World of Deez

Joe

19th November 2021

1 First Section

so we do a lot of writing

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin congue leo nec ligula euismod, sit amet efficitur mi condimentum. Nulla pulvinar, justo sed sollicitudin euismod, mi velit vestibulum diam, et mollis odio felis id nulla. Nam id velit nec lacus fringilla cursus. Nulla quis tempus erat, a dignissim urna. Curabitur justo risus, varius vel iaculis finibus, eleifend quis metus. Proin id bibendum purus, non elementum ante. Vestibulum eu orci non erat luctus iaculis. Suspendisse potenti. Praesent libero leo, condimentum eu mauris et, mollis efficitur ligula. Duis eget nisi erat. Donec urna ipsum, convallis at faucibus non, maximus quis orci. Donec vitae neque ac dui tempor scelerisque id faucibus nulla. Nullam vel suscipit metus. Mauris vel semper nisl, eu elementum arcu.

Vivamus in leo et enim laoreet tempor. Phasellus feugiat volutpat neque, a faucibus justo fringilla eleifend. Pellentesque condimentum ipsum nec justo blandit aliquam ac et nisl. Donec placerat et dui ac varius. Integer faucibus odio est, nec hendrerit ex scelerisque ac. In viverra tortor id aliquet eleifend. Sed ut dignissim ligula. Aenean dapibus ornare venenatis.

2 Second

Nulla vitae sodales tellus. Donec vel placerat velit. Sed ac consequat magna. Maecenas lacus magna, consequat eget dictum a, accumsan non nisl. Pellentesque vel mauris ullamcorper, pellentesque ipsum et, congue nisl. Aliquam faucibus nulla aliquam pulvinar ultricies. In ut odio id lectus tristique sodales eu ac ante. Donec commodo commodo scelerisque. Praesent ullamcorper, lacus non malesuada dignissim, tortor lectus elementum mauris, eget tempus ante nunc quis lectus. Sed gravida metus et urna tempus porta. Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Sed lorem metus, accumsan ut erat ut, vestibulum porta lorem. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia curae; Cras nec nisi mauris. Integer nisi nisi, gravida sed sodales quis, dignissim eu nisi. Vestibulum sollicitudin mattis ligula, vitae rutrum magna rhoncus ut. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam consectetur odio id diam fermentum, malesuada vehicula sapien accumsan. Fusce auctor quam sit amet imperdiet venenatis. Duis gravida pellentesque nibh, nec luctus mauris. Cras ipsum libero, rhoncus nec pretium non, hendrerit a sem. Fusce blandit dapibus est, vitae fermentum lectus mattis eu. Morbi et pretium nisi. Donec congue, nunc quis posuere hendrerit, elit risus vulputate velit, ut tempus nisi ipsum quis sapien. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Nunc libero lorem, feugiat sit amet erat ut, rutrum scelerisque purus.

Ut efficitur odio non tincidunt volutpat. Vivamus et fermentum libero, sit amet volutpat elit. Etiam egestas orci non pharetra facilisis. Mauris faucibus turpis vel purus luctus aliquam. Quisque vehicula felis id nulla mollis aliquet. In hac habitasse platea dictumst. Sed aliquet elementum nisl sed efficitur. In lobortis leo purus, nec pretium nunc suscipit sed. Aenean pretium porttitor est. Nam mollis eget quam et facilisis. Phasellus tincidunt tellus eu sapien gravida laoreet. Lorem ipsum dolor sit amet, consectetur adipiscing elit. In pellentesque convallis euismod. Sed ut lobortis enim. Aliquam a felis metus. Quisque id nibh tincidunt, tincidunt urna sed, interdum risus.

Curabitur velit dui, luctus ut tincidunt vel, condimentum ut nisi. Ut porttitor odio nisl, non facilisis turpis molestie non. Sed consectetur ipsum eu rhoncus bibendum. In iaculis scelerisque semper. Nunc ac nisi nec justo tempus sollicitudin non sed diam. Integer lacinia sit amet orci vitae posuere. Aliquam vehicula sapien id turpis tempor malesuada rhoncus sed risus. Cras et scelerisque turpis, sed tristique nisi. Sed id dui sed lorem feugiat egestas et id nisl. Nunc ut venenatis mauris.

Etiam vulputate, sem non laoreet facilisis, leo odio viverra lorem, at elementum quam ipsum et nunc. Pellentesque a augue volutpat, ultricies ligula vel, egestas felis. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia curae; Phasellus tincidunt quam justo, at condimentum purus rhoncus nec. Vivamus vel nisi sit amet ex mattis viverra quis ac leo.

```java
package uk.insrt.coursework.zuul.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.EventTick;
import uk.insrt.coursework.zuul.world.Room;

public class SimpleWanderAI implements IEventListener<EventTick> {
    private Entity entity;
    private Room[] path;
    private int chance;

    private int index;
    private Random random;

    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;

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

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

import java.util.regex.Matcher;

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

public class Arguments {
    private Matcher matcher;

    public Arguments(Matcher matcher) {
        this.matcher = matcher;
    }

    public String group(String group) {
        try {
            return this.matcher.group(group);
        } catch (Exception e) {
            return null;
        }
    }

    public Direction direction() {
        return Direction.fromString(this.group("direction"));
    }
}
```

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

import java.util.regex.Pattern;

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 usage;

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

    /**
     * Get information about how to use the command.
     * @return String Information about how to use the command
     */
    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;
    }

    /**
     * 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);
}
```

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

import java.util.ArrayList;
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.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.CommandUse;
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();
    }

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

    public void registerCommands(Command[] commands) {
        for (Command command : commands) {
            this.registerCommand(command);
        }
    }

    public ArrayList<Command> getCommands() {
        return this.commands;
    }

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

        this.registerCommands(DEFAULT_COMMANDS);
    }

    /**
     * Interpret a given command and execute it within the scope of a given world.
     * @param world Current World object
     * @param cmd Arbitrary input to match against
     * @return Boolean indicating whether the game loop should exit.
     */
    public boolean runCommand(World world, String cmd) {
        for (Command command : this.commands) {
            for (Pattern pattern : command.getPatterns()) {
                Matcher matcher = pattern.matcher(cmd);
                if (matcher.find()) {
                    Arguments arguments = new Arguments(matcher);
                    return command.run(world, arguments);
                }
            }
        }

        System.out.println("Not sure what you're trying to do.");
        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;

public class CommandBack extends Command {
    public CommandBack() {
        super("back: go back to the previous room",
            new Pattern[] {
                Pattern.compile("^back(?!\\w)"),
            });
    }

    @Override
    public boolean run(World world, Arguments arguments) {
        world.getPlayer().back();
        return false;
    }
}
```

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

import java.util.regex.Pattern;

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

public class CommandGo extends Command {
    public CommandGo() {
        super("go <direction>: go in a certain direction",
            new Pattern[] {
                Pattern.compile("^go\\s+(?<direction>[\\w\\s]+)"),
                Pattern.compile("^go")
            });
    }

    @Override
    public boolean run(World world, Arguments arguments) {
        Direction direction = arguments.direction();
        if (direction == null) {
            System.out.println("Where are you going?");
            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.world.World;

public class CommandHelp extends Command {
    private CommandManager commandManager;

    public CommandHelp(CommandManager commandManager) {
        super("help: show help menu",
            new Pattern[] {
                Pattern.compile("^help(?!\\w)")
            });

        this.commandManager = commandManager;
    }

    @Override
    public boolean run(World world, Arguments arguments) {
        System.out.println("You can run the following commands:");
        System.out.println(
            this.commandManager
                .getCommands()
                .stream()
                .filter(c -> !(c instanceof CommandHelp))
                .map(c -> "- " + c.getUsage())
                .collect(Collectors.joining("\n"))
        );

        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.world.World;

public class CommandPet extends Command {
    public CommandPet() {
        super("pet <something>: pet something in current room",
            new Pattern[] {
                Pattern.compile("^pet\\s+(?<entity>[\\w\\s]+)"),
                Pattern.compile("^pet")
            });
    }

    @Override
    public boolean run(World world, Arguments arguments) {
        String name = arguments.group("entity");
        if (name == null) {
            System.out.println("Pet what?");
            return false;
        }

        Entity entity = world.findEntity(name);
        if (entity != null) {
            if (!entity.pet()) {
                System.out.println("You cannot pet " + name + ".");
            }

            return false;
        }

        System.out.println("You look around for " + name + " but can't find
anything.");
        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;

public class CommandQuit extends Command {
    public CommandQuit() {
        super("quit: quit the game",
            new Pattern[] {
                Pattern.compile("^quit(?!\\w)"),
            });
    }

    @Override
    public boolean run(World world, Arguments arguments) {
        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.world.World;

public class CommandTake extends Command {
    public CommandTake() {
        super("take <something>: put something in your bag",
            new Pattern[] {
                Pattern.compile("^take\\s+(?<entity>[\\w\\s]+)"),
                Pattern.compile("^take")
            });
    }

    @Override
    public boolean run(World world, Arguments arguments) {
        String name = arguments.group("entity");
        if (name == null) {
            System.out.println("Take what?");
            return false;
        }

        Entity entity = world.findEntity(name);
        if (entity != null) {
            if (entity.take(world.getPlayer())) {
                System.out.println("You take " + name + " and put it in your
bag.");
            } else {
                System.out.println("You cannot take " + name + ".");
            }

            return false;
        }

        System.out.println("You look around for " + name + " but can't find
anything.");
        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.world.World;

public class CommandUse extends Command {
    public CommandUse() {
        super("use <something>: use an object or something in your inventory",
            new Pattern[] {
                Pattern.compile("^use\\s+(?<entity>[\\w\\s]+)"),
                Pattern.compile("^use")
            });
    }

    @Override
    public boolean run(World world, Arguments arguments) {
        String name = arguments.group("entity");
        if (name == null) {
            System.out.println("Use what?");
            return false;
        }

        Entity entity = world.findEntity(name);
        if (entity != null) {
            if (!entity.use(world.getPlayer())) {
                System.out.println("You cannot use " + name + ".");
            }

            return false;
        }

        System.out.println("You look around for " + name + " but can't find 
anything.");
        return false;
    }
}
```

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

import java.util.ArrayList;
import java.util.stream.Collectors;

import uk.insrt.coursework.zuul.behaviours.SimpleWanderAI;
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.RoomMedicalCentreRecepti
on;
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.entities.Entity;
import uk.insrt.coursework.zuul.entities.EntityCat;
import uk.insrt.coursework.zuul.entities.EntityPlayer;
import uk.insrt.coursework.zuul.events.EventEntityEnteredRoom;
import uk.insrt.coursework.zuul.events.EventEntityLeftRoom;
import uk.insrt.coursework.zuul.events.IEventListener;
import uk.insrt.coursework.zuul.world.Room;
import uk.insrt.coursework.zuul.world.World;

// https://democracy.york.gov.uk/documents/s2116/
Annex%20C%20REcycling%20Report%20frnweights2005.pdf
// https://www.google.com/maps/@50.4293559,18.9742453,16.12z
// https://twitter.com/Yarung3/status/1258670295520628736/photo/1
// https://twitter.com/jgilleard/status/1242354985351786497
// [3:02] https://brand-new-animal.fandom.com/wiki/Runaway_Raccoon

public class CampaignWorld extends World {
    private ArrayList<Room> visitedRooms;

    public CampaignWorld() {
        super();
        this.visitedRooms = new ArrayList<>();

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

    public boolean hasVisited(Room room) {
        return this.visitedRooms.contains(room);
    }

    private void buildWorld() {
        final Room[] rooms = {
            new RoomCityCentre(this),
            new RoomStreet(this),
            new RoomShop(this),
            new RoomBackAlley(this),
            new RoomApartmentsReception(this),
            new RoomApartmentsHome(this),
            new RoomMedicalCentreReception(this),
                new RoomMedicalCentreOffice(this),
                new RoomCoastline(this),
                new RoomMainlandCoastline(this),
                new RoomForest(this),
                new RoomWormHole(this)
            };

            for (Room room : rooms) {
                this.addRoom(room);
            }

            this.linkRooms();
        }

        private void spawnEntities() {
            for (Room room : this.rooms.values()) {
                room.spawnEntities();
            }
        }

        /**
         * Register all the game logic
         */
        private void registerEvents() {
            super.registerDefaultEvents();

            this.eventSystem.addListener(EventEntityEnteredRoom.class, (IEventListe
ner<EventEntityEnteredRoom>) this.getRoom("Worm Hole"));
            this.eventSystem.addListener(EventEntityEnteredRoom.class,
                (EventEntityEnteredRoom event) -> {
                    Entity entity = event.getEntity();
                    if (entity instanceof EntityPlayer) {
                        Room room = entity.getRoom();

                        // Mark current room as previously visited.
                        this.visitedRooms.add(room);

                        // When we enter a new room, list what we can see.
                        String entities = this.getEntitiesInRoom(entity.getRoom())
                            .stream()
                            .filter(e -> !(e instanceof EntityPlayer))
                            .map(e -> "- " + e.describe())
                            .collect(Collectors.joining("\n"));

                        if (entities.length() > 0) {
                            System.out.println("You can see:\n" + entities);
                        }
                    } else {
                        // If another entity enters the room,
                        // conditionally mention this to the player.
                        EntityPlayer player = this.getPlayer();
                        if (entity.getRoom() == player.getRoom()) {
                            if (entity instanceof EntityCat) {
                                System.out.println("\nA cat has wandered in.");
                            }
                        }
                    }
                });

            this.eventSystem.addListener(EventEntityLeftRoom.class,
                (EventEntityLeftRoom event) -> {
                    Entity entity = event.getEntity();
                    if (entity instanceof EntityPlayer) return;

                    Room room = event.getRoom();
                    if (room != this.player.getRoom()) return;

                    // If another entity leaves the room,
                    // conditionally mention this to the player.
                    if (entity instanceof EntityCat) {
                        System.out.println("\nYou see a cat leave.");
                    }
                });
    }

    @Override
    public void spawnPlayer() {
        this.player.setLocation(this.rooms.get("City Centre"));
    }
}
```

```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.IActionUse;
import uk.insrt.coursework.zuul.events.EventTick;
import uk.insrt.coursework.zuul.world.Location;
import uk.insrt.coursework.zuul.world.World;

public class EntityBed extends EntityObject implements IActionUse {
    public EntityBed(World world, Location location) {
        super(world, location, 80, new String[] { "bed" }, "Bed");
    }

    public void use(Entity target) {
        // for example, we could move the world forwards by 20 ticks
        for (int i=0;i<20;i++) {
            world.emit(new EventTick());
        }

        System.out.println("You take a nap.");
    }
}
```

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

import uk.insrt.coursework.zuul.entities.Entity;
import uk.insrt.coursework.zuul.entities.actions.IActionUse;
import uk.insrt.coursework.zuul.world.Location;
import uk.insrt.coursework.zuul.world.Room;
import uk.insrt.coursework.zuul.world.World;

public class EntityBoat extends Entity implements IActionUse {
    private Room destination;

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

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

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

    public void use(Entity target) {
        target.setLocation(this.destination);
    }

    @Override
    public boolean pet() {
        return false;
    }
}
```

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

import uk.insrt.coursework.zuul.content.campaign.entities.EntityBed;
import uk.insrt.coursework.zuul.entities.EntityObject;
import uk.insrt.coursework.zuul.world.Direction;
import uk.insrt.coursework.zuul.world.Room;
import uk.insrt.coursework.zuul.world.World;

public class RoomApartmentsHome extends Room {
    public RoomApartmentsHome(World world) {
        super(world, "Apartments: Home");
    }

    public String describe() {
        return this.getName();
    }

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

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

        world.spawnEntity("bed", new EntityBed(world, this.toLocation()));
        world.spawnEntity("laptop", new EntityObject(world, this.toLocation(), 2, new String[] { "laptop" }, "Laptop"));
    }
}
```

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

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

public class RoomApartmentsReception extends Room {
    public RoomApartmentsReception(World world) {
        super(world, "Apartments: Reception");
    }

    public String describe() {
        return this.getName();
    }

    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"));
    }
}
```

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

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

public class RoomBackAlley extends Room {
    public RoomBackAlley(World world) {
        super(world, "Back Alley");
    }

    public String describe() {
        return this.getName();
    }

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

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

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

public class RoomCityCentre extends Room {
    public RoomCityCentre(World world) {
        super(world, "City Centre");
    }

    public String describe() {
        if (((CampaignWorld) this.getWorld()).hasVisited(this)) {
            return "you've been here before";
        }

        return "something something long description.";
    }

    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"));
    }

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

        EntityCat cat = new EntityCat(world, this.toLocation());
        world.spawnEntity("cat", cat);
        cat.useWanderAI(
            new Room[] {
                world.getRoom("City Centre"),
                world.getRoom("Street"),
                world.getRoom("Shop"),
                world.getRoom("Street"),
                world.getRoom("City Centre"),
                world.getRoom("Back Alley"),
                world.getRoom("City Centre")
            },
            8
        );
    }
}
```

```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.Room;
import uk.insrt.coursework.zuul.world.World;

public class RoomCoastline extends Room {
    public RoomCoastline(World world) {
        super(world, "Coastline");
    }

    public String describe() {
        return this.getName();
    }

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

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

public class RoomForest extends Room {
    public RoomForest(World world) {
        super(world, "Forest");
    }

    public String describe() {
        return this.getName();
    }

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

```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.Room;
import uk.insrt.coursework.zuul.world.World;

public class RoomMainlandCoastline extends Room {
    public RoomMainlandCoastline(World world) {
        super(world, "Mainland: Coastline");
    }

    public String describe() {
        return this.getName();
    }

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

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

public class RoomMedicalCentreOffice extends Room {
    public RoomMedicalCentreOffice(World world) {
        super(world, "Medical Centre: Office");
    }

    public String describe() {
        return "You find yourself at the Medical Centre's office.\nYou definitely shouldn't be here...";
    }

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

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

import uk.insrt.coursework.zuul.entities.EntityNPC;
import uk.insrt.coursework.zuul.world.Direction;
import uk.insrt.coursework.zuul.world.Room;
import uk.insrt.coursework.zuul.world.World;

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

    public String describe() {
        return "You're now at the Medical Centre's reception.";
    }

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

    public boolean canLeave(Direction direction) {
        World world = this.getWorld();
        if (direction == Direction.DOWN) {
            if (world.getEntitiesInRoom(this)
                .contains(world.getEntity("guard1"))) {
                System.out.println("There is security watching the stairs,
there's no way to get past them.");
                return false;
            }
        }

        return true;
    }

    public void spawnEntities() {
        World world = this.getWorld();
        world.spawnEntity("guard1", new EntityNPC(world, this.toLocation()) {
            @Override
            public String[] getAliases() {
                return new String[] { "security guard", "guard" };
            }

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

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

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

public class RoomShop extends Room {
    public RoomShop(World world) {
        super(world, "Shop");
    }

    public String describe() {
        return this.getName();
    }

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

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

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

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

    public String describe() {
        return this.getName();
    }

    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"));
    }
}
```

```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.events.EventEntityEnteredRoom;
import uk.insrt.coursework.zuul.events.IEventListener;
import uk.insrt.coursework.zuul.world.Room;
import uk.insrt.coursework.zuul.world.World;

public class RoomWormHole extends Room implements IEventListener<EventEntityEnteredRoom> {
    public RoomWormHole(World world) {
        super(world, "Worm Hole");
    }

    public String describe() {
        return this.getName();
    }

    protected void setupDirections() {}

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

        final Random random = new Random();
        final String[] locations = {
            "City Centre",
            "Coastline",
            "Mainland: Coastline",
            "Forest",
            "Street",
            "Back Alley"
        };

        System.out.println("\nYou step into the worm hole...\n");

        try {
            Thread.sleep(1000);
        } catch (Exception e) { }

        final int WIDTH = 42;

        // Transport animation, this will take 1800 ms.
        for (int i=0;i<5;i++) {
            System.out.println("*".repeat(i*3) + "\\" + " ".repeat(WIDTH - i * 6 - 2) + "/" + "*".repeat(i*3));
            try {
                Thread.sleep(60);
            } catch (Exception e) { }
        }

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

            System.out.println(out);

            try {
                Thread.sleep(40);
            } catch (Exception e) { }
        }

        for (int i=5;i>0;i--) {
            System.out.println("*".repeat(i*3) + "/" + " ".repeat(WIDTH - i * 6 -
2) + "\\" + "*".repeat(i*3));
            try {
                Thread.sleep(60);
            } catch (Exception e) { }
        }

        System.out.println();

        String location = locations[random.nextInt(locations.length)];
        Room target = this.getWorld().getRoom(location);
        entity.setLocation(target);
    }
}
```

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

public class StoryFlags {

}
```

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

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

public interface IActionUse {
    /**
     * Use this entity.
     * @param target The Entity taking this entity.
     * @return Whether this entity can be used.
     */
    public void use(Entity target);
}
```

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

import uk.insrt.coursework.zuul.events.EventEntityEnteredRoom;
import uk.insrt.coursework.zuul.events.EventEntityLeftRoom;
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 int 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, int weight) {
        this.world = world;
        this.location = location;
        this.inventory = new Inventory();
        this.weight = weight;
    }

    /**
     * Construct a new Entity.
     * 
     * Weight value is set to Integer.MAX_VALUE.
     * @param world Current World object
     * @param location Initial Location of this Entity
    */
    public Entity(World world, Location location) {
        this(world, location, Integer.MAX_VALUE);
    }

    /**
     * Get this Entity's weight.
     * @return Weight (in kg)
     */
    public int getWeight() {
        return this.weight;
    }

    /**
     * Get the Inventory that this Entity holds.
     * @return Inventory
     */
    public Inventory getInventory() {
            return this.inventory;
        }

        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();
        }

        /**
         * Move the Entity into a Room.
         * @param room Destination Room
         */
        public void setLocation(Room room) {
            Inventory inventory = this.location.getInventory();
            if (inventory != null) inventory.remove(this);

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

            this.location.setLocation(room);
            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.location.setLocation(inventory);
                return true;
            }

            return false;
        }

        /**
         * Take this entity.
         * @param target The Entity taking this entity
         * @return Whether we managed to take this entity.
         */
        public boolean take(Entity target) {
            Inventory inventory = target.getInventory();
            return this.setLocation(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.
         * @return String describing the Entity
         */
        public abstract String describe();

        /**
         * Pet this entity.
         * @return Whether this entity can be pet.
         */
        public abstract boolean pet();
    }
```

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

import uk.insrt.coursework.zuul.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 {
    public EntityCat(World world, Location startingLocation) {
        super(world, startingLocation, 5);
    }

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

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

    @Override
    public boolean pet() {
        System.out.println("You pet the cat.");
        return true;
    }

    public void useWanderAI(Room[] rooms, int chance) {
        this.getWorld()
            .getEventSystem()
            .onTick(new SimpleWanderAI(this, rooms, chance));
    }
}
```

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

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

/**
 * NPC entity which provides dialog.
 */
public abstract class EntityNPC extends Entity {
    public EntityNPC(World world, Location startingLocation) {
        super(world, startingLocation, 75);
    }

    @Override
    public boolean pet() {
        return false;
    }
}
```

```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;

    public EntityObject(World world, Location location, int weight, String[] aliases, String description) {
        super(world, location, weight);
        this.description = description;
        this.aliases = aliases;
    }

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

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

    @Override
    public boolean pet() {
        return false;
    }
}
```

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

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 Room previousRoom;
    private Direction retreatingDirection;

    public EntityPlayer(World world) {
        super(world, new Location(), 70);
        this.inventory.setMaxWeight(35);
    }

    /**
     * Override method for setLocation which
     * keeps track of previous room.
     */
    @Override
    public void setLocation(Room room) {
        this.previousRoom = this.getRoom();
        super.setLocation(room);
    }

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

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

    @Override
    public boolean take(Entity target) {
        return false;
    }

    @Override
    public boolean pet() {
        return false;
    }

    /**
     * Move in a direction as instructed by command.
     * @param direction Target Direction
     */
    public void go(Direction direction) {
        Room room = this.getRoom();
        if (room == null) {
            System.out.println("You appear to be trapped.");
            return;
        }

        if (!room.canLeave(direction)) return;

        Room destination = room.getAdjacent(direction);
        if (destination == null) {
            System.out.println("You cannot go this way.");
            return;
        }

        this.retreatingDirection = direction.flip();
        this.setLocation(destination);
    }

    /**
     * Move to the previous room the player was in.
     */
    public void back() {
        if (this.retreatingDirection == null) {
            System.out.println("Nowhere to go back to!");
            return;
        }

        if (this.getRoom().hasExit(this.retreatingDirection)) {
            this.setLocation(this.previousRoom);
            this.retreatingDirection = this.retreatingDirection.flip();
        } else {
            System.out.println("Cannot leave the room this way.");
        }
    }
}
```

```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 int 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(int maxWeight) {
        this.maxWeight = maxWeight;
    }

    /**
     * Get the current weight of this inventory.
     * @return Weight (in kg)
     */
    private int getWeight() {
        return this
            .items
            .stream()
            .mapToInt(Entity::getWeight)
            .sum();
    }

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

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

    /**
     * Remove an entity from this inventory.
     * @param entity Target Entity
     * @return Whether there was any change to the inventory.
     */
    public boolean remove(Entity entity) {
        return this.items.remove(entity);
    }
}
```

```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;

    public boolean canRun() {
        return this.propagating;
    }

    public void stopPropagation() {
        this.propagating = false;
    }
}
```

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

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

/**
 * 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;

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

/**
 * 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;

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

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

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

        return list;
    }

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

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

    /**
     * Shorthand for addListener(EventTick.class, listener)
     * @param listener Event listener callback
     */
    public void onTick(IEventListener<EventTick> listener) {
        this.addListener(EventTick.class, 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;

public class EventTick extends Event {}
```

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

public interface IEventListener<E extends Event> {
    public void onEvent(E event);
}
```

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

import java.util.Scanner;

import uk.insrt.coursework.zuul.commands.CommandManager;
import uk.insrt.coursework.zuul.content.campaign.CampaignWorld;
import uk.insrt.coursework.zuul.events.EventProcessCommand;
import uk.insrt.coursework.zuul.events.EventTick;
import uk.insrt.coursework.zuul.world.World;

public class Game {
    private World world;
    private CommandManager commands;

    private Scanner reader;

    public static void main(String[] args) {
        new Game().start();
    }

    public Game() {
        this.world = new CampaignWorld();
        this.commands = new CommandManager();
        this.reader = new Scanner(System.in);
    }

    public void start() {
        this.world.spawnPlayer();

        while (true) {
            System.out.print("\n$ ");
            String input = this.reader.nextLine().toLowerCase();
            System.out.print("\n----\n\n");

            EventProcessCommand event = new EventProcessCommand(input);
            this.world.emit(event);

            if (this.commands.runCommand(this.world, event.getCommand())) {
                break;
            }

            this.world.emit(new EventTick());
        }

        System.out.println("you were game ended");
    }
}
```

```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);
    }

    /**
     * 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;

public class Location {
    private Room room;
    private Inventory inventory;

    public Location() {}

    public Location(Room room) {
        this.room = room;
    }

    public Location(Inventory inventory) {
        this.inventory = inventory;
    }

    public void setLocation(Room room) {
        this.room = room;
        this.inventory = null;
    }

    public void setLocation(Inventory inventory) {
        this.room = null;
        this.inventory = inventory;
    }

    public Room getRoom() {
        return this.room;
    }

    public Inventory getInventory() {
        return this.inventory;
    }
}
```

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

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

public abstract class Room {
    private World world;
    private String name;
    private HashMap<Direction, Room> adjacentRooms;

    public Room(World world, String name) {
        this.world = world;
        this.name = name;
        this.adjacentRooms = new HashMap<>();
    }

    public World getWorld() {
        return this.world;
    }

    public String getName() {
        return this.name;
    }

    public void setAdjacent(Direction direction, Room room) {
        if (room == null) System.err.println("Warning: assigned null Room to direction " + direction + " for the Room " + this.name);
        this.adjacentRooms.put(direction, room);
    }

    public Room getAdjacent(Direction direction) {
        return this.adjacentRooms.get(direction);
    }

    /**
     * Whether the player can leave in any particular direction.
     * Should print reason if not.
     * @param direction Direction which we are checking
     * @return Whether the player can leave
     */
    public boolean canLeave(Direction direction) {
        return true;
    }

    public Set<Direction> getDirections() {
        return this.adjacentRooms.keySet();
    }

    public boolean hasExit(Direction direction) {
        return this.adjacentRooms.containsKey(direction);
    }

    public void linkRooms() {
        this.adjacentRooms.clear();
        this.setupDirections();
    }

    public void spawnEntities() {}

    public Location toLocation() {
            return new Location(this);
        }

        public abstract String describe();
        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.EventEntityEnteredRoom;
import uk.insrt.coursework.zuul.events.EventSystem;

public class World {
    protected Map<String, Room> rooms = new HashMap<>();
    protected Map<String, Entity> entities = new HashMap<>();
    protected EntityPlayer player;

    protected EventSystem eventSystem;

    public World() {
        this.eventSystem = new EventSystem();
        this.player = new EntityPlayer(this);
        this.entities.put("player", this.player);
    }

    public Entity getEntity(String id) {
        return this.entities.get(id);
    }

    public EntityPlayer getPlayer() {
        return this.player;
    }

    public EventSystem getEventSystem() {
        return this.eventSystem;
    }

    public Room getRoom(String room) {
        return this.rooms.get(room);
    }

    protected void addRoom(Room room) {
        this.rooms.put(room.getName(), room);
    }

    public void spawnEntity(String id, Entity entity) {
        this.entities.put(id, entity);
    }

    public List<Entity> getEntitiesInRoom(Room room) {
        return this
            .entities
            .values()
            .stream()
            .filter(e -> e.getRoom() == room)
            .collect(Collectors.toList());
    }

    protected void registerDefaultEvents() {
        this.eventSystem.addListener(EventEntityEnteredRoom.class,
            (EventEntityEnteredRoom event) -> {
                Entity entity = event.getEntity();
                if (entity instanceof EntityPlayer) {
                    Room room = entity.getRoom();
                    System.out.println(
                        room.describe()
                            + "\nYou may go in "
                            + room.getDirections().size()
                            + " directions: "
                            + room.getDirections()
                                .stream()
                                .map(x -> x.toString().toLowerCase())
                                .collect(Collectors.joining(", "))
                    );
                }
            });
    }

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

    public Entity findEntity(String name) {
        List<Entity> entities = this.getEntitiesInRoom(this.getPlayer().getRoom(
));

        for (Entity entity : entities) {
            String[] aliases = entity.getAliases();
            for (String alias : aliases) {
                if (name.equalsIgnoreCase(alias)) {
                    return entity;
                }
            }
        }

        return null;
    }

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

    /**
     * Try to spawn the player in the first available room.
     */
    public void spawnPlayer() {
        this.player.setLocation(this.rooms.values().iterator().next());
    }
}
```