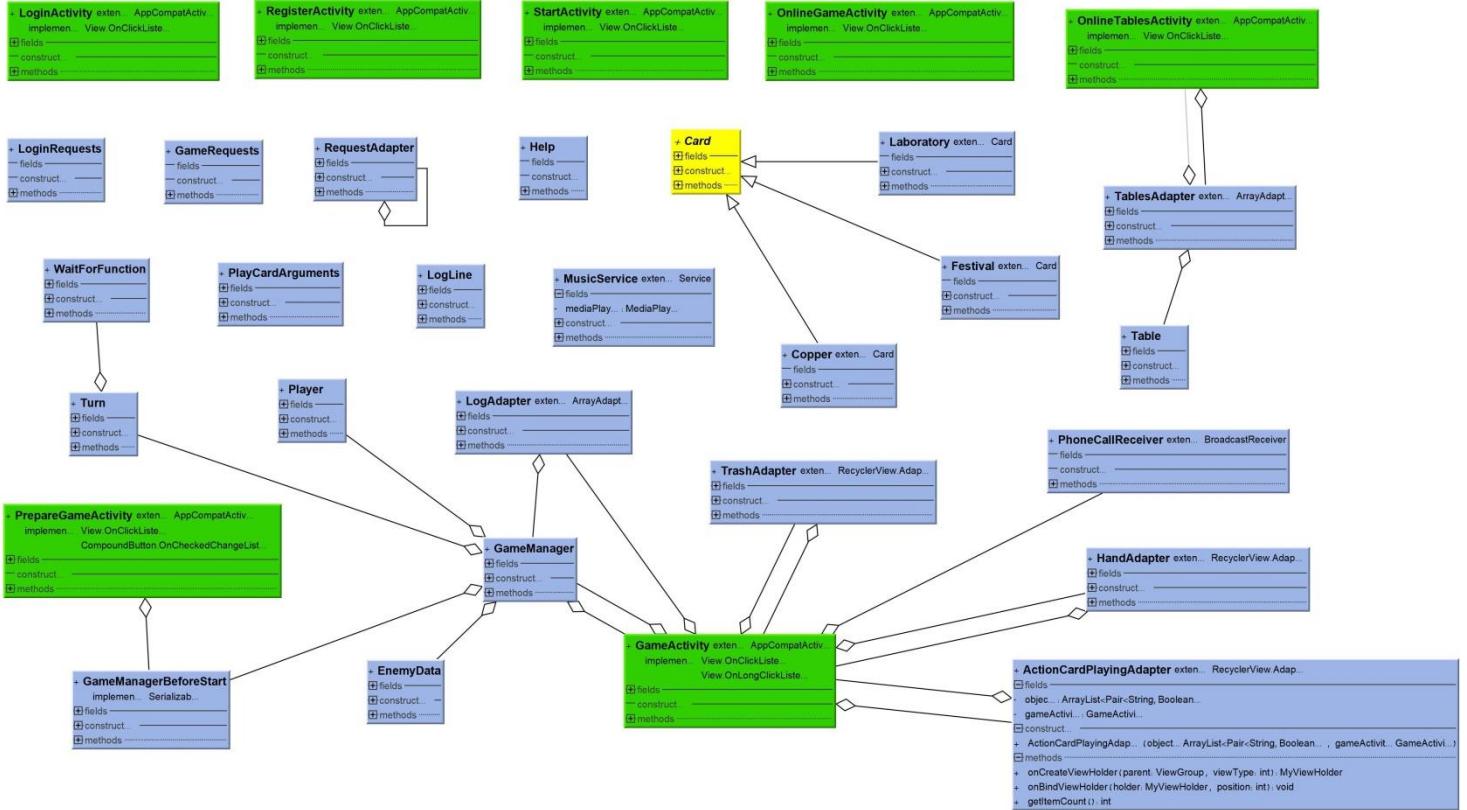
Undo: Request to undo the most recent thing that happened. Your opponent(s) must allow the undo in order for it to happen. While refusing may seem unsportsmanlike, it is every player’s right to refuse an undo request.

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 מוקוצר



פירוט מחלקות עם הסברים

LoginActivity:

```
+ LoginActivity exten... AppCompatActivity
  implemen... View.OnClickListener
  fields
  ~ etUserName EditText
  ~ etPassword EditText
  ~ btnLogin Button
  ~ btnRegister Button
  ~ staySignedIn CheckBox
  ~ sharedPrefs SharedPreferences
  ~ progressDialog ProgressDialog
  ~ sound ImageView
  ~ construct...
  methods
  # onCreate(savedInstanceState: Bundle): void
  + onBackPress(): void
  + onClick(view: View): void
  # onActivityResult(requestCode: int, resultCode: int, intent: Intent): void
  + loginDetailsToJson(): JSONObject
  + loginSuccess(username: String): void
  + endProgressBar(): void
```

מוך 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 הירשות אם איןו נמצא במערכת. ה layout של מסך זה הוא .activity_login

RegisterActivity:

```
+ RegisterActivity exten... AppCompatActivity
  implemen... View.OnClickListener
  fields
  ~ etUserName EditText
  ~ etEmail EditText
  ~ etPassword EditText
  ~ etRepeatPassw... EditText
  ~ btnRegister Button
  ~ btnBack Button
  ~ progressDialog ProgressDialog
  ~ sound ImageView
  ~ sharedPrefs SharedPreferences
  ~ construct...
  methods
  # onCreate(savedInstanceState: Bundle): void
  + onBackPress(): void
  + onClick(view: View): void
  + registerDetailsToJson(): JSONObject
  + isEmailValid(): boolean
  + registerSuccess(username: String): void
  + endProgressBar(): void
```

מסך הירשות הכלול יצירט שם משתמש וסיסמה. בנוסף ייתן המשתמש את המail (על מנת לפתוח את האפשרות להוסיף "שחתי סיסמה"). שם המשתמש והסיסמה יתווסף במידה שם המשתמש לא נמצא כבר במערכת. לאחר מכן יעבור המשתמש אל מסך .Login Activity. כולל כפתור חזרה אל .activity_register של מסך זה הוא layout

LoginRequests:

```
+ LoginRequests
  fields
  ~ construct...
  methods
  + getServerIP(): String
  + getPort(): String
  + register(registerActivit... RegisterActivit..., registerRequest: JSONObject): void
  + login(loginActivit... LoginActivity, loginRequest: JSONObject): void
```

מחלקה סטטית שדואגת להעביר לשרת בקשות login ו register, ומתקבלת תשובה בהתאם האם ההרשמה או ההתחברות הzbוצעה בהצלחה או לא. הפעולות במחלקה נקבעות מהמסכים Login Activity או Register Activity

StartActivity:

```
+ StartActivity exten... AppCompatActivity
  implemen... View.OnClickListener
  fields
  ~ btnOnlineGame Button
  ~ btnSetting... Button
  ~ sharedPrefs SharedPreferences
  ~ sound ImageView
  ~ construct...
  methods
  # onCreate(savedInstanceState: Bundle): void
  + onBackPress(): void
  + onClick(view: View): void
```

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

OnlineGameActivity:

```
+ OnlineGameActivity extends AppCompatActivity  
    implements View.OnClickListener  
  
fields  
- btnTables Button  
- btnCreateTable Button  
- sharedPreferences SharedPreferences  
- sound ImageView  
  
construct...  
  
methods  
# onCreate(savedInstanceState: Bundle): void  
+ onBackPress(): void  
+ onClick(view: View): void  
# onActivityResult(requestCode: int, resultCode: int, intent: Intent): void
```

מסר עבור מי שרוצה להתחיל משחק. יש לבחור אם רוצים ליצור משחק או להצטרף למשחק קיימ. כולל כפטור חזרה אל Start Activity. ה layout של מסך זה הוא `activity_online_game`

OnlineTablesActivity:

```
+ OnlineTablesActivity extends AppCompatActivity  
    implements View.OnClickListener  
  
fields  
- btnBack Button  
- lv ListView  
- tableList ArrayList<Table>  
- tablesAdapter TablesAdapter  
- progressDialog ProgressDialog  
- background ImageView  
- isFinished boolean  
- sound ImageView  
- sharedPreferences SharedPreferences  
  
construct...  
  
methods  
# onCreate(savedInstanceState: Bundle): void  
+ onBackPress(): void  
+ onClick(view: View): void  
+ prepareToLeave(): void  
+ isFinished(): boolean  
+ setFinish(boolean): void  
+ uploadTables(response: JSONObject): void  
+ updateTables(response: JSONObject): void  
+ endProgressBar(): void
```

מסר המראה את טבלת המשחקים אונליין. ניתן להצטרף למשחק ולהגיע למסך ההכנה של המשחק, אך לא תהיה אפשרות לעירק את המשחק (מבחינת קלפים או הגדרות נוספות). כולל כפטור חזרה אל Game .Online Game `activity_online_tables`

TablesAdapter:

```
+ TablesAdapter extends ArrayAdapter  
  
fields  
- context Context  
- data List<Table>  
- tempTable Table  
- onlineTablesActiv... OnlineTablesActiv...  
- sharedPreferences SharedPreferences  
  
construct...  
+ TablesAdapter(context: Context, resource: int, textViewResource: int, data: List<Table>: List<Table>, onlineTablesActiv... OnlineTablesActiv...)  
  
methods  
+ getView(position: int, convertView: View, parent: ViewGroup): View  
+ isEnabled(position: int): boolean
```

מחלקה שמעדכנת ב ListView `OnlineTablesActivity` ובמסגרת בתוכו. המחלקה מכילה את המשחקים שנינט לשחק בהם (נלקח מהשרת) ונותנת אפשרות להיכנס למשחק (יש כפטור play בכל שורה בטבלה). ה layout של כל table הוא `table_layout`

Table:

```
+ Table  
  
fields  
- creator String  
- id String  
  
construct...  
+ Table(creator: String, id: String)  
  
methods  
+ getCreator(): String  
+ setCreator(creator: String): void  
+ getId(): String  
+ setId(id: String): void
```

מחלקה ששומרת את כל התכונות של table זה המשחק (בdomינו) שהם מי יוצר את המשחק ואת ה-ID של המשחק. המחלקה מוכלת ב TablesAdapter שבה נשמרים כל המשחקים.

RequestAdapter:

```
+ RequestAdapter
fields
- requestQueue: RequestQueue
- instance: RequestAdapter
- ctx: Context
construct...
- RequestAdapter(context: Context)
methods
+ synchronize... getInstan... (context: Context) RequestAdapter
+ getRequestQueue(): RequestQueue
+ addToRequestQueue(req: JsonObjectRequ...): void
```

מחלקה שמנהלה את כל בקשות ה `http` של האפליקציה אשר נקבעו מהמחלקות `GameRequests` ו `LoginRequests`. היא כוללת `RequestQueue` שזה תור של בקשות כבאה באופן סינכרוני מעלה בקשות לתור ומפעילה בקשה אחר בקשה.

PrepareGameActivity:

```
+ PrepareGameActivity exten... AppCompatActivity...
imlemen... View.OnClickListener...
CompoundButton.OnCheckedChangeListener...
fields
- gameManagerBeforeSt... GameManagerBeforeSt...
- btnReady: Button
- btnLeaveTable: Button
- tvP1: TextView
- tvP2: TextView
- tvIsRated: TextView
- tvWinner: TextView
- tvP1vP: TextView
- tvP2vP: TextView
- sw: Switch
- background: ImageView
- progressDialog: ProgressDialog
- sound: ImageView
- sharedpreferences: SharedPreferences
construct...
methods
# onCreate(savedInstanceState: Bundle): void
+ onBackPress... (): void
+ onClick(view: View): void
+ updateUi(onlyMe: boolean): void
+ getGameManagerBeforeSt...(): GameManagerBeforeSt...
+ leaveTable(): void
+ prepareToLeave(): void
+ startGa... (): void
# onActivityResult(requestCode: int, resultCode: int, intent: Intent): void
+ onCheckedChange(compoundButt... CompoundButt... isRated: boolean): void
+ endProgressBar(): void
```

מסמך שבו רואים את השחקנים שמצטרפים למשחק. ניתן לעורר את המשחק, אם יש לך הרשאה, כמובן אם יצרת את המשחק. אם כל השחקנים המציגים מאשרים שהם מוכנים להתחילה המשחק, עוברים אל מסך ה `GameActivity`. אותו מסך משמש גם בסוף המשחק שמסך שבו רואים את תוצאות המשחק. ניתן לראות את טבלת קלפי הניצחון, מי ניצח וכי הפסיד. יש גם אפשרות להתחילה את המשחק מחדש עם אותם אנשים. כולל כפתור חזרה אל `layout` של מסך זה הוא `.activity_prepare_game`

GameManagerBeforeStart:

```
+ GameManagerBeforeStart
imlemen... Serializable
fields
- gameId: String
- idP1: String
- idP2: String
- isCreator: boolean
- isReady1: boolean
- isReady2: boolean
- isStar1: boolean
- isStar2: boolean
- isRated: boolean
construct...
+ GameManagerBeforeSt... (creatorId: String)
+ GameManagerBeforeSt... (tableId: String, nonCreatorId: String)
methods
+ restartReady(): void
+ getGameId(): String
+ setGameId(gameId: String): void
+ getIdP1(): String
+ setIdP1(idP1: String): void
+ getIdP2(): String
+ setIdP2(idP2: String): void
+ isCreator(): boolean
+ setCreator(creator: boolean): void
+ isReady1(): boolean
+ setRead... (ready1: boolean): void
+ isReady2(): boolean
+ setRead... (ready2: boolean): void
+ isStar1(): boolean
+ setStar... (start1: boolean): void
+ isStar2(): boolean
+ setStar... (start2: boolean): void
+ isRated(): boolean
+ setRated(rated: boolean): void
+ getMyId(): String
+ getEnemyId(): String
+ gameManagerBeforeStartToJ... (): JSONObject
+ jsonToGameManagerBeforeS... (jsonObj... JSONObject, isCreator: boolean, afterReady: boolean): void
```

מחלקה ששומרת בתוכה את ה-`ip` של המשחק, את השמות של שני השחקנים שבמשחק, אם שני השחקנים מוכנים ותוכנות נוספות על המשחק. **Prepare Game** מוכנת במחלקות `GameManager` ו `Activity` את כל המידע החינוי על המשחק על מנת לתקשר עם השרת, כמובן ברוב בקשות המשחק, מביאים את `GameManagerBeforeStart` כ `json` על מנת שהשרת יידע לאן להכנס או מאיפה לשולף את המידע. בנוסף המחלקה שומרת האם שני השחקנים מוכנים (כלומר אם לחזו על `ready` במסך `Prepare Game Activity` והאם לחזו על `Game Activity start`

GameActivity:

```

- GameActivity extends AppCompatActivity
  implements View.OnClickListener,
             View.OnLongClickListener

fields
- game: GameManager
- btnStart: Button
- btnEnd: Button
- btnAutoplay: Button
- btnAction: ArrayList<Button>
- tvTurn: TextView
- clTurn: ConstraintLayout
- victoryCards: HashMap<String, View>
- treasureCards: HashMap<String, View>
- actionCards: HashMap<String, View>
- actionBigCar: HashMap<String, View>
- cardDialog: Dialog
- myDeck: View
- enemyHand: View
- enemyDeck: View
- ivEnemyDiscard: ImageView
- ivMyDiscard: ImageView
- tvMyVP: TextView
- tvEnemyVP: TextView
- tvMyName: TextView
- tvEnemyName: TextView
- lvLog: ListView
- logAdapter: LogAdapter
- rvTrash: RecyclerView
- trashAdapter: TrashAdapter
- tvInfoTitle: TextView
- rvHand: RecyclerView
- handAdapter: Adapter
- rvActionCardsPlaying: RecyclerView
- actionCardsPlayingAdapter: Adapter
- clActionCardsPlaying: ConstraintLayout
- ivArrowActionCardsPlaying: ImageView
- positionOfIRActionCardsPlaying: String
- background: ImageView
- progressDialog: ProgressDialog
- btnKingdomOrPlayer: Button
- btnTrashOrLog: Button
- btnResign: Button
- clPlayArea: ConstraintLayout
- clKingdom: ConstraintLayout
- rlButtonsInPlayer: RelativeLayout
- phoneCallReceiver: PhoneCallReceiver
- construct...
methods
+ onCreate(savedInstanceState: Bundle): void
+ checkAndRequestPermissions(requestCode: Int, permission: String[], grantResults: Int[]): void
+ onDestroy(): void
+ onBackPress(): void
+ onLongClick(v: View): boolean
+ useCard(cardName: String): void
+ buyCard(cardName: String): void
+ onClick(view: View): void
+ beforeStart(): void
+ startGame(): void
+ insertDataToHashMap(): void
+ endGame(): void
+ startPrepareActivity(): void
+ uploadBoard(): void
+ uploadByType(hashMap: HashMap<String, View>): void
+ updateCards(updateHand: boolean): void
+ updateCountOfBoard(): void
+ updateCountByType(hashMap: HashMap<String, View>): void
+ turnAction(): void
+ turnUI(): void
+ actionsUI(): void
+ buysUI(): void
+ waitForHand(cardName: String, minAmount: int, maxAmount: int, typeOfAction: String, handleClickOnCard: boolean): void
+ waitForBoard(cardName: String, minAmount: int, maxAmount: int): void
+ setVisibilityForAction(i: int, visibility: int): void
+ invisibleButton(n: int): void
+ uploadActionButton(textOnButton: String, hasUndoAndConfirm: boolean): void
+ updateActionButton(): void
+ getHandAdapter(): Adapter
+ getActionCardsPlayingAdapter(): Adapter
+ setVisibilityForRVActionCardsPlaying(visibility: int): void
+ getTrashAdapter(): Adapter
+ endProgressBar(isHandleBackground: boolean): void

```

מסך המשחק שமולק לחלקים רבים: כל קלפי המשחק, הקלפים שיש לי ביד, כמהות הקלפים ב deck עבורי וuboar היריב, הקלף האחרון ב discard עבורי וuboar היריב, נקודות הניצחון עבורי וuboar היריב, ergo של המשחק שסוקר את כל הפעולות שנעשות, מסך הודעות שניתן לכתחוב (עדין לא פעול) וכפתורים שונים, למשל כפתור כדי להיכנע ולסימן את המשחק. המסך מכיל את **GameManager**, **LogAdapter**, **TrashAdapter**, **HandAdapter**, **PhoneCallReceiver** ו **ActionCardPlayingAdapter** ה שאל layout activity_game שכולל בתוכו עוד כמה של layouts (קלפים של board (קלפים של kingdom (קלפים שבאזור kingdom (קלף המוגדל שמצוג בדיאלוג (קלף המוגדל שמצוג בדיאלוג

GameManager:

```
+ GameManager
fields
- final int numberOfCards
- final int victoryAmount
- final int curseAmount
- final int actionAmount
- final int copperAmount
- final int silverAmount
- final int goldAmount
- final int decksToEndGame
- gameManagerBeforeStart... GameManagerBeforeSt...
- gameActivity... GameActivit...
- actionCards String[]
- player Player
- enemyData EnemyData
- board HashMap<String, Integer>
- trash HashMap<String, Integer>
- arrayTrash ArrayList<Pair<String, Integer>>
- turn Turn
- log ArrayList<LogLine>
- isStart... boolean
- isGameEnd... boolean
- resigned boolean
- playsAttack... boolean
- doneAttack... boolean
- timer Stack<Integer>
construct...
+ GameManager(gameManagerBeforeSt..., GameManagerBeforeSt..., gameActivit... GameActivit...)
methods
+ startUploadAnd... () void
+ beforeGame() void
+ startGame() void
+ addTurnNumberToLog() void
+ useCard(cardName String, n int, forceUse boolean, autoPlay boolean) void
+ playCardOrAddToWaitList(cardName String, n int, i int, forceUse boolean, autoPlay boolean) void
+ playCard(cardName String, n int, i int, forceUse boolean, autoPlay boolean) void
+ useAfterPL_(cardNameUsed String, addWallForEnemy boolean) void
+ endCard() void
+ buyCard(cardName String) void
+ getCard(cardName String) String
+ cleanUpPhase() void
+ addToDiscardByType(type String) void
+ autoPlayTreasure() void
+ addHashToLog hm HashMap<String, Integer>, action String, textBefore String) void
+ getActionCar... () String
+ setActionCar... (actionCards String) void
+ getBoard() HashMap<String, Integer>
+ setBoard(board HashMap<String, Integer>) void
+ getTrash() HashMap<String, Integer>
+ setTrash(trash HashMap<String, Integer>) void
+ getArrayTrash() ArrayList<Pair<String, Integer>>
+ addToArrayTrash(cardName String, n int) void
+ updateArrayTrash() void
+ getPlayer() Player
+ setPlay... (player Player) void
+ gameEnd(boolean) void
+ changeTurn() void
+ getTurn() Turn
+ setTurn(turn Turn) void
+ getLog() ArrayList<LogLine>
+ deleteLastLineFromLog() void
+ getLastLineFromLog() LogLine
+ getTimes() Stack<Integer>
+ getTabs() int
+ getGameActivit... () GameActivit...
+ setGameActivit... (gameActivit... GameActivit...) void
+ gameManagerToJsonS... () JSONObje...
+ gameManagerToJsonRealTim() JSONObje...
+ jsonToGameManagerS... (jsonObje... JSONObje...) void
+ jsonToGameManagerRealTl... (jsonObje... JSONObje...) void
+ myDataToJson() JSONObje...
+ logToJson() ArrayList<JSONObje...
+ jsonToLog(jsonLogLine... ArrayList<JSONObje...>) void
+ getGameManagerBeforeSt... () GameManagerBeforeSt...
+ getEnemyD... () EnemyData
+ isStart... () boolean
+ setStart... (started boolean) void
+ isGameEnd... () boolean
+ setGameEnd... (gameEnded boolean) void
+ isResigned() boolean
+ setResign... (resigned boolean) void
+ isPlaysAttack... () boolean
+ setPlaysAttack... (playsAttack... boolean) void
+ isDoneAttack... () boolean
+ setDoneAttack... (doneAttack... boolean) void
```

מחלקה ששומרת בתוכה את כל המידע על המשחק עבור כל שחקן ומיליה מחלקות רבות, כמו עוד. המחלקה מכילה ומוכלת במסך GameManagerBeforeStart, Turn, Player GameRequests ולשלוף מהשרת את המידע הרלוונטי. המחלקה דוגמת תחילת המשחק ובודקת את סיום המשחק, היא דוגמת להחליף בין התורות ולעדכן את pog של המשחק לכל שחקן. יש במחלקה זו גם קבועים רבים שקובעים את אופי המשחק וניתן לשנות אותם. בנוסף, המחלקה מגילה בתחילת המשחק את קלפי הפעולה שישווקו במשחק.

PlayCardArguments:

```
+ PlayCardArguments
fields
- cardName: String
- n: int
- i: int
- forceUse: boolean
- autoPl...: boolean
construct...
+ PlayCardArgume... (cardName: String, n: int, i: int, forceUse: boolean, autoPl...: boolean)
methods
+ getCardName(): String
+ getN(): int
+ getI(): int
+ isForceU...(): boolean
+ isAutoPl...(): boolean
```

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

GameRequests:

```
+ GameRequests
fields
construct...
methods
+ getServerIP(): String
+ getPort(): String
+ creator_st... (prepareGameActivit... PrepareGameActivit...): void
+ non_creator_st... (prepareGameActivit... PrepareGameActivit...): void
+ uploadAndGetPlayerD... (gameManager.GameManager): void
+ uploadDataInGame(isStart: boolean, gameManager.GameManager): void
+ getDataInGame(isStart: boolean, gameManager.GameManager): void
+ getTables(onlineTablesActivit... OnlineTablesActivit...): void
+ uploadGameManagerBeforeS... (prepareGameActivit... PrepareGameActivit...): void
+ updateReady(prepareGameActivit... PrepareGameActivit...): void
+ updateReadyToStar(gameManager.GameManager): void
+ waitForPlayerToEnterTa... (prepareGameActivit... PrepareGameActivit...): void
+ waitForStartGa... (gameManager.GameManager): void
+ waitForStartGa... (isCreator: boolean, prepareGameActivit... PrepareGameActivit..., countChe... int): void
+ delete_game_manager_before_... (prepareGameActivit... PrepareGameActivit...): void
+ deletePz(prepareGameActivit... PrepareGameActivit...): void
+ endGame(gameManager.GameManager): void
```

מחלקה סטטית שמעלה בקשות בנוגע למשחק לשרת, למשל קבלת מידע "real time" והעלאת מידע. הפעולות ב"real time" נקראות עיקרי מGameManager ולפחות צו נקשרו בין השרת לבין כלומר בין שני השחקנים במהלך המשחק, לפני ואחריו.

LogLine:

```
+ LogLine
fields
- text: String
- playerId: String
- isBold: boolean
- isItalic: boolean
- color: String
- tabs: int
- type: String
construct...
+ LogLine(te... String, playerId: String, isBold: boolean, isItalic: boolean, tabs: int, type: String)
+ LogLine(te... String, playerId: String, isBold: boolean, isItalic: boolean, color: String, tabs: int, type: String)
+ LogLine(jsonObj... JSONObje...)
methods
+ getText(): String
+ setText(te... String): void
+ getPlayerId(): String
+ setPlayerId(playerId: String): void
+ isBold(): boolean
+ setBold(bold: boolean): void
+ isItalic(): boolean
+ setItalic(italic: boolean): void
+ getColor(): String
+ setColor(color: String): void
+ getTabs(): int
+ setTabs(tabs: int): void
+ getType(): String
+ setType(type: String): void
+ logLineToJson(): JSONObje...
```

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

Help:

```
+ Help
- fields
- construct...
- methods
+ hashToArray(hm: HashMap<String, Integer>): ArrayList<String>
+ sizeOfHashMap(hm: HashMap<String, Integer>): int
+ arrayListOfPairsToArray(alPairs: ArrayList<Pair<String, Boolean>>): ArrayList<String>
+ nameToCard(name: String): Card
+ getExpansionCards(expansion: String): Set<String>
+ base(): Set<String>
+ getCardsRandom(expansion: String): String[], int numberOfCards, String cardsToInsert: String)
+ sort(cards: String[]): String[]
```

מחלקה סטטית שכוללת פעולות כלליות במשחק, בין היתר למשל פעולה שמחזירה כמות הקლפים שיש בדיקת hashmap שהוא סכום של כל הערכים. בנוסף, במחלקה זו יש פעולה חשובה מאוד שמירה שם של קלף למחלקה של הקלף עצמו (על ידי switch case). בנוסף יש פעולה אשר יוצרת חבילת קלפים רנדומלית למשחק חדש.

Turn:

```
+ Turn
- fields
- phase: String
- turnId: String
- actionCardsPlayed: ArrayList<String>
- treasureCardsPlayed: HashMap<String, Integer>
- cardsBought: HashMap<String, Integer>
- lastActionCardForPlayer: String
- cardsForWaitForPlayer: ArrayList<PlayCardArgument>
- lastCardForWaitForPlayer: String
- isWaitingForEnemy: boolean
- waitForFunction: WaitForFunction
- actions: int
- buys: int
- treasure: int
- turnNumber: int
- forcedActionEnabled: boolean
- construct...
+ Turn(turnId: String)
+ Turn()
- methods
+ getActionCardsPlayed(): ArrayList<String>
+ setActionCardsPlayed(actionCardsPlayed: ArrayList<String>): void
+ getTreasureCardsPlayed(): HashMap<String, Integer>
+ getCardsBought(): HashMap<String, Integer>
+ getPhase(): String
+ setPhase(phase: String): void
+ getTurnId(): String
+ setTurnId(turnId: String): void
+ changeTurn(gameManagerBeforeStart: GameManagerBeforeStart): void
+ isMyTurn(gameManagerBeforeStart: GameManagerBeforeStart): boolean
+ addActionCard(cardName: String): void
+ addWaitForEnemy(cardName: String): void
+ addWaitForPlayer(cardName: String, n: int, i: int, forceUse: boolean, autoPlay: boolean): void
+ removeLastAttack(): void
+ getLastActionCardForPlayer(): String
+ setLastActionCardForPlayer(lastActionCardForPlayer: String): void
+ getCardsForWaitForPlayer(): ArrayList<PlayCardArgument>
+ setCardsForWaitForPlayer(cardsForWaitForPlayer: ArrayList<PlayCardArgument>): void
+ getLastCardForWaitForPlayer(): String
+ setLastCardForWaitForPlayer(lastCardForWaitForPlayer: String): void
+ setWaitingForEnemy(waitingForEnemy: boolean)
+ addTreasureCard(cardName: String): void
+ addCardBought(cardName: String): void
+ getActions(): int
+ setActions(actions: int): void
+ addActions(actions: int): void
+ useActions(): void
+ getBuys(): int
+ setBuys(buys: int): void
+ addBuys(buys: int): void
+ useBuy(): void
+ getTreasure(): int
+ setTreasure(treasure: int): void
+ addTreasure(treasure: int): void
+ useTreasure(treasure: int): void
+ getTurnNumber(): int
+ setTurnNumber(turnNumber: int): void
+ getWaitForFunction(): WaitForFunction
+ getForcedActionEnabled(): boolean
+ setForcedActionEnabled(forcedActionEnabled: boolean): void
+ toString(): String
+ turnToJson(doneAttacks: boolean): JSONObject
+ jsonToTurn(jsonObject: JSONObject): gameManager.GameManager
```

מחלקה שמכילה במחלקה GameManager ושומרת את המידע הרלוונטי לתור. בתוכה יש פעולה changeTurn שמשנה את התור ומאפשרת את הערכים בturn כדי שככל תור יהיה חדש. הערכים בturn בעליים בGameManager(text view) שמאירה את המידע הרלוונטי לשחקן.

WaitForFunction:

```
+ WaitForFunction
fields
- isWaitingForActionCardsDial ... boolean
- isWaitingForButtonsO... : boolean
- isWaitingForBoa... : boolean
- isWaitingForHa... : boolean
- cardName: String
- minAmou... int
- maxAmou... : int
- minPriceForGa... int
- maxPriceForG... : int
- typeOfActi... String
- cardsForActionCardPl...: HashMap<String, Integ...
- cardsForDial...: ArrayList<Pair<String, Boolean...
- handleClickOnC... : boolean
- hasUndoAndConfirm: boolean
construct...
+ WaitForFunct... ()
methods
+ handleWaitingForHa... (cardName: String, minAmou... int, maxAmou... int, typeOfActio... String, handleClickOnCar.. boolean): void
+ handleWaitingForActionCardsDia... (cardName: String, minAmou... int, maxAmou... int, typeOfActio... String, handleClickOnCar.. boolean, cards: ArrayList<String... ): void
+ addToCardsForDialog(card: String): void
+ handleWaitingForBo... (cardName: String, minAmou... int, maxAmou... int): void
+ handleWaitingForButtonsO... (cardName: String): void
+ clear(): void
+ undo(): void
+ cardsSelectedForDial...(): ArrayList<String...
+ cardsLeftForDialog(): ArrayList<String...
+ updateCardsForDialogByPosit... (position: int, isSelected... : boolean): void
+ order(): void
+ insertCardSelectedInHa... (cardName: String): void
+ getCardAmountInCardsInHan( cardName: String): int
+ isHandleClickOnC... (): boolean
+ setHandleClickOnC... (handleClickOnCar.. boolean): void
+ getMinPriceForG... (): int
+ setMinPriceForG... (minPriceForGal...int): void
+ getMaxPriceForG... (): int
+ setMaxPriceForG... (maxPriceForGal...int): void
+ getCardsForActionCardPl...(): HashMap<String, Integ...
+ setCardsForActionCardP... (cardsForActionCardPl...: HashMap<String, Integ... ): void
+ getCardsForDial... (): ArrayList<Pair<String, Boolean...
+ setCardsForDial... (cardsForDial...: ArrayList<Pair<String, Boolean... ): void
+ isWaitingForActionCardsDial...(): boolean
+ setWaitingForActionCardsDial... (waitingForActionCardsDial... boolean): void
+ isWaitingForButtonsO... (): boolean
+ setWaitingForButtonsO... (waitingForButtonsOn...boolean): void
+ isWaitingForBoa... (): boolean
+ setWaitingForBo... (waitingForBoard boolean): void
+ isWaitingForHa... (): boolean
+ setWaitingForHa... (waitingForHand boolean): void
+ getCardName(): String
+ setCardName(cardName: String): void
+ getMinAmo... (): int
+ setMinAmo... (minAmou... int): void
+ getMaxAmo... (): int
+ setMaxAmo... (maxAmou... int): void
+ getTypeOfActi... (): String
+ setTypeOfActi... (typeOfActio...String): void
+ isHasUndoAndConfi... (): boolean
+ setHasUndoAndConfi... (hasUndoAndConfirm boolean): void
```

מחלקה שמודלת במחלקה Turn ושומרת את המידע הרלוונטי לכל הדברים שבו יש לחכות במהלך המשחק, למשל לשחק היריב, לבחירת מסוימות ועוד. המשתנים במחלקה זו מוגבלים את המשחק מלממשכו בקר שיש בדיקה בכל פעולה האם השחקן מכחה למשהו ורק אם הוא לא מכחה, הוא יכול להמשיך משחק.

Player:

```
+ Player
fields
- discard: ArrayList<String...
- deck: ArrayList<String...
- hand: HashMap<String, Integer...
- arrayHand: ArrayList<Pair<String, Integer...
construct...
+ Player()
methods
+ discardToDeck(... (gameManager.GameManager, tabs.int): void
+ handToDeck(cardName.String, gameActivit... GameActivi..): void
+ putArrayInDiscard(cards: ArrayList<String... ): void
+ putArrayInDe... (cards: ArrayList<String... ): void
+ takeCardsToHand(numberOfCards.int, gameActivit... GameActivi..., gameManager.GameM anager, tabs.int): void
+ takeCards:numberOfCards.int, gameManager.GameManager, tabs.int): ArrayList<String...
+ containsTypeCar... (type: String): boolean
+ containsC... (cardName: String): boolean
+ allCards(gameManager.GameManager: HashMap<String, Integer...
+ getVictoryPoi... (gameManager.GameManager: int
+ getDiscard(): ArrayList<String...
+ getDeck(): ArrayList<String...
+ addToDeck(cardName.String): void
+ addToDiscard(cardName.String): void
+ addToHand(cardName.String): boolean
+ removeFromHand(cardName.String): boolean
+ getHand(): HashMap<String, Integer...
+ setHand(hand: HashMap<String, Integer... ): void
+ getArrayHand(): ArrayList<Pair<String, Integer...
+ getPositionByName... (cardName.String): int
+ updateArrayHand(): void
```

מחלקה שמכילה במחלקה GameManager ושמירת את המידע הרלוונטי לכל שחקן לגביו הקלפים שיש לו – הקלפים ביד, הקלפים ששחן והקלפים לדעת וחלק מהמידע גם אותו שחקן לא צריך לדעת ולכן לא מעלים אותו לשרת, אלא הוא נשמר במחלקה זו. המחלקה דואגת גם ללקחת קלפים ליד ולעוד פעולות שמתבצעות במהלך המשחק.

EnemyData:

```
+ EnemyData
fields
- lastCardOnDiscard: String
- deckSize: int
- handSize: int
- victoryPoint: int
- isEmpty: boolean
- resigned: boolean
construct...
+ EnemyData()
methods
+ getLastCardOnDiscard(): String
+ setLastCardOnDiscard(lastCardOnDiscard: String): void
+ getDeckSize(): int
+ setDeckSize(deckSize: int): void
+ getHandSize(): int
+ setHandSize(handSize: int): void
+ getVictoryPoint(): int
+ setVictoryPoint(victoryPoint: int): void
+ isEmpty(): boolean
+ isResigned(): boolean
+ jsonToEnemyD... (jsonObjec... JSONObjec..., gameManager.GameManager): void
```

מחלקה שמכילה במחלקה GameManager ושמירת את המידע הרלוונטי על כל שחקן, אבל רק המידע שככל שחקן במשחק אמור לדעת, למשל כמות הקלפים שיש לשחקן ביד והקלף האחרון בdiscard. מידע זה מועלה לשרת ונשלח מהשחקן השני על מנת לעדכן על המסך GameActivity בכל המידע הנדרש על היריב.

Card:

```
+ Card
fields
- type: String
- name: String
- price: int
- imageSour... int
- shortImageSour... int
construct
+ Card(name:String, price:int, imageSour... int, shortImageSour... int, type:String)
+ Card(type: String)
methods
+ getImageSource(): int
+ setImageSour... (imageSour... int): void
+ getShortImageSour... (): int
+ setShortImageSour... (shortImageSour... int): void
+ getPrice(): int
+ getName(): String
+ setName(name:String): void
+ setPrice(price:int): void
+ getType(): String
+ setType(type: String)
+ play(game.GameManager, gameActivit... GameActivi...): void
+ atta... (game.GameManager, gameActivit... GameActivi...): void
+ reaction(game.GameManager, gameActivit... GameActivi...): void
+ enemyPl... (game.GameManager, gameActivit... GameActivi...): void
+ handleButtonClic... (buttonText:String, game.GameManager, gameActivit... GameActivi...): void
+ handleClickOnHandOrDial... (cardName String, game.GameManager, gameActivit... GameActivi...): void
+ isMarkCardSelectedFromHandWhenHa... (): boolean
+ isCardToUse(cardName String, game.GameManager, gameActivit... GameActivi...): boolean
+ isCardToGetFromBo... (cardName String, game.GameManager, gameActivit... GameActivi...): boolean
+ clickOnBo... (cardName String, game.GameManager, gameActivit... GameActivi...): void
+ getValue(gameManager.GameManager): int
+ getNameToDisplay(): String
+ afterPlay(game.GameManager, cardNameUsed:String): void
+ toString(): String
```

מחלקה אבסטרקטית של קלף שכוללת את הסוג שלו, השם שלו, המחיר שלו ועוד. במחלקה זו יש פעולה אבסטרקטית, `play`. שמנומשת עבור כל הקלפים שיורשים מהמחלקה. מהמחלקה יורש כל סוג קלף במשחק (יש כאן דוגמאות לכמה קלפים). יש במחלקה פעולות רבות נוספות שנויות לדרישת על ידי קלפים שיורשים מ `Card`. ב `GameManager` ובמחלקות שהן תכונה שלו נשמר אך ורק שם הקלף. על מנת להפעיל פעולה במחלקה של קלף מסוים, קיימת הפעולה `nameToCard` שנמצאת ב `Help` והוא מmirה שם הקלף למחלקה של `Card`.

Copper:

```
+ Copper exten... Card
fields
construct...
+ Copper()
methods
+ play(game.GameManager, gameActivit... GameActivi...): void
```

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

Festival:

```
+ Festival exten... Card
fields
construct...
+ Festival()
methods
+ play(game.GameManager, gameActivit... GameActivi...): void
```

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

Laboratory:

```
+ Laboratory exten... Card
fields
construct...
+ Laboratory()
methods
+ play(game.GameManager, gameActivit... GameActivi...): void
```

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

מחלקה שמעדכנת ב ListView שבמסגר Game Activity את כל שורות ה log שבסחק ומצבת כל שורה בהתאם לתוכנות שלה ב LogLine. המחלקה מוכלת ב Game Activity ומיכילה מצביע ל GameActivity. ה layout של GameManager כל שורה הוא log_layout.

LogAdapter:

```
+ LogAdapter exten... ArrayAdapter...
fields
~ context: Context
~ data: ArrayList<LogLin...
~ gameManager: GameManager
construct...
+ LogAdapter(context: Context, resource: int, textViewResource: int, data: ArrayList<LogLin..., gameManager: GameManager)
methods
+ getView(position: int, convertView: View, parent: ViewGroup): View
+ isEnabled(position: int): boolean
```

HandAdapter:

```
+ HandAdapter exten... RecyclerView.Adapter...
fields
- objec... ArrayList<Pair<String, Integer...
- gameActivi... GameActivi...
construct...
+ HandAdapter(object: ArrayList<Pair<String, Integer..., gameActivit... GameActivi...)
methods
+ onCreateViewHolder(parent: ViewGroup, viewType: int): MyViewHolder
+ onBindViewHolder(holder: MyViewHolder, position: int): void
+ getItemCount(): int
```

מחלקה שמעדכנת ב RecyclerView שבמסגר Game Activity את כל הקלפים שיש לשחקן ביד. המחלקה מוכלת ב Game Activity ומיכילה מצביע ל GameActivity. ה layout של כל סוג קלף ביד הוא .card

TrashAdapter:

```
+ TrashAdapter exten... RecyclerView.Adapter...
fields
- objec... ArrayList<Pair<String, Integer...
- gameActivi... GameActivi...
construct...
+ TrashAdapter(object: ArrayList<Pair<String, Integer..., gameActivit... GameActivi...)
methods
+ onCreateViewHolder(parent: ViewGroup, viewType: int): MyViewHolder
+ onBindViewHolder(holder: MyViewHolder, position: int): void
+ getItemCount(): int
```

מחלקה שמעדכנת ב RecyclerView שבמסגר Game Activity את כל הקלפים שיש ב trash במשחק. המחלקה מוכלת ב Game Activity ומיכילה מצביע ל GameActivity. ה layout של כל סוג קלף ב trash הוא .card_trash

ActionCardPlayingAdapter:

```
+ ActionCardPlayingAdapter exten... RecyclerView.Adapter...
fields
- objec... ArrayList<Pair<String, Boolean...
- gameActivi... GameActivi...
construct...
+ ActionCardPlayingAdap... (object: ArrayList<Pair<String, Boolean..., gameActivit... GameActivi...)
methods
+ onCreateViewHolder(parent: ViewGroup, viewType: int): MyViewHolder
+ onBindViewHolder(holder: MyViewHolder, position: int): void
+ getItemCount(): int
```

מחלקה שמעדכנת ב RecyclerView Game Activity על המסגר GameActivity. שמודעה על המסגר GameActivity. אם מופעל סוג מסוים של קלפי פעולה ובודק את הקלפים שם בהתאם לקלף שהופעל. המחלקה מוכלת ב GameActivity. המיכילה מצביע ל GameActivity. ה layout של כל סוג קלף ב Dialog זה הוא .card_for_action_cards_playing

PhoneCallReceiver:

```
+ PhoneCallReceiver exten... BroadcastReceiver  
- fields  
- construct...  
- methods  
+ onReceive(context: Context, intent: Intent): void
```

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

MusicService:

```
+ MusicService exten... Service  
- fields  
- mediaPlay...: MediaPlay...  
- construct...  
+ MusicServ... ()  
- methods  
+ onStartComma... (intent: Intent, flags: int, startId: int): int  
+ onBind(intent: Intent): IBinder  
+ onDestroy(): void
```

מחלקה שירושת מ Service ומפעילה מוזיקת רקע. בכל Activity בלבד GameActivity ניתן להפעיל ולכבות את המוסיקה על ידי כפתור בצד המסך.

מסכים נוספים שיתווסף ועד לא קיימים

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

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

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

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

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

מסר Choose Cards - מסר שבו ניתן לבחור את הקלפים עבור המשחק (בדרכ' זה רנדומלי). יחד עם אופציות לשחק רק עם הקלפים האוהבים ולא הקלפים שלא אוהבים. כולל כפתור אישור שמחזיר אל Prepare Table.

אלגוריתמים מעוניינים

. 1 - פועלה שנמצאת במחלקה Player ומטרתה לחתת 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 בלולאה for ככמות הקלפים שציריך לחתת בתחילת כל תור (קבוע שנקרא `numberOfCards` ובדרכו כלל שווה 5). הלולאה תעבור ככמות `numberOfCards` לאם גם `deck` ריק וגם `hand` המשמעות היא שלא נותרו לו קלפים לחתת, ובמצב זה יש לסיים את הלולאה. בתוך הלולאה בודקים אם `deck` ריק ואם כן אז קוראים לפעולה `discardToDeck` שמעבירה את כל הקלפים מה-`deck` ל-`discard` כמו שציריך לעשות לפי החוקים. לאחר מכן הפעולה בודקת האם ביד השחקן כבר קיימים קלף מסווג שנשלה מ-`deck` (הקלף האחרון שב-`deck`). במקרה שאכן הוא קיימ, הפעולה פשוט מוסיפה אחד לערך ב-`hashmap` של אותו קלף שנשלה, אחרת הפעולה יוצרת ערך חדש ב-`map` בשם של הקלף שעומד להישלח ומכוונסה לערך אחד, כי יש רק קלף אחד צזה. כשהלולאה נגמרת ונשלפו הקלפים, הפעולה מעדכנת את הקלפים שביראו על המסר

שנשלפו ((notifyDataSetChanged()) ולבסוף מוסיפה ל log של המשחק שאותו שחקן
לפחות 5 קלפים.

.2 – פועלה שנמצאת ב GameManager ומפעילה את הקלף שהצץ עליי playCard .
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 לאוטו שחקן שהפעיל את הקלף אותו קלח מהיד, מוסיפה אל ה log על
כך שהשחקן השתמש בקלף זהה (יש אפשרות לשולשה תנאים שלא אפרט כי הם קשורים

לחוקי המשחק עצמו, אבל באופן כללי דואגים למספר הטעבים שציריך בסוגו, כלומר אם זו הפעולה של קולף שנבעה מהפעלה קודמת או הפעלה לאחר שמהו room (throne room). הפעולה מורידה לשחקן פעולה אחת כיון שהשתמש בקהלף בתנאי שהקהלף לא הופעל ב"הכרכה", כלומר שלא צריך להוריד על נספחה. לבסוף, הפעולה מוסיפה אל הקლפים ששוחקנו את אותו קולף בהתאם לסוגו (...action, treasure). לבסוף הפעולה מפעילה את הקולף על ידי play שפועלת על ה class המתאים לשם של הקולף אשר יירוש מ Card. מכיוון שככל המחלקות של הקלאסים יירושו מ Card, ניתן פשוט להפעיל על כל קולף את הפעולה play והיא תבצע את הפעולה שנדרצה במחלקה הספציפית של הקולף.

3. 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 מתאפשרת בהעלאת בקשה http מסוג post מהלקוח אבל בפועל מתרחשת לו לאו שחוזרת על עצמה מתחילת המשחק ונגמר רק בסופו. הפעולה מקבלת חסז' שכולל את הנתונים של היריב. בנויסוף קיימם המידע על השחקן שהעליה את הבקשת והוא מועלה לבסיס הנתונים במקומם שמור שהמפתח שלו הוא שמו של השחקן. לאחר מכן הפעולה מקבלת מבסיס הנתונים את המידע על השחקן השני (רק המידע שמותר לו לדעתן הסתם). אם הוא מצא את המידע, הוא מחזיר חסז' של המידע יחד עם success עם ערך true, כלומר הוא הצליח להביא את המידע ואם הוא לא הצליח, אז מוחזר חסז' עם ערך false.

שימוש ב-Resources

Broadcast Receiver

המחלקה BroadcastReceiver יורשת מ PhoneCallReceiver ויש בה את הפעולה `onReceive` שהיא פועלה שוכבה לדרוס כאשר יורשים מBroadcastReceiver. המחלקה `onReceive` מאפשרת לשינויים במצב השיחות של הטלפון, למשל אם מישחו 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 שמיישו 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, אך יש בדיקה של הגרסה). בנוסף, הפעולה שולחת הודעה 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 `Broadcast` היא שבמהלך המשחק לא יפריעו למשתמש עם שיחות נוכנות, מה שעלול להקשות עליו ולהוציא אותו מריכוז. כך יותר לשחק בעניין, כאשר מיידעים את מי שהתקשר שהוא עסוק כרגע ויחזור אליו אחר כך, וכך יצרתי את אותו `Broadcast` `Receiver`.

המחלקה מופעלת ב `GameActivity` כאשר מייצרים מופיע שלאה ומפעילים את הפעולה `registerReceiver`. אך לפני כן, נדרשות מהמשתמש הרשות על מנת לעשות זאת. הרשות אלן מתבקשות גם ב `manifest` אך המשתמש צריך לאשר אותן כדי להפעיל את ה `Broadcast`. הרשות הנדרשת הן: `SEND_SMS`, `READ_PHONE_STATE`, `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 עדין פתוחה. במחלקה יש את הפעולה `onStartCommand` שפועלת כאשר מפעילים את ה `Service`, את הפעולה `(Intent intent) onBind` שהיא פועלה שוכבה למשוך אך עצמו לא עושה כלום ואת הפעולה `(Service) onDestroy` שפועלת כশמכבים את ה `Service`.

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

Thread

הפעולות שפונות אל השירות כבקשות http מופעלות באופן אוטומטי על thread אחר, כאשר היא מחייבת לתשובה מהשרת. לאחר יצירת הבקשה, מכנים את הבקשה לטור של בקשות http שנקרא מהמחלקה RequestAdapter. לאחר שליחת הבקשה, מוחכים לתשובה מהשרת שקורא לפועלה onResponse, אך הקוד ממשיך לרווח ואינו מחייב לחרזת התשובה, שכן הוא פועל אוטומטית על thread אחר.

Firebase

השרת בפרויקט נעזר בסיס נתונים שנקרא firebase ומעלה אליו את כל המידע ושולף ממנו. המבנה של בסיס הנתונים הוא מבנה של json, כלומר לפי key | value (NoSQL). המבנה של השירות דינמי ולא חייב להיות קבוע מראש אך בכל זאת יש לשרת של 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 realtime database שמאפשר העלאת מידע על ידי הלקוחות בזמן אמיתי ובאופן סינכרוני והמידע נשאר זמן גם כאשר האפליקציה כביה. הנתונים ב firebase מחולקים למשחקים (games) ולמשתמשים (users), כאשר במשחקים נמצאים כל המשחקים שמשחקים כרגע ובמשתמשים יש את כל המשתמשים שנרשמו אי פעם יחד עם הסיסמות והמידע עליהם.

על מנת למש את העלאת המידע ושליפתו מה firebase דרך הפיתון, השתמשתי בסיפוריה של פיתון אשר נקראה firebase_admin שמאפשרת הצד השירות לשלב את firebase. בסיפוריה יש פעולות רבות שמאפשרות העלה, שטיפה,CHANKE, עדכון מידע בסיס הנתונים בקלות ובמהירות.

מבנה הנתונים ב firebase נראה כך (עמוד הבא):



טבלת פונקציות בפייטון

הסביר	הfonקציה
פעולה שמקבלת בקשה HTTP מסוג GET שיצרת משחק בשרת עם הפעולה <code>push</code> שיצרת מפתח רנדומלי ייחודי (אין עוד אחד כמוהו). הפעולה מחזירה חסז' שיש בו את מפתח המשחק והודעה בהתאם להצלחה או כשלון.	<code>@app.route('/creator_start', methods=['GET'])</code> <code>def creator_start():</code>
פעולה שמקבלת בקשה HTTP מסוג POST שהתבצע עם הפעולה <code>gameManagerBeforeStart</code> , את שבו יש את המפתח הייחודי של המשחק, את שמות השחקנים ועוד. הפעולה מוסיף את השחקן למשחק אם אין שני שחקנים כבר במשחק. הפעולה מחזירה חסז' שיש בו הודעה בהתאם להצלחה או כישלון.	<code>@app.route('/non_creator_start', methods=['POST'])</code> <code>def non_creator_start():</code>
פעולה שמקבלת בקשה HTTP מסוג POST שהתבצע עם הפעולה <code>gameManagerBeforeStart</code> , את שבו יש את המפתח הייחודי של המשחק, את שמות השחקנים ועוד עם 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 זהה לשרת. הפעולה מחזירה חסז' שיש בו הודעה בהתאם להצלחה או כשלון.	<code>@app.route('/upload_all_data', methods=['POST'])</code> <code>def upload_all_data():</code>
פעולה שמקבלת בקשה HTTP מסוג POST שהתבצע עם הפעולה <code>gameManagerBeforeStart</code> , את שבו יש את המפתח הייחודי של המשחק, את שמות השחקנים ועוד עם המידע הרלוונטי על המשחק אחרי תחילת המשחק. הפעולה מעלה את כל המידע זהה לשרת. מי שפעיל את הפעולה הוא מי שהטור שלו. הפעולה מחזירה חסז' שיש בו הודעה בהתאם להצלחה או כשלון.	<code>@app.route('/upload_real_time', methods=['POST'])</code> <code>def upload_real_time():</code>
פעולה שמקבלת בקשה HTTP מסוג POST	<code>@app.route('/get_all_data', methods=['POST'])</code>

<p>gameManagerBeforeStart</p> <p>ובהו json עם</p> <p>שבו יש את המפתח הייחודי של המשחק, את שמות השחקנים ועוד. הפעולה מקבלת את כל המידע הרלוונטי על המשחק לפני תחילת המשחק. הפעולהמחזירה json שיש בו את המידע הזה והודעה בהתאם להצלחה או כשלון.</p>	<pre>def get_all_data()</pre>
<p>פעולה שמקבלת בבקשת HTTP POST מסוג POST</p> <p>gameManagerBeforeStart</p> <p>ובהו json עם</p> <p>שבו יש את המפתח הייחודי של המשחק, את שמות השחקנים ועוד. הפעולה מקבלת את כל המידע הרלוונטי על המשחק אחרי תחילת המשחק. מי שמבצע את הפעולה הוא מי שהתרור לא שלו כדי לקבל את המידע על מה שהיריב עושה. הפעולהמחזירה json שיש בו את המידע הזה והודעה בהתאם להצלחה או כישלון.</p>	<pre>@app.route('/get_real_time', methods=['POST']) def get_real_time()</pre>
<p>פעולה שמקבלת בבקשת HTTP POST מסוג POST</p> <p>gameManagerBeforeStart</p> <p>ובהו json עם</p> <p>שבו יש את המפתח הייחודי של המשחק, את שמות השחקנים ועוד עם המידע הרלוונטי על שחקן שהביא את הבקשה שהשחקן השני צריך לדעת. הפעולה מעלה את המידע הזה לשרת ומקבלת את אותו מידע הרלוונטי על שחקן השני. הפעולהמחזירה json שיש בו את המידע הזה והודעה בהתאם להצלחה או כשלון.</p>	<pre>@app.route('/get_and_upload_player_data', methods=['POST']) def get_and_upload_player_data()</pre>
<p>פעולה שמקבלת בבקשת HTTP POST מסוג POST</p> <p>gameManagerBeforeStart</p> <p>ובהו json עם</p> <p>שבו יש את המפתח הייחודי של המשחק, את שמות השחקנים ועוד. הפעולה מקבלת מידע מעודכן על gameManagerBeforeStart הפעולהמחזירה json שיש בו את המידע הזה והודעה בהתאם להצלחה או כישלון.</p>	<pre>@app.route('/get_game_manager_before_start', methods=['POST']) def get_game_manager_before_start()</pre>
<p>פעולה שמקבלת בבקשת HTTP POST מסוג POST</p> <p>gameManagerBeforeStart</p> <p>ובהו json עם</p>	<pre>@app.route('/upload_game_manager_before_start', methods=['POST']) def upload_game_manager_before_start()</pre>

<p>שבו יש את המפתח הייחודי של המשחק, את שמות השחקנים ועוד. הפעולה מעלה את המידע על gameManagerBeforeStart אל השרת. הפעולה מhzירה json שיש בו הודעה בהתאם להצלחה או כישלון.</p>	
<p>פעולה שמקבלת בקשת HTTP POST מסוג gameManagerBeforeStart שבה json עם שמו יש את המפתח הייחודי של המשחק, את שמות השחקנים ועוד. הפעולה מעדכנת את השרת במידע על אותו שחקן שהעלתה את הבקשה – אם הוא מוכן לעבר למסך המשחק או לא. הפעולה מhzירה json שיש בו הודעה בהתאם להצלחה או כישלון.</p>	<pre>@app.route('/update_ready', methods=['POST']) def update_ready()</pre>
<p>פעולה שמקבלת בקשת HTTP POST מסוג gameManagerBeforeStart שבה json עם שמו יש את המפתח הייחודי של המשחק, את שמות השחקנים ועוד. הפעולה מעדכנת את השרת במידע על אותו שחקן שהעלתה את הבקשה – אם הוא מוכן להתחיל את המשחק או לא. הפעולה מhzירה json שיש בו הודעה בהתאם להצלחה או כישלון.</p>	<pre>@app.route('/update_ready_to_start', methods=['POST']) def update_ready_to_start()</pre>
<p>פעולה שמקבלת בקשת HTTP POST מסוג gameManagerBeforeStart שבה json עם שמו יש את המפתח הייחודי של המשחק, את שמות השחקנים ועוד. הפעולה מוחקת את אותו משחק מהשרת. הפעולה מhzירה json שיש בו הודעה בהתאם להצלחה או כישלון.</p>	<pre>@app.route('/delete_game_manager_before_start', methods=['POST']) def delete_game_manager_before_start()</pre>
<p>פעולה שמקבלת בקשת HTTP POST מסוג gameManagerBeforeStart שבה json עם שמו יש את המפתח הייחודי של המשחק, את שמות השחקנים ועוד. הפעולה מוחקת את אותו שחקן מהמשחק. הפעולה מhzירה json שיש בו הודעה בהתאם להצלחה או כישלון.</p>	<pre>@app.route('/delete_p2', methods=['POST']) def delete_p2()</pre>
<p>פעולה שמקבלת בקשת HTTP GET מסוג gameManagerBeforeStart שבה json עם שמו יש את כל המשחקים שימושיים כרגע. הפעולה מhzירה json שיש</p>	<pre>@app.route('/get_tables', methods=['GET']) def get_tables()</pre>

<p>בו את כל המשחקים עם המפתחות הייחודיים שלהם, יחד עם יוצר המשחק והודעה בהתאם להצלחה או כישלון.</p>	
<p>פעולה שמקבלת בקשת HTTP POST מסוג gameManagerBeforeStart שבה json עם 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.</p>	<pre>@app.route('/end_game', methods=['POST']) def end_game()</pre>
<p>פעולה שמקבלת בקשת HTTP POST מסוג POSTgameManagerBeforeStart שבה json עם המידע על המשתמש השני 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 המשתמש של השחקן בתנאי שהרשמה הצליחה.</p>	<pre>@app.route('/register', methods=['POST']) def register()</pre>
<p>פעולה שמקבלת בקשת HTTP POST מסוג POSTgameManagerBeforeStart שבה json עם המידע על המשתמש השני להתחבר. הפעולה בודקת אם השם המשתמש קיים במערכת או לא ואם כן, בודקת שהסיסמה מתאימה. הפעולה מחזירה json שיש בו הודעה בהתאם להצלחה או כישלון ואת השם המשתמש של השחקן בתנאי שההתחברות הצליחה.</p>	<pre>@app.route('/login', methods=['POST']) def login()</pre>
<p>פעולה שמחזירה את ה reference למקום ב - database שבנו נשמר המידע על המשחקים.</p>	<pre>def get_games_ref()</pre>
<p>פעולה שמחזירה את ה reference למקום ב - database שבנו נשמר המידע על המשתמשים.</p>	<pre>def get_users_ref()</pre>

רפלקציה

הפרויקט היה לי מאד מעניין ותרם לי המון לידע, הן מבחינה תכנית והן מבחינת ניהול פרויקטים גדולים. מכיוון שזו האפליקציה הראשונה שהכנתה, התמודדתי עם קשיים רבים אשר הקשו עלי' בהמשך הפרויקט. נעזרתי המון באינטראנס ובסוף הצלחתி למצוא תשובות ופתרונות לביעות שהוא לי. אני חשב שהקשיים האלה יעזרו לי בעתיד להכין פרויקטים טובים יותר עם פחות קשיים ממה שהוא לי בפרויקט זהה. למורת זאת, לדעת ניהلت 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 ידעת כיצד להתחיל לכתוב את הקוד, אבל מרגע שהתחלתי לעבוד ולתיכנת, נהיה לי קל מאוד ונهاית מודע מכל מה שעשית, ואפילו ישבתי בלילה לתכנן מקום לשון לפעמים.

הרגשתי לעיתים במהלך הכנות הפרויקט שלקחתי על עצמי אוֹל' משימה גדולה מדי ולפעמים רציתי להקל על עצמי, אבל בכל מה שיכלתי עשית את המיטב והשתדלתי לעשות הכל בצוורה שתיראה וכי טוב ותהיה וכי נוחה ויעילה.

באربעה – חמישה חודשים האחרונים השקעת המון זמן בהכנות הפרויקט ואני מרגיש כעת סיפוק רב שהצלחת להגיע לנצח מרשימה בעיני שני גאה בה ושמח לעמוד מאחוריה, זאת אחרי קשיים ומאמצים רבים שהובילו אותו לסיום.

למרות זאת, אני מתכוון להמשיך ולשפר את הפרויקט שלי על מנת בסופו של דבר לפנות לחברת Dominion כדי לנסות לעשות אתכם شيئا' פועל או לקבל מהם את האפשרות להעלות את האפליקציה לחנות האפליקציות תמורה כסף, כי לדעת האפליקציה יכולה להיות שימושה מאוד עבור אהובי המשחק דומיניון ויש לא מעט כאלה.

קוד מלך

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 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.AlertDialog;
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 (!isValidEmail()) {  
    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 isValidEmail()
{
    String emailRegex = "[a-zA-Z0-9_.+&*-]+(?:\\.|[a-zA-Z0-9_.+&*-]*@[a-zA-Z0-9_.+&*-]+(?:\\.|[a-zA-Z0-9_-]+\\.)+[a-zA-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("gameManagerBefore  
Start");  
  
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);
cITurn = findViewById(R.id.cITurn);

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().height
Pixels*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("gameManagerBefore
Start");
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 resourcev;
}

```

```

// 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()).handleB
uttonClicks(((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.setOnClickListener(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);

    }

}

/** 
 * 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()).conc
```

```

at(" 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).getShortImage
Source()));

    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().getCardsForActionCardPla
y()) > 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().getCardsForActionCardPla
y()) <= game.getTurn().getWaitForFunction().getMaxAmount()

|| game.getTurn().getWaitForFunction().getMaxAmount() == -1)

&&

Help.sizeOfHash(game.getTurn().getWaitForFunction().getCardsForActionCardPla
y()) >= 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;
```



```

*/
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().getCardNa

```

```
me())  
    .isCardToUse(resourcelv.getContentDescription().toString(),  
gameActivity.game, gameActivity)  
    ||  
Help.sizeOfHash(gameActivity.game.getTurn().getWaitForFunction().getCardsForA  
ctionCardPlay()) ==  
gameActivity.game.getTurn().getWaitForFunction().getMaxAmount())  
return;  
  
if  
(gameActivity.game.getTurn().getWaitForFunction().isHandleClickOnCard()  
&&  
!Help.nameToCard(gameActivity.game.getTurn().getWaitForFunction().getCardNa  
me()).isMarkCardSelectedFromHandWhenHandle()) {  
  
gameActivity.game.getTurn().getWaitForFunction().insertCardSelectedInHand(reso  
urcelv.getContentDescription().toString());  
  
Help.nameToCard(gameActivity.game.getTurn().getWaitForFunction().getCardNam  
e())  
.handleClickOnHandOrDialog(resourcelv.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(
resourceLv.getContentDescription().toString());

if (countSelected == count)
return;

if
(gameActivity.game.getTurn().getWaitForFunction().getTypeOfAction().equals("tras
h"))
    view.findViewById(R.id.ivRedX).setVisibility(View.VISIBLE);
else
    view.findViewById(R.id.ivRedX).setVisibility(View.INVISIBLE);

if
(!gameActivity.game.getTurn().getWaitForFunction().getTypeOfAction().equals("tra
sh"))
    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(reso
urcelv.getContentDescription().toString());

if

(gameActivity.game.getTurn().getWaitForFunction().isHandleClickOnCard()

&&

Help.nameToCard(gameActivity.game.getTurn().getWaitForFunction().getCardNam
e()).isMarkCardSelectedFromHandWhenHandle()) {

Help.nameToCard(gameActivity.game.getTurn().getWaitForFunction().getCardNam
e())

.handleClickOnHandOrDialog(resourcelv.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().getCardNam
e()))

    .isCardToUse(this.objects.get(position).first, gameActivity.game,
gameActivity)) {

    if

(gameActivity.game.getTurn().getWaitForFunction().getTypeOfAction().equals("tras
h")) {

        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.getTur

n().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("tra
sh"))
    &&
!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 resourcelv = view.findViewById(R.id.ivAction);

        iv.setImageResource(Help.nameToCard(resourcelv.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().getId
entifier(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 resourcelv = view.findViewById(R.id.ivAction);

    iv.setImageResource(Help.nameToCard(resourcelv.getContentDescription().toString()
g()).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 resourcelv = view.findViewById(R.id.ivAction);

    if

(!Help.nameToCard(gameActivity.game.getTurn().getWaitForFunction().getCardName())

```

```
.isCardToUse(resourcelv.getContentDescription().toString(),  
gameActivity.game, gameActivity)  
  
||  
  
Help.sizeOfHash(gameActivity.game.getTurn().getWaitForFunction().getCardsForA  
ctionCardPlay()) ==  
  
gameActivity.game.getTurn().getWaitForFunction().getMaxAmount()  
  
return;  
  
  
  
if  
  
(gameActivity.game.getTurn().getWaitForFunction().isHandleClickOnCard()  
  
&&  
  
!Help.nameToCard(gameActivity.game.getTurn().getWaitForFunction().getCardNa  
me()).isMarkCardSelectedFromHandWhenHandle() {  
  
  
  
gameActivity.game.getTurn().getWaitForFunction().updateCardsForDialogByPositi  
on(getAdapterPosition(), true);  
  
  
  
Help.nameToCard(gameActivity.game.getTurn().getWaitForFunction().getCardNam  
e())  
  
  
  
.handleClickOnHandOrDialog(resourcelv.getContentDescription().toString(),  
gameActivity.game, gameActivity);  
  
return;  
}
```

```

    if

(gameActivity.game.getTurn().getWaitForFunction().getTypeOfAction().equals("tras
h"))

    view.findViewById(R.id.ivRedX).setVisibility(View.VISIBLE);

    if

(!gameActivity.game.getTurn().getWaitForFunction().getTypeOfAction().equals("tra
sh"))

    &&

!gameActivity.game.getTurn().getWaitForFunction().getTypeOfAction().equals("ord
er"))

    view.findViewById(R.id.ivYellowX).setVisibility(View.VISIBLE);

    if

(gameActivity.game.getTurn().getWaitForFunction().getTypeOfAction().equals("ord
er")) {

    if

(gameActivity.game.getTurn().getWaitForFunction().getCardsForDialog().get(getAd
apterPosition()).second) {

        view.findViewById(R.id.ivYellowMargin).setVisibility(View.INVISIBLE);

gameActivity.game.getTurn().getWaitForFunction().updateCardsForDialogByPositi
on(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)
```

```

e()).isMarkCardSelectedFromHandWhenHandle()) {

    Help.nameToCard(gameActivity.game.getTurn().getWaitForFunction().getCardName()
        e())

    .handleClickOnHandOrDialog(resourceId.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.setBackground().setAlpha(150);

if
(!gameActivity.game.getTurn().getWaitForFunction().isWaitingForActionCardsDialo
g()
```

```

||

!Help.nameToCard(gameActivity.game.getTurn().getWaitForFunction().getCardNa
me())

    .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("tras
h")) {

    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("ord
er"))

    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("tra
sh"))
    &&
!gameActivity.game.getTurn().getWaitForFunction().getTypeOfAction().equals("use
")) {
    if
(!gameActivity.game.getTurn().getWaitForFunction().getTypeOfAction().equals("ord
er"))
        || this.objects.get(position).second)
    holder.ivYellowMargin.setVisibility(View.VISIBLE);
else
    holder.ivYellowMargin.setVisibility(View.INVISIBLE);

if
(!gameActivity.game.getTurn().getWaitForFunction().getTypeOfAction().equals("ord
er"))
    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.getTreasureCardsPlaye
d()));

    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(cardN
ame));

        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(cardN
ame));

        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);
}

```

```

        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 autoComplete) {
    this.cardsForWaitForPlay.add(new PlayCardArguments(cardName, n, i,
forceUse, autoComplete));
}

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 gameId A String with the table id
 * @param nonCreatorId A String with the id of the non creator
 */
public GameManagerBeforeStart(String gameId, String nonCreatorId) {
    this.gameId = gameId;
    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"));
                }
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
    }
}

```

```

g("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(jsonObj
ctRequest);

}

/***
 * 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().gameManagerBeforeStartToJ
son(),

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 uploadAndGetData(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()).addToRequestQue
ue(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);
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

```

    }

    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()).addToRequestQue
```

```

        ue(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()).enemyPi
ay(gameManager, gameManager.getGameActivity());

            if (gameManager.getPlayer().containsCard("Moat"))

                Help.nameToCard("Moat").reaction(gameManager,
gameManager.getGameActivity());

            else {

Help.nameToCard(gameManager.getTurn().getLastActionCardForWait()).attack(ga
meManager, 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(jsonObject);
```

```

tRequest);

}

/** 
 * 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().gameManagerBeforeStartToJ
son(),
    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(jsonObje
ctRequest);

}

/***
 * 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().gameManagerBeforeStartToJ
son(),
    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(jsonObject);

}

/** 
 * 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.getRequestQueue(prepareGameActivity).addRequestQueue(jsonObje
ctRequest);

}

/**



 * 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(j
onObject, 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()).addToRequestQue
ue(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().gameManagerBeforeStartToJ
son(),
    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().jsonToGameManagerBeforeS
tart(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().gameManagerBeforeStartToJ
son(),

    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(jsonObje
ctRequest);

}

/**



 * 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().gameManagerBeforeStartToJ
son(),


    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(jsonObj
ctRequest);

}

/**
 * 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",
    null, 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
         */
        @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.getRequestAdapter().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(jsonObjectReq
uest);
}

/**
 * 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(jsonObjectReques

```

```
t);
```

```
}
```

```
}
```

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 this 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(s
2).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 explicitly 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.
```

```
*/\n\npackage com.example.dominion_game.classes;\n\n\nimport com.example.dominion_game.activities.GameActivity;\n\n\n\npublic abstract class Card {\n\n    private String type;\n\n    private String name;\n\n    private int price;\n\n    private int imageSource;\n\n    private int shortImageSource;\n\n\n    /**\n     * The constructor\n     *\n     * @param name A String with the name of the card\n     *\n     * @param price An Integer with the price of the card\n     *\n     * @param imageSource An Integer with the image resource of the card\n     *\n     * @param shortImageSource An Integer with the short image resource of the\n     * card\n     *\n     * @param type A String with the type of the card (action, treasure or victory)\n     */\n\n    public Card(String name, int price, int imageSource, int shortImageSource, String\n        type) {\n\n        this.name = name;\n\n        this.price = price;\n\n    }\n}
```

```
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 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.get
Name(), 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().getWaitForF
unction().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;
}
}

```

```
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);
    }
}

```

```
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.get
tPlayer().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.arrayListPairsToArrayList(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().getWaitForFu
nction().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(card
Name).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.uploadActionButton(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().getWaitForFu
nction().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(card
Name).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.get
Name(), 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;

}

}
```

```
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="Is Rated"
    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/lGamePanel"
        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"
    </include>
</androidx.constraintlayout.widget.ConstraintLayout>
```

```
    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/cIBasicCards"
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/cIPlayArea"
```

```
    android:layout_width="0dp"
    android:layout_height="match_parent"
    android:layout_weight="55"
    android:orientation="vertical"
    android:visibility="visible">

<RelativeLayout
    android:id="@+id/rButtonsInPlay"
    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/rIButtonsInPlay"
    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/rButtonsInPlay"
    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/rButtonsInPlay"
    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/lIMessages"
    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/lIMessages"
    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 {"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": 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": 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-
        052b5aeeeaa.json"),
        {'databaseURL': 'https://dominion-project.firebaseio.com/'})
    app.run(host='0.0.0.0', port=8888)
```