

Danmarks  
Tekniske  
Universitet



02314 | 62531 | 62532

Introductory Programming | Development methods for IT-systems | Version control and test methods

---

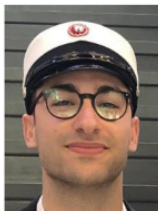
## CDIO 2 Group 7

---

Friday 29<sup>th</sup> September, 2023 18:00

### AUTHORS

Mustafa Baker - s235092



Adel Wisam - s236076



Elias Larsen - s235125



Mads Hartwich - s235102



Emil Savino - s235096



Frederik Bastrup - s235117

# 1 Hourly Accounting

Very early on we realized, that it would not be possible for everyone to contribute equally to all parts of the project. Some have therefore spent more time programming, while others have made more diagrams or contributed more to the report. We have all, however, spent roughly 6 hours a week each, which totals to around 108 hours.

## Contents

<b>1</b>	<b>Hourly Accounting</b>	<b>1</b>
<b>2</b>	<b>Summary</b>	<b>3</b>
<b>3</b>	<b>Introduction</b>	<b>3</b>
<b>4</b>	<b>Version control</b>	<b>3</b>
<b>5</b>	<b>Requirement/Analysis</b>	<b>5</b>
5.1	Diagrams/Models . . . . .	7
<b>6</b>	<b>Implementation</b>	<b>10</b>
6.1	Testing . . . . .	11
<b>7</b>	<b>Conclusion</b>	<b>11</b>
<b>8</b>	<b>Appendix</b>	<b>12</b>
8.1	Litterature . . . . .	12
8.2	Code Upload . . . . .	12

## 2 Summary

Using Java code, Git, and GitHub, we have developed a two-player adventure game. The report includes everything from the project. This includes all the thoughts and processes the group has had during the development of the program, all the requirements for the program, both the ones the group have made for the program itself and the ones which was given from the costumer, and how they are met. There are also all the diagrams/models that have been used in the development process of the program. It also includes how the group-work process has been, how we tested the program, the code used for the program, and the repository on GitHub, that the group has used for the program development. We have also chosen specific parts of the code and described it.<sup>[1]</sup>

## 3 Introduction

Introducing our Java project, which is a adventure board game. The game is made for a customer who described a program they wanted made and the requirements for that specific program. The group have also made some requirements for the game itself. The board game is a two-player game, where you play 1 against 1 on the same computer. The players will roll two dice at once, one at a time.<sup>[1]</sup> Depending on the player's dice roll result, they will either get or lose gold. Almost as the last program we programmed for the costumer, just with some extra things to it. The player also has the chance to get an extra dice roll if the player rolls 10 with the two dice. Each player starts with 1.000 gold and the game ends when a player has 3.000 or more gold.

## 4 Version control

We have created a public repository on GitHub<sup>[2]</sup>, where the members of the group have access to see and edit everything in the repository. The customer also have access. to the repository, but can only see the code and information in it, they can't edit. The repository contains two branches: the main branch and the development branch. The development branch is used to pull and push the versions where the code is still under development, and the main branch is used to push and have the code that is either finished or works, but still needs some work to it. In the repository, you can also see who, what, and when each group member has committed something to the repository. This was also one of the requirements from the customer. They wanted to be able to see who, what, and when someone from the group had committed something to the repository, and the latest version of the code that works. Just like the dice game program the group also have made for the same costumer. In the repository, there is a README- file, which contains information about the program for both the customer and the user. For the customer, there is information about the program and how it works and for the user, there is information about how to start the program. The user will not get any instructions or information about how the program works, since this also was one of the requirements from the customer last time and we have therefore also

seen this as a requirement this time as well. The program has to be playable without having to give them any instructions.<sup>[1]</sup>

## 5 Requirement/Analysis

### Requirement specification

- The program has to work on the "databarens" computers.
- The program has to be playable without any given instructions before playing.
- The result of the dice roll should be shown in under 1/3 of a second.
- The program has to be playable for two players at the same time.
- The dice in the program have to be 6-sided dice with the numbers 1-6.
- The game is done as soon as one of the two players gets 3.000 gold.
- The player will get an extra turn if they roll 10 with the two dice.
- The program has to pass a test, where the result of 1000 dice rolls must only have a deviation of 10.

### Usecases

Usecases are used to describe how the users will perform specific tasks in the program. This is therefore from the user's point of view. This also describes the behavior of the program, when it has to respond to the user's requests doing the usecases.

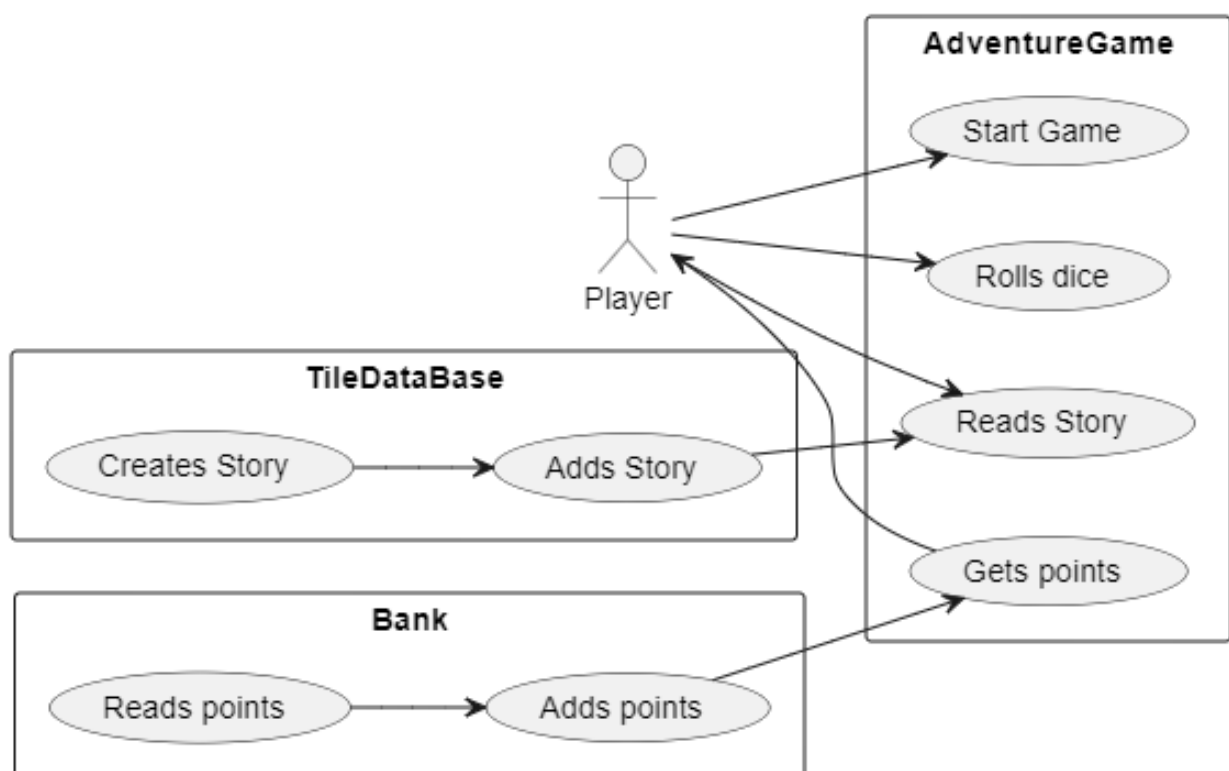


Figure 1: Use Case Diagram

## Usecases

**Usecase 1 - Start a game** The first usecase is "start a game". Here the user has to start a game, which means they have to open the board game program and then press the "enter bottom" on their keyboard to start the game when the program is open. The user will also get a message in the program, which tells them they have to press the "enter button" to start the game.

**Usecase 2 - Roll the dice and play a turn** The second usecase is "Roll the dice and play a turn". After the user has completed the first usecase, the board game program is opened and a game has been started. Now they will have to use their turn by rolling the dice and see the outcome of the dice roll.

**Usecase 3 - Play a game** The third usecase is "Play a game". Now that the user has opened the board game program, started a game, and played a turn. They will now have to play and finish a game. They will just have to roll the dice and see the outcome each time it's their turn. Until one of the two players won the game by having at least 3.000 gold.

## Usecase fully dressed

Use Case	Description
Use Case Name	Usecase 2 - Roll the dice and play a turn
Scope	Adventure game
Level	User goal
Primary Actor	Player
Stakeholders and Interest	Customer(IOOuterActive) - Has an interest in creating an adventure game for 2 players
Preconditions	Game must be started, on a system that meets the game requirements.
Succes Guarantee	System prints tile story, and chnage the balance of the player. Balance gets printed, and user turn either continues or ends depending on the roll
Main Success Scenario	Player opens the RunGame.bat, and the CLI starts the game. Player 1 presses enter and gets a tile story, and a change in balance accordingly. According to the roll they either continue, or the turn is passed to player 2.
Extensions	2A. The tile story is displayed correctly, but the balance change is incorrect. 2B. The tile story is displayed incorrectly, but the balance change is correct. 2C. Nothing happens upon pressing "Enter". 2D. Players turn is not passed on correctly
Special Requirements	Player must be able to operate the program, read the tile stories and use a keyboard.

Figure 2: Usecase 2 - Fully Dressed

## 5.1 Diagrams/Models

### Class diagram

The class diagram is used to describe and show all the classes there will be in the software program.<sup>[1]</sup> We have made a class diagram for the adventure board game, which includes all the classes in the program, what they contain and their relations.

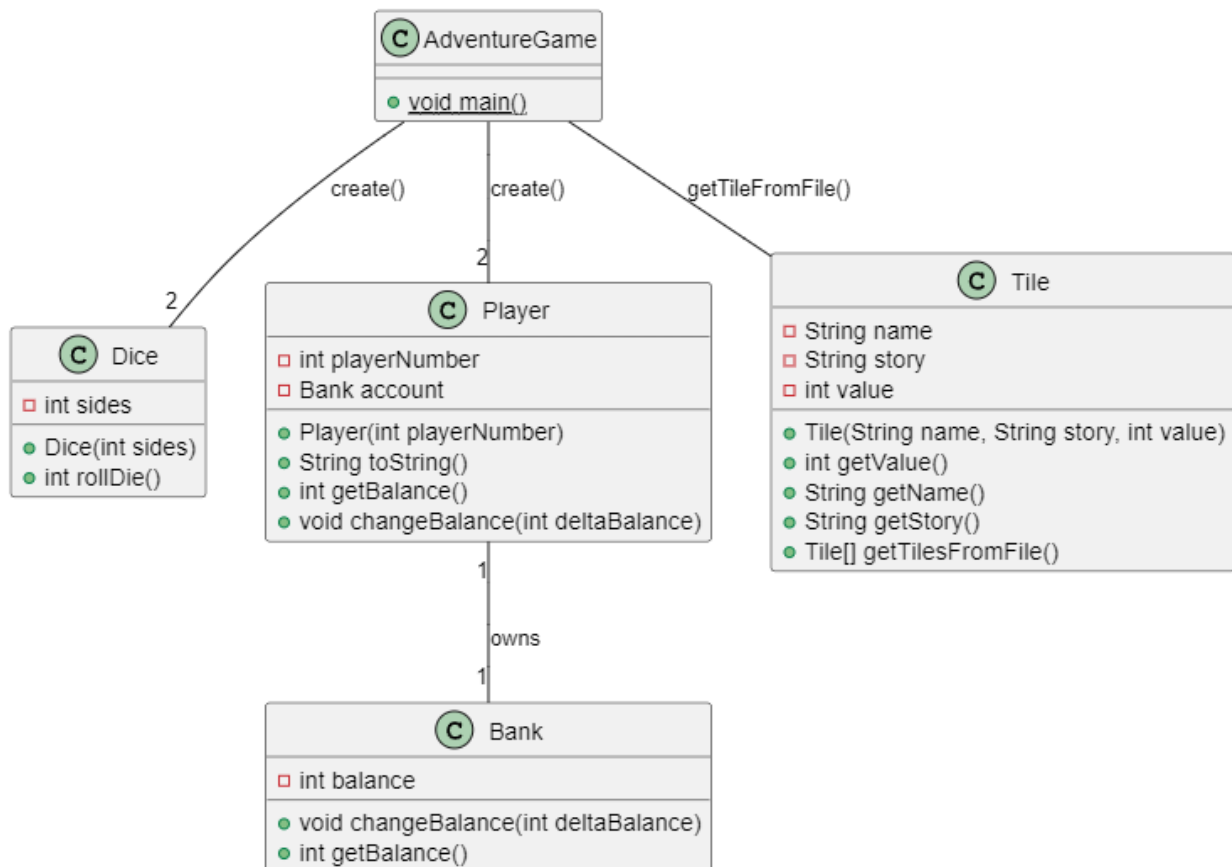


Figure 3: Class diagram



### System Sequence diagram

The system sequence diagram is used to show how the user interacts with the program. This means how the user moves through the program from the start of the program to the end of it. We have therefore made a system sequence diagram for an adventure board game. The diagram contains the player and the different parts of the program the player will have interactions with throughout the game.

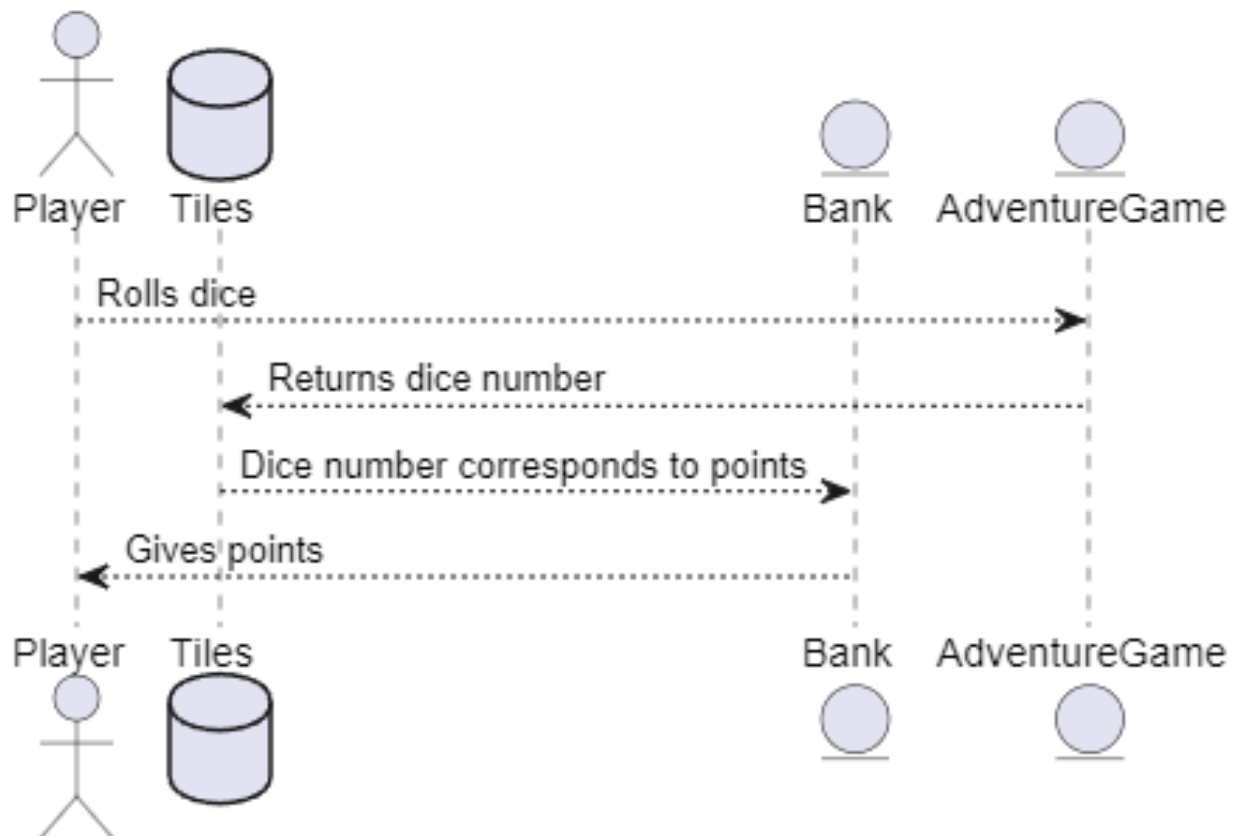


Figure 4: System Sequence Diagram

### Domain model

The domain model is very similar to the class diagram, but the class diagram shows all the classes in the program, what each class contains and how the relations between the classes. On the other hand, the domain model shows all the objects that the program contains and how they interact with each other. The domain model is usually more user-friendly and easy to understand for someone who knows nothing about the program or coding and are therefor good to show someone like the costumer. The group has for that reason also decided to construct a domain model for the program.<sup>[1]</sup>

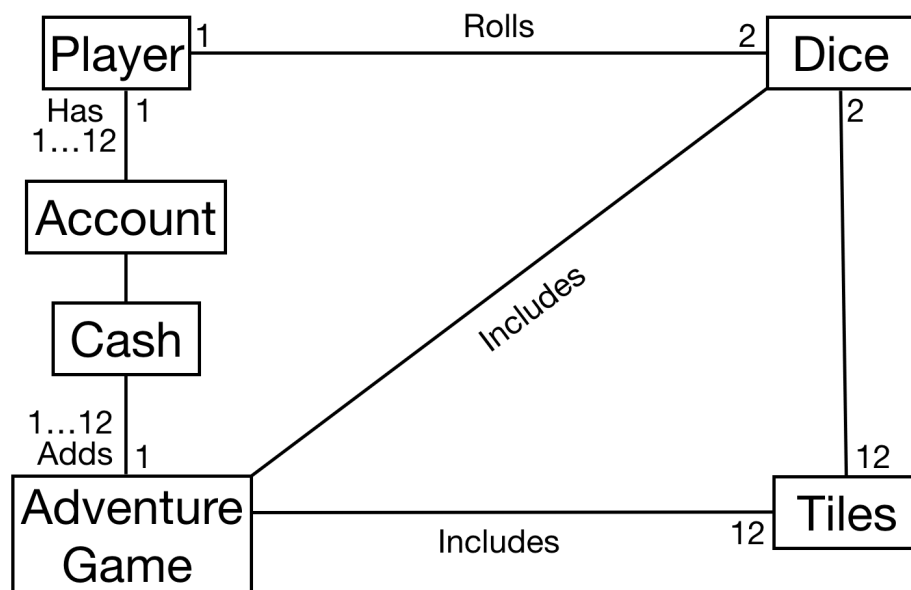


Figure 5: Domain model

## 6 Implementation

When it comes to implementation of the project the group has had many thoughts and work before starting making and programming the program, to see how different things would turn out and how we could make the best possible program and still meet the requirements and expectations of the customer. The group has therefore used FURPS ("functionality", "usability", "reliability", "performance" and "supportability").<sup>[1]</sup>

### **FURPS**

FURPS which stands for "functionality", "usability", "reliability", "performance" and "supportability". The group has used the model to classify the software quality.<sup>[1]</sup>

### **Functionality**

The functionality of the program is very simple. It is not a game with many requirements and therefore the functionality of the program is as said, very simple. It is always beneficial to have a program with a very simple functionality, if possible, as this means the program is easy for the users to use and learn. Which means the program is more user friendly.<sup>[1]</sup>

### **Usability**

The usability of the program is good. To improve usability we have created a .bat file that launches the game for the user. This means that you can launch the game with a simple double-click of the mouse and once you have opened the game, only one button is necessary to play. Therefore the usability could almost not be any better for the user.<sup>[1]</sup>

### **Reliability**

The reliability of the program is great. The software should work almost perfectly and the chance of a program crash is close to zero. It's also a small program and there are not that many requirements for the software to work.<sup>[1]</sup>

### **Performance**

The performance is extremely good. The program requires almost nothing to run. The user is firstly required to download the folder in which the game is made. You will be able to run the program on any modern computer no matter the specifications of the device, however to run the game, java 13, or a newer version, must be installed, furthermore the computer should support running a ".bat" file, which compiles everything needed for the user. The game can also run on the computers in "DTU's databarer". The response time for the program is very good as well, and is good enough to meet the requirement for the response time, which is no higher than 1/3 of a second. <sup>[1]</sup>

### **Supportability**

The supportability of the program is good since it is a small program. Therefore it is simple to test, adjust, adapt, install, maintain, configure, and localize.<sup>[1]</sup>

## 6.1 Testing

We have made iterative tests to confirm that our changes to the code have been effective and yielded the correct result. We have also constructed a probability test. The probability test is used to see if the dices are giving the statistically correct results for the dice throws. The probability test analyses probability for sums and doubles over the course of 1000 hypothetical dice throws and outputs the statistical deviation from the theoretical probability.<sup>[1]</sup>

Additionally we have created a JUnit test to ensure that a players bank balance never can be negative. The test confirms that our code is working as expected, as it shows that if a player loses more money than it has, the players bank balance will go to zero.

We have also done a great amount of testing on the game's interaction with the user. Here we have tested the game to make sure that it performs as expected, depending on the value of the thrown dice. As we received wrong outputs, we have optimized and adjusted the program and used this testing to better understand our flaws in the code and the perspective of the user.

## 7 Conclusion

We have created a board game,. Where two players play against each other to accumulate 3.000 gold to win. Which is done by rolling two dices and gather gold by landing on different tiles on the board. The customers vision was a bit unclear, so we have clarified everything to give ourselves something concrete to work with, by creating a requirement specification. Diagrams and models have been constructed to showcase how the program works. To ensure maximum accessibility for all the members of the group to our project, we have used Git and GitHub to push and pull our progress. A test was created to make sure our dice cup was as close to the statistical probabilities of rolling two dice as possible. At last we made sure that the customer does not need an IDE by converting the game into a bat-file.

## 8 Appendix

### 8.1 Litterature

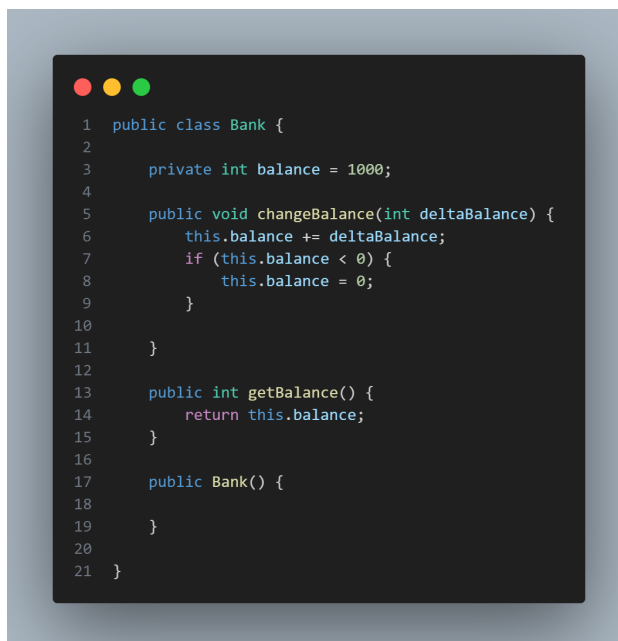
1. CDIO1 made by Group 07 [CDIO1-G7]

- As the CDIO 2 project is a further developed version of the CDIO 1 project, some of the contents of the report are quite similar. We have referenced in places where this is the case, but it is important to note that as the game is built on the grounds of DiceGame, much of the report is similar.

2. Github repository [[https://github.com/adel051700/G07\\_CDIO-2/](https://github.com/adel051700/G07_CDIO-2/)]

### 8.2 Code Upload

This is our code, for the CDIO 2 java project:



```
1 public class Bank {
2
3     private int balance = 1000;
4
5     public void changeBalance(int deltaBalance) {
6         this.balance += deltaBalance;
7         if (this.balance < 0) {
8             this.balance = 0;
9         }
10    }
11
12    public int getBalance() {
13        return this.balance;
14    }
15
16    public Bank() {
17
18    }
19
20
21 }
```

Figure 6: bank class

```
1 public class Player {
2     private int playerNumber;
3     private Bank account;
4
5     public Player(int playerNumber) {
6         this.playerNumber = playerNumber;
7         this.account = new Bank();
8     }
9
10    public String toString(){
11        return "Player " + this.playerNumber + " bank balance: " + getBalance();
12    }
13
14    public int getBalance() {
15        return this.account.getBalance();
16    }
17
18    public void changeBalance(int deltaBalance) {
19        this.account.changeBalance(deltaBalance);
20    }
21 }
22
23
24
```

Figure 7: player class

```
1 public Tile(String name, String story, int value)
2 {
3     this.name = name;
4     this.story = story;
5     this.value = value;
6 }
7
8 public int getValue()
9 {
10    return this.value;
11 }
12 public String getName()
13 {
14    return this.name;
15 }
16 public String getStory()
17 {
18    return this.story;
19 }
```

Figure 8: constructor for the Tile class

```
1 public static Tile[] getTilesFromFile(String language)
2 {
3     //Create Tiles from csv file of the chosen language, and create array of type Tile
4     // with all tiles
5
6     int numberOfTiles = 11;
7     Tile[] tileArr = new Tile[numberOfTiles];
8     try
9     {
10         var fileToRead = new File(language + ".csv");
11         var Scanner = new java.util.Scanner(fileToRead);
12         int i = 0;
13         while (Scanner.hasNextLine())
14         {
15             String tile = Scanner.nextLine();
16             String[] tileValues = tile.split(";");
17             String name = tileValues[0];
18             String story = tileValues[1];
19             int value = Integer.parseInt(tileValues[2]);
20             tileArr[i] = new Tile(name, story, value);
21             i++;
22         }
23         Scanner.close();
24     }
25     catch (Exception e)
26     {
27         System.out.println(e + " Error");
28     }
29
30     return tileArr;
31 }
32
33 }
```

Figure 9: creating the tiles from an array

```
1 if (player1Turn) {
2     int sum = dice1.rollDie() + dice2.rollDie();
3     System.out.println("You rolled a " + sum + "\n");
4     if (sum <= 10) {
5         player1Turn = false;
6     }
7     System.out.println(tileArr[sum - 2].getStory());
8     player1.changeBalance(tileArr[sum - 2].getValue());
9     System.out.println("\n" + player1.toString() + "\n" + ".....");
10 }
```

Figure 10: Player 1's turn