World of Deez nuts

Pawel Makles

(K21002534)

Created: 19th November 2021
Last Modified: 3rd December 2021

1 User Level Description

This game is largely open world but has a main storyline which the player can choose to follow to complete the game, as well as other side-quests which can be done alongside or after the story.

In this game, you are one of millions of citizens in an isolated city home to a Beastman[1] society, the city being loosely based off Animacity[2] from the series Brand New Animal.[3] The plot revolves Sylvasta who are deeply intertwined with the city and own the city's medical centre.

There have been reports that their research has been highly unethical and leaks have been coming out from former employees which have been riling up protests throughout the city, your goal is to figure out what's really happening and to put a stop to it.

I ended up not adding any information about the main character as to add to the open world, 'your adventure', feeling. The story is also loosely based off Utopia,[4] in which a group of people are finding the truth behind a manuscript and find themselves in the middle of a conspiracy.[9]

2 Implementation

The project is split up into several packages, most of which work independently of each other, in general, the `Game` uses the `commands` package (providing actions the player can perform) along with the `world` (providing the things that the player can interact with in the game) package to run the game. They are laid out as follows:

2.1 Commands

This package provides the classes required to parse and run commands, it includes the `CommandManager` which registers objects of type `Command`. More about parsing commands is in the challenge task section.

This package also includes a subpackage called `core` containing all of the 'core' commands required to run any game world, this includes things such as picking items up, quiting the game, and so forth.

2.2 Entities

This package provides the basic tools required to create simple and complex entities within the world, more about entity detail is discussed later.

It also contains a subpackage `actions` which contains common interfaces which entities can implement to allow the ways that they can be interacted with to be quite modular.

2.3 Content / Campaign

The `content.campaign` package contains a lot of custom content derived from the base game used to construct the story and the story world.

2.4 World

This package has the basic building blocks for creating worlds including the `World` and `Room` classes which can be easily extended for more functionality. The only limitations are that traveling between rooms must be done in a direction specified in the `Direction` enum and that the `Location` of any entity must be either a room, inventory or neither.

2.5 Util

This package contains a bunch of small utility classes, including:

- **BlueJ.java**:[6] Contains improved methods on detecting whether the application is currently running in BlueJ. (taken from my previous coursework assignment)

- **Localisation.java**: A pretty straightforward implementation of a localisation engine. Simply maps (period-separated) keys to their respective (nested) keys in a language file.

- **Search.java**: Methods for searching through various data structures in the game.

- **Tree.java**: A very simple implementation of a tree with basic traversal methods.

2.6 Dialogue & IO & UI & Events

These are all discussed later on in the challenge task section.

3 Base Tasks

This section describes how I implemented the basic task requirements.

- *"The game has several locations the player can walk through."*

  I began by first designing the world map (see Figure 1), using Excalidraw,[5] then implemented each location as a `Room`.

  To build the world, I made a `World` class to house all the locations which exist in the game world and then extended this using the `CampaignWorld` class which builds the story world, creates rooms and registers events.

  I ended up adding 10 locations into my game:

  - **City Centre**: Centre of the city connecting major areas with the coast.
  - **Apartments**: This is the player's residence.
  - **Street**: This is the main city street connecting important buildings.
  - **Shop**: The local city shop where the player frequents to get necessary items.
  - **Back Alley**: This is where the player can start the main story mission.
  - **Coastline**: There are two coasts, one on the city side and one on the mainland.
  - **Forest**: The forest grants access to the Worm Hole and to other side-quests.
  - **Worm Hole**: For challenge task 3.

  The layout is heavily inspired by Animacity, although since there's no available map of the actual city, I made my own interpretation based off various pieces of art, (see Figure 2). I used an existing real life location to determine the size of the river.[7]

- *"There are items in some rooms that may or may not be picked up by players."*

  To achieve this, I considered all entities to be items which may or may not be picked up by other entities, each entity has its own `Inventory` which is in effect a list of other entities which it is holding.

- *"Each item has a weight and the player can only carry items up to a certain weight."*

  To do this, I added a new private field `weight` of type double to the `Entity` class, which is used to store the entity's weight. It is a double as I wanted access to fractional weights (say 0.01kg) and I wanted to have access to `Stream::mapToDouble` for summations.

  For the second part, I made it so each `Inventory` has a maximum weight it can store, which by default is set to 0 as each entity

has an inventory but may not necessarily have the ability to store anything.

When putting anything in an inventory, we check that the following is satisfied:

$$\mathsf{currentWeight} + \mathsf{itemWeight} \leq \mathsf{maxWeight}$$

To determine the weight of common items, I referred to a document I found online published by the City of York Council.[8]

- *"Player can win."*

  The player may win by completing the main story mission (detailed in the walkthrough) which sets a flag that the game has been completed, the player may choose to keep playing in the open world or run *win* to end the game.

- *"There is a command *back* which takes you back to the last room."*

  I added two new private fields to the player entity, which were `previousRooms` and `retreatingDirection`, these are used to store the path back through the room and the direction which we need to go to get back there respectively. The direction is stored in order to run a check whether the player can actually go back in the direction they intend to, to verify this, the 'retreating direction' is used to call the method `canLeave` on the current room the player is in.

- *"Add at least four new commands."*

  I added several additional commands which are listed below:

  - *bag* : Allows the player to look at their or another entity's inventory.
  - *drop* : Drop any specified item from the player's inventory into current room.
  - *give* : Give a specified item to another entity, we ensure that the entity implements `IGiveable` and give the item to the entity using `IGiveable::give`.
  - *pet* : Pet a specified entity, has to implement `IPettable`, we use `IPettable::pet`.
  - *take* : Take any specified item from another entity's inventory.
  - *talk* : Initiate a conversation with an entity, must implement `ITalkwith`, we call `ITalkwith::talk` to start the conversation.
  - *use* : 'Use' an entity, we call `IUseable::use` to use the entity. (implemented by entity)
  - *where am i* : Tells the game to print out room information again, this is done by emitting `EventEntityEnteredRoom` again.

  See Figure 4 for an example of the help menu.

4   Challenge Tasks

This section describes how I implemented the challenge tasks and what I did in addition.

4.1   Required Tasks

- *"Add characters to your game."*

  I ended up adding 8 characters to my game, 3 of which can move on their own.

  - **Static NPCs**: There are several static NPCs which the player can interact with, talk with or complete missions for. Including the receptionist, city NPC, the old man, the security guard, the shopkeeper and Marie who is part of the main story.

  - **Dynamic NPCs**: There are 2 NPCs which appear or disappear depending on the state of the game, this includes the protestors and security guard.

  - **Moving NPCs**: There is a single NPC, the cat, which randomly follows a fixed path to move around the map. This is done by listening to `EventTick` from world events and then randomly deciding to move.

  Note on character creation: I decided to stick to mostly generic characters in the game, mainly since the people you encounter have not as much impact on the story, with the exception of Marie,[?] who I took straight from BNA,[3] see Figure 3, she is a sly and cunning character which I think perfectly fits the role I need.

- *"Extend the parser."*

  I entirely replaced how the parser worked, I began by implementing a basic model of `Command`, it took a few iterations but I settled on providing Regex `Pattern`s.

  This approach has several benefits:

  - Powerful Regex at low performance cost due to the low number of commands.

  - Ability to have named capture groups which are then interpreted as arguments.

  For each known `Command`, I would create a `Matcher` for each `Pattern`, execute it against the arbitrary command from the user and then pass it into a wrapper class `Arguments` which lets me safely pull out named groups, directions or any other argument type I need.

- *"Add a magic transporter room."*

  To do this, I added a `RoomWormHole` which I made implement `EventEntityEnteredRoom` to listen for when any entities entered the

room, as soon as one is detected, a short animation is played and a room is selected at random to teleport the user to. Allowing the user to go to **any room** may interfere with my story so I chose to only spawn the user at any outside areas of the map.

4.2   World Event System

"Add a system for managing world events."

I began by creating a basic `Event` class which can be extended to hold any arbitrary data, it also provides basic functionality to prevent event propagation (see **??**).

I created an `EventSystem` class which would manage incoming and outgoing events, I decided that it should have the ability to handle any event and allow anything to create listeners for any object that extends Event.

To map these, I had a `HashMap` which mapped unique `Class<? extends Event>` to linked (as to keep order of registration) hash sets of `IEventListener<? extends Event>`. The interface simply provides a single method, `onEvent(Event e)`, which is called when the event is triggered.

Since Java has type erasure, it means my generic type annotations are gone once the program is compiled, so I can't do things such as object is 'instanceof' class. As such, I decided to ignore cast warnings and entirely rely on compiler type checking to verify that the objects provided are correct, I achieved this by adding a generic type argument to the add listener method and making the compiler check that the target class and event listener match, see **??**.

4.3   Dialogue

"Give NPCs interactive dynamic dialogue."

To implement dialogue, I made a simple data structure implemented as follows:

```
Dialogue
├─ DialogueNode
│  ├─ DialogueOption
│  └─ DialogueOption
└─ DialogueNode
   └─ DialogueOption
```
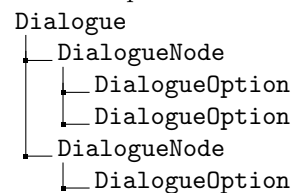
The `Dialogue` houses several nodes each of which have options which lead to other nodes. With this system, I can add dynamic conversations, such as with the shopkeeper, by extending the `DialogueNode` class and providing my own dynamically generated description and options.

For simpler tasks, such as changing game state through conversation, I can create a new `DialogueOption` with an `IDialogueHandler` which provides a method that the dialogue system calls to determine whether anything else needs to happen and what the next node is.

4.4 Terminal Emulator
"Implement a terminal emulator."

I began by creating a new `IOSystem` interface which provides common I/O methods, such as `print` , `println` and `readLine` . I then made all game output come through this interface and created two new classes that implemented it, `StandardIO` as a fallback to stdout and the `TerminalEmulator` which creates a new window.

To render the window I have a class extending `JFrame` which shows my `JTerminalView` . To render the text I override into `paintComponent` and draw my own content using the provided `Graphics` context.[?] (example: **??**)

The font used is VT323 Regular.[?]

4.4.1 Ansi Escape Codes
One of the main reasons I built this was for greater flexibility, so naturally I added support for Ansi escape codes. I matched the escape codes in the `TextBuffer` and adjusted the buffer accordingly. (**??** for the implementation)

4.4.2 Emoji Support
I also added a way to load and display small images like emojis (see **??**), although I had some difficulties with Unicode being mangled when working with BlueJ.[?]

4.4.3 EventDraw and the Map
Since I already had a way to load images, I also provided a way for anything to draw to the terminal. I added an `EventDraw` (**??**) which is emitted whenever the terminal view paints and used it to draw a map (see **??**).

5 Code Quality
5.1 Coupling
An example of loose coupling in my project is the `EventSystem` , it is completely independent from the rest of the project and can be used with anything (see **??**), as such I am using it with both the `World` and `TerminalEmulator` for different purposes to provide different sets of events. Another example of loose coupling is the `Dialogue<T>` system, it accepts any kind of key through the generic `T` so it can be used with practically anything and if needed, the type can be constricted (example: **??**). The only dependency it has is the `IOSystem` interface.

5.2 Cohesion
One example of high cohesion is the `IOSystem` interface (see **??**) and the classes which implement it. It is used throughtout the project to refer to an arbitrary input or output, as such, the first implementation I made was StandardIO (see **??**) which passes these methods through to `System.out/in` . One benefit is that I can easily slot in additional processing layers, such as `LocalisedIO` (see **??**) which can take in data, process it and pass it through to an arbitrary `IOSystem` .

5.3 Responsibility-driven design
I considered responsibility-driven design when first building the structure of my project, for example, the `World` class is responsible entirely for keeping track of rooms and entities and nothing else, to add functionality to it you must extend it as I have done with the `CampaignWorld` . As another example, each `Entity` just stores any information directly relating to the entity such as a name and its location, to implement additional actions such as 'using the entity', we must extend `Entity` and then implement `IUseable` .

5.4 Maintainability
One example of where I considered maintainability is the `Inventory` system, the logic is entirely self-contained and quite simple to understand. Fields such as `items` , `maxWeight` are both private and inaccessible outside of the inventory class, as such I can guarantee that the inventory always stays consistent and hence easily maintainable, these values may only be modified by methods with distinct tasks and appropriate validity checks (examples: **??**).

6 Walkthrough
This guide will skip over some parts of the plot, it serves as a way to just get to the end, I would recommend playing through first and then falling back on this if you are stuck.

You wake up in your home, *go down* , then *east* then *north* . Here, you must *talk to Marie* , go through the prompts $(1 \rightarrow 1 \rightarrow 2 \rightarrow 1)$ to progress the story.

Now, *go south* , *north west* , *north* and *talk with shopkeeper* , buy the communicator device and leave. Now *go south* , *east* , *north* and *talk to Marie* again. Continue by going *south* , *north west* , *west* . Sit down by *use couch* then *use comms* and go through the prompts, afterwards *go down* . Here, you must pick up documents 2, 4 and 5 by using *take doc2* , ...

Now, *go up* , *east* , *south* , *up* to get back to your home and *use the laptop* . Go through the prompts $(1 \rightarrow 4 \rightarrow 2)$. At this point, you can use the command *win* . If you want to do the side-quest, you'll have to figure it out yourself :-)

6.1 Access to teleporter room
To get to the teleporter room, you must go to the shopkeeper and buy the 'speed boat key', from there you can go to the south of the city and use the boat to get to the mainland. Going south to the forest then east to get to the teleporter room.

7 Known Issues
- Anything starting with 'b' will show inventory.

- NPC dialogue not changing between chapters.

- Can't include Unicode in Java source.

References

[1] Brand New Animal Wiki. Beastman
`https://brand-new-animal.fandom.com/wiki/Beastman`

[2] Brand New Animal Wiki. Animacity
`https://brand-new-animal.fandom.com/wiki/Anima_City`

[3] Brand New Animal Wiki. Marie Itami
`https://brand-new-animal.fandom.com/wiki/Marie_Itami`

[4] IMDb. BNA (TV Mini Series 2020)
`https://www.imdb.com/title/tt12013558/`

[5] IMDb. Utopia (TV Series 2013-2014)
`https://www.imdb.com/title/tt2384811/`

[6] Excalidraw.
`https://excalidraw.com/`

[7] GitHub. maven-bluej / BlueJ.java
`https://github.com/KCLOSS/maven-bluej/blob/master/BlueJ.java`

[8] Google Maps. Jezioro Świerklaniec, Poland
`https://www.google.com/maps/@50.4293559,18.9742453,16.12z`

[9] PDF. Set of average weights for furniture, appliances and other items `https://democracy.york.gov.uk/documents/s2116/Annex%20C%20REcycling%20Report%20frnweights2005.pdf`

[10] YouTube. The best (and worst) show you haven't seen
`https://youtu.be/PFx2QMOZ8Qo`

[11] GitHub. BlueJ Bug Reproductions (wip)
`https://github.com/insertish/bluej-bug-demo`

[12] Fontsource. VT323
`https://fontsource.org/fonts/vt323`

References in code.

[13] `Ansi.java` StackOverflow. How to print color in console using System.out.println?
`https://stackoverflow.com/a/5762502`

[14] `LocalisedIO.java` StackOverflow. What is the equivalent of Regex-replace-with-function-evaluation in Java 7? `https://stackoverflow.com/a/27359491`

[15] `JTerminalFrame.java` How to detect a key press in Java
`https://stackoverflow.com/a/21970006`

[16] `JTerminalView.java` How to "do something" on Swing component resizing?
`https://stackoverflow.com/a/8917978`

[17] `JTerminalView.java` Java Documentation. Font Concepts
`https://docs.oracle.com/javase/tutorial/2d/text/fontconcepts.html`

[18] `JTerminalView.java` Java Documentation. Java Thread Primitive Deprecation `https://docs.oracle.com/javase/1.5.0/docs/guide/misc/threadPrimitiveDeprecation.html`

[19] `JTerminalView.java` StackOverflow. Drawing Canvas on JFrame
`https://stackoverflow.com/a/17922749`

[20] `TerminalEmulator.java` StackOverflow. Passing values between 2 threads without intrrrupting each other `https://stackoverflow.com/a/23413506`
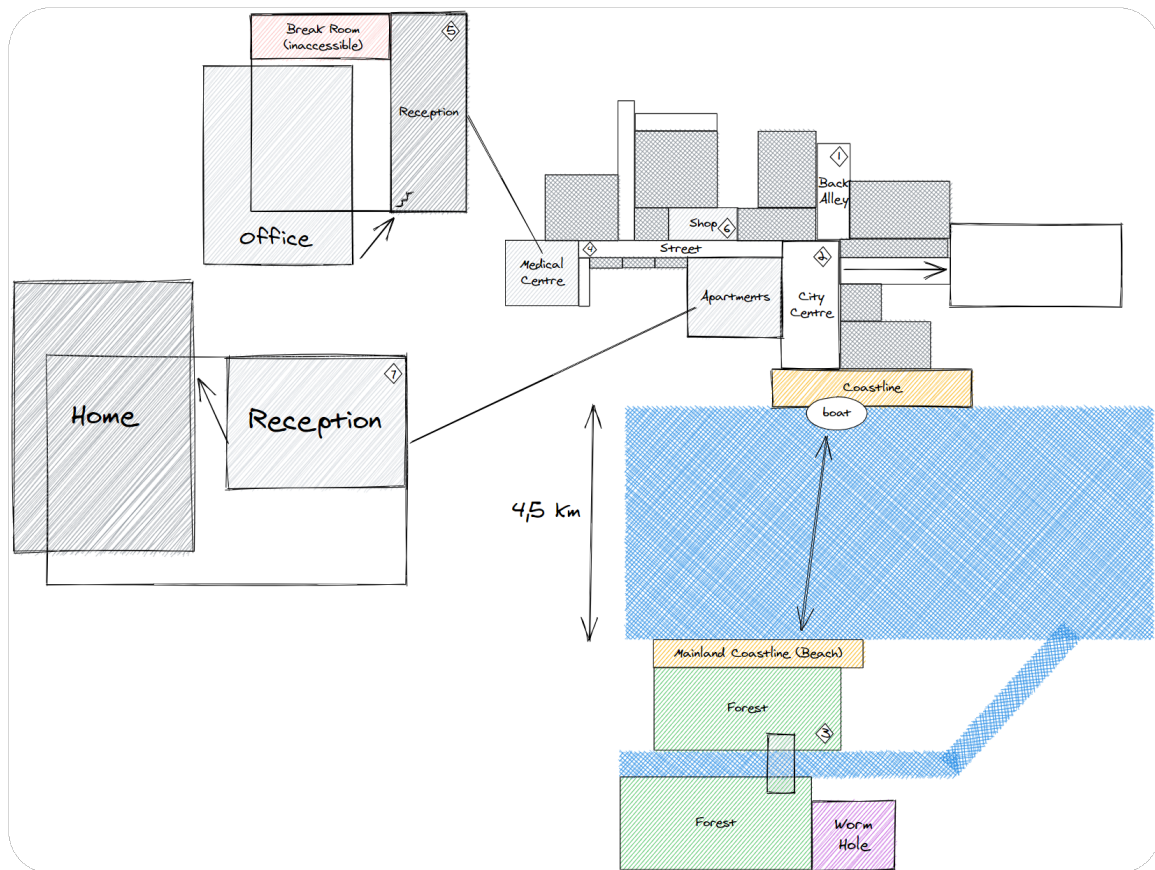
Figure 1: World Map

Figure 2: Shot of Animacity as seen in Episode 1 at 3:02 of
Brand New Animal[3]

Figure 3: Michiru Kagemori and Marie Itami pictured left
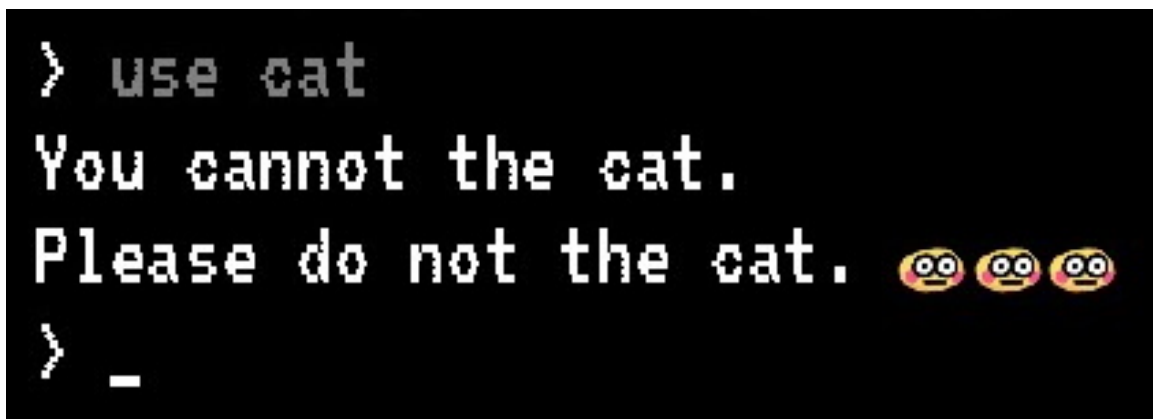to right in Episode 1 at 12:51 of Brand New Animal[3]
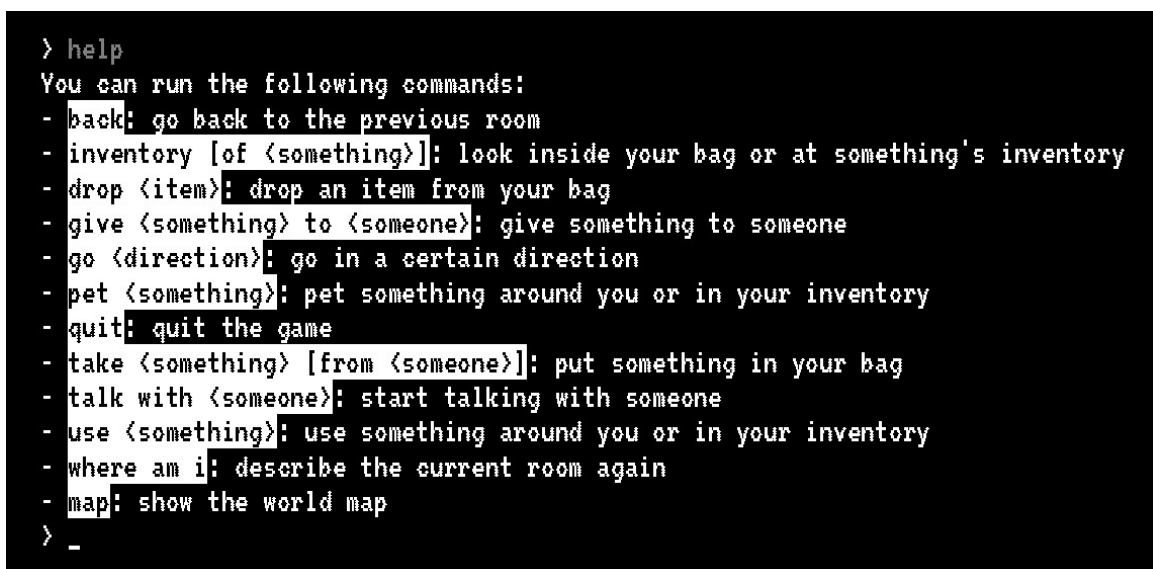

Figure 4: Sample emoji output
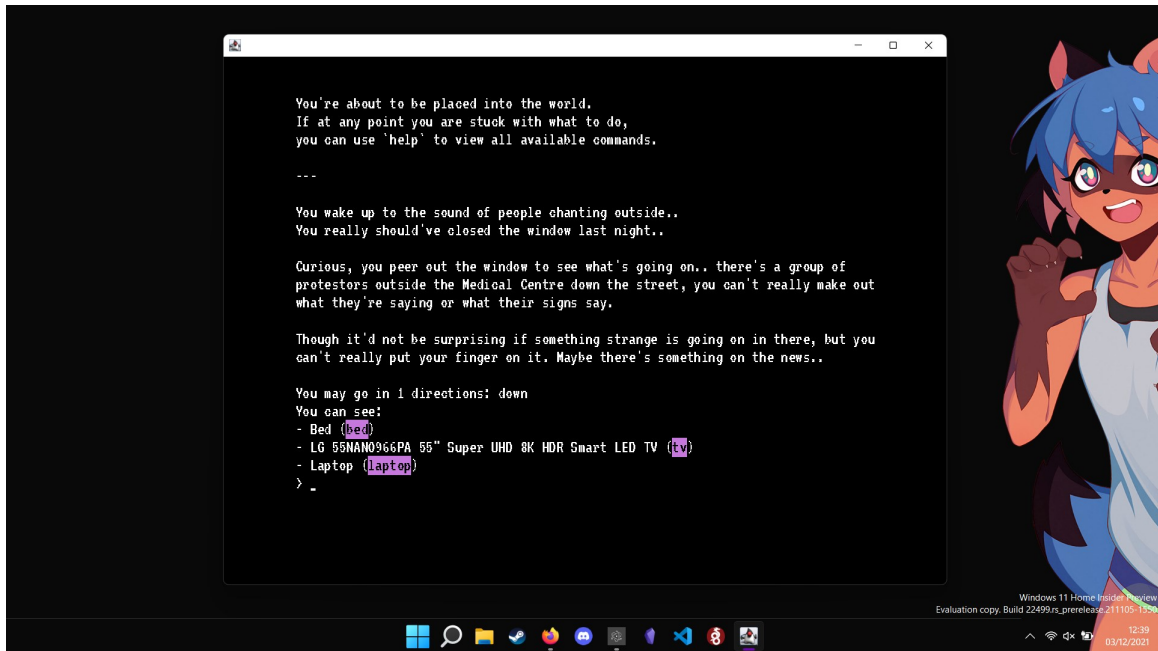

Figure 5: Output from the help command

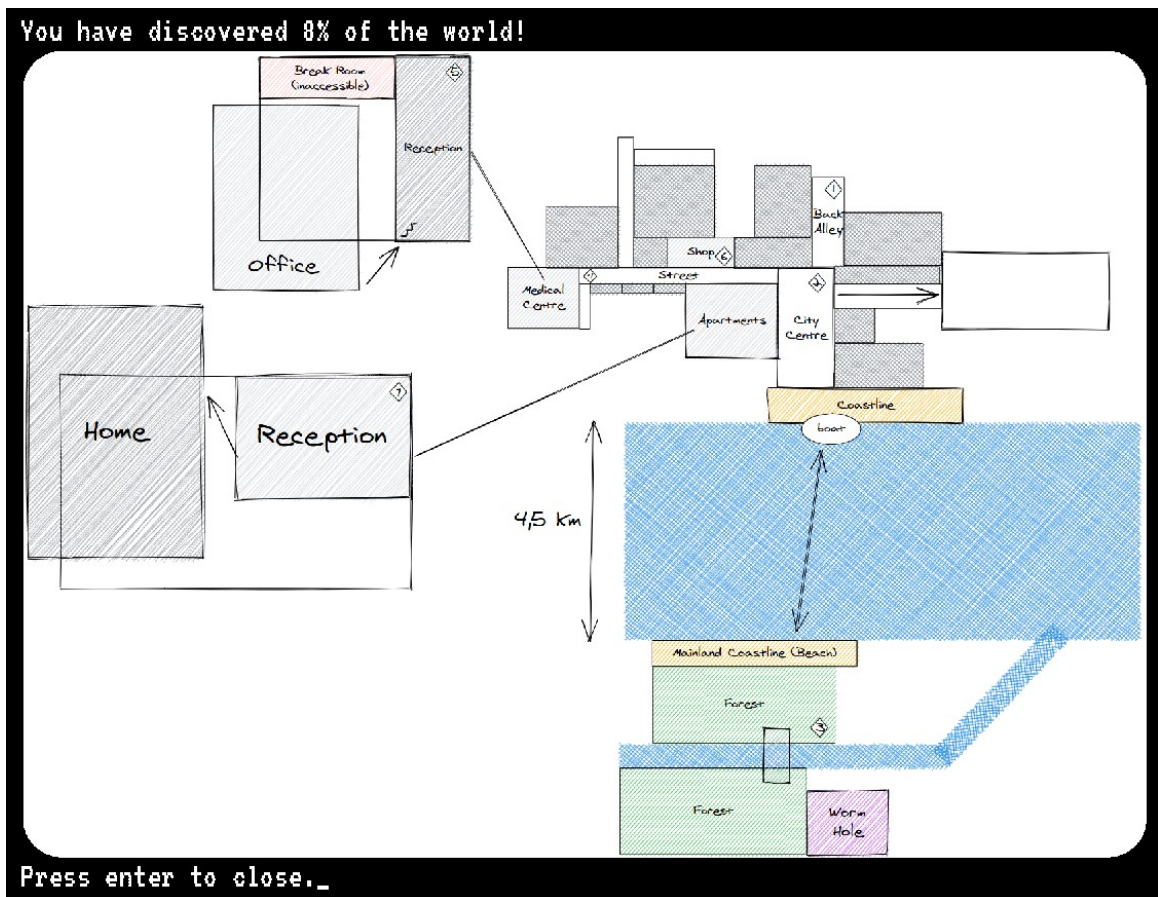Figure 6: The terminal emulator

Figure 7: Output from the map command

```java
1  package uk.insrt.coursework.zuul.events;
2
3  /**
4   * Represents a single event fired from
5   * any source to be consumed by anything.
6   */
7  public class Event {
8      private boolean propagating = true;
9
10     /**
11      * Whether this event can continue running.
12      * @return Whether propogation of this event was stopped
13      */
14     public boolean canRun() {
15         return this.propagating;
16     }
17
18     /**
19      * Stop further propagation of this event.
20      */
21     public void stopPropagation() {
22         this.propagating = false;
23     }
24 }
```

Figure 8: Event.java
(uk.insrt.coursework.zuul.events.Event)

```java
1  package uk.insrt.coursework.zuul.io;
2
3  /**
4   * Interface representing an arbitrary IO system.
5   * This can be implemented to input or output from various interfaces.
6   */
7  public interface IOSystem {
8      /**
9       * Print a string out through an arbitrary output channel.
10      * @param out String to print
11      */
12     public void print(String out);
13
14     /**
15      * Print a string out through an arbitrary output channel and append {@code \n}.
16      * @param out String to print
17      */
18     public void println(String out);
19
20     /**
21      * Read a String up until the first encountered {@code \n} from an arbitrary
      input channel.
22      * @return String of line read in
23      */
24     public String readLine();
25
26     /**
27      * Dispose of the arbitrary input and output channels.
28      */
29     public void dispose();
30 }
```

Figure 9: IOSystem.java
(uk.insrt.coursework.zuul.io.IOSystem)

```java
package uk.insrt.coursework.zuul.io;

import java.util.Scanner;

/**
 * A simple IO system implementation which feeds
 * into System.out and takes data from System.in
 */
public class StandardIO implements IOSystem {
    private Scanner reader;

    /**
     * Construct a new StandardIO.
     */
    public StandardIO() {
        this.reader = new Scanner(System.in);
    }

    @Override
    public void print(String out) {
        System.out.print(out);
    }

    @Override
    public void println(String out) {
        System.out.println(out);
    }

    @Override
    public String readLine() {
        return this.reader.nextLine();
    }

    @Override
    public void dispose() {}
}
```

Figure 10: StandardIO.java
(uk.insrt.coursework.zuul.io.StandardIO)

```java
package uk.insrt.coursework.zuul.io;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

import uk.insrt.coursework.zuul.util.Localisation;

/**
 * Translate and localise any incoming output.
 */
public class LocalisedIO implements IOSystem {
    private final Pattern pattern = Pattern.compile("<([\\w\\.]+?)>");

    private IOSystem io;
    private Localisation locale;

    /**
     * Construct a new LocalisedIO.
     * @param io Provided IO system we should feed into
     * @param locale Locale to apply to any i18n strings
     */
    public LocalisedIO(IOSystem io, Localisation locale) {
        this.io = io;
        this.locale = locale;
    }

    /**
     * Replace i18n strings in any given String with their actual localised values.
     * Using replacement code from https://stackoverflow.com/a/27359491.
     * @param input String to process
     * @return Final processed string
     */
    private String replace(String input) {
        StringBuffer result = new StringBuffer();
        Matcher matcher = this.pattern.matcher(input);

        while (matcher.find()) {
            matcher.appendReplacement(result, this.locale.get(matcher.group(1)));
        }

        matcher.appendTail(result);
        return result.toString();
    }

    @Override
    public void print(String out) {
        this.io.print(this.replace(out));
    }

    @Override
    public void println(String out) {
        this.io.println(this.replace(out));
    }

    @Override
    public String readLine() {
        return this.io.readLine();
    }

    @Override
    public void dispose() {
        this.io.dispose();
    }
}
```

Figure 11: LocalisedIO.java
(uk.insrt.coursework.zuul.io.LocalisedIO)

```java
package uk.insrt.coursework.zuul.events;

import java.util.HashMap;
import java.util.HashSet;
import java.util.LinkedHashSet;

/**
 * Event system which manages taking in events
 * from different sources and handles them
 * by firing callbacks on event listeners.
 */
public class EventSystem {
    private HashMap<Class<? extends Event>, LinkedHashSet<IEventListener<? extends
    Event>>> listeners = new HashMap<>();

    /**
     * Get existing Event listener list or create a new one if not exists.
     * @param event Event
     * @return Set of event listeners
     */
    private HashSet<IEventListener<? extends Event>> getList(Class<? extends Event>
    event) {
        var list = this.listeners.get(event);
        if (list == null) {
            list = new LinkedHashSet<>();
            this.listeners.put(event, list);
        }

        return list;
    }

    /**
     * Add a new event listener to this system.
     * @param <E> Generic Event type
     * @param event Event to remove from
     * @param listener Event listener callback
     */
    public<E extends Event> void addListener(Class<E> event, IEventListener<E>
    listener) {
        this.getList(event).add(listener);
    }

    /**
     * Remove an new event listener from this system.
     * @param <E> Generic Event type
     * @param event Event to remove from
     * @param listener Event listener callback
     */
    public<E extends Event> void removeListener(Class<E> event, IEventListener<E>
    listener) {
        this.getList(event).remove(listener);
    }

    /**
     * Emit an Event.
     * @param <E> Generic Event type
     * @param event Event to emit
     */
    @SuppressWarnings("unchecked")
    public <E extends Event> void emit(E event) {
        var listeners = this.listeners.get(event.getClass());
        if (listeners == null) return;

        for (@SuppressWarnings("rawtypes") IEventListener listener : listeners) {
            listener.onEvent(event);
            // Previously, there was a try catch ClassCastException
            // but I've since constricted the types on `addListener`
            // and `removeListener` so this should never happen.

            if (!event.canRun())
                break;
        }
    }
}
```

Figure 12: EventSystem.java
(uk.insrt.coursework.zuul.events.EventSystem)

```java
package uk.insrt.coursework.zuul.dialogue;

import uk.insrt.coursework.zuul.io.IOSystem;

/**
 * An option which branches off a {@link DialogueNode} into another node.
 */
public class DialogueOption<T> {
    private IDialogueHandler<T> handler;

    private String description;
    private boolean shouldExit;
    private T target;

    /**
     * Construct a new simple DialogueOption with a description and destination.
     * @param description Description of this option
     * @param target Target node to jump to
     */
    public DialogueOption(String description, T target) {
        this.target = target;
        this.description = description;
    }

    /**
     * Construct a complex DialogueOption with a description and select handler.
     * @param description Description of this option
     * @param handler Method called when this option is selected
     */
    public DialogueOption(String description, IDialogueHandler<T> handler) {
        this.handler = handler;
        this.description = description;
    }

    /**
     * Tell the Dialogue system to exit if this option is selected.
     * @return This dialogue option so method calls can be chained
     */
    public DialogueOption<T> mustExit() {
        this.shouldExit = true;
        return this;
    }

    /**
     * Get the description of this option.
     * @return Description string
     */
    public String getDescription() {
        return this.description;
    }

    /**
     * Get the destination of this option.
     * @return Destination if it exists
     */
    public T getTarget() {
        return this.target;
    }

    /**
     * Handle the player selecting this dialogue option.
     * @param io Provided IO system
     * @return The new node or null if we should exit and stay put.
     */
    public T handle(IOSystem io) {
        if (this.handler != null) {
            return this.handler.onAction(io);
        } else if (!this.shouldExit) {
            return this.target;
        }

        return null;
    }
}
```

Figure 13: DialogueOption.java
(uk.insrt.coursework.zuul.dialogue.DialogueOption)
13

```java
package uk.insrt.coursework.zuul.ui;

import java.awt.Graphics;

import uk.insrt.coursework.zuul.events.Event;

/**
 * Event fired when the terminal emulator draws a new frame.
 */
public class EventDraw extends Event {
    private Graphics g;
    private float ox;
    private float oy;
    private float fw;
    private float fh;

    /**
     * Construct a new EventDraw Event.
     * @param g Graphics context
     * @param ox Origin X position
     * @param oy Origin Y position
     * @param fw Font character width
     * @param fh Font character height
     */
    public EventDraw(Graphics g, float ox, float oy, float fw, float fh) {
        this.g = g;
        this.ox = ox;
        this.oy = oy;
        this.fw = fw;
        this.fh = fh;
    }

    /**
     * Get the Graphics relating to this event.
     * @return Graphics
     */
    public Graphics getGraphics() {
        return this.g;
    }

    /**
     * Get the origin X position of the contents of the terminal.
     * @return X position
     */
    public float getOriginX() {
        return this.ox;
    }

    /**
     * Get the origin Y position of the contents of the terminal.
     * @return Y position
     */
    public float getOriginY() {
        return this.oy;
    }

    /**
     * Get the character width.
     * @return Character width
     */
    public float getCharWidth() {
        return this.fw;
    }

    /**
     * Get the character height.
     * @return Character height
     */
    public float getCharHeight() {
        return this.fh;
    }
}
```

Figure 14: EventDraw.java
(uk.insrt.coursework.zuul.ui.EventDraw)

```java
public<E extends Event> void addListener(Class<E> event,
    IEventListener<E> listener) {
    this.getList(event).add(listener);
}
```

Figure 15: Excerpt from EventSystem.java
(uk.insrt.coursework.zuul.events.EventSystem)

```java
/**
 * Get the current weight of this inventory.
 * @return Weight (in kg)
 */
public double getWeight() {
    return this
        .items
        .stream()
        .mapToDouble(Entity::getWeight)
        .sum();
}

/**
 * Check if the inventory is full.
 * @return True if the weight is greater than the max weight
 */
public boolean isFull() {
    return this.getWeight() >= this.getMaxWeight();
}

/**
 * Add an entity to this inventory.
 *
 * There must be sufficient space for the entity.
 * @param entity Target Entity
 * @return Whether we successfully added the new entity.
 */
public boolean add(Entity entity) {
    if (this.getWeight() + entity.getWeight() > this.maxWeight) {
        return false;
    }

    this.items.add(entity);
    return true;
}
```

Figure 16: Excerpt from Inventory.java
(uk.insrt.coursework.zuul.entities.Inventory)

```java
/**
 * Write a string value to the text buffer.
 * @param value String value to write
 */
public void write(String value) {
    // Write each character sequentially.
    for (int i=0;i<value.length();i++) {
        char c = value.charAt(i);

        // If we encounter an Ansi escape character, then take the
        // substring from this point on and determine if it is a valid
        // escape code. If it is, apply any changes before continuing.
        if (c == '\u001B') {
            Matcher matcher = Ansi.AnsiPattern.matcher(value.substring(i));
            if (matcher.find()) {
                int v = Integer.parseInt(matcher.group(1));
                i += 3 + (v > 9 ? 1 : 0);

                if (v == 0) {
                    this.bg = Color.BLACK;
                    this.fg = Color.WHITE;
                } else if (v >= 30 && v < 38) {
                    this.fg = Ansi.fromEscapeCode(v);
                } else if (v >= 40 && v < 48) {
                    this.bg = Ansi.fromEscapeCode(v);
                }

                continue;
            }
        }

        this.write(c);
    }
}
```

Figure 17: Excerpt from TextBuffer.java
(uk.insrt.coursework.zuul.ui.TextBuffer)

```java
package uk.insrt.coursework.zuul.commands;

import java.util.regex.Matcher;

import uk.insrt.coursework.zuul.world.Direction;

/**
 * Wrapper around regex Matcher for deriving the values of given arguments in commands.
 */
public class Arguments {
    private Matcher matcher;

    /**
     * Construct a new Arguments wrapper.
     * @param matcher Regex Matcher
     */
    public Arguments(Matcher matcher) {
        this.matcher = matcher;
    }

    /**
     * Take a named group from the Matcher.
     * We assume that the provided Regex doesn't match the group if it's empty.
     * @param group Named group
     * @return String value of named group or null if it doesn't exist
     */
    public String group(String group) {
        // We ignore and return null on error as a bit of convenience.
        // This may not be best practice but it is justified by the fact
        // that it avoids an incredible amount of boilerplate further up
        // the chain, and in my opinion that's worth this design decision.
        try {
            return this.matcher.group(group);
        } catch (Exception e) {
            return null;
        }
    }

    /**
     * Check whether this named group was matched.
     * @param group Named group
```

```java
     * @return Whether this named group was matched
     */
    public boolean has(String group) {
        return this.group(group) != null;
    }

    /**
     * Get the provided Direction.
     * @return Parsed Direction value
     */
    public Direction direction() {
        return Direction.fromString(this.group("direction"));
    }
}
```

```java
package uk.insrt.coursework.zuul.commands;

import java.util.List;
import java.util.regex.Pattern;

import uk.insrt.coursework.zuul.entities.Entity;
import uk.insrt.coursework.zuul.entities.Inventory;
import uk.insrt.coursework.zuul.util.Search;
import uk.insrt.coursework.zuul.world.World;

/**
 * Representation of an action which can be performed by the user.
 */
public abstract class Command {
    private Pattern[] patterns;
    private String syntax;
    private String usage;

    /**
     * Construct a new Command.
     * @param syntax Information about how to use the command
     * @param usage Information about what the command does
     * @param patterns Patterns to execute this command on
     */
    public Command(String syntax, String usage, Pattern[] patterns) {
        this.patterns = patterns;
        this.syntax = syntax;
        this.usage = usage;
    }

    /**
     * Get information about how to use the command.
     * @return String Information about how to use the command
     */
    public String getSyntax() {
        return this.syntax;
    }

    /**
     * Get information about what the command does.
     * @return String Information about what the command does
```

```java
42          */
43         public String getUsage() {
44             return this.usage;
45         }
46
47         /**
48          * Get all applicable patterns to match to execute this command.
49          * @return Regex Pattern array
50          */
51         public Pattern[] getPatterns() {
52             return this.patterns;
53         }
54
55         /**
56          * Check whether this command is visible to the player in the help menu.
57          * @return True if visible
58          */
59         public boolean isVisible() {
60             return true;
61         }
62
63         /**
64          * Run this command within the scope of a world and with any parsed arguments.
65          * @param world Current World object
66          * @param args Arguments passed into command
67          * @return Boolean indicating whether the game loop should exit.
68          */
69         public abstract boolean run(World world, Arguments args);
70
71         /**
72          * Filter entities by those that are in the current room.
73          */
74         public static final int FILTER_ROOM = 1;
75
76         /**
77          * Filter entities by those that are in the player's inventory.
78          */
79         public static final int FILTER_INVENTORY = 2;
80
81         /**
82          * Don't filter entities and instead search through both the current room and the player's inventory.
```

```java
 83          */
 84         public static final int FILTER_ALL = FILTER_ROOM + FILTER_INVENTORY;
 85
 86         /**
 87          * Given a World and filter, use the provided Arguments and the relevant group to find an entity.
 88          * If an Entity is not found or not provided, the appropriate error is displayed to the player.
 89          * @param world World to look for the Entity within
 90          * @param filter Integer value which represents the filtering, specify one of: {@link #FILTER_ROOM}, {@link
#FILTER_INVENTORY}, {@link #FILTER_ALL}
 91          * @param args Arguments object to pull information out of
 92          * @param group Group we should pull the Entity query out of
 93          * @param failure Failure message if an Entity is not specified
 94          * @return An Entity if one is found, or null if one isn't.
 95          */
 96         public Entity findEntity(World world, int filter, Arguments args, String group, String failure) {
 97             String name = args.group(group);
 98             if (name == null) {
 99                 world.getIO().println(failure);
100                 return null;
101             }
102
103             // Search the inventory first.
104             Entity player = world.getPlayer();
105             Entity entity = null;
106             if ((filter & FILTER_INVENTORY) == FILTER_INVENTORY) {
107                 Inventory inventory = player.getInventory();
108                 entity = Search.findEntity(inventory.getItems(), name, true);
109             }
110
111             // If we haven't found an entity yet, search the room.
112             if (entity == null
113              && (filter & FILTER_ROOM) == FILTER_ROOM) {
114                 List<Entity> entities = world.getEntitiesInRoom(player.getRoom());
115                 entity = Search.findEntity(entities, name, true);
116             }
117
118             if (entity == null) {
119                 world.getIO().println("<selectors.cant_find.1> " + name + " <selectors.cant_find.2>.");
120             }
121
122             return entity;
```

```java
123        }
124
125        /**
126         * Given a World and filter, use the provided Arguments and using the group "entity" to find an entity.
127         * If an Entity is not found or not provided, the appropriate error is displayed to the player.
128         * @param world World to look for the Entity within
129         * @param filter Integer value which represents the filtering, specify one of: {@link #FILTER_ROOM}, {@link
#FILTER_INVENTORY}, {@link #FILTER_ALL}
130         * @param args Arguments object to pull information out of
131         * @param failure Failure message if an Entity is not specified
132         * @return An Entity if one is found, or null if one isn't.
133         */
134        public Entity findEntity(World world, int filter, Arguments args, String failure) {
135            return this.findEntity(world, filter, args, "entity", failure);
136        }
137
138        /**
139         * Given a World and using the {@link #FILTER_ROOM} filter, use the provided Arguments and using the group "entity"
to find an entity.
140         * If an Entity is not found or not provided, the appropriate error is displayed to the player.
141         * @param world World to look for the Entity within
142         * @param args Arguments object to pull information out of
143         * @param failure Failure message if an Entity is not specified
144         * @return An Entity if one is found, or null if one isn't.
145         */
146        public Entity findEntity(World world, Arguments args, String failure) {
147            return this.findEntity(world, FILTER_ROOM, args, failure);
148        }
149
150        /**
151         * Given a World and using the {@link #FILTER_ROOM} filter, use the provided Arguments and the relevant group to
find an entity.
152         * If an Entity is not found or not provided, the appropriate error is displayed to the player.
153         * @param world World to look for the Entity within
154         * @param args Arguments object to pull information out of
155         * @param group Group we should pull the Entity query out of
156         * @param failure Failure message if an Entity is not specified
157         * @return An Entity if one is found, or null if one isn't.
158         */
159        public Entity findEntity(World world, Arguments args, String group, String failure) {
160            return this.findEntity(world, FILTER_ROOM, args, group, failure);
```

```
161        }
162    }
```

```java
package uk.insrt.coursework.zuul.commands;

import java.util.ArrayList;
import java.util.List;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import uk.insrt.coursework.zuul.commands.core.CommandBack;
import uk.insrt.coursework.zuul.commands.core.CommandBag;
import uk.insrt.coursework.zuul.commands.core.CommandDrop;
import uk.insrt.coursework.zuul.commands.core.CommandGive;
import uk.insrt.coursework.zuul.commands.core.CommandGo;
import uk.insrt.coursework.zuul.commands.core.CommandHelp;
import uk.insrt.coursework.zuul.commands.core.CommandPet;
import uk.insrt.coursework.zuul.commands.core.CommandQuit;
import uk.insrt.coursework.zuul.commands.core.CommandTake;
import uk.insrt.coursework.zuul.commands.core.CommandTalk;
import uk.insrt.coursework.zuul.commands.core.CommandUse;
import uk.insrt.coursework.zuul.commands.core.CommandWhereAmI;
import uk.insrt.coursework.zuul.world.World;

/**
 * Command handler which constructs, then resolves
 * and executes commands from an arbitrary input.
 */
public class CommandManager {
    private ArrayList<Command> commands = new ArrayList<>();

    /**
     * Construct a new CommandManager.
     *
     * You should only need one present at any given time.
     */
    public CommandManager() {
        this.initialiseCommands();
    }

    /**
     * Register a new Command.
     * @param command Command
     */
```

```java
42      public void registerCommand(Command command) {
43          this.commands.add(command);
44      }
45
46      /**
47       * Register multiple commands.
48       * @param commands Command array
49       */
50      public void registerCommands(Command[] commands) {
51          for (Command command : commands) {
52              this.registerCommand(command);
53          }
54      }
55
56      /**
57       * Get Commands provided by this Command manager.
58       * @return List of Commands
59       */
60      public List<Command> getCommands() {
61          return this.commands;
62      }
63
64      /**
65       * Initialise all the commands a player can execute.
66       */
67      private void initialiseCommands() {
68          final Command[] DEFAULT_COMMANDS = {
69              new CommandBack(),
70              new CommandBag(),
71              new CommandDrop(),
72              new CommandGive(),
73              new CommandGo(),
74              new CommandHelp(this),
75              new CommandPet(),
76              new CommandQuit(),
77              new CommandTake(),
78              new CommandTalk(),
79              new CommandUse(),
80              new CommandWhereAmI(),
81          };
82
```

```java
 83            this.registerCommands(DEFAULT_COMMANDS);
 84        }
 85
 86        /**
 87         * Interpret a given command and execute it within the scope of a given world.
 88         * @param world Current World object
 89         * @param cmd Arbitrary input to match against
 90         * @return Boolean indicating whether the game loop should exit.
 91         */
 92        public boolean runCommand(World world, String cmd) {
 93            for (Command command : this.commands) {
 94                for (Pattern pattern : command.getPatterns()) {
 95                    Matcher matcher = pattern.matcher(cmd);
 96                    if (matcher.find()) {
 97                        Arguments arguments = new Arguments(matcher);
 98                        return command.run(world, arguments);
 99                    }
100                }
101            }
102
103            world.getIO().println("<commands.unknown>");
104            return false;
105        }
106    }
```

```java
package uk.insrt.coursework.zuul.commands.core;

import java.util.regex.Pattern;

import uk.insrt.coursework.zuul.commands.Arguments;
import uk.insrt.coursework.zuul.commands.Command;
import uk.insrt.coursework.zuul.world.World;

/**
 * Command which allows the Player to walk back through the previous Rooms.
 */
public class CommandBack extends Command {
    public CommandBack() {
        super("back", "<commands.back>",
            new Pattern[] {
                Pattern.compile("^(?:(?:go|walk)\\s+)*back(?!\\w)"),
                // back, go back, walk back
            });
    }

    @Override
    public boolean run(World world, Arguments arguments) {
        // We call a specialised method on the player as we keep
        // track of visited rooms within the Player class itself.
        world.getPlayer().back();
        return false;
    }
}
```

```java
package uk.insrt.coursework.zuul.commands.core;

import java.text.DecimalFormat;
import java.util.regex.Pattern;

import uk.insrt.coursework.zuul.commands.Arguments;
import uk.insrt.coursework.zuul.commands.Command;
import uk.insrt.coursework.zuul.entities.Entity;
import uk.insrt.coursework.zuul.entities.Inventory;
import uk.insrt.coursework.zuul.io.Ansi;
import uk.insrt.coursework.zuul.io.IOSystem;
import uk.insrt.coursework.zuul.world.World;

/**
 * Command which allows the Player to look at their or another Entity's inventory.
 */
public class CommandBag extends Command {
    private final DecimalFormat format = new DecimalFormat("0.00");

    public CommandBag() {
        super("inventory [of <selectors.something>]", "<commands.bag.usage>",
            new Pattern[] {
                Pattern.compile("^(?:b(?:ag)*|inv(?:entory)*)(?:\\s+(?<entity>[\\w\\s]+))*"),
                // b, bag, inv, inventory, bag of <entity>, inventory of <entity>, (+2)
            });
    }

    @Override
    public boolean run(World world, Arguments arguments) {
        IOSystem io = world.getIO();

        // Figure out if we're checking our own inventory or another entity's inventory.
        Entity entity;
        boolean ours;
        if (arguments.has("entity")) {
            ours = false;
            entity = this.findEntity(world, arguments, "<commands.bag.cant_find>");
            if (entity == null) return false;
        } else {
            ours = true;
            entity = world.getPlayer();
```

```java
42          }

43

44          // Get the selected entity's inventory and provide output if empty.
45          Inventory inv = entity.getInventory();
46          if (inv.getWeight() == 0) {
47              if (ours) {
48                  io.println("<commands.bag.empty> <commands.bag.can_carry_kg> "
49                      + inv.getMaxWeight() + " kg.");
50              } else {
51                  io.println(entity.getHighlightedName() + " <commands.bag.entity_empty>.");
52              }

53

54              return false;
55          }

56

57          // Otherwise describe some statistics about the inventory.
58          if (ours) {
59              io.println("<commands.bag.are_carrying_kg> " + this.format.format(inv.getWeight())
60                  + " / " + inv.getMaxWeight() + " kg.\n<commands.bag.look_in_bag>:");
61          } else {
62              io.println(entity.getHighlightedName() + " <commands.bag.entity_appears_to_have>:");
63          }

64

65          // Describe all the items in this inventory we are currently looking at.
66          for (Entity item : inv.getItems()) {
67              io.println("- " + Ansi.Yellow + item.getWeight() + " kg"
68                  + Ansi.Reset + " " + item.describe()
69                  + " (" + item.getHighlightedName() + ")");
70          }

71

72          return false;
73      }
74  }
```

```java
package uk.insrt.coursework.zuul.commands.core;

import java.util.regex.Pattern;

import uk.insrt.coursework.zuul.commands.Arguments;
import uk.insrt.coursework.zuul.commands.Command;
import uk.insrt.coursework.zuul.entities.Entity;
import uk.insrt.coursework.zuul.io.Ansi;
import uk.insrt.coursework.zuul.world.World;

/**
 * Command which allows the Player to drop any item in their inventory.
 */
public class CommandDrop extends Command {
    public CommandDrop() {
        super("drop <selectors.item>", "<commands.drop.usage>",
            new Pattern[] {
                Pattern.compile("^(?:drop|place|put down)(?:\\s+(?<entity>[\\w\\s]+))*")
                // drop, place, put down, drop <item>, place <item>, put down <item>
            });
    }

    @Override
    public boolean run(World world, Arguments args) {
        // Find the given entity within our inventory and drop it if it's found.
        Entity entity = this.findEntity(world, Command.FILTER_INVENTORY, args, "<commands.drop.nothing_specified>");
        if (entity != null) {
            world.getIO().println("<commands.drop.dropped.1> " + Ansi.BackgroundWhite + Ansi.Black
                + entity.getName() + Ansi.Reset + " <commands.drop.dropped.2>!");
            entity.setLocation(world.getPlayer().getRoom());
        }

        return false;
    }
}
```

```java
package uk.insrt.coursework.zuul.commands.core;

import java.util.regex.Pattern;

import uk.insrt.coursework.zuul.commands.Arguments;
import uk.insrt.coursework.zuul.commands.Command;
import uk.insrt.coursework.zuul.entities.Entity;
import uk.insrt.coursework.zuul.entities.EntityPlayer;
import uk.insrt.coursework.zuul.entities.actions.IGiveable;
import uk.insrt.coursework.zuul.world.World;

/**
 * Command which allows the player to give something to someone.
 */
public class CommandGive extends Command {
    public CommandGive() {
        super("give <selectors.something> to <selectors.someone>", "<commands.give.usage>",
            new Pattern[] {
                Pattern.compile("^(?:give|put)(?:\\s+(?<item>[\\w\\s]+)\\s+(?:to|in)\\s+(?<entity>[\\w\\s]+))*")
                // give, put, give <something> to <someone>, put <something> in <something>, (+2)
            });
    }

    @Override
    public boolean run(World world, Arguments args) {
        // Find the entity we want to give in the room or our inventory.
        Entity item = this.findEntity(world, Command.FILTER_ALL, args, "item", "<commands.give.nothing_specified>");
        if (item == null) return false;

        // Explicitly deny the player being given to anything, otherwise we will end up in an inventory.
        var io = world.getIO();
        if (item instanceof EntityPlayer) {
            io.println("<commands.give.denied_player>");
            return false;
        }

        // Find the target to give to.
        Entity target = this.findEntity(world, args, "<commands.give.no_target>");
        if (target == null) return false;

        // If the target entity is an IGiveable, check if they accept this item.
```

```java
        if (target instanceof IGiveable) {
            ((IGiveable) target).give(item);
        } else {
            io.println("<commands.give.denied.1> " + item.getHighlightedName()
                    + " <commands.give.denied.2> " + target.getHighlightedName() + "!");
        }

        return false;
    }
}
```

```java
package uk.insrt.coursework.zuul.commands.core;

import java.util.regex.Pattern;

import uk.insrt.coursework.zuul.commands.Arguments;
import uk.insrt.coursework.zuul.commands.Command;
import uk.insrt.coursework.zuul.world.Direction;
import uk.insrt.coursework.zuul.world.World;

/**
 * Command which allows the Player to walk in a particular Direction.
 */
public class CommandGo extends Command {
    public CommandGo() {
        super("go <selectors.direction>", "<commands.go.usage>",
            new Pattern[] {
                Pattern.compile("^(?:go|walk)(?:\\s+(?<direction>[\\w\\s]+))*")
                // go, walk, go <direction>, walk <direction>
            });
    }

    @Override
    public boolean run(World world, Arguments arguments) {
        Direction direction = arguments.direction();
        if (direction == null) {
            world.getIO().println("<commands.go.nothing_specified>");
            return false;
        }

        world.getPlayer().go(direction);
        return false;
    }
}
```

```java
package uk.insrt.coursework.zuul.commands.core;

import java.util.regex.Pattern;
import java.util.stream.Collectors;

import uk.insrt.coursework.zuul.commands.Arguments;
import uk.insrt.coursework.zuul.commands.Command;
import uk.insrt.coursework.zuul.commands.CommandManager;
import uk.insrt.coursework.zuul.io.Ansi;
import uk.insrt.coursework.zuul.world.World;

/**
 * Command which allows the player to list all the available commands.
 */
public class CommandHelp extends Command {
    private CommandManager commandManager;

    public CommandHelp(CommandManager commandManager) {
        super("help", "<commands.help.usage>",
            new Pattern[] {
                Pattern.compile("^(?:h(?:elp)*)(?!\\w)")
                // h, help
            });

        this.commandManager = commandManager;
    }

    @Override
    public boolean run(World world, Arguments arguments) {
        // Describe all the commands the player can run.
        world.getIO()
            .println(
            "<commands.help.can_run>\n" +
            this.commandManager
                .getCommands()
                .stream()
                .filter(Command::isVisible)
                .map(c -> "- " + Ansi.BackgroundWhite + Ansi.Black
                    + c.getSyntax() + Ansi.Reset + ": " + c.getUsage())
                .collect(Collectors.joining("\n"))
        );
```

```java
42
43            return false;
44        }
45
46        @Override
47        public boolean isVisible() {
48            return false;
49        }
50    }
```

```java
package uk.insrt.coursework.zuul.commands.core;

import java.util.regex.Pattern;

import uk.insrt.coursework.zuul.commands.Arguments;
import uk.insrt.coursework.zuul.commands.Command;
import uk.insrt.coursework.zuul.entities.Entity;
import uk.insrt.coursework.zuul.entities.actions.IPettable;
import uk.insrt.coursework.zuul.world.World;

/**
 * Command which allows the player to pet another entity.
 */
public class CommandPet extends Command {
    public CommandPet() {
        super("pet <selectors.something>", "<commands.pet.usage>",
            new Pattern[] {
                Pattern.compile("^pet(?:\\s+(?<entity>[\\w\\s]+))*")
                // pet, pet <something>
            });
    }

    @Override
    public boolean run(World world, Arguments args) {
        // Scan the room for entities that have IPettable.
        Entity entity = this.findEntity(world, Command.FILTER_ALL, args, "<commands.pet.nothing_specified>");
        if (entity != null) {
            if (entity instanceof IPettable) {
                ((IPettable) entity).pet();
            } else {
                world.getIO().println("<commands.pet.denied> " + entity.getHighlightedName() + ".");
            }
        }

        return false;
    }
}
```

```java
package uk.insrt.coursework.zuul.commands.core;

import java.util.regex.Pattern;

import uk.insrt.coursework.zuul.commands.Arguments;
import uk.insrt.coursework.zuul.commands.Command;
import uk.insrt.coursework.zuul.world.World;

/**
 * Command which allows the player to quit the game.
 */
public class CommandQuit extends Command {
    public CommandQuit() {
        super("quit", "<commands.quit>",
            new Pattern[] {
                Pattern.compile("^quit|exit(?!\\w)"),
                // quit
            });
    }

    @Override
    public boolean run(World world, Arguments arguments) {
        // We return true from run() in order to tell the game loop to exit.
        return true;
    }
}
```

```java
package uk.insrt.coursework.zuul.commands.core;

import java.util.regex.Pattern;

import uk.insrt.coursework.zuul.commands.Arguments;
import uk.insrt.coursework.zuul.commands.Command;
import uk.insrt.coursework.zuul.entities.Entity;
import uk.insrt.coursework.zuul.entities.Inventory;
import uk.insrt.coursework.zuul.io.IOSystem;
import uk.insrt.coursework.zuul.util.Search;
import uk.insrt.coursework.zuul.world.World;

/**
 * Command which allows the Player to take an Entity and put it in their Inventory.
 * They may also take these Entities from other Entity Inventories.
 */
public class CommandTake extends Command {
    public CommandTake() {
        super("take <selectors.something> [from <selectors.someone>]", "<commands.take.usage>",
            new Pattern[] {
                Pattern.compile("^take\\s+(?<entity>[\\w\\s]+)\\s+from\\s+(?<other>[\\w\\s]+)"),
                Pattern.compile("^take(?:\\s+(?<entity>[\\w\\s]+))*")
                // take, take <item>, take <item> from <entity>
            });
    }

    @Override
    public boolean run(World world, Arguments args) {
        IOSystem io = world.getIO();
        Entity player = world.getPlayer();
        Inventory target = player.getInventory();

        // Detect if we are taking from another entity, in that case run different logic.
        if (args.has("other")) {
            String name = args.group("entity");
            Entity other = this.findEntity(world, args, "other", "<commands.take.nothing_specified>");
            if (other == null) return false;

            Entity item = Search.findEntity(other.getInventory().getItems(), name, true);
            if (item == null) {
                io.println(other.getHighlightedName() + " <commands.take.entity_does_not_have_entity> " + name + "!");
```

```java
42                return false;
43            }
44
45            if (item.setLocation(target)) {
46                io.println("<commands.take.took.1> " + item.getHighlightedName()
47                    + " <commands.take.took.2> " + other.getHighlightedName()
48                    + " <commands.take.took.3>.");
49            } else {
50                io.println("<commands.take.denied.1> " + item.getName()
51                    + ", <commands.take.denied.2>.");
52            }
53
54            return false;
55        }
56
57        // Otherwise, look around the room and find something we can take.
58        Entity entity = this.findEntity(world, args, "<commands.take.item_not_specified>");
59        if (entity != null) {
60            if (entity == player) {
61                io.println("\u1F633");
62                return false;
63            }
64
65            if (entity.setLocation(target)) {
66                io.println("<commands.take.took.1> " + entity.getHighlightedName()
67                    + " <commands.take.took.3>.");
68            } else {
69                io.println("<commands.take.denied.1> " + entity.getHighlightedName()
70                    + ", <commands.take.denied.2>.");
71            }
72        }
73
74        return false;
75    }
76 }
```

```java
package uk.insrt.coursework.zuul.commands.core;

import java.util.regex.Pattern;

import uk.insrt.coursework.zuul.commands.Arguments;
import uk.insrt.coursework.zuul.commands.Command;
import uk.insrt.coursework.zuul.entities.Entity;
import uk.insrt.coursework.zuul.entities.actions.ITalkwith;
import uk.insrt.coursework.zuul.world.World;

/**
 * Command which allows the Player to talk with other Entities.
 */
public class CommandTalk extends Command {
    public CommandTalk() {
        super("talk with <selectors.someone>", "<commands.talk.usage>",
            new Pattern[] {
                Pattern.compile("^talk(?:(?:\\s*with|to)*(?:\\s+(?<entity>[\\w\\s]+))*)*")
                // talk, talk with <entity>, talk to <entity>
            });
    }

    @Override
    public boolean run(World world, Arguments args) {
        Entity entity = this.findEntity(world, args, "<commands.talk.nothing_specified>");
        if (entity != null) {
            if (entity instanceof ITalkwith) {
                ((ITalkwith) entity).talk();
            } else {
                world.getIO().println("<commands.talk.denied> " + entity.getHighlightedName() + ".");
            }
        }

        return false;
    }
}
```

```java
package uk.insrt.coursework.zuul.commands.core;

import java.util.regex.Pattern;

import uk.insrt.coursework.zuul.commands.Arguments;
import uk.insrt.coursework.zuul.commands.Command;
import uk.insrt.coursework.zuul.entities.Entity;
import uk.insrt.coursework.zuul.entities.actions.IUseable;
import uk.insrt.coursework.zuul.world.World;

/**
 * Command which allows the Player to use an Entity.
 */
public class CommandUse extends Command {
    public CommandUse() {
        super("use <selectors.something>", "<commands.use.usage>",
            new Pattern[] {
                Pattern.compile("^use(?:\\s+(?<entity>[\\w\\s]+))*")
                // use, use <entity>
            });
    }

    @Override
    public boolean run(World world, Arguments args) {
        Entity entity = this.findEntity(world, Command.FILTER_ALL, args, "<commands.use.nothing_specified>");
        if (entity != null) {
            if (entity instanceof IUseable) {
                ((IUseable) entity).use(world.getPlayer());
            } else {
                world.getIO().println("<commands.use.denied> " + entity.getHighlightedName() + ".");
            }
        }

        return false;
    }
}
```

```java
package uk.insrt.coursework.zuul.commands.core;

import java.util.regex.Pattern;

import uk.insrt.coursework.zuul.commands.Arguments;
import uk.insrt.coursework.zuul.commands.Command;
import uk.insrt.coursework.zuul.events.world.EventEntityEnteredRoom;
import uk.insrt.coursework.zuul.world.World;

/**
 * Command which allows the player to reorient themselves in the world.
 */
public class CommandWhereAmI extends Command {
    public CommandWhereAmI() {
        super("where am i", "<commands.where_am_i>",
            new Pattern[] {
                Pattern.compile("^where(\\s+am\\s+(i(?!\\w))*)*"),
                // where am i
            });
    }

    @Override
    public boolean run(World world, Arguments arguments) {
        // We can just re-emit the enter room event to
        // trigger the room description logic to run again.
        world.emit(new EventEntityEnteredRoom(world.getPlayer()));
        return false;
    }
}
```

```java
package uk.insrt.coursework.zuul.content.campaign;

import java.io.IOException;
import java.util.HashSet;
import java.util.stream.Collectors;

import uk.insrt.coursework.zuul.content.campaign.StoryFlags.Stage;
import uk.insrt.coursework.zuul.content.campaign.entities.EntityCat;
import uk.insrt.coursework.zuul.content.campaign.entities.EntityWithDialogue;
import uk.insrt.coursework.zuul.content.campaign.events.EventGameStageChanged;
import uk.insrt.coursework.zuul.content.campaign.rooms.RoomApartmentsHome;
import uk.insrt.coursework.zuul.content.campaign.rooms.RoomApartmentsReception;
import uk.insrt.coursework.zuul.content.campaign.rooms.RoomBackAlley;
import uk.insrt.coursework.zuul.content.campaign.rooms.RoomCityCentre;
import uk.insrt.coursework.zuul.content.campaign.rooms.RoomCoastline;
import uk.insrt.coursework.zuul.content.campaign.rooms.RoomForest;
import uk.insrt.coursework.zuul.content.campaign.rooms.RoomMainlandCoastline;
import uk.insrt.coursework.zuul.content.campaign.rooms.RoomMedicalCentreOffice;
import uk.insrt.coursework.zuul.content.campaign.rooms.RoomMedicalCentreReception;
import uk.insrt.coursework.zuul.content.campaign.rooms.RoomShop;
import uk.insrt.coursework.zuul.content.campaign.rooms.RoomStreet;
import uk.insrt.coursework.zuul.content.campaign.rooms.RoomWormHole;
import uk.insrt.coursework.zuul.dialogue.DialogueLoader;
import uk.insrt.coursework.zuul.entities.Entity;
import uk.insrt.coursework.zuul.entities.EntityPlayer;
import uk.insrt.coursework.zuul.events.IEventListener;
import uk.insrt.coursework.zuul.events.world.EventEntityEnteredRoom;
import uk.insrt.coursework.zuul.events.world.EventEntityLeftRoom;
import uk.insrt.coursework.zuul.io.Ansi;
import uk.insrt.coursework.zuul.io.IOSystem;
import uk.insrt.coursework.zuul.world.Room;
import uk.insrt.coursework.zuul.world.World;

/**
 * The main campaign World.
 */
public class CampaignWorld extends World {
    private StoryFlags flags;
    private HashSet<Room> visitedRooms;
    private DialogueLoader dialogueLoader;

```

```java
42          /**
43           * Construct a new Campaign World
44           * @param io Provided IO system
45           */
46          public CampaignWorld(IOSystem io) {
47              super(io);
48
49              this.visitedRooms = new HashSet<>();
50              this.dialogueLoader = new DialogueLoader();
51              this.flags = new StoryFlags(this.getEventSystem());
52
53              try {
54                  this.dialogueLoader.load("/dialogue.toml");
55              } catch (IOException e) {
56                  System.err.println("Failed to load resources for campaign world!");
57                  e.printStackTrace();
58              }
59
60              this.buildWorld();
61              this.spawnEntities();
62              this.registerEvents();
63          }
64
65          /**
66           * Get this World's Dialogue Loader
67           * @return Dialogue Loader
68           */
69          public DialogueLoader getDialogueLoader() {
70              return this.dialogueLoader;
71          }
72
73          /**
74           * Get the global story flags.
75           * @return Story flags instance
76           */
77          public StoryFlags getStoryFlags() {
78              return this.flags;
79          }
80
81          /**
82           * Check whether the Player has visited a certain Room yet
```

```java
 83        * @param room Room to check
 84        * @return True if the Player has visited the given Room
 85        */
 86       public boolean hasVisited(Room room) {
 87           return this.visitedRooms.contains(room);
 88       }
 89
 90       /**
 91        * Get a rounded whole number percentage of how much the World has been explored.
 92        * @return Integer representing percentage of World explored
 93        */
 94       public int percentVisited() {
 95           return Math.round((float) this.visitedRooms.size() / this.rooms.size() * 100.0f);
 96       }
 97
 98       /**
 99        * Create all the Worlds and link adjacent Rooms together.
100        */
101       private void buildWorld() {
102           final Room[] rooms = {
103               new RoomCityCentre(this),
104               new RoomStreet(this),
105               new RoomShop(this),
106               new RoomBackAlley(this),
107               new RoomApartmentsReception(this),
108               new RoomApartmentsHome(this),
109               new RoomMedicalCentreReception(this),
110               new RoomMedicalCentreOffice(this),
111               new RoomCoastline(this),
112               new RoomMainlandCoastline(this),
113               new RoomForest(this),
114               new RoomWormHole(this)
115           };
116
117           for (Room room : rooms) {
118               this.addRoom(room);
119           }
120
121           this.linkRooms();
122       }
123
```

```java
124        /**
125         * Spawn and setup any Entities within this World.
126         */
127        private void spawnEntities() {
128            for (Room room : this.rooms.values()) {
129                room.spawnEntities();
130            }
131
132            // Entangle boat inventories.
133            Entity boat1 = this.entities.get("boat1");
134            Entity boat2 = this.entities.get("boat2");
135            boat1.entangleInventory(boat2.getInventory());
136        }
137
138        /**
139         * Register all the game logic.
140         */
141        private void registerEvents() {
142            // Capture all Events for Entities entering Rooms.
143            this.eventSystem.addListener(EventEntityEnteredRoom.class,
144                (EventEntityEnteredRoom event) -> {
145                    Entity entity = event.getEntity();
146                    if (entity instanceof EntityPlayer) {
147                        Room room = entity.getRoom();
148
149                        // Whenever the Player enters a Room, we should print the
150                        // description of the Room and list things found in the Room.
151                        this.io.println(
152                            room.describe()
153                                + "\n<global.can_go_in_x_directions.1> "
154                                + room.getDirections().size()
155                                + " <global.can_go_in_x_directions.2>: "
156                                + room.getDirections()
157                                    .stream()
158                                    .map(x ->
159                                        x.toString()
160                                         .toLowerCase()
161                                         .replaceAll("_", " ")
162                                    )
163                                    .collect(Collectors.joining(", "))
164                        );
```

```java
165
166                    // Mark current room as previously visited.
167                    this.visitedRooms.add(room);
168
169                    // When we enter a new room, list what we can see.
170                    String entities = this.getEntitiesInRoom(entity.getRoom())
171                        .stream()
172                        .filter(e -> !(e instanceof EntityPlayer))
173                        .map(e -> "- " + e.describe() + " ("
174                            + Ansi.BackgroundPurple + Ansi.Black
175                            + e.getName() + Ansi.Reset + ")")
176                        .collect(Collectors.joining("\n"));
177
178                    if (entities.length() > 0) {
179                        this.io.println("<global.sight>\n" + entities);
180                    }
181                } else {
182                    // If another entity enters the room,
183                    // conditionally mention this to the player.
184                    EntityPlayer player = this.getPlayer();
185                    if (entity.getRoom() == player.getRoom()) {
186                        if (entity instanceof EntityCat) {
187                            this.io.println("\n<entities.cat.enter>");
188                        }
189                    }
190                }
191            });
192
193        // Capture all Events for Entities leaving Rooms.
194        this.eventSystem.addListener(EventEntityLeftRoom.class,
195            (EventEntityLeftRoom event) -> {
196                Entity entity = event.getEntity();
197                if (entity instanceof EntityPlayer) return;
198
199                Room room = event.getRoom();
200                if (room != this.player.getRoom()) return;
201
202                // If another entity leaves the room,
203                // conditionally mention this to the player.
204                if (entity instanceof EntityCat) {
205                    this.io.println("\n<entities.cat.leave>");
```

```java
206                    }
207                });
208
209            // Register event for game stage changing.
210            this.eventSystem.addListener(EventGameStageChanged.class,
211                (EventGameStageChanged event) -> {
212                    Stage stage = event.getStage();
213                    switch (stage) {
214                        case Recon: {
215                            for (Entity entity : this.entities.values()) {
216                                if (entity instanceof EntityWithDialogue) {
217                                    ((EntityWithDialogue<?>) entity).setDialogueNodeIfPresent("recon");
218                                }
219                            }
220                            break;
221                        }
222                        case End: {
223                            io.println("<stage.reached_conclusion>");
224                            break;
225                        }
226                        default: break;
227                    }
228                });
229
230            // Register required Events for Worm Hole room to function.
231            @SuppressWarnings("unchecked")
232            var wh = (IEventListener<EventEntityEnteredRoom>) this.getRoom("Worm Hole");
233            this.eventSystem.addListener(EventEntityEnteredRoom.class, wh);
234
235            // Register required Events for the protestors to disappear.
236            @SuppressWarnings("unchecked")
237            var st = (IEventListener<EventGameStageChanged>) this.getRoom("Street");
238            this.eventSystem.addListener(EventGameStageChanged.class, st);
239        }
240
241        @Override
242        public void spawnPlayer() {
243            this.player.setLocation(this.rooms.get("Apartments: Home"));
244        }
245    }
```

```java
package uk.insrt.coursework.zuul.content.campaign.commands;

import java.awt.Image;
import java.io.InputStream;
import java.util.regex.Pattern;

import javax.imageio.ImageIO;

import uk.insrt.coursework.zuul.commands.Arguments;
import uk.insrt.coursework.zuul.commands.Command;
import uk.insrt.coursework.zuul.content.campaign.CampaignWorld;
import uk.insrt.coursework.zuul.events.IEventListener;
import uk.insrt.coursework.zuul.io.IOSystem;
import uk.insrt.coursework.zuul.ui.EventDraw;
import uk.insrt.coursework.zuul.world.World;

/**
 * Command available for the terminal emulator which displays a graphical map.
 */
public class CommandMap extends Command implements IEventListener<EventDraw> {
    private boolean visible;
    private Image image;

    public CommandMap() {
        super("map", "<commands.map.usage>",
            new Pattern[] {
                Pattern.compile("^(?:m(?:ap)*)(?!\\w)")
                // m, map
            });

        this.visible = false;

        try {
            InputStream stream = this.getClass().getResourceAsStream("/map/base.png");
            this.image = ImageIO.read(stream);
        } catch (Exception e) {}
    }

    @Override
    public boolean run(World world, Arguments arguments) {
        CampaignWorld campaignWorld = (CampaignWorld) world;
```

```java
        IOSystem io = world.getIO();
        io.print("<commands.map.discovered.1> " + campaignWorld.percentVisited()
            + "% <commands.map.discovered.2>!" + "\n".repeat(24) + "<commands.map.close>");

        // We make the map visible and block on user input,
        // Once the user interacts, the map is hidden again.
        this.visible = true;
        io.readLine();
        this.visible = false;
        return false;
    }

    @Override
    public void onEvent(EventDraw event) {
        if (!this.visible) return;

        // We are drawing the map from [0,1] to [80,24].
        float fw = event.getCharWidth();
        float fh = event.getCharHeight();

        var g = event.getGraphics();
        g.drawImage(
            this.image,
            Math.round(event.getOriginX()),
            Math.round(event.getOriginY() + fh),
            Math.round(fw * 80),
            Math.round(fh * 23),
            null
        );
    }
}
```

```java
package uk.insrt.coursework.zuul.content.campaign.commands;

import java.util.regex.Pattern;

import uk.insrt.coursework.zuul.commands.Arguments;
import uk.insrt.coursework.zuul.commands.Command;
import uk.insrt.coursework.zuul.content.campaign.CampaignWorld;
import uk.insrt.coursework.zuul.content.campaign.StoryFlags.Stage;
import uk.insrt.coursework.zuul.world.World;

/**
 * This command is unlocked after the player completes the final mission.
 */
public class CommandWin extends Command {
    public CommandWin() {
        super("win", "<commands.win.usage>",
            new Pattern[] {
                Pattern.compile("^win(?!\\w)"),
                // win
            });
    }

    @Override
    public boolean run(World world, Arguments args) {
        var io = world.getIO();
        var w = (CampaignWorld) world;
        var flags = w.getStoryFlags();
        if (flags.getStage() != Stage.End) return false;

        io.println("<commands.win.conclusion>\n<commands.win.stats>\n"
            + "<commands.win.total_ticks>" + flags.getTicks() + "\n"
            + "<commands.win.sidequests_complete>"
            + flags.getCompletedQuests() + " / " + flags.getTotalQuests()
            + "\n\n<commands.win.press_enter_key>");

        io.readLine();
        return true;
    }

    @Override
    public boolean isVisible() {
```

```
42            return false;
43        }
44    }
```

```java
package uk.insrt.coursework.zuul.content.campaign.entities;

import uk.insrt.coursework.zuul.entities.Entity;
import uk.insrt.coursework.zuul.entities.EntityObject;
import uk.insrt.coursework.zuul.entities.actions.IUseable;
import uk.insrt.coursework.zuul.events.world.EventTick;
import uk.insrt.coursework.zuul.world.Location;
import uk.insrt.coursework.zuul.world.World;

/**
 * Bed entity which lets the player tick the World forwards.
 */
public class EntityBed extends EntityObject implements IUseable {
    public EntityBed(World world, Location location) {
        super(world, location, 80, new String[] { "bed" }, "<entities.bed.description>");
    }

    public void use(Entity target) {
        // We emit EventTick an arbitrary amount of times to
        // in-effect push the time forwards. This will trigger
        // all random events which listen to this event.
        for (int i=0;i<20;i++) {
            world.emit(new EventTick());
        }

        this.world.getIO().println("<entities.bed.use>");
    }
}
```

```java
package uk.insrt.coursework.zuul.content.campaign.entities;

import uk.insrt.coursework.zuul.content.campaign.CampaignWorld;
import uk.insrt.coursework.zuul.content.campaign.StoryFlags.Stage;
import uk.insrt.coursework.zuul.entities.Entity;
import uk.insrt.coursework.zuul.entities.Inventory;
import uk.insrt.coursework.zuul.entities.actions.IGiveable;
import uk.insrt.coursework.zuul.entities.actions.IUseable;
import uk.insrt.coursework.zuul.io.IOSystem;
import uk.insrt.coursework.zuul.world.Location;
import uk.insrt.coursework.zuul.world.Room;
import uk.insrt.coursework.zuul.world.World;

/**
 * Boat entity which ferries the Player to an arbitrary location.
 * There is no restriction on location but they should be place as
 * appropriate and where it would be realistic to put a boat.
 *
 * Boats may not be operated by the player while they are carrying
 * anything so instead they must use the boat's storage.
 */
public class EntityBoat extends Entity implements IUseable, IGiveable {
    private Room destination;

    public EntityBoat(World world, Location location, Room destination) {
        super(world, location, 200);
        this.destination = destination;
        this.inventory.setMaxWeight(100);
    }

    @Override
    public String[] getAliases() {
        return new String[] { "boat" };
    }

    @Override
    public String describe() {
        return "<entities.boat.description>";
    }

    @Override
```

```java
42      public void use(Entity target) {
43          CampaignWorld world = (CampaignWorld) this.getWorld();
44          IOSystem io = world.getIO();
45          Inventory inventory = target.getInventory();
46
47          // Check if the player has the key to this boat.
48          boolean hasKey = false;
49          for (Entity item : inventory.getItems()) {
50              if (item instanceof EntityBoatKey) {
51                  hasKey = true;
52              }
53          }
54
55          if (!hasKey) {
56              if (world.getStoryFlags().getStage() == Stage.Exposition) {
57                  io.println("<entities.boat.locked>");
58              } else {
59                  io.println("<entities.boat.locked_for_sale>");
60              }
61
62              return;
63          }
64
65          // Check whether the player is carrying too much.
66          if (inventory.getWeight() > 1) {
67              io.println("<entities.boat.denied>");
68              return;
69          }
70
71          // If we're good to go, travel to the other side.
72          io.println("<entities.boat.travel>\n");
73          target.setLocation(this.destination);
74      }
75
76      @Override
77      public void give(Entity item) {
78          var io = this.getWorld().getIO();
79          if (item.setLocation(this.getInventory())) {
80              io.println("<entities.boat.give.1> " + item.getHighlightedName() + " <entities.boat.give.2>.");
81          } else {
82              io.println("<entities.boat.too_heavy>");
```

```
83              }
84          }
85      }
```

```java
package uk.insrt.coursework.zuul.content.campaign.entities;

import uk.insrt.coursework.zuul.entities.EntityObject;
import uk.insrt.coursework.zuul.world.Location;
import uk.insrt.coursework.zuul.world.World;

/**
 * Boat key object which is used to unlock and start the
 * speed boat present on the coast.
 */
public class EntityBoatKey extends EntityObject {
    public EntityBoatKey(World world, Location location) {
        super(world, location, 0.01d,
            new String[] { "key" },
            "<entities.boat_key>");
    }
}
```

```java
package uk.insrt.coursework.zuul.content.campaign.entities;

import uk.insrt.coursework.zuul.entities.Entity;
import uk.insrt.coursework.zuul.entities.actions.IPettable;
import uk.insrt.coursework.zuul.entities.actions.IUseable;
import uk.insrt.coursework.zuul.events.world.EventTick;
import uk.insrt.coursework.zuul.events.world.behaviours.SimpleWanderAI;
import uk.insrt.coursework.zuul.world.Location;
import uk.insrt.coursework.zuul.world.Room;
import uk.insrt.coursework.zuul.world.World;

/**
 * Cat entity which wanders around the map.
 */
public class EntityCat extends Entity implements IPettable, IUseable {
    public static final int WEIGHT = 5;

    public EntityCat(World world, Location startingLocation) {
        super(world, startingLocation, WEIGHT);
    }

    @Override
    public String[] getAliases() {
        return new String[] {
            "cat",
            "the cat"
        };
    }

    @Override
    public String describe() {
        return "<entities.cat.description>";
    }

    @Override
    public void pet() {
        this.getWorld().getIO().println("<entities.cat.pet>");
    }

    @Override
    public void use(Entity target) {
```

```java
42              this.getWorld().getIO().println("<entities.cat.use>");
43          }
44
45          /**
46           * Enable a simple wander behaviour for entity within given bounds.
47           * @param rooms Path that this Entity should follow
48           * @param chance The chance x that this entity moves, where x gives a 1/x fractional chance of moving.
49           */
50          public void useWanderAI(Room[] rooms, int chance) {
51              this.getWorld()
52                  .getEventSystem()
53                  .addListener(EventTick.class, new SimpleWanderAI(this, rooms, chance));
54          }
55      }
```

```java
package uk.insrt.coursework.zuul.content.campaign.entities;

import uk.insrt.coursework.zuul.content.campaign.CampaignWorld;
import uk.insrt.coursework.zuul.content.campaign.StoryFlags.Stage;
import uk.insrt.coursework.zuul.content.campaign.rooms.RoomMedicalCentreOffice;
import uk.insrt.coursework.zuul.content.campaign.rooms.RoomMedicalCentreReception;
import uk.insrt.coursework.zuul.dialogue.Dialogue;
import uk.insrt.coursework.zuul.dialogue.DialogueOption;
import uk.insrt.coursework.zuul.entities.Entity;
import uk.insrt.coursework.zuul.entities.actions.IUseable;
import uk.insrt.coursework.zuul.world.Location;
import uk.insrt.coursework.zuul.world.Room;
import uk.insrt.coursework.zuul.world.World;

/**
 * Comms entity which is used to communicate between
 * Marie and the player during the mission.
 */
public class EntityComms extends EntityWithDialogue<String> implements IUseable {
    public EntityComms(World world, Location location) {
        super(world, location, 0.3d);
        this.setupDialogue();
    }

    @Override
    public void use(Entity target) {
        var w = (CampaignWorld) this.world;
        var io = w.getIO();

        if (w.getStoryFlags().getStage() == Stage.Stealth) {
            Room room = w.getPlayer().getRoom();
            if (room instanceof RoomMedicalCentreOffice) {
                this.dialogue.setNodeIfPresent("office");
            } else if (room instanceof RoomMedicalCentreReception) {
                if (((RoomMedicalCentreReception) room).getCouch().isSitting()) {
                    this.dialogue.setNodeIfPresent("in_position");
                } else {
                    this.dialogue.setNodeIfPresent("complaint");
                }
            } else {
                this.dialogue.setNodeIfPresent("orientation");
```

```java
42              }
43
44              this.dialogue.run(io);
45          } else {
46              io.println("<entities.comms.off>");
47          }
48      }
49
50      @Override
51      public void setupDialogue(Dialogue<String> dialogue) {
52          this.setupDialogueFromId(dialogue, "comms_marie");
53
54          // Make the guards disappear at this point.
55          dialogue.getPart("distraction")
56              .addOption(new DialogueOption<>(
57                  "<marie.comms.distraction.option_1>",
58                  io -> {
59                      ((RoomMedicalCentreReception) this.world.getRoom("Medical Centre: Reception"))
60                          .getGuard()
61                          .consume(false);
62
63                      return "coast_is_clear";
64                  }
65              ));
66      }
67
68      @Override
69      public String[] getAliases() {
70          return new String[] { "comms" };
71      }
72
73      @Override
74      public String describe() {
75          return "<entities.comms.description>";
76      }
77  }
```

```java
package uk.insrt.coursework.zuul.content.campaign.entities;

import uk.insrt.coursework.zuul.entities.Entity;
import uk.insrt.coursework.zuul.entities.EntityObject;
import uk.insrt.coursework.zuul.entities.actions.IUseable;
import uk.insrt.coursework.zuul.events.IEventListener;
import uk.insrt.coursework.zuul.events.world.EventEntityLeftRoom;
import uk.insrt.coursework.zuul.world.Location;
import uk.insrt.coursework.zuul.world.World;

/**
 * Couch present in the Medical Centre reception area.
 */
public class EntityCouch extends EntityObject implements IUseable, IEventListener<EventEntityLeftRoom> {
    private boolean isSitting;

    /**
     * Construct a new EntityCouch.
     * @param world World
     * @param location Location
     */
    public EntityCouch(World world, Location location) {
        super(world, location, Double.MAX_VALUE, "couch", "<entities.couch.description>");
    }

    @Override
    public void use(Entity target) {
        var io = this.getWorld().getIO();
        if (this.isSitting) {
            io.println("<entities.couch.sitting>");
        } else {
            io.println("<entities.couch.sit>");
            this.isSitting = true;
        }
    }

    @Override
    public void onEvent(EventEntityLeftRoom event) {
        this.isSitting = false;
    }
}
```

```java
    /**
     * Whether the player is sitting on the couch.
     * @return True if the player is sat down
     */
    public boolean isSitting() {
        return this.isSitting;
    }
}
```

```java
package uk.insrt.coursework.zuul.content.campaign.entities;

import uk.insrt.coursework.zuul.entities.Entity;
import uk.insrt.coursework.zuul.entities.EntityObject;
import uk.insrt.coursework.zuul.entities.actions.IUseable;
import uk.insrt.coursework.zuul.world.Location;
import uk.insrt.coursework.zuul.world.World;

/**
 * Documents which the player needs to find and take.
 */
public class EntityDocument extends EntityObject implements IUseable {
    private int count;

    /**
     * Construct a new EntityDocument
     * @param world World
     * @param location Location
     * @param count Document Id
     */
    public EntityDocument(World world, Location location, int count) {
        super(world, location, 10, "doc" + count, "<medical_centre_office.books." + count + ".title>");
        this.count = count;
    }

    @Override
    public void use(Entity target) {
        this.getWorld()
            .getIO()
            .println("<medical_centre_office.books." + count + ".contents>");
    }

    /**
     * Check whether these are the documents we want.
     * @return Whether we want this document
     */
    public boolean getIsValid() {
        switch (this.count) {
            case 2:
            case 4:
            case 5: return true;
```

```
42            }
43
44            return false;
45        }
46    }
```

```java
package uk.insrt.coursework.zuul.content.campaign.entities;

import java.awt.Desktop;
import java.net.URI;

import uk.insrt.coursework.zuul.content.campaign.CampaignWorld;
import uk.insrt.coursework.zuul.content.campaign.StoryFlags.Stage;
import uk.insrt.coursework.zuul.dialogue.Dialogue;
import uk.insrt.coursework.zuul.dialogue.DialogueNode;
import uk.insrt.coursework.zuul.dialogue.DialogueOption;
import uk.insrt.coursework.zuul.entities.Entity;
import uk.insrt.coursework.zuul.entities.Inventory;
import uk.insrt.coursework.zuul.entities.actions.IUseable;
import uk.insrt.coursework.zuul.world.Location;
import uk.insrt.coursework.zuul.world.World;

/**
 * This is the player's laptop which resides in their home.
 */
public class EntityLaptop extends EntityWithDialogue<String> implements IUseable {
    /**
     * Construct a new EntityLaptop.
     * @param world World
     * @param location Location
     */
    public EntityLaptop(World world, Location location) {
        super(world, location, 2);
        this.setupDialogue();
    }

    @Override
    public void use(Entity target) {
        this.dialogue.run(this.getWorld().getIO());
    }

    @Override
    public void setupDialogue(Dialogue<String> dialogue) {
        this.setupDialogueFromId(dialogue, "entity_laptop");

        // Add funny cat videos.
        this.dialogue.addPart("funny_cat_videos",
```

```java
42                new DialogueNode<String>("<entities.laptop.cat_videos.dialog>")
43                    .addOption(new DialogueOption<String>("<entities.laptop.cat_videos.option_q>",
44                        io -> {
45                            try {
46                                Desktop.getDesktop().browse(new URI("https://youtu.be/k35aiQPgNI4"));
47                            } catch (Exception e) { /* If we fail, just ignore this ever happen. */ }
48
49                            return "home";
50                        })));
51
52        // Story handler for sending documents to Marie.
53        this.dialogue.getPart("document")
54            .addOption(new DialogueOption<String>("<entities.laptop.document.option_1>",
55                io -> {
56                    var w = (CampaignWorld) this.getWorld();
57                    var flags = w.getStoryFlags();
58                    if (flags.getStage() != Stage.Stealth) {
59                        io.println("<marie.comms.no_access>");
60                        return "document";
61                    }
62
63                    Inventory inv = w.getPlayer().getInventory();
64                    int validPieces = 0;
65                    for (Entity item : inv.getItems()) {
66                        if (item instanceof EntityDocument) {
67                            if (((EntityDocument) item).getIsValid()) {
68                                validPieces++;
69                            }
70                        }
71                    }
72
73                    if (validPieces == 3) {
74                        io.println("<marie.comms.received>");
75                        flags.setStage(Stage.End);
76                        return null;
77                    }
78
79                    io.println("<marie.comms.bad_documents>");
80                    return "document";
81                }));
82    }
```

```java
83
84        @Override
85        public String[] getAliases() {
86            return new String[] { "laptop" };
87        }
88
89        @Override
90        public String describe() {
91            return "<entities.laptop.description>";
92        }
93    }
```

```java
package uk.insrt.coursework.zuul.content.campaign.entities;

import uk.insrt.coursework.zuul.content.campaign.CampaignWorld;
import uk.insrt.coursework.zuul.content.campaign.StoryFlags.Stage;
import uk.insrt.coursework.zuul.dialogue.Dialogue;
import uk.insrt.coursework.zuul.dialogue.DialogueOption;
import uk.insrt.coursework.zuul.entities.Entity;
import uk.insrt.coursework.zuul.entities.Inventory;
import uk.insrt.coursework.zuul.world.Location;
import uk.insrt.coursework.zuul.world.World;

/**
 * Marie Itami
 * https://brand-new-animal.fandom.com/wiki/Marie_Itami
 */
public class EntityMarie extends EntityNpc {
    /**
     * Construct a new EntityMarie.
     * @param world World
     * @param location Location
     */
    public EntityMarie(World world, Location location) {
        super(world, location,
            "npc_marie",
            "<marie.description>",
            new String[] { "marie", "itami", "mink" });
    }

    @Override
    public void setupDialogue(Dialogue<String> dialogue) {
        super.setupDialogue(dialogue);
        var w = (CampaignWorld) this.world;

        // Progress story if player accepts mission.
        dialogue.getPart("confirm")
            .addOption(new DialogueOption<>("<marie.alley.confirm.option_1>",
                io -> {
                    w.getStoryFlags()
                     .setStage(Stage.Recon);

                    return "recon";
```

```java
42                    }));
43            }
44
45            @Override
46            public void talk() {
47                if (this.dialogue.getCurrentNode().equals("waiting")) {
48                    var w = ((CampaignWorld) this.getWorld());
49                    Inventory inv = w.getPlayer().getInventory();
50                    for (Entity item : inv.getItems()) {
51                        if (item instanceof EntityComms) {
52                            this.dialogue.setNodeIfPresent("mission_brief");
53                            w.getStoryFlags().setStage(Stage.Stealth);
54                        }
55                    }
56                }
57
58                super.talk();
59            }
60    }
```

```java
package uk.insrt.coursework.zuul.content.campaign.entities;

import uk.insrt.coursework.zuul.dialogue.Dialogue;
import uk.insrt.coursework.zuul.entities.actions.ITalkwith;
import uk.insrt.coursework.zuul.world.Location;
import uk.insrt.coursework.zuul.world.World;

/**
 * NPC entity which provides dialog and can be talked with by the Player.
 */
public class EntityNpc extends EntityWithDialogue<String> implements ITalkwith {
    private String description;
    private String alias[];
    private String id;

    public EntityNpc(World world, Location location, String id, String description, String alias[]) {
        super(world, location, 75, null);

        this.description = description;
        this.alias = alias;
        this.id = id;

        this.setupDialogue();
    }

    public void talk() {
        this.dialogue.run(this.getWorld().getIO());
    }

    @Override
    public String[] getAliases() {
        return this.alias;
    }

    @Override
    public String describe() {
        return this.description;
    }

    @Override
    public void setupDialogue(Dialogue<String> dialogue) {
```

```
42            this.setupDialogueFromId(dialogue, id);
43        }
44    }
```

```java
package uk.insrt.coursework.zuul.content.campaign.entities;

import uk.insrt.coursework.zuul.content.campaign.CampaignWorld;
import uk.insrt.coursework.zuul.content.campaign.StoryFlags.Quest;
import uk.insrt.coursework.zuul.entities.Entity;
import uk.insrt.coursework.zuul.entities.actions.IGiveable;
import uk.insrt.coursework.zuul.world.Location;
import uk.insrt.coursework.zuul.world.World;

/**
 * The old man who is in the forest.
 */
public class EntityOldMan extends EntityNpc implements IGiveable {
    /**
     * Construct a new EntityOldMan.
     * @param world World
     * @param location Location
     */
    public EntityOldMan(World world, Location location) {
        super(world, location, "npc_old_man",
            "<forest.old_man.description>",
            new String[] { "oldman", "man" });

        this.inventory.setMaxWeight(EntityCat.WEIGHT);
    }

    @Override
    public void give(Entity item) {
        var io = this.getWorld().getIO();
        if (this.inventory.isFull()) {
            io.println("<forest.old_man.full>");
            return;
        }

        if (item instanceof EntityCat) {
            item.setLocation(this.inventory);
            this.dialogue.setNodeIfPresent("praise");
            ((CampaignWorld) this.getWorld())
                .getStoryFlags()
                .completeSideQuest(Quest.Cat);
```

```
42                io.println("<forest.old_man.accept>");
43            } else {
44                io.println("<forest.old_man.deny> " + item.getHighlightedName());
45            }
46        }
47    }
```

```java
package uk.insrt.coursework.zuul.content.campaign.entities;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

import uk.insrt.coursework.zuul.content.campaign.CampaignWorld;
import uk.insrt.coursework.zuul.content.campaign.StoryFlags.Stage;
import uk.insrt.coursework.zuul.dialogue.Dialogue;
import uk.insrt.coursework.zuul.dialogue.DialogueNode;
import uk.insrt.coursework.zuul.dialogue.DialogueOption;
import uk.insrt.coursework.zuul.entities.Entity;
import uk.insrt.coursework.zuul.entities.EntityObject;
import uk.insrt.coursework.zuul.entities.Inventory;
import uk.insrt.coursework.zuul.io.Ansi;
import uk.insrt.coursework.zuul.world.Location;
import uk.insrt.coursework.zuul.world.World;

/**
 * Shop keeper which the player can buy items from in the town.
 */
public class EntityShopkeeper extends EntityNpc {
    private HashMap<Stage, Entity[]> items;
    private HashMap<Entity, Integer> stock;
    private HashMap<Entity, Integer> price;
    private HashMap<Entity, IEntityFactory> entityFactory;

    /**
     * Construct a new EntityShopkeeper.
     * @param world World
     * @param location Location
     */
    public EntityShopkeeper(World world, Location location) {
        super(world, location,
            "npc_shopkeeper",
            "<shop.npc.description>",
            new String[] { "shopkeeper", "shop", "keeper" });

        this.items = new HashMap<>();
        this.stock = new HashMap<>();
        this.price = new HashMap<>();
```

```java
42             this.entityFactory = new HashMap<>();
43
44             this.createItems(world);
45         }
46
47         /**
48          * Interface implemented by entity factories for producing entities.
49          */
50         private interface IEntityFactory {
51             /**
52              * Produce a new Entity of a certain type.
53              * @return New entity
54              */
55             public Entity produce();
56         }
57
58         /**
59          * Generate all the items and populate the data.
60          * @param world World to place items in
61          */
62         private void createItems(World world) {
63             EntityObject itemBoatKey = new EntityBoatKey(world, new Location());
64             this.stock.put(itemBoatKey, 1);
65             this.price.put(itemBoatKey, 39_260);
66             this.entityFactory.put(itemBoatKey, () -> new EntityBoatKey(world, new Location()));
67
68             EntityComms itemComms = new EntityComms(world, new Location());
69             this.stock.put(itemComms, 3);
70             this.price.put(itemComms, 3_100);
71             this.entityFactory.put(itemComms, () -> new EntityComms(world, new Location()));
72
73             EntityObject itemCat = new EntityObject(world, new Location(), 5, "", "<shop.npc.fake_item.cat>");
74             this.stock.put(itemCat, 0);
75             this.price.put(itemCat, 21_300);
76
77             this.items.put(Stage.Exposition, new Entity[] {  });
78             this.items.put(Stage.Recon, new Entity[] { itemBoatKey, itemComms });
79             this.items.put(Stage.Stealth, new Entity[] { itemBoatKey, itemComms, itemCat });
80             this.items.put(Stage.End, new Entity[] { itemBoatKey, itemCat });
81         }
82
```

```java
 83        /**
 84         * Index dialogue node, displays things that the player can buy.
 85         * Implicitly takes the context of the EntityShopkeeper.
 86         */
 87        private class IndexNode extends DialogueNode<String> {
 88            /**
 89             * Construct a new IndexNode.
 90             * We may ignore the description since we override {@code #getDescription}.
 91             */
 92            public IndexNode() {
 93                super(null);
 94            }
 95
 96            @Override
 97            public String getDescription() {
 98                var w = (CampaignWorld) world;
 99                return "<shop.npc.greeting."
100                    + w.getStoryFlags().getStage().toString()
101                    + ">\n"
102                    + "<shop.npc.currently_have_amount_of_money> ¥ "
103                    + w.getStoryFlags().getBalance()
104                    + "\n";
105            }
106
107            @Override
108            protected List<DialogueOption<String>> getOptions() {
109                ArrayList<DialogueOption<String>> options = new ArrayList<>();
110
111                var w = (CampaignWorld) world;
112                var flags = w.getStoryFlags();
113                var player = w.getPlayer();
114
115                // Get all the items we can access at this story stage.
116                Entity[] list = items.get(flags.getStage());
117                for (Entity item : list) {
118                    int count = stock.get(item);
119                    int cost = price.get(item);
120                    IEntityFactory factory = entityFactory.get(item);
121
122                    // Add the option for this item.
123                    options.add(new DialogueOption<String>(
```

```java
124                     item.describe()
125                         + " ["
126                         + Ansi.Yellow
127                         + item.getWeight()
128                         + " kg"
129                         + Ansi.Reset
130                         + "] (¥ "
131                         + Ansi.Green
132                         + cost
133                         + Ansi.Reset
134                         + ") - "
135                         + (count == 0 ?
136                             "<shop.npc.out_of_stock>!"
137                             : count + " <shop.npc.x_left>"),
138                     io -> {
139                         // Check that this item is in stock.
140                         if (count == 0) {
141                             io.println("\n\n<shop.npc.item_out_of_stock.1> "
142                                 + Ansi.Red
143                                 + "<shop.npc.out_of_stock>"
144                                 + Ansi.Reset
145                                 + ", <shop.npc.item_out_of_stock.2>!");
146                         } else {
147                             // Make sure the player can hold this item without
148                             // going over their inventory weight limit.
149                             Inventory inv = player.getInventory();
150                             if (inv.getWeight() + item.getWeight() > inv.getMaxWeight()) {
151                                 io.println("\n\n<shop.npc.too_heavy>");
152                             } else {
153                                 // Try to deduct money from the player's money.
154                                 if (flags.deductFromBalance(cost)) {
155                                     io.println("\n\n<shop.npc.bought.1> "
156                                         + item.getHighlightedName()
157                                         + " <shop.npc.bought.2>!");
158
159                                     stock.put(item, count - 1);
160
161                                     Entity entity = factory.produce();
162                                     entity.setLocation(inv);
163                                 } else {
164                                     io.println("\n\n<shop.npc.not_enough> "
```

```
165                                    + item.getHighlightedName() + "!");
166                                }
167                            }
168                        }
169
170                        return "index";
171                    }
172                ));
173            }
174
175            options.add(new DialogueOption<String>("<shop.npc.leave>", "index").mustExit());
176            return options;
177        }
178    }
179
180    @Override
181    public void setupDialogue(Dialogue<String> dialogue) {
182        dialogue.addPart("index", new IndexNode());
183        dialogue.setNodeIfPresent("index");
184    }
185 }
```

```java
package uk.insrt.coursework.zuul.content.campaign.entities;

import uk.insrt.coursework.zuul.dialogue.Dialogue;
import uk.insrt.coursework.zuul.entities.Entity;
import uk.insrt.coursework.zuul.entities.actions.IUseable;
import uk.insrt.coursework.zuul.world.Location;
import uk.insrt.coursework.zuul.world.World;

/**
 * TV entity present in the player's room which they interact
 * with at the start of the game to learn more about the world.
 */
public class EntityTV extends EntityWithDialogue<String> implements IUseable {
    public EntityTV(World world, Location location) {
        super(world, location, 40);
        this.setupDialogue();
    }

    @Override
    public void use(Entity target) {
        this.dialogue.run(this.getWorld().getIO());
    }

    @Override
    public void setupDialogue(Dialogue<String> dialogue) {
        this.setupDialogueFromId(dialogue, "home_tv");
    }

    @Override
    public String[] getAliases() {
        return new String[] { "tv", "television" };
    }

    @Override
    public String describe() {
        return "<home.tv.description>";
    }
}
```

```java
package uk.insrt.coursework.zuul.content.campaign.entities;

import uk.insrt.coursework.zuul.content.campaign.CampaignWorld;
import uk.insrt.coursework.zuul.dialogue.Dialogue;
import uk.insrt.coursework.zuul.entities.Entity;
import uk.insrt.coursework.zuul.world.Location;
import uk.insrt.coursework.zuul.world.World;

/**
 * Abstract implementation of an entity which provides some sort of dialogue.
 */
public abstract class EntityWithDialogue<T> extends Entity {
    protected Dialogue<T> dialogue;

    /**
     * Construct a new EntityWithDialogue with a starting dialogue node
     * @param world Current World object
     * @param location Initial Location of this Entity
     * @param weight The weight (in kg) of this Entity
     * @param startNode The starting dialogue node
     */
    public EntityWithDialogue(World world, Location location, double weight, T startNode) {
        super(world, location, weight);

        Dialogue<T> dialogue = new Dialogue<T>(startNode);
        this.dialogue = dialogue;
    }

    /**
     * Construct a new EntityWithDialogue without a starting dialogue node
     * @param world Current World object
     * @param location Initial Location of this Entity
     * @param weight The weight (in kg) of this Entity
     */
    public EntityWithDialogue(World world, Location location, double weight) {
        this(world, location, weight, null);
    }

    /**
     * Configure this Entity's dialogue,
     * create nodes and options to add to this Entity.
```

```java
 42         * @param dialogue Entity Dialogue
 43         */
 44        public abstract void setupDialogue(Dialogue<T> dialogue);
 45
 46        /**
 47         * Configure dialogue.
 48         */
 49        public void setupDialogue() {
 50            this.setupDialogue(this.dialogue);
 51        }
 52
 53        /**
 54         * Use the CampaignWorld's DialogueLoader to populate this Entity's Dialogue
 55         * @param dialogue Entity Dialogue
 56         * @param id Target dialogue ID in file
 57         */
 58        public void setupDialogueFromId(Dialogue<String> dialogue, String id) {
 59            var world = (CampaignWorld) this.getWorld();
 60            world.getDialogueLoader().populate(dialogue, id);
 61        }
 62
 63        /**
 64         * Set the current dialogue node if the given node is present.
 65         * @param node Target node
 66         */
 67        public void setDialogueNodeIfPresent(Object node) {
 68            try {
 69                @SuppressWarnings("unchecked")
 70                T n = (T) node;
 71
 72                this.dialogue.setNodeIfPresent(n);
 73            } catch (ClassCastException ex) {
 74                // Ignore the error since if we can't cast it
 75                // to whatever type this is, then obviously this
 76                // node is not present within this Dialogue.
 77            }
 78        }
 79    }
```

```java
package uk.insrt.coursework.zuul.content.campaign.events;

import uk.insrt.coursework.zuul.content.campaign.StoryFlags.Stage;
import uk.insrt.coursework.zuul.events.Event;

/**
 * Event fired when the story stage (chapter) changes.
 */
public class EventGameStageChanged extends Event {
    private Stage stage;

    /**
     * Construct a new GameStageChanged event.
     * @param stage New stage
     */
    public EventGameStageChanged(Stage stage) {
        this.stage = stage;
    }

    /**
     * Get the new game stage.
     * @return Stage
     */
    public Stage getStage() {
        return this.stage;
    }
}
```

```java
package uk.insrt.coursework.zuul.content.campaign.rooms;

import uk.insrt.coursework.zuul.content.campaign.CampaignWorld;
import uk.insrt.coursework.zuul.world.Room;
import uk.insrt.coursework.zuul.world.World;

/**
 * Class which overrides getWorld to instead provide the CampaignWorld.
 */
public abstract class CampaignRoom extends Room {
    /**
     * Construct a new CampaignRoom.
     * @param world World
     * @param name Internal name used to refer to this Room
     */
    public CampaignRoom(World world, String name) {
        super(world, name);
    }

    @Override
    public CampaignWorld getWorld() {
        return (CampaignWorld) super.getWorld();
    }
}
```

```java
package uk.insrt.coursework.zuul.content.campaign.rooms;

import uk.insrt.coursework.zuul.content.campaign.entities.EntityBed;
import uk.insrt.coursework.zuul.content.campaign.entities.EntityLaptop;
import uk.insrt.coursework.zuul.content.campaign.entities.EntityTV;
import uk.insrt.coursework.zuul.world.Direction;
import uk.insrt.coursework.zuul.world.World;

/**
 * The player's home in the apartments complex.
 */
public class RoomApartmentsHome extends CampaignRoom {
    public RoomApartmentsHome(World world) {
        super(world, "Apartments: Home");
    }

    @Override
    public String describe() {
        var world = this.getWorld();
        if (!world.hasVisited(this)) {
            return "<home.first_load>";
        }

        return "<home.enter>";
    }

    @Override
    protected void setupDirections() {
        this.setAdjacent(Direction.DOWN, this.getWorld().getRoom("Apartments: Reception"));
    }

    @Override
    public void spawnEntities() {
        World world = this.getWorld();

        world.spawnEntity("tv", new EntityTV(world, this.toLocation()));
        world.spawnEntity("bed", new EntityBed(world, this.toLocation()));
        world.spawnEntity("laptop", new EntityLaptop(world, this.toLocation()));
    }
}
```

```java
package uk.insrt.coursework.zuul.content.campaign.rooms;

import uk.insrt.coursework.zuul.content.campaign.entities.EntityNpc;
import uk.insrt.coursework.zuul.world.Direction;
import uk.insrt.coursework.zuul.world.World;

/**
 * The reception of the apartments complex.
 */
public class RoomApartmentsReception extends CampaignRoom {
    public RoomApartmentsReception(World world) {
        super(world, "Apartments: Reception");
    }

    @Override
    public String describe() {
        return "<apartments.enter>";
    }

    @Override
    protected void setupDirections() {
        World world = this.getWorld();
        this.setAdjacent(Direction.NORTH, world.getRoom("Street"));
        this.setAdjacent(Direction.EAST, world.getRoom("City Centre"));
        this.setAdjacent(Direction.UP, world.getRoom("Apartments: Home"));
    }

    @Override
    public void spawnEntities() {
        World world = this.getWorld();
        world.spawnEntity("receptionist",
            new EntityNpc(
                world,
                this.toLocation(),
                "npc_receptionist",
                "<apartments.receptionist.description>",
                new String[] { "receptionist" }
        ));
    }
}
```

```java
package uk.insrt.coursework.zuul.content.campaign.rooms;

import uk.insrt.coursework.zuul.content.campaign.entities.EntityMarie;
import uk.insrt.coursework.zuul.world.Direction;
import uk.insrt.coursework.zuul.world.World;

/**
 * The back alley in the North East side of the map.
 */
public class RoomBackAlley extends CampaignRoom {
    public RoomBackAlley(World world) {
        super(world, "Back Alley");
    }

    @Override
    public String describe() {
        var world = this.getWorld();
        if (!world.hasVisited(this)) {
            return "<back_alley.first_load>";
        }

        return "<back_alley.enter>";
    }

    @Override
    protected void setupDirections() {
        this.setAdjacent(Direction.SOUTH, this.getWorld().getRoom("City Centre"));
    }

    @Override
    public void spawnEntities() {
        World world = this.getWorld();
        world.spawnEntity("npc_marie", new EntityMarie(world, this.toLocation()));
    }
}
```

```java
package uk.insrt.coursework.zuul.content.campaign.rooms;

import uk.insrt.coursework.zuul.content.campaign.entities.EntityCat;
import uk.insrt.coursework.zuul.content.campaign.entities.EntityNpc;
import uk.insrt.coursework.zuul.world.Direction;
import uk.insrt.coursework.zuul.world.Room;
import uk.insrt.coursework.zuul.world.World;

/**
 * The city centre connecting most major locations.
 */
public class RoomCityCentre extends CampaignRoom {
    public RoomCityCentre(World world) {
        super(world, "City Centre");
    }

    @Override
    public String describe() {
        var world = this.getWorld();
        if (!world.hasVisited(this)) {
            return "<city_centre.first_load>";
        }

        return "<city_centre.enter>";
    }

    @Override
    protected void setupDirections() {
        World world = this.getWorld();
        this.setAdjacent(Direction.NORTH, world.getRoom("Back Alley"));
        this.setAdjacent(Direction.NORTH_WEST, world.getRoom("Street"));
        this.setAdjacent(Direction.WEST, world.getRoom("Apartments: Reception"));
        this.setAdjacent(Direction.SOUTH, world.getRoom("Coastline"));
    }

    @Override
    public void spawnEntities() {
        World world = this.getWorld();

        EntityCat cat = new EntityCat(world, this.toLocation());
        world.spawnEntity("cat", cat);
```

```java
42            cat.useWanderAI(
43                new Room[] {
44                    world.getRoom("City Centre"),
45                    world.getRoom("Street"),
46                    world.getRoom("Shop"),
47                    world.getRoom("Street"),
48                    world.getRoom("City Centre"),
49                    world.getRoom("Back Alley"),
50                    world.getRoom("City Centre")
51                },
52                8
53            );
54
55        world.spawnEntity("city_npc",
56            new EntityNpc(
57                world,
58                this.toLocation(),
59                "npc_city_centre",
60                "<city_centre.npc.description>",
61                new String[] { "stranger", "person", "people" }
62            ));
63    }
64 }
```

```java
package uk.insrt.coursework.zuul.content.campaign.rooms;

import uk.insrt.coursework.zuul.content.campaign.entities.EntityBoat;
import uk.insrt.coursework.zuul.world.Direction;
import uk.insrt.coursework.zuul.world.World;

/**
 * The coast which connects the main city to the mainland.
 */
public class RoomCoastline extends CampaignRoom {
    public RoomCoastline(World world) {
        super(world, "Coastline");
    }

    @Override
    public String describe() {
        return "<coastline.enter>";
    }

    @Override
    protected void setupDirections() {
        this.setAdjacent(Direction.NORTH, this.getWorld().getRoom("City Centre"));
    }

    @Override
    public void spawnEntities() {
        World world = this.getWorld();
        world.spawnEntity("boat1",
            new EntityBoat(world, this.toLocation(),
                world.getRoom("Mainland: Coastline")));
    }
}
```

```java
package uk.insrt.coursework.zuul.content.campaign.rooms;

import uk.insrt.coursework.zuul.content.campaign.entities.EntityOldMan;
import uk.insrt.coursework.zuul.world.Direction;
import uk.insrt.coursework.zuul.world.World;

/**
 * A forest on the mainland side.
 */
public class RoomForest extends CampaignRoom {
    public RoomForest(World world) {
        super(world, "Forest");
    }

    @Override
    public String describe() {
        return "<forest.enter>";
    }

    @Override
    protected void setupDirections() {
        World world = this.getWorld();
        this.setAdjacent(Direction.NORTH, world.getRoom("Mainland: Coastline"));
        this.setAdjacent(Direction.EAST, world.getRoom("Worm Hole"));
    }

    @Override
    public void spawnEntities() {
        World world = this.getWorld();
        world.spawnEntity("npc_old_man", new EntityOldMan(world, this.toLocation()));
    }
}
```

```java
package uk.insrt.coursework.zuul.content.campaign.rooms;

import uk.insrt.coursework.zuul.content.campaign.entities.EntityBoat;
import uk.insrt.coursework.zuul.world.Direction;
import uk.insrt.coursework.zuul.world.World;

/**
 * The coast which connects the mainland to the main city.
 */
public class RoomMainlandCoastline extends CampaignRoom {
    public RoomMainlandCoastline(World world) {
        super(world, "Mainland: Coastline");
    }

    @Override
    public String describe() {
        return "<mainland_coastline.enter>";
    }

    @Override
    protected void setupDirections() {
        this.setAdjacent(Direction.SOUTH, this.getWorld().getRoom("Forest"));
    }

    @Override
    public void spawnEntities() {
        World world = this.getWorld();
        world.spawnEntity("boat2",
            new EntityBoat(world, this.toLocation(),
                world.getRoom("Coastline")));
    }
}
```

```java
package uk.insrt.coursework.zuul.content.campaign.rooms;

import uk.insrt.coursework.zuul.content.campaign.entities.EntityDocument;
import uk.insrt.coursework.zuul.world.Direction;
import uk.insrt.coursework.zuul.world.World;

/**
 * Private, usually inaccessible room within the Medical Centre complex.
 */
public class RoomMedicalCentreOffice extends CampaignRoom {
    public RoomMedicalCentreOffice(World world) {
        super(world, "Medical Centre: Office");
    }

    @Override
    public String describe() {
        return "<medical_centre_office.enter>";
    }

    @Override
    protected void setupDirections() {
        this.setAdjacent(Direction.UP, this.getWorld().getRoom("Medical Centre: Reception"));
    }

    @Override
    public void spawnEntities() {
        World world = this.getWorld();
        for (int i=1;i<=6;i++) {
            world.spawnEntity("doc" + i, new EntityDocument(world, this.toLocation(), i));
        }
    }
}
```

```java
package uk.insrt.coursework.zuul.content.campaign.rooms;

import uk.insrt.coursework.zuul.content.campaign.entities.EntityCouch;
import uk.insrt.coursework.zuul.content.campaign.entities.EntityNpc;
import uk.insrt.coursework.zuul.entities.Entity;
import uk.insrt.coursework.zuul.events.world.EventEntityLeftRoom;
import uk.insrt.coursework.zuul.world.Direction;
import uk.insrt.coursework.zuul.world.World;

/**
 * Reception of the Medical Centre complex.
 */
public class RoomMedicalCentreReception extends CampaignRoom {
    private Entity guardEntity;
    private EntityCouch couchEntity;

    public RoomMedicalCentreReception(World world) {
        super(world, "Medical Centre: Reception");
    }

    @Override
    public String describe() {
        return "<medical_centre.enter>";
    }

    @Override
    protected void setupDirections() {
        World world = this.getWorld();
        this.setAdjacent(Direction.EAST, world.getRoom("Street"));
        this.setAdjacent(Direction.DOWN, world.getRoom("Medical Centre: Office"));
    }

    @Override
    public boolean canLeave(Direction direction) {
        if (direction == Direction.DOWN) {
            if (this.guardEntity.getRoom() == this) {
                this.getWorld()
                    .getIO()
                    .println("<medical_centre.guard.blocking>");

                return false;
```

```java
42                }
43            }
44
45            return true;
46        }
47
48        @Override
49        public void spawnEntities() {
50            World world = this.getWorld();
51
52            this.guardEntity = new EntityNpc(
53                world,
54                this.toLocation(),
55                "npc_security_guard",
56                "<medical_centre.guard.description>",
57                new String[] { "guard", "security" }
58            );
59            world.spawnEntity("npc_guard", this.guardEntity);
60
61            this.couchEntity = new EntityCouch(world, this.toLocation());
62            world.spawnEntity("couch", this.couchEntity);
63            world.getEventSystem().addListener(EventEntityLeftRoom.class, this.couchEntity);
64        }
65
66        public EntityCouch getCouch() {
67            return this.couchEntity;
68        }
69
70        public Entity getGuard() {
71            return this.guardEntity;
72        }
73    }
```

```java
package uk.insrt.coursework.zuul.content.campaign.rooms;

import uk.insrt.coursework.zuul.content.campaign.entities.EntityNpc;
import uk.insrt.coursework.zuul.content.campaign.entities.EntityShopkeeper;
import uk.insrt.coursework.zuul.world.Direction;
import uk.insrt.coursework.zuul.world.World;

/**
 * A shop within the city, the only one the player can interact with.
 */
public class RoomShop extends CampaignRoom {
    public RoomShop(World world) {
        super(world, "Shop");
    }

    @Override
    public String describe() {
        return "<shop.enter>";
    }

    @Override
    protected void setupDirections() {
        this.setAdjacent(Direction.SOUTH, this.getWorld().getRoom("Street"));
    }

    @Override
    public void spawnEntities() {
        World world = this.getWorld();
        world.spawnEntity("npc_shopkeeper",
            new EntityShopkeeper(world, this.toLocation()));
    }
}
```

```java
package uk.insrt.coursework.zuul.content.campaign.rooms;

import uk.insrt.coursework.zuul.content.campaign.StoryFlags.Stage;
import uk.insrt.coursework.zuul.content.campaign.entities.EntityNpc;
import uk.insrt.coursework.zuul.content.campaign.events.EventGameStageChanged;
import uk.insrt.coursework.zuul.entities.Entity;
import uk.insrt.coursework.zuul.events.IEventListener;
import uk.insrt.coursework.zuul.world.Direction;
import uk.insrt.coursework.zuul.world.World;

/**
 * One of the major connecting points between locations in the city.
 */
public class RoomStreet extends CampaignRoom implements IEventListener<EventGameStageChanged> {
    private Entity protestorsEntity;

    public RoomStreet(World world) {
        super(world, "Street");
    }

    @Override
    public String describe() {
        var world = this.getWorld();
        if (!world.hasVisited(this)) {
            return "<street.first_load>";
        }

        return "<street.enter>";
    }

    @Override
    protected void setupDirections() {
        World world = this.getWorld();
        this.setAdjacent(Direction.SOUTH, world.getRoom("Apartments: Reception"));
        this.setAdjacent(Direction.EAST, world.getRoom("City Centre"));
        this.setAdjacent(Direction.NORTH, world.getRoom("Shop"));
        this.setAdjacent(Direction.WEST, world.getRoom("Medical Centre: Reception"));
    }

    @Override
    public boolean canLeave(Direction direction) {
```

```java
42          if (direction == Direction.WEST) {
43              if (this.protestorsEntity.getRoom() == this) {
44                  this.getWorld()
45                      .getIO()
46                      .println("<street.protestors.blocking>");
47
48                  return false;
49              }
50          }
51
52          return true;
53      }
54
55      @Override
56      public void spawnEntities() {
57          World world = this.getWorld();
58          this.protestorsEntity = new EntityNpc(
59              world,
60              this.toLocation(),
61              "npc_protestors",
62              "<street.protestors.description>",
63              new String[] { "protestors", "protestor" }
64          );
65          world.spawnEntity("npc_protestors", this.protestorsEntity);
66      }
67
68      @Override
69      public void onEvent(EventGameStageChanged event) {
70          // Remove the protestors if we are in Stealth chapter.
71          if (event.getStage() == Stage.Stealth) {
72              this.protestorsEntity.consume(false);
73          } else {
74              this.protestorsEntity.setLocation(this);
75          }
76      }
77  }
```

```java
package uk.insrt.coursework.zuul.content.campaign.rooms;

import java.util.Random;

import uk.insrt.coursework.zuul.entities.Entity;
import uk.insrt.coursework.zuul.entities.EntityPlayer;
import uk.insrt.coursework.zuul.events.IEventListener;
import uk.insrt.coursework.zuul.events.world.EventEntityEnteredRoom;
import uk.insrt.coursework.zuul.world.Room;
import uk.insrt.coursework.zuul.world.World;

/**
 * Teleporter room implemented as required by the challenge tasks.
 * Any Entity that walks into the worm hole is transported into a random public location.
 */
public class RoomWormHole extends CampaignRoom implements IEventListener<EventEntityEnteredRoom> {
    public RoomWormHole(World world) {
        super(world, "Worm Hole");
    }

    @Override
    public String describe() {
        return "";
    }

    @Override
    protected void setupDirections() {}

    @Override
    public void onEvent(EventEntityEnteredRoom event) {
        Entity entity = event.getEntity();
        Room room = entity.getRoom();
        if (room != this) return;
        event.stopPropagation();

        // This is a restricted set of locations as to not break
        // the game's plot, say if we were transported to the medical
        // centre complex office when we're not meant to go there yet.
        final Random random = new Random();
        final String[] locations = {
            "City Centre",
```

```java
                "Coastline",
                "Mainland: Coastline",
                "Forest",
                "Street",
                "Back Alley"
            };

        var io = this.getWorld().getIO();
        io.println("\n<worm_hole.enter>");

        try {
            Thread.sleep(1000);

            final int WIDTH = 79;

            // Transport animation, this will take 1800 ms.
            for (int i=0;i<5;i++) {
                io.println("*".repeat(i*3) + "\\"
                    + " ".repeat(WIDTH - i * 6 - 2) + "/" + "*".repeat(i*3));

                Thread.sleep(60);
            }

            for (int i=0;i<30;i++) {
                var out = "";
                for (int j=0;j<WIDTH;j++) {
                    out += random.nextInt(8) == 0 ? "*" : " ";
                }

                io.println(out);
                Thread.sleep(40);
            }

            for (int i=5;i>0;i--) {
                io.println("*".repeat(i*3) + "/"
                    + " ".repeat(WIDTH - i * 6 - 2) + "\\" + "*".repeat(i*3));

                Thread.sleep(60);
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
```

```
83                io.println("There was a disruption when travelling.");
84            }
85
86        io.print("\n");
87
88        // Pick a random location and put the entering entity in it.
89        String location = locations[random.nextInt(locations.length)];
90        Room target = this.getWorld().getRoom(location);
91        entity.setLocation(target);
92
93        // If it was the player, clear their walk history.
94        if (entity instanceof EntityPlayer) {
95            ((EntityPlayer) entity).clearHistory();
96        }
97    }
98 }
```

```java
package uk.insrt.coursework.zuul.content.campaign;

import java.util.HashSet;

import uk.insrt.coursework.zuul.content.campaign.events.EventGameStageChanged;
import uk.insrt.coursework.zuul.events.EventSystem;
import uk.insrt.coursework.zuul.events.world.EventTick;

/**
 * Class which controls story progression within the Campaign World.
 */
public class StoryFlags {
    /**
     * The current story chapter.
     */
    public enum Stage {
        Exposition, // Ch 1.
        Recon, // Ch 2.
        Stealth, // Ch 3.
        End, // Current Ending

        Twist, // Ch 4. Skipped
        Conclusion, // Ch 5. Skipped
    }

    /**
     * Side-quests available in the game.
     */
    public enum Quest {
        Cat
    }

    private EventSystem eventSystem;
    private HashSet<Quest> quests;
    private int balance;
    private Stage stage;
    private int ticks;

    /**
     * Construct a new instance of StoryFlags
     * @param eventSystem World event system
```

```java
     */
    public StoryFlags(EventSystem eventSystem) {
        this.eventSystem = eventSystem;
        this.stage = Stage.Exposition;
        this.quests = new HashSet<>();
        this.balance = 100_000;
        this.ticks = 0;

        this.eventSystem.addListener(EventTick.class, e -> this.ticks++);
    }

    /**
     * Get the current stage (chapter) of the story.
     * @return Current stage
     */
    public Stage getStage() {
        return this.stage;
    }

    /**
     * Set the current stage (chapter) of the story.
     * @param stage New stage
     */
    public void setStage(Stage stage) {
        this.stage = stage;
        this.eventSystem.emit(new EventGameStageChanged(stage));
    }

    /**
     * Get the player's balance
     * @return Player's balance
     */
    public int getBalance() {
        return this.balance;
    }

    /**
     * Set player's new balance.
     * @param balance New balance
     */
    public void setBalance(int balance) {
```

```java
 83            this.balance = balance;
 84        }
 85
 86        /**
 87         * Deduct money from the player's balance.
 88         * @param value Amount to deduct
 89         * @return Whether we could deduct the balance without going below zero
 90         */
 91        public boolean deductFromBalance(int value) {
 92            if (value > this.balance) {
 93                return false;
 94            }
 95
 96            this.balance -= value;
 97            return true;
 98        }
 99
100        /**
101         * Get ticks since start of the World.
102         * @return Number of ticks since start
103         */
104        public int getTicks() {
105            return this.ticks;
106        }
107
108        /**
109         * Mark a side-quest as complete
110         */
111        public void completeSideQuest(Quest quest) {
112            this.quests.add(quest);
113        }
114
115        /**
116         * Get completed side-quests.
117         * @return Number of completed side-quests
118         */
119        public int getCompletedQuests() {
120            return this.quests.size();
121        }
122
123        /**
```

```java
         * Get total number of side-quests.
         * @return Total number of side-quests
         */
        public int getTotalQuests() {
            return Quest.values().length;
        }
    }
```

```java
package uk.insrt.coursework.zuul.dialogue;

import java.util.HashMap;

import uk.insrt.coursework.zuul.io.IOSystem;

/**
 * Simple dialogue engine which navigates between {@link DialogueNode}(s).
 */
public class Dialogue<T> {
    private HashMap<T, DialogueNode<T>> parts;
    private T currentNode;

    /**
     * Contruct a new Dialogue engine.
     */
    public Dialogue() {
        this.parts = new HashMap<>();
    }

    /**
     * Construct a new Dialogue engine and initialise us at a starting node.
     * @param start Starting node
     */
    public Dialogue(T start) {
        this();
        this.currentNode = start;
    }

    /**
     * Get the current node
     */
    public T getCurrentNode() {
        return this.currentNode;
    }

    /**
     * Set the current node
     * @param node New node
     */
    public void getCurrentNode(T node) {
```

```java
42            this.currentNode = node;
43        }
44
45        /**
46         * Change the current node to a different one if it exists
47         * @param node New node
48         */
49        public void setNodeIfPresent(T node) {
50            if (this.parts.containsKey(node)) {
51                this.currentNode = node;
52            }
53        }
54
55        /**
56         * Add a new part to the dialogue
57         * @param part What this node is identified by
58         * @param node The new node
59         */
60        public void addPart(T part, DialogueNode<T> node) {
61            this.parts.put(part, node);
62        }
63
64        /**
65         * Get an existing part from the dialogue
66         * @param part What the node is identified by
67         * @return The node if it exists, otherwise null
68         */
69        public DialogueNode<T> getPart(T part) {
70            return this.parts.get(part);
71        }
72
73        /**
74         * Run the Dialogue engine until one of the options exits us out
75         * @param io Provided IO system
76         */
77        public void run(IOSystem io) {
78            var part = this.parts.get(this.currentNode);
79            io.println("\n" + part.getDescription());
80            DialogueOption<T> option = part.pickOption(io);
81
82            T target = option.handle(io);
```

```java
83              if (target == null) {
84                  T newTarget = option.getTarget();
85                  if (newTarget != null) {
86                      this.currentNode = newTarget;
87                  }
88
89
90                  return;
91              }
92
93          this.currentNode = target;
94          this.run(io);
95      }
96  }
```

```java
package uk.insrt.coursework.zuul.dialogue;

import java.io.IOException;
import java.io.InputStream;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;

import com.moandjiezana.toml.Toml;

/**
 * This is a helper class for loading and populating {@link Dialogue}.
 * This DialogueLoader assumes that i18n is being used in the dialogue data.
 */
public class DialogueLoader {
    private Map<String, Object> data;

    /**
     * Construct a new DialogueLoader
     */
    public DialogueLoader() {
        this.data = new HashMap<>();
    }

    /**
     * Load all necessary data for populating Dialogue.
     * @param path Path to the dialogue resource file
     * @throws IOException if we can't read the dialogue file
     */
    public void load(String path) throws IOException {
        InputStream stream = DialogueLoader.class.getResourceAsStream(path);
        this.data = new Toml().read(stream).toMap();
    }

    /**
     * Populate a Dialogue system using a specific dialog definition represented by a given key.
     *
     * This method is unchecked as we expect a valid data structure to have
     * been loaded from the resource, this should be verified by the developer.
     * @param dialogue Dialogue system
```

```java
42          * @param key Key to lookup
43          */
44         @SuppressWarnings("unchecked")
45         public void populate(Dialogue<String> dialogue, String key) {
46             Map<String, Object> nodes = (Map<String, Object>) this.data.get(key);
47
48             // Process any special keys first before we continue.
49             String prefix = "";
50             for (String nodeKey : nodes.keySet()) {
51                 if (nodeKey.equals("_prefix")) {
52                     prefix = (String) nodes.get(nodeKey);
53                 } else if (nodeKey.equals("_start")) {
54                     dialogue.getCurrentNode((String) nodes.get(nodeKey));
55                 }
56             }
57
58             for (Entry<String, Object> node : nodes.entrySet()) {
59                 // Ignore any keys starting with _, as they are used above.
60                 String nodeKey = node.getKey();
61                 if (nodeKey.startsWith("_")) continue;
62
63                 // Read each node's values and find the description and options.
64                 Map<String, Object> values = (Map<String, Object>) node.getValue();
65
66                 // Description strings are assumed to be i18n paths.
67                 String description = "<" + prefix + (String) values.get("description") + ">";
68                 List<Map<String, Object>> options = (List<Map<String, Object>>) values.get("options");
69
70                 // Construct a new Dialogue Node with the given data.
71                 DialogueNode<String> dialogueNode = new DialogueNode<>(description);
72                 for (Map<String, Object> object : options) {
73                     String desc = "<" + prefix + (String) object.get("description") + ">";
74                     String to = (String) object.get("to");
75                     Boolean mustExit = (Boolean) object.get("mustExit");
76
77                     if (mustExit == null) {
78                         dialogueNode.addOption(desc, to);
79                     } else {
80                         dialogueNode.addOption(desc, to, mustExit);
81                     }
82                 }
```

```
83
84                dialogue.addPart(nodeKey, dialogueNode);
85            }
86        }
87    }
```

```java
package uk.insrt.coursework.zuul.dialogue;

import java.util.ArrayList;
import java.util.List;

import uk.insrt.coursework.zuul.io.IOSystem;

/**
 * A node in a {@link Dialogue} system.
 */
public class DialogueNode<T> {
    private String description;
    private ArrayList<DialogueOption<T>> options;

    /**
     * Construct a new node.
     * @param description Description of this node
     */
    public DialogueNode(String description) {
        this.description = description;
        this.options = new ArrayList<>();
    }

    /**
     * Get this node's description
     * @return Description string
     */
    public String getDescription() {
        return this.description;
    }

    /**
     * Add a new option which branches off this node.
     * @param option Dialogue Option
     * @return This Dialogue Node so other method calls can be chained
     */
    public DialogueNode<T> addOption(DialogueOption<T> option) {
        this.options.add(option);
        return this;
    }
```

```java
42          /**
43           * Create a new option which branches off this node.
44           * @param description Description of this option
45           * @param stage The next stage of the dialogue this should jump to
46           * @param mustExit Whether we must exit from the dialogue after selecting this option
47           * @return This Dialogue Node so other method calls can be chained
48           */
49          public DialogueNode<T> addOption(String description, T stage, boolean mustExit) {
50              var option = new DialogueOption<T>(description, stage);
51              if (mustExit) option.mustExit();
52              this.options.add(option);
53              return this;
54          }
55
56          /**
57           * Create a new option which branches off this node.
58           * @param description Description of this option
59           * @param stage The next stage of the dialogue this should jump to
60           * @return This Dialogue Node so other method calls can be chained
61           */
62          public DialogueNode<T> addOption(String description, T stage) {
63              return this.addOption(description, stage, false);
64          }
65
66          /**
67           * Get the options available to this node.
68           * @return List of options
69           */
70          protected List<DialogueOption<T>> getOptions() {
71              return this.options;
72          }
73
74          /**
75           * Ask the player to pick one of the valid options branching off this node.
76           * @param io Provided IO system
77           * @return The selected option
78           */
79          public DialogueOption<T> pickOption(IOSystem io) {
80              List<DialogueOption<T>> options = this.getOptions();
81              for (int i=0;i<options.size();i++) {
82                  io.println((i + 1) + ". " + options.get(i).getDescription());
```

```java
83            }

85            while (true) {
86                io.print("Choice: ");
87                String value = io.readLine();
88                try {
89                    int v = Integer.parseInt(value);
90                    if (v < 1 || v > options.size()) {
91                        io.println("Provide a valid option!");
92                        continue;
93                    }

95                    return options.get(v - 1);
96                } catch (Exception e) {
97                    io.println("Provide a valid number!");
98                }
99            }
100        }
101    }
```

```java
package uk.insrt.coursework.zuul.dialogue;

import uk.insrt.coursework.zuul.io.IOSystem;

/**
 * An option which branches off a {@link DialogueNode} into another node.
 */
public class DialogueOption<T> {
    private IDialogueHandler<T> handler;

    private String description;
    private boolean shouldExit;
    private T target;

    /**
     * Construct a new simple DialogueOption with a description and destination.
     * @param description Description of this option
     * @param target Target node to jump to
     */
    public DialogueOption(String description, T target) {
        this.target = target;
        this.description = description;
    }

    /**
     * Construct a complex DialogueOption with a description and select handler.
     * @param description Description of this option
     * @param handler Method called when this option is selected
     */
    public DialogueOption(String description, IDialogueHandler<T> handler) {
        this.handler = handler;
        this.description = description;
    }

    /**
     * Tell the Dialogue system to exit if this option is selected.
     * @return This dialogue option so method calls can be chained
     */
    public DialogueOption<T> mustExit() {
        this.shouldExit = true;
        return this;
```

```java
42          }
43
44          /**
45           * Get the description of this option.
46           * @return Description string
47           */
48          public String getDescription() {
49              return this.description;
50          }
51
52          /**
53           * Get the destination of this option.
54           * @return Destination if it exists
55           */
56          public T getTarget() {
57              return this.target;
58          }
59
60          /**
61           * Handle the player selecting this dialogue option.
62           * @param io Provided IO system
63           * @return The new node or null if we should exit and stay put.
64           */
65          public T handle(IOSystem io) {
66              if (this.handler != null) {
67                  return this.handler.onAction(io);
68              } else if (!this.shouldExit) {
69                  return this.target;
70              }
71
72              return null;
73          }
74      }
```

```java
package uk.insrt.coursework.zuul.dialogue;

import uk.insrt.coursework.zuul.io.IOSystem;

/**
 * Interface implemented to provide an onAction method.
 */
public interface IDialogueHandler<T> {
    /**
     * Handle the selection of a dialogue option.
     * @param io Provided IO system
     * @return Destination node, may be null
     */
    public T onAction(IOSystem io);
}
```

```java
package uk.insrt.coursework.zuul.entities.actions;

import uk.insrt.coursework.zuul.entities.Entity;

/**
 * Interface implemented to provide the ability for
 * an Entity to have other Entities given to them.
 */
public interface IGiveable {
    /**
     * Give this entity another entity.
     * @param item Item being given
     */
    public void give(Entity item);
}
```

```java
package uk.insrt.coursework.zuul.entities.actions;

/**
 * Interface implemented to provide the
 * ability for an Entity to be pet.
 */
public interface IPettable {
    /**
     * Pet this entity.
     */
    public void pet();
}
```

```java
package uk.insrt.coursework.zuul.entities.actions;

/**
 * Interface implemented to provide the
 * ability for an Entity to talk with the player.
 */
public interface ITalkwith {
    /**
     * Talk with this entity.
     */
    public void talk();
}
```

```java
package uk.insrt.coursework.zuul.entities.actions;

import uk.insrt.coursework.zuul.entities.Entity;

/**
 * Interface implemented to provide the
 * ability for an Entity to be used by the player.
 */
public interface IUseable {
    /**
     * Use this entity.
     * @param target The Entity taking this entity.
     */
    public void use(Entity target);
}
```

```java
package uk.insrt.coursework.zuul.entities;

import uk.insrt.coursework.zuul.events.world.EventEntityEnteredRoom;
import uk.insrt.coursework.zuul.events.world.EventEntityLeftRoom;
import uk.insrt.coursework.zuul.io.Ansi;
import uk.insrt.coursework.zuul.world.Location;
import uk.insrt.coursework.zuul.world.Room;
import uk.insrt.coursework.zuul.world.World;

/**
 * Representation of any Entity in the World.
 *
 * Any living beings, items, or otherwise things that
 * exist in the world are considered an Entity. Each
 * Entity also has an Inventory so things may be stored
 * inside of it.
 */
public abstract class Entity {
    protected World world;
    protected Inventory inventory;

    private Location location;
    private double weight;

    /**
     * Construct a new Entity.
     * @param world Current World object
     * @param location Initial Location of this Entity
     * @param weight The weight (in kg) of this Entity
     */
    public Entity(World world, Location location, double weight) {
        this.world = world;
        this.location = location;
        this.inventory = new Inventory();
        this.weight = weight;
    }

    /**
     * Construct a new Entity.
     *
     * Weight value is set to {@link Integer#MAX_VALUE}.
```

```java
42          * @param world Current World object
43          * @param location Initial Location of this Entity
44          */
45         public Entity(World world, Location location) {
46             this(world, location, Integer.MAX_VALUE);
47         }
48
49         /**
50          * Get this Entity's weight.
51          * @return Weight (in kg)
52          */
53         public double getWeight() {
54             return this.weight;
55         }
56
57         /**
58          * Get the name of this Entity.
59          * Shorthand for {@link #getAliases()}[0].
60          * @return First matched alias
61          */
62         public String getName() {
63             return this.getAliases()[0];
64         }
65
66         /**
67          * Get a highlighted representation of this Entity's name.
68          * @return Ansi hightlighted name
69          */
70         public String getHighlightedName() {
71             return Ansi.BackgroundWhite + Ansi.Black + this.getName() + Ansi.Reset;
72         }
73
74         /**
75          * Get the Inventory that this Entity holds.
76          * @return Inventory
77          */
78         public Inventory getInventory() {
79             return this.inventory;
80         }
81
82         /**
```

```java
 83          * Get the World this Entity resides in.
 84          * @return World
 85          */
 86         public World getWorld() {
 87             return this.world;
 88         }
 89
 90         /**
 91          * Get the Room that this Entity is currently in.
 92          * @return Room
 93          */
 94         public Room getRoom() {
 95             return this.location.getRoom();
 96         }
 97
 98         /**
 99          * Get the Inventory that this Entity is currently in.
100          * @return Inventory
101          */
102         public Inventory getInventoryWithin() {
103             return this.location.getInventory();
104         }
105
106         /**
107          * Remove this Entity from any existing place.
108          * Provides a consistent way to clean up the Entity before placing it anywhere.
109          * @param suppressEvents Whether to suppress the "Entity Left Room" Event
110          * @return Whether this Entity was removed from an Inventory
111          */
112         public boolean consume(boolean suppressEvents) {
113             boolean consumed = false;
114             Inventory inventory = this.location.getInventory();
115             if (inventory != null) consumed = inventory.remove(this);
116
117             Room previousRoom = this.getRoom();
118             if (previousRoom != null && !suppressEvents) this.world.emit(new EventEntityLeftRoom(this, previousRoom));
119
120             this.location.clear();
121             return consumed;
122         }
123
```

```java
124        /**
125         * Move the Entity into a Room.
126         * @param room Destination Room
127         */
128        public void setLocation(Room room) {
129            boolean consumed = this.consume(false);
130            this.location.setLocation(room);
131            if (!consumed) this.world.emit(new EventEntityEnteredRoom(this));
132        }
133
134        /**
135         * Move the Entity into an Inventory.
136         * @param inventory Destination Inventory
137         * @return Whether we successfully moved the entity into the inventory.
138         */
139        public boolean setLocation(Inventory inventory) {
140            if (inventory.add(this)) {
141                this.consume(true);
142                this.location.setLocation(inventory);
143                return true;
144            }
145
146            return false;
147        }
148
149        /**
150         * Link this Entity's inventory with an existing inventory.
151         * @param inventory Target inventory
152         */
153        public void entangleInventory(Inventory inventory) {
154            this.inventory = inventory;
155        }
156
157        /**
158         * Get names that this Entity can be called by.
159         * @return String array of names for this Entity
160         */
161        public abstract String[] getAliases();
162
163        /**
164         * Get a description of this Entity.
```

```
165         * @return String describing the Entity
166         */
167     public abstract String describe();
168 }
```

```java
package uk.insrt.coursework.zuul.entities;

import uk.insrt.coursework.zuul.world.Location;
import uk.insrt.coursework.zuul.world.World;

/**
 * Generic object class which avoids some boilerplate.
 * Use this for entities which are guaranteed to never change.
 */
public class EntityObject extends Entity {
    private String description;
    private String[] aliases;

    /**
     * Construct a new EntityObject
     * @param world Current World object
     * @param location Initial Location of this Entity
     * @param weight The weight (in kg) of this Entity
     * @param aliases Aliases which this object can be referred to by
     * @param description A description of this object
     */
    public EntityObject(World world, Location location, double weight, String[] aliases, String description) {
        super(world, location, weight);
        this.description = description;
        this.aliases = aliases;
    }

    /**
     * Construct a new EntityObject
     * @param world Current World object
     * @param location Initial Location of this Entity
     * @param weight The weight (in kg) of this Entity
     * @param name Name of this object
     * @param description A description of this object
     */
    public EntityObject(World world, Location location, double weight, String alias, String description) {
        this(world, location, weight, new String[] { alias }, description);
    }

    @Override
    public String describe() {
```

```java
42          return this.description;
43      }
44
45      @Override
46      public String[] getAliases() {
47          return this.aliases;
48      }
49  }
```

```java
package uk.insrt.coursework.zuul.entities;

import java.util.ArrayList;

import uk.insrt.coursework.zuul.io.IOSystem;
import uk.insrt.coursework.zuul.world.Direction;
import uk.insrt.coursework.zuul.world.Location;
import uk.insrt.coursework.zuul.world.Room;
import uk.insrt.coursework.zuul.world.World;

/**
 * Player entity which we can control and move around.
 */
public class EntityPlayer extends Entity {
    private ArrayList<Room> previousRooms;
    private ArrayList<Direction> retreatingDirection;

    /**
     * Construct a new Player Entity.
     * @param world World to place Player in
     */
    public EntityPlayer(World world) {
        super(world, new Location(), 70);
        this.previousRooms = new ArrayList<>();
        this.retreatingDirection = new ArrayList<>();
        this.inventory.setMaxWeight(35);
    }

    @Override
    public String[] getAliases() {
        return new String[] {
            "player", "me", "myself", "self", "yourself"
        };
    }

    @Override
    public String describe() {
        // We may skip defining how the Player looks,
        // this is because EntityPlayer is ignored
        // when looking around the room.
        return "";
```

```java
42         }
43
44         /**
45          * Move in a direction as instructed by command.
46          * @param direction Target Direction
47          */
48         public void go(Direction direction) {
49             var io = this.getWorld().getIO();
50
51             Room room = this.getRoom();
52             if (room == null) {
53                 io.println("You appear to be trapped.");
54                 return;
55             }
56
57             if (!room.canLeave(direction)) return;
58
59             Room destination = room.getAdjacent(direction);
60             if (destination == null) {
61                 io.println("You cannot go this way.");
62                 return;
63             }
64
65             this.retreatingDirection.add(direction.flip());
66             this.previousRooms.add(this.getRoom());
67             this.setLocation(destination);
68         }
69
70         /**
71          * Move to the previous room the player was in.
72          */
73         public void back() {
74             IOSystem io = this.getWorld().getIO();
75             int index = this.retreatingDirection.size() - 1;
76
77             if (index < 0) {
78                 io.println("Nowhere to go back to!");
79                 return;
80             }
81
82             Direction lastDirection = this.retreatingDirection.get(index);
```

```java
 83            if (this.getRoom().hasExit(lastDirection)) {
 84                this.retreatingDirection.remove(index);
 85                this.setLocation(this.previousRooms.remove(index));
 86            } else {
 87                io.println("Cannot leave the room this way.");
 88            }
 89        }
 90
 91        /**
 92         * Clear walk history.
 93         */
 94        public void clearHistory() {
 95            this.retreatingDirection.clear();
 96            this.previousRooms.clear();
 97        }
 98    }
```

```java
package uk.insrt.coursework.zuul.entities;

import java.util.ArrayList;

/**
 * Representation of an Entity's inventory
 * and what they are holding.
 */
public class Inventory {
    private ArrayList<Entity> items = new ArrayList<>();
    private double maxWeight;

    /**
     * Construct a new Inventory.
     */
    public Inventory() {
        super();
        this.maxWeight = 0;
    }

    /**
     * Set the max weight that can be carried in this inventory.
     * @param maxWeight Max weight (in kg)
     */
    public void setMaxWeight(double maxWeight) {
        this.maxWeight = maxWeight;
    }

    /**
     * Get the maximum weight that can be carried in this inventory.
     * @return Maximum weight that can be carried
     */
    public double getMaxWeight() {
        return this.maxWeight;
    }

    /**
     * Get the current weight of this inventory.
     * @return Weight (in kg)
     */
    public double getWeight() {
```

```java
42            return this
43                .items
44                .stream()
45                .mapToDouble(Entity::getWeight)
46                .sum();
47        }
48
49        /**
50         * Check if the inventory is full.
51         * @return True if the weight is greater than the max weight
52         */
53        public boolean isFull() {
54            return this.getWeight() >= this.getMaxWeight();
55        }
56
57        /**
58         * Add an entity to this inventory.
59         *
60         * There must be sufficient space for the entity.
61         * @param entity Target Entity
62         * @return Whether we successfully added the new entity.
63         */
64        public boolean add(Entity entity) {
65            if (this.getWeight() + entity.getWeight() > this.maxWeight) {
66                return false;
67            }
68
69            this.items.add(entity);
70            return true;
71        }
72
73        /**
74         * Remove an entity from this inventory.
75         * @param entity Target Entity
76         * @return Whether there was any change to the inventory.
77         */
78        public boolean remove(Entity entity) {
79            return this.items.remove(entity);
80        }
81
82        /**
```

```
83         * Get an Iterable over the Entities within this inventory.
84         * @return Iterable over Entities
85         */
86        public Iterable<Entity> getItems() {
87            return this.items;
88        }
89    }
```

```java
package uk.insrt.coursework.zuul.events;

/**
 * Represents a single event fired from
 * any source to be consumed by anything.
 */
public class Event {
    private boolean propagating = true;

    /**
     * Whether this event can continue running.
     * @return Whether propogation of this event was stopped
     */
    public boolean canRun() {
        return this.propagating;
    }

    /**
     * Stop further propagation of this event.
     */
    public void stopPropagation() {
        this.propagating = false;
    }
}
```

```java
package uk.insrt.coursework.zuul.events;

import java.util.HashMap;
import java.util.HashSet;
import java.util.LinkedHashSet;

/**
 * Event system which manages taking in events
 * from different sources and handles them
 * by firing callbacks on event listeners.
 */
public class EventSystem {
    private HashMap<Class<? extends Event>, LinkedHashSet<IEventListener<? extends Event>>> listeners = new HashMap<>();

    /**
     * Get existing Event listener list or create a new one if not exists.
     * @param event Event
     * @return Set of event listeners
     */
    private HashSet<IEventListener<? extends Event>> getList(Class<? extends Event> event) {
        var list = this.listeners.get(event);
        if (list == null) {
            list = new LinkedHashSet<>();
            this.listeners.put(event, list);
        }

        return list;
    }

    /**
     * Add a new event listener to this system.
     * @param <E> Generic Event type
     * @param event Event to remove from
     * @param listener Event listener callback
     */
    public<E extends Event> void addListener(Class<E> event, IEventListener<E> listener) {
        this.getList(event).add(listener);
    }

    /**
     * Remove an new event listener from this system.
```

```java
42          * @param <E> Generic Event type
43          * @param event Event to remove from
44          * @param listener Event listener callback
45          */
46         public<E extends Event> void removeListener(Class<E> event, IEventListener<E> listener) {
47             this.getList(event).remove(listener);
48         }
49
50         /**
51          * Emit an Event.
52          * @param <E> Generic Event type
53          * @param event Event to emit
54          */
55         @SuppressWarnings("unchecked")
56         public <E extends Event> void emit(E event) {
57             var listeners = this.listeners.get(event.getClass());
58             if (listeners == null) return;
59
60             for (@SuppressWarnings("rawtypes") IEventListener listener : listeners) {
61                 listener.onEvent(event);
62                 // Previously, there was a try catch ClassCastException
63                 // but I've since constricted the types on `addListener`
64                 // and `removeListener` so this should never happen.
65
66                 if (!event.canRun())
67                     break;
68             }
69         }
70     }
```

```java
package uk.insrt.coursework.zuul.events;

/**
 * Interface implementing an listener for an arbitrary {@link Event}.
 */
public interface IEventListener<E extends Event> {
    /**
     * Method called when this specific Event is emitted.
     * @param event Event to handle
     */
    public void onEvent(E event);
}
```

```java
package uk.insrt.coursework.zuul.events.world.behaviours;

import java.util.Random;

import uk.insrt.coursework.zuul.entities.Entity;
import uk.insrt.coursework.zuul.events.IEventListener;
import uk.insrt.coursework.zuul.events.world.EventTick;
import uk.insrt.coursework.zuul.world.Room;

/**
 * This is a simple behaviour which just randomly decides to move an Entity
 * through a set path whenever the game ticks forwards.
 */
public class SimpleWanderAI implements IEventListener<EventTick> {
    private Entity entity;
    private Room[] path;
    private int chance;

    private int index;
    private Random random;

    /**
     * Construct a new wandering behaviour for an Entity with a given path.
     * @param entity Entity which should be moved
     * @param path Path that this Entity should follow
     * @param chance The chance x that this entity moves, where x gives a 1/x fractional chance of moving.
     */
    public SimpleWanderAI(Entity entity, Room[] path, int chance) {
        this.entity = entity;
        this.path = path;
        this.chance = chance;

        this.index = 0;
        this.random = new Random();
    }

    @Override
    public void onEvent(EventTick event) {
        if (this.entity.getRoom() != this.path[this.index]) return;
        if (random.nextInt(this.chance) > 0) return;
```

```
42            this.index = (this.index + 1) % this.path.length;
43            this.entity.setLocation(this.path[this.index]);
44        }
45    }
```

```java
package uk.insrt.coursework.zuul.events.world;

import uk.insrt.coursework.zuul.entities.Entity;
import uk.insrt.coursework.zuul.events.Event;

/**
 * Event fired when an Entity enters a room.
 */
public class EventEntityEnteredRoom extends Event {
    private Entity entity;

    /**
     * Construct a new EntityEnteredRoom Event.
     * @param entity Target Entity
     */
    public EventEntityEnteredRoom(Entity entity) {
        this.entity = entity;
    }

    /**
     * Get the Entity relating to this event.
     * @return Entity
     */
    public Entity getEntity() {
        return this.entity;
    }
}
```

```java
package uk.insrt.coursework.zuul.events.world;

import uk.insrt.coursework.zuul.entities.Entity;
import uk.insrt.coursework.zuul.events.Event;
import uk.insrt.coursework.zuul.world.Room;

/**
 * Event fired when an Entity enters a room.
 */
public class EventEntityLeftRoom extends Event {
    private Entity entity;
    private Room room;

    /**
     * Construct a new EntityLeftRoom Event.
     * @param entity Target Entity
     * @param room Room the entity left
     */
    public EventEntityLeftRoom(Entity entity, Room room) {
        this.entity = entity;
        this.room = room;
    }

    /**
     * Get the Entity relating to this event.
     * @return Entity
     */
    public Entity getEntity() {
        return this.entity;
    }

    /**
     * Get the Room relating to this event.
     * @return Room
     */
    public Room getRoom() {
        return this.room;
    }
}
```

```java
package uk.insrt.coursework.zuul.events.world;

import uk.insrt.coursework.zuul.events.Event;

/**
 * Event fired when an arbitrary command is about to be run.
 */
public class EventProcessCommand extends Event {
    private String cmd;

    /**
     * Construct a new EventProcessCommand Event.
     * @param cmd Target command
     */
    public EventProcessCommand(String cmd) {
        this.cmd = cmd;
    }

    /**
     * Set command for this event.
     * @param cmd Overwrite current command
     */
    public void setCommand(String cmd) {
        this.cmd = cmd;
    }

    /**
     * Get the command relating to this event.
     * @return Arbitrary command
     */
    public String getCommand() {
        return this.cmd;
    }
}
```

```java
package uk.insrt.coursework.zuul.events.world;

import uk.insrt.coursework.zuul.events.Event;

/**
 * Event fired when the game ticks forward.
 * Such as when the player performs an action or goes to sleep.
 */
public class EventTick extends Event {}
```

```java
package uk.insrt.coursework.zuul;

import java.io.IOException;

import javax.swing.JOptionPane;

import uk.insrt.coursework.zuul.commands.CommandManager;
import uk.insrt.coursework.zuul.content.campaign.CampaignWorld;
import uk.insrt.coursework.zuul.content.campaign.commands.CommandMap;
import uk.insrt.coursework.zuul.content.campaign.commands.CommandWin;
import uk.insrt.coursework.zuul.events.world.EventProcessCommand;
import uk.insrt.coursework.zuul.events.world.EventTick;
import uk.insrt.coursework.zuul.io.IOSystem;
import uk.insrt.coursework.zuul.io.LocalisedIO;
import uk.insrt.coursework.zuul.io.SanitiseIO;
import uk.insrt.coursework.zuul.io.StandardIO;
import uk.insrt.coursework.zuul.ui.EventDraw;
import uk.insrt.coursework.zuul.ui.TerminalEmulator;
import uk.insrt.coursework.zuul.util.BlueJ;
import uk.insrt.coursework.zuul.util.Localisation;
import uk.insrt.coursework.zuul.world.World;

/**
 * Class for managing the game loop and initialising the world.
 */
public class Game {
    public static final String GAME_NAME = "World of Deez";

    private World world;
    private IOSystem io;
    private CommandManager commands;

    /**
     * Entrypoint to our application.
     * @param args Arguments provided to the application
     */
    public static void main(String[] args) {
        new Game().play();
    }

    /**
```

```java
42            * Initialise and start the game.
43            */
44          public void play() {
45              this.init();
46              this.start();
47          }
48
49          /**
50           * Initialise all required resources for the game to run.
51           */
52          private void init() {
53              // Determine how the game should run.
54              boolean inBlueJ = BlueJ.isRunningInBlueJ();
55              int selection = JOptionPane.showConfirmDialog(null, "Play full experience?\nUses custom terminal emulator.
   \n(recommended option)", GAME_NAME, JOptionPane.YES_NO_OPTION);
56              if (selection == 0) {
57                  if (inBlueJ) {
58                      // Fullscreen minimises itself immediately
59                      // when running from BlueJ, not sure what's
60                      // going on exactly, just disabling it in general.
61                      selection = 1;
62                  } else {
63                      selection = JOptionPane.showConfirmDialog(null, "Immersive mode?\nRuns emulator in fullscreen.
   \n(recommended option)", GAME_NAME, JOptionPane.YES_NO_OPTION);
64                  }
65
66                  this.io = new TerminalEmulator(selection == 0);
67              } else {
68                  this.io = new StandardIO();
69
70                  if (inBlueJ) {
71                      selection = JOptionPane.showConfirmDialog(null, "Is this running from inside BlueJ?", GAME_NAME, JOption
   Pane.YES_NO_OPTION);
72                      if (selection == 0) {
73                          this.io = new SanitiseIO(this.io);
74                      }
75                  }
76              }
77
78              // Setup the command manager.
79              this.commands = new CommandManager();
```

```java
 80            this.commands.registerCommand(new CommandWin());
 81
 82            // Register the Map command if we're in term emu mode.
 83            // We draw images here so it's not available generally.
 84            if (this.io instanceof TerminalEmulator) {
 85                CommandMap map = new CommandMap();
 86                this.commands.registerCommand(map);
 87                ((TerminalEmulator) this.io).getEventSystem().addListener(EventDraw.class, map);
 88            }
 89
 90            // Load all the data we need and initialise world.
 91            Localisation locale = new Localisation();
 92            this.io = new LocalisedIO(this.io, locale);
 93
 94            try {
 95                locale.loadLocale("en_GB");
 96            } catch (IOException e) {
 97                System.err.println("Failed to load translations!");
 98                e.printStackTrace();
 99            }
100
101            this.world = new CampaignWorld(this.io);
102        }
103
104        /**
105         * Start the game loop.
106         */
107        private void start() {
108            this.world.spawnPlayer();
109
110            while (true) {
111                this.io.print("> ");
112                String input = this.io.readLine().toLowerCase();
113
114                EventProcessCommand event = new EventProcessCommand(input);
115                this.world.emit(event);
116
117                if (this.commands.runCommand(this.world, event.getCommand())) {
118                    break;
119                }
120
```

```java
121                this.world.emit(new EventTick());
122            }
123
124        this.io.println("Goodbye.");
125
126        try {
127            Thread.sleep(1000);
128            this.io.dispose();
129        } catch (Exception e) {}
130    }
131 }
```

```java
package uk.insrt.coursework.zuul.io;

import java.awt.Color;
import java.util.regex.Pattern;

/**
 * ANSI escape codes
 * Used https://stackoverflow.com/a/5762502 as a reference.
 */
public class Ansi {
    public static final String Reset = "\u001B[0m";
    public static final String Black = "\u001B[30m";
    public static final String Red = "\u001B[31m";
    public static final String Green = "\u001B[32m";
    public static final String Yellow = "\u001B[33m";
    public static final String Blue = "\u001B[34m";
    public static final String Purple = "\u001B[35m";
    public static final String Cyan = "\u001B[36m";
    public static final String White = "\u001B[37m";

    public static final String BackgroundBlack = "\u001B[40m";
    public static final String BackgroundRed = "\u001B[41m";
    public static final String BackgroundGreen = "\u001B[42m";
    public static final String BackgroundYellow = "\u001B[43m";
    public static final String BackgroundBlue = "\u001B[44m";
    public static final String BackgroundPurple = "\u001B[45m";
    public static final String BackgroundCyan = "\u001B[46m";
    public static final String BackgroundWhite = "\u001B[47m";

    /**
     * Regex Pattern used to match Ansi codes forwards.
     */
    public static final Pattern AnsiPattern = Pattern.compile("^\\u001B\\[(\\d{1,3})m");

    private static final Color ColorBlack = new Color(0, 0, 0);
    private static final Color ColorRed = new Color(224, 108, 117);
    private static final Color ColorGreen = new Color(152, 195, 121);
    private static final Color ColorYellow = new Color(229, 192, 123);
    private static final Color ColorBlue = new Color(97, 175, 239);
    private static final Color ColorMagenta = new Color(198, 120, 221);
    private static final Color ColorCyan = new Color(86, 182, 194);
```

```java
42        private static final Color ColorWhite = new Color(255, 255, 255);
43
44        /**
45         * Convert a given escape code value, {@code (\d+?)} in {@link #AnsiPattern}, to a Color.
46         * @param value Escape code value
47         * @return Resolved Java awt Color
48         */
49        public static Color fromEscapeCode(int value) {
50            switch (value % 10) {
51                case 0: return ColorBlack;
52                case 1: return ColorRed;
53                case 2: return ColorGreen;
54                case 3: return ColorYellow;
55                case 4: return ColorBlue;
56                case 5: return ColorMagenta;
57                case 6: return ColorCyan;
58                case 7:
59                default: return ColorWhite;
60            }
61        }
62    }
```

```java
package uk.insrt.coursework.zuul.io;

/**
 * Interface representing an arbitrary IO system.
 * This can be implemented to input or output from various interfaces.
 */
public interface IOSystem {
    /**
     * Print a string out through an arbitrary output channel.
     * @param out String to print
     */
    public void print(String out);

    /**
     * Print a string out through an arbitrary output channel and append {@code \n}.
     * @param out String to print
     */
    public void println(String out);

    /**
     * Read a String up until the first encountered {@code \n} from an arbitrary input channel.
     * @return String of line read in
     */
    public String readLine();

    /**
     * Dispose of the arbitrary input and output channels.
     */
    public void dispose();
}
```

```java
package uk.insrt.coursework.zuul.io;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

import uk.insrt.coursework.zuul.util.Localisation;

/**
 * Translate and localise any incoming output.
 */
public class LocalisedIO implements IOSystem {
    private final Pattern pattern = Pattern.compile("<([\\w\\.]+?)>");

    private IOSystem io;
    private Localisation locale;

    /**
     * Construct a new LocalisedIO.
     * @param io Provided IO system we should feed into
     * @param locale Locale to apply to any i18n strings
     */
    public LocalisedIO(IOSystem io, Localisation locale) {
        this.io = io;
        this.locale = locale;
    }

    /**
     * Replace i18n strings in any given String with their actual localised values.
     * Using replacement code from https://stackoverflow.com/a/27359491.
     * @param input String to process
     * @return Final processed string
     */
    private String replace(String input) {
        StringBuffer result = new StringBuffer();
        Matcher matcher = this.pattern.matcher(input);

        while (matcher.find()) {
            matcher.appendReplacement(result, this.locale.get(matcher.group(1)));
        }

        matcher.appendTail(result);
```

```java
42              return result.toString();
43          }
44
45          @Override
46          public void print(String out) {
47              this.io.print(this.replace(out));
48          }
49
50          @Override
51          public void println(String out) {
52              this.io.println(this.replace(out));
53          }
54
55          @Override
56          public String readLine() {
57              return this.io.readLine();
58          }
59
60          @Override
61          public void dispose() {
62              this.io.dispose();
63          }
64      }
```

```java
package uk.insrt.coursework.zuul.io;

/**
 * Sanitise incoming output and remove any Ansi escape sequences.
 * This is required to print out into the BlueJ console without additional characters.
 */
public class SanitiseIO implements IOSystem {
    private final String ansiPattern = "\\u001B\\[(\\d{1,3})m";
    private IOSystem io;

    /**
     * Construct a new SanitiseIO.
     * @param io Provided IO system we should feed into
     */
    public SanitiseIO(IOSystem io) {
        this.io = io;
    }

    @Override
    public void print(String out) {
        this.io.print(out.replaceAll(this.ansiPattern, " "));
    }

    @Override
    public void println(String out) {
        this.io.println(out.replaceAll(this.ansiPattern, " "));
    }

    @Override
    public String readLine() {
        return this.io.readLine();
    }

    @Override
    public void dispose() {
        this.io.dispose();
    }
}
```

```java
package uk.insrt.coursework.zuul.io;

import java.util.Scanner;

/**
 * A simple IO system implementation which feeds
 * into System.out and takes data from System.in
 */
public class StandardIO implements IOSystem {
    private Scanner reader;

    /**
     * Construct a new StandardIO.
     */
    public StandardIO() {
        this.reader = new Scanner(System.in);
    }

    @Override
    public void print(String out) {
        System.out.print(out);
    }

    @Override
    public void println(String out) {
        System.out.println(out);
    }

    @Override
    public String readLine() {
        return this.reader.nextLine();
    }

    @Override
    public void dispose() {}
}
```

```java
package uk.insrt.coursework.zuul.ui;

import java.awt.Image;

/**
 * Representation of a single Emoji which can be rendered in the terminal emulator.
 */
public class Emoji {
    private Image image;
    private int width;
    private int height;

    /**
     * Construct a new Emoji given the image and unicode representation.
     * @param image Image to render when this Emoji is used
     * @param unicode Unicode representation of this Emoji, used to determine width
     */
    public Emoji(Image image, String unicode) {
        this.image = image;
        this.width = (int) unicode.chars().count();
        this.height = 1;
    }

    /**
     * Construct a new Emoji given the image and size constraints.
     * @param image Image to render when this Emoji is used
     * @param width Width of this Emoji
     * @param height Height of this Emoji
     */
    public Emoji(Image image, int width, int height) {
        this.image = image;
        this.width = width;
        this.height = height;
    }

    /**
     * Get the Image for this Emoji
     * @return Image
     */
    public Image getImage() {
        return this.image;
```

```
42          }
43
44          /**
45           * Get the calculated width of this Emoji
46           * @return Width
47           */
48          public int getWidth() {
49              return this.width;
50          }
51
52          /**
53           * Get the calculated height of this Emoji
54           * @return Height
55           */
56          public int getHeight() {
57              return this.height;
58          }
59      }
```

```java
package uk.insrt.coursework.zuul.ui;

import java.awt.Image;
import java.io.IOException;
import java.io.InputStream;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;

import javax.imageio.ImageIO;

import com.moandjiezana.toml.Toml;

import org.apache.commons.io.IOUtils;

import uk.insrt.coursework.zuul.util.Tree;

/**
 * Class which helps manage loading and resolving Emojis.
 */
public class EmojiManager {
    private HashMap<String, Emoji> emojis;
    private Tree<Character, String> emojiTree;
    private Tree<Character, String> currentNode;

    /**
     * Construct a new EmojiManager.
     */
    public EmojiManager() {
        this.emojis = new HashMap<>();
        this.emojiTree = new Tree<>();
        this.currentNode = this.emojiTree;
    }

    /**
     * Check whether a given Emoji is present in this manager.
     * @param emoji Unicode representation of Emoji
     * @return True if the Emoji is available
     */
```

```java
    public boolean hasEmoji(String emoji) {
        return this.emojis.containsKey(emoji);
    }

    /**
     * Get an Emoji by its Unicode representation.
     * @param emoji Unicode representation of Emoji
     * @return The Emoji or null if it doesn't exist
     */
    public Emoji getEmoji(String emoji) {
        return this.emojis.get(emoji);
    }

    /**
     * Get currently matched emoji and resets position.
     * @return Emoji if it was found, or null if not
     */
    public Emoji getEmoji() {
        String value = this.currentNode.getValue();
        Emoji emoji = this.emojis.get(value);
        this.resetState();
        return emoji;
    }

    /**
     * Reset the state of the matching mechanism.
     */
    public void resetState() {
        this.currentNode = this.emojiTree;
    }

    /**
     * The matching mechanism has not matched any characters to potential emojis.
     */
    public static final int MATCH_NONE = 0;

    /**
     * The matching mechanism has matched some characters to potential emojis.
     */
    public static final int MATCH_SOME = 1;
```

```java
 83        /**
 84         * The matching mechanism has matched an emoji.
 85         */
 86        public static final int MATCH_FOUND = 2;
 87
 88        /**
 89         * Match the next character.
 90         * @param c Character to match against
 91         * @return One of {@link #MATCH_NONE}, {@link #MATCH_SOME} or {@link #MATCH_FOUND}
 92         */
 93        public int match(char c) {
 94            var child = this.currentNode.getChild(c);
 95            if (child != null) {
 96                this.currentNode = child;
 97                if (child.getValue() == null) return MATCH_SOME;
 98                return MATCH_FOUND;
 99            }
100
101            if (this.currentNode != this.emojiTree) {
102                this.currentNode = this.emojiTree;
103                child = this.emojiTree.getChild(c);
104                if (child != null) {
105                    if (child.getValue() != null) return MATCH_SOME;
106                    return MATCH_FOUND;
107                }
108            }
109
110            return MATCH_NONE;
111        }
112
113        /**
114         * Load emoji definitions and resources from a given resource directory.
115         * @param rootDir Root directory at which we expect a valid {@code definitions.toml} to exist
116         * @throws IOException if the definition file is missing or defined emojis are invalid
117         */
118        public void loadResources(String rootDir) throws IOException {
119            InputStream defnStream = this.getClass().getResourceAsStream(rootDir + "/definitions.toml");
120            // We need to force UTF-8 encoding or else unicode emojis may get mangled.
121            String defnString = IOUtils.toString(defnStream, StandardCharsets.UTF_8);
122            Toml defn = new Toml().read(defnString);
123            List<HashMap<String, Object>> emojis = defn.getList("emojis");
```

```java
124
125          // Load each emoji in sequence
126          for (var emoji : emojis) {
127              String path = (String) emoji.get("path");
128              String unicode = (String) emoji.get("unicode");
129
130              InputStream stream = this.getClass().getResourceAsStream(rootDir + "/" + path);
131              Image image = ImageIO.read(stream);
132
133              Emoji newEmoji;
134              if (emoji.containsKey("width") && emoji.containsKey("height")) {
135                  int width = ((Long) emoji.get("width")).intValue();
136                  int height = ((Long) emoji.get("height")).intValue();
137                  newEmoji = new Emoji(image, width, height);
138              } else {
139                  newEmoji = new Emoji(image, unicode);
140              }
141              this.emojis.put(unicode, newEmoji);
142              this.emojiTree.addChildWithPath(
143                  new ArrayList<>(Arrays.asList(
144                      unicode.chars()
145                          .mapToObj(c -> (char)c)
146                          .toArray(Character[]::new))),
147                  unicode
148              );
149          }
150      }
151  }
```

```java
package uk.insrt.coursework.zuul.ui;

import java.awt.Graphics;

import uk.insrt.coursework.zuul.events.Event;

/**
 * Event fired when the terminal emulator draws a new frame.
 */
public class EventDraw extends Event {
    private Graphics g;
    private float ox;
    private float oy;
    private float fw;
    private float fh;

    /**
     * Construct a new EventDraw Event.
     * @param g Graphics context
     * @param ox Origin X position
     * @param oy Origin Y position
     * @param fw Font character width
     * @param fh Font character height
     */
    public EventDraw(Graphics g, float ox, float oy, float fw, float fh) {
        this.g = g;
        this.ox = ox;
        this.oy = oy;
        this.fw = fw;
        this.fh = fh;
    }

    /**
     * Get the Graphics relating to this event.
     * @return Graphics
     */
    public Graphics getGraphics() {
        return this.g;
    }

    /**
```

```java
42          * Get the origin X position of the contents of the terminal.
43          * @return X position
44          */
45         public float getOriginX() {
46             return this.ox;
47         }
48
49         /**
50          * Get the origin Y position of the contents of the terminal.
51          * @return Y position
52          */
53         public float getOriginY() {
54             return this.oy;
55         }
56
57         /**
58          * Get the character width.
59          * @return Character width
60          */
61         public float getCharWidth() {
62             return this.fw;
63         }
64
65         /**
66          * Get the character height.
67          * @return Character height
68          */
69         public float getCharHeight() {
70             return this.fh;
71         }
72     }
```

```java
package uk.insrt.coursework.zuul.ui;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GraphicsEnvironment;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;

import javax.swing.JFrame;

/**
 * Window frame for {@link TerminalEmulator}
 */
public class JTerminalFrame extends JFrame {
    private JTerminalView view;
    private boolean fullscreen;

    /**
     * Construct a new JTerminalFrame
     * @param emulator Terminal emulator this frame belongs to
     */
    public JTerminalFrame(TerminalEmulator emulator) {
        this.view = new JTerminalView(emulator);
        this.fullscreen = emulator.isFullscreen();
        this.makeFrame(emulator);
    }

    /**
     * Make and display all of the elements within this frame.
     * @param emulator Terminal emulator this frame belongs to
     */
    public void makeFrame(TerminalEmulator emulator) {
        this.setLayout(new BorderLayout());
        this.add(this.view);
        this.pack();
        this.setLocationRelativeTo(null);
        this.setMinimumSize(new Dimension(640, 640));
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Add a listener for any user input.
        // https://stackoverflow.com/a/21970006
```

```java
42            this.addKeyListener(new KeyListener() {
43                public void keyPressed(KeyEvent event) {
44                    int code = event.getKeyCode();
45                    switch (code) {
46                        case 8: {
47                            emulator.pop();
48                            return;
49                        }
50                        case 10: {
51                            emulator.flush();
52                            break;
53                        }
54                        default: {
55                            if ((code >= 65 && code <= 90) || (code >= 48 && code <= 57) || code == 32) {
56                                emulator.push(event.getKeyChar());
57                            }
58                        }
59                    }
60                }
61
62                public void keyTyped(KeyEvent e) {}
63                public void keyReleased(KeyEvent e) {}
64            });
65
66            // We are ready to display, show everything.
67            // Prerequisite to making the frame fullscreen too.
68            this.setVisible(true);
69
70            // If we are allowed to launch in fullscreen, switch to that mode now.
71            if (this.fullscreen) {
72                GraphicsEnvironment
73                    .getLocalGraphicsEnvironment()
74                    .getDefaultScreenDevice()
75                    .setFullScreenWindow(this);
76            }
77        }
78
79        @Override
80        public void dispose() {
81            super.dispose();
82            this.view.dispose();
```

```
83        }
84    }
```

```java
package uk.insrt.coursework.zuul.ui;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.FontFormatException;
import java.awt.Graphics;
import java.awt.event.ComponentAdapter;
import java.awt.event.ComponentEvent;
import java.awt.font.FontRenderContext;
import java.awt.geom.AffineTransform;
import java.io.IOException;
import java.io.InputStream;

import javax.swing.JPanel;

import com.moandjiezana.toml.Toml;

/**
 * Rendering component of {@link TerminalEmulator}
 */
public class JTerminalView extends JPanel {
    private TerminalEmulator emulator;

    private EmojiManager emojiManager;
    private Font derivedFont;
    private Font font;

    private Thread blinkThread;
    private boolean blinkState;

    private float fw, fh, foffset, fratio;

    /**
     * Construct a TerminalView
     * @param emulator Terminal emulator this view belongs to
     */
    public JTerminalView(TerminalEmulator emulator) {
        this.emulator = emulator;
        this.emojiManager = new EmojiManager();
        this.blinkState = false;
```

```java
42              this.loadResources();
43              this.makeFrame();
44          }
45
46          /**
47           * Prepare the terminal view for rendering.
48           */
49          public void makeFrame() {
50              var view = this;
51              this.setBackground(Color.BLACK);
52
53              // Register a listener to repaint and adjust measurements when resizing window.
54              // https://stackoverflow.com/a/8917978
55              this.addComponentListener(new ComponentAdapter() {
56                  public void componentResized(ComponentEvent e) {
57                      view.setBackground(Color.BLACK);
58                      view.deriveFont();
59                      view.repaint();
60                  }
61              });
62
63              // Start a new thread for blinking the cursor.
64              this.blinkThread = new Thread("Blink Thread") {
65                  public void run() {
66                      try {
67                          while (true) {
68                              view.blinkState = !view.blinkState;
69                              view.repaint();
70
71                              Thread.sleep(500);
72                          }
73                      } catch (InterruptedException e) {}
74                  }
75              };
76
77              this.blinkThread.start();
78          }
79
80          /**
81           * Load any resources required by the terminal view to render itself properly.
82           */
```

```java
 83         public void loadResources() {
 84             try {
 85                 InputStream stream = this.getClass().getResourceAsStream("/emulator.toml");
 86                 Toml defn = new Toml().read(stream);
 87
 88                 // If a font is defined, load it.
 89                 String font = defn.getString("font");
 90                 if (font != null) {
 91                     this.loadFont(font, 32.0f / 12.8f);
 92                 }
 93
 94                 // If an emoji root directory is defined, load it.
 95                 String rootDir = defn.getString("emojis");
 96                 if (rootDir != null) {
 97                     this.emojiManager.loadResources(rootDir);
 98                 }
 99             } catch (Exception e) {
100                 System.err.println("Failed to load any resources for terminal view!");
101                 e.printStackTrace();
102             }
103         }
104
105         /**
106          * Load a specific font with a known ratio.
107          * We expect this font to be monoscape.
108          * @param source Path to the font to be loaded
109          * @param ratio Ratio of width to height for this font
110          * @throws IOException if the font cannot be loaded from a given path
111          * @throws FontFormatException if the font is of an incorrect format, we expect a TTF
112          */
113         public void loadFont(String source, float ratio) throws IOException, FontFormatException {
114             InputStream stream = this.getClass().getResourceAsStream(source);
115             this.fratio = ratio;
116             this.font = Font.createFont(Font.TRUETYPE_FONT, stream);
117         }
118
119         /**
120          * Derive the font measurements before we continue rendering.
121          */
122         public void deriveFont() {
123             final int padding = 100;
```

```java
124
125          // To find the height of the font, we take the smallest
126          // side of the window height or the proportional height
127          // found from the window width, and then we divide it
128          // by our fixed terminal height.
129          float height = Math.min(
130              this.fratio *
131                  (float) (this.getWidth() - padding)
132                  / (float) TerminalEmulator.TERMINAL_WIDTH,
133              (this.getHeight() - padding)
134                  / (float) TerminalEmulator.TERMINAL_HEIGHT
135          );
136
137          this.derivedFont = this.font.deriveFont(height);
138
139          // The FontRenderContext is used to determine the font dimensions
140          var frc = new FontRenderContext(new AffineTransform(), true, true);
141          var bounds = this.derivedFont.getStringBounds(" ", frc);
142
143          this.fw = (float) bounds.getWidth();
144          this.fh = (float) bounds.getHeight();
145
146          // We need to find the distance between the baseline
147          // and the ascender so we can properly align everything.
148          // https://docs.oracle.com/javase/tutorial/2d/text/fontconcepts.html
149          this.foffset = this.derivedFont.getLineMetrics(" ", frc).getAscent();
150      }
151
152      /**
153       * Dispose of this terminal view.
154       */
155      public void dispose() {
156          // We need to kill the blink thread when disposing
157          // of the UI, but that's not possible so instead I
158          // am interrupting the thread, catching that and
159          // exiting out peacefully.
160          // https://docs.oracle.com/javase/1.5.0/docs/guide/misc/threadPrimitiveDeprecation.html
161          this.blinkThread.interrupt();
162      }
163
164      @Override
```

```java
public Dimension getPreferredSize() {
    return new Dimension(1280, 960);
}

@Override
protected void paintComponent(Graphics g) {
    // https://stackoverflow.com/a/17922749
    super.paintComponent(g);

    // Setup our canvas for rendering.
    g.setColor(Color.BLACK);
    g.setFont(this.derivedFont);
    g.fillRect(0, 0, this.getWidth(), this.getHeight());

    // Find our topleft-most (x,y) to start rendering from.
    TextBuffer buffer = this.emulator.getBuffer();
    float ox = (this.getWidth() - this.fw * buffer.getWidth()) / 2;
    float oy = (this.getHeight() - this.fh * buffer.getHeight()) / 2;

    // Render each cell individually.
    for (int y=0;y<buffer.getHeight();y++) {
        int skipChars = 0;
        for (int x=0;x<buffer.getWidth();x++) {
            // If needs be, skip chars in this line.
            if (skipChars > 0) {
                skipChars--;
                continue;
            }

            // Get the character to render.
            char c = buffer.getChar(x, y);

            // Match each char for emoji codepoints.
            // If we start to match an emoji, peek ahead.
            int emojiMatch = this.emojiManager.match(c);
            int offset = 0;
            while (emojiMatch == EmojiManager.MATCH_SOME) {
                emojiMatch = this.emojiManager.match(buffer.getChar(x + ++offset, y));
            }

            // Get this cell's background and foreground colours.
```

```java
206                    Color bg = buffer.getBg(x, y);
207                    Color fg = buffer.getFg(x, y);
208
209                    // Find this char's offset.
210                    int drawX = Math.round(ox + this.fw * x);
211                    int drawY = Math.round(oy + this.fh * y);
212
213                    // Draw rect if there's a background present.
214                    if (bg != null && bg != Color.BLACK) {
215                        g.setColor(bg);
216                        g.fillRect(drawX, drawY, (int) Math.ceil(this.fw), (int) Math.ceil(this.fh));
217                    }
218
219                    // If we're drawing an emoji, get the image and skip text.
220                    if (emojiMatch == EmojiManager.MATCH_FOUND) {
221                        Emoji emoji = this.emojiManager.getEmoji();
222                        g.drawImage(
223                            emoji.getImage(),
224                            drawX, drawY,
225                            Math.round(this.fw * emoji.getWidth()),
226                            Math.round(this.fh * emoji.getHeight()),
227                            this
228                        );
229
230                        skipChars = offset;
231                        continue;
232                    }
233
234                    // Drawing the char if it's not a space, we have to
235                    // take care to add the offset we previously found or
236                    // otherwise the distance between the baseline and
237                    // the ascender. This is because Graphics.drawString
238                    // draws text from the leftmost baseline equal to (x,y).
239                    if (c != 0 && c != ' ') {
240                        g.setColor(fg);
241                        g.drawString(
242                            String.valueOf(c),
243                            drawX,
244                            Math.round(drawY + this.foffset)
245                        );
246                    }
```

```java
247                }
248
249                this.emojiManager.resetState();
250            }
251
252            // Draw a blinker to indicate the user can type here.
253            // Offset it a bit to not interfere with any text on-screen.
254            if (this.blinkState) {
255                g.setColor(Color.WHITE);
256                g.fillRect(
257                    (int) (ox + this.fw * buffer.getPosX() + 1),
258                    (int) (oy + this.fh * buffer.getPosY() + this.foffset),
259                    (int) this.fw - 2,
260                    (int) (this.fh / 8)
261                );
262            }
263
264            // Fire draw event for custom rendering.
265            this.emulator.getEventSystem().emit(new EventDraw(g, ox, oy, this.fw, this.fh));
266        }
267    }
```

```java
package uk.insrt.coursework.zuul.ui;

import java.awt.Color;
import java.awt.EventQueue;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.LinkedBlockingQueue;

import uk.insrt.coursework.zuul.events.EventSystem;
import uk.insrt.coursework.zuul.io.IOSystem;

/**
 * A terminal emulator which implements an IO system to
 * be arbitrarily plugged into any existing components.
 */
public class TerminalEmulator implements IOSystem {
    public static final int TERMINAL_WIDTH = 80;
    public static final int TERMINAL_HEIGHT = 25;

    private BlockingQueue<String> queue;
    private EventSystem eventSystem;
    private JTerminalFrame frame;
    private boolean fullscreen;
    private TextBuffer buffer;
    private String input;

    /**
     * Construct and build a new TerminalEmulator
     * @param fullscreen Whether to launch the emulator in fullscreen
     */
    public TerminalEmulator(boolean fullscreen) {
        this.queue = new LinkedBlockingQueue<>();
        this.eventSystem = new EventSystem();
        this.buffer = new TextBuffer(TERMINAL_WIDTH, TERMINAL_HEIGHT);
        this.fullscreen = fullscreen;
        this.input = new String();

        this.buildFrame();
    }

    /**
     * Construct a new TerminalEmulator and force windowed mode.
```

```java
42         */
43        public TerminalEmulator() {
44            this(false);
45        }
46
47        /**
48         * Build and show the terminal emulator.
49         */
50        public void buildFrame() {
51            var emulator = this;
52            EventQueue.invokeLater(new Runnable() {
53                @Override
54                public void run() {
55                    emulator.frame = new JTerminalFrame(emulator);
56                }
57            });
58        }
59
60        /**
61         * Get the local event system for this emulator.
62         * @return The event system
63         */
64        public EventSystem getEventSystem() {
65            return this.eventSystem;
66        }
67
68        /**
69         * Get the emulator's text buffer.
70         * @return The text buffer
71         */
72        public TextBuffer getBuffer() {
73            return this.buffer;
74        }
75
76        /**
77         * Tell the terminal frame to repaint contents.
78         */
79        private void repaint() {
80            if (this.frame != null) {
81                this.frame.repaint();
82            }
```

```java
 83            }
 84
 85            /**
 86             * Check whether we are in fullscreen mode.
 87             * @return True if we are fullscreen
 88             */
 89            public boolean isFullscreen() {
 90                return this.fullscreen;
 91            }
 92
 93            /**
 94             * Push a new character to the input buffer.
 95             * @param c Character to push
 96             */
 97            public void push(char c) {
 98                if (this.buffer.getPosX() + 1 == TERMINAL_WIDTH) return;
 99
100                this.input += c;
101                this.buffer.write(new String(new char[] { c }));
102                this.buffer.setLastFg(Color.GRAY);
103                this.repaint();
104            }
105
106            /**
107             * Pop last character from the input buffer.
108             */
109            public void pop() {
110                if (this.input.length() > 0) {
111                    this.input = this.input.substring(0, this.input.length() - 1);
112                    this.buffer.backspace();
113                    this.repaint();
114                }
115            }
116
117            /**
118             * Flush input from terminal emulator thread and send it to whatever is
119             * waiting for it on another thread. Uses a blocking queue to send data
120             * between threads, as seen here: https://stackoverflow.com/a/23413506
121             */
122            public void flush() {
123                this.queue.add(this.input);
```

```java
124             this.input = new String();
125         }
126
127         @Override
128         public void print(String out) {
129             buffer.write(out);
130             this.repaint();
131         }
132
133         @Override
134         public void println(String out) {
135             buffer.write(out + "\n");
136             this.repaint();
137         }
138
139         @Override
140         public String readLine() {
141             try {
142                 String line = this.queue.take();
143                 this.print("\n");
144                 return line;
145             } catch (Exception err) {
146                 err.printStackTrace();
147                 System.exit(1);
148                 return " ";
149             }
150         }
151
152         @Override
153         public void dispose() {
154             this.frame.dispose();
155         }
156     }
```

```java
package uk.insrt.coursework.zuul.ui;

import java.awt.Color;
import java.util.regex.Matcher;

import uk.insrt.coursework.zuul.io.Ansi;

/**
 * Representation of a text and colour buffer.
 * Provides various utilities for manipulating text on screen.
 */
public class TextBuffer {
    private char[][] buffer;
    private Color[][] bufferBg;
    private Color[][] bufferFg;

    private int width;
    private int height;

    private int posX;
    private int posY;

    private Color bg;
    private Color fg;

    private boolean overflow;

    /**
     * Construct a new TextBuffer with given constraints.
     * @param width Buffer width
     * @param height Buffer height
     */
    public TextBuffer(int width, int height) {
        this.buffer = new char[height][width];
        this.bufferBg = new Color[height][width];
        this.bufferFg = new Color[height][width];

        this.width = width;
        this.height = height;

        this.posX = 0;
```

```java
42            this.posY = 0;
43
44            this.bg = Color.BLACK;
45            this.fg = Color.WHITE;
46
47            this.overflow = false;
48        }
49
50        /**
51         * Remove top-most row and move all the other rows up.
52         */
53        public void shift() {
54            for (int i=0;i<this.height-1;i++) {
55                this.buffer[i] = this.buffer[i + 1];
56                this.bufferBg[i] = this.bufferBg[i + 1];
57                this.bufferFg[i] = this.bufferFg[i + 1];
58            }
59
60            this.bufferBg[this.height - 1] = new Color[this.width];
61            this.bufferFg[this.height - 1] = new Color[this.width];
62            this.buffer[this.height - 1] = new char[this.width];
63        }
64
65        /**
66         * Remove the last previous character written to the buffer.
67         */
68        public void backspace() {
69            if (this.posX == 0) return;
70            this.buffer[this.posY][--this.posX] = ' ';
71        }
72
73        /**
74         * Retroactively set the foreground for the previous character written.
75         * @param color Foreground colour
76         */
77        public void setLastFg(Color color) {
78            this.bufferFg[this.posY][this.posX - 1] = color;
79        }
80
81        /**
82         * Retroactively set the background for the previous character written.
```

```java
 83         * @param color Background colour
 84         */
 85        public void setLastBg(Color color) {
 86            this.bufferBg[this.posY][this.posX - 1] = color;
 87        }
 88
 89        /**
 90         * Write a new character to the text buffer.
 91         * This will move the cursor forwards.
 92         * @param c Character to write
 93         */
 94        public void write(char c) {
 95            // If we encounter a newline, shift downwards or go on to a new line.
 96            if (c == '\n') {
 97                if (this.overflow) {
 98                    this.overflow = false;
 99                    return;
100                }
101
102                this.posX = 0;
103
104                if (this.posY == this.height - 1) {
105                    this.shift();
106                } else {
107                    this.posY++;
108                }
109
110                return;
111            }
112
113            // Clear any overflow value.
114            this.overflow = false;
115
116            // Commit new character to buffer.
117            this.bufferBg[this.posY][this.posX] = this.bg;
118            this.bufferFg[this.posY][this.posX] = this.fg;
119            this.buffer[this.posY][this.posX++] = c;
120
121            // If we're at the end of the line, shift downwards or move to new line.
122            if (this.posX == this.width) {
123                this.posX = 0;
```

```java
124
125                if (this.posY == this.height - 1) {
126                    this.shift();
127                } else {
128                    this.posY++;
129                }
130
131                // Set a flag to say we just naturally overflowed and to ignore
132                // the next newline character that may appear.
133                this.overflow = true;
134            }
135        }
136
137        /**
138         * Write a string value to the text buffer.
139         * @param value String value to write
140         */
141        public void write(String value) {
142            // Write each character sequentially.
143            for (int i=0;i<value.length();i++) {
144                char c = value.charAt(i);
145
146                // If we encounter an Ansi escape character, then take the
147                // substring from this point on and determine if it is a valid
148                // escape code. If it is, apply any changes before continuing.
149                if (c == '\u001B') {
150                    Matcher matcher = Ansi.AnsiPattern.matcher(value.substring(i));
151                    if (matcher.find()) {
152                        int v = Integer.parseInt(matcher.group(1));
153                        i += 3 + (v > 9 ? 1 : 0);
154
155                        if (v == 0) {
156                            this.bg = Color.BLACK;
157                            this.fg = Color.WHITE;
158                        } else if (v >= 30 && v < 38) {
159                            this.fg = Ansi.fromEscapeCode(v);
160                        } else if (v >= 40 && v < 48) {
161                            this.bg = Ansi.fromEscapeCode(v);
162                        }
163
164                        continue;
```

```java
                    }
                }

                this.write(c);
            }
        }

        /**
         * Get a character at a certain position
         * @param x X position
         * @param y Y position
         * @return Character at given position
         */
        public char getChar(int x, int y) {
            return this.buffer[y][x];
        }

        /**
         * Get the background colour at a certain position
         * @param x X position
         * @param y Y position
         * @return Background colour at given position
         */
        public Color getBg(int x, int y) {
            return this.bufferBg[y][x];
        }

        /**
         * Get the foreground colour at a certain position
         * @param x X position
         * @param y Y position
         * @return Foreground colour at given position
         */
        public Color getFg(int x, int y) {
            return this.bufferFg[y][x];
        }

        /**
         * Get the width of this buffer.
         * @return Width
         */
```

```java
206        public int getWidth() {
207            return this.width;
208        }
209
210        /**
211         * Get the height of this buffer.
212         * @return Height
213         */
214        public int getHeight() {
215            return this.height;
216        }
217
218        /**
219         * Get the current X position of the cursor.
220         * @return X position of cursor
221         */
222        public int getPosX() {
223            return this.posX;
224        }
225
226        /**
227         * Get the current Y position of the cursor.
228         * @return Y position of cursor
229         */
230        public int getPosY() {
231            return this.posY;
232        }
233    }
```

```java
package uk.insrt.coursework.zuul.util;

import java.lang.reflect.Field;
import java.net.URLClassLoader;
import java.util.Vector;

/**
 * Utilities for detecting we are running in BlueJ.
 *
 * @author Paul Makles <https://insrt.uk>
 * @version 2.0
 */
public class BlueJ {
    /**
     * Whether to ignore deprecation warnings.
     * Enable to allow isRunningInBlueJ() to confidently determine status.
     */
    private static boolean liveOnTheEdge = false;

    /**
     * Check whether this is being exported as BlueJ using maven-bluej
     * https://github.com/KCLOSS/maven-bluej
     * @return Whether this was exported as a BlueJ project.
     */
    public static boolean isExportedAsBlueJ() {
        return BlueJ.class.getResource("/ThisIsABlueJProject") != null;
    }

    /**
     * Detect whether we are currently running under BlueJ.
     * @return Whether we are running from BlueJ.
     */
    public static boolean isRunningInBlueJ() {
        ClassLoader classLoader = Thread.currentThread().getContextClassLoader();

        // When we load the project typically, i.e. from a JAR file, it is instead
        // loaded by jdk.internal.loader.ClassLoaders$PlatformClassLoader and then
        // $AppClassLoader, which we should also see further up the chain from the
        // java.net.URLClassLoader loader.
        if (classLoader instanceof URLClassLoader) {
            if (getJavaVersion() > 8 && !liveOnTheEdge) {
```

```java
42                    // Using setAccessible() as below is deprecated in Java 9 onwards,
43                    // so to avoid any errors in stderr, we can take a safe bet and
44                    // assume that we are in BlueJ given the way we are being loaded.
45                    return true;
46                }

47
48            // We can verify we are running under BlueJ by looping through all
49            // classes which exist on the parent class loader and to check if
50            // a BlueJ class is present.
51            try {
52                // Finding classes loaded by ClassLoader.
53                // https://stackoverflow.com/a/10261850
54                Field f = ClassLoader.class.getDeclaredField("classes");
55                f.setAccessible(true);

56
57                @SuppressWarnings("unchecked")
58                Vector<Class<?>> classes = (Vector<Class<?>>) f.get(classLoader.getParent());

59
60                for (Class<?> cls : classes) {
61                    if (cls.getName().startsWith("bluej.runtime")) {
62                        return true;
63                    }
64                }
65            } catch(NoSuchFieldException | IllegalAccessException | ClassCastException e) {}
66        }

67
68        return false;
69    }

70
71    /**
72     * Gets the current Java version as a single integer.
73     * Taken from https://stackoverflow.com/a/2591122
74     * @return Current Java major version number.
75     */
76    private static int getJavaVersion() {
77        String version = System.getProperty("java.version");
78        return Integer.parseInt(
79            version.startsWith("1.")
80                ? version.substring(2, 3)
81                : (
82                    version.indexOf(".") != -1
```

```
83                        ? version.substring(0, version.indexOf("."))
84                        : version
85                )
86        );
87    }
88 }
```

```java
package uk.insrt.coursework.zuul.util;

import java.io.IOException;
import java.io.InputStream;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;
import java.util.stream.Collectors;

import com.moandjiezana.toml.Toml;

/**
 * This class provides localisation capabilities for the game.
 */
public class Localisation {
    private Map<String, Object> map;

    /**
     * Construct a new instance of Localisation.
     */
    public Localisation() {
        this.map = new HashMap<>();
    }

    /**
     * Load a certain locale by name.
     * @param locale The target locale to load
     * @throws IOException if the locale does not exist in resources
     */
    public void loadLocale(String locale) throws IOException {
        InputStream stream = this
            .getClass()
            .getResourceAsStream("/locale/" + locale + ".toml");

        this.map = new Toml().read(stream).toMap();
    }

    /**
     * Given a path of keys, find the value at the end of the path.
     *
     * Unchecked errors are supressed as they would only occur if the
```

```java
42          * developer provides an incorrect data structure, in that case the
43          * error will be emitted from within this method. It is not a critical
44          * error but it should be handled immediately.
45          * @param path Path to value we want
46          * @return The value at the given path
47          */
48         @SuppressWarnings("unchecked")
49         public String from(String... path) {
50             if (path.length == 0) return "<empty string>";
51
52             try {
53                 var index = 1;
54                 var node = this.map.get(path[0]);
55                 while (index != path.length) {
56                     Map<String, Object> map = (Map<String, Object>) node;
57                     node = map.get(path[index++]);
58                 }
59
60                 if (node != null) {
61                     return (String) node;
62                 }
63             } catch (Exception e) {
64                 // We don't want this to be a fatal error,
65                 // we instead return the original template.
66             }
67
68             return "<" + Arrays.asList(path).stream().collect(Collectors.joining(".")) + ">";
69         }
70
71         /**
72          * Given a path, find the value at the end of the path.
73          * @param path Path to value we want, keys separated by period
74          * @return The value at the given path
75          */
76         public String get(String path) {
77             return this.from(path.split("\\."));
78         }
79     }
```

```java
package uk.insrt.coursework.zuul.util;

import java.util.Arrays;

import uk.insrt.coursework.zuul.entities.Entity;

/**
 * Utilities for searching through data structures related to the game
 */
public class Search {
    /**
     * Find an Entity within an Iterable of entities given certain parameters.
     * @param entities Iterable of Entities which we search through
     * @param name Query which we are matching for
     * @param fuzzy Whether to match whether the alias contains this name in contrast to just doing exact matching
     * @return The Entity if it is found or null
     */
    public static Entity findEntity(Iterable<Entity> entities, String name, boolean fuzzy) {
        String normalised = name.toLowerCase();
        for (Entity entity : entities) {
            String[] aliases = entity.getAliases();
            for (String alias : aliases) {
                if (fuzzy) {
                    if (Arrays.asList(normalised.split("\\s"))
                            .contains(alias)) {
                        return entity;
                    }
                } else if (normalised.equals(alias)) {
                    return entity;
                }
            }
        }

        return null;
    }
}
```

```java
package uk.insrt.coursework.zuul.util;

import java.util.HashMap;
import java.util.List;

/**
 * This is an implementation of a Tree-like data structure.
 * Each node has a one or more children identified by a key
 * and each node can have more children or have a value.
 */
public class Tree<K, V> {
    private HashMap<K, Tree<K, V>> children = new HashMap<>();
    private Tree<K, V> parent;
    private V value;

    /**
     * Construct a new Empty Tree node.
     */
    public Tree() {}

    /**
     * Construct a new Tree node with a parent only.
     * @param parent Tree node which owns this node
     */
    public Tree(Tree<K, V> parent) {
        this.parent = parent;
    }

    /**
     * Construct a new Tree node with parent and value.
     * @param parent Tree node which owns this node
     * @param value The value this node should hold
     */
    public Tree(Tree<K, V> parent, V value) {
        this.parent = parent;
        this.value = value;
    }

    /**
     * Get a child of this Tree node with a given key K.
     * @param key Given key
```

```java
42          * @return The child represented by this key if it exists, otherwise returns null
43          */
44         public Tree<K, V> getChild(K key) {
45             return this.children.get(key);
46         }
47
48         /**
49          * Private method used to accumulate the edges travelled up to the root node.
50          * @param acc Accumulator value
51          * @return The current accumulator value
52          */
53         private int getHeight(int acc) {
54             if (this.parent == null) return acc;
55             return this.parent.getHeight(++acc);
56         }
57
58         /**
59          * The height of the Tree from this point.
60          * This is the number of edges to get from this node to the root node.
61          * @return The height of the tree
62          */
63         public int getHeight() {
64             return this.getHeight(0);
65         }
66
67         /**
68          * Whether this Tree node has a value.
69          * @return True if this node has a value
70          */
71         public boolean hasValue() {
72             return this.value != null;
73         }
74
75         /**
76          * Get the value of this Tree node.
77          * @return Value stored if there is one, otherwise null.
78          */
79         public V getValue() {
80             return this.value;
81         }
82
```

```java
 83        /**
 84         * Add a child to this Tree node represented by a key K.
 85         * @param key Key to represent this new child
 86         * @param node Child node to add
 87         */
 88        public void addChild(K key, Tree<K, V> node) {
 89            this.children.put(key, node);
 90        }
 91
 92        /**
 93         * Recurse through a given key path and add value as a node at the bottom of the path.
 94         * @param keys Keys to iterate through
 95         * @param value Value to add at the end of the path
 96         */
 97        public void addChildWithPath(List<K> keys, V value) {
 98            Tree<K, V> node = this;
 99            while (keys.size() > 0) {
100                K key = keys.remove(0);
101                Tree<K, V> child = node.getChild(key);
102                if (child == null) {
103                    child = new Tree<>(node, keys.size() == 0 ? value : null);
104                    node.addChild(key, child);
105                }
106
107                node = child;
108            }
109        }
110    }
```

```java
package uk.insrt.coursework.zuul.world;

import java.util.Arrays;
import java.util.List;

/**
 * Enum which represents a Cardinal direction.
 */
public enum Direction {
    NORTH(new String[] { "N" }),
    NORTH_EAST(new String[] { "NE", "NORTH EAST" }),
    EAST(new String[] { "E" }),
    SOUTH_EAST(new String[] { "SE", "SOUTH EAST" }),
    SOUTH(new String[] { "S" }),
    SOUTH_WEST(new String[] { "SW", "SOUTH WEST" }),
    WEST(new String[] { "W" }),
    NORTH_WEST(new String[] { "NW", "NORTH WEST" }),

    UP(new String[] {}),
    DOWN(new String[] {});

    private List<String> aliases;

    /**
     * Consturct a new Direction
     * @param aliases Alternative ways to refer to this Direction
     */
    private Direction(String[] aliases) {
        this.aliases = Arrays.asList(aliases);
    }

    /**
     * Check whether this Direction matches the given aliases.
     * @param direction Direction in String format
     * @return Whether it matches.
     */
    private boolean matches(String direction) {
        return this.aliases.contains(direction);
    }

    /**
```

```java
42          * Flip a given Direction in the opposite direction.
43          * @return Direction in the opposite direction.
44          */
45         public Direction flip() {
46             switch (this) {
47                 default:
48                 case NORTH: return Direction.SOUTH;
49                 case NORTH_EAST: return Direction.SOUTH_WEST;
50                 case EAST: return Direction.WEST;
51                 case SOUTH_EAST: return Direction.NORTH_WEST;
52                 case SOUTH: return Direction.NORTH;
53                 case SOUTH_WEST: return Direction.NORTH_EAST;
54                 case WEST: return Direction.EAST;
55                 case NORTH_WEST: return Direction.SOUTH_EAST;
56                 case UP: return Direction.DOWN;
57                 case DOWN: return Direction.UP;
58             }
59         }
60
61         /**
62          * Convert an arbitrary String to a Direction.
63          * @param direction Raw string representing a Direction
64          * @return Direction or null from given string
65          */
66         public static Direction fromString(String direction) {
67             if (direction == null) return null;
68
69             String directionFormatted = direction.toUpperCase();
70             try {
71                 return Direction.valueOf(directionFormatted);
72             } catch (Exception ex) {
73                 for (Direction dir : Direction.values()) {
74                     if (dir.matches(directionFormatted)) {
75                         return dir;
76                     }
77                 }
78
79                 return null;
80             }
81         }
82     }
```

```java
package uk.insrt.coursework.zuul.world;

import uk.insrt.coursework.zuul.entities.Inventory;

/**
 * Representation of a physical location in the world,
 * whether it is a room or inventory or neither but not both.
 */
public class Location {
    private Room room;
    private Inventory inventory;

    /**
     * Construct a new Location outside of the World.
     */
    public Location() {}

    /**
     * Construct a new Location pointing to a Room.
     * @param room Room
     */
    public Location(Room room) {
        this.room = room;
    }

    /**
     * Construct a new Location pointing to an Inventory.
     * @param inventory Inventory
     */
    public Location(Inventory inventory) {
        this.inventory = inventory;
    }

    /**
     * Change this Location to point to a Room.
     * @param room Room
     */
    public void setLocation(Room room) {
        this.room = room;
        this.inventory = null;
    }
```

```java
42
43        /**
44         * Change this Location to point to an Inventory.
45         * @param inventory Inventory
46         */
47        public void setLocation(Inventory inventory) {
48            this.room = null;
49            this.inventory = inventory;
50        }
51
52        /**
53         * Reset the Location and put us outside of the World.
54         */
55        public void clear() {
56            this.room = null;
57            this.inventory = null;
58        }
59
60        /**
61         * Get the current Room this Location represents.
62         * @return Room or null if in an inventory or out of the World.
63         */
64        public Room getRoom() {
65            return this.room;
66        }
67
68        /**
69         * Get the current Inventory this Location represents.
70         * @return Inventory or null if in a room or out of this World.
71         */
72        public Inventory getInventory() {
73            return this.inventory;
74        }
75    }
```

```java
package uk.insrt.coursework.zuul.world;

import java.util.HashMap;
import java.util.Set;

/**
 * Representation of a Room within the World.
 * Handles how entities can move from this to other Rooms.
 */
public abstract class Room {
    private World world;
    private String name;
    private HashMap<Direction, Room> adjacentRooms;

    /**
     * Construct a new Room in a given World with a given name.
     * @param world World
     * @param name Internal name used to refer to this Room
     */
    public Room(World world, String name) {
        this.world = world;
        this.name = name;
        this.adjacentRooms = new HashMap<>();
    }

    /**
     * Get the World that this Room is in.
     * @return World
     */
    public World getWorld() {
        return this.world;
    }

    /**
     * Get the internal name of this Room.
     * @return Internal name
     */
    public String getName() {
        return this.name;
    }
```

```java
42          /**
43           * Make another Room adjacent to this Room in a particular Direction.
44           * @param direction Direction the other Room is in
45           * @param room Room which we are making adjacent
46           */
47          public void setAdjacent(Direction direction, Room room) {
48              if (room == null) System.err.println("Warning: assigned null Room to direction " + direction + " for the Room " +
this.name);
49              this.adjacentRooms.put(direction, room);
50          }
51
52          /**
53           * Get an adjacent Room in a particular Direction.
54           * @param direction Direction to look at
55           * @return The Room if one is present in that Direction, otherwise null
56           */
57          public Room getAdjacent(Direction direction) {
58              return this.adjacentRooms.get(direction);
59          }
60
61          /**
62           * Whether the player can leave in any particular direction.
63           * Should print reason if not.
64           * @param direction Direction which we are checking
65           * @return Whether the player can leave
66           */
67          public boolean canLeave(Direction direction) {
68              return true;
69          }
70
71          /**
72           * Get Directions that you can leave this Room in.
73           * @return Set of Directions we can leave in
74           */
75          public Set<Direction> getDirections() {
76              return this.adjacentRooms.keySet();
77          }
78
79          /**
80           * Check whether there is an exit in a particular Direction.
81           * @param direction Direction to check
```

```java
 82        * @return True if there is an exit in a given Direction
 83        */
 84       public boolean hasExit(Direction direction) {
 85           return this.adjacentRooms.containsKey(direction);
 86       }
 87
 88       /**
 89        * Reset adjacent Rooms and reconfigure adjacent Rooms.
 90        * This should be called after all Rooms have been spawned into the World.
 91        */
 92       public void linkRooms() {
 93           this.adjacentRooms.clear();
 94           this.setupDirections();
 95       }
 96
 97       /**
 98        * Spawn Entities in this World.
 99        * By default, nothing is done but this should be used further up to spawn
100        * the Entities for this particular Room.
101        */
102       public void spawnEntities() {}
103
104       /**
105        * Convert this Room into a Location.
106        * @return Location representation of Room
107        */
108       public Location toLocation() {
109           return new Location(this);
110       }
111
112       /**
113        * Describe what this Room looks like.
114        * @return Description of this Room
115        */
116       public abstract String describe();
117
118       /**
119        * Setup adjacent Rooms.
120        */
121       protected abstract void setupDirections();
122   }
```

```java
package uk.insrt.coursework.zuul.world;

import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

import uk.insrt.coursework.zuul.entities.Entity;
import uk.insrt.coursework.zuul.entities.EntityPlayer;
import uk.insrt.coursework.zuul.events.Event;
import uk.insrt.coursework.zuul.events.EventSystem;
import uk.insrt.coursework.zuul.io.IOSystem;

/**
 * Representation of the game World.
 * Contains all the Rooms and Entities as well as the Player.
 * Has its own Event system for signaling when things should happen.
 * Also has access to the IO system which is provided to all Rooms and Entities.
 */
public class World {
    protected Map<String, Room> rooms = new HashMap<>();
    protected Map<String, Entity> entities = new HashMap<>();
    protected EntityPlayer player;

    protected IOSystem io;
    protected EventSystem eventSystem;

    /**
     * Consturct a new game World with a given IO system.
     * @param io IO system to provide to everything
     */
    public World(IOSystem io) {
        this.io = io;
        this.eventSystem = new EventSystem();
        this.player = new EntityPlayer(this);
        this.entities.put("player", this.player);
    }

    /**
     * Find an Entity by its ID.
     * @param id Entity ID
```

```java
42          * @return Entity if it exists, otherwise null.
43          */
44         public Entity getEntity(String id) {
45             return this.entities.get(id);
46         }
47
48         /**
49          * Find an Room by its ID.
50          * @param room Room ID
51          * @return Room if it exists, otherwise null.
52          */
53         public Room getRoom(String room) {
54             return this.rooms.get(room);
55         }
56
57         /**
58          * Get the Player entity.
59          * @return The player entity
60          */
61         public EntityPlayer getPlayer() {
62             return this.player;
63         }
64
65         /**
66          * Get the IO system provided to this World.
67          * @return IO system
68          */
69         public IOSystem getIO() {
70             return this.io;
71         }
72
73         /**
74          * Get this World's event system.
75          * @return World event system
76          */
77         public EventSystem getEventSystem() {
78             return this.eventSystem;
79         }
80
81         /**
82          * Add a Room to this World.
```

```java
 83         * @param room Room to add
 84         */
 85        protected void addRoom(Room room) {
 86            this.rooms.put(room.getName(), room);
 87        }
 88
 89        /**
 90         * Spawn a new Entity in the World.
 91         * @param id Unique Entity ID
 92         * @param entity Entity to spawn
 93         */
 94        public void spawnEntity(String id, Entity entity) {
 95            this.entities.put(id, entity);
 96        }
 97
 98        /**
 99         * Get all the Entities found in a given Room.
100         * @param room Room to search for
101         * @return List of Entities in the World in a given Room
102         */
103        public List<Entity> getEntitiesInRoom(Room room) {
104            return this
105                .entities
106                .values()
107                .stream()
108                .filter(e -> e.getRoom() == room)
109                .collect(Collectors.toList());
110        }
111
112        protected void linkRooms() {
113            for (Room room : this.rooms.values()) {
114                room.linkRooms();
115            }
116        }
117
118        public void emit(Event event) {
119            this.eventSystem.emit(event);
120        }
121
122        /**
123         * Try to spawn the player in the first available room.
```

```java
        */
    public void spawnPlayer() {
        this.player.setLocation(this.rooms.values().iterator().next());
    }
}
```

```toml
emojis = [
    { unicode = "\u1F633", path = "flosh.png" }, #  😳
    { unicode = "\u1F601", path = "trol.png" }, #  😁
    { unicode = "\u1F438", path = "monkaStare.png" }, #  🐸
    { unicode = "\u1F642", path = "pauseChamp.png" }, #  🙂
    { unicode = "\u1F610", path = "weirdChamp.png" }, #  😐
    { unicode = "\u1F604", path = "peepoHappy.png" }, #  😄
    { unicode = "\u1F99D", path = "they is stuck.jpg", width = 24, height = 8 }, #  🦝

    # Example definitions:
    #{ unicode = "[wideChamp]", path = "weirdChamp.png" },
    #{ unicode = "[widePeepo]", path = "peepoHappy.png" },
    #{ unicode = "😄", path = "peepoHappy.png", width = 8, height = 4 },
]
```

```toml
1   # Translations for World of Deez
2
3   [global]
4   sight = "You can see:"
5
6       [global.can_go_in_x_directions]
7       1 = "You may go in"
8       2 = "directions"
9
10  [selectors]
11  direction = "<direction>"
12  something = "<something>"
13  someone = "<someone>"
14  item = "<item>"
15
16      [selectors.cant_find]
17      1 = "You look around for"
18      2 = "but can't find anything"
19
20  [commands]
21  unknown = "Not sure what you're trying to do."
22
23  back = "go back to the previous room"
24  quit = "quit the game"
25  where_am_i = "describe the current room again"
26
27      [commands.bag]
28      usage = "look inside your bag or at something's inventory"
29      cant_find = "Can't find what you want to look at."
30      empty = "Your bag is empty!"
31      entity_empty = "doesn't appear to have anything"
32      can_carry_kg = "You can carry"
33      are_carrying_kg = "You are carrying"
34      look_in_bag = "You look in your bag to see"
35      entity_appears_to_have = "appears to have"
36
37      [commands.drop]
38      usage = "drop an item from your bag"
39      nothing_specified = "What do you want to drop?"
40
41          [commands.drop.dropped]
```

```toml
42          1 = "You drop"
43          2 = "out of your bag"
44
45      [commands.give]
46      usage = "give something to someone"
47      nothing_specified = "What do you want to give?"
48      no_target = "What / who are you putting this in?"
49      denied_player = "You cannot give yourself to anyone or anything. \u1F438"
50
51          [commands.give.denied]
52          1 = "Cannot give"
53          2 = "to"
54
55      [commands.go]
56      usage = "go in a certain direction"
57      nothing_specified = "Where are you going?"
58
59      [commands.help]
60      usage = "show help menu"
61      can_run = "You can run the following commands:"
62
63      [commands.pet]
64      usage = "pet something around you or in your inventory"
65      nothing_specified = "What are you trying to pet?"
66      denied = "You cannot pet"
67
68      [commands.take]
69      usage = "put something in your bag"
70      nothing_specified = "From who?"
71      entity_does_not_have_entity = "does not have"
72      item_not_specified = "What do you want to take?"
73
74          [commands.take.took]
75          1 = "You take"
76          2 = "from"
77          3 = "and put it in your bag"
78
79          [commands.take.denied]
80          1 = "You cannot take"
81          2 = "it's too heavy to put in your bad"
82
```

```toml
 83        [commands.talk]
 84        usage = "start talking with someone"
 85        nothing_specified = "What do you want to talk with?"
 86        denied = "You cannot talk with"
 87
 88        [commands.use]
 89        usage = "use something around you or in your inventory"
 90        nothing_specified = "What do you want to use?"
 91        denied = "You cannot use"
 92
 93        [commands.map]
 94        usage = "show the world map"
 95        close = "Press enter to close."
 96
 97            [commands.map.discovered]
 98            1 = "You have discovered"
 99            2 = "of the world"
100
101        [commands.win]
102        usage = "win the game"
103        conclusion = """...
104
105    Chapter 4.:
106    The files are released into the internet for anyone to read, people are quick to
107    analyse through every single tiny detail, some immediate details come to light:
108    - Slyvasta was researching ethically questionable areas of science, in
109      particular, they were running hundreds of tests daily on a variety of beastman
110      test subjects. But nobody could prove anyone was there against their will.
111    - However, the research did also get immediately picked up by foreign powers who
112      quickly discovered that the beastman society living in the city is a far
113      greater threat than they initially anticipated.
114    - The city, with Sylvasta's public advice, immediately ordered evacuation of all
115      citizens to any area they could find fearing a potentially deadly conflict on
116      the horizon. The city quickly came under fire over the coming days.
117    """
118        stats = "\u001B[47m\u001B[30mYour final game stats\u001B[0m"
119        total_ticks = "Total game ticks: "
120        sidequests_complete = "Side-quests completed: "
121        press_enter_key = "Press enter to close the game."
122
123    [entities]
```

```toml
124    boat_key = "A key to the speed boat"
125
126        [entities.bed]
127        description = "Bed"
128        use = "You take a nap."
129
130        [entities.boat]
131        description = "Speedboat docked on the coast"
132        locked = "The boat is locked."
133        locked_for_sale = "The boat is locked.\nThere is a note which says to contact the shopkeeper to buy this boat."
134        denied = "You must not be carrying anything to use the boat.\nYou can however put things in the boat."
135        travel = "You hop in the boat and travel to the other side..."
136        too_heavy = "The boat is carrying too much stuff already!"
137
138            [entities.boat.give]
139            1 = "Put"
140            2 = "in the boat"
141
142        [entities.cat]
143        description = "A stray black cat"
144        pet = "You pet the cat."
145        use = "You cannot the cat.\nPlease do not the cat. \u1F633\u1F633\u1F633"
146        enter = "A cat has wandered in."
147        leave = "You see a cat leave."
148
149        [entities.comms]
150        description = "Communicator device"
151        off = "The device is off."
152
153        [entities.couch]
154        description = "A brown leather couch"
155        sitting = "You are already sitting in the couch."
156        sit = "You sit down on the couch."
157
158        [entities.laptop]
159        description = "Laptop"
160
161            [entities.laptop.boot]
162            dialog = "You turn the computer on..."
163            option_1 = "[wait]"
164
```

```toml
165        [entities.laptop.home]
166        dialog = "Select an option:"
167        option_q = "Power off"
168        option_1 = "/My Pictures"
169        option_2 = "/Funny cat videos"
170        option_3 = "/Marie's document scanner"
171
172        [entities.laptop.pictures]
173        dialog = """There is only one picture in your pictures folder:

\u1F99D




"""
184        option_q = "Go back."
185
186        [entities.laptop.cat_videos]
187        dialog = "You look at funny cat videos..."
188        option_q = "Neat."
189
190        [entities.laptop.document]
191        dialog = """\u001B[35mMarie\u001B[0m's document scanner"""
192        option_q = "Quit"
193        option_1 = "Send documents"
194
195 [home]
196 first_load = '''
197 You're about to be placed into the world.
198 If at any point you are stuck with what to do,
199 you can use `help` to view all available commands.
200
201 ---
202
203 You wake up to the sound of people chanting outside..
204 You really should've closed the window last night..
205
```

```
206    Curious, you peer out the window to see what's going on.. there's a group of
207    protestors outside the Medical Centre down the street, you can't really make out
208    what they're saying or what their signs say.
209
210    Though it'd not be surprising if something strange is going on in there, but you
211    can't really put your finger on it. Maybe there's something on the news..
212    '''
213
214    enter = '''
215    You enter your apartment.
216    '''
217
218        [home.tv]
219        description = "LG 55NANO966PA 55\" Super UHD 8K HDR Smart LED TV"
220        off = "Turn the TV off."
221        keep_watching = "Keep watching..."
222
223            # Red: \u001B[31m
224            # Green: \u001B[32m
225            # Yellow: \u001B[33m
226            # Cyan: \u001B[36m
227            [home.tv.first_on]
228            dialog = "You turn the TV on.\n\nThe news channel comes up..."
229            dialog_a="""\u001B[31mNews Anchor\u001B[0m: Civil unrest is rising, Sylvasta is facing criticism from all
230    sides, and many people are uneasy about their future as rising tension between
231    human and beastman societies is causing escalated conflict around the city
232    borders.
233
234    \u001B[31mNews Anchor\u001B[0m: We bring you now to scenes outside of the Medical Centre where a
235    group of protestors have shown up in opposition to the research being led at
236    Slyvasta.
237
238    \u001B[36mCorrespondent\u001B[0m: I am here, standing outside with the group of protestors..
239
240    \u001B[36mCorrespondent\u001B[0m to \u001B[33mProtestor\u001B[0m: What brings you here today?
241    """
242
243            dialog_b="""\u001B[33mProtestor\u001B[0m:
244    Protestor: They're taking the city's money and using it for their own gain, they
245    shouldn't receive any funding let alone be allowed to operate here.
246
```

```
247  \u001B[31mNews Anchor\u001B[0m: Bold claims coming straight from outside Sylvasta, whether these
248  claims are grounded in anything is yet to be discovered, we've seen months and
249  months of leaks come out from former employees and internal mishaps but are yet
250  to truly find out the intention behind the people at Sylvasta.
251
252  \u001B[31mNews Anchor\u001B[0m: Sylvasta has personally announced that they refuse to communicate
253  or elaborate any further on their internal research citing public safety,
254  whatever that means nobody outside of their internal staff knows.
255  """
256
257  dialog_c="""\u001B[31mNews Anchor\u001B[0m: It also begs the question whether the protests are unfounded and
258  just there to stir up trouble in the city. Earlier today we also spoke to local
259  residents living in the centre of the city..
260
261  \u001B[32mShopkeeper\u001B[0m: I think these guys just want to cause trouble, Sylvasta was vital in
262  establishing this city and letting us live in peace without having to worry
263  about being attacked, I just don't think there's enough reason to protest.
264
265  \u001B[32mShopkeeper\u001B[0m: We're already seeing the world turn against the city and these sorts
266  of internal conflicts will just give them reason to step in and take control.
267
268  \u001B[31mNews Anchor\u001B[0m: That concludes our broadcast for this morning, and we'll be back
269  for the 1pm News Hour.
270  """
271
272  [apartments]
273  enter = '''
274  You enter the apartment complex reception.
275  '''
276
277      # Cyan: \u001B[36m
278      [apartments.receptionist]
279      description = "The receptionist sitting behind a desk"
280
281          [apartments.receptionist.first_encounter]
282          dialog = "\u001B[36mReceptionist\u001B[0m: Good morning, how are you doing today?"
283          option_1 = "What is that racket outside?"
284          option_q = "Nevermind."
285
286          [apartments.receptionist.protestors]
287          dialog = """\u001B[36mReceptionist\u001B[0m: I don't know much but it looks like a group of people shouting
```

```
288   outside the Medical Centre..."""
289           option_1 = "Do you know anything more?"
290           option_q = "Alright."
291
292           [apartments.receptionist.protestors2]
293           dialog = """\u001B[36mReceptionist\u001B[0m: They were handing out flyers as they came up, maybe someone
294   nearby would know..."""
295           option_q = "Thanks."
296
297           [apartments.receptionist.repeated]
298           dialog = "\u001B[36mReceptionist\u001B[0m: Hello, what can I help you with?"
299           option_1 = "What's going on with those protestors outside?"
300           option_q = "That's all, thanks."
301
302   [city_centre]
303   first_load = '''
304   You enter the city square, it's quite busy in the mornings. There's a lot of
305   people running around, unlikely you could stop most of them for a chat.
306
307   There's a general unease in the area, some people look quite tense and others
308   look like they're spending their last day on Earth...
309   '''
310
311   enter = '''
312   You enter the city square.
313   '''
314
315       # Yellow: \u001B[33m
316       [city_centre.npc]
317   description = "A person sitting at a bench"
318
319           [city_centre.npc.small_talk]
320           dialog = "\u001B[33mStranger\u001B[0m: Hello, could I help you?"
321           option_1 = "You don't happen to have seen the protestors go by here?"
322           option_2 = "What are you reading?"
323           option_q = "Nevermind."
324
325           [city_centre.npc.protestors]
326           dialog = """\u001B[33mStranger\u001B[0m: Oh yes, they were quite loud..
327   They were rather annoying, disturbed my morning.
328   ...
```

```toml
329    They did leave me this leaflet though"""
330            option_1 = "Can I see it?"
331
332            [city_centre.npc.enquire]
333            dialog = "\u001B[33mStranger\u001B[0m: These guys went past and left me a leaflet."
334
335            [city_centre.npc.leaflet]
336            # dialog = "\u001B[33mStranger\u001B[0m: Sure, here you go, keep it.\n[\u001B[33mStranger\u001B[0m gave you an
item.]"
337            dialog = """\u001B[33mStranger\u001B[0m: Sure, I've got it here...
338
339    You look at the leaflet:
340
341    \u001B[41mUNETHICAL RESEARCH
342    \u001B[40m\u001B[31mOur city is in danger, Slyvasta further
343    funding into their research yet do nothing
344    to help the city survive.
345
346    Sylvasta is abusing their position to
347    continue work into completely ILLEGAL
348    bioengineering of beastmen.\u001B[0m
349
350    Support us: \u001B[36mhttps://sylvasta.vercel.app\u001B[0m
351    """
352            option_1 = "Thanks."
353
354    [street]
355    first_load = '''
356    You enter the main city street connecting the major parts of the city.
357
358    There's a lot of chaos and noise here, to your west you can see protestors
359    outside of the medical centre complex, they're holding signs up, "UNETHICAL
360    RESEARCH", "OUR RELATIONS AT STAKE", "UNFORSEEN CONSEQUENCES".
361    '''
362
363    enter = '''
364    You enter the city street.
365    '''
366
367        # Red: \u001B[31m
368        [street.protestors]
```

```toml
369        description = "A group of protestors holding signs"
370        blocking = "There is a group of protestors blocking the way in."
371
372            [street.protestors.small_talk]
373            dialog = "\u001B[31mProtestor\u001B[0m: Have you heard about what Sylvasta is doing and why we're out here?"
374            option_1 = "No, enlighten me."
375            option_2 = "Nevermind."
376
377            [street.protestors.enquire]
378            dialog = """\u001B[31mProtestor\u001B[0m: They are using the city's money to further their frankly illegal
379    research, you've seen the leaks right?"""
380            option_1 = "No."
381            option_2 = "Right.. I'll leave you to it."
382
383            [street.protestors.leaks]
384            dialog = """\u001B[31mProtestor\u001B[0m: Well some former employees leaked information about their current
385    research projects and from what we can tell is that they're doing bioengineering
386    on beastmen, this shouldn't be tolerated let alone funded by the city."""
387            option_1 = "Ok."
388
389            [street.protestors.confrontation]
390            dialog = "\u001B[31mProtestor\u001B[0m: You taking me for some sort of conspiracy theorist?"
391            option_1 = "Yes."
392            option_2 = "I'm leaving now."
393
394    [shop]
395    enter = '''
396    You enter the shop, the bell rings as you close the door behind you.
397    '''
398
399        # Green: \u001B[32m
400        [shop.npc]
401        description = "The shop keeper"
402        leave = "Nevermind."
403
404        currently_have_amount_of_money = "You currently have:"
405        not_enough = "You don't have enough money to buy"
406        too_heavy = "Too heavy for you to carry!"
407
408        out_of_stock = "out of stock"
409        x_left = "left"
```

```toml
        [shop.npc.fake_item]
        cat = "A singular cat"

        [shop.npc.item_out_of_stock]
        1 = "This item is"
        2 = "you may not buy it"

        [shop.npc.bought]
        1 = "You buy"
        2 = "and put it in your bag"

        [shop.npc.greeting]
        Exposition = "\u001B[32mShopkeeper\u001B[0m: Good day, what would you like to buy?"
        Recon = "\u001B[32mShopkeeper\u001B[0m: Hello, anything in mind today?"
        Stealth = "\u001B[32mShopkeeper\u001B[0m: Looking to buy something?"
        End = "\u001B[32mShopkeeper\u001B[0m: Need anything new?"

[back_alley]
first_load = '''
You enter a dark alley, it is quite quiet here, a stark contrast to the city
centre yet located nearly right in the middle. The sides of the alley are lined
with derelict buildings and dim lighting.

You make out a familiar figure further ahead.
'''

enter = '''
You enter the back alley.
'''

[coastline]
enter = '''
You arrive at the city's coastline, the water is rather calm and still.
'''

[mainland_coastline]
enter = '''
You arrive at the coastline on the mainland,
the sea is crashing against rocks lined against it,
it is significantly colder out here away from the concrete jungle.
```

```toml
451    '''
452
453    [forest]
454    enter = '''
455    You wander through the forest, you hear birds whistling at the peaks of trees...
456    There's a river flowing through the middle with an old wooden bridge above it.
457    It would be quite easy to lose yourself here for a few hours.
458    '''
459
460        # Yellow: \u001B[33m
461        [forest.old_man]
462        description = "An old man wandering through the forest"
463        full = "There's nothing else you can give them."
464        accept = "\u001B[33mOld Man\u001B[0m: Thank you for bringing him back to me."
465        deny = "\u001B[33mOld Man\u001B[0m: I don't want your"
466
467            [forest.old_man.small_talk]
468            dialog = "\u001B[33mOld Man\u001B[0m: You haven't seen my cat anywhere have you?"
469            option_1 = "I've seen this black stray around town..."
470            option_q = "No."
471
472            [forest.old_man.request]
473            dialog = "\u001B[33mOld Man\u001B[0m: Could you bring him back to me?"
474            option_1 = "Sure."
475            option_q = "Sorry, not right now."
476
477            [forest.old_man.praise]
478            dialog = "\u001B[33mOld Man\u001B[0m: Thank you for helping me!"
479            option_q = "[leave]"
480
481    [worm_hole]
482    enter = '''
483    You step into the worm hole...
484    '''
485
486    [medical_centre]
487    enter = '''
488    You're now at the Medical Centre's reception.
489    '''
490
491        # Red: \u001B[31m
```

```toml
492        [medical_centre.guard]
493        description = "Security guard stationed at the door"
494        blocking = "There is security watching the stairs, there's no way to get past them."
495
496            [medical_centre.guard.small_talk]
497            dialog = "\u001B[31mGuard\u001B[0m: What do you want?"
498            option_q = "Nevermind."
499
500    [medical_centre_office]
501    enter = '''
502    You find yourself at the Medical Centre's office.
503    You definitely shouldn't be here...
504    '''
505
506        [medical_centre_office.books]
507            [medical_centre_office.books.1]
508            title = "sus"
509            contents = "gdfghdfs"
510
511            [medical_centre_office.books.2] # yes
512            title = "sus"
513            contents = "gdfghdfs"
514
515            [medical_centre_office.books.3]
516            title = "sus"
517            contents = "gdfghdfs"
518
519            [medical_centre_office.books.4] # yes
520            title = "sus"
521            contents = "gdfghdfs"
522
523            [medical_centre_office.books.5] # yes
524            title = "sus"
525            contents = "gdfghdfs"
526
527            [medical_centre_office.books.6]
528            title = "sus"
529            contents = "gdfghdfs"
530
531    [marie]
532    description = "\u001B[35mMarie the Mink\u001B[0m"
```

```toml
533
534        # Purple: \u001B[35m
535        [marie.alley]
536
537            [marie.alley.small_talk]
538            dialog = "\u001B[35mMarie\u001B[0m: What brings you here?"
539            option_1 = "Did you hear about the protests?"
540            option_2 = "Sylvasta is up to something..."
541            option_q = "Nevermind, have a good day."
542
543            [marie.alley.protests]
544            dialog = """\u001B[35mMarie\u001B[0m: Yeah, those Sylvasta guys are definitely up to
545   something but I don't know whether I believe the crowds showing up."""
546            option_1 = "Do you want to help me find out?"
547            option_q = "Nevermind."
548
549            [marie.alley.sylvasta]
550            dialog = "\u001B[35mMarie\u001B[0m: You think I don't know? You want to help me find out?"
551            option_1 = "Sure."
552            option_q = "No thanks."
553
554            [marie.alley.confirm]
555            dialog = "\u001B[35mMarie\u001B[0m: I suppose we could break in and find out the truth."
556            option_1 = "Let's do it! [progress story]"
557            option_q = "No way!"
558
559            [marie.alley.recon]
560            dialog = """\u001B[35mMarie\u001B[0m: Ok, while I go figure out how we get in, I need you to fetch something we
561   can talk over, I'm sure the shop will have something, meet me back here when you
562   get something."""
563            option_q = "Got it."
564
565            [marie.alley.waiting]
566            dialog = "\u001B[35mMarie\u001B[0m: Well? Go get it."
567            option_q = "Ok."
568
569            [marie.alley.mission_brief]
570            dialog = """\u001B[35mMarie\u001B[0m: Ok so the plan is, you go to the Medical Centre, casually stroll in to
571   the reception, try to blend in though... Maybe find a chair to sit on or
572   something, either way, once you're there and you think security isn't looking,
573   use your comms device to let me know.
```

```
574
575     \u001B[35mMarie\u001B[0m: From there, I will send a distraction and lead them into their breakroom
576     while you sneak into the office and try to find anything you can... although
577     once you're down there, let me know.
578
579     \u001B[35mMarie\u001B[0m: If you want to, you can test the comms device now."""
580             option_q = "Ok."
581
582         [marie.comms]
583         received = """\u001B[35mMarie\u001B[0m: Alright, I've received the documents, I'm sending them out far and wide,
584     good job on getting in and getting these."""
585         bad_documents = "\u001B[35mMarie\u001B[0m: These don't look like the right documents..."
586         no_access = "\u001B[31mAccess denied!\u001B[0m"
587
588             [marie.comms.orientation]
589             dialog = "\u001B[35mMarie\u001B[0m: What do you want?"
590             option_1 = "Where do I go again?"
591             option_2 = "Can you hear me?"
592             option_q = "Nevermind."
593
594             [marie.comms.directions]
595             dialog = """\u001B[35mMarie\u001B[0m: You need to go to the Medical Centre, once you're there, try to blend in
596     and let me know once you're in position."""
597             option_q = "Got it."
598
599             [marie.comms.hear]
600             dialog = "\u001B[35mMarie\u001B[0m: Loud and clear."
601             option_q = "Neat."
602
603             [marie.comms.complaint]
604             dialog = "\u001B[35mMarie\u001B[0m: Try to blend in..."
605             option_q = "Ok."
606
607             [marie.comms.in_position]
608             dialog = "\u001B[35mMarie\u001B[0m: Ok, I'll create a distraction, give me a moment..."
609             option_1 = "[wait]"
610
611             [marie.comms.distraction]
612             dialog = """\u001B[31mGuard\u001B[0m: Who's there?
613     You better come out or we're going to have a problem.
614
```

```
615     (you see the guard go into the breakroom)
616     """
617             option_1 = "[wait]"
618
619             [marie.comms.coast_is_clear]
620             dialog = "\u001B[35mMarie\u001B[0m: Ok, now is your chance."
621             option_q = "Got it."
622
623             [marie.comms.gogogo]
624             dialog = "\u001B[35mMarie\u001B[0m: What are you waiting for??"
625             option_q = "AAAAAAA"
626
627             [marie.comms.office]
628             dialog = "\u001B[35mMarie\u001B[0m: What do you see?"
629             option_1 = "A bunch of documents scattered around."
630
631             [marie.comms.documents]
632             dialog = """\u001B[35mMarie\u001B[0m: Ok, look through them and find anything that looks relevant, look out for
633     anything that mentions the stuff they were protesting about. Once you do, you'll
634     need to upload them to me, I guess go back to your apartments and we'll do it
635     from there, let me know once you're there.
636
637     \u001B[35mMarie\u001B[0m: I'll be making sure your cover is not blown.
638
639     \u001B[33mHint: you can use the documents to read them.\u001B[0m"""
640             option_q = "Ok."
641
642             [marie.comms.home]
643             dialog = """\u001B[35mMarie\u001B[0m: Presumably you have the documents, turn your computer on, go to the link
644     I've sent you, it should show up, and upload them."""
645
646     [stage]
647     reached_conclusion = """\n
648     You've reached the end of the game, you're now in an open world mode, keep
649     exploring anything you haven't to maximise your score.
650
651     You can conclude by typing \u001B[47m\u001B[30mwin\u001B[0m
652     """"
```

```toml
# 1. Marie located in Back Alley
[npc_marie]
    _prefix = "marie.alley."
    _start = "small_talk"
    small_talk = { description = "small_talk.dialog", options = [
        { description = "small_talk.option_1", to = "protests" },
        { description = "small_talk.option_2", to = "sylvasta" },
        { description = "small_talk.option_q", to = "small_talk", mustExit = true }
    ] }
    protests = { description = "protests.dialog", options = [
        { description = "protests.option_1", to = "confirm" },
        { description = "protests.option_q", to = "small_talk", mustExit = true }
    ] }
    sylvasta = { description = "sylvasta.dialog", options = [
        { description = "sylvasta.option_1", to = "confirm" },
        { description = "sylvasta.option_q", to = "small_talk", mustExit = true }
    ] }
    confirm = { description = "confirm.dialog", options = [
        { description = "confirm.option_q", to = "small_talk", mustExit = true }
    ] }

    recon = { description = "recon.dialog", options = [
        { description = "recon.option_q", to = "waiting", mustExit = true }
    ] }
    waiting = { description = "waiting.dialog", options = [
        { description = "waiting.option_q", to = "waiting", mustExit = true }
    ] }

    mission_brief = { description = "mission_brief.dialog", options = [
        { description = "mission_brief.option_q", to = "mission_brief", mustExit = true }
    ] }

# 2. Random City NPC located in the City Centre
[npc_city_centre]
    _prefix = "city_centre.npc."
    _start = "small_talk"
    small_talk = { description = "small_talk.dialog", options = [
        { description = "small_talk.option_1", to = "protestors" },
        { description = "small_talk.option_2", to = "enquire" },
        { description = "small_talk.option_q", to = "small_talk", mustExit = true }
    ] }
```

```toml
42      protestors = { description = "protestors.dialog", options = [
43          { description = "protestors.option_1", to = "leaflet" },
44          { description = "small_talk.option_q", to = "small_talk", mustExit = true }
45      ] }
46      enquire = { description = "enquire.dialog", options = [
47          { description = "protestors.option_1", to = "leaflet", },
48          { description = "small_talk.option_q", to = "small_talk", mustExit = true }
49      ] }
50      leaflet = { description = "leaflet.dialog", options = [
51          { description = "leaflet.option_1", to= "small_talk", mustExit = true }
52      ] }
53      #recon = { description = "testing", options = [
54      #    { description = "sususus!", to = "recon", mustExit = true }
55      #] }
56
57  # 3. Old Man located in Forest
58  [npc_old_man]
59      _prefix = "forest.old_man."
60      _start = "small_talk"
61      small_talk = { description = "small_talk.dialog", options = [
62          { description = "small_talk.option_1", to = "request" },
63          { description = "small_talk.option_q", to = "small_talk", mustExit = true }
64      ] }
65      request = { description = "request.dialog", options = [
66          { description = "request.option_1", to = "small_talk", mustExit = true },
67          { description = "request.option_q", to = "small_talk", mustExit = true }
68      ] }
69      praise = { description = "praise.dialog", options = [
70          { description = "praise.option_q", to = "praise", mustExit = true }
71      ] }
72
73  # 4. Protestors located on Street
74  [npc_protestors]
75      _prefix = "street.protestors."
76      _start = "small_talk"
77      small_talk = { description = "small_talk.dialog", options = [
78          { description = "small_talk.option_1", to = "enquire" },
79          { description = "small_talk.option_2", to = "small_talk", mustExit = true }
80      ] }
81      enquire = { description = "enquire.dialog", options = [
82          { description = "enquire.option_1", to = "leaks" },
```

```
 83            { description = "enquire.option_2", to = "confrontation" }
 84        ] }
 85        leaks = { description = "leaks.dialog", options = [
 86            { description = "leaks.option_1", to = "small_talk", mustExit = true }
 87        ] }
 88        confrontation = { description = "confrontation.dialog", options = [
 89            { description = "confrontation.option_1", to = "small_talk", mustExit = true },
 90            { description = "confrontation.option_2", to = "small_talk", mustExit = true }
 91        ] }
 92
 93    # 5. Security guard stationed at Medical Centre
 94    [npc_security_guard]
 95        _prefix = "medical_centre.guard."
 96        _start = "small_talk"
 97        small_talk = { description = "small_talk.dialog", options = [
 98            { description = "small_talk.option_q", to = "small_talk", mustExit = true }
 99        ] }
100
101    # 6. Shop shopkeeper in the Shop
102    [npc_shopkeeper]
103        _prefix = "shop.npc."
104        _start = "index"
105        index = { description = "index", options = [
106            { description = "leave", to = "index", mustExit = true }
107        ] }
108        recon = { description = "recon", options = [
109            { description = "leave", to = "recon", mustExit = true }
110        ] }
111        stealth = { description = "stealth", options = [
112            { description = "leave", to = "stealth", mustExit = true }
113        ] }
114
115    # 7. Receptionist located in Apartments
116    [npc_receptionist]
117        _prefix = "apartments.receptionist."
118        _start = "first_encounter"
119        first_encounter = { description = "first_encounter.dialog", options = [
120            { description = "first_encounter.option_1", to = "protestors" },
121            { description = "first_encounter.option_q", to = "repeated", mustExit = true }
122        ] }
123        protestors = { description = "protestors.dialog", options = [
```

```toml
124            { description = "protestors.option_1", to = "protestors2" },
125            { description = "protestors.option_q", to = "repeated" }
126        ] }
127        protestors2 = { description = "protestors2.dialog", options = [
128            { description = "protestors2.option_q", to = "repeated" }
129        ] }
130        repeated = { description = "repeated.dialog", options = [
131            { description = "repeated.option_1", to = "protestors" },
132            { description = "repeated.option_q", to = "repeated", mustExit = true }
133        ] }
134
135    # 8. TV located in Apartments
136    [home_tv]
137        _prefix = "home.tv."
138        _start = "first_on"
139        first_on = { description = "first_on.dialog", options = [
140            { description = "keep_watching", to = "dialog_a" },
141            { description = "off", to = "first_on", mustExit = true }
142        ] }
143        dialog_a = { description = "first_on.dialog_a", options = [
144            { description = "keep_watching", to = "dialog_b" },
145            { description = "off", to = "first_on", mustExit = true }
146        ] }
147        dialog_b = { description = "first_on.dialog_b", options = [
148            { description = "keep_watching", to = "dialog_c" },
149            { description = "off", to = "first_on", mustExit = true }
150        ] }
151        dialog_c = { description = "first_on.dialog_c", options = [
152            { description = "off", to = "first_on", mustExit = true }
153        ] }
154
155    # 9. Communicator device with Marie
156    [comms_marie]
157        _prefix = "marie.comms."
158        _start = "orientation"
159
160        orientation = { description = "orientation.dialog", options = [
161            { description = "orientation.option_1", to = "directions" },
162            { description = "orientation.option_2", to = "hear" },
163            { description = "orientation.option_q", to = "orientation", mustExit = true }
164        ] }
```

```toml
165      directions = { description = "directions.dialog", options = [
166          { description = "directions.option_q", to = "orientation" }
167      ] }
168      hear = { description = "hear.dialog", options = [
169          { description = "hear.option_q", to = "orientation" }
170      ] }
171
172      complaint = { description = "complaint.dialog", options = [
173          { description = "complaint.option_q", to = "complaint", mustExit = true }
174      ] }
175
176      in_position = { description = "in_position.dialog", options = [
177          { description = "in_position.option_1", to = "distraction" }
178      ] }
179      distraction = { description = "distraction.dialog", options = [
180          #{ description = "in_position.option_1", to = "coast_is_clear" }
181      ] }
182      coast_is_clear = { description = "coast_is_clear.dialog", options = [
183          { description = "coast_is_clear.option_q", to = "gogogo", mustExit = true }
184      ] }
185      gogogo = { description = "gogogo.dialog", options = [
186          { description = "gogogo.option_q", to = "gogogo", mustExit = true }
187      ] }
188
189      office = { description = "office.dialog", options = [
190          { description = "office.option_1", to = "documents" }
191      ] }
192      documents = { description = "office.dialog", options = [
193          { description = "office.option_q", to = "office", mustExit = true }
194      ] }
195
196      home = { description = "home.dialog", options = [
197          { description = "home.option_q", to = "home", mustExit = true }
198      ] }
199
200  # 10. Laptop
201  [entity_laptop]
202      _prefix = "entities.laptop."
203      _start = "boot"
204
205      boot = { description = "boot.dialog", options = [
```

```
206            { description = "boot.option_1", to = "home" }
207        ] }
208        home = { description = "home.dialog", options = [
209            { description = "home.option_q", to = "boot", mustExit = true },
210            { description = "home.option_1", to = "pictures" },
211            { description = "home.option_2", to = "funny_cat_videos" },
212            { description = "home.option_3", to = "document" }
213        ] }
214        pictures = { description = "pictures.dialog", options = [
215            { description = "pictures.option_q", to = "home" }
216        ] }
217        document = { description = "document.dialog", options = [
218            { description = "document.option_q", to = "home" }
219        ] }
```

```toml
1    # Path to emoji root directory
2    emojis = "/emojis"
3
4    # Path to font file
5    # https://fonts.google.com/specimen/VT323?category=Monospace#standard-styles
6    font = "/VT323-Regular.ttf"
```