World of Deez

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:

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

  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."*
- *"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 Dialogue and Localisation

"Give NPCs interactive dynamic dialogue."

4.3 World Event System

"Add a system for managing world events."

4.4 Terminal Emulator

"Implement a terminal emulator."

I implemented this by creating a new class `TerminalEmulator` which implements the `IOSystem` so that it could be easily slotted into the game.

4.4.1 Ansi Escape Codes

4.4.2 Emoji Support

4.4.3 EventDraw and the Map

5 Code Quality

6 Walkthrough

7 Known Issues

- find issue

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

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

[5] Excalidraw. https://excalidraw.com/

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

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

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

[9] YouTube. The best (and worst) show you haven't seen https://youtu.be/PFx2QM0Z8Qo

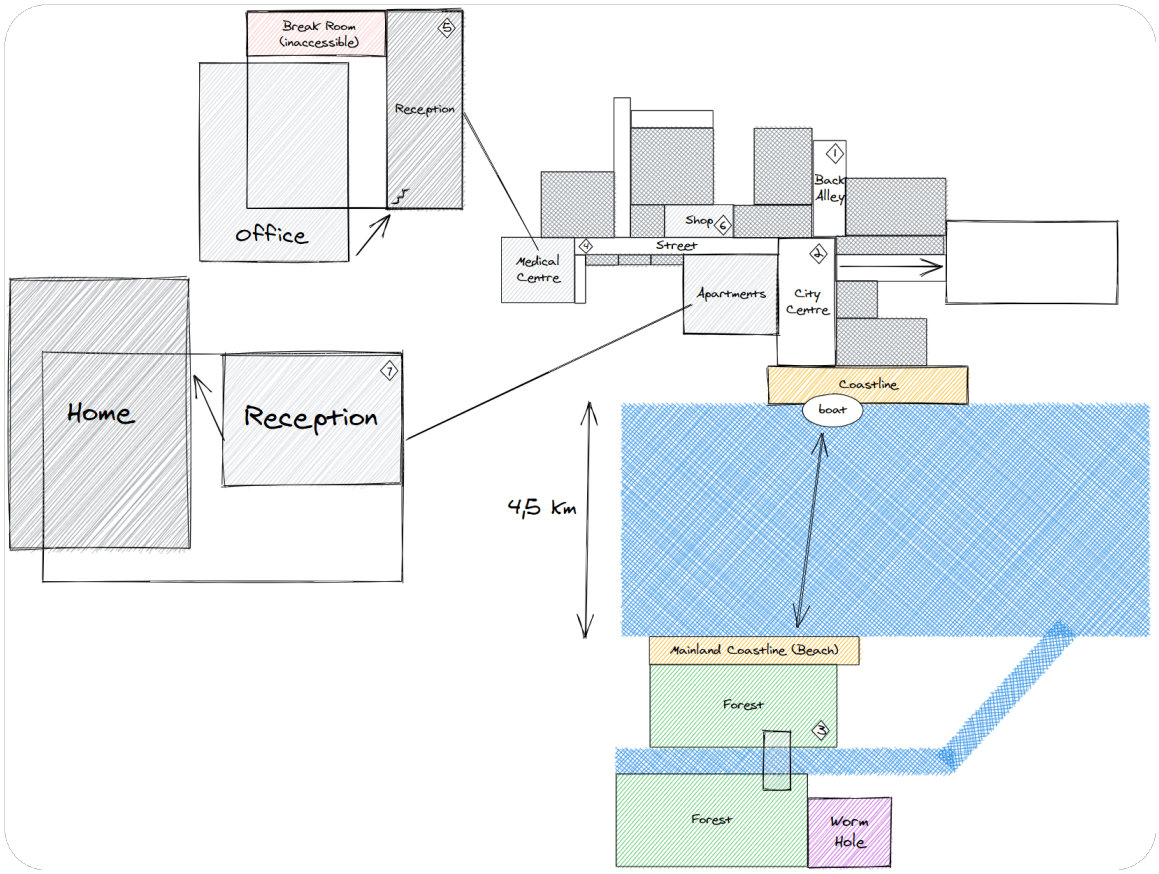

Figure 1: World Map

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]

```
> help
You can run the following commands:
- back: go back to the previous room
- inventory [of <something>]: look inside your bag or at something's inventory
- drop <item>: drop an item from your bag
- give <something> to <someone>: give something to someone
- go <direction>: go in a certain direction
- pet <something>: pet something around you or in your inventory
- quit: quit the game
- take <something> [from <someone>]: put something in your bag
- talk with <someone>: start talking with someone
- use <something>: use something around you or in your inventory
- where am i: describe the current room again
- map: show the world map
> _
```

Figure 4: Output from the help command

```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
     */
    public String getUsage() {
        return this.usage;
    }

    /**
     * Get all applicable patterns to match to execute this command.
     * @return Regex Pattern array
     */
    public Pattern[] getPatterns() {
        return this.patterns;
    }

    /**
     * Check whether this command is visible to the player in the help menu.
     * @return True if visible
     */
    public boolean isVisible() {
        return true;
    }

    /**
     * Run this command within the scope of a world and with any parsed arguments.
     * @param world Current World object
     * @param args Arguments passed into command
     * @return Boolean indicating whether the game loop should exit.
     */
    public abstract boolean run(World world, Arguments args);

    /**
     * Filter entities by those that are in the current room.
     */
    public static final int FILTER_ROOM = 1;

    /**
     * Filter entities by those that are in the player's inventory.
     */
    public static final int FILTER_INVENTORY = 2;

    /**
     * Don't filter entities and instead search through both the current room and the player's inventory.
```

```
 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;
```

```
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
    public void registerCommand(Command command) {
        this.commands.add(command);
    }

    /**
     * Register multiple commands.
     * @param commands Command array
     */
    public void registerCommands(Command[] commands) {
        for (Command command : commands) {
            this.registerCommand(command);
        }
    }

    /**
     * Get Commands provided by this Command manager.
     * @return List of Commands
     */
    public List<Command> getCommands() {
        return this.commands;
    }

    /**
     * Initialise all the commands a player can execute.
     */
    private void initialiseCommands() {
        final Command[] DEFAULT_COMMANDS = {
            new CommandBack(),
            new CommandBag(),
            new CommandDrop(),
            new CommandGive(),
            new CommandGo(),
            new CommandHelp(this),
            new CommandPet(),
            new CommandQuit(),
            new CommandTake(),
            new CommandTalk(),
            new CommandUse(),
            new CommandWhereAmI(),
        };

```

```
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();
```

```
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.
```

```
42          if (target instanceof IGiveable) {
43              ((IGiveable) target).give(item);
44          } else {
45              io.println("<commands.give.denied.1> " + item.getHighlightedName()
46                  + " <commands.give.denied.2> " + target.getHighlightedName() + "!");
47          }
48
49          return false;
50      }
51  }
```

```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"))
        );
```

```
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 + "!");
```

```
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
    /**
     * Construct a new Campaign World
     * @param io Provided IO system
     */
    public CampaignWorld(IOSystem io) {
        super(io);

        this.visitedRooms = new HashSet<>();
        this.dialogueLoader = new DialogueLoader();
        this.flags = new StoryFlags(this.getEventSystem());

        try {
            this.dialogueLoader.load("/dialogue.toml");
        } catch (IOException e) {
            System.err.println("Failed to load resources for campaign world!");
            e.printStackTrace();
        }

        this.buildWorld();
        this.spawnEntities();
        this.registerEvents();
    }

    /**
     * Get this World's Dialogue Loader
     * @return Dialogue Loader
     */
    public DialogueLoader getDialogueLoader() {
        return this.dialogueLoader;
    }

    /**
     * Get the global story flags.
     * @return Story flags instance
     */
    public StoryFlags getStoryFlags() {
        return this.flags;
    }

    /**
     * Check whether the Player has visited a certain Room yet
```

```
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
    /**
     * Spawn and setup any Entities within this World.
     */
    private void spawnEntities() {
        for (Room room : this.rooms.values()) {
            room.spawnEntities();
        }

        // Entangle boat inventories.
        Entity boat1 = this.entities.get("boat1");
        Entity boat2 = this.entities.get("boat2");
        boat1.entangleInventory(boat2.getInventory());
    }

    /**
     * Register all the game logic.
     */
    private void registerEvents() {
        // Capture all Events for Entities entering Rooms.
        this.eventSystem.addListener(EventEntityEnteredRoom.class,
            (EventEntityEnteredRoom event) -> {
                Entity entity = event.getEntity();
                if (entity instanceof EntityPlayer) {
                    Room room = entity.getRoom();

                    // Whenever the Player enters a Room, we should print the
                    // description of the Room and list things found in the Room.
                    this.io.println(
                        room.describe()
                            + "\n<global.can_go_in_x_directions.1> "
                            + room.getDirections().size()
                            + " <global.can_go_in_x_directions.2>: "
                            + room.getDirections()
                                .stream()
                                .map(x ->
                                    x.toString()
                                     .toLowerCase()
                                     .replaceAll("_", " ")
                                )
                                .collect(Collectors.joining(", "))
                    );
```

```
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>");
```

```
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;
```

```
42          IOSystem io = world.getIO();
43          io.print("<commands.map.discovered.1> " + campaignWorld.percentVisited()
44              + "% <commands.map.discovered.2>!" + "\n".repeat(24) + "<commands.map.close>");
45
46          // We make the map visible and block on user input,
47          // Once the user interacts, the map is hidden again.
48          this.visible = true;
49          io.readLine();
50          this.visible = false;
51          return false;
52      }
53
54      @Override
55      public void onEvent(EventDraw event) {
56          if (!this.visible) return;
57
58          // We are drawing the map from [0,1] to [80,24].
59          float fw = event.getCharWidth();
60          float fh = event.getCharHeight();
61
62          var g = event.getGraphics();
63          g.drawImage(
64              this.image,
65              Math.round(event.getOriginX()),
66              Math.round(event.getOriginY() + fh),
67              Math.round(fw * 80),
68              Math.round(fh * 23),
69              null
70          );
71      }
72  }
```

```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
    public void use(Entity target) {
        CampaignWorld world = (CampaignWorld) this.getWorld();
        IOSystem io = world.getIO();
        Inventory inventory = target.getInventory();

        // Check if the player has the key to this boat.
        boolean hasKey = false;
        for (Entity item : inventory.getItems()) {
            if (item instanceof EntityBoatKey) {
                hasKey = true;
            }
        }

        if (!hasKey) {
            if (world.getStoryFlags().getStage() == Stage.Exposition) {
                io.println("<entities.boat.locked>");
            } else {
                io.println("<entities.boat.locked_for_sale>");
            }

            return;
        }

        // Check whether the player is carrying too much.
        if (inventory.getWeight() > 1) {
            io.println("<entities.boat.denied>");
            return;
        }

        // If we're good to go, travel to the other side.
        io.println("<entities.boat.travel>\n");
        target.setLocation(this.destination);
    }

    @Override
    public void give(Entity item) {
        var io = this.getWorld().getIO();
        if (item.setLocation(this.getInventory())) {
            io.println("<entities.boat.give.1> " + item.getHighlightedName() + " <entities.boat.give.2>.");
        } else {
            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) {
```

```
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
            }

            this.dialogue.run(io);
        } else {
            io.println("<entities.comms.off>");
        }
    }

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

        // Make the guards disappear at this point.
        dialogue.getPart("distraction")
            .addOption(new DialogueOption<>(
                "<marie.comms.distraction.option_1>",
                io -> {
                    ((RoomMedicalCentreReception) this.world.getRoom("Medical Centre: Reception"))
                        .getGuard()
                        .consume(false);

                    return "coast_is_clear";
                }
            ));
    }

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

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

```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;
    }

```

```
42      /**
43       * Whether the player is sitting on the couch.
44       * @return True if the player is sat down
45       */
46      public boolean isSitting() {
47          return this.isSitting;
48      }
49  }
```

```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
            new DialogueNode<String>("<entities.laptop.cat_videos.dialog>")
                .addOption(new DialogueOption<String>("<entities.laptop.cat_videos.option_q>",
                    io -> {
                        try {
                            Desktop.getDesktop().browse(new URI("https://youtu.be/k35aiQPgNI4"));
                        } catch (Exception e) { /* If we fail, just ignore this ever happen. */ }

                        return "home";
                    })));

        // Story handler for sending documents to Marie.
        this.dialogue.getPart("document")
            .addOption(new DialogueOption<String>("<entities.laptop.document.option_1>",
                io -> {
                    var w = (CampaignWorld) this.getWorld();
                    var flags = w.getStoryFlags();
                    if (flags.getStage() != Stage.Stealth) {
                        io.println("<marie.comms.no_access>");
                        return "document";
                    }

                    Inventory inv = w.getPlayer().getInventory();
                    int validPieces = 0;
                    for (Entity item : inv.getItems()) {
                        if (item instanceof EntityDocument) {
                            if (((EntityDocument) item).getIsValid()) {
                                validPieces++;
                            }
                        }
                    }

                    if (validPieces == 3) {
                        io.println("<marie.comms.received>");
                        flags.setStage(Stage.End);
                        return null;
                    }

                    io.println("<marie.comms.bad_documents>");
                    return "document";
                }));
    }
```

```
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";
```

```
                }));
    }

    @Override
    public void talk() {
        if (this.dialogue.getCurrentNode().equals("waiting")) {
            var w = ((CampaignWorld) this.getWorld());
            Inventory inv = w.getPlayer().getInventory();
            for (Entity item : inv.getItems()) {
                if (item instanceof EntityComms) {
                    this.dialogue.setNodeIfPresent("mission_brief");
                    w.getStoryFlags().setStage(Stage.Stealth);
                }
            }
        }

        super.talk();
    }
}
```

```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<>();
```

```
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
    /**
     * Index dialogue node, displays things that the player can buy.
     * Implicitly takes the context of the EntityShopkeeper.
     */
    private class IndexNode extends DialogueNode<String> {
        /**
         * Construct a new IndexNode.
         * We may ignore the description since we override {@code #getDescription}.
         */
        public IndexNode() {
            super(null);
        }

        @Override
        public String getDescription() {
            var w = (CampaignWorld) world;
            return "<shop.npc.greeting."
                + w.getStoryFlags().getStage().toString()
                + ">\n"
                + "<shop.npc.currently_have_amount_of_money> ¥ "
                + w.getStoryFlags().getBalance()
                + "\n";
        }

        @Override
        protected List<DialogueOption<String>> getOptions() {
            ArrayList<DialogueOption<String>> options = new ArrayList<>();

            var w = (CampaignWorld) world;
            var flags = w.getStoryFlags();
            var player = w.getPlayer();

            // Get all the items we can access at this story stage.
            Entity[] list = items.get(flags.getStage());
            for (Entity item : list) {
                int count = stock.get(item);
                int cost = price.get(item);
                IEntityFactory factory = entityFactory.get(item);

                // Add the option for this item.
                options.add(new DialogueOption<String>(
```

```
                item.describe()
                    + " ["
                    + Ansi.Yellow
                    + item.getWeight()
                    + " kg"
                    + Ansi.Reset
                    + "] (¥ "
                    + Ansi.Green
                    + cost
                    + Ansi.Reset
                    + ") - "
                    + (count == 0 ?
                          "<shop.npc.out_of_stock>!"
                        : count + " <shop.npc.x_left>"),
                io -> {
                    // Check that this item is in stock.
                    if (count == 0) {
                        io.println("\n\n<shop.npc.item_out_of_stock.1> "
                            + Ansi.Red
                            + "<shop.npc.out_of_stock>"
                            + Ansi.Reset
                            + ", <shop.npc.item_out_of_stock.2>!");
                    } else {
                        // Make sure the player can hold this item without
                        // going over their inventory weight limit.
                        Inventory inv = player.getInventory();
                        if (inv.getWeight() + item.getWeight() > inv.getMaxWeight()) {
                            io.println("\n\n<shop.npc.too_heavy>");
                        } else {
                            // Try to deduct money from the player's money.
                            if (flags.deductFromBalance(cost)) {
                                io.println("\n\n<shop.npc.bought.1> "
                                    + item.getHighlightedName()
                                    + " <shop.npc.bought.2>!");

                                stock.put(item, count - 1);

                                Entity entity = factory.produce();
                                entity.setLocation(inv);
                            } else {
                                io.println("\n\n<shop.npc.not_enough> "
```

```
                                        + item.getHighlightedName() + "!");
                                }
                            }
                        }

                        return "index";
                    }
                ));
            }

            options.add(new DialogueOption<String>("<shop.npc.leave>", "index").mustExit());
            return options;
        }
    }

    @Override
    public void setupDialogue(Dialogue<String> dialogue) {
        dialogue.addPart("index", new IndexNode());
        dialogue.setNodeIfPresent("index");
    }
}
```

```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
     * @param dialogue Entity Dialogue
     */
    public abstract void setupDialogue(Dialogue<T> dialogue);

    /**
     * Configure dialogue.
     */
    public void setupDialogue() {
        this.setupDialogue(this.dialogue);
    }

    /**
     * Use the CampaignWorld's DialogueLoader to populate this Entity's Dialogue
     * @param dialogue Entity Dialogue
     * @param id Target dialogue ID in file
     */
    public void setupDialogueFromId(Dialogue<String> dialogue, String id) {
        var world = (CampaignWorld) this.getWorld();
        world.getDialogueLoader().populate(dialogue, id);
    }

    /**
     * Set the current dialogue node if the given node is present.
     * @param node Target node
     */
    public void setDialogueNodeIfPresent(Object node) {
        try {
            @SuppressWarnings("unchecked")
            T n = (T) node;

            this.dialogue.setNodeIfPresent(n);
        } catch (ClassCastException ex) {
            // Ignore the error since if we can't cast it
            // to whatever type this is, then obviously this
            // node is not present within this Dialogue.
        }
    }
}
```

```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);
```

```
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
            }
        }

        return true;
    }

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

        this.guardEntity = new EntityNpc(
            world,
            this.toLocation(),
            "npc_security_guard",
            "<medical_centre.guard.description>",
            new String[] { "guard", "security" }
        );
        world.spawnEntity("npc_guard", this.guardEntity);

        this.couchEntity = new EntityCouch(world, this.toLocation());
        world.spawnEntity("couch", this.couchEntity);
        world.getEventSystem().addListener(EventEntityLeftRoom.class, this.couchEntity);
    }

    public EntityCouch getCouch() {
        return this.couchEntity;
    }

    public Entity getGuard() {
        return this.guardEntity;
    }
}
```

```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
        if (direction == Direction.WEST) {
            if (this.protestorsEntity.getRoom() == this) {
                this.getWorld()
                    .getIO()
                    .println("<street.protestors.blocking>");

                return false;
            }
        }

        return true;
    }

    @Override
    public void spawnEntities() {
        World world = this.getWorld();
        this.protestorsEntity = new EntityNpc(
            world,
            this.toLocation(),
            "npc_protestors",
            "<street.protestors.description>",
            new String[] { "protestors", "protestor" }
        );
        world.spawnEntity("npc_protestors", this.protestorsEntity);
    }

    @Override
    public void onEvent(EventGameStageChanged event) {
        // Remove the protestors if we are in Stealth chapter.
        if (event.getStage() == Stage.Stealth) {
            this.protestorsEntity.consume(false);
        } else {
            this.protestorsEntity.setLocation(this);
        }
    }
}
```

```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",
```

```
            "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
```

```
42          */
43         public StoryFlags(EventSystem eventSystem) {
44             this.eventSystem = eventSystem;
45             this.stage = Stage.Exposition;
46             this.quests = new HashSet<>();
47             this.balance = 100_000;
48             this.ticks = 0;
49 
50             this.eventSystem.addListener(EventTick.class, e -> this.ticks++);
51         }
52 
53         /**
54          * Get the current stage (chapter) of the story.
55          * @return Current stage
56          */
57         public Stage getStage() {
58             return this.stage;
59         }
60 
61         /**
62          * Set the current stage (chapter) of the story.
63          * @param stage New stage
64          */
65         public void setStage(Stage stage) {
66             this.stage = stage;
67             this.eventSystem.emit(new EventGameStageChanged(stage));
68         }
69 
70         /**
71          * Get the player's balance
72          * @return Player's balance
73          */
74         public int getBalance() {
75             return this.balance;
76         }
77 
78         /**
79          * Set player's new balance.
80          * @param balance New balance
81          */
82         public void setBalance(int balance) {
```

```
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     /**
```

```
124         * Get total number of side-quests.
125         * @return Total number of side-quests
126         */
127        public int getTotalQuests() {
128            return Quest.values().length;
129        }
130    }
```

```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
        this.currentNode = node;
    }

    /**
     * Change the current node to a different one if it exists
     * @param node New node
     */
    public void setNodeIfPresent(T node) {
        if (this.parts.containsKey(node)) {
            this.currentNode = node;
        }
    }

    /**
     * Add a new part to the dialogue
     * @param part What this node is identified by
     * @param node The new node
     */
    public void addPart(T part, DialogueNode<T> node) {
        this.parts.put(part, node);
    }

    /**
     * Get an existing part from the dialogue
     * @param part What the node is identified by
     * @return The node if it exists, otherwise null
     */
    public DialogueNode<T> getPart(T part) {
        return this.parts.get(part);
    }

    /**
     * Run the Dialogue engine until one of the options exits us out
     * @param io Provided IO system
     */
    public void run(IOSystem io) {
        var part = this.parts.get(this.currentNode);
        io.println("\n" + part.getDescription());
        DialogueOption<T> option = part.pickOption(io);

        T target = option.handle(io);
```

```
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
     * @param key Key to lookup
     */
    @SuppressWarnings("unchecked")
    public void populate(Dialogue<String> dialogue, String key) {
        Map<String, Object> nodes = (Map<String, Object>) this.data.get(key);

        // Process any special keys first before we continue.
        String prefix = "";
        for (String nodeKey : nodes.keySet()) {
            if (nodeKey.equals("_prefix")) {
                prefix = (String) nodes.get(nodeKey);
            } else if (nodeKey.equals("_start")) {
                dialogue.getCurrentNode((String) nodes.get(nodeKey));
            }
        }

        for (Entry<String, Object> node : nodes.entrySet()) {
            // Ignore any keys starting with _, as they are used above.
            String nodeKey = node.getKey();
            if (nodeKey.startsWith("_")) continue;

            // Read each node's values and find the description and options.
            Map<String, Object> values = (Map<String, Object>) node.getValue();

            // Description strings are assumed to be i18n paths.
            String description = "<" + prefix + (String) values.get("description") + ">";
            List<Map<String, Object>> options = (List<Map<String, Object>>) values.get("options");

            // Construct a new Dialogue Node with the given data.
            DialogueNode<String> dialogueNode = new DialogueNode<>(description);
            for (Map<String, Object> object : options) {
                String desc = "<" + prefix + (String) object.get("description") + ">";
                String to = (String) object.get("to");
                Boolean mustExit = (Boolean) object.get("mustExit");

                if (mustExit == null) {
                    dialogueNode.addOption(desc, to);
                } else {
                    dialogueNode.addOption(desc, to, mustExit);
                }
            }
```

```
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
    /**
     * Create a new option which branches off this node.
     * @param description Description of this option
     * @param stage The next stage of the dialogue this should jump to
     * @param mustExit Whether we must exit from the dialogue after selecting this option
     * @return This Dialogue Node so other method calls can be chained
     */
    public DialogueNode<T> addOption(String description, T stage, boolean mustExit) {
        var option = new DialogueOption<T>(description, stage);
        if (mustExit) option.mustExit();
        this.options.add(option);
        return this;
    }

    /**
     * Create a new option which branches off this node.
     * @param description Description of this option
     * @param stage The next stage of the dialogue this should jump to
     * @return This Dialogue Node so other method calls can be chained
     */
    public DialogueNode<T> addOption(String description, T stage) {
        return this.addOption(description, stage, false);
    }

    /**
     * Get the options available to this node.
     * @return List of options
     */
    protected List<DialogueOption<T>> getOptions() {
        return this.options;
    }

    /**
     * Ask the player to pick one of the valid options branching off this node.
     * @param io Provided IO system
     * @return The selected option
     */
    public DialogueOption<T> pickOption(IOSystem io) {
        List<DialogueOption<T>> options = this.getOptions();
        for (int i=0;i<options.size();i++) {
            io.println((i + 1) + ". " + options.get(i).getDescription());
```

```
83          }
84
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                  }
94
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
    }

    /**
     * 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;
    }
}
```

```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}.
```

```
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
     * Get the World this Entity resides in.
     * @return World
     */
    public World getWorld() {
        return this.world;
    }

    /**
     * Get the Room that this Entity is currently in.
     * @return Room
     */
    public Room getRoom() {
        return this.location.getRoom();
    }

    /**
     * Get the Inventory that this Entity is currently in.
     * @return Inventory
     */
    public Inventory getInventoryWithin() {
        return this.location.getInventory();
    }

    /**
     * Remove this Entity from any existing place.
     * Provides a consistent way to clean up the Entity before placing it anywhere.
     * @param suppressEvents Whether to suppress the "Entity Left Room" Event
     * @return Whether this Entity was removed from an Inventory
     */
    public boolean consume(boolean suppressEvents) {
        boolean consumed = false;
        Inventory inventory = this.location.getInventory();
        if (inventory != null) consumed = inventory.remove(this);

        Room previousRoom = this.getRoom();
        if (previousRoom != null && !suppressEvents) this.world.emit(new EventEntityLeftRoom(this, previousRoom));

        this.location.clear();
        return consumed;
    }

```

```java
    /**
     * Move the Entity into a Room.
     * @param room Destination Room
     */
    public void setLocation(Room room) {
        boolean consumed = this.consume(false);
        this.location.setLocation(room);
        if (!consumed) this.world.emit(new EventEntityEnteredRoom(this));
    }

    /**
     * Move the Entity into an Inventory.
     * @param inventory Destination Inventory
     * @return Whether we successfully moved the entity into the inventory.
     */
    public boolean setLocation(Inventory inventory) {
        if (inventory.add(this)) {
            this.consume(true);
            this.location.setLocation(inventory);
            return true;
        }

        return false;
    }

    /**
     * Link this Entity's inventory with an existing inventory.
     * @param inventory Target inventory
     */
    public void entangleInventory(Inventory inventory) {
        this.inventory = inventory;
    }

    /**
     * Get names that this Entity can be called by.
     * @return String array of names for this Entity
     */
    public abstract String[] getAliases();

    /**
     * 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() {
```

```
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 "";
```

```
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);
```

```
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() {
```

```
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
     * @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;
        }
    }
}
```

```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();
    }

    /**
```

```
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();
```

```
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
```

```
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);
```

```
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);
```

```
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
    /**
     * The matching mechanism has matched an emoji.
     */
    public static final int MATCH_FOUND = 2;

    /**
     * Match the next character.
     * @param c Character to match against
     * @return One of {@link #MATCH_NONE}, {@link #MATCH_SOME} or {@link #MATCH_FOUND}
     */
    public int match(char c) {
        var child = this.currentNode.getChild(c);
        if (child != null) {
            this.currentNode = child;
            if (child.getValue() == null) return MATCH_SOME;
            return MATCH_FOUND;
        }

        if (this.currentNode != this.emojiTree) {
            this.currentNode = this.emojiTree;
            child = this.emojiTree.getChild(c);
            if (child != null) {
                if (child.getValue() != null) return MATCH_SOME;
                return MATCH_FOUND;
            }
        }

        return MATCH_NONE;
    }

    /**
     * Load emoji definitions and resources from a given resource directory.
     * @param rootDir Root directory at which we expect a valid {@code definitions.toml} to exist
     * @throws IOException if the definition file is missing or defined emojis are invalid
     */
    public void loadResources(String rootDir) throws IOException {
        InputStream defnStream = this.getClass().getResourceAsStream(rootDir + "/definitions.toml");
        // We need to force UTF-8 encoding or else unicode emojis may get mangled.
        String defnString = IOUtils.toString(defnStream, StandardCharsets.UTF_8);
        Toml defn = new Toml().read(defnString);
        List<HashMap<String, Object>> emojis = defn.getList("emojis");
```

```
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;
    }

    /**
```

```
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
```

```
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;
```

```
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
    public void loadResources() {
        try {
            InputStream stream = this.getClass().getResourceAsStream("/emulator.toml");
            Toml defn = new Toml().read(stream);

            // If a font is defined, load it.
            String font = defn.getString("font");
            if (font != null) {
                this.loadFont(font, 32.0f / 12.8f);
            }

            // If an emoji root directory is defined, load it.
            String rootDir = defn.getString("emojis");
            if (rootDir != null) {
                this.emojiManager.loadResources(rootDir);
            }
        } catch (Exception e) {
            System.err.println("Failed to load any resources for terminal view!");
            e.printStackTrace();
        }
    }

    /**
     * Load a specific font with a known ratio.
     * We expect this font to be monoscape.
     * @param source Path to the font to be loaded
     * @param ratio Ratio of width to height for this font
     * @throws IOException if the font cannot be loaded from a given path
     * @throws FontFormatException if the font is of an incorrect format, we expect a TTF
     */
    public void loadFont(String source, float ratio) throws IOException, FontFormatException {
        InputStream stream = this.getClass().getResourceAsStream(source);
        this.fratio = ratio;
        this.font = Font.createFont(Font.TRUETYPE_FONT, stream);
    }

    /**
     * Derive the font measurements before we continue rendering.
     */
    public void deriveFont() {
        final int padding = 100;
```

```
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 blind 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
```

```
165     public Dimension getPreferredSize() {
166         return new Dimension(1280, 960);
167     }
168 
169     @Override
170     protected void paintComponent(Graphics g) {
171         // https://stackoverflow.com/a/17922749
172         super.paintComponent(g);
173 
174         // Setup our canvas for rendering.
175         g.setColor(Color.BLACK);
176         g.setFont(this.derivedFont);
177         g.fillRect(0, 0, this.getWidth(), this.getHeight());
178 
179         // Find our topleft-most (x,y) to start rendering from.
180         TextBuffer buffer = this.emulator.getBuffer();
181         float ox = (this.getWidth() - this.fw * buffer.getWidth()) / 2;
182         float oy = (this.getHeight() - this.fh * buffer.getHeight()) / 2;
183 
184         // Render each cell individually.
185         for (int y=0;y<buffer.getHeight();y++) {
186             int skipChars = 0;
187             for (int x=0;x<buffer.getWidth();x++) {
188                 // If needs be, skip chars in this line.
189                 if (skipChars > 0) {
190                     skipChars--;
191                     continue;
192                 }
193 
194                 // Get the character to render.
195                 char c = buffer.getChar(x, y);
196 
197                 // Match each char for emoji codepoints.
198                 // If we start to match an emoji, peek ahead.
199                 int emojiMatch = this.emojiManager.match(c);
200                 int offset = 0;
201                 while (emojiMatch == EmojiManager.MATCH_SOME) {
202                     emojiMatch = this.emojiManager.match(buffer.getChar(x + ++offset, y));
203                 }
204 
205                 // Get this cell's background and foreground colours.
```

```
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                 }
```

```
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.
```

```
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             }
```

```
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);
```

```
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;
```

```
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.
```

```
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;
```

```
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;
```

```
165                 }
166             }
167
168             this.write(c);
169         }
170     }
171
172     /**
173      * Get a character at a certain position
174      * @param x X position
175      * @param y Y position
176      * @return Character at given position
177      */
178     public char getChar(int x, int y) {
179         return this.buffer[y][x];
180     }
181
182     /**
183      * Get the background colour at a certain position
184      * @param x X position
185      * @param y Y position
186      * @return Background colour at given position
187      */
188     public Color getBg(int x, int y) {
189         return this.bufferBg[y][x];
190     }
191
192     /**
193      * Get the foreground colour at a certain position
194      * @param x X position
195      * @param y Y position
196      * @return Foreground colour at given position
197      */
198     public Color getFg(int x, int y) {
199         return this.bufferFg[y][x];
200     }
201
202     /**
203      * Get the width of this buffer.
204      * @return Width
205      */
```

```
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) {
```

```
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
     * developer provides an incorrect data structure, in that case the
     * error will be emitted from within this method. It is not a critical
     * error but it should be handled immediately.
     * @param path Path to value we want
     * @return The value at the given path
     */
    @SuppressWarnings("unchecked")
    public String from(String... path) {
        if (path.length == 0) return "<empty string>";

        try {
            var index = 1;
            var node = this.map.get(path[0]);
            while (index != path.length) {
                Map<String, Object> map = (Map<String, Object>) node;
                node = map.get(path[index++]);
            }

            if (node != null) {
                return (String) node;
            }
        } catch (Exception e) {
            // We don't want this to be a fatal error,
            // we instead return the original template.
        }

        return "<" + Arrays.asList(path).stream().collect(Collectors.joining(".")) + ">";
    }

    /**
     * Given a path, find the value at the end of the path.
     * @param path Path to value we want, keys separated by period
     * @return The value at the given path
     */
    public String get(String path) {
        return this.from(path.split("\\."));
    }
}
```

```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
     * @return The child represented by this key if it exists, otherwise returns null
     */
    public Tree<K, V> getChild(K key) {
        return this.children.get(key);
    }

    /**
     * Private method used to accumulate the edges travelled up to the root node.
     * @param acc Accumulator value
     * @return The current accumulator value
     */
    private int getHeight(int acc) {
        if (this.parent == null) return acc;
        return this.parent.getHeight(++acc);
    }

    /**
     * The height of the Tree from this point.
     * This is the number of edges to get from this node to the root node.
     * @return The height of the tree
     */
    public int getHeight() {
        return this.getHeight(0);
    }

    /**
     * Whether this Tree node has a value.
     * @return True if this node has a value
     */
    public boolean hasValue() {
        return this.value != null;
    }

    /**
     * Get the value of this Tree node.
     * @return Value stored if there is one, otherwise null.
     */
    public V getValue() {
        return this.value;
    }

```

```java
    /**
     * Add a child to this Tree node represented by a key K.
     * @param key Key to represent this new child
     * @param node Child node to add
     */
    public void addChild(K key, Tree<K, V> node) {
        this.children.put(key, node);
    }

    /**
     * Recurse through a given key path and add value as a node at the bottom of the path.
     * @param keys Keys to iterate through
     * @param value Value to add at the end of the path
     */
    public void addChildWithPath(List<K> keys, V value) {
        Tree<K, V> node = this;
        while (keys.size() > 0) {
            K key = keys.remove(0);
            Tree<K, V> child = node.getChild(key);
            if (child == null) {
                child = new Tree<>(node, keys.size() == 0 ? value : null);
                node.addChild(key, child);
            }

            node = child;
        }
    }
}
```

```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
     * Flip a given Direction in the opposite direction.
     * @return Direction in the opposite direction.
     */
    public Direction flip() {
        switch (this) {
            default:
            case NORTH: return Direction.SOUTH;
            case NORTH_EAST: return Direction.SOUTH_WEST;
            case EAST: return Direction.WEST;
            case SOUTH_EAST: return Direction.NORTH_WEST;
            case SOUTH: return Direction.NORTH;
            case SOUTH_WEST: return Direction.NORTH_EAST;
            case WEST: return Direction.EAST;
            case NORTH_WEST: return Direction.SOUTH_EAST;
            case UP: return Direction.DOWN;
            case DOWN: return Direction.UP;
        }
    }

    /**
     * Convert an arbitrary String to a Direction.
     * @param direction Raw string representing a Direction
     * @return Direction or null from given string
     */
    public static Direction fromString(String direction) {
        if (direction == null) return null;

        String directionFormatted = direction.toUpperCase();
        try {
            return Direction.valueOf(directionFormatted);
        } catch (Exception ex) {
            for (Direction dir : Direction.values()) {
                if (dir.matches(directionFormatted)) {
                    return dir;
                }
            }

            return null;
        }
    }
}
```

```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

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

    /**
     * Reset the Location and put us outside of the World.
     */
    public void clear() {
        this.room = null;
        this.inventory = null;
    }

    /**
     * Get the current Room this Location represents.
     * @return Room or null if in an inventory or out of the World.
     */
    public Room getRoom() {
        return this.room;
    }

    /**
     * Get the current Inventory this Location represents.
     * @return Inventory or null if in a room or out of this World.
     */
    public Inventory getInventory() {
        return this.inventory;
    }
}
```

```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;
    }

```

```
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
     * @return True if there is an exit in a given Direction
     */
    public boolean hasExit(Direction direction) {
        return this.adjacentRooms.containsKey(direction);
    }

    /**
     * Reset adjacent Rooms and reconfigure adjacent Rooms.
     * This should be called after all Rooms have been spawned into the World.
     */
    public void linkRooms() {
        this.adjacentRooms.clear();
        this.setupDirections();
    }

    /**
     * Spawn Entities in this World.
     * By default, nothing is done but this should be used further up to spawn
     * the Entities for this particular Room.
     */
    public void spawnEntities() {}

    /**
     * Convert this Room into a Location.
     * @return Location representation of Room
     */
    public Location toLocation() {
        return new Location(this);
    }

    /**
     * Describe what this Room looks like.
     * @return Description of this Room
     */
    public abstract String describe();

    /**
     * Setup adjacent Rooms.
     */
    protected abstract void setupDirections();
}
```

```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
```

```
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
     * @param room Room to add
     */
    protected void addRoom(Room room) {
        this.rooms.put(room.getName(), room);
    }

    /**
     * Spawn a new Entity in the World.
     * @param id Unique Entity ID
     * @param entity Entity to spawn
     */
    public void spawnEntity(String id, Entity entity) {
        this.entities.put(id, entity);
    }

    /**
     * Get all the Entities found in a given Room.
     * @param room Room to search for
     * @return List of Entities in the World in a given Room
     */
    public List<Entity> getEntitiesInRoom(Room room) {
        return this
            .entities
            .values()
            .stream()
            .filter(e -> e.getRoom() == room)
            .collect(Collectors.toList());
    }

    protected void linkRooms() {
        for (Room room : this.rooms.values()) {
            room.linkRooms();
        }
    }

    public void emit(Event event) {
        this.eventSystem.emit(event);
    }

    /**
     * Try to spawn the player in the first available room.
```

```
124         */
125        public void spawnPlayer() {
126            this.player.setLocation(this.rooms.values().iterator().next());
127        }
128    }
```

```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
# Translations for World of Deez

[global]
sight = "You can see:"

    [global.can_go_in_x_directions]
    1 = "You may go in"
    2 = "directions"

[selectors]
direction = "<direction>"
something = "<something>"
someone = "<someone>"
item = "<item>"

    [selectors.cant_find]
    1 = "You look around for"
    2 = "but can't find anything"

[commands]
unknown = "Not sure what you're trying to do."

back = "go back to the previous room"
quit = "quit the game"
where_am_i = "describe the current room again"

    [commands.bag]
    usage = "look inside your bag or at something's inventory"
    cant_find = "Can't find what you want to look at."
    empty = "Your bag is empty!"
    entity_empty = "doesn't appear to have anything"
    can_carry_kg = "You can carry"
    are_carrying_kg = "You are carrying"
    look_in_bag = "You look in your bag to see"
    entity_appears_to_have = "appears to have"

    [commands.drop]
    usage = "drop an item from your bag"
    nothing_specified = "What do you want to drop?"

        [commands.drop.dropped]
```

```toml
        1 = "You drop"
        2 = "out of your bag"

    [commands.give]
    usage = "give something to someone"
    nothing_specified = "What do you want to give?"
    no_target = "What / who are you putting this in?"
    denied_player = "You cannot give yourself to anyone or anything. \u1F438"

        [commands.give.denied]
        1 = "Cannot give"
        2 = "to"

    [commands.go]
    usage = "go in a certain direction"
    nothing_specified = "Where are you going?"

    [commands.help]
    usage = "show help menu"
    can_run = "You can run the following commands:"

    [commands.pet]
    usage = "pet something around you or in your inventory"
    nothing_specified = "What are you trying to pet?"
    denied = "You cannot pet"

    [commands.take]
    usage = "put something in your bag"
    nothing_specified = "From who?"
    entity_does_not_have_entity = "does not have"
    item_not_specified = "What do you want to take?"

        [commands.take.took]
        1 = "You take"
        2 = "from"
        3 = "and put it in your bag"

        [commands.take.denied]
        1 = "You cannot take"
        2 = "it's too heavy to put in your bad"

```

```
    [commands.talk]
    usage = "start talking with someone"
    nothing_specified = "What do you want to talk with?"
    denied = "You cannot talk with"

    [commands.use]
    usage = "use something around you or in your inventory"
    nothing_specified = "What do you want to use?"
    denied = "You cannot use"

    [commands.map]
    usage = "show the world map"
    close = "Press enter to close."

        [commands.map.discovered]
        1 = "You have discovered"
        2 = "of the world"

    [commands.win]
    usage = "win the game"
    conclusion = """...

Chapter 4.:
The files are released into the internet for anyone to read, people are quick to
analyse through every single tiny detail, some immediate details come to light:
- Slyvasta was researching ethically questionable areas of science, in
  particular, they were running hundreds of tests daily on a variety of beastman
  test subjects. But nobody could prove anyone was there against their will.
- However, the research did also get immediately picked up by foreign powers who
  quickly discovered that the beastman society living in the city is a far
  greater threat than they initially anticipated.
- The city, with Sylvasta's public advice, immediately ordered evacuation of all
  citizens to any area they could find fearing a potentially deadly conflict on
  the horizon. The city quickly came under fire over the coming days.
"""
    stats = "\u001B[47m\u001B[30mYour final game stats\u001B[0m"
    total_ticks = "Total game ticks: "
    sidequests_complete = "Side-quests completed: "
    press_enter_key = "Press enter to close the game."

[entities]
```

```
boat_key = "A key to the speed boat"

    [entities.bed]
    description = "Bed"
    use = "You take a nap."

    [entities.boat]
    description = "Speedboat docked on the coast"
    locked = "The boat is locked."
    locked_for_sale = "The boat is locked.\nThere is a note which says to contact the shopkeeper to buy this boat."
    denied = "You must not be carrying anything to use the boat.\nYou can however put things in the boat."
    travel = "You hop in the boat and travel to the other side..."
    too_heavy = "The boat is carrying too much stuff already!"

        [entities.boat.give]
        1 = "Put"
        2 = "in the boat"

    [entities.cat]
    description = "A stray black cat"
    pet = "You pet the cat."
    use = "You cannot the cat.\nPlease do not the cat. \u1F633\u1F633\u1F633"
    enter = "A cat has wandered in."
    leave = "You see a cat leave."

    [entities.comms]
    description = "Communicator device"
    off = "The device is off."

    [entities.couch]
    description = "A brown leather couch"
    sitting = "You are already sitting in the couch."
    sit = "You sit down on the couch."

    [entities.laptop]
    description = "Laptop"

        [entities.laptop.boot]
        dialog = "You turn the computer on..."
        option_1 = "[wait]"

```

```
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:
174
175 \u1F99D
176
177
178
179
180
181
182
183 """
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
```

```
Curious, you peer out the window to see what's going on.. there's a group of
protestors outside the Medical Centre down the street, you can't really make out
what they're saying or what their signs say.

Though it'd not be surprising if something strange is going on in there, but you
can't really put your finger on it. Maybe there's something on the news..
'''

enter = '''
You enter your apartment.
'''

    [home.tv]
    description = "LG 55NANO966PA 55\" Super UHD 8K HDR Smart LED TV"
    off = "Turn the TV off."
    keep_watching = "Keep watching..."

        # Red: \u001B[31m
        # Green: \u001B[32m
        # Yellow: \u001B[33m
        # Cyan: \u001B[36m
        [home.tv.first_on]
        dialog = "You turn the TV on.\n\nThe news channel comes up..."
        dialog_a="""\u001B[31mNews Anchor\u001B[0m: Civil unrest is rising, Sylvasta is facing criticism from all
sides, and many people are uneasy about their future as rising tension between
human and beastman societies is causing escalated conflict around the city
borders.

\u001B[31mNews Anchor\u001B[0m: We bring you now to scenes outside of the Medical Centre where a
group of protestors have shown up in opposition to the research being led at
Slyvasta.

\u001B[36mCorrespondent\u001B[0m: I am here, standing outside with the group of protestors..

\u001B[36mCorrespondent\u001B[0m to \u001B[33mProtestor\u001B[0m: What brings you here today?
"""

        dialog_b="""\u001B[33mProtestor\u001B[0m:
Protestor: They're taking the city's money and using it for their own gain, they
shouldn't receive any funding let alone be allowed to operate here.

```

```
\u001B[31mNews Anchor\u001B[0m: Bold claims coming straight from outside Sylvasta, whether these
claims are grounded in anything is yet to be discovered, we've seen months and
months of leaks come out from former employees and internal mishaps but are yet
to truly find out the intention behind the people at Sylvasta.

\u001B[31mNews Anchor\u001B[0m: Sylvasta has personally announced that they refuse to communicate
or elaborate any further on their internal research citing public safety,
whatever that means nobody outside of their internal staff knows.
"""

dialog_c="""\u001B[31mNews Anchor\u001B[0m: It also begs the question whether the protests are unfounded and
just there to stir up trouble in the city. Earlier today we also spoke to local
residents living in the centre of the city..

\u001B[32mShopkeeper\u001B[0m: I think these guys just want to cause trouble, Sylvasta was vital in
establishing this city and letting us live in peace without having to worry
about being attacked, I just don't think there's enough reason to protest.

\u001B[32mShopkeeper\u001B[0m: We're already seeing the world turn against the city and these sorts
of internal conflicts will just give them reason to step in and take control.

\u001B[31mNews Anchor\u001B[0m: That concludes our broadcast for this morning, and we'll be back
for the 1pm News Hour.
"""

[apartments]
enter = '''
You enter the apartment complex reception.
'''

    # Cyan: \u001B[36m
    [apartments.receptionist]
    description = "The receptionist sitting behind a desk"

        [apartments.receptionist.first_encounter]
        dialog = "\u001B[36mReceptionist\u001B[0m: Good morning, how are you doing today?"
        option_1 = "What is that racket outside?"
        option_q = "Nevermind."

        [apartments.receptionist.protestors]
        dialog = """\u001B[36mReceptionist\u001B[0m: I don't know much but it looks like a group of people shouting
```

```
outside the Medical Centre..."""
        option_1 = "Do you know anything more?"
        option_q = "Alright."

        [apartments.receptionist.protestors2]
        dialog = """\u001B[36mReceptionist\u001B[0m: They were handing out flyers as they came up, maybe someone
nearby would know..."""
        option_q = "Thanks."

        [apartments.receptionist.repeated]
        dialog = "\u001B[36mReceptionist\u001B[0m: Hello, what can I help you with?"
        option_1 = "What's going on with those protestors outside?"
        option_q = "That's all, thanks."

[city_centre]
first_load = '''
You enter the city square, it's quite busy in the mornings. There's a lot of
people running around, unlikely you could stop most of them for a chat.

There's a general unease in the area, some people look quite tense and others
look like they're spending their last day on Earth...
'''

enter = '''
You enter the city square.
'''

    # Yellow: \u001B[33m
    [city_centre.npc]
    description = "A person sitting at a bench"

        [city_centre.npc.small_talk]
        dialog = "\u001B[33mStranger\u001B[0m: Hello, could I help you?"
        option_1 = "You don't happen to have seen the protestors go by here?"
        option_2 = "What are you reading?"
        option_q = "Nevermind."

        [city_centre.npc.protestors]
        dialog = """\u001B[33mStranger\u001B[0m: Oh yes, they were quite loud..
They were rather annoying, disturbed my morning.
...
```

```
They did leave me this leaflet though"""
        option_1 = "Can I see it?"

        [city_centre.npc.enquire]
        dialog = "\u001B[33mStranger\u001B[0m: These guys went past and left me a leaflet."

        [city_centre.npc.leaflet]
        # dialog = "\u001B[33mStranger\u001B[0m: Sure, here you go, keep it.\n[\u001B[33mStranger\u001B[0m gave you an item.]"
        dialog = """\u001B[33mStranger\u001B[0m: Sure, I've got it here...

You look at the leaflet:

\u001B[41mUNETHICAL RESEARCH
\u001B[40m\u001B[31mOur city is in danger, Slyvasta further
funding into their research yet do nothing
to help the city survive.

Sylvasta is abusing their position to
continue work into completely ILLEGAL
bioengineering of beastmen.\u001B[0m

Support us: \u001B[36mhttps://sylvasta.vercel.app\u001B[0m
"""
        option_1 = "Thanks."

[street]
first_load = '''
You enter the main city street connecting the major parts of the city.

There's a lot of chaos and noise here, to your west you can see protestors
outside of the medical centre complex, they're holding signs up, "UNETHICAL
RESEARCH", "OUR RELATIONS AT STAKE", "UNFORSEEN CONSEQUENCES".
'''

enter = '''
You enter the city street.
'''

    # Red: \u001B[31m
    [street.protestors]
```

```
    description = "A group of protestors holding signs"
    blocking = "There is a group of protestors blocking the way in."

        [street.protestors.small_talk]
        dialog = "\u001B[31mProtestor\u001B[0m: Have you heard about what Sylvasta is doing and why we're out here?"
        option_1 = "No, enlighten me."
        option_2 = "Nevermind."

        [street.protestors.enquire]
        dialog = """\u001B[31mProtestor\u001B[0m: They are using the city's money to further their frankly illegal
research, you've seen the leaks right?"""
        option_1 = "No."
        option_2 = "Right.. I'll leave you to it."

        [street.protestors.leaks]
        dialog = """\u001B[31mProtestor\u001B[0m: Well some former employees leaked information about their current
research projects and from what we can tell is that they're doing bioengineering
on beastmen, this shouldn't be tolerated let alone funded by the city."""
        option_1 = "Ok."

        [street.protestors.confrontation]
        dialog = "\u001B[31mProtestor\u001B[0m: You taking me for some sort of conspiracy theorist?"
        option_1 = "Yes."
        option_2 = "I'm leaving now."

[shop]
enter = '''
You enter the shop, the bell rings as you close the door behind you.
'''

    # Green: \u001B[32m
    [shop.npc]
    description = "The shop keeper"
    leave = "Nevermind."

    currently_have_amount_of_money = "You currently have:"
    not_enough = "You don't have enough money to buy"
    too_heavy = "Too heavy for you to carry!"

    out_of_stock = "out of stock"
    x_left = "left"
```

```
410
411         [shop.npc.fake_item]
412         cat = "A singular cat"
413
414         [shop.npc.item_out_of_stock]
415         1 = "This item is"
416         2 = "you may not buy it"
417
418         [shop.npc.bought]
419         1 = "You buy"
420         2 = "and put it in your bag"
421
422         [shop.npc.greeting]
423         Exposition = "\u001B[32mShopkeeper\u001B[0m: Good day, what would you like to buy?"
424         Recon = "\u001B[32mShopkeeper\u001B[0m: Hello, anything in mind today?"
425         Stealth = "\u001B[32mShopkeeper\u001B[0m: Looking to buy something?"
426         End = "\u001B[32mShopkeeper\u001B[0m: Need anything new?"
427
428 [back_alley]
429 first_load = '''
430 You enter a dark alley, it is quite quiet here, a stark contrast to the city
431 centre yet located nearly right in the middle. The sides of the alley are lined
432 with derelict buildings and dim lighting.
433
434 You make out a familiar figure further ahead.
435 '''
436
437 enter = '''
438 You enter the back alley.
439 '''
440
441 [coastline]
442 enter = '''
443 You arrive at the city's coastline, the water is rather calm and still.
444 '''
445
446 [mainland_coastline]
447 enter = '''
448 You arrive at the coastline on the mainland,
449 the sea is crashing against rocks lined against it,
450 it is significantly colder out here away from the concrete jungle.
```

```
'''

[forest]
enter = '''
You wander through the forest, you hear birds whistling at the peaks of trees...
There's a river flowing through the middle with an old wooden bridge above it.
It would be quite easy to lose yourself here for a few hours.
'''

    # Yellow: \u001B[33m
    [forest.old_man]
    description = "An old man wandering through the forest"
    full = "There's nothing else you can give them."
    accept = "\u001B[33mOld Man\u001B[0m: Thank you for bringing him back to me."
    deny = "\u001B[33mOld Man\u001B[0m: I don't want your"

        [forest.old_man.small_talk]
        dialog = "\u001B[33mOld Man\u001B[0m: You haven't seen my cat anywhere have you?"
        option_1 = "I've seen this black stray around town..."
        option_q = "No."

        [forest.old_man.request]
        dialog = "\u001B[33mOld Man\u001B[0m: Could you bring him back to me?"
        option_1 = "Sure."
        option_q = "Sorry, not right now."

        [forest.old_man.praise]
        dialog = "\u001B[33mOld Man\u001B[0m: Thank you for helping me!"
        option_q = "[leave]"

[worm_hole]
enter = '''
You step into the worm hole...
'''

[medical_centre]
enter = '''
You're now at the Medical Centre's reception.
'''

    # Red: \u001B[31m
```

```toml
    [medical_centre.guard]
    description = "Security guard stationed at the door"
    blocking = "There is security watching the stairs, there's no way to get past them."

        [medical_centre.guard.small_talk]
        dialog = "\u001B[31mGuard\u001B[0m: What do you want?"
        option_q = "Nevermind."

[medical_centre_office]
enter = '''
You find yourself at the Medical Centre's office.
You definitely shouldn't be here...
'''

    [medical_centre_office.books]
        [medical_centre_office.books.1]
        title = "sus"
        contents = "gdfghdfs"

        [medical_centre_office.books.2] # yes
        title = "sus"
        contents = "gdfghdfs"

        [medical_centre_office.books.3]
        title = "sus"
        contents = "gdfghdfs"

        [medical_centre_office.books.4] # yes
        title = "sus"
        contents = "gdfghdfs"

        [medical_centre_office.books.5] # yes
        title = "sus"
        contents = "gdfghdfs"

        [medical_centre_office.books.6]
        title = "sus"
        contents = "gdfghdfs"

[marie]
description = "\u001B[35mMarie the Mink\u001B[0m"
```

```
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
```

```
(you see the guard go into the breakroom)
"""
        option_1 = "[wait]"

        [marie.comms.coast_is_clear]
        dialog = "\u001B[35mMarie\u001B[0m: Ok, now is your chance."
        option_q = "Got it."

        [marie.comms.gogogo]
        dialog = "\u001B[35mMarie\u001B[0m: What are you waiting for??"
        option_q = "AAAAAAA"

        [marie.comms.office]
        dialog = "\u001B[35mMarie\u001B[0m: What do you see?"
        option_1 = "A bunch of documents scattered around."

        [marie.comms.documents]
        dialog = """\u001B[35mMarie\u001B[0m: Ok, look through them and find anything that looks relevant, look out for
anything that mentions the stuff they were protesting about. Once you do, you'll
need to upload them to me, I guess go back to your apartments and we'll do it
from there, let me know once you're there.

\u001B[35mMarie\u001B[0m: I'll be making sure your cover is not blown.

\u001B[33mHint: you can use the documents to read them.\u001B[0m"""
        option_q = "Ok."

        [marie.comms.home]
        dialog = """\u001B[35mMarie\u001B[0m: Presumably you have the documents, turn your computer on, go to the link
I've sent you, it should show up, and upload them."""

[stage]
reached_conclusion = """\n
You've reached the end of the game, you're now in an open world mode, keep
exploring anything you haven't to maximise your score.

You can conclude by typing \u001B[47m\u001B[30mwin\u001B[0m
""""
```

```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 }
    ] }
```

```
    protestors = { description = "protestors.dialog", options = [
        { description = "protestors.option_1", to = "leaflet" },
        { description = "small_talk.option_q", to = "small_talk", mustExit = true }
    ] }
    enquire = { description = "enquire.dialog", options = [
        { description = "protestors.option_1", to = "leaflet", },
        { description = "small_talk.option_q", to = "small_talk", mustExit = true }
    ] }
    leaflet = { description = "leaflet.dialog", options = [
        { description = "leaflet.option_1", to= "small_talk", mustExit = true }
    ] }
    #recon = { description = "testing", options = [
    #    { description = "susususus!", to = "recon", mustExit = true }
    #] }

# 3. Old Man located in Forest
[npc_old_man]
    _prefix = "forest.old_man."
    _start = "small_talk"
    small_talk = { description = "small_talk.dialog", options = [
        { description = "small_talk.option_1", to = "request" },
        { description = "small_talk.option_q", to = "small_talk", mustExit = true }
    ] }
    request = { description = "request.dialog", options = [
        { description = "request.option_1", to = "small_talk", mustExit = true },
        { description = "request.option_q", to = "small_talk", mustExit = true }
    ] }
    praise = { description = "praise.dialog", options = [
        { description = "praise.option_q", to = "praise", mustExit = true }
    ] }

# 4. Protestors located on Street
[npc_protestors]
    _prefix = "street.protestors."
    _start = "small_talk"
    small_talk = { description = "small_talk.dialog", options = [
        { description = "small_talk.option_1", to = "enquire" },
        { description = "small_talk.option_2", to = "small_talk", mustExit = true }
    ] }
    enquire = { description = "enquire.dialog", options = [
        { description = "enquire.option_1", to = "leaks" },
```

```
        { description = "enquire.option_2", to = "confrontation" }
    ] }
    leaks = { description = "leaks.dialog", options = [
        { description = "leaks.option_1", to = "small_talk", mustExit = true }
    ] }
    confrontation = { description = "confrontation.dialog", options = [
        { description = "confrontation.option_1", to = "small_talk", mustExit = true },
        { description = "confrontation.option_2", to = "small_talk", mustExit = true }
    ] }

# 5. Security guard stationed at Medical Centre
[npc_security_guard]
    _prefix = "medical_centre.guard."
    _start = "small_talk"
    small_talk = { description = "small_talk.dialog", options = [
        { description = "small_talk.option_q", to = "small_talk", mustExit = true }
    ] }

# 6. Shop shopkeeper in the Shop
[npc_shopkeeper]
    _prefix = "shop.npc."
    _start = "index"
    index = { description = "index", options = [
        { description = "leave", to = "index", mustExit = true }
    ] }
    recon = { description = "recon", options = [
        { description = "leave", to = "recon", mustExit = true }
    ] }
    stealth = { description = "stealth", options = [
        { description = "leave", to = "stealth", mustExit = true }
    ] }

# 7. Receptionist located in Apartments
[npc_receptionist]
    _prefix = "apartments.receptionist."
    _start = "first_encounter"
    first_encounter = { description = "first_encounter.dialog", options = [
        { description = "first_encounter.option_1", to = "protestors" },
        { description = "first_encounter.option_q", to = "repeated", mustExit = true }
    ] }
    protestors = { description = "protestors.dialog", options = [
```

```
        { description = "protestors.option_1", to = "protestors2" },
        { description = "protestors.option_q", to = "repeated" }
    ] }
    protestors2 = { description = "protestors2.dialog", options = [
        { description = "protestors2.option_q", to = "repeated" }
    ] }
    repeated = { description = "repeated.dialog", options = [
        { description = "repeated.option_1", to = "protestors" },
        { description = "repeated.option_q", to = "repeated", mustExit = true }
    ] }

# 8. TV located in Apartments
[home_tv]
    _prefix = "home.tv."
    _start = "first_on"
    first_on = { description = "first_on.dialog", options = [
        { description = "keep_watching", to = "dialog_a" },
        { description = "off", to = "first_on", mustExit = true }
    ] }
    dialog_a = { description = "first_on.dialog_a", options = [
        { description = "keep_watching", to = "dialog_b" },
        { description = "off", to = "first_on", mustExit = true }
    ] }
    dialog_b = { description = "first_on.dialog_b", options = [
        { description = "keep_watching", to = "dialog_c" },
        { description = "off", to = "first_on", mustExit = true }
    ] }
    dialog_c = { description = "first_on.dialog_c", options = [
        { description = "off", to = "first_on", mustExit = true }
    ] }

# 9. Communicator device with Marie
[comms_marie]
    _prefix = "marie.comms."
    _start = "orientation"

    orientation = { description = "orientation.dialog", options = [
        { description = "orientation.option_1", to = "directions" },
        { description = "orientation.option_2", to = "hear" },
        { description = "orientation.option_q", to = "orientation", mustExit = true }
    ] }
```

```toml
    directions = { description = "directions.dialog", options = [
        { description = "directions.option_q", to = "orientation" }
    ] }
    hear = { description = "hear.dialog", options = [
        { description = "hear.option_q", to = "orientation" }
    ] }

    complaint = { description = "complaint.dialog", options = [
        { description = "complaint.option_q", to = "complaint", mustExit = true }
    ] }

    in_position = { description = "in_position.dialog", options = [
        { description = "in_position.option_1", to = "distraction" }
    ] }
    distraction = { description = "distraction.dialog", options = [
        #{ description = "in_position.option_1", to = "coast_is_clear" }
    ] }
    coast_is_clear = { description = "coast_is_clear.dialog", options = [
        { description = "coast_is_clear.option_q", to = "gogogo", mustExit = true }
    ] }
    gogogo = { description = "gogogo.dialog", options = [
        { description = "gogogo.option_q", to = "gogogo", mustExit = true }
    ] }

    office = { description = "office.dialog", options = [
        { description = "office.option_1", to = "documents" }
    ] }
    documents = { description = "office.dialog", options = [
        { description = "office.option_q", to = "office", mustExit = true }
    ] }

    home = { description = "home.dialog", options = [
        { description = "home.option_q", to = "home", mustExit = true }
    ] }

# 10. Laptop
[entity_laptop]
    _prefix = "entities.laptop."
    _start = "boot"

    boot = { description = "boot.dialog", options = [
```

```
        { description = "boot.option_1", to = "home" }
    ] }
    home = { description = "home.dialog", options = [
        { description = "home.option_q", to = "boot", mustExit = true },
        { description = "home.option_1", to = "pictures" },
        { description = "home.option_2", to = "funny_cat_videos" },
        { description = "home.option_3", to = "document" }
    ] }
    pictures = { description = "pictures.dialog", options = [
        { description = "pictures.option_q", to = "home" }
    ] }
    document = { description = "document.dialog", options = [
        { description = "document.option_q", to = "home" }
    ] }
```

```toml
# Path to emoji root directory
emojis = "/emojis"

# Path to font file
# https://fonts.google.com/specimen/VT323?category=Monospace#standard-styles
font = "/VT323-Regular.ttf"
```