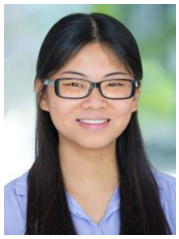
02312 62531 62532

Introductory Programming, Development Methods for IT Systems and Version Control and Test Methods

---

CDIO PART 2

Group 18

Maj Kyllesbech s216249	Gabriel H. Kierkegaard s215825	Mark Bidstrup s215782	Xiao Chen s215821
			

Deadline: 29. October 2021

GitHub repo: https://github.com/GabeKierk889/gruppe-18/tree/master/18_del2

1. Abstract

This report details a software project as a follow up to a previous client request. The aim is to design and develop a dice game to the specifications of the client (including game logic / rules) that can be played by two players using the machines in the DTU databar. We have used agile methods to design and develop the game, and throughout the process produced a number of artifacts to aid us in the development process. We have also conducted a number of unit tests to ensure that various methods used in the program work as intended.

2. Account of work hours

Date	Name(s)	Hours	Brief summary of work done
5/10	Maj, Mark, Gabriel, Xiao	2	Requirements gathering, domain model, class diagram, initial use case analysis
6/10	Xiao	1	Field class, field descriptions
6/10	Mark	1	Working with GUI
9/10	Gabriel	0.5	Created the account class
11/10	Mark	2	Working with GUI
12/10	Maj, Mark, Gabriel, Xiao	3	Requirements specification, use case diagram, fully dressed use case, sequence diagrams
13/10	Xiao	1	Working with GUI (starting to link GUI with Game elements)
19/10	Mark, Gabriel, Xiao	2	Creating general game flow
19/10	Xiao	1	Fixing minor bugs, minor code and github clean up
20/10	Mark	1.5	Writing implementation section, general readability of the game code and design sequence diagram
20/10	Gabriel	1	Wrote a simple test method for Account and updated pom to include junit5
23/10	Gabriel	1.5	Finished writing the tests for the Account class and rewrote some of the code, to ensure that the tests passed as inteded and also added to the report.
24/10	Xiao	1.5	Writing abstract, introduction, and conclusion, various report/github cleanup
26/10	Maj	2	Writing configuration requirements

3. Table of contents

4. Introduction

As a follow up to a previous project, the client has asked us to develop another game that can be played between two players using the machines in the DTU databar. The game should be playable without any noticeable delays, and if possible use a GUI. The game works as follows: The players each start with a balance of 1000, and take turns rolling two dice. The player lands on a field corresponding to the sum of the two dice, which has an effect on the player's balance (e.g., positive or negative). The first player to reach or exceed 3000 in their balance wins.
The project will be completed using agile methods to develop the program. In addition, we will develop the program in such a way that facilitates potential future reuse of objects (such as Player and Account objects), as well as write the code in a way that makes it easy to switch to other types of dice or other languages (e.g., grouping the in-game messages within the code).

5. Project planning

Below is the project plan that we developed at the start of the project.

	Duration	Start	Finish
Business modelling			
N/A			
Requirements			
Requirements gathering	2 days	04-10-2021	05-10-2021
Requirements validation	1 day	08-10-2021	08-10-2021
Analysis & Design			
Use case analysis	1 day	05-10-2021	05-10-2021
Design model creation	1 day	05-10-2021	05-10-2021
Sequence analysis	1 day	05-10-2021	05-10-2021
Implementation			
Code development - classes, methods	5 days	6-10-2021	11-10-2021
Code development - core game	1 day	12-10-2021	12-10-2021
Test			
Develop / build test	1 day	25-10-2021	25-10-2021
Conduct test	1 day	26-10-2021	26-10-2021
Deployment			
Release/ hand over to client	1 day		29-10-2021

6. Requirements

We have identified the following requirements from the client:

01: The game should show the results of the dice throw (i.e., the face values of the dice) within 333 milliseconds (when used without gui).

02: The game should show player name and player balance.

03: The game should end when a player reaches a balance of 3000.

04: The game language should be easy to change.

05: The game should make it easy to switch dice.

06: The game should be playable by two players.

07: The game should be playable on the databars' computers without significant delay.

08: The game should work according to the specified rules.

09: The game's player and player balance should be usable in other games.

10: The game should test to ensure that the balance cannot be negative.

7. Analysis

We have made the following use case diagram.

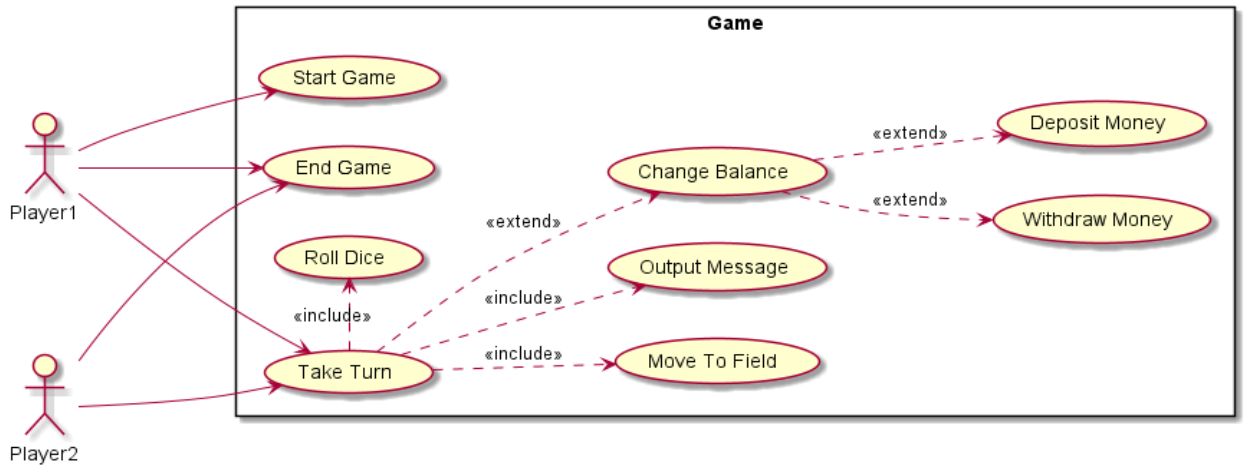


Figure 7.1: Use case diagram

Below is the use case in brief form for the whole game:

Start the game: The system outputs an announcement to the players to start the game. Player one starts the game.

- Take a turn *(fully dressed use case below)*
    - Roll dice
    - Move to field
    - Output message
    - Deposit money or Withdraw money (update account)

End the game: The player who first reaches 3000 in their account is declared the winner. The system outputs a message. The game ends.

We have detailed a fully dressed use case for the main use case in the game: Take a turn

Use case: Take a turn

Scope: Dice Game application

Level: user goal

Primary actor: the players

Stakeholders and interests: The players want to interact with the game to play it while observing their score and try to win.

Preconditions: Both players are established and each have an account. The system has determined the current player who needs to take a turn.

Success guarantee: Current player has thrown the dice and moved to a corresponding field. Current player has had their account balance updated accordingly. The system passes the turn to the next player.

Main success scenario:

1. Current player rolls the dice
2. System shows the outcome of the throw (the two dice)
3. System moves the current player to the field corresponding to the sum of the dice
4. System gives an output message about the relevant field
5. System deposits/withdraws an amount to the player's balance according to the effect of the relevant field
6. System shows the player's new balance
7. System passes the turn to the next player

Extensions:

*a. On a player's turn, if the player rolls a pair of dice with sum 10:

1. The system gives the current player an extra turn

*b. On a player's turn, if the player's account balance reaches 3000 or more:

2. The system records the player as the winner, outputs a message, and ends the game.

Special requirements:

- None

Technology and data variations list:

- Operating system with Java installed

Frequency of occurrence: Nearly continuous (discrete but short turns)

We have developed the following domain model for the game.

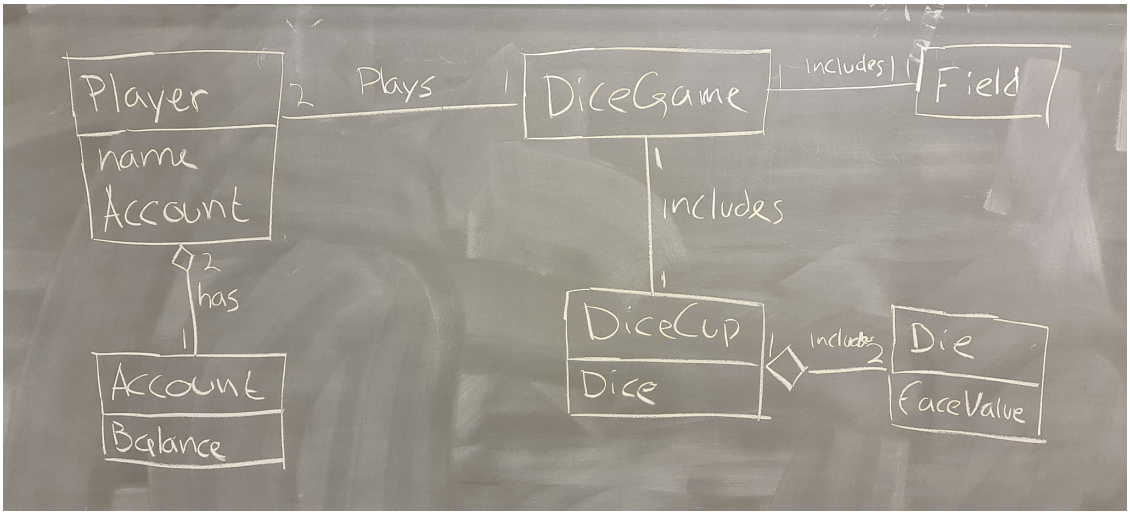


Figure 7.2: Domain model

We have made the following system sequence diagram

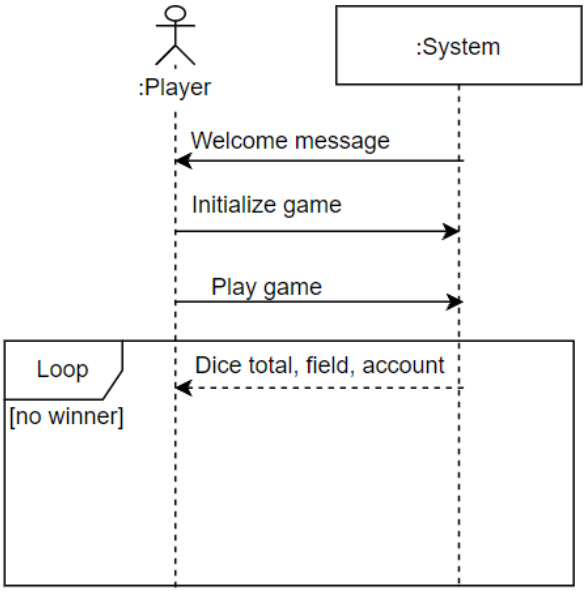


Figure 7.3: System sequence diagram

8. Design

We developed the following design class diagram for the game.

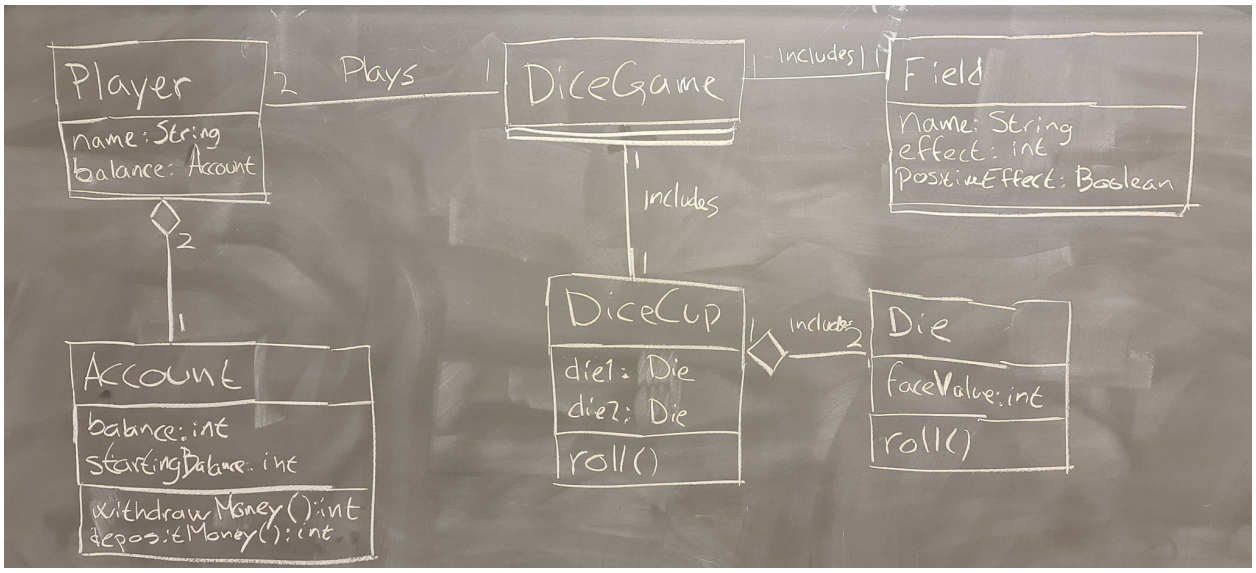


Figure 8.1: Design class diagram

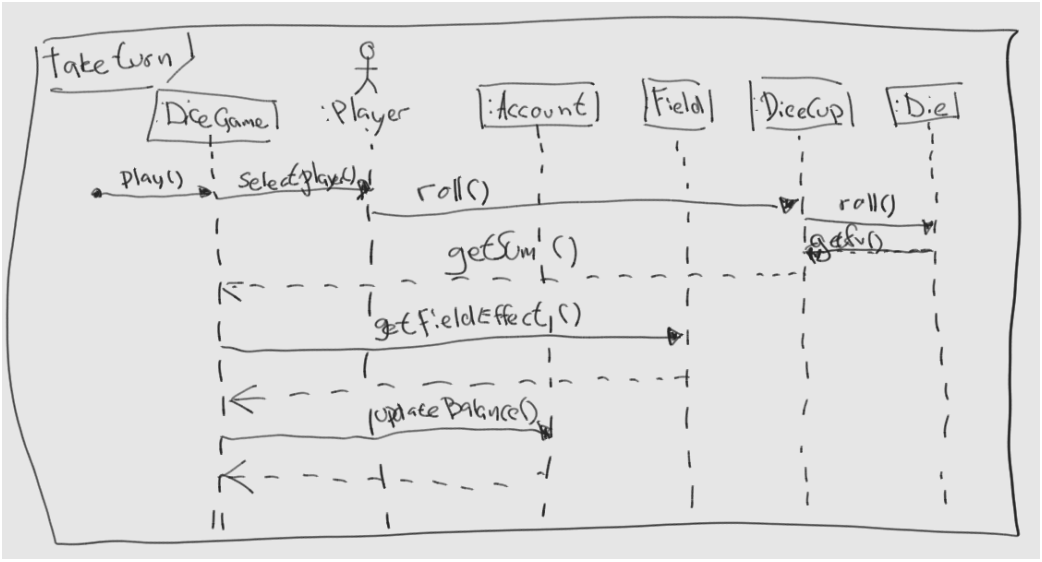


Figure 8.2: Design sequence diagram sketch

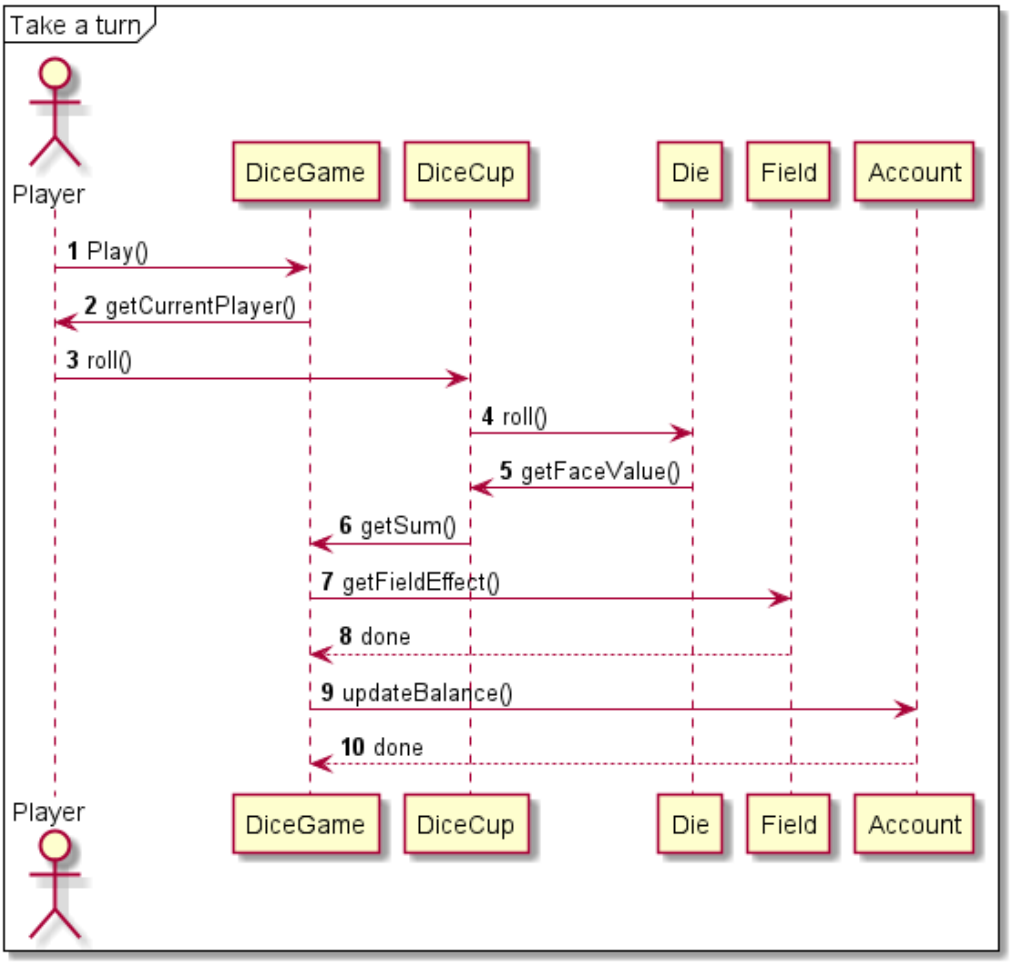


Figure 8.3: Design sequence diagram

9. Implementation

Note: Please see the full code in the appendix.

We have implemented the Matador GUI for this game by setting up a custom board with an array of 12 streets and setting them up by instantiating them and setting the street specifics in a for loop.

```
// Setting up fields
GUI_Field[] fields = new GUI_Field[11];
GUI_Street[] streets = new GUI_Street[11];

for (int i = 0; i < 11; i++) {
    streets[i] = new GUI_Street();
    streets[i].setTitle(""+(i+2));
    streets[i].setSubText(game.getField(i).getFieldSubtext());
    streets[i].setDescription(game.getField(i).getFieldDescription());
    streets[i].setBackGroundColor(Color.gray);
    fields[i] = streets[i];
}

GUI_Board board = new GUI_Board(fields, Color.lightGray); // Setting up fields and background color
```

Figure 9.1: Setting up the board.

The player objects have been assigned to an array, so that we can use our getCurrentPlayer() and nextPlayer() methods to manage player turns and who wins.

```
// Setting up players
GUI_Player[] player = new GUI_Player[game.getTotalPlayers()];
player[0]= new GUI_Player(game.getPlayerObject( playernumber: 1).getName(),
        game.getPlayerObject( playernumber: 1).getAccount().getBalance(),
        car1);
player[1]= new GUI_Player(game.getPlayerObject( playernumber: 2).getName(),
        game.getPlayerObject( playernumber: 2).getAccount().getBalance(),
        car2);

board.addPlayer(player[1]);
board.addPlayer(player[0]);
```

Figure 9.2: Setting up the players on the board.

```java
public void switchTurn(boolean extraTurn) {
    if (!extraTurn) {
        if (currentPlayer < totalPlayers)
            currentPlayer++;
        else
            currentPlayer = 1; }
}

public int nextPlayer(boolean extraTurn) {
    if (!extraTurn) {
        if (currentPlayer < totalPlayers)
            nextPlayer = currentPlayer+1;
        else
            nextPlayer = 1; }
    return nextPlayer;
}
```

Figure 9.3: switchTurn() and nextPlayer() methods.

The field objects have been initialized with 4 arguments that describe the effect of the corresponding field and used in the updateBalance() method to manage the player balance.

```java
gameFields = new Field[]{
        new Field( fieldSubtext: "Tower",  fieldDescription: "You climbed an abandoned Tower and found a treasure at the top which you sell for 250",
                fieldPositiveEffect: true, fieldEffect: 250),
        new Field( fieldSubtext: "Crater", fieldDescription: "You dropped your watch down a deep Crater - it will cost you 100 to replace it",
                fieldPositiveEffect: false, fieldEffect: 100),
        new Field( fieldSubtext: "Palace Gates", fieldDescription: "You made it to the Palace Gates and receive 100 from the generous king",
                fieldPositiveEffect: true, fieldEffect: 100),
        new Field( fieldSubtext: "Cold Desert", fieldDescription: "You find yourself in the Cold Desert - you need to pay 20 for a warm scarf",
                fieldPositiveEffect: false,  fieldEffect: 20),
        new Field( fieldSubtext: "Walled City", fieldDescription: "You made it to the Walled City - you spend the day working at the inn and earn 180",
                fieldPositiveEffect: true, fieldEffect: 180),
        new Field( fieldSubtext: "Monastery", fieldDescription: "You made a retreat to the Monastery. You meditate in peace. Your balance is unchanged",
                fieldPositiveEffect: true, fieldEffect: 0),
        new Field( fieldSubtext: "Black Cave", fieldDescription: "You need to venture into the Black Cave - you have to pay 70 to buy some high quality headlights",
                fieldPositiveEffect: false, fieldEffect: 70),
        new Field( fieldSubtext: "MountainHut", fieldDescription: "You made it to some Huts in the Mountain - a generous mountain dweller gives you 60 in pocket money for your onward journey",
                fieldPositiveEffect: true,  fieldEffect: 60),
        new Field( fieldSubtext: "WereWall", fieldDescription: "You accidentally went to the WereWall! You lose 80 of your money as you run away from the werewolves, but you get an extra turn",
                fieldPositiveEffect: false, fieldEffect: 80),
        new Field( fieldSubtext: "The Pit", fieldDescription: "Oh no, you fell into The Pit - you are completely muddy and dirty and have to pay 50 to buy new clothes",
                fieldPositiveEffect: false, fieldEffect: 50),
        new Field( fieldSubtext: "Goldmine", fieldDescription: "You hit the jackpot! You found a Goldmine in the mountains and cash in 650!",
                fieldPositiveEffect: true, fieldEffect: 650),
};
```

Figure 9.4: Initialization of game fields.

```java
public void updateBalance(int playernumber, int fieldnumber) {
    if (gameFields[fieldnumber-2].getFieldPosiveEffect()) //because field array number starts at 0 while field number starts at 2
        player[playernumber-1].getAccount().depositMoney(gameFields[fieldnumber-2].getFieldEffect());
    else
        player[playernumber-1].getAccount().withdrawMoney(gameFields[fieldnumber-2].getFieldEffect());
}
```

Figure 9.5: The update balance method.

A mouse input listener is used to track mouse click events from the players to implement player action to make the GUI interactive to the players.

```
// User input: turn action on mouse released
MouseInputListener listen = new MouseInputListener() {
    @Override
    public void mouseClicked(MouseEvent e) {
        // checking if player balance is lower than 3000
        if (!game.isGameOver()) {
```

Figure 9.6: Snippet of implementation of mouse input listener.

10. Test

We have used a number of JUnit tests to test that the program will run as intended.

A test to make sure that the withdrawal of money can't be negative and also that the account balance never can be negative.

```
@org.junit.jupiter.api.Test
void withdrawMoney() {
    for (int i = -1000; i <= 1000; i++) {
        if (i >= 0) {
            account.withdrawMoney(i);
            assertEquals(account.getBalance(), actual: 1000 - i);
            account.setCurrentBalance(1000);
        }

        if (i < 0) {
            account.withdrawMoney(i);
            assertEquals( expected: 1000,account.getBalance());
        }
    }
}
```

Figure 10.1: Test of withdrawMoney method

A test to ensure that deposits can only be positive.

```
@org.junit.jupiter.api.Test
void depositMoney() {

    for (int i = -1000; i<= 1000; i++) {
        if (i >= 0) {
            account.depositMoney(i);
            assertEquals(account.getBalance(), actual: 1000 + i);
            account.setCurrentBalance(1000);
        }

        if (i < 0) {
            account.depositMoney(i);
            assertEquals( expected: 1000,account.getBalance());
        }
    }
}
```

Figure 10.2: Test of depositMoney method

Test that ensures that the balance can only be set to a positive number.

```
@org.junit.jupiter.api.Test
void setCurrentBalance() {
    for (int i = -1000; i <= 1000; i++) {
        if (i >= 0) {
            account.setCurrentBalance(i);
            assertEquals(account.getBalance(), i);
        }

        if (i < 0) {
            account.setCurrentBalance(i);
            assertEquals( expected: 1000, account.getBalance());
        }
    }
}
```

Figure 10.3: Test of setCurrentBalance method

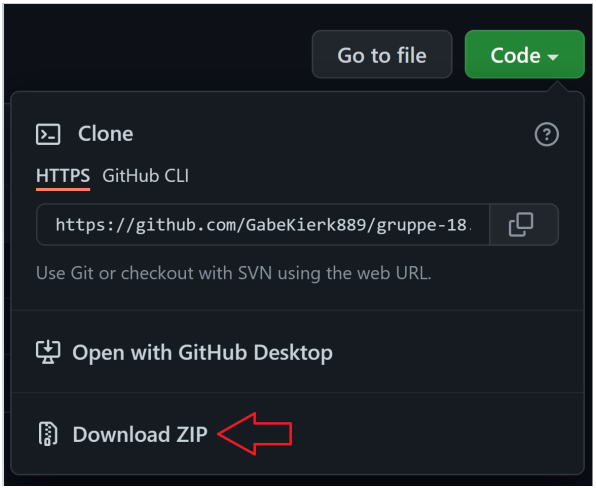
11. Configuration requirements

11.1 Minimum requirements

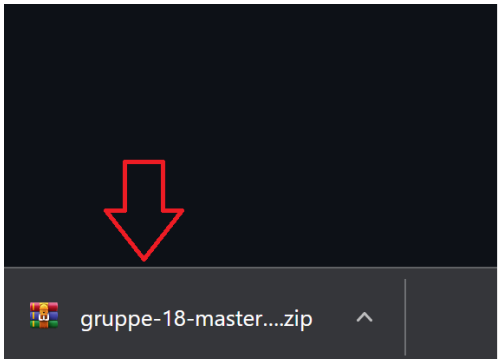
- Pc with intellij installed
- Zip unzipper like winrar
- Java installed

11.2 Installation and activation of the code

1. Go to github https://github.com/GabeKierk889/gruppe-18
2. Download as a zip file under code

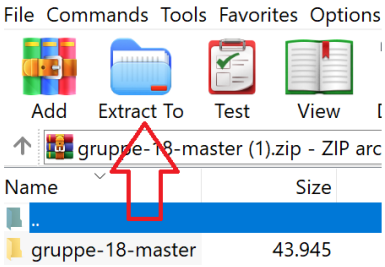


3. Find file in download



4. Open it with winrar or other unzipping program

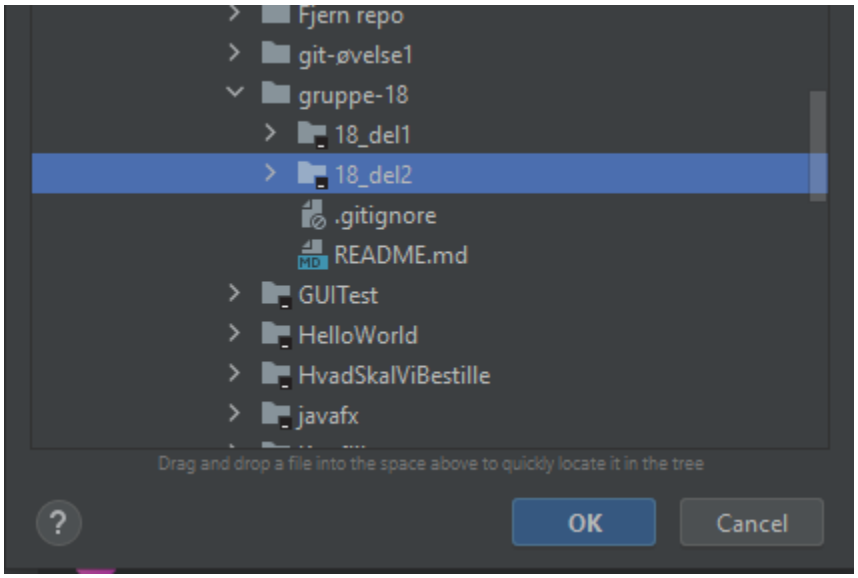
5. Extract the file



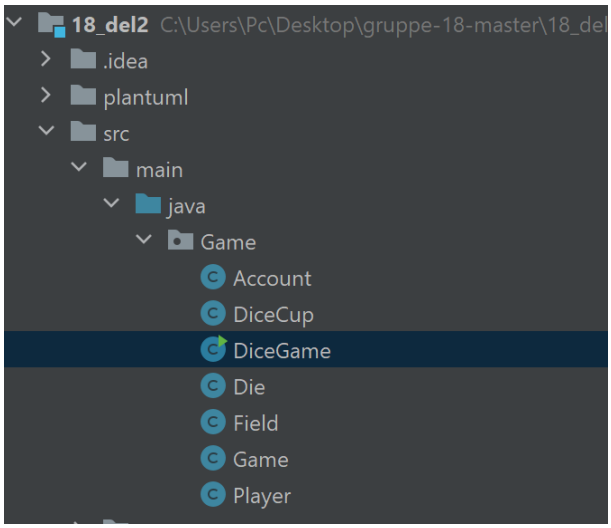
6. Put the file where you can find it

7. Open intellij

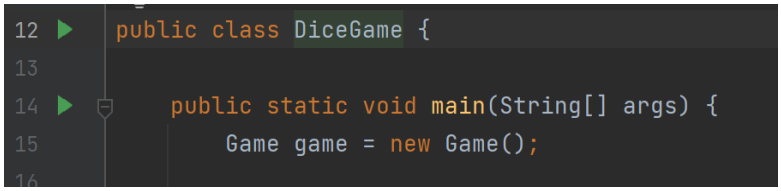
8. Select the 18_del2 project directory, press ok



9. Go to src → main → java → game → DiceGame.java



10. Go to line 12 in the code and press the green play button



12. Conclusion

We successfully developed the program to the specifications of the client using agile methods and an iterative development approach. We were able to implement a GUI to enhance the visuals of the game. We have also conducted a number of tests on the program to ensure that various individual methods of the program all run as intended, and to ensure that the game can be played in the DTU databars, as specified by the client.
Overall, the team worked well together to complete the project, and made good use of shared/ remote repositories and dividing work between team members.
As for improvements/ suggestions for future projects, we could undertake more thorough testing including integration and system testing in addition to unit tests, in order to ensure the overall integrity of the program. We could also further practice how to conduct code development work in parallel by using git branching and by assigning clear individual areas of responsibility within code development.

13. Appendix

13.1 Bibliography and references

[Blank]

13.2 Code

See following pages

```java
package Game;

public class Account {
    private int currentBalance;
    private static final int STARTINGBALANCE = 1000;

    public Account () {
        currentBalance = STARTINGBALANCE;
    }

    // Withdraws money.
    public void withdrawMoney (double withdrawal) {
        // Only allows withdrawal if the amount is greater than zero and also the withdrawal amount is greater than
        // the current balance.
        if (withdrawal > 0) {
            if (withdrawal <= currentBalance) {
                this.currentBalance -= withdrawal;
            } else {
                // If the withdrawal amount is greater than the account balance, then the account balance i reset
                // to zero.
                this.currentBalance = 0;
            }
        }
    }

    // Deposits money.
    public void depositMoney(double deposit) {
        // Only allows depositing a positive amount to the current balance.
        if (deposit > 0) {
            this.currentBalance += deposit;
        }
    }

    public void setCurrentBalance(int balance) {
        if (balance >= 0) {
            this.currentBalance = balance;
```

```
37          }
38      }
39
40      public int getBalance() {
41          return currentBalance;
42      }
43  }
44
```

```java
package Game;

public class DiceCup {
    private Die die1, die2;

    public DiceCup() {
        die1 = new Die();
        die2 = new Die();
    }

    public void roll() {
        die1.roll();
        die2.roll();
    }

    public int getSum() {
        int sum = die1.getFaceValue() + die2.getFaceValue();
        return sum;
    }

    public int getDie1Value() {
        return die1.getFaceValue();
    }
    public int getDie2Value() {
        return die2.getFaceValue();
    }

    public Die getDie1() {
        return die1;
    }
    public Die getDie2() {
        return die2;
    }

    public int sameFaceValue() {
        int result = 0;
        if (die1.getFaceValue() == die2.getFaceValue()) {
            result = 1;
        }
```

```
40          return result;
41      }
42
43      public String toString() {
44          String result = "Die 1: " + die1.getFaceValue
    () + "\t" + "Die 2: " + die2.getFaceValue() + "\t" +
    "Sum: " + (die1.getFaceValue()+die2.getFaceValue());
45          return result;
46      }
47  }
48
```

```java
package Game;

import gui_fields.GUI_Board;
import gui_fields.GUI_Field;
import gui_fields.GUI_Player;
import gui_fields.GUI_Street;
import gui_fields.*;
import javax.swing.event.MouseInputListener;
import java.awt.*;
import java.awt.event.MouseEvent;

public class DiceGame {

    public static void main(String[] args) {
        Game game = new Game();

        // Setting up fields
        GUI_Field[] fields = new GUI_Field[11];
        GUI_Street[] streets = new GUI_Street[11];

        for (int i = 0; i < 11; i++) {
            streets[i] = new GUI_Street();
            streets[i].setTitle(""+(i+2));
            streets[i].setSubText(game.getField(i).
getFieldSubtext());
            streets[i].setDescription(game.getField(i
).getFieldDescription());
            streets[i].setBackGroundColor(Color.gray
);
            fields[i] = streets[i];
        }

        GUI_Board board = new GUI_Board(fields, Color
.lightGray); // Setting up fields and background
color

        // Setting up cars
        GUI_Car car1 = new GUI_Car();
        car1.setPrimaryColor(Color.RED);
        car1.setSecondaryColor(Color.ORANGE);

```

```
37          GUI_Car car2 = new GUI_Car();
38          car2.setPrimaryColor(Color.BLUE);
39          car2.setSecondaryColor(Color.CYAN);
40
41          // Setting up players
42          GUI_Player[] player = new GUI_Player[game.
    getTotalPlayers()];
43          player[0]= new GUI_Player(game.
    getPlayerObject(1).getName(),
44                  game.getPlayerObject(1).getAccount().
    getBalance(),
45                  car1);
46          player[1]= new GUI_Player(game.
    getPlayerObject(2).getName(),
47                  game.getPlayerObject(2).getAccount().
    getBalance(),
48                  car2);
49
50          board.addPlayer(player[1]);
51          board.addPlayer(player[0]);
52
53          // Message (can be used for user input with
    second arg)
54          board.getUserInput("Welcome to DiceGame v2!
    Each player starts with a balance of 1000. " +
55                  "The first to reach 3000 wins.\n\n
    Player 1's turn\nClick anywhere to roll the dice");
56
57          // Referencing the dice to the DiceCup set up
     by the game
58          Die die1 = game.getCup().getDie1();
59          Die die2 = game.getCup().getDie2();
60
61          // User input: turn action on mouse released
62          MouseInputListener listen = new
    MouseInputListener() {
63              @Override
64              public void mouseClicked(MouseEvent e) {
65                  // checking if player balance is
    lower than 3000
66                  if (!game.isGameOver()) {
```

```
67
68                      // removes car on the previous
   field
69                      fields[game.getCup().getSum() -
   2].removeAllCars();
70
71                      game.getCup().roll();
72
73                      // Sets the dice on the board
74                      board.setDice((int) (Math.random
   () * 4 + 3),
75                              (int) (Math.random() * 5
    + 2),
76                              die1.getFaceValue(),
77                              (int) (Math.random() *
   359),
78                              (int) (Math.random() * 4
    + 7),
79                              (int) (Math.random() * 5
    + 2),
80                              die2.getFaceValue(),
81                              (int) (Math.random() *
   359));
82
83                      // Sets the players car on the
   street corresponding to the dice roll
84                      streets[game.getCup().getSum
   () - 2].setCar(player[game.getCurrentPlayer() - 1],
   true);
85
86                      // Displays player turn and
   player action
87                      board.getUserInput("Player " +
   game.getCurrentPlayer() +
88                              ": " + game.getField(
   game.getCup().getSum() - 2).getFieldDescription() +
89                              "\n\nPlayer " + game.
   nextPlayer(game.getCup().getSum() == 10) + "'s" +
90                              " turn\nClick anywhere
   to roll the dice");
91
```

```
 92                      // Updating player balance and
    showing the updated player balances
 93                      game.updateBalance(game.
    getCurrentPlayer(), game.getCup().getSum());
 94                      player[0].setBalance(game.
    getPlayerObject(1).getAccount().getBalance());
 95                      player[1].setBalance(game.
    getPlayerObject(2).getAccount().getBalance());
 96
 97                      // Checking if player balance is
     higher than 3000
 98                      if (game.isGameOver()) {
 99                          game.setWinner(game.
    getCurrentPlayer());
100
101                          // Displays winning player
102                          board.getUserInput("Player "
     + game.getCurrentPlayer() +
103                                  ": " + game.getField
    (game.getCup().getSum() - 2).getFieldDescription() +
104                                  "\n\nPlayer " + game
    .getWinner()+ " wins this game" );
105                      }
106
107                      // Changes "currentplayer" to
    the next player (unless current player gets an extra
     turn)
108                      game.switchTurn(game.getCup().
    getSum() == 10);
109                  }
110              }
111
112              @Override
113              public void mousePressed(MouseEvent e) {
114              }
115
116              @Override
117              public void mouseReleased(MouseEvent e
    ) {
118
119              }
```

```
120
121             @Override
122             public void mouseEntered(MouseEvent e) {
123
124             }
125
126             @Override
127             public void mouseExited(MouseEvent e) {
128
129             }
130
131             @Override
132             public void mouseDragged(MouseEvent e) {
133
134             }
135
136             @Override
137             public void mouseMoved(MouseEvent e) {
138
139             }
140         };
141         board.addMouseListener(listen);
142
143
144     }
145 }
146
147
```

```java
package Game;

public class Die {
    private int faceValue;
    private final int MAX = 6;

    public Die() {
        faceValue = 1;
    }

    public int roll() {
        faceValue = (int) (Math.random() * MAX) + 1;
        return faceValue;
    }

    public int getFaceValue() {
        return faceValue;
    }

    public String toString () {
        String str = Integer.toString(faceValue);
        return str;
    }
}

```

```java
package Game;

public class Field {
    private static int totalnumberOfFields = 0;
    private final int fieldNumber;
    private final String fieldDescription;
    private final String fieldSubtext;
    private final int fieldEffect;
    private final boolean fieldPositiveEffect; // false for withdraw, true for deposit

    public Field(String fieldSubtext, String fieldDescription, boolean fieldPositiveEffect, int fieldEffect) {
        this.fieldSubtext = fieldSubtext;
        this.fieldDescription = fieldDescription;
        this.fieldNumber = totalnumberOfFields;
        totalnumberOfFields++;
        this.fieldPositiveEffect = fieldPositiveEffect;
        this.fieldEffect = fieldEffect;
    }
    //no setters as all attributes are final variables
    public boolean getFieldPosiveEffect (){ return fieldPositiveEffect; }
    public int getFieldEffect (){ return fieldEffect ; }
    public int getFieldNumber (){ return fieldNumber ; }
    public int getTotalnumberOfFields (){ return totalnumberOfFields-1; }
    public String getFieldDescription (){ return fieldDescription; }
    public String getFieldSubtext() {return fieldSubtext; }

    public String toString() {
        return "Field " + fieldNumber  + ". " + fieldDescription;
    }
```

```
30 }
31
```

```java
package Game;

public class Game {
    private Player player[];
    private Field gameFields[];
    private DiceCup cup;
    private int currentPlayer;
    private int nextPlayer;
    private final int totalPlayers = 2;
    private int winner;
    private final int winningAmount = 3000;

    public Game() {
        currentPlayer = 1;
        nextPlayer = 1;
        cup = new DiceCup();
        player = new Player[]{new Player("Player 1"), new Player("Player 2")};
        gameFields = new Field[]{
                new Field("Tower", "You climbed an abandoned Tower and found a treasure at the top which you sell for 250",
                        true,250),
                new Field("Crater","You dropped your watch down a deep Crater - it will cost you 100 to replace it",
                        false,100),
                new Field("Palace Gates","You made it to the Palace Gates and receive 100 from the generous king",
                        true,100),
                new Field("Cold Desert","You find yourself in the Cold Desert - you need to pay 20 for a warm scarf",
                        false, 20),
                new Field("Walled City","You made it to the Walled City - you spend the day working at the inn and earn 180",
                        true,180),
                new Field("Monastery","You made a retreat to the Monastery. You meditate in peace. Your
```

```
29  balance is unchanged",
30                          true,0),
31                  new Field("Black Cave","You need to
   venture into the Black Cave - you have to pay 70 to
   buy some high quality headlights",
32                          false,70),
33                  new Field("MountainHut","You made it
   to some Huts in the Mountain - a generous mountain
   dweller gives you 60 in pocket money for your onward
   journey",
34                          true, 60),
35                  new Field("WereWall","You
   accidentally went to the WereWall! You lose 80 of
   your money as you run away from the werewolves, but
   you get an extra turn",
36                          false,80),
37                  new Field("The Pit","Oh no, you fell
   into The Pit - you are completely muddy and dirty and
    have to pay 50 to buy new clothes",
38                          false,50),
39                  new Field("Goldmine","You hit the
   jackpot! You found a Goldmine in the mountains and
   cash in 650!",
40                          true,650),
41          };
42      }
43
44      public void updateBalance(int playernumber, int
   fieldnumber) {
45          if (gameFields[fieldnumber-2].
   getFieldPosiveEffect()) //because field array number
   starts at 0 while field number starts at 2
46              player[playernumber-1].getAccount().
   depositMoney(gameFields[fieldnumber-2].getFieldEffect
   ());
47          else
48              player[playernumber-1].getAccount().
   withdrawMoney(gameFields[fieldnumber-2].
   getFieldEffect());
49      }
50
```

```
51      public void switchTurn(boolean extraTurn) {
52          if (!extraTurn) {
53              if (currentPlayer < totalPlayers)
54                  currentPlayer++;
55              else
56                  currentPlayer = 1; }
57      }
58
59      public int nextPlayer(boolean extraTurn) {
60          if (!extraTurn) {
61              if (currentPlayer < totalPlayers)
62                  nextPlayer = currentPlayer+1;
63              else
64                  nextPlayer = 1; }
65          return nextPlayer;
66      }
67
68      public boolean isGameOver() {
69          return ( (player[0].getAccount().getBalance
    () >= winningAmount) ||
70                  (player[1].getAccount().getBalance
    () >= winningAmount) );
71      }
72
73      public void setWinner(int winner) {
74          this.winner = winner;
75      }
76
77      public DiceCup getCup() { return cup;  }
78      public Field getField(int no) { return gameFields
    [no]; }
79      public Player getPlayerObject(int playernumber
    ) { return player[playernumber-1]; }
80      public int getCurrentPlayer() { return
    currentPlayer; }
81      public int getTotalPlayers() {return totalPlayers
    ; }
82      public int getNextPlayer () { return nextPlayer;}
83      public int getWinner() { return winner; }
84  }
85
```

```java
package Game;

public class Player {
    private String name;
    private Account account;

    public Player(String name){
        Account acct = new Account();
        account = acct;
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public Account getAccount() {
        return account;
    }

}

```

```java
package Game;

import org.junit.jupiter.api.Assertions;

import static org.junit.jupiter.api.Assertions.*;

class AccountTest {
    Account account = new Account();

    // Test that checks for the balance withdrawed can never be negative, and also that the account balance never can
    // go under zero.
    @org.junit.jupiter.api.Test
    void withdrawMoney() {
        for (int i = -1000; i <= 1000; i++) {
            if (i >= 0) {
                account.withdrawMoney(i);
                assertEquals(account.getBalance(), 1000 - i);
                account.setCurrentBalance(1000);
            }

            if (i < 0) {
                account.withdrawMoney(i);
                assertEquals(1000,account.getBalance());
            }
        }
    }

    // Test that checks for the balance deposited can only be positive.
    @org.junit.jupiter.api.Test
    void depositMoney() {

        for (int i = -1000; i<= 1000; i++) {
            if (i >= 0) {
                account.depositMoney(i);
                assertEquals(account.getBalance(), 1000 + i);
```

```
36                  account.setCurrentBalance(1000);
37              }
38
39              if (i < 0) {
40                  account.depositMoney(i);
41                  assertEquals(1000,account.getBalance
   ());
42              }
43          }
44      }
45
46      // Test that checks that it's possible to set the
    balance, and also that the balance can not be set to
    a negative
47      // number.
48      @org.junit.jupiter.api.Test
49      void setCurrentBalance() {
50          for (int i = -1000; i <= 1000; i++) {
51              if (i >= 0) {
52                  account.setCurrentBalance(i);
53                  assertEquals(account.getBalance(), i
   );
54              }
55
56              if (i < 0) {
57                  account.setCurrentBalance(i);
58                  assertEquals(1000, account.getBalance
   ());
59              }
60          }
61      }
62
63      // Test that checks that it's possible to get the
    current balance.
64      @org.junit.jupiter.api.Test
65      void getBalance() {
66          assertEquals(account.getBalance(),account.
   getBalance());
67      }
68 }
```