Developed with passion and love by:

Andrei Martins
Ivaylo Gatev
Joshua Kornfeld
Micah D
Rami Bader
Sevde Yanik

Setting and Story:

In a nice and beautiful town called Kratzerburg, where the apples are always red and the grass is always green, a young adventurer called Adlez searches for the meaning of life.

His first acquaintance is Brutus, the giant but gentle mayor of the town. He opens up about the challenges the place is facing and asks Adlez for help. Brutus sends the player to complete specific quests for each resident, starting with Olga. Unfortunately monsters are attacking the town`s beach, not allowing Olga`s kid to play there. She sends Adlez on an amazing quest to find the old sword belonging to her grandfather. By giving Adlez the key to the chest, the player equips the sword and defeats all the monsters living on the beach. As a proof of this arduous task, he collects their bones and brings them to Olga.

The next quest is helping Steve, a brilliant Mathematic Professor who has a pathological fear of pumpkins. He asks our hero to help him get rid of a terrible pumpkin next to his house. The second quest proves to be even more challenging since now Adlez has to eat poor defenseless food, but he perseveres and finds that he enjoys the perverseness of taking the life of a pumpkin so young.

The third quest is helping Bob, a solitary resident who has lost his cute buddies, a dog called Dog and a cat named Cat. Again Adlez must face dangers worse than his darkest fears, find the 2 animals and bring them back to their poor owner, who is looking suspiciously hungry.

Finally, when all the residents are safe from danger, Brutus gives Adlez one final adventure: kill the Monster King who is destroying the town with his mighty power. The name of this terrible monster, you ask? Darth Avaj. Legend says he was once a Java programmer who went mad because of one too many null pointer exceptions. The sword might not be enough to destroy this powerful monster, so Brutus gives Adlez a bow made from the trees of the Amazon Forest (before it was completely burned down). Doubts start to form in the player's mind, Am I really on the right side here? Those monsters at the beach looked an awful lot like sea urchins. But he puts them aside and focuses on the task at hand: maximum damage.

After hunting it down in its home nearby, using his bow and its arrows, Adlez is finally able to kill Darth Avaj, just like Brutus has always wanted.

World Map and UI:

Map Elements:

The map consists of multiple layers such as:

- Bottom Layer: represents the grass, beach and bridge which Adlez walks over.
- Houses Layer: represents the houses in Kratzerburg.
- Furniture Layer: for the table and chairs, mostly for decoration.

- Pumpkin Layer: has a pumpkin object, which Adlez picks in order to complete a quest.
- Lava Layer: animated lava layer, Adlez gets hurt when he touches the lava and makes the Darth Avaj fight trickier.
- Chest Layer: has a chest object, which Adlez finds a beautiful sword from and uses in order to kill the monsters.
- Bow & Arrow Layer: consists of a bow & arrow object. Adlez uses the bow and arrow to be able to kill Darth Avaj.
- Characters Layer: represents the characters in the game such as NPCs (Brutus, Bob, Olga, Steve, Cat, Dog), Monsters, Adlez and Darth Avaj.
- Sword and Swing Layer: consists of a powerful sword and its swing, Adlez uses the sword to defeat the monsters and destroy the rocks to reach Darth Avaj.
- Water Layer: represents the water in the map, decoration purposes mostly.
- Rocks Layer: the layer has a rock wall surrounding the lava.
- Destroyable Rocks Layer: the layer has a rock object in it. This is a quest that opens only after Adlez completes all the previous quests given by the fellow villagers. Adlez destroys the rocks with his sword and the way to Darth Avaj is now clear.

Objects in the map:

- Housebase.
- Pumpkin.
- Chest.
- Bow.
- Arrow.
- Monsters.
- Lava.
- Swing.
- Sword.
- Walls.
- Npcs: Brutus, Bob, Olga, Steve, Dog and Cat.
- Treebase.
- Player: Adlez.
- Boss: Darth Avaj.
- Water.

- DestroyableRocks.

UI Elements:

- Start Screen.
- Health bar.
- Inventory.
- Help text for instructions.
- Pause Screen(Esc key).
- Lose Game Screen (Restart and Quit).
- Win Game Screen.

Gameplay:

The gameplay is very similar to The Legend of Zelda games from a top down perspective. The player controls the character using the keyboard “WASD”. “Spacebar” is used to attack and “E” to interact with NPCs and objects. Once the bow is acquired, the player can switch between the sword and bow by pressing 1 or 2. “Esc” leads to the Pause menu and “Q” quits the game.

Algorithms:

Rendering the Map:

The map is built in Tiled and saved as a TMX file. There are many different layers for NPCs, objects, and the environment itself (see Map Elements). There are three main types of objects defined in the tiled TMX map, layers containing a list of integers, object layers containing named and selected regions or shapes, and tile sets that provide the file path and meta data to actually draw from. Each tile in a layer references a global id in a particular tile set, each tile set is assigned a range of global ids. Tiles within a layer can also have information about orientation. Specifically, there are three flags: horizontally flipped, diagonally flipped, and vertically flipped that are represented by the first three bits on the tile number. Once those orientation bits are stripped away, the tile is a gid that can be used to look up a specific section of an image from a tile set.
Tiles are mapped to ImageRefTo's, an extension of an ImageRef (which only specifies a source image and bounds) to include a destination bounds (starting/ending x's and y's)

and orientation. Lists of these ImageRefTo's are combined with a name into a "VirtualImage" class and are drawn into buffered images that can then themselves be drawn from. This is also done for the foreground and background, with the exception that their bounds aren't defined by object layers, but by the screen itself.
Image URIs are provided to the graphics layer through an interface implemented by the map, which is passed in on creation. Once the graphics layer has all of the images loaded into memory, it caches them in a hash map by either, path, if they are a directly loaded image, or by name, if they are a virtual image. Once we have these images loaded in the graphics layer, the background is drawn on every clear, and the foreground is drawn after all game objects have been drawn.

Collision Detection:

The “GameObject” and its children are responsible for the collision mechanics. There are two main types of GameObjects: RectangularGameObjects and CircularGameObjects. Both types of objects must support collision with both its type and the other type.

The “Avatar” class calculates all the collisions between the player and the other elements, such as objects, NPCs and enemies. The collision is not the same for all elements. For objects such as trees, rocks and water, the player simply can't go forward. Enemies and lava will draw player life while NPCs will trigger dialogs. Specific objects such as a chest or a pumpkin will disappear from the map when interacted with during certain quests. A Switch statement is used to check with which object the player is colliding and what is the appropriate response.

An example of collision with the bones left by the enemies. Each bone will heal a portion of the player`s life and disappear from the world (isLiving=false).

```
case BONES:
        hit(-0.25); // Heal some
        counterBones++;
        ((RPGWorld)world).addNotification("Bones picked up");
        obj.isLiving = false;
        break;
```

Dialog / Quest System:

The dialogs are stored as an array of Strings. Each NPC of the game has its own class with its own dialogs stored. Most of the dialogs are triggered only after the player made some progress in the story. For example, at the beginning of the game, with the exception of Brutus, all other NPCs will give the "ignore" dialog until the player talks to Brutus, the NPC that starts the story. Different dialogs are triggered according to how far the player has progressed in a quest. The boolean methods hasQuestProgress(QuestState q) and makeQuestProgress(QuestState q) handle these mechanics. The quest state itself is represented by an enum. The method getNpcQuestText(QuestState q) in each NPC returns the appropriate quest dialog for each character. This is done by a Switch statement that checks which is the state of the quest. Below is an example of the handling of the dialogs for the character "Olga".

```
public String[] getNpcQuestText(QuestState q) {
    switch(q) {
        case START:
            return olgaNpcQuestWaiting;
        case OLGA:
            return olgaNpcQuestStartText;
        case OLGA_SWORD_SEARCH:
            return olgaNpcChestQuestInProgText;
        case OLGA_SWORD_SEARCH_COMPLETED:
            return olgaNpcChestQuestCompleteText;
        case OLGA_MONSTERS:
            return olgaNpcMonsterQuestStartText;
        case OLGA_COMPLETED:
            return olgaNpcMonsterQuestCompletedText;
```

Nevertheless, the dialog /quest system works closely with the "Game State" system, which will be explained further.

Sound System:

The sound system is responsible for managing the background music and all other sounds in the game. There is in Java already a library which helps loading sounds from "wav" files called javax.sound.

A class called "Sound" imports this library and saves the audio from the files in a "Clip" object (which comes from the java standard sound library). With the audio saved in this file, the music can be manipulated at will. There are 3 methods for the sound: playBackgroundMusic(), playsound() for the sword sound and setVolume(), which lowers the music when the game is paused.

An example of the code responsible for playing the music and looping during the game:

```
public void playBackgroundMusic()  {
    clip.start();
    clip.loop(Clip.LOOP_CONTINUOUSLY);
 }
```

Game States:

The game state sets the overall state of the world and how the gameplay changes accordingly. There are 6 game states, represented by an enum of: MAIN_MENU (the initial screen), PLAY (the main play mode where the character can be controlled), PAUSE (the pause menu), DIALOG (where the NPCs dialogs and the quests are displayed. This is heavily connected with the dialog /quest mechanics), DEATH (showing the death screen and the option to replay or quit the game) and COMPLETE (the endgame screen, when the player defeats the final boss). The game states are heavily used throughout the code. Below is an example of the drawing of the buttons showing on the screen depending on the state of class "World".

```
// draw play button of main_menu
        if (gameState == GameState.MAIN_MENU) {
           for (int i = 0; i < mainMenuObjects.size(); i++) {
              mainMenuObjects.get(i).draw(graphicSystem, currentTick);
```

```
    }
}
if (gameState == GameState.COMPLETE) {
    for (int i = 0; i < completeGameMenuObjects.size(); i++) {
        completeGameMenuObjects.get(i).draw(graphicSystem, currentTick);
    }
}

// draw play button of death_menu
if (gameState == GameState.DEATH) {
    for (int i = 0; i < deathMenuObjects.size(); i++) {
        deathMenuObjects.get(i).draw(graphicSystem, currentTick);
    }
}
```