SPŠE Ječná

Information Technology

Secondary Industrial School of Electrical Engineering, Prague 2, Ječná 30

SPACE PARKOUR

Jiří Holub C2c

Information Technology

2025

Contents

1. Goal of the work 3
2. Game features 3-4-5-6
3. System requirements 6
4. Basic structure 6-7
5. Test data 7-8
6. User manual 8
7. Conclusion 8-9

1. Goal of the work:

The goal of this project was to create a simple and fun 2D game. The player starts in the middle of space, and the objective is to jump as high as possible on space platforms in order to achieve the highest score. While jumping, the player can also collect coins, which can later be used in the shop to purchase skins for the character, the platforms, and even the background.

2. Game features:

The game has several features. One of the main mechanics is, of course, player movement. Player movement works in a way that the player's character itself doesn't move; instead, everything around it moves, so the character always stays in the center. The player can move using the **A** and **D** keys (left and right), **W** or **SPACE** (jump), and **SHIFT** (sprint).

Next important mechanic is individual gravity, which prevents the player from jumping infinitely. It is implemented in the main game loop and is disabled only when the player is colliding with a platform (standing on it) or briefly when the player jumps.

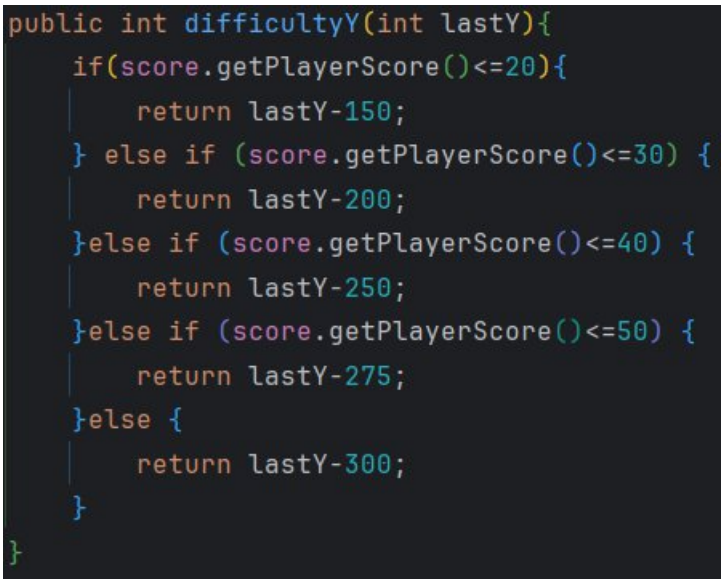
```
public void gameGravity(){
    boolean colision = collisionManager.isOnPlatform(player,platforms);
    if(!colision){
        for(int i = 0;i<platforms.size();i++){
            platforms.get(i).setLocation(platforms.get(i).getX(), y: platforms.get(i).getY()-5);
            player.setStaying(false);
        }
        for(int i = 0;i<coinGenerator.getCoins().size();i++){
            Coin coin = coinGenerator.getCoins().get(i);
            coin.setLocation(coin.getX(), y: coin.getY()-5);
        }
    }
}
```

Method that handles the player's fall

```
public void verticalMove(){
    if(player.isJumping() || !player.isStaying()){
        player.gameJump(platforms,coinGenerator);
    }else {
        gameGravity();
    }
}
```

Method called directly in the Game Loop

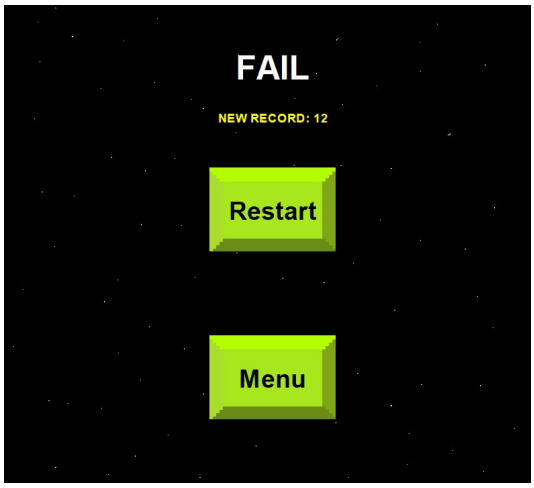
Another important function of the program is the generation and deletion of platforms. The game always has a limited number of platforms, so new platforms are generated gradually. Platforms are generated at a fixed Y height based on the difficulty, which increases as your score gets higher. The X coordinate is set randomly within a certain range, which also expands as your score increases. Platforms are deleted when they fall below a certain Y height.



```
public int difficultyY(int lastY){
    if(score.getPlayerScore()<=20){
        return lastY-150;
    } else if (score.getPlayerScore()<=30) {
        return lastY-200;
    }else if (score.getPlayerScore()<=40) {
        return lastY-250;
    }else if (score.getPlayerScore()<=50) {
        return lastY-275;
    }else {
        return lastY-300;
    }
}
```

Method that sets the Y coordinate of a platform based on the score

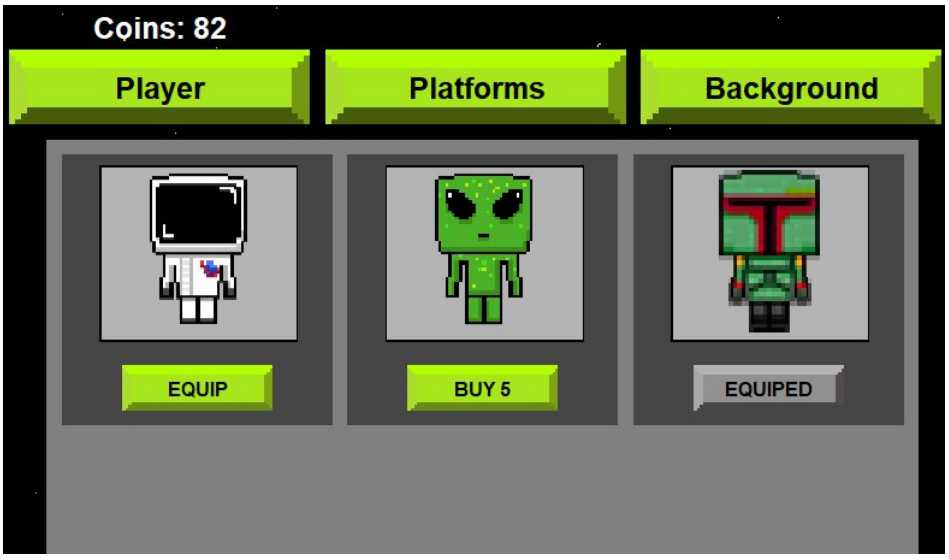
When the player falls or misses landing on another platform and doesn't catch themselves, the game is over and a Restart menu appears with buttons for RESTART and MENU. If the player beats their highest score, a yellow text message appears in this Restart menu, congratulating them on setting a new record.



Sample of the Restart menu

Coin generation works on the principle that there is a 1 in 5 chance that a coin will be generated on a newly created platform.

With the collected coins, the player can buy skins in the Shop for the character, platforms, or the background.



Sample of the Shop with character skins

The final important part is saving and loading the game. The game is automatically saved when the application is closed normally using the

red X button. It saves the player's highest score, the number of collected coins, and all purchased skins.
Note: If the player is in the middle of a game and has already surpassed their previous high score but closes the application during gameplay, the new high score will **not** be saved — high scores are saved **only upon death**.

```
public void saveScore(Menu menu) throws IOException {
    bw.write( str: "bestScore;"+menu.getScore().getBestScore());
    bw.newLine();
}
```

Sample method for saving the highest score

3. System requirements:

The project is developed in Java version 22, and a compatible Java version is required to run it. The program mainly uses the Swing library, which is, however, included in the JDK.
An Internet connection is not required to run the program.
The program is designed exclusively for desktop computers or laptops. It is not suitable for mobile devices or others.
The program can be run in any development environment that supports Java, or as a JAR file in the command line (cmd).
– (**java -jar project_name.jar**)

4. Basic structure:

The program is written using object-oriented principles and divided into several classes. The main class is Main, which launches the entire program and the main Frame (window). Here are other important classes:

1. **Menu:**
   This class functions as a JPanel containing two buttons that move the player either to the JPanel with the Shop or to the JPanel with the Game.

2. **Game:**
   The Game class also functions as a JPanel where the most important actions take place. Game contains an instance of the Player class, which represents the main player character. Additionally, the class includes a generator for generating platforms and a generator for generating coins.
3. **Player:**
   This class represents the main character in the game and inherits from the JLabel class. It primarily handles the player's movement by shifting other objects while keeping the player centered.
4. **Shop:**
   This class also functions as a JPanel. It contains three buttons for switching between smaller panels where the player can buy skins.

5. **Saver/Loader:**
   These classes handle saving and loading data. They create a new folder for the user at Users/home(HP)/SpaceParkour, where Saver saves the data whenever the program is closed, and Loader loads it when the program is started.

5. Test data:

The best way for the user to test the program is by playing it thoroughly and experimenting. The program is designed so that no errors should occur, but nevertheless, some issues may still arise.
The program also includes 5 UNIT tests, each consisting of individual methods:

1. **CollisionManagerTest:**
   This test checks the method isOnPlatform(Player, ArrayList<Platform>), which determines whether the player is touching any platform.
2. **GeneratorTest:**
   This test checks two methods, difficultyY() and difficultyX(). These methods set the distance between platforms based on the player's current score.
3. **LoaderTest:**

This test checks the method typeSelect(String type), which essentially converts the saved skin state (selected, owned, purchasable, expensive) from a String to an ENUM.

4. **PlayerTest:**
   This test checks the method died(ArrayList<Platform>), which determines whether the player is low enough for the game to end.
5. **ScoreTest:**
   The last test checks the method setBestScoreOnMenu(), which, upon the player's death, compares whether their current score is higher than their previous record.

6. User manual:

The controls of this game are very simple. The player moves using the keys A (to the right) and D (to the left). The player can also jump using the W key or SPACE. By holding down the SHIFT key, the player can sprint and run faster. The player can exit the Game, Shop, and Restart panels by pressing the ESC key.
Additionally, when the player collects the required number of coins, they can buy a skin of their choice in the Shop. Skins can be purchased for the Character, Platforms, and Background. The player can then mix and match individual skins. Purchased skins remain with the player forever.

7. Conclusion:

The programming process went relatively smoothly, but some difficulties did occur. My first problem was how to implement gravity. I solved this by adding a method to the main game loop that continuously pulls the player down whenever the player is neither jumping nor standing on a platform. Another issue was with the sound. At first, I used a Timer for the game loop, but it didn’t work properly together with the AudioInputStream and Clip classes. So, I created my own thread for the game loop, and everything worked again. The last major problem was saving and loading files so that it would also work with the .jar file.

Thanks to a presentation from Moodle and ChatGPT, I solved this problem, and now the game should run smoothly even when launched from the jar file.
On the other hand, I was surprised at how quickly I managed to create the methods for writing to and reading from files.