Střední průmyslová škola elektrotechnická Ječná

Informační a komunikační technologie

Ječná 517, 120 00 Nové Město

FINAL PROJECT 2025

Ondřej Líška

Informační a komunikační technologie

2025

Contents

1. Work goal..........................3

2. Game description...................3

2.1 Story/algorithm......................3

2.2 Mechanics............................3

3. System requirements.................4

4. Basic structure.....................5-7

5. Testing data........................7

6. User manual.........................8

7. Conclusion..........................8

8. Sources.............................8

1. Work goal

The goal of this project is to create a simple interactive game, allowing the player to interact with the environment in a comfortable way. The game's User Interface should be easy to understand and easily adjustable.

2. Game description

The game is supposed to be an interactive, sort of inventory and resource management, using in-game items to progress in the game, requiring player to search around the map in order to finish the game.

2.1 Story/algorithm

The player will awaken in a house, not knowing anything about how they got there, who they are, and how to get OUT of there. The player can move around the map quite freely, snooping around the map, attempting to find items, which might help them with the escape from this unknown place.

2.2 Mechanics

- Player movement around the map, map contains collisions, blocking and preventing player from accessing certain areas/moving out of bounds

- Sprinting ability - player can use the SHIFT buton to move around the map faster, as long as they have stamina. If they run out of stamina, they will have to wait a bit, before it recharges again

- Search spots and items - player can interact with search spots(through prompts) placed around the map, allowing him to search them and obtain the items hidden inside. Speaking of items, player can interact with items in following ways :

Using items - Player can use items to obtain benefits, for example, player can use bandages in order to regain some of their health

Dropping items - At any time, player can drop an item to the ground(unless the item is an essential item, like Knife), and being able to pick it up later

Picking items up - As mentioned, dropped items can be picked back up again.

- Hiding spots - at certain places, same with hiding spots, player can interact with hiding spots(through prompts), allowing him to hide from any potenational danger

- Saving - in Cassette Player locations around the map, player can use the item Cassette in order to save their progress into a new file, or overwritting a previously made save

- Pause menu - At most times of the game, player can access the pause menu using the ESC key to access the pause menu, which consists of the following options:

Continue - Simply closes the pause menu, returning back to the game

Load - opens the load menu(same as in the menu), allowing the player to load any of the previously made saves

Main menu - makes the player go back to the main menu(more below)

Exit - exits the program

- Main Menu - a menu, which shows up after running the program/entering it through the pause menu. Similarly to the pause menu, the main menu consists of the following buttons:

New Game - starts a new game from scratch

Continue from latest save - automatically loads the most recent save

Load - works the same way as the pause menu load, opens the load menu, showing all of the saved files, allowing the player to freely pick whichever one they want to load

Exit game - Exits the game

- In-game console - an amature version of the console, informing the player about important things and information

POTENTIONAL FUTURE IMPROVEMENTS

- Proper lock feature - originally, it was planned to make the items much more interactable, player having to use items to unlock rooms and search spots, in order to progress with the game. That is unfortunately not the case in the current version

- Enemies and combat- original console version also contained enemies and turn-based combat, which would be really nice to add in the future, to further increase the importance of item management and weapon usage

- Improved story and graphics

System requirements

To run my game, you will need to use the at time of creation version of IntellijIDEA(2024.2.3), or Eclipse/NetBeans with a similar date version.

to run this game, you will also need the following modules:

javafx-sdk-24.0.1

[DOWNLOAD LINK (https://download2.gluonhq.com/openjfx/23.0.1/openjfx-23.0.1_windows-x64_bin-sdk.zip)](https://download2.gluonhq.com/openjfx/23.0.1/openjfx-23.0.1_windows-x64_bin-sdk.zip)

you will also have to add these VM options for the javafx library:

```
--module-path "[path to your javafx lib file]" --add-modules javafx.controls,javafx.fxml,javafx.media,javafx.graphics
```

Mockito(for testing)

Junit jupiter(for testing)

gson(2.10.1 version or higher)

json

SDK configurement

OpenJDK 24.0.1 or higher for the game

OpenJDK 21.0.1 for testing

Commander runtime components - for .JAR file

Base Structure

the game uses objective programming, many of the classes had remained the same from the original game, but here's the list of the most important classes both GUI and console wise:

- Player.java : Class used to implement the user/player, his fields, values and behavior, choices

```
public class Player extends Character implements Serializable {
}
```

- GameGUI.java : Class used to set, create, design and configure major part of the game's features and their behavior

```
public class GameGUI extends Application {


}
```

Important methods:

```
    /**
     * Method used to load all the individual room from RoomManager based on their names
     * @param roomManager Which RoomManager instance to load the rooms from
     */
    private void loadRooms(RoomManager roomManager) {
        try {
            if (loadedState != null) {
                roomManager.loadRoom(loadedState.getCurrentRoomName().toLowerCase());
            }
            roomManager.loadRoom("enter_hall");
            ...
             } catch (Exception e) {
            e.printStackTrace();
            System.err.println("Failed to start game: " + e.getMessage());
        }
    }
```

```
/**
     * Method used to regularly update the game's attributes, based on player's current location and situation
     */
    private void update() {
        if (hidingSpotManager.isHiding()) {
            activeSearchSpots.clear();
            currentNearbyItem = null;
            return;
        }
        ...
    }
```

```
/**
     * Method used to handle the player's movement, calculating the speed of his movement
     */
    private void handlePlayerMovement() {
        if (player.isTransitioning() || !player.isMovementEnabled()) return;
    }
```

- *InventoryGUI.java: Class used to create, implement, visualize and set the inventory's behavior*

```
public class InventoryGUI extends StackPane implements Inventory.InventoryObserver {
}
```

Important methods:

```
/**
     * Method used to dropping an item, removing it from the inventory, etc.
     * @param item which item is the player attempting to drop
     */
    private void handleDropItem(Item item) {
        World world = gameGUI.getWorld();
    ...
    }
```

```
 /**
     * Method used to toggle the inventory's visibility
     */
    public void toggle() {
        boolean newVisibility = !isInventoryVisible();
        setInventoryVisible(newVisibility);
        if (newVisibility) {
            ...
        }
    }
```

- *GameStateGUI.java: Class used to store the game's values, used for serialization*

```
public class GameStateGUI implements Serializable {
}
```

- *Inventory.java:Class used for the implementation of inventory*

```
public class Inventory implements Serializable {
}
```

Important methods:

```
/**
     * Method to add items into the inventory
     *
     * @param item to add into the inventory
     * @return true/false based on whether the item can and had been added
     */
    public boolean addItem(Item item) {
        if (items.size() < capacity) {
            items.add(item);
            notifyObservers();
            return true;
        }
        return false;
    }
```

```
/**
     * Method used to locate a certain item by name
     *
     * @param itemName name of the item we are looking for
     * @return the item, if it's found in the inventory, null if not
     */
    public Item findItem(String itemName) {
        for (Item item : items) {
            if (item.getName().equalsIgnoreCase(itemName)) {
                return item;
            }
        }
        return null;
    }
```

Testing data

There is no reason to set any specific values to atributes, here's the list of recommended functions to test:

- W A S D movement - ensure player movement works, and also doesn't allow the player to walk through walls/out of bounds

- SHIFT sprinting - ensure stamina dynamically drains and regenerates when not sprinting, and the change correctly displays on the stamina bar

- Searching search spots - attempt interacting and searching a searchspot, ensure the notification about obtained items appear, the items are correctly added into the inventory,etc.

- Inventory - ensure all items are being displayed correctly, all buttons for interacting with the items work(Use, Info, Drop,...), also try to pick the items back up, ensuring the item is added into the inventory

-Room transition and prompts - Try going from room to room, to see if the correctly changes rooms and spawns at the entrance, not in a wall. Also try to interact with hiding spots,etc. to try out functionality of prompts

- Saving and loading - Using Cassette, try saving, and then loading the save, to see if the changes apply

- Test classes - If you want to further test the code automatically, you can use the classes in test root 'testGUI'(with JDK 21.0.1 set-up)

User manual

The game is fully controlled through mouse and keyboard inputs, but there is no instance, where you would have to interact with the console/type something, the game plays like a basic tile-set game. In case you're not sure about what you can do in the game, you can go back to the point 2.2. If you're curious about specific keys used in the game, you should go check the Controls menu in the game. The game doesn't end until you leave the house.

Conclusion

Having spent around 100 hours on this project since its creation back in February as a school mandatory console project, i am not really satisfied with the current GUI implementation of the project. Due to lack of time on my time, i had not been able to add so many features, that i think would really enhance the user experience while playing my 'game', some of which were mentioned in game mechanics part. One thing i'm quite satisfied with is the map. It may not be so pretty looking, but it had given me the opportunity to learn working with a new program, Tiled. All of the maps had been created by me, tilesets and graphic objects had been downloaded from itch.io (https://itch.io/). Overall, this project left me with mixed feelings about my performance, especially compared to my CookieClicker project, which i'd argue had been more worked through, despite only having half the time.

Sources

- StackOverflow (https://stackoverflow.com/questions) forums

- DeepSeek (https://chat.deepseek.com/)(only used for explanations of yet to me unfamiliar classes/libraries)

- personal knowledge