"بسم الله الرحمن الرحیم"

مریم اجل لوئیان

۹۹۲۲۲۰۰۱

گزارشی از پروژه پایانی

پروژه دوم

بازی معمایی سرنخ

توضیحات زیر کد اولیه ای است که من زدم. اما پس از اجرا، متعدد خطا گرفتم. خطاهایی از این دست که مثلا در متد setupGame در قسمت int numPlayers = players.size(); تقسیم بر صفر اتفاق می‌افتد. یا کلا اجرا نمیشد و یا تابع ها تعریف نشده بودند.

نکاتی که با رنگ قرمز اضافه میکنم چالش هایی هست که با تغییر کد آنها را حل کردم.

نکاتی که با رنگ سبز نوشته میشود در ادامه نکات قرمز است اما بعد از ساعت ٦ صبح نوشته شده است.

کلاس ClueGame:

در ابتدا ما یک کلاس ClueGame را تعریف می‌کنیم برای آنکه متغیر ها و داده ها را در آن ذخیره کنیم.

این کلاس،کلاس اصلی بازی است و منطق و اجرای بازی را مدیریت خواهد کرد. یعنی در این کلاس قرار است تنظیم بازیکنان، شخصیت‌ها، مکان‌ها و اتاق‌ها و همچنین تعریف راه‌حل و مدیریت حلقه اصلی بازی صورت بگیرد.

این کلاس شامل متد اصلی main است که نقطه شروع برنامه است.

```
public class ClueGame {
  public static void main(String[] args) {
```

```
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of players (between 3 and 6): ");
        int numPlayers = scanner.nextInt();
        scanner.nextLine();
        Game game = new Game();
        for (int i = 0; i < numPlayers; i++) {
            System.out.print("Enter name for player " + (i + 1) + ": ");
            String playerName = scanner.nextLine();
            game.addPlayer(new Player(playerName));
        }
        game.start();
    }
}
```

برای شی از کلاس Game استفاده می‌کنیم و متد start را فرای شروع فراخوانی می‌کنیم.

```
for (int i = 0; i < numPlayers; i++) {
        System.out.print("Enter name for player " + (i + 1) + ": ");
        String playerName = scanner.nextLine();
        game.addPlayer(new Player(playerName));
```

حلقه ای را شروع می کنیم که تعداد دفعات 'numPlayers' را برای هر بازیکن یک بار اجرا می کند.

از کاربر می خواهیم نام player فعلی را وارد کند. «i + 1» برای نمایش اعداد بازیکنان از ۱ استفاده میشود.

نام player را به عنوان یک رشته از ورودی می خواند.

یک شیء جدید "Player" با نام وارد شده ایجاد می کند و آن را به بازی اضافه می کند.

کلاس Game:

این کلاس مدیریت کل بازی را انجام خواهد داد. در این کلاس خواهیم داشت:

List<Player> players: لیست بازیکنان

List<String> characters: لیست شخصیت ها مثل Emma, Liam, Jack...

List<String> locations: لیست مکان ها مثل زیر گلدان، کشوی مخفی، پشت عکس و...

List<String> rooms: لیست اتاق ها مثل گلخانه، اتاق بیلیارد، اتاق مطالعه، پذیرایی و...

Map<String, String> solution: نقشه راه حل برای ذخیره راه حل صحیح بازی استفاده می شود. "دزد"، "مکان" و "اتاق" این ترکیبی است که بازیکنان سعی در حدس زدن آن دارند.

سپس مقادیر لازم به لیست ها داده می‌شود.

players به عنوان یک لیست خالی.

characters با یک لیست پیش‌تعریف شده از نام‌های شخصیت.

locations با یک لیست پیش‌تعریف شده از مکان‌های ممکن.

rooms با یک لیست پیش‌تعریف شده از نام‌های اتاق.

solution به عنوان یک نقشه خالی برای بعداً ذخیره حل بازی.

gameWon به عنوان false، نشان دادن اینکه بازی در آغاز برنده نشده است.

در قسمت solution ما یک Hash Map خالی برای ذخیره داده‌های کاربر تعریف می‌کنیم که می‌تواند جفت‌های کلید و مقدار را در جایی که هم کلیدها و هم مقادیر از نوع «String» هستند ذخیره کند.

زمانی که بازیکنان حدس می‌زنند، نقشه solution برای بررسی درستی حدس‌هایشان استفاده می‌شود.

متغیر gameWon وضعیت برد بازی را نگه می‌دارد.

این بخش اضافه شد:

```
public void addPlayer(Player player) {
    players.add(player);
}
```

با گذاشتن این بخش، میتوان یک بازیکن به بازی اضافه کرد. در واقع بازیکن ارائه شده را به لیست بازیکنان اضافه می‌کند.

پارامتر: Player player – بازیکن اضافه شده به بازی.

متد start بازی را با فراخوانی متدهای setupGame و playGame شروع می‌کند.

متد setupGame کارت‌های راه‌حل را به صورت تصادفی انتخاب کرده و بین بازیکنان باقی‌مانده پخش می‌کند.

متد playGame حلقه اصلی بازی را مدیریت می‌کند.

کلاس Player:

این کلاس نماینده هر بازیکن در بازی است.

name: نام بازیکن را ذخیره می‌کند.

cards: لیستی از کارت‌هایی که بازیکن در دست دارد.

currentRoom: اتاق فعلی که بازیکن در آن قرار دارد.

characters: لیست ثابت از شخصیت‌های بازی.

locations: لیست ثابت از مکان‌های بازی.

rooms: لیست ثابت از اتاق‌های بازی.

در اینجا نام بازیکن را به عنوان ورودی گرفته میشود و متغیرهای cards و currentRoom را مقداردهی اولیه می‌کند.

لیست‌های ثابت characters, locations, و rooms نیز در اینجا مقداردهی می‌شوند.

متد getName نام بازیکن را برمی‌گرداند.

متد getCurrentRoom اتاق فعلی بازیکن را برمی‌گرداند.

متد setCurrentRoom اتاق فعلی بازیکن را تنظیم می‌کند.

متد addCard یک کارت به دست بازیکن اضافه می‌کند.

متد hasCard بررسی می‌کند که آیا بازیکن یک کارت خاص را دارد یا خیر.

متد randomGuess یک حدس تصادفی برای بازیکن ایجاد می‌کند.

از Random برای انتخاب تصادفی یک شخصیت و یک مکان از لیست‌های characters و locations استفاده می‌کند.

نتیجه حدس به صورت یک رشته شامل شخصیت و مکان به همراه هم بازگردانده می‌شود.

پیاده‌سازی منطق بازی

متد setupGame:

این متد کارت‌های راه‌حل را به صورت تصادفی انتخاب کرده و بین بازیکنان باقی‌مانده پخش می‌کند.

وقتی قرار است کارت ها بر بخورند، باید ابتدا بر بزنیم بعد کارت اول را که به نفر اول دادیم آن کارت را از دسته کارت ها حذف کنیم، سپس دوباره برای فرد جدید کارت ها را بر بزنیم و کارتی که برای آن فرد کشیدیم را از دسته کارت ها حذف کنیم و دوباره این کار تا نفر آخر تکرار شود.

Random rand = new Random():ابتدا یک شیء از کلاس Random ایجاد می‌شود تا بتوانیم از آن برای انتخاب تصادفی عناصر استفاده کنیم.

از متد random.nextInt برای انتخاب تصادفی یک کارت از هر دسته (شخصیت، مکان و اتاق) استفاده میکنم.

این کارت‌ها به عنوان راه‌حل نهایی بازی در متغیر solution ذخیره می‌شوند.

```
characters.remove(solutionCharacter);
locations.remove(solutionLocation);
rooms.remove(solutionRoom);
```

کارت‌های انتخاب شده به عنوان راه‌حل از لیست‌های شخصیت‌ها، مکان‌ها و اتاق‌ها حذف می‌شوند تا در مراحل بعدی پخش کارت‌ها در دست بازیکنان قرار نگیرند.

```
List < String > remainingCards = new ArrayList <> ();
remainingCards.addAll(characters);
remainingCards.addAll(locations);
remainingCards.addAll(rooms);
Collections.shuffle(remainingCards);
```

یک لیست جدید به نام remainingCards ایجاد می‌شود و تمام کارت‌های باقی‌مانده (شخصیت‌ها، مکان‌ها و اتاق‌ها) به آن اضافه می‌شوند.

سپس کارت‌های این لیست با استفاده از متد Collections.shuffle به صورت تصادفی ترکیب می‌شوند.

```
int numPlayers = players.size();
int cardsPerPlayer = remainingCards.size() / numPlayers;
```

این بخش اضافه شد.

```
int extraCards = remainingCards.size() % numPlayers;
```

هر کارت اضافی که نمی‌توان به صورت یکسان توزیع شود، شمرده می‌شود.

```
for (int i = 0; i < numPlayers; i++) {
  for (int j = 0; j < cardsPerPlayer; j++) {
    players.get(i).addCard(remainingCards.remove(0));
  }
}
```

این بخش اضافه شد.

```
if (extraCards > 0) {
        players.get(i).addCard(remainingCards.remove(0));
        extraCards--;
      }
```

ابتدا تعداد بازیکنان (numPlayers) و تعداد کارت‌های هر بازیکن (cardsPerPlayer) محاسبه می‌شود.

سپس کارت‌ها به ترتیب بین بازیکنان توزیع می‌شوند.

از متد remove برای برداشتن کارت‌ها از لیست remainingCards و افزودن آن‌ها به دست هر بازیکن استفاده می‌شود.

اگر کارت‌های اضافی وجود داشته باشد، به ترتیب به بازیکنان داده می‌شوند تا هروقت که کارت اضافی باقی نماند.

متد playGame:

random: برای تولید اعداد تصادفی استفاده می‌شود.

currentPlayerIndex: اندیس بازیکن فعلی در لیست players.

حلقه اصلی بازی:

```
while (!gameWon) {
```

این حلقه تا زمانی که متغیر gameWon برابر false باشد، ادامه می‌یابد. این متغیر نشان می‌دهد که آیا بازی برنده‌ای دارد یا نه.

```
Player currentPlayer = players.get(currentPlayerIndex);
```

بازیکن فعلی از لیست players با استفاده از اندیس currentPlayerIndex انتخاب می‌شود.

```
int diceRoll = rollDice();
```

متد rollDice() برای انداختن تاس و تولید عددی بین ۱ تا ٦ استفاده می‌شود.

```
String newRoom = "";
```

این بخش تغییر کرد:

```
List<String> possibleRooms = new ArrayList<>();
    if (diceRoll % 2 == 0) {
      // Move to the even rooms
      for (int i = 1; i <= rooms.size(); i += 2) {
        possibleRooms.add(rooms.get(i - 1));
      }
    } else {
      // Move to the odd rooms
      for (int i = 2; i <= rooms.size(); i += 2) {
        possibleRooms.add(rooms.get(i - 1));
      }
    }
```

```
newRoom
    = possibleRooms.get(random.nextInt(possibleRooms.size()));
currentPlayer.setCurrentRoom(newRoom);
```

یک لیست از اتاق‌های ممکن که بازیکن می‌تواند به آن حرکت کند، ایجاد می‌شود.

اگر عدد حاصل از پرتاب تاس زوج باشد، بازیکن می‌تواند به اتاق‌های زوج حرکت کند.

اگر عدد حاصل از پرتاب تاس فرد باشد، بازیکن می‌تواند به اتاق‌های فرد حرکت کند.

یک اتاق تصادفی از لیست اتاق‌های ممکن به عنوان اتاق جدید انتخاب می‌شود.

اتاق فعلی بازیکن به اتاق جدید انتخاب شده بروزرسانی می‌شود.

```
if (diceRoll % 2 == 0) {
// Move to the couple's rooms
newRoom = rooms.get((diceRoll / 2 - 1) * 2);
} else {
//Move to individual rooms
newRoom = rooms.get((diceRoll - 1) * 2);
}
currentPlayer.setCurrentRoom(newRoom);
```

newRoom: اتاق جدیدی که بازیکن به آن حرکت می‌کند.

اگر عدد تاس زوج باشد (diceRoll % 2 == 0)، بازیکن به اتاق‌های زوج حرکت می‌کند.

اگر عدد تاس فرد باشد، بازیکن به اتاق‌های فرد حرکت می‌کند.

اتاق جدید محاسبه شده و با استفاده از متد setCurrentRoom به بازیکن اختصاص می‌یابد.

```
String guess = currentPlayer.randomGuess();
    System.out.println(currentPlayer.getName() + " guesses: "
        + guess);
```

بازیکن فعلی یک حدس تصادفی با استفاده از متد randomGuess() می‌زند.

حدس بازیکن به همراه نام او در کنسول چاپ می‌شود.

```
boolean guessCorrect = checkGuess(currentPlayer, guess);
    if (guessCorrect) {
      System.out.println(currentPlayer.getName()
        + " won the game! ");
      gameWon = true;
    } else {
      System.out.println(wrong guess.);
      revealCardFromOtherPlayer(currentPlayer, guess);
    }
```

متد checkGuess حدس بازیکن را بررسی می‌کند و نتیجه را به صورت بولین (درست یا نادرست) برمی‌گرداند.

اگر حدس درست باشد (guessCorrect برابر true)، نام بازیکن به عنوان برنده اعلام شده و gameWon برابر true قرار می‌گیرد تا حلقه بازی پایان یابد.

اگر حدس نادرست باشد، پیامی مبنی بر نادرست بودن حدس در کنسول چاپ می‌شود. و یک کارت از یک بازیکن دیگر فاش می‌شود.

```
currentPlayerIndex = (currentPlayerIndex + 1) % players.size();
```

اندیس بازیکن فعلی به بازیکن بعدی تغییر می‌کند.

با استفاده از عملگر modulus (٪) تضمین می‌شود که اندیس بازیکن بعدی در محدوده لیست players باقی بماند و پس از رسیدن به انتهای لیست دوباره به اول لیست بازگردد.

متد rollDice:

Random random = new Random(); یک شیء از کلاس Random ایجاد می‌کند. کلاس Random از کتابخانه java.util برای تولید اعداد تصادفی استفاده می‌شود.

$random.nextInt(6) + 1$ این خط کد عددی تصادفی بین ۰ و ۵ تولید می‌کند و سپس ۱ به آن اضافه می‌کند تا عددی بین ۱ و ٦ بدست آید.

متد makeGuess:

return player.randomGuess(); : این خط متد randomGuess را از شیء player فراخوانی می‌کند و نتیجه (یک حدس تصادفی) را برمی‌گرداند.

این متد حذف شد.

متد checkGuess:

String[] parts = guess.split(","); این خط حدس بازیکن را به دو بخش تقسیم می‌کند؛ یک بخش برای شخصیت و یک بخش برای مکان. این کار با استفاده از متد split و جداکننده ", " انجام می‌شود.

parts: یک آرایه از رشته‌ها که شامل اجزای جداگانه حدس (شخصیت و مکان) است.

این خطوط بخش اول حدس جدا شده را به guessedCharacter و بخش دوم را به guessedLocation اختصاص می‌دهد.

parts[0] : شخصیت حدس زده شده توسط بازیکن.

parts[1] : مکان حدس زده شده توسط بازیکن.

String guessedCharacter = parts[0]; این خط اولین بخش از آرایه parts را که نشان‌دهنده شخصیت حدس زده شده است، استخراج می‌کند.

String guessedLocation = parts[1]; : این خط دومین بخش از آرایه parts را که نشان‌دهنده مکان حدس زده شده است، استخراج می‌کند.

return guessedCharacter.equals(solution.get("character")) && guessedLocation.equals(solution.get("location")); : این خط شخصیت و مکان حدس زده شده را با شخصیت و مکان موجود در راه‌حل مقایسه می‌کند. اگر هر دو درست باشند، مقدار true برمی‌گرداند؛ در غیر این صورت، مقدار false برمی‌گرداند.

متد revealCard:

این متد دو پارامتر ورودی دارد:

player: بازیکنی که باید بررسی شود.

card: کارتی که باید بررسی شود آیا در دست بازیکن هست یا نه.

متد hasCard از کلاس Player استفاده می‌شود تا بررسی کند که آیا بازیکن کارت مشخص شده را دارد یا خیر.

این متد یک مقدار بولی (boolean) برمی‌گرداند که نشان‌دهنده این است که آیا بازیکن کارت را دارد یا نه.

اگر بازیکن کارت را داشته باشد، نام بازیکن و پیام "It has." " چاپ می‌شود.

اگر بازیکن کارت را نداشته باشد، نام بازیکن و پیام "It does not have". " چاپ می‌شود.

این بخش از کد تقریبا کامل عوض شده است:

متد revealCardFromOtherPlayer:

```
for (Player player : players) {
    if (player != currentPlayer) {
```

for (Player player : players): از طریق هر بازیکن در لیست players حرکت می‌کند.

$if\ (player\ !=\ currentPlayer)$: بازیکن فعلی (currentPlayer) را نادیده می‌گیرد زیرا ما به دنبال بازیکن‌های دیگر هستیم.

```
String[] guessParts = guess.split(",");
for (String part : guessParts) {
    if (player.hasCard(part)) {
        System.out.println(player.getName() + " has the card: " + part);
        return;
    }
}
```

$String[]\ guessParts\ =\ guess.split(",");$: رشته حدس را به بخش‌های مختلف تقسیم می‌کند.

$for\ (String\ part\ :\ guessParts)$: از طریق هر بخش از حدس حرکت می‌کند.

if (player.hasCard(part)) : بررسی می‌کند که آیا بازیکن فعلی (player) دارای یک کارت است که با بخش فعلی تطابق دارد یا خیر.

بقیه تغییرات و توضیحج چالش ها به علت کمبود وقت نشد که کامل بنویسم

فایل ویرایش شده در سات گذاشته خواهد شد