LEVEL 52

THE FINAL BATTLE

Speedrun

- The final battle has arrived. Use these challenges to prove what you have learned!

This level contains no new material—only a set of challenges that, when combined, form the most significant program you will create in this book: The Final Battle. This is not an easy challenge; even a veteran developer would recognize it as something beyond a mere toy or exercise. Expect it to take some time, but you will come away with something substantial to show for it. (And as you work on it, think of how far you've come! Remember Hello World!) As always, you can find my solution for each of these challenges on the book's website, so if you are struggling or just want to compare answers, take a look at my solution. It is okay if your solution is wildly different from mine. Any code that does the described job is a win.

In this level, you will build a turn-based battle game, like the combat system of many turn-based RPG games, where human and computer players control two parties of characters as they use attacks, items, and equipment to try to defeat their foes. We'll get into the details soon.

There are a total of 18 challenges in this level, but you are not expected to do all of them. They are split into nine core challenges and nine expansion challenges. You have a choice before you. You can take either the Core Path or the Expansion Path:

- The Core Path: Complete the nine core challenges and two of the expansion challenges of your choice.
- The Expansion Path: Retrieve my solution to the core challenges and use it as a starting point to complete seven of the expansion challenges of your choice.

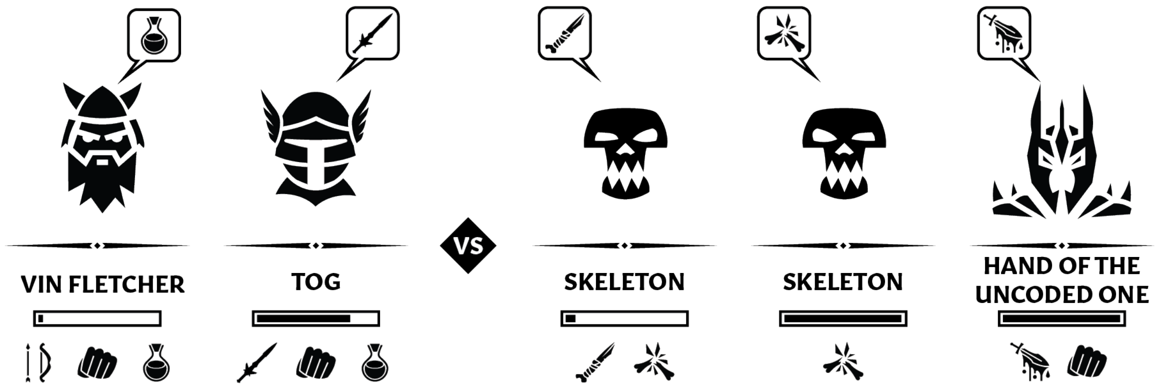
Depending on your interests and skills, you may prefer one path over the other. (Or do all 18 if you want to spend the time and effort.)

The Expansion Path may be more suitable if you feel unsure about object-oriented design, as my version of the core challenges establishes a framework to build on. It comes at a cost: you have to figure out how my code works, and understanding existing code is its own challenge.

Narrative **The Uncoded One**

The final battle has arrived. On a volcanic island, enshrouded in a cloud of ash, you have reached the lair of the Uncoded One. You have prepared for this fight, and you will return Programming to the lands. Your allies have gathered to engage the Uncoded One's minions on the volcano's slopes while you (and maybe a companion or two) strike into the heart of the Uncoded One's lair to battle and destroy it. Only a True Programmer will be able to survive the encounter, defeat the Uncoded One, and escape!

Overview



Let's look at what we're trying to build in some detail. The game works this way:

The game is composed of two collections of *characters*, formed into one of two *parties*. Those parties are the *heroes* (the good guys) and the *monsters* (the bad guys). There are many different types of characters. For example, the core game will include several *skeleton* characters and a character type that is *The Uncoded One*—the final boss. On the heroes' side, we will start with a single *True Programmer* character, whose name will come from the player. This is the hero of the game. Expansions will add more character types.

Each character will have a certain number of hit points (HP). When a character's hit points reach 0, it will be removed from the battle. When a party has no more characters, that party loses the battle. We will start with just a single battle but will eventually have multiple battles in a row. With multiple battles, the heroes must defeat each set of monsters in turn. The monsters win if they can win any battle.

Each battle is a series of rounds in which every remaining character gets a turn to pick one of several actions. As we go, we will create new action types, each of which can be used by the character for their turn. In the core challenges, we will start with only two: *do nothing*, which skips the character's turn, and *attack*, which is the primary action type of the game, driving the game forward. Characters may eventually have more than one attack option, but we will keep things simple initially and give each character a single basic attack. Each attack type has a name and, when used, produces a set of data that represents the attack underway. For example, we will start the True Programmer with a simple *punch* attack that will deal a fixed, reliable, 1 point of damage when used. In contrast, a skeleton has a *bone crunch* attack that randomly deals 0 or 1 point of damage. By attacking their enemies, characters will eventually erode their opponents' HP down to 0, eliminating them from the battle.

A player controls each party. The player decides which action to use for each character. As we go, we will build a simple computer player (a basic AI) and then eventually a human-controlled player via the console window. We will be able to run the game with the computer controlling both parties, humans controlling both parties, and the human in charge of the heroes against the computer in charge of the monsters (the standard setup).

CORE CHALLENGES

The following nine challenges form the core of the game, building the system so that multiple characters in each of two parties can engage in turn-based combat to defeat each other. The expansion challenges flesh the game out with extra features, but this is where the foundation is built. If you are doing the Core Path, you will complete all of these challenges. If you are doing the Expansion Path, read through these challenges to know what to expect from your starting point, retrieve the code on the book's website, and move along to the expansion challenges.

Boss Battle **Core Game: Building Character** **300 XP**

This challenge is deceptively complex: it will require building out enough of the game's foundation to get two characters taking turns in a loop. (Sure, they'll be doing nothing, but that's still a big step forward!)

Objectives:

- The game needs to be able to represent characters with a name and able to take a turn. (We'll change this a little in the challenge *Characters, Actions, and Players*.)
- The game should be able to have skeleton characters with the name SKELETON.
- The game should be able to represent a party with a collection of characters.
- The game should be able to run a battle composed of two parties—heroes and monsters. A battle needs to run a series of rounds where each character in each party (heroes first) can take a turn.
- Before a character takes their turn, the game should report to the user whose turn it is. For example, "It is SKELETON's turn...."
- The only action the game needs to support at this point is the action of doing nothing (skipping a turn). This action is done by displaying text about doing nothing, resting, or skipping a turn in the console window. For example, "SKELETON did NOTHING."
- The game must run a battle with a single skeleton in both the hero and the monster party. At this point, the two skeletons should do nothing repeatedly. The game might look like the following:

```
It is SKELETON's turn...
SKELETON did NOTHING.

It is SKELETON's turn...
SKELETON did NOTHING.
...
```

- **Optional:** Put a blank line between each character's turn to differentiate one turn from another.
- **Optional:** At this point, the game will run automatically. Consider adding a **`Thread.Sleep(500);`** to slow the game down enough to allow the user to see what is happening over time.

Boss Battle Core Game: The True Programmer 100 XP

Our skeletons need a hero to fight. We'll do that by adding in the focal character of the game, the player character, which represents the player directly, called the True Programmer by in-game role. The player will choose the True Programmer's name.

Objectives:

- The game must represent a *True Programmer* character with a name supplied by the user.
- Before getting started, ask the user for a name for this character.
- The game should run a battle with the True Programmer in the hero party vs. a skeleton in the monster party. The game might look like the following this challenge:

```
It is TOG's turn...
TOG did NOTHING.

It is SKELETON's turn...
SKELETON did NOTHING.
...
```

Boss Battle Core Game: Actions and Players 300 XP

The previous two challenges have had the characters taking turns directly. But instead of characters deciding actions, the player controlling the character's team should pick the action for each character. Eventually, there will be several action types to choose from (do nothing, attack, use item, etc.). There will also be multiple player types (computer/AI and human input from the console window). A player is responsible for picking an action for each character in their party. The game should ask the *players* to choose the action rather than asking the *characters* to act for themselves.

For now, the only action type will be a *do-nothing* action, and the only player type will be a *computer player*.

This challenge does not demand that you add new externally visible capabilities but make any needed changes to allow the game to work as described above, with players choosing actions instead of characters. If you are confident your design already supports this, claim the XP now and move on.

Objectives:

- The game needs to be able to represent action types. Each action should be able to run when asked.
- The game needs to include a *do-nothing* action, which displays the same text as in previous challenges (for example, "SKELETON did NOTHING.")
- The game needs to be able to represent different player types. A player needs the ability to pick an action for a character they control, given the battle's current state.
- The game needs a sole player type: a computer player (a simple AI of sorts). For now, the computer player will simply pick the one available option: do nothing (and optionally wait a bit first with **`Thread.Sleep`**).
- The game must know which player controls each party to ask the correct player to pick each character's action. Set up the game to ask the player to select an action for each of their characters and then run the chosen action.
- Put a simple computer player in charge of each party.
- **Note:** To somebody watching, the end result of this challenge may look identical to before this challenge.

Boss Battle — Core Game: Attacks — 200 XP

In this challenge, we will extend our game by giving characters attacks and allowing players to pick an attack instead of doing nothing. We won't address tracking or dealing out damage yet.

Objectives:

- The game needs to be able to represent specific types of attacks. Attacks all have a name (and will have other capabilities later).
- Each character has a standard attack, but this varies from character to character. The True Programmer's (the player character's) attack is called *punch*. The Skeleton's attack is called *bone crunch*.
- The program must be capable of representing an *attack* action, our second action type. An attack action must represent which attack is being used and the target of the attack. When this action runs, it should state the attacker, the attack, and the target. For example: "TOG used PUNCH on SKELETON."
- Our computer player should pick an attack action instead of a do-nothing action. The attack action can be simple for now: always use the character's standard attack and always target the other party's first character. (If you want to choose a random target or some other logic, you can.)
- The game should now run more like this:

```
It is TOG's turn...
TOG used PUNCH on SKELETON.

It is SKELETON's turn...
SKELETON used BONE CRUNCH on TOG.
...
```

- **Hint:** The player will need access to the list of characters that are potential targets. In my case, I passed my **`Battle`** object (which represents the whole battle and had access to both parties and all their characters) to the player. I then added methods to **`Battle`** where I could give it a character, and it would return the character's party (**`GetPartyFor(Character)`**) or the opposing party (**`GetEnemyPartyFor(Character)`**).

Boss Battle — Core Game: Damage and HP — 150 XP

Now that our characters are attacking each other, it is time to let those attacks matter. In this challenge, we will enhance the game to give characters hit points (HP). Attacking should reduce the HP of the target down to 0 but not past it. Reaching 0 HP means death, which we will deal with in the next challenge.

Objectives:

- Characters should be able to track both their initial/maximum HP and their current HP. The True Programmer should have 25 HP, while skeletons should have 5.
- Attacks should be able to produce *attack data* for a specific use of the attack. For now, this is simply the amount of damage that they will deal this time, though keep in mind that other challenges will add more data to this, including things like the frequency of hitting or missing and damage types.
- The *punch* attack should deliver 1 point of damage every time, while the *bone crunch* attack should randomly deal 0 or 1. **Hint:** Remember that **`Random`** can be used to generate random numbers. **`random.Next(2)`** will generate a 0 or 1 with equal probability.

- The attack action should ask the attack to determine how much damage it caused this time and then reduce the target's HP by that amount. A character's HP should not be lowered below 0.
- The attack action should report how much damage the attack did and what the target's HP is now at. For example, "PUNCH dealt 1 damage to SKELETON." "SKELETON is now at 4/5 HP."
- When the game runs after this challenge, the output might look something like this:

```
It is TOG's turn...
TOG used PUNCH on SKELETON.
PUNCH dealt 1 damage to SKELETON.
SKELETON is now at 4/5 HP.

It is SKELETON's turn...
SKELETON used BONE CRUNCH on TOG.
BONE CRUNCH dealt 0 damage to TOG.
TOG is now at 25/25 HP.
...
```

Boss Battle Core Game: Death 150 XP

When a character's HP reaches 0, it has been defeated and should be removed from its party. If a party has no characters left, the battle is over.

Objectives:

- After an attack deals damage, if the target's HP has reached 0, remove them from the game.
- When you remove a character from the game, display text to illustrate this. For example, "SKELETON has been defeated!"
- Between rounds (or between character turns), the game should see if a party has no more living characters. If so, the battle (and the game) should end.
- After the battle is over, if the heroes won (there are still surviving characters in the hero party), then display a message stating that the heroes won, and the Uncoded One was defeated. If the monsters won, then display a message saying that the heroes lost and the Uncoded One's forces have prevailed.

Boss Battle Core Game: Battle Series 150 XP

The game runs as a series of battles, not just one. The heroes do not win until every battle has been won, while the monsters win if they can stop the heroes in any battle.

Objectives:

- There is one party of heroes but multiple parties of monsters. For now, build two monster parties: the first should have one skeleton. The second has two skeletons.
- Start a battle with the heroes and the first party of monsters. When the heroes win, advance to the next party of monsters. If the heroes lose a battle, end the game. If the monsters lose a battle, move to the next battle unless it is the last.

Boss Battle Core Game: The Uncoded One 100 XP

It is time to put the final boss into the game: The Uncoded One itself. We will add this in as a third battle.

Objectives:

- Define a new type of monster: *The Uncoded One*. It should have 15 HP and an *unraveling* attack that randomly deals between 0 and 2 damage when used. (The Uncoded One ought to have more HP than the True Programmer, but much more than 15 HP means the Uncoded One wins every time. We can adjust these numbers later.)
- Add a third battle to the series that contains the Uncoded One.

Boss Battle **Core Game: The Player Decides** **200 XP**

We have one critical missing piece to add before our core game is done: letting a human play it.

Objectives:

- The game should allow a human player to play it by retrieving their action choices through the console window. For a human-controlled character, the human can use that character's standard attack or do nothing. It is acceptable for all attacks selected by the human to target the first (or random) target without allowing the player to pick one specifically. (You can let the player pick if you want, but it is not required.) The following is one possible approach:

```
It is TOG's turn...
1 – Standard Attack (PUNCH)
2 – Do Nothing
What do you want to do? 2
```

- As the game is starting, allow the user to choose from the three following gameplay modes: player vs. computer (the human in charge of the heroes and the computer controlling the monsters), computer vs. computer (a computer player running each team, as we have done so far), and human vs. human, where a human picks actions for both sides.
- **Hint:** There are many ways you could approach this. My solution was to build a **`MenuItem`** record that held information about options in the menu. It included the properties **`string Description`**, **`bool IsEnabled`**, and **`IAction ActionToPerform`**. **`IAction`** is my interface representing any of the action types, with implementations like **`DoNothingAction`** and **`AttackAction`**. I have a method that creates the list of menu items, and it produces a new **`MenuItem`** instance for each choice. The code that draws the menu iterates through the **`MenuItem`** instances and displays them with a number. After getting the number, I find the right **`MenuItem`**, extract the **`IAction`**, and return it. That means I create many **`IAction`** objects that don't get used, but it is a system that is easy to extend in future challenges.

Expansions

The following challenges are expansion challenges. If you do the Core Path, you only need to complete two of the following challenges of your choice. If you choose to do the Expansion Path, you will start by acquiring my solution to the core challenges and then complete seven of the following challenges. In both cases, note the *Making It Yours* challenge, which is repeatable, and the *Restoring Balance* challenge. They are both worthwhile.

Boss Battle **Expansion: The Game's Status** **100 XP**

This challenge gives us a clearer representation of the status of the game.

Objectives:

- Before a character gets their turn, display the overall status of the battle. This status must include all characters in both parties with their current and total HP.
- You must also somehow distinguish the character whose turn it is from the others with color, a text marker, or something similar.
- **Note:** You have flexibility in how you approach this challenge, but the following shows one possibility (with the current character colored yellow instead of white):

```
============================================ BATTLE ============================================
TOG                                   ( 25/25 )
---------------------------------------------- VS ----------------------------------------------
                                                                              SKELETON (  5/5  )
                                                                              SKELETON (  5/5  )
================================================================================================
```

Boss Battle Expansion: Items 200 XP

Each party has a shared inventory of items. Players can choose to use an item as an action. We will add a health potion item that players can use as an action.

Objectives:

- The game must support adding consumable items with the ability to use one as an action. Item types could be potentially very broad (keep that in mind when choosing your design), but all that is required now is a health potion item type. Items are usable, and when used, the reaction depends on the item type.
- A health potion is the only item type we need to add here. It should increase the user's HP by 10 points when used. In doing so, the HP should not rise above the character's maximum HP.
- The entire party shares inventory. Ensure parties can hold a collection of items.
- Start the hero party with three health potions. Give each monster party one health potion.
- The game must support the inclusion of a *use item* action, along with the item to use. When this action runs, it should cause the item's effect to occur and remove the item from the inventory.
- The computer player should consider using a potion when (a) the team has a potion in their inventory and (b) the character's health is under half. Under these conditions, use a potion 25% of the time.
- The console player should have the option to use a potion if the party has one.
- **Note:** Digging through the party inventory for potions is a little tricky. It is reasonable to assume (for now) that all items in the inventory are health potions. That will be a correct assumption until you make other item types. This assumption simplifies the changes you need to make to the different player types.

Boss Battle Expansion: Gear 300 XP

Characters can equip gear that allows them to have a second special attack. A party can have gear in their shared inventory, but unlike items, gear is not usable until a character equips it, and it takes a turn to equip gear from inventory.

Objectives:

- The game must support the concept of gear. All gear has a name and an attack they provide.
- Each character can equip one piece of gear.
- Each party also has a collection of unequipped gear.

- Add the ability to perform an *equip* action, which knows what gear is being equipped. When this action runs, it should move the gear from the party's inventory to the character.
- If a character already has something equipped, the previously equipped gear should be unequipped and moved back to the party's shared inventory.
- The computer player should equip gear. If a character has nothing equipped but the party has equipable gear, the computer player should choose to equip the gear 50% of the time.
- **Note:** If you also did the Items challenge, using potions should be a priority over equipping gear.
- The console player should also have the option to equip gear. If there is more than one thing to equip, allow the player to choose from all available options.
- The computer player should prefer the attack provided by equipped gear when one is available. Gear-based attacks are typically stronger.
- If gear is equipped, the human player should be able to pick either the standard attack or the gear-based attack.
- The True Programmer character should start the game with a *sword* item equipped. The sword should have a *slash* attack that deals two damage.
- Create a *dagger* with the attack *stab* that reliably deals 1 point of damage.
- Start the first battle's skeleton with a dagger equipped. This one was prepared for battle.
- Put two daggers in the team inventory for the second battle. Both skeletons will be able to use a dagger, but they will have to equip it first. These two were less prepared.
- **Optional:** If you did *The Game's Status*, consider showing what gear each character has equipped.

Boss Battle — Expansion: Stolen Inventory — 200 XP

Requires that you have done either *Items* or *Gear*.

This feature will allow us to have a basic loot system. When a character dies, the opposing side should immediately recover the dead character's equipped gear. When the monster party is eliminated, the monster party's unequipped gear and items should be transferred to the hero party.

If you only did *Items* or *Gear*, some of the objectives below will not apply to you. Do the ones that apply.

Objectives:

- If you did the *Items* challenge, when a party is eliminated, transfer all items from the losing party's inventory items to the winning party. Display a message that indicates which items have been acquired. **Note:** It is okay only to do this when the hero party wins. When the monster party wins a battle, the game ends.
- If you did the *Gear* challenge, when a party is eliminated, transfer all unequipped gear from the losing party's inventory to the winning party. Display a message that indicates when gear has been acquired.
- If you did the *Gear* challenge, when a character is eliminated, transfer any equipped gear to the winning party's inventory. Display a message that states gear that was acquired.

Boss Battle — Expansion: Vin Fletcher — 200 XP

The True Programmer does not have to fight the Uncoded One alone! The hero party can have other heroes (companions) that should each get their turn to fight. In this challenge, we will add our favorite arrow maker, Vin Fletcher, to the game. This challenge will also add in the possibility for an attack to sometimes miss.

Objectives:

- When an attack generates attack data, it must also include a probability of success. 0 is guaranteed failure, 1 is guaranteed success, 0.5 is 50/50, etc.
- Modify your attack action to account for the possibility of missing. If an attack misses, don't damage the target and instead report that the attack missed. For example, "VIN FLETCHER MISSED!"
- Create a new character type to represent Vin Fletcher. He starts with 15 HP. If you did the *Gear* challenge, Vin should have the same standard *punch* attack the True Programmer has and equip him with a *Vin's Bow* gear with an attack called *quick shot* that deals 3 damage but only succeeds 50% of the time. If you did not do the *Gear* challenge, give Vin *quick shot* as his standard attack.

Boss Battle — Expansion: Attack Modifiers — 200 XP

An *attack modifier* is a character attribute that adjusts attacks involving them. We will make only defensive attack modifiers for this challenge, which apply when a character is on the receiving end of an attack. You can add offensive attack modifiers as a part of the *Making It Yours* challenge. Attack modifiers are applied when an attack action is being resolved. An attack modifier takes the current attack data as input and produces new, potentially altered attack data. For example, it could reduce the damage done (a resistance to the attack) or increase it (a weakness to the attack). This challenge will add a new monster type with resistance to all attacks, reducing incoming damage by 1.

Objectives:

- The game must be able to represent attack modifiers. Attack modifiers take attack data as an input and produce new attack data as a result, possibly tweaking the attack data in the process. Attack modifiers also have names.
- All characters should be able to have a defensive attack modifier.
- When performing an attack, see if the target character has a defensive attack modifier before delivering damage to the target. If it does, allow the modifier to manipulate the attack data and use the modified results instead.
- When attack modifiers adjust damage, they should report that they changed the damage and by how much. For example, "STONE ARMOR reduced the attack by 1 point."
- The game must support a new *stone amarok* character. Stone amaroks have 4 HP and a standard *bite* attack that deals 1 damage. They also have a defensive attack modifier called *stone armor* that reduces all attacks by 1 damage. If your heroes still only have a 1-point *punch* attack, change something so that the heroes can survive an encounter with stone amaroks.
- Add a battle that includes two stone amaroks as monsters.

Boss Battle — Expansion: Damage Types — 200 XP

Requires that you have done *Attack Modifiers*.

Attacks should have a damage type that may affect how it works. For now, our damage types must include at least *normal* and *decoding*, but you can add additional damage types (such as *fire* or *ice*) if you want. The damage type primarily adds flavor to the game but also leads to additional game mechanics. For example, we will give the True Programmer a defensive attack modifier called *Object Sight*, which gives the character resistance to *decoding* damage.

Objectives:

- Attack data should include a damage type. You are free to define any damage types that you think make sense, but include the damage types of *normal* (or similar) and *decoding*. **Hint:** An enumeration might be a good choice for this.
- Have each attack use one of these types when producing damage data. Use whatever damage types you wish, but make The Uncoded One's *unraveling* attack be *decoding* damage.
- Give the True Programmer character a defensive attack modifier called *Object Sight* that reduces *decoding* damage by 2.
- Increase the *unraveling* attack to deal 0 to 4 damage randomly, instead of 0 to 2, ensuring that this attack is the single most powerful attack in the game (but one that the hero has resistance to).

Boss Battle — Expansion: Making it Yours — 50 to 400 XP

This is perhaps the most exciting challenge: the one where you get to do whatever you want to the game.

This challenge is also repeatable. If you'd rather make four of your own enhancements instead of the other challenges listed above, go for it.

Use your best judgment when awarding yourself XP, comparing what you've chosen with the other challenges here.

The possibilities are limitless, but here are some ideas:

- More heroes. You could add Mylara and Skorin, whose gear (or standard attack) is the Cannon of Consolas, which deals 1 damage most of the time, 2 damage every multiple of 3 or 5, and 5 damage every multiple of 3 and 5. (This adds some continuity from the early part of the book!)
- Add more item types. Maybe Simula's soup is a full heal, restoring HP to its maximum.
- Add offensive attack modifiers.
- Let characters equip more than one piece of gear.
- Let gear provide offensive and defensive attack modifiers. (A Binary Helm that reduces all damage done by 1 when equipped, for example.)
- Experience Points. As heroes defeat monsters, give them XP to track their accomplishments.
- More monster types.
- Add attack side effects, which allow an attack to run other logic when an attack happens. Maybe a Shadow Octopoid with a *grapple* attack that has a chance of stealing equipped gear.
- Load levels from a file instead of hardcoding them in the game.
- Display a small indicator in the status display that gives you a warning if a character is close to death. Perhaps a yellow `[!]` if the character is under 25% and a red `[!]` if the character is under 10%.

- Allow characters to taunt their enemies with messages like the Uncoded One saying <<THE UNRAVELLING OF ALL THINGS IS INEVITABLE>> and a skeleton saying, "We will repel your spineless assault!"
- Allow characters to have temporary effects applied to them. For example, a poison effect that deals 1 damage per turn for three turns.
- Strip out all *C# Player's Guide*-specific elements and re-theme the game into your own creation.

Boss Battle — Expansion: Restoring Balance — 150 XP

Depending on the challenges you completed, the game is likely not very balanced anymore. Either the heroes or the monsters win all the time. Without changing logic and mechanics, adjust things like potion counts, attack strengths and probabilities, the number of battles and monsters they contain to produce a version of the game where if you play wisely, you are likely to win, but if you play poorly, you will lose.

Objectives:

- Tweak aspects of the game (no need to write new logic, just change parameters and character counts, etc.) until you have something challenging and fun.
- Do what you can to ensure that The Uncoded One feels formidable.

Narrative — The True Programmer

You slice your sword through the smoky form of the Uncoded One for the last time. It begins to disintegrate, binary streams flowing out of it until it bursts apart in a dazzling blue light. You have done it. You have defeated the Uncoded One.

You step out of the cave's entrance and back onto the slopes of the volcano as ash falls like snow from the sky. Your allies have turned the tide of the battle against the Uncoded One's minions, and they surge through their final ranks as the remnants begin to scatter in retreat. You and your allies have saved these realms from the grasp of the Uncoded One.

As the sun sets, alighting the ash cloud in a crimson glow, you meet up with your old friends to celebrate with a feast. They're all there. Vin Fletcher with his arrows. Simula with her soups. Mylara and Skorin, with an improved cannon that helped break the Uncoded One's lines during the battle. Everyone else that you met as you voyaged through the Realms.

As the feast begins to wind down—and after you have had three full bowls of stew—Simula speaks to you. "You have shown that you are, without question, a True Programmer. To me. To us. To the Uncoded One. The power of Programming can return to these islands once again. But tell me, brave Programmer, what will you do next?"

You take a slow, deep breath as you ponder the question before you respond.

LEVEL 53

INTO LANDS UNCHARTED

Speedrun

- Your next step might be to dive into an app model like smartphone development, web development, desktop development, or game development.
- There is always more to learn. It's a journey. Keep taking steps.
- The only way to get better at programming is to program.
- We've reached the end, but the adventure is just beginning: press the Start button!

We've reached the end of our journey together. It is not *the* end but *an* end. And also a beginning. Programming is many things. Fun. Hard. Strange. But above all, it is a lifelong path of learning and growing. Perhaps it is this very thing that makes it fun, challenging, and worthwhile.

So while our time is running out, your time programming is just getting underway. As we part ways, let me give you some thoughts on where your journey may go next.

KEEP LEARNING

Other Frameworks and Libraries

As you finish this book, you have a solid understanding of the C# language. If you've gone through this book cover to cover, you already know much about the C# language.

You will continue learning about the many things contained in the Base Class Library. Don't be afraid to search the documentation to discover what is there and learn more about it: **https://docs.microsoft.com/en-us/dotnet/api/**.

You are also more than ready to dive into an app model and start building more sophisticated apps than console applications. That could be web, desktop, mobile, or game development (or anything else!). Level 50 covered each of these briefly, but the following link to an article on this book's website could help you figure out that next step: **csharpplayersguide.com/c-sharp/articles/app-model-recommendations**

Don't forget that nuget.org can provide you with packages that do all sorts of nifty things. Don't reinvent the wheel; check if somebody else has already solved some of your problems for you.

Other Topics

There is much more to building software than just knowing how to write lines of code. Making software is so much more than "coding." I can't truly capture the breadth and depth of topics that you might find helpful, but a few worthwhile topics are data structures and algorithms, software engineering, version control, Agile and Scrum, and unit testing.

Don't be afraid to learn other languages either. I think you'll find that C# is a fantastic language, but knowing others can help you see your problems in a different light and make you a better C# programmer.

Here is a small list of some of my favorite books that I'd recommend to a budding programmer:

- *Clean Code*, Robert C. Martin. This book talks about making your code as readable and maintainable as possible.
- *Clean Coder*, Robert C. Martin. This book doesn't focus on code, but on you, as a programmer. Its subtitle is "A Code of Conduct for Professional Programmers."
- *The Pragmatic Programmer*, Andrew Hunt and David Thomas. This book is full of advice and lessons that the authors learned the hard way so that you don't have to.

Make Software

Reading stuff is great, but building software is where you discover how much you know and what gaps you might have. Hopefully, you have done at least some of the Challenge problems in this book, but if not, start there.

Don't pick problems or projects that are too easy or too hard. If they're too easy, you get bored and don't learn as much. If they're too hard, they tend to overwhelm you. (And everything is a bit harder than they initially appear.)

Do the Side Quests

If you have skipped the side quest sections, make a plan for returning to them. It could be right now or in two weeks or two months when you have more C# programming experience behind you. Decide now and put it into your calendar to remind yourself!

Where Do I Go to Get Help?

The best part about making software is that you're creating something that has never been done before. It is as much a creative process as it is mathematical or analytical. You're not just manufacturing the same thing over and over again. That means you will run into problems that have never been encountered before. Let's talk about a few strategies for dealing with these new and confusing situations.

First, a Google search will go a very long way. That's always a good starting point.

Second, make sure you fully understand what the code you are working with is doing. If you're having trouble with a class that someone else created (like `List`, for instance) and a method isn't doing what you thought it would, then take the time to make sure you understand all of the parts involved. If you're running into a problem with code you don't quite understand, you're not likely to figure the whole situation out until you've figured out how to use all of the individual pieces that are in play. Learn what each parameter in a method does. Learn what exceptions it might throw. Figure out what all of its properties do. Once you truly understand the things you're working on, the solution is often self-evident.

Along those lines, the Microsoft Documentation website has tons of details about every type that the Base Class Library has, with plenty of examples of using them. It's a trove of information. (**https://docs.microsoft.com/en-us/dotnet/api/**)

If you're stuck and have team members who might know the answer, talking to them is always a good idea. They'll be happy you're asking questions and trying to learn.

I'd be remiss if I didn't mention Stack Overflow (**http://stackoverflow.com/**). Stack Overflow is the best site out there for programming Q&A. It covers everything from the basics to very specific, very advanced questions, and they all have good, intelligent answers. They are picky about their rules, so read and follow them before posting a question.

Parting Words

The end credits for the original *Legend of Zelda* are:

Thanks Link, you're the hero of Hyrule.
Finally, peace returns to Hyrule.
This ends the story.
Another quest will start from here.
Press the Start button.

You've built a solid C# foundation, and now it's time to take the next step. From one programmer to another, I wish you the best of luck as you continue your journey and make fantastic software.

This ends the story. Another quest will start from here. Press the Start button!