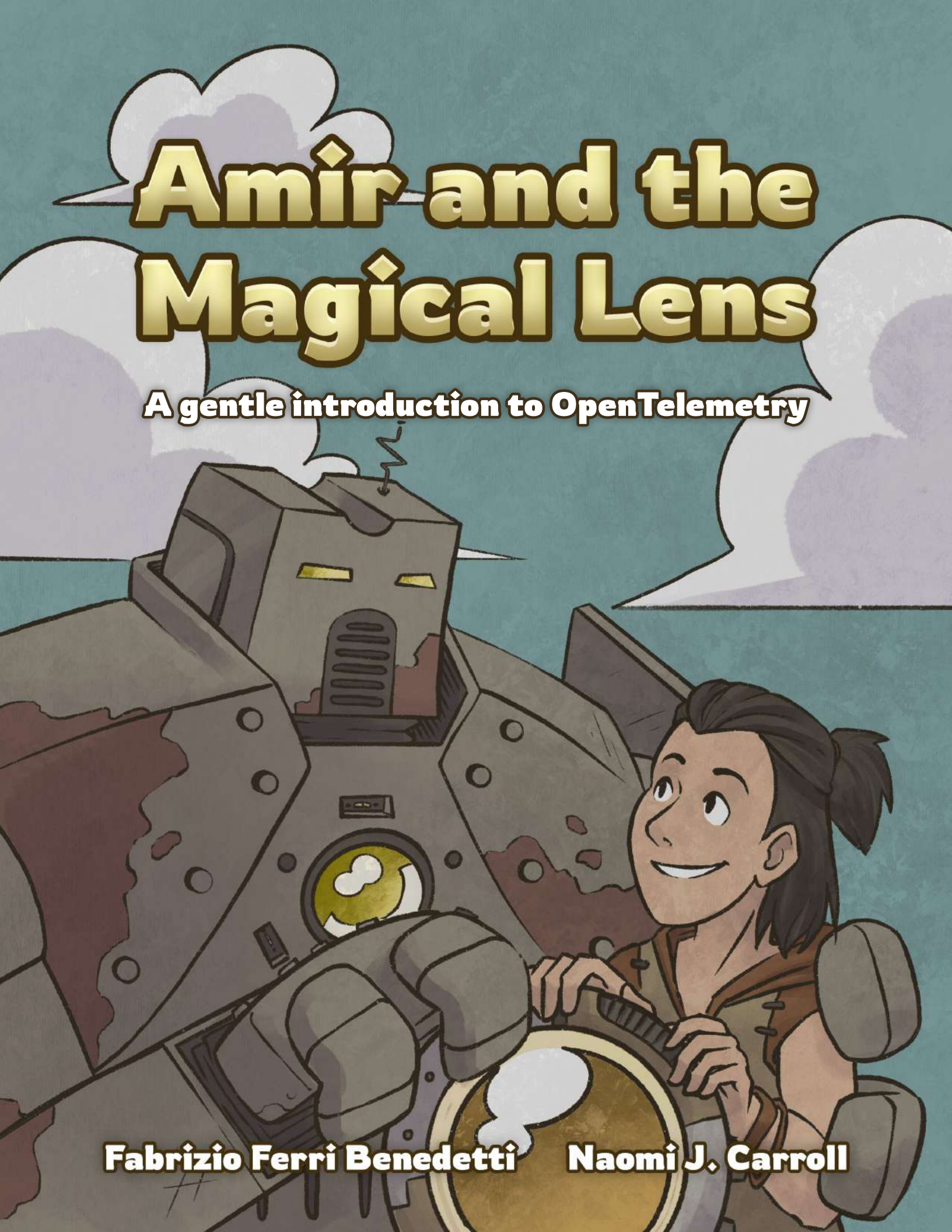
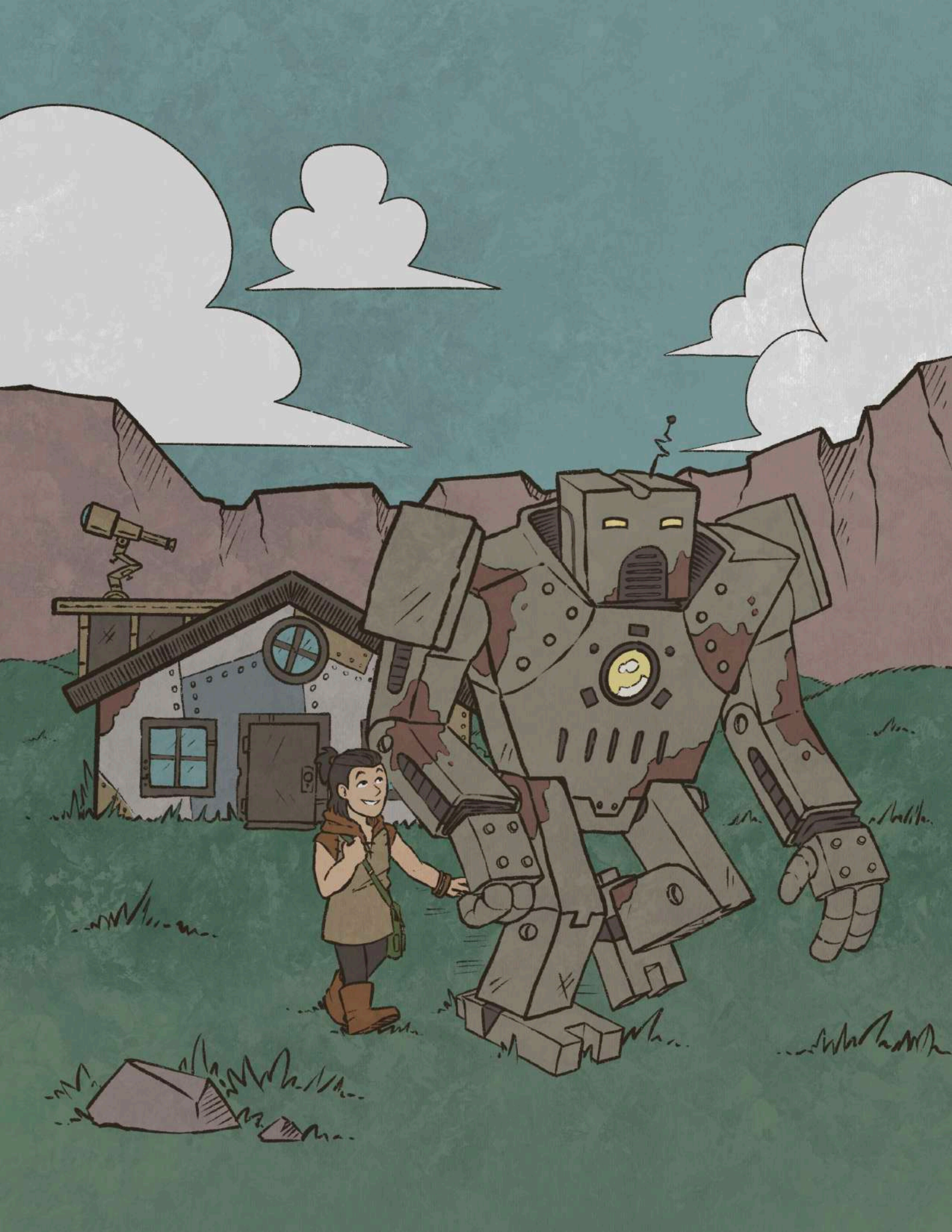

Amir and the Magical Lens
A gentle introduction to OpenTelemetry
Fabrizio Ferri Benedetti
Naomi J. Carroll

Amir and the Magical Lens

Written By
Fabrizio Ferri Benedetti

Illustrated By
Naomi J. Carroll

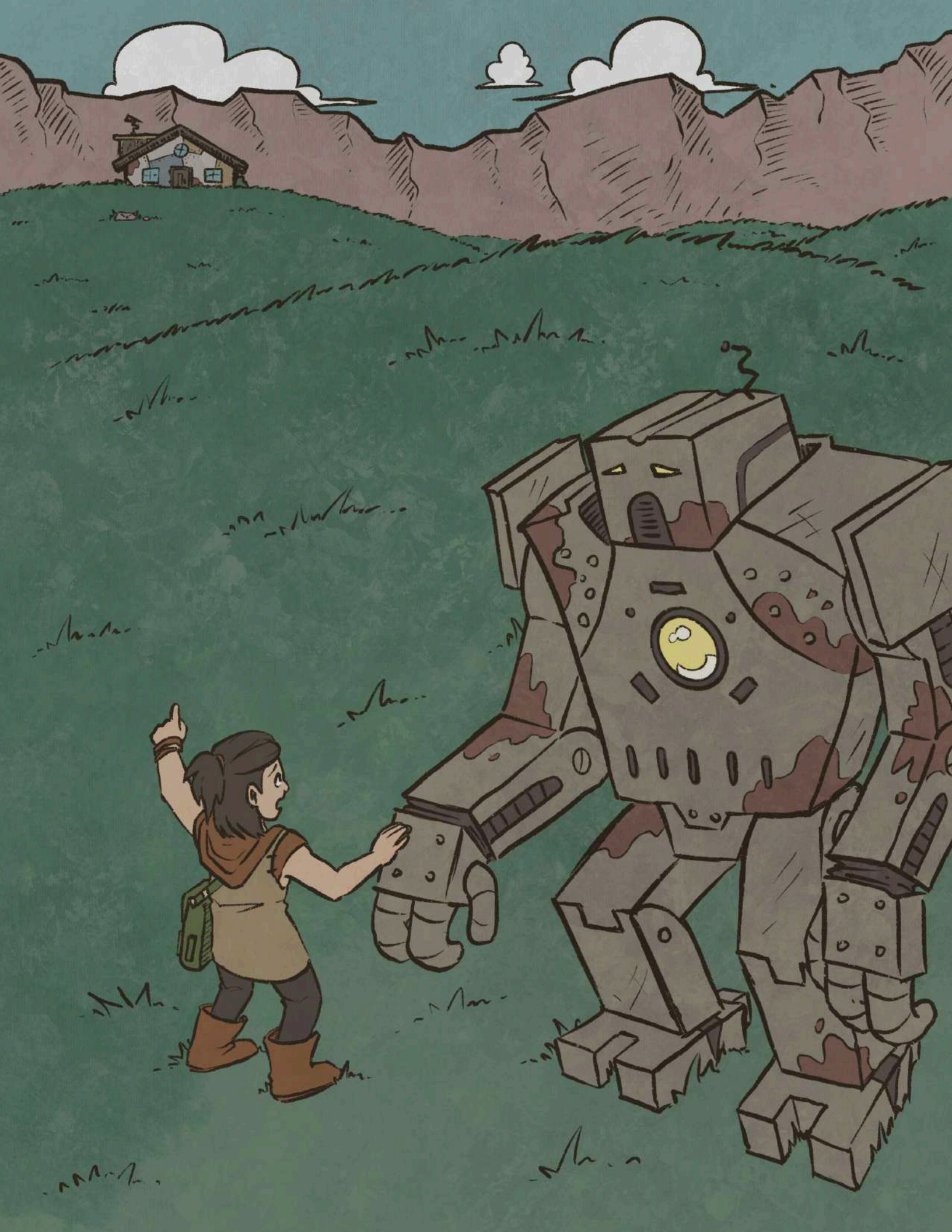


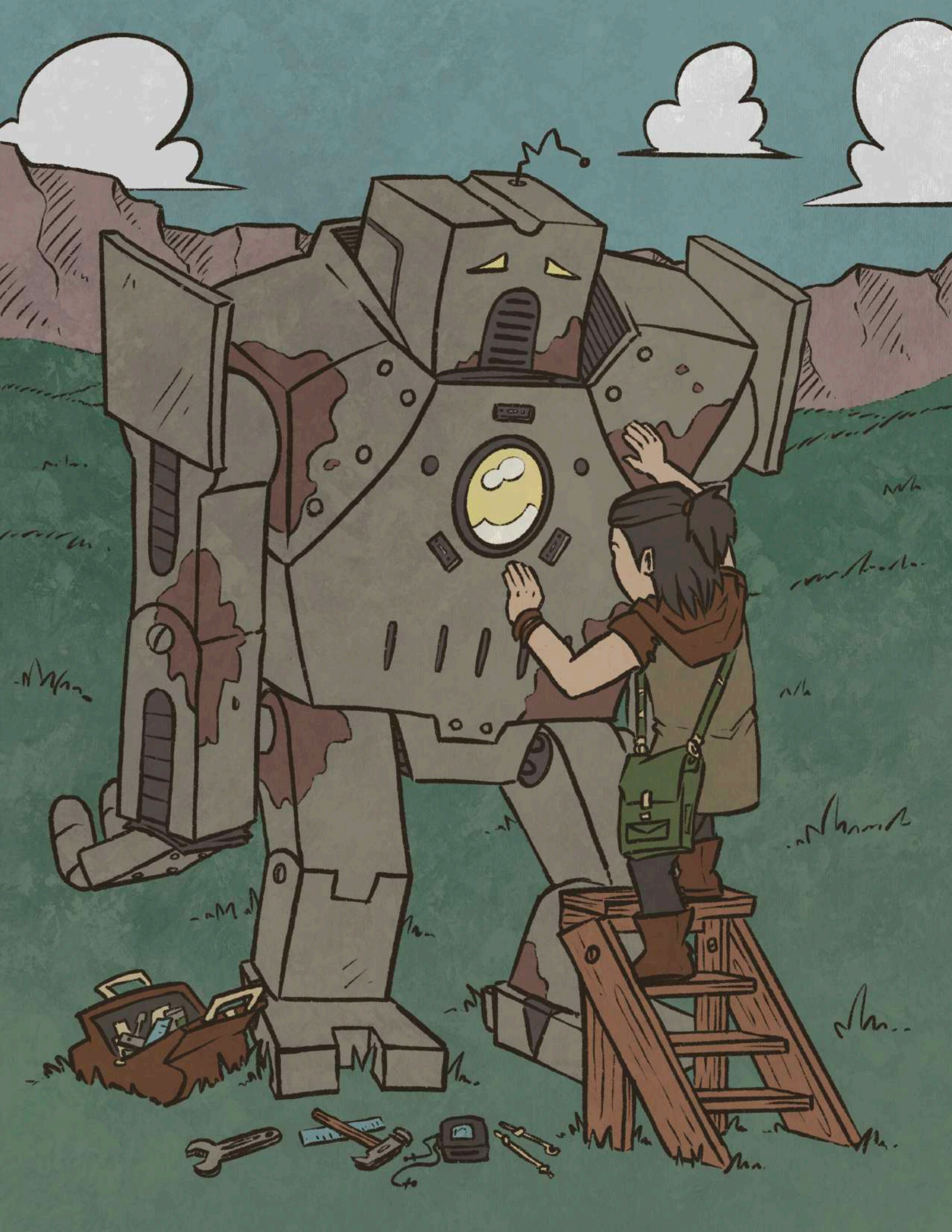
Amir was a smart little boy. He lived alone in the steppe with a huge, gentle robot, Rusty. Amir had built Rusty from scrap metal to have someone who'd help him and keep him company.

For a robot built using used parts, Rusty was remarkably sturdy. He always moved at a slow pace, so Amir had to wait for him sometimes. But Amir didn't mind: Rusty was his friend.

One day, Rusty stopped walking. "What's going on?" Amir asked. Rusty didn't know how to talk, so he pointed at his legs. If he had shoulders, he would have shrugged.

They were only half a mile away from home. "Stay here. I'm going to find out what's wrong with you," said Amir. Rusty nodded and watched his friend rushing for their hut.



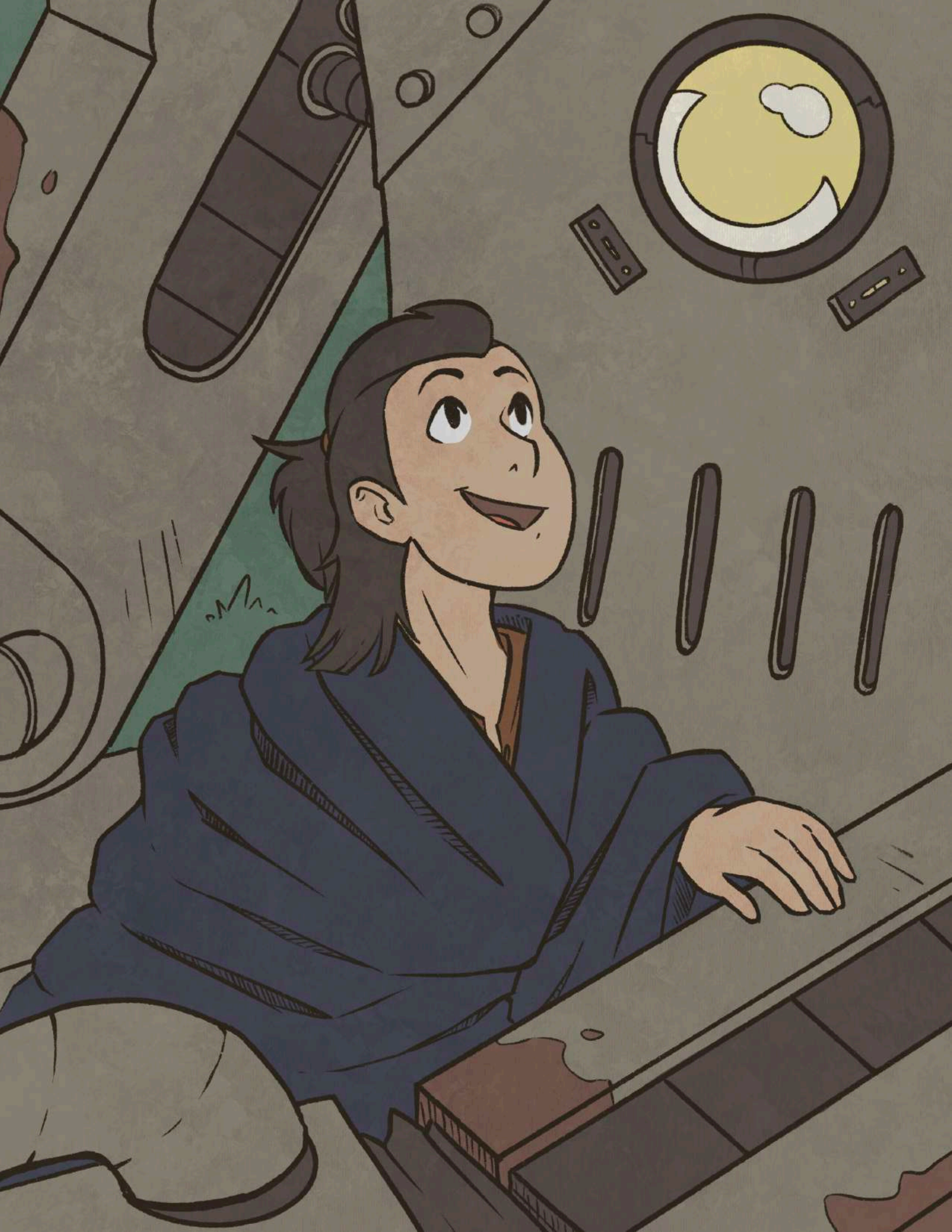


Amir returned with his toolbag. With some effort, he inspected every part that might have failed. Nothing seemed to be broken.

Then, he opened Rusty's power compartment and took a long look at the heart, which was merrily glowing. "I can't tell what's the problem. All your pieces are in good shape," he said.

They waited until night arrived. Rusty lowered one of his heavy hands to shield Amir from the cold, bitter wind of the steppe.

The robot felt that the connection between some of his parts was faulty, but couldn't look inside himself.
He didn't know how he had been assembled. He couldn't help Amir at all, and that thought saddened him.

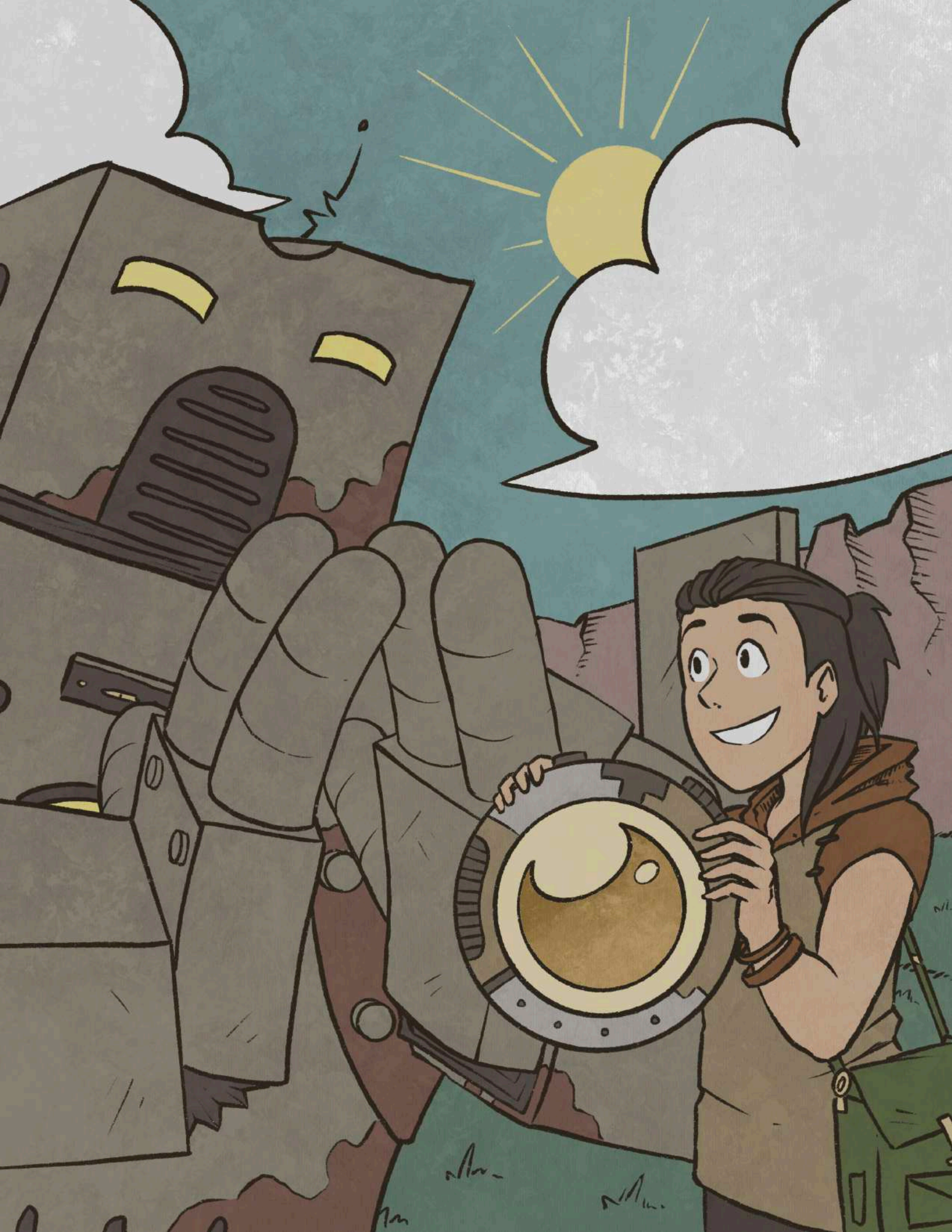


The next day, Amir woke up and found Rusty helplessly looking at him with his yellow eyes. He patted the gigantic arms of his friend. "I have an idea," he said. "I'll be back. Don't worry."

Amir ran home and got into his lab, a tiny shack with a tin roof. He worked for two whole days and nights, sleeping little.

On the morning of the third day, Amir triumphantly raised his new invention: It was a beautiful lens, one that would allow him to look inside Rusty's heart and brain. It was shaped like a big "O" and rimmed with circuits, and could speak of all the light it collected.

Amir called it the "O-Tel."

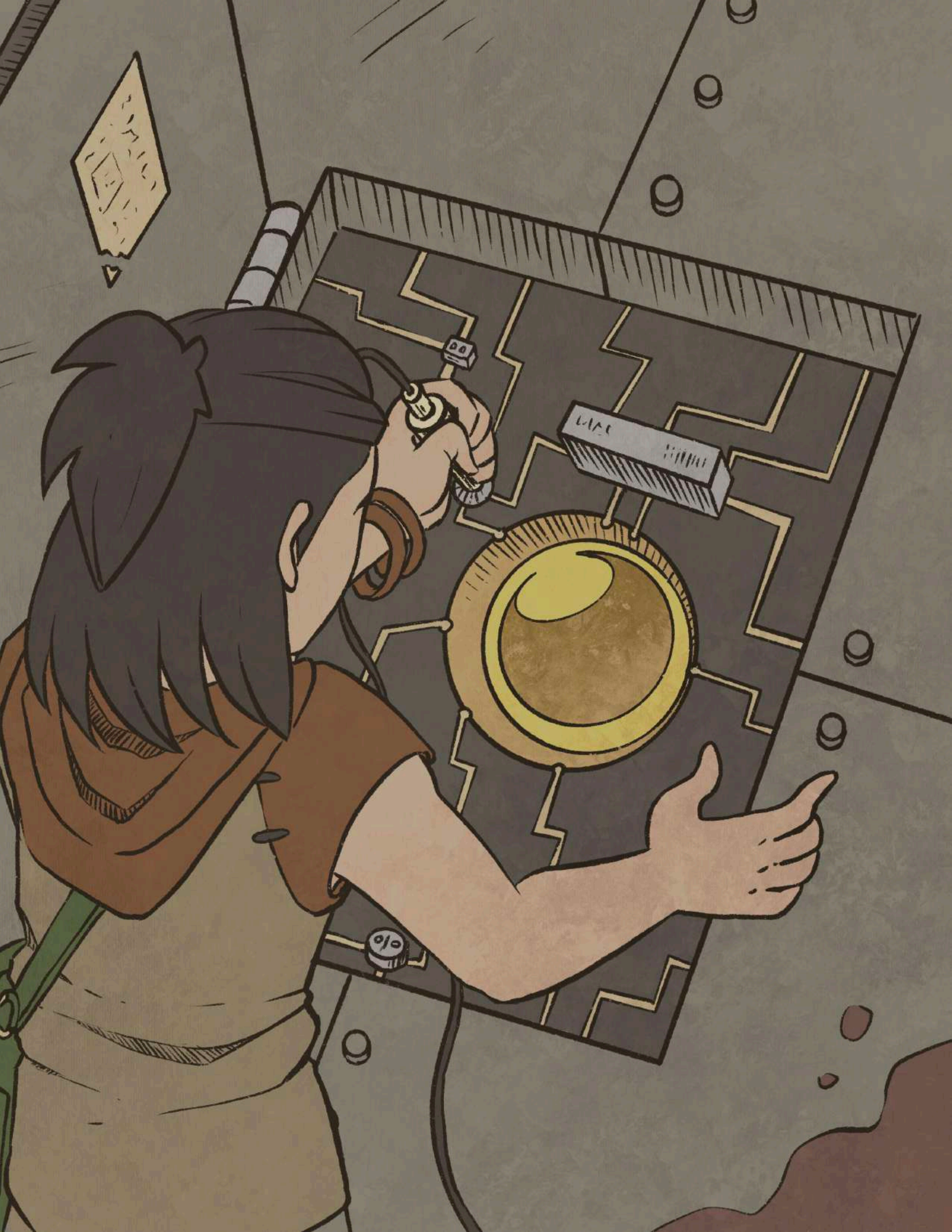


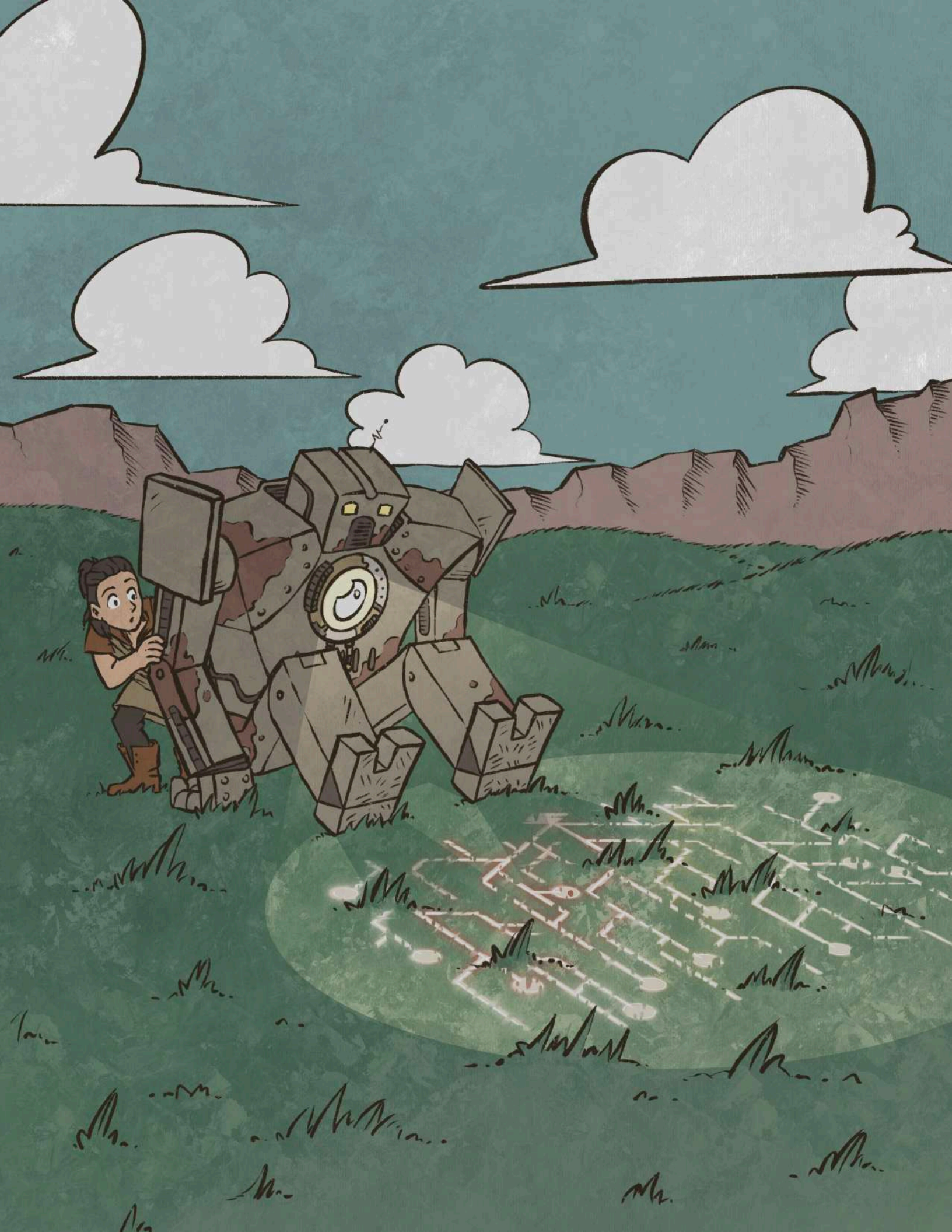
He ran back to Rusty. Amir smiled and raised his arms to show his recent invention. "This is the O-Tel, a special kind of eye. Once I plug it into your heart, it'll allow me to see your thoughts and feelings. Perhaps it'll help us find out what's wrong."

The robot clapped his hands, producing thunderous but joyous booms that echoed for miles. "May I plug it in?" Amir asked. Rusty nodded as fast as he could, which was very slowly.

Amir heartily laughed and jumped to the back of his robot. He opened the rear hatch and kindly plugged the O-Tel in one of the sockets next to Rusty's big heart, instrumenting it.

The lens started glowing with the light of all the signals it collected, which would have been otherwise invisible.



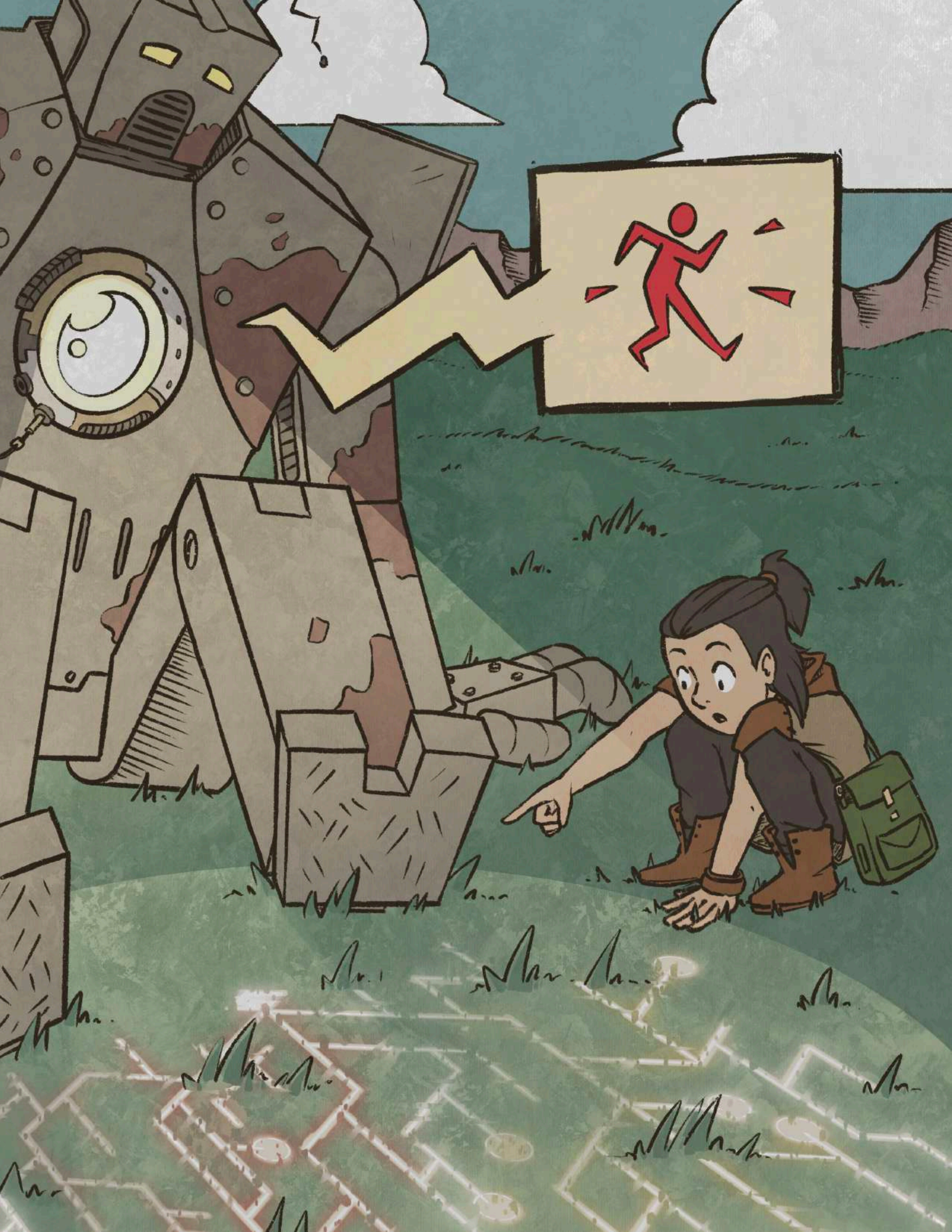


"Rise," Amir ordered with a calm and sweet voice. Rusty didn't move, but light started projecting in front of him: The O-Tel had cast a dozen different traces onto the sandy ground. Each was several meters long, branching like a tree of golden light.

Amir crouched and began examining them all. It was beautiful, but raw. Understanding what each branch meant would take time.

"Look! This branch here: It's red and long. What is this?" Amir asked. "It's the trace for the walking function," the O-Tel immediately replied. Rusty turned his head and saw it too: A long streak of ruby light departed from one of the traces and then stopped.

This much was clear: Rusty couldn't walk because something prevented his walking code from running. But what was it? The web of traces – vast and complex – was distorted by the numerous bushes and rocks.



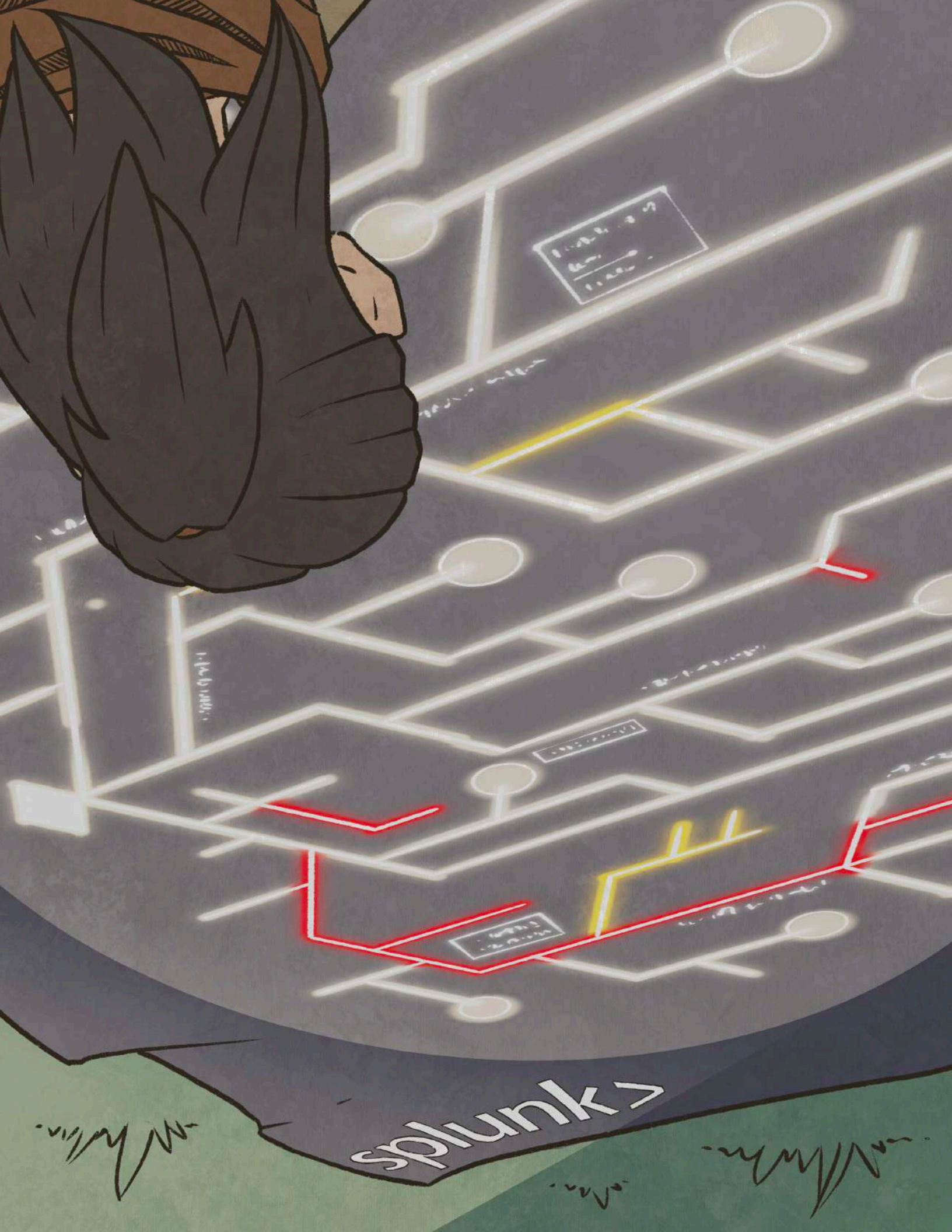

splunk>

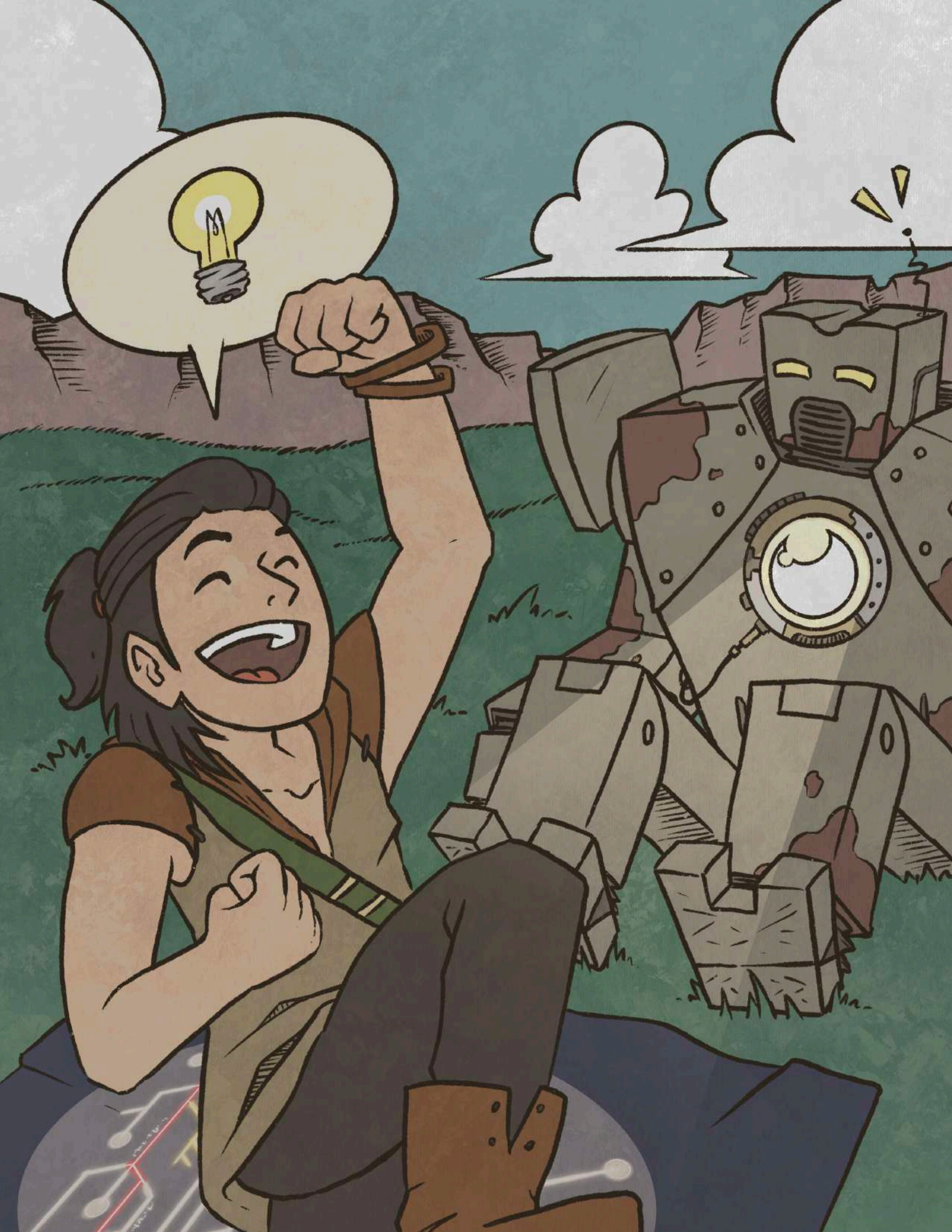
"We need a smoother surface," Amir decided. He pulled a blanket from his backpack and laid it on the ground. The rays of light shed by the O-Tel, previously raw and broken, became sharper and easier to follow when cast onto the dark, wooly background.

Rusty, finally realizing what the traces meant, let out a metallic sound of satisfaction.

Amir looked again and saw that some of the other traces contained the red, faulty span. Some looked normal. For each span he touched, the O-Tel said what it was, how long it took to run, and what parts it used.

Amir was learning a lot about how Rusty worked. He could finally observe what was going on inside his friend.


splunk>



"Rusty has tried to walk at least 157 times since you brought me here. Both of his legs failed to move, though," the O-Tel commented. The lens could read Rusty's memory, which contained messages that bore the time when something happened inside the robot, what he did, and which components were involved. "Which of these branches has something to do with that?" Amir asked.

Suddenly, there it was: A small orange span intersected with the walking trace, flashing intermittently. "That is the energy pipeline that goes to your motion module!" Amir exclaimed. The O-Tel felt compelled to provide additional information. "Rusty has all his energy intact," he calmly stated. If Rusty still had energy, then something was blocking the flow.

Amir didn't wait any longer. He took a screwdriver, climbed over his friend's shoulders, and started poking at the main power line in Rusty's neck.

In a matter of seconds, he found a bolt nested between the metallic plates and the cables that carried energy and data to Rusty's brain. Amir dislodged the bolt with a swift movement.

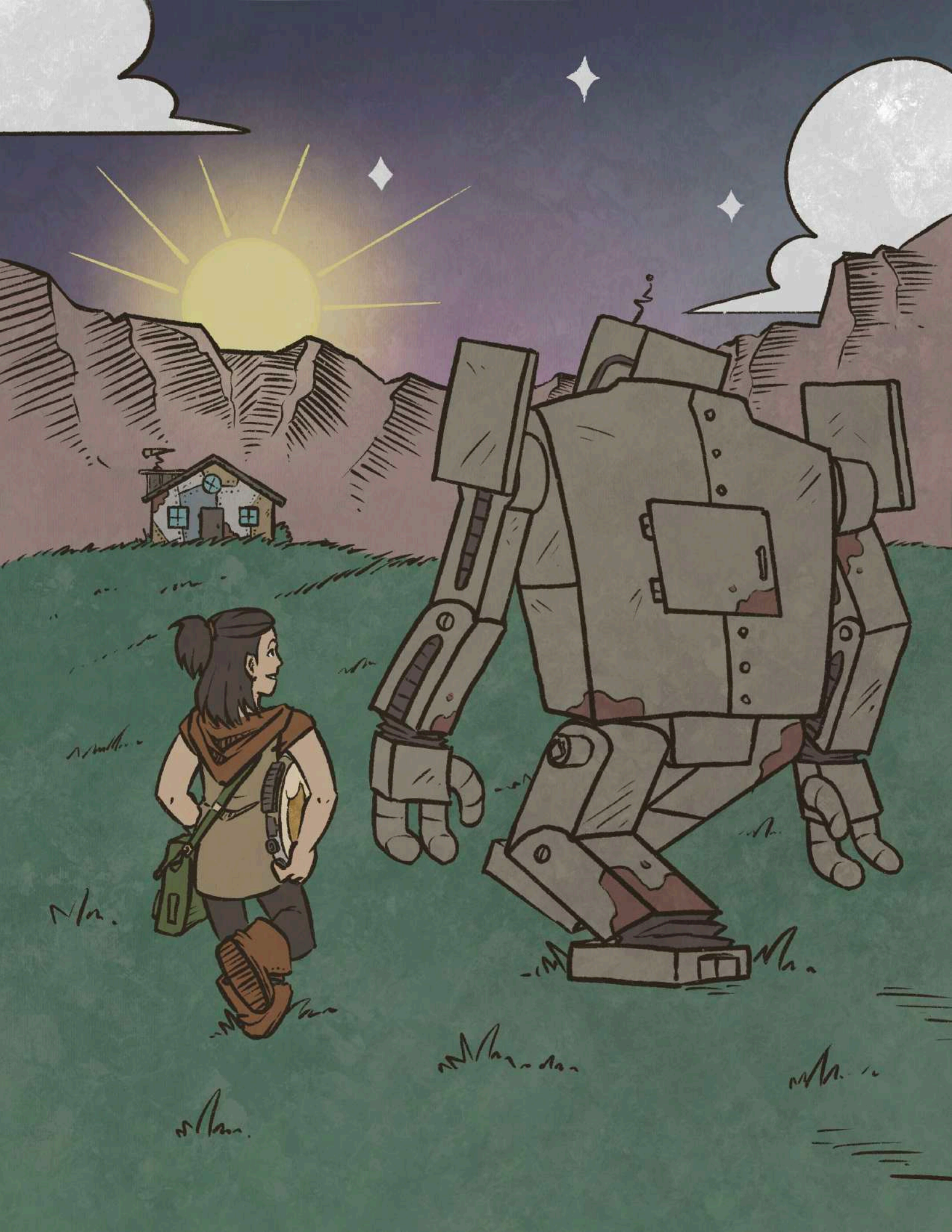
!!
<clunk>

Rusty flashed his eyes in surprise and stood up. He could walk at last. O-Tel, which was still projecting, politely coughed. "Look at the traces now," he said.

Amir glanced at the trees of light and saw how the ruby spans vanished and turned golden. All was well. "Energy levels have been fully restored," the O-Tel said with a hint of satisfaction.

Relieved, Amir and Rusty walked home, the boy pacing briskly next to his towering robot. They'd never feel worried again.

The End



The Story Behind the Lens

Rusty is a robot made of many disparate components, an artificial being that can accomplish lots of things on his own. Many of the software applications we use every day are also made of different components, much like Rusty, and they can do many wonderful things. Assembling different parts is what modern software developers do to build applications, because it's faster and more efficient.

When Rusty stopped moving, Amir couldn't quite figure out what was wrong. When big software applications fail, understanding what's causing issues is not straightforward, since the way each different part interacts with the others is difficult to see. With each component saying lots of things at the same time, peering into Rusty could have been like entering a room full of people yelling at each other.

Amir could have disassembled Rusty to solve the issue, but that would have harmed his friend. Instead, he thought of something better: He went back to his lab and created the O-Tel lens, which could speak of all the light it collected. For modern

software, the O-Tel lens of this story is called the OpenTelemetry framework, which everyone can use to see inside applications. That's what developers call "software observability."

OpenTelemetry, also known as OTel, is called that because it's open and because it collects telemetry data. "Open" means that anyone can contribute to his development, as its source code is open for all to see and comment. "Telemetry" is all kinds of data that OTel can collect from systems as complex and big as Rusty.

The three main types of telemetry data are logs, metrics and traces. Let's see what each of those contribute to the understanding of technical issues in software applications.

Logs

"Rusty has tried to walk at least 157 times since you brought me here. Both of his legs failed to move, though," the O-Tel commented.

Logs are messages or records that describe events that happened at a particular point in time. Think of them as entries in a diary. When O-Tel told Amir how many times Rusty had tried to walk, it was reading through all of Rusty's log entries, filtering out all that weren't related to the action of walking. Logs can be hard to search, but they provide a treasure trove of useful information when one knows how to read them.

Metrics

The O-Tel felt compelled to provide additional information: "Rusty has all his energy intact," it calmly stated. If Rusty still had energy, then something was blocking the flow.

Metrics are measurements made over time. When the O-Tel observed that Rusty still had all of his energy, it provided a metric that contained a value (100%) and the name of the thing being measured (energy). Metric data is useful to track the health of a system over time, and can be used to trigger alerts in observability backends (more on this in a minute). Even in the real world we're surrounded by metrics: for example, speedometers in cars, or pressure gauges in pipelines.

Traces

Rusty didn't move, but light started projecting in front of him: The O-Tel had cast a dozen different traces onto the sandy ground. Each was several meters long, branching like a tree of golden light.

Traces are the core of observability data: They represent the journey of a request through a system. Tracing is the action of creating representations of requests and their journeys: When the O-Tel projected light onto the ground, it was tracing all that was happening inside of Rusty. In that case it was "distributed tracing," because the requests for walking went through several parts of Rusty, like his brain and heart.

Amir looked again and saw that some of the other traces connected to the red, faulty span. Some looked normal. For each span he touched, the O-Tel said what it was, how long it took to run, and what parts it used.

Each branch of the tree of light is a span. Traces are defined by their spans, which are the smaller branches connected to the big ones. Spans can be short or long, depending on the duration of each action inside the system. What about the red spans, though? How did the O-Tel know that that particular branch was faulty?

Context propagation, resources and baggage

The magic of OpenTelemetry comes from its ability to connect different data sources, a mechanism called context propagation. Each span carries an identifier (a signature) that can be used to connect causes and effects. This data is complemented by the baggage, which, as the name implies, is a bunch of name-value pairs, such as "leg:right." Finally, spans can carry resource attributes, such as the environment in which code is running, or the name of the application–in this case, "Rusty."

Instrumentation

How is all this data collected? OpenTelemetry defines the format of the data and how all components talk to each other in a specification, which is like a book of rules. That only tells how to build the "lens," though, and how the lens should cast the rays of light. How is data collected?

> *Amir heartily laughed and jumped to the back of his robot. He opened the rear hatch and kindly plugged the O-Tel in one of the sockets next to Rusty's big heart, instrumenting it. The lens started glowing with the light of all the signals it collected, which would have been otherwise invisible.*

What Amir does when plugging the O-Tel next to Rusty's heart is to instrument it. OpenTelemetry does something very

similar when developers instrument their software with it: To let the code "speak" to OTel, one must connect to it using SDKs, which are kits that allow code to be plugged into other code in a way that doesn't break applications.

Once the application is instrumented, OpenTelemetry collects telemetry using the Collector, which is a special listener that quietly sits in the system, grabbing all the "light" that comes to it. In the story, that Collector is the inner face of the lens. Data is then "exported" or sent away (the outer face of the lens). Where is data sent, though?

Observability backend

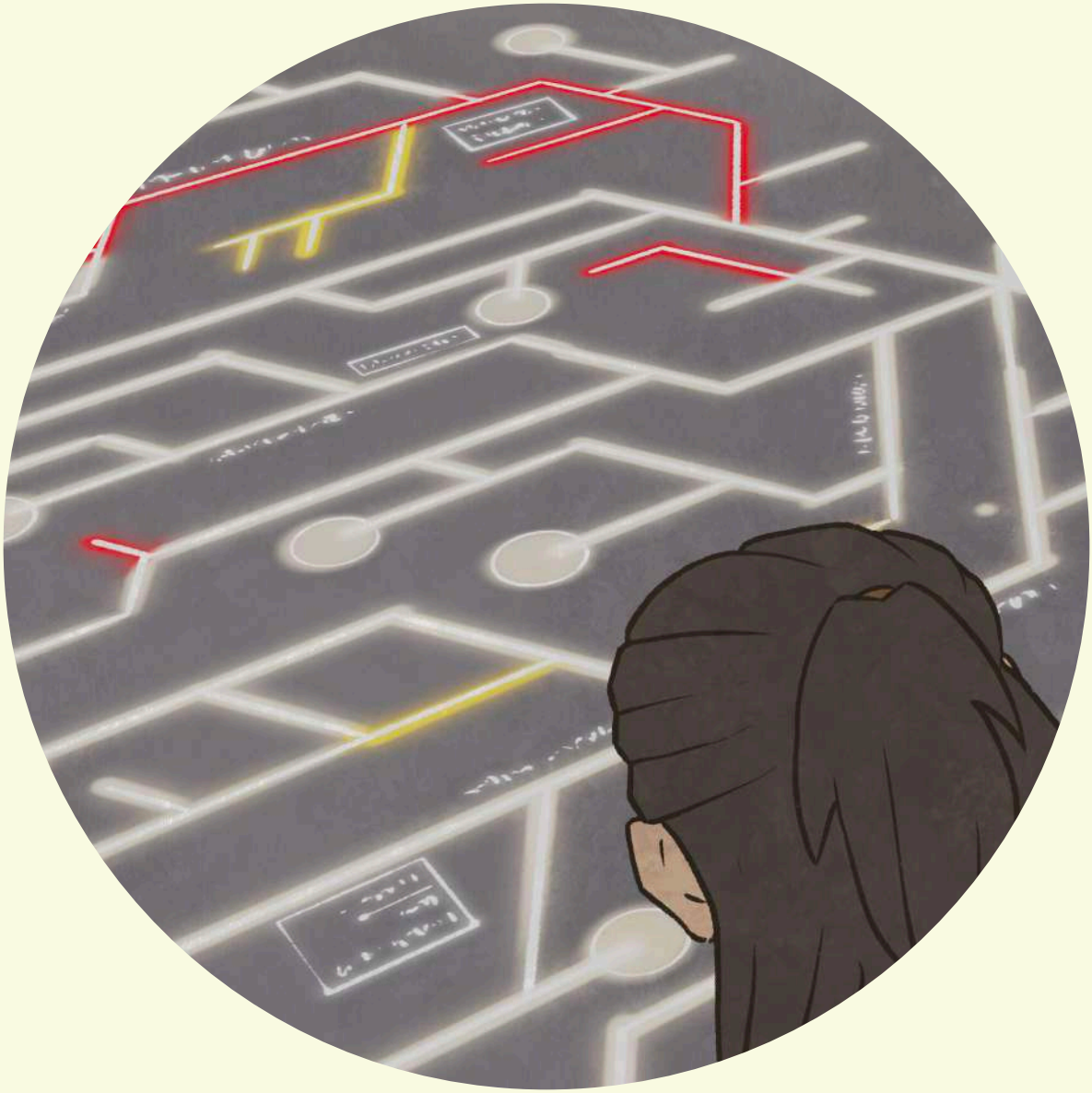
The web of traces, vast and complex, was distorted by the numerous bushes and rocks.

Analyzing all the data exported by the OpenTelemetry Collector can be a daunting task. What to make of all the spans, metrics and logs? How can one extract sensible information from all the collected data? The answer lies in the observability backend, which is an application that can visualize and process software telemetry.

"We need a smoother surface," Amir decided. He pulled a blanket from his backpack and laid it on the ground. The rays of light shed by the O-Tel, previously raw and broken, became sharper and easier to follow when cast onto the dark, wooly background.

For this purpose, OpenTelemetry not only collects data, but also exports it in different formats, depending on the target backend. Some backends are open source, such as Jaeger or Prometheus. Other backends are commercial, and include features that the open source backends may lack. OpenTelemetry is

not picky in this sense, and can export data to many different backends, including Splunk Observability Cloud.



A gentle introduction to OpenTelemetry and software observability in the DevOps era.

To learn more, read
5 Reasons Managers
Choose OpenTelemetry.

splunk>®