

"بسم الله الرحمن الرحيم"

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 مثلاً در متد `setupGame` در قسمت `int numPlayers = players.size()`؛ تقسیم بر صفر اتفاق می افتد. یا کلاً 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.

کلاس `ClueGame`:

در ابتدا ما یک کلاس `ClueGame` را تعریف می کنیم 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، شخصیت ها، مکان ها و اتاق ها و همچنین تعریف راه حل و مدیریت حلقه اصلی بازی صورت بگیرد.

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

برای شی از کلاس `Game` استفاده می کنیم و متد `start` را فرا می خوانیم.

```
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 بیلارد، 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 می شود.

در قسمت `solution` ما یک Hash Map خالی برای ذخیره داده های کاربر تعریف می کنیم که می تواند

جفت های کلید و مقدار را در جایی که هم کلیدها و هم مقادیر از نوع «String» هستند ذخیره کند.

زمانی که بازیکنان حدس می زنند، نقشه `solution` برای بررسی درستی حدس هایشان استفاده می شود.

متغیر `gameWon` وضعیت برد بازی را نگه می دارد.

این بخش اضافه شد:

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

متد **start** بازی را با فراخوانی متدهای **setupGame** و **playGame** شروع می کند.

متد **setupGame** کارت های راه حل را به صورت تصادفی انتخاب کرده و بین بازیکنان باقی مانده پخش می کند.

متد **playGame** حلقه اصلی بازی را مدیریت می کند.

کلاس **Player**:

این کلاس نماینده هر بازیکن در بازی است.

name: نام بازیکن را ذخیره می کند.

cards: لیستی از کارت هایی که بازیکن در دست دارد.

currentRoom: اتاق فعلی که بازیکن در آن قرار دارد.

characters: لیست ثابت از شخصیت های بازی.

locations: لیست ثابت از مکان های بازی.

rooms: لیست ثابت از اتاق های بازی.

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

لیست های ثابت **characters**, **locations** و **rooms** نیز در اینجا مقداردهی می شوند.

متد `getName` نام بازیکن را برمی گرداند.

متد `getCurrentRoom` اتاق فعلی بازیکن را برمی گرداند.

متد `setCurrentRoom` اتاق فعلی بازیکن را تنظیم می کند.

متد `addCard` یک کارت به دست بازیکن اضافه می کند.

متد `hasCard` بررسی می کند که آیا بازیکن یک کارت خاص را دارد یا خیر.

متد `randomGuess` یک حدس تصادفی برای بازیکن ایجاد می کند.

از `Random` برای انتخاب تصادفی یک شخصیت و یک مکان از لیست های `characters` و `locations` استفاده می کند.

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

پیاده سازی منطق بازی

متد `setupGame`:

این متد کارت های راه حل را به صورت تصادفی انتخاب کرده و بین بازیکنان باقی مانده پخش می کند.

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 می شود 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 می کنیم.

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;
```

```
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.

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 ۶ استفاده می‌شود.

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

```
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);  
  
    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 ($\text{diceRoll} \% 2 == 0$)، 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. ");
```

```
}
```

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.

```
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();
```

`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 ۶ 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.

`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 می شود.

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

فایل ویرایش شده در سات گذاشته خواهد شد