**Master PDF**

Things that are **bolded and underlined** are the title for a file, all code below it (but before the next title) belonged to the file.

These files were inside a seperate folder titled “components” that was still used by **main.py:**
**ai.py**
**base_component.py**
**consumable.py**
**equipment.py**
**equippable.py**
**fighter.py**
**inventory.py**
**level.py**

**Credits.txt:**

RogueFishing Credits:

Game made by me using original code as well as imports and code snippets from:

Python-TCOD:
Copyright (c) 2009-2023, Kyle Benesch and the python-tcod contributors.
All rights reserved. Python-TCOD's Github page can be found at https://github.com/libtcod/python-tcod. Documentation for Python-TCOD can be found at https://python-tcod.readthedocs.io/en/latest/#. Python-tcod is distributed under the Simplified 2-clause FreeBSD license.
BSD 2-Clause License

Roguelike Tutorials:
Most of the framework for RogueFishing was created using this tutorial, including code taken directly from it. The website for Roguelike tutorials can be found at https://rogueliketutorials.com. Roguelike tutorials was created by Tyler Standridge and is licensed under Creative Commons Legal Code CC0 1.0 Universal

Home screen "New Piskel.png" by me

Font sheet "dejavu10x10_gs_tc.png" is made by the python-tcod developers with many changes by me. "dejavu10x10_gs_tc.png" is Liscensed under BSD 2-Clause License, as is Python-TCOD.

**main.py:**

```
#RogueFishing
#Started 3/5/25
#written with python v 3.13.1 and tcod (a roguelike engine) v 16.2.3

#detailed credits, including copyright information, can be found in Credits.txt

#comments noted by """ are also not by me.
#my comments are (for the most point)above lines!
#for files with lots of comments by the original creator (which is not all of them), I will denote their comments with ##

#Home screen "New Piskel.png" by me
#Font sheet "dejavu10x10_gs_tc.png" is made by the python-tcod developers with changes by me.
```

```
#credits, detailed credits can be found at credits.txt:
#general TCOD reference: https://python-tcod.readthedocs.io/en/latest/index.html
#Used code by "Roguelike Tutorials" created by Tyler Standridge, website found at
roguelike tutorials.com with addendums/modifications by me

#this is the master python file for roguefishing



#Imports-----------

#Used code from "Roguelike Tutorials" created by Tyler Standridge, website found at
roguelike tutorials.com with addendums/modifications by me

import traceback
import tcod
import color
import exceptions
import input_handlers
import setup_game


#this function manages savefiles and calls them from engine.py
def save_game(handler: input_handlers.BaseEventHandler, filename: str) -> None:
    """If the current event handler has an active Engine then save it."""
    if isinstance(handler, input_handlers.EventHandler):
        handler.engine.save_as(filename)
        print("Game saved.")

def main() -> None:
    #sets screen size
    screen_width = 80
    screen_height = 50


    #load tiles from tileset#
    tileset = tcod.tileset.load_tilesheet(
        "dejavu10x10_gs_tc.png", 32, 8, tcod.tileset.CHARMAP_TCOD
    )

    handler: input_handlers.BaseEventHandler = setup_game.MainMenu()

    #set custom tileset font and setup some windo info/create the screen
    with tcod.context.new_terminal(
        screen_width,
        screen_height,
        tileset=tileset,
        title="RogueFishing",
        vsync=True,
    ) as context:
        root_console = tcod.console.Console(screen_width, screen_height, order="F")

        try:
            while True:
                root_console.clear()
                handler.on_render(console=root_console)
                context.present(root_console)

                try:
                    for event in tcod.event.wait():
                        context.convert_event(event)
                        handler = handler.handle_events(event)
                except Exception:  # Handle exceptions in game.
                    traceback.print_exc()  # Print error to stderr.
                    # Then print the error to the message log.
                    if isinstance(handler, input_handlers.EventHandler):
                        handler.engine.message_log.add_message(
                            traceback.format_exc(), color.error
                        )
        except exceptions.QuitWithoutSaving:
            raise
        except SystemExit:  # Save and quit.
            save_game(handler, "savegame.sav")
            raise
        except BaseException:  # Save on any other unexpected exception.
            save_game(handler, "savegame.sav")
            raise


if __name__ == "__main__":
    main()

#end of used code---------

#end of line
```

**<u>tile_types.py:</u>**

```
#RogueFishing
#tile types file

#This file is responsible for creating and managing tiles and all tile related elements
```

```
#credits:
#general TCOD reference: https://python-tcod.readthedocs.io/en/latest/index.html
#roguelike tutorial: rogueliketutorials.com

#Used code by "Roguelike Tutorials", website found at rogueliketutorials.com with slight
addendums/modifications by me

#imports
from typing import Tuple

import numpy as np  # type: ignore

### IN THIS SECTION, NOT ALL COMMENTS ARE WRITTEN BY ME ###
#this is essentially setting up tiles and tile properties!

# Tile graphics structured type compatible with Console.tiles_rgb.
graphic_dt = np.dtype(
    [
        ("ch", np.int32),  # Unicode codepoint.
        ("fg", "3B"),  # 3 unsigned bytes, for RGB colors.
        ("bg", "3B"),
    ]
)

# Tile struct used for statically defined tile data.
tile_dt = np.dtype(
    [
        ("walkable", np.bool),  # True if this tile can be walked over.
        ("transparent", np.bool),  # True if this tile doesn't block FOV.
        ("dark", graphic_dt),  # Graphics for when this tile is not in FOV.
        ("light", graphic_dt),  # Graphics for when the tile is in FOV.
    ]
)


def new_tile(
    *,  # Enforce the use of keywords, so that parameter order doesn't matter.
    walkable: int,
    transparent: int,
    dark: Tuple[int, Tuple[int, int, int], Tuple[int, int, int]],
    light: Tuple[int, Tuple[int, int, int], Tuple[int, int, int]],
) -> np.ndarray:
    """Helper function for defining individual tile types """
    return np.array((walkable, transparent, dark, light), dtype=tile_dt)

# SHROUD represents unexplored, unseen tiles
```

```
SHROUD = np.array((ord(" "), (255, 255, 255), (0, 0, 0)), dtype=graphic_dt)

#floor tile
floor = new_tile(
    walkable=True,
    transparent=True,
    dark=(ord("%"), (101, 67, 33), (0, 0, 0)),
    light=(ord("%"), (150, 75, 0), (0, 0, 0)),
)
#wall tile
wall = new_tile(
    walkable=False,
    transparent=False,
    dark=(ord("#"), (100, 100, 100), (0, 0, 0)),
    light=(ord("#"), (200, 200, 200), (0, 0, 0)),
)
#cave passage tile
up_passage = new_tile(
    walkable=True,
    transparent=True,
    dark=(ord(">"), (100, 100, 100), (0, 0, 0)),
    light=(ord(">"), (200, 200, 200), (0, 0, 0)),
)
```

**setup_game.py:**

```
#game setup file

#This file acts as a game initialization/setup screen. It basically is the main menu.

#credits:
#general TCOD reference: https://python-tcod.readthedocs.io/en/latest/index.html
#roguelike tutorial: rogueliketutorials.com

#Used code from "Roguelike Tutorials" created by Tyler Standridge, website found at rogueliketutorials.com with addendums/modifications by me

#imports

"""Handle the loading and initialization of game sessions."""
from __future__ import annotations

import copy
import lzma
import pickle
import traceback
from typing import Optional
```

```
import tcod
import color
from engine import Engine
import entity_factories
from game_map import GameWorld
import input_handlers
import random


# Load the background image and remove the alpha channel.
background_image = tcod.image.load("New Piskel.png")[:, :, :3]

#This is now where the game is configured!!!!!!
#moved from main
def new_game() -> Engine:
    """Return a brand new game session as an Engine instance."""
    map_width = 80
    map_height = 43

    room_max_size = 12
    room_min_size = 2
    max_rooms = 45


    player = copy.deepcopy(entity_factories.player)

    engine = Engine(player=player)

    engine.game_world = GameWorld(
        engine=engine,
        max_rooms=max_rooms,
        room_min_size=room_min_size,
        room_max_size=room_max_size,
        map_width=map_width,
        map_height=map_height,

    )
    engine.game_world.generate_floor()
    engine.update_fov()
#randomize this
    engine.message_log.add_message(
        "You regain consiousness in the caves. Again. Maybe this time you'll reach the
surface.", color.welcome_text
    )
#autoequip weapons
    Shortsword = copy.deepcopy(entity_factories.Shortsword)
    Gambeson = copy.deepcopy(entity_factories.Gambeson)
    Shortsword.parent = player.inventory
    Gambeson.parent = player.inventory

    player.inventory.items.append(Shortsword)
    player.equipment.toggle_equip(Shortsword, add_message=False)

    player.inventory.items.append(Gambeson)
    player.equipment.toggle_equip(Gambeson, add_message=False)
    return engine

#load function
def load_game(filename: str) -> Engine:
    """Load an Engine instance from a file."""
    with open(filename, "rb") as f:
        engine = pickle.loads(lzma.decompress(f.read()))
        assert isinstance(engine, Engine)
    return engine


class MainMenu(input_handlers.BaseEventHandler):
    """Handle the main menu rendering and input."""

    def on_render(self, console: tcod.Console) -> None:
        """Render the main menu on a background image."""
        console.draw_semigraphics(background_image, 0, 0)

#print the title of the game
        console.print(
            console.width // 2,
            console.height // 2 - 4,
            "RogueFishing",
            fg=color.menu_title,
            alignment=tcod.CENTER,
        )

#entirely by me vvv
        # Display random splash text
        console.print(
            console.width // 2,
            console.height - 2,
            #calls splash text func
            st_procgen(),
            fg=color.menu_title,
            alignment=tcod.CENTER,
        )
#entirely by me ^^^
```

```
#These are options to start the game/ quit it!
        menu_width = 24
        for i, text in enumerate(
            ["[N] Play a new game", "[C] Continue last game", "[Q] Quit"]
        ):
            console.print(
                console.width // 2,
                console.height // 2 - 2 + i,
                text.ljust(menu_width),
                fg=color.menu_text,
                bg=color.black,
                alignment=tcod.CENTER,
                bg_blend=tcod.BKGND_ALPHA(64),
            )

    def ev_keydown(
        self, event: tcod.event.KeyDown
    ) -> Optional[input_handlers.BaseEventHandler]:
        if event.sym in (tcod.event.K_q, tcod.event.K_ESCAPE):
            raise SystemExit()
        elif event.sym == tcod.event.K_c:
            try:
                return input_handlers.MainGameEventHandler(load_game("savegame.sav"))
            except FileNotFoundError:
                return input_handlers.PopupMessage(self, "No saved game to load.")
            except Exception as exc:
                traceback.print_exc()  # Print to stderr.
                return input_handlers.PopupMessage(self, f"Failed to load save:\n{exc}")
        elif event.sym == tcod.event.K_n:
            return input_handlers.MainGameEventHandler(new_game())

        return None

#entirely by me vvv
#splash text list
splashtexts = [
    "Fishtastic",
    "Fish not included!",
    "Infinite!?",
    "Replay value!",
    "Time has little to do with infinity and jelly doughnuts!",
    "Also try...I haven't really made anything else!",
    "The peak of roguelikes!",
    "Fishing? Not yet!",
    "0.35 is Water!",
    "Don't cave straight down!",
    "Reach the surface!",
    "If you make any input, I'll change!",
    "Made by me... for the most part!",
    "I am Steve!",
    "Deep combat!",
    "Mouse not included!",
    "Hardcore mode? We're in UltraHardcore mode!",
    "I yearn for the depths!",
    "Breathe out before going through tight squeezes!",
    "Press > to ascend!",
    "Tutorials are for the weak (I'm not lazy I swear)",
    "proper, grammer!",
    "No language support!",
    "Edgy!",
    "AP Computer Science Principals!",
    "Shoutout to my teacher!",
    "At least one gameplay mechanic!",
    "No mining OR crafting!",
    '"Shift" will not make you sprint!',
    "The impala is NOT tame!",
    "Lots of foes!",
    "Web fishing (but not on the web)!",
    "Gaben was not here!",
    "Wake up samurai, we've got a fish to catch!",
    "Better than Skyrim!",
    "The rocks were tricked into thinking!",
    "Philisophically deep",
    "Do NOT eat Plato!",
    "I think therefore I fish!",
    "The west has risen! Billions must fish!",
    "Unfinished? More like full of potential!",
    "Back in monochrome!",
    "Who needs fancy graphics when you have font?!",
    "The fewer the merrier!",
    "Fear the walking fish!",
    "Fishing Time!",
    "The fishing game with NO fishing Mechanics!",
    "Tilde does not open console!",
    "I'll lock in 2026!",
    "This is a list!",
    "Pure organic Python!",
    "Wholesome!",
    "We need to Fish!",
    "Have fun!",
    ":D",
    "If I can do this, you can too!",
    "What doth life?",
    "I like trains!",
    "Your adventure begins here (until you ragequit)!",
    "Now with saving!",
    "Contains no caffiene!",
    "Diet!",
    "It's a bittersweet roguelike this game!",
    "Do not consume!",
    "A set tone, what's that?!",
    "Fishing. Fishing never changes!",
    "Now with Mac support!",
    "Don't mind the bugs!",
    "Don't worry, be happy!",
    "Everything will be alright if you let it go!",
    "I see a red door and I want it painted in RGB!",

]
def st_procgen():
    return random.choice(splashtexts)
#entirely by me ^^^
```

**render_order.py:**

```
#RogueFishing entity rendering order file

#This file controls order in which various entities are rendered

#credits:
#general TCOD reference: https://python-tcod.readthedocs.io/en/latest/index.html
#roguelike tutorial: rogueliketutorials.com

#Used code by "Roguelike Tutorials", website found at rogueliketutorials.com with slight
addendums/modifications by me

#imports
from enum import auto, Enum


class RenderOrder(Enum):
    CORPSE = auto()
    ITEM = auto()
    ACTOR = auto()
```

**render_functions.py:**

```
#RogueFishing
#Render Functions file
```

```
#This file is responsible for displaying ui functions

#credits:
#general TCOD reference: https://python-tcod.readthedocs.io/en/latest/index.html
#roguelike tutorial: rogueliketutorials.com

#Used code from "Roguelike Tutorials" created by Tyler Standridge, website found at
rogueliketutorials.com with addendums/modifications by me



#imports
from __future__ import annotations

from typing import Tuple, TYPE_CHECKING

import color

if TYPE_CHECKING:
    from tcod import Console
    from engine import Engine
    from game_map import GameMap


def get_names_at_location(x: int, y: int, game_map: GameMap) -> str:
    if not game_map.in_bounds(x, y) or not game_map.visible[x, y]:
        return ""

    names = ", ".join(
        entity.name for entity in game_map.entities if entity.x == x and entity.y == y
    )

    return names.capitalize()

#displays cave level
def render_cave_level(
    console: Console, cave_level: int, location: Tuple[int, int]
) -> None:
    """
    Render the level the player is currently on, at the given location.
    """
    x, y = location

    console.print(x=x, y=y, string=f"Cave level: {cave_level}")

def render_names_at_mouse_location(
    console: Console, x: int, y: int, engine: Engine
) -> None:
    mouse_x, mouse_y = engine.mouse_location

    names_at_mouse_location = get_names_at_location(
        x=mouse_x, y=mouse_y, game_map=engine.game_map
    )

    console.print(x=x, y=y, string=names_at_mouse_location)
```

**procgen.py:**

```
#RogueFishing
#Procedural Generation File

#This file is responsible for procedurally generating the map

#comments are by me unless otherwise denoted

#credits:
#general TCOD reference: https://python-tcod.readthedocs.io/en/latest/index.html
#roguelike tutorial: rogueliketutorials.com

#vvv Used code by "Roguelike Tutorials", website found at rogueliketutorials.com with
slight addendums/modifications vvv

#imports
from __future__ import annotations

import random
from typing import Dict, Iterator, List, Tuple, TYPE_CHECKING

import tcod

import entity_factories
from game_map import GameMap

import tile_types

if TYPE_CHECKING:
    from engine import Engine
    from entity import Entity

#these control entities on each floor
max_items_by_floor = [
    (1, 1),
    (4, 2),
]
```

```
max_monsters_by_floor = [
    (1, 2),
    (4, 3),
    (6, 5),
]

#the first number is the level the item spawns on!
item_chances: Dict[int, List[Tuple[Entity, int]]] = {
    0: [(entity_factories.health_potion, 35)],
    2: [(entity_factories.Gambeson, 20)],
    4: [(entity_factories.Single_Shot_Musket, 25), (entity_factories.Shortsword, 5)],
    6: [(entity_factories.Single_Shot_Blunderbuss, 25)],
}

enemy_chances: Dict[int, List[Tuple[Entity, int]]] = {
    0: [(entity_factories.orc, 80), (entity_factories.water, 90)],
    3: [(entity_factories.troll, 15)],
    5: [(entity_factories.troll, 30)],
    7: [(entity_factories.troll, 60)],
}


def get_max_value_for_floor(
    max_value_by_floor: List[Tuple[int, int]], floor: int
) -> int:
    current_value = 0

    for floor_minimum, value in max_value_by_floor:
        if floor_minimum > floor:
            break
        else:
            current_value = value

    return current_value
#thia uses the previously created lists to actually randomly generate the entites
def get_entities_at_random(
    weighted_chances_by_floor: Dict[int, List[Tuple[Entity, int]]],
    number_of_entities: int,
    floor: int,
) -> List[Entity]:
    entity_weighted_chances = {}

    for key, values in weighted_chances_by_floor.items():
        if key > floor:
            break
        else:
```

```
        for value in values:
            entity = value[0]
            weighted_chance = value[1]

            entity_weighted_chances[entity] = weighted_chance

    entities = list(entity_weighted_chances.keys())
    entity_weighted_chance_values = list(entity_weighted_chances.values())

    chosen_entities = random.choices(
        entities, weights=entity_weighted_chance_values, k=number_of_entities
    )

    return chosen_entities


#defs
#creates room parameters
class RectangularRoom:
    def __init__(self, x: int, y: int, width: int, height: int):
        self.x1 = x
        self.y1 = y
        self.x2 = x + width
        self.y2 = y + height
    #defines room center
    @property
    def center(self) -> Tuple[int, int]:
        center_x = int((self.x1 + self.x2) / 2)
        center_y = int((self.y1 + self.y2) / 2)

        return center_x, center_y
    #defines what will be the inside of room
    @property
    def inner(self) -> Tuple[slice, slice]:
        """Return the inner area of this room as a 2D array index."""
        return slice(self.x1 + 1, self.x2), slice(self.y1 + 1, self.y2)
    #determines if two rooms intersect
    def intersects(self, other: RectangularRoom) -> bool:
        """Return True if this room overlaps with another RectangularRoom."""
        return (
            self.x1 <= other.x2
            and self.x2 >= other.x1
            and self.y1 <= other.y2
            and self.y2 >= other.y1
        )
#entity placement system VERY IMPORTANT
def place_entities(room: RectangularRoom, dungeon: GameMap, floor_number: int,) ->
```

```
None:
    number_of_monsters = random.randint(
        0, get_max_value_for_floor(max_monsters_by_floor, floor_number)
    )
    number_of_items = random.randint(
        0, get_max_value_for_floor(max_items_by_floor, floor_number)
    )

    monsters: List[Entity] = get_entities_at_random(
        enemy_chances, number_of_monsters, floor_number
    )
    items: List[Entity] = get_entities_at_random(
        item_chances, number_of_items, floor_number
    )

    for i in range(number_of_monsters):
        x = random.randint(room.x1 + 1, room.x2 - 1)
        y = random.randint(room.y1 + 1, room.y2 - 1)

        #Item SPAWNING, VERY IMPORTANT, (less important now, uses info from lists
and tubles above)
    for entity in monsters + items:
        x = random.randint(room.x1 + 1, room.x2 - 1)
        y = random.randint(room.y1 + 1, room.y2 - 1)

        if not any(entity.x == x and entity.y == y for entity in dungeon.entities):
            entity.spawn(dungeon, x, y)


def tunnel_between(
    start: Tuple[int, int], end: Tuple[int, int]
) -> Iterator[Tuple[int, int]]:
    """Return an L-shaped tunnel between these two points."""
    x1, y1 = start
    x2, y2 = end
    if random.random() < 0.5:  # 50% chance.
        # Move horizontally, then vertically.
        corner_x, corner_y = x2, y1
    else:
        # Move vertically, then horizontally.
        corner_x, corner_y = x1, y2

    # Generate the coordinates for this tunnel.
    for x, y in tcod.los.bresenham((x1, y1), (corner_x, corner_y)).tolist():
        yield x, y
    for x, y in tcod.los.bresenham((corner_x, corner_y), (x2, y2)).tolist():
        yield x, y
```

```
#procedurally generates a dungeon
def generate_dungeon(
    #parameters for procgen process
    max_rooms: int,
    room_min_size: int,
    room_max_size: int,
    map_width: int,
    map_height: int,
    engine: Engine,
) -> GameMap:
    """Generate a new dungeon map."""
    player = engine.player
    dungeon = GameMap(engine, map_width, map_height, entities=[player])

    rooms: List[RectangularRoom] = []
    #generate downstair/ladder
    center_of_last_room = (0, 0)

#vvv comments for this function not by me vvv
    for r in range(max_rooms):
        room_width = random.randint(room_min_size, room_max_size)
        room_height = random.randint(room_min_size, room_max_size)

        x = random.randint(0, dungeon.width - room_width - 1)
        y = random.randint(0, dungeon.height - room_height - 1)

        # "RectangularRoom" class makes rectangles easier to work with
        new_room = RectangularRoom(x, y, room_width, room_height)

        # Run through the other rooms and see if they intersect with this one.
        if any(new_room.intersects(other_room) for other_room in rooms):
            continue  # This room intersects, so go to the next attempt.
        # If there are no intersections then the room is valid.

        # Dig out this rooms inner area.
        dungeon.tiles[new_room.inner] = tile_types.floor

        if len(rooms) == 0:
            # The first room, where the player starts.
            player.place(*new_room.center, dungeon)
        else:  # All rooms after the first.
            # Dig out a tunnel between this room and the previous one.
            for x, y in tunnel_between(rooms[-1].center, new_room.center):
                dungeon.tiles[x, y] = tile_types.floor

            center_of_last_room = new_room.center
        #spawn items and mobs
        place_entities(new_room, dungeon, engine.game_world.current_floor)

        #spawn the up passage
        dungeon.tiles[center_of_last_room] = tile_types.up_passage
        dungeon.downstairs_location = center_of_last_room

        # Finally, append the new room to the list.
        rooms.append(new_room)
#^^^ comments for this function not by me ^^^

    return dungeon

#^^^ Used code by "Roguelike Tutorials", website found at rogueliketutorials.com with
slight addendums/modifications ^^^
```

**<u>message_log.py:</u>**

```
#RogueFishing in game log manager


#This file manages the ingame log which reports events in classic roguelike style!

#credits:
#general TCOD reference: https://python-tcod.readthedocs.io/en/latest/index.html
#roguelike tutorial: rogueliketutorials.com

#Used code from "Roguelike Tutorials" created by Tyler Standridge, website found at
rogueliketutorials.com with addendums/modifications by me


#imports
from typing import Iterable, List, Reversible, Tuple
import textwrap

import tcod

import color

#displays messages
class Message:
    def __init__(self, text: str, fg: Tuple[int, int, int]):
        self.plain_text = text
        self.fg = fg
        self.count = 1
    @property
    def full_text(self) -> str:
        """The full text of this message, including the count if necessary."""
        if self.count > 1:
            return f"{self.plain_text} (x{self.count})"
        return self.plain_text

#the message log
class MessageLog:
    def __init__(self) -> None:
        self.messages: List[Message] = []
    #stores past messages
    def add_message(
        self, text: str, fg: Tuple[int, int, int] = color.white, *, stack: bool = True,
    ) -> None:
        """Add a message to this log.
        `text` is the message text, `fg` is the text color.
        If `stack` is True then the message can stack with a previous message
        of the same text.
        """
        if stack and self.messages and text == self.messages[-1].plain_text:
            self.messages[-1].count += 1
        else:
            self.messages.append(Message(text, fg))

    def render(
        self, console: tcod.Console, x: int, y: int, width: int, height: int,
    ) -> None:
        """Render this log over the given area.
        `x`, `y`, `width`, `height` is the rectangular region to render onto
        the `console`.
        """
        self.render_messages(console, x, y, width, height, self.messages)

    @staticmethod
    def wrap(string: str, width: int) -> Iterable[str]:
        """Return a wrapped text message."""
        for line in string.splitlines():  # Handle newlines in messages.
            yield from textwrap.wrap(
                line, width, expand_tabs=True,
            )

    @classmethod
    def render_messages(
        cls,
        console: tcod.Console,
        x: int,
        y: int,
        width: int,
        height: int,
        messages: Reversible[Message],
    ) -> None:
        """Render the messages provided.
        The `messages` are rendered starting at the last message and working
        backwards.
        """
        y_offset = height - 1

        for message in reversed(messages):
            for line in reversed(list(cls.wrap(message.full_text, width))):
                console.print(x=x, y=y + y_offset, string=line, fg=message.fg)
                y_offset -= 1
                if y_offset < 0:
                    return  # No more space to print messages.
```

**input_handlers.py:**

```
#RogueFishing
#Input Handlers File

#This file is responsible for managing inputs

#credits:
#general TCOD reference: https://python-tcod.readthedocs.io/en/latest/index.html
#roguelike tutorial: rogueliketutorials.com

#Used code from "Roguelike Tutorials" created by Tyler Standridge, website found at
rogueliketutorials.com with addendums/modifications by me

from __future__ import annotations

import os

from typing import Callable, Optional, Tuple, TYPE_CHECKING, Union

import tcod.event

import actions
from actions import (
    Action,
    BumpAction,
    PickupAction,
    WaitAction
```

```
)
import color
import exceptions

if TYPE_CHECKING:
    from engine import Engine
    from entity import Item

#These control ALL inputs from controls to escape to item pickup/drop!

MOVE_KEYS = {
    # Arrow keys.
    tcod.event.K_UP: (0, -1),
    tcod.event.K_DOWN: (0, 1),
    tcod.event.K_LEFT: (-1, 0),
    tcod.event.K_RIGHT: (1, 0),
    tcod.event.K_HOME: (-1, -1),
    tcod.event.K_END: (-1, 1),
    tcod.event.K_PAGEUP: (1, -1),
    tcod.event.K_PAGEDOWN: (1, 1),
    # Numpad keys.
    tcod.event.K_KP_1: (-1, 1),
    tcod.event.K_KP_2: (0, 1),
    tcod.event.K_KP_3: (1, 1),
    tcod.event.K_KP_4: (-1, 0),
    tcod.event.K_KP_6: (1, 0),
    tcod.event.K_KP_7: (-1, -1),
    tcod.event.K_KP_8: (0, -1),
    tcod.event.K_KP_9: (1, -1),
    # Vi keys.
    tcod.event.K_h: (-1, 0),
    tcod.event.K_j: (0, 1),
    tcod.event.K_k: (0, -1),
    tcod.event.K_l: (1, 0),
    tcod.event.K_y: (-1, -1),
    tcod.event.K_u: (1, -1),
    tcod.event.K_b: (-1, 1),
    tcod.event.K_n: (1, 1),
}

WAIT_KEYS = {
    tcod.event.K_PERIOD,
    tcod.event.K_KP_5,
    tcod.event.K_CLEAR,
}

CONFIRM_KEYS = {
    tcod.event.K_RETURN,
    tcod.event.K_KP_ENTER,
}


ActionOrHandler = Union[Action, "BaseEventHandler"]
"""An event handler return value which can trigger an action or switch active handlers.

If a handler is returned then it will become the active handler for future events.
If an action is returned it will be attempted and if it's valid then
MainGameEventHandler will become the active handler.
"""


class BaseEventHandler(tcod.event.EventDispatch[ActionOrHandler]):
    def handle_events(self, event: tcod.event.Event) -> BaseEventHandler:
        """Handle an event and return the next active event handler."""
        state = self.dispatch(event)
        if isinstance(state, BaseEventHandler):
            return state
        assert not isinstance(state, Action), f"{self!r} can not handle actions."
        return self

    def on_render(self, console: tcod.Console) -> None:
        raise NotImplementedError()

    def ev_quit(self, event: tcod.event.Quit) -> Optional[Action]:
        raise SystemExit()

#this displays a popup that can be dismissed
class PopupMessage(BaseEventHandler):
    """Display a popup text window."""

    def __init__(self, parent_handler: BaseEventHandler, text: str):
        self.parent = parent_handler
        self.text = text

    def on_render(self, console: tcod.Console) -> None:
        """Render the parent and dim the result, then print the message on top."""
        self.parent.on_render(console)
        console.tiles_rgb["fg"] //= 8
        console.tiles_rgb["bg"] //= 8

        console.print(
            console.width // 2,
            console.height // 2,
            self.text,
            fg=color.white,
            bg=color.black,
            alignment=tcod.CENTER,
        )

    def ev_keydown(self, event: tcod.event.KeyDown) -> Optional[BaseEventHandler]:
        """Any key returns to the parent handler."""
        return self.parent


#This sets up a general event handler that I can use for a variety of scenarios
class EventHandler(BaseEventHandler):
    def __init__(self, engine: Engine):
        self.engine = engine

    def handle_events(self, event: tcod.event.Event) -> BaseEventHandler:
        """Handle events for input handlers with an engine."""
        action_or_state = self.dispatch(event)
        if isinstance(action_or_state, BaseEventHandler):
            return action_or_state
        if self.handle_action(action_or_state):
            # A valid action was performed.
            if not self.engine.player.is_alive:
                # The player was killed sometime during or after the action.
                return GameOverEventHandler(self.engine)
            return MainGameEventHandler(self.engine)# Return to the main handler.
        elif self.engine.player.level.requires_level_up:
            return LevelUpEventHandler(self.engine)
        return self

    def handle_action(self, action: Optional[Action]) -> bool:
        """Handle actions returned from event methods.

        Returns True if the action will advance a turn.
        """
        if action is None:
            return False

        try:
            action.perform()
        except exceptions.Impossible as exc:
            self.engine.message_log.add_message(exc.args[0], color.impossible)
            return False  # Skip enemy turn on exceptions.

        self.engine.handle_enemy_turns()

        self.engine.update_fov()
        return True
    #tracks mouse
    def ev_mousemotion(self, event: tcod.event.MouseMotion) -> None:
        if self.engine.game_map.in_bounds(event.tile.x, event.tile.y):
            self.engine.mouse_location = event.tile.x, event.tile.y

    def on_render(self, console: tcod.Console) -> None:
        self.engine.render(console)

class AskUserEventHandler(EventHandler):
    """Handles user input for actions which require special input."""


    def ev_keydown(self, event: tcod.event.KeyDown) -> Optional[ActionOrHandler]:
        """By default any key exits this input handler."""
        if event.sym in {  # Ignore modifier keys.
            tcod.event.K_LSHIFT,
            tcod.event.K_RSHIFT,
            tcod.event.K_LCTRL,
            tcod.event.K_RCTRL,
            tcod.event.K_LALT,
            tcod.event.K_RALT,
        }:
            return None
        return self.on_exit()

    def ev_mousebuttondown(
        self, event: tcod.event.MouseButtonDown
    ) -> Optional[ActionOrHandler]:
        """By default any mouse click exits this input handler."""
        return self.on_exit()

    def on_exit(self) -> Optional[ActionOrHandler]:
        """Called when the user is trying to exit or cancel an action.

        By default this returns to the main event handler.
        """
        return MainGameEventHandler(self.engine)

#chracter stats screen
class CharacterScreenEventHandler(AskUserEventHandler):
    TITLE = "Character Information"

    def on_render(self, console: tcod.Console) -> None:
        super().on_render(console)

        if self.engine.player.x <= 30:
            x = 40
        else:
            x = 0

        y = 0

        width = len(self.TITLE) + 4

        console.draw_frame(
            x=x,
            y=y,
            width=width,
            height=8,
            title=self.TITLE,
            clear=True,
            fg=(255, 255, 255),
            bg=(0, 0, 0),
        )

        console.print(
            x=x + 1, y=y + 1, string=f"Level: {self.engine.player.level.current_level}"
        )
        console.print(
            x=x + 1, y=y + 2, string=f"XP: {self.engine.player.level.current_xp}"
        )
        console.print(
            x=x + 1,
            y=y + 3,
            string=f"XP for next Level: {self.engine.player.level.experience_to_next_level}",
        )

        console.print(
            x=x + 1, y=y + 4, string=f"Violence: {self.engine.player.fighter.power}"
        )
        console.print(
            x=x + 1, y=y + 5, string=f"Fortitude: {self.engine.player.fighter.defense}"
        )
        console.print(
            x=x + 1, y=y + 6, string=f"Vitality: {self.engine.player.fighter.hp}"
        )

#this manages leveling up
class LevelUpEventHandler(AskUserEventHandler):
    TITLE = "Level Up"

    def on_render(self, console: tcod.Console) -> None:
        super().on_render(console)
        if self.engine.player.x <= 30:
            x = 40
        else:
            x = 0

        console.draw_frame(
            x=x,
            y=0,
            width=35,
            height=8,
            title=self.TITLE,
            clear=True,
            fg=(255, 255, 255),
            bg=(0, 0, 0),
        )

        console.print(x=x + 1, y=1, string="You reach a breakthrough!")
        console.print(x=x + 1, y=2, string="Enchance one:")

        console.print(
            x=x + 1,
            y=4,
            string=f"a) Vitality (+20 HP, from {self.engine.player.fighter.max_hp})",
        )
        console.print(
            x=x + 1,
            y=5,
            string=f"b) Violence (+1 attack, from {self.engine.player.fighter.power})",
        )
        console.print(
            x=x + 1,
            y=6,
            string=f"c) Fortitude (+1 defense, from {self.engine.player.fighter.defense})",
        )

    def ev_keydown(self, event: tcod.event.KeyDown) -> Optional[ActionOrHandler]:
        player = self.engine.player
        key = event.sym
        index = key - tcod.event.K_a

        if 0 <= index <= 2:
            if index == 0:
                player.level.increase_max_hp()
            elif index == 1:
                player.level.increase_power()
            else:
                player.level.increase_defense()
            else:
                self.engine.message_log.add_message("Invalid entry.", color.invalid)

                return None

            return super().ev_keydown(event)

        def ev_mousebuttondown(
            self, event: tcod.event.MouseButtonDown
        ) -> Optional[ActionOrHandler]:
            """
            Don't allow the player to click to exit the menu, like normal.
            """
            return None

class InventoryEventHandler(AskUserEventHandler):
    """This handler lets the user select an item.

    What happens then depends on the subclass.
    """

    TITLE = "<missing title>"

    def on_render(self, console: tcod.Console) -> None:
        """Render an inventory menu, which displays the items in the inventory, and the
letter to select them.
        Will move to a different position based on where the player is located, so the player
can always see where
        they are.
        """
        super().on_render(console)
        number_of_items_in_inventory = len(self.engine.player.inventory.items)

        height = number_of_items_in_inventory + 2

        if height <= 3:
            height = 3

        if self.engine.player.x <= 30:
            x = 40
        else:
            x = 0

        y = 0

        width = len(self.TITLE) + 4
        console.draw_frame(
            x=x,
            y=y,
            width=width,
            height=height,
            title=self.TITLE,
            clear=True,
            fg=(255, 255, 255),
            bg=(0, 0, 0),
        )

        if number_of_items_in_inventory > 0:
            for i, item in enumerate(self.engine.player.inventory.items):
                item_key = chr(ord("a") + i)
                is_equipped = self.engine.player.equipment.item_is_equipped(item)

                item_string = f"({item_key}) {item.name}"

                if is_equipped:
                    item_string = f"{item_string} (E)"

                console.print(x + 1, y + i + 1, item_string)
        else:
            console.print(x + 1, y + 1, "(Empty)")

    def ev_keydown(self, event: tcod.event.KeyDown) -> Optional[ActionOrHandler]:
        player = self.engine.player
        key = event.sym
        index = key - tcod.event.K_a

        if 0 <= index <= 26:
            try:
                selected_item = player.inventory.items[index]
            except IndexError:
                self.engine.message_log.add_message("Invalid entry.", color.invalid)
                return None
            return self.on_item_selected(selected_item)
        return super().ev_keydown(event)

    def on_item_selected(self, item: Item) -> Optional[ActionOrHandler]:
        """Called when the user selects a valid item."""
        raise NotImplementedError()

class InventoryActivateHandler(InventoryEventHandler):
    """Handle using an inventory item."""
    TITLE = "Select an item to use"

    def on_item_selected(self, item: Item) -> Optional[ActionOrHandler]:
        if item.consumable:
            # Return the action for the selected item.
            return item.consumable.get_action(self.engine.player)
        elif item.equippable:
            return actions.EquipAction(self.engine.player, item)
        else:
            return None


class InventoryDropHandler(InventoryEventHandler):
    """Handle dropping an inventory item."""

    TITLE = "Select an item to drop"

    def on_item_selected(self, item: Item) -> Optional[ActionOrHandler]:
        """Drop this item."""
        return actions.DropItem(self.engine.player, item)

class SelectIndexHandler(AskUserEventHandler):
    """Handles asking the user for an index on the map."""

    def __init__(self, engine: Engine):
        """Sets the cursor to the player when this handler is constructed."""
        super().__init__(engine)
        player = self.engine.player
        engine.mouse_location = player.x, player.y

    def on_render(self, console: tcod.Console) -> None:
        """Highlight the tile under the cursor."""
        super().on_render(console)
        x, y = self.engine.mouse_location
        console.tiles_rgb["bg"][x, y] = color.white
        console.tiles_rgb["fg"][x, y] = color.black

    def ev_keydown(self, event: tcod.event.KeyDown) -> Optional[ActionOrHandler]:
        """Check for key movement or confirmation keys."""
        key = event.sym
        if key in MOVE_KEYS:
            modifier = 1  # Holding modifier keys will speed up key movement.
            if event.mod & (tcod.event.KMOD_LSHIFT | tcod.event.KMOD_RSHIFT):
                modifier *= 5
            if event.mod & (tcod.event.KMOD_LCTRL | tcod.event.KMOD_RCTRL):
                modifier *= 10
            if event.mod & (tcod.event.KMOD_LALT | tcod.event.KMOD_RALT):
                modifier *= 20

            x, y = self.engine.mouse_location
            dx, dy = MOVE_KEYS[key]
            x += dx * modifier
            y += dy * modifier
            # Clamp the cursor index to the map size.
            x = max(0, min(x, self.engine.game_map.width - 1))
            y = max(0, min(y, self.engine.game_map.height - 1))
            self.engine.mouse_location = x, y
            return None
        elif key in CONFIRM_KEYS:
            return self.on_index_selected(*self.engine.mouse_location)
        return super().ev_keydown(event)

    def ev_mousebuttondown(
        self, event: tcod.event.MouseButtonDown
    ) -> Optional[ActionOrHandler]:
        """Left click confirms a selection."""
        if self.engine.game_map.in_bounds(*event.tile):
            if event.button == 1:
                return self.on_index_selected(*event.tile)
        return super().ev_mousebuttondown(event)

    def on_index_selected(self, x: int, y: int) -> Optional[ActionOrHandler]:
        """Called when an index is selected."""
        raise NotImplementedError()


class LookHandler(SelectIndexHandler):
    """Lets the player look around using the keyboard."""

    def on_index_selected(self, x: int, y: int) -> MainGameEventHandler:
        """Return to main handler."""
        return MainGameEventHandler(self.engine)

#ranged targeting
class SingleRangedAttackHandler(SelectIndexHandler):
    """Handles targeting a single enemy. Only the enemy selected will be affected."""

    def __init__(
        self, engine: Engine, callback: Callable[[Tuple[int, int]], Optional[Action]]
    ):
        super().__init__(engine)

        self.callback = callback
    def on_index_selected(self, x: int, y: int) -> Optional[Action]:
        return self.callback((x, y))
#AOE attack
class AreaRangedAttackHandler(SelectIndexHandler):
    """Handles targeting an area within a given radius. Any entity within the area will be
affected."""

    def __init__(
        self,
        engine: Engine,
        radius: int,
        callback: Callable[[Tuple[int, int]], Optional[Action]],
    ):
        super().__init__(engine)

        self.radius = radius
        self.callback = callback

    def on_render(self, console: tcod.Console) -> None:
        """Highlight the tile under the cursor."""
        super().on_render(console)

        x, y = self.engine.mouse_location

        # Draw a rectangle around the targeted area, so the player can see the affected
tiles.
        console.draw_frame(
            x=x - self.radius - 1,
            y=y - self.radius - 1,
            width=self.radius ** 2,
            height=self.radius ** 2,
            fg=color.red,
            clear=False,
        )

    def on_index_selected(self, x: int, y: int) -> Optional[Action]:
        return self.callback((x, y))

class MainGameEventHandler(EventHandler):

    def ev_keydown(self, event: tcod.event.KeyDown) -> Optional[ActionOrHandler]:
        action: Optional[Action] = None

        key = event.sym
        modifier = event.mod

        player = self.engine.player
        if key == tcod.event.K_PERIOD and modifier & (
            tcod.event.KMOD_LSHIFT | tcod.event.KMOD_RSHIFT
        ):
            return actions.TakePassageAction(player)
        #player movement controls
        if key in MOVE_KEYS:
            dx, dy = MOVE_KEYS[key]
            action = BumpAction(player, dx, dy)
        elif key in WAIT_KEYS:
            action = WaitAction(player)
        #escape kills the game
        elif key == tcod.event.K_ESCAPE:
            raise SystemExit()
        elif key == tcod.event.K_v:
            return HistoryViewer(self.engine)
        elif key == tcod.event.K_g:
            action = PickupAction(player)
        elif key == tcod.event.K_i:
            return InventoryActivateHandler(self.engine)
        elif key == tcod.event.K_d:
            return InventoryDropHandler(self.engine)
        elif key == tcod.event.K_c:
            return CharacterScreenEventHandler(self.engine)
        elif key == tcod.event.K_SLASH:
            return LookHandler(self.engine)


        # No valid key was pressed
        return action

class GameOverEventHandler(EventHandler):
    def on_quit(self) -> None:
        """Handle exiting out of a finished game."""
        if os.path.exists("savegame.sav"):
            os.remove("savegame.sav")  # Deletes the active save file.
        raise exceptions.QuitWithoutSaving()  # Avoid saving a finished game.

    def ev_quit(self, event: tcod.event.Quit) -> None:
        self.on_quit()

    def ev_keydown(self, event: tcod.event.KeyDown) -> None:
        if event.sym == tcod.event.K_ESCAPE:
            self.on_quit()

CURSOR_Y_KEYS = {
    tcod.event.K_UP: -1,
    tcod.event.K_DOWN: 1,
    tcod.event.K_PAGEUP: -10,
    tcod.event.K_PAGEDOWN: 10,
}


class HistoryViewer(EventHandler):
    """Print the history on a larger window which can be navigated."""

    def __init__(self, engine: Engine):
        super().__init__(engine)
        self.log_length = len(engine.message_log.messages)
        self.cursor = self.log_length - 1

    def on_render(self, console: tcod.Console) -> None:
        super().on_render(console)  # Draw the main state as the background.

        log_console = tcod.Console(console.width - 6, console.height - 6)

        # Draw a frame with a custom banner title.
        log_console.draw_frame(0, 0, log_console.width, log_console.height)
        log_console.print_box(
            0, 0, log_console.width, 1, "┤ Message history ├", alignment=tcod.CENTER
        )

        # Render the message log using the cursor parameter.
        self.engine.message_log.render_messages(
            log_console,
            1,
            1,
            log_console.width - 2,
            log_console.height - 2,
            self.engine.message_log.messages[: self.cursor + 1],
        )
        log_console.blit(console, 3, 3)

    def ev_keydown(self, event: tcod.event.KeyDown) ->
Optional[MainGameEventHandler]:
        # Fancy conditional movement to make it feel right.
        if event.sym in CURSOR_Y_KEYS:
            adjust = CURSOR_Y_KEYS[event.sym]
            if adjust < 0 and self.cursor == 0:
                # Only move from the top to the bottom when you're on the edge.
                self.cursor = self.log_length - 1
            elif adjust > 0 and self.cursor == self.log_length - 1:
                # Same with bottom to top movement.
                self.cursor = 0
            else:
                # Otherwise move while staying clamped to the bounds of the history log.
                self.cursor = max(0, min(self.cursor + adjust, self.log_length - 1))
        elif event.sym == tcod.event.K_HOME:
            self.cursor = 0  # Move directly to the top message.
        elif event.sym == tcod.event.K_END:
            self.cursor = self.log_length - 1  # Move directly to the last message.
        else:  # Any other key moves back to the main game state.
            return MainGameEventHandler(self.engine)
        return None
```

**level.py:**

```
#RogueFishing
#leveling file

#This file is responsible for managing leveling and its effects on the player

#credits:
#general TCOD reference: https://python-tcod.readthedocs.io/en/latest/index.html
#roguelike tutorial: rogueliketutorials.com

#Used code from "Roguelike Tutorials" created by Tyler Standridge, website found at
rogueliketutorials.com with addendums/modifications by me


from __future__ import annotations

from typing import TYPE_CHECKING

from components.base_component import BaseComponent

if TYPE_CHECKING:
    from entity import Actor

#default leveling info
class Level(BaseComponent):
    parent: Actor

    def __init__(
        self,
        current_level: int = 1,
        current_xp: int = 0,
        level_up_base: int = 0,
        level_up_factor: int = 150,
        xp_given: int = 0,
    ):
        self.current_level = current_level
        self.current_xp = current_xp
        self.level_up_base = level_up_base
        self.level_up_factor = level_up_factor
        self.xp_given = xp_given
#leveling equation
    @property
    def experience_to_next_level(self) -> int:
        return self.level_up_base + self.current_level * self.level_up_factor

    @property
    def requires_level_up(self) -> bool:
        return self.current_xp > self.experience_to_next_level
#leveling system
    def add_xp(self, xp: int) -> None:
        if xp == 0 or self.level_up_base == 0:
            return

        self.current_xp += xp

        self.engine.message_log.add_message(f"You learn more from the caves and gain
{xp} exp.")

        if self.requires_level_up:
            self.engine.message_log.add_message(
                f"You reach a breakthrough in your knowledge and achieve level
{self.current_level + 1}!"
            )

    def increase_level(self) -> None:
        self.current_xp -= self.experience_to_next_level

        self.current_level += 1

    def increase_max_hp(self, amount: int = 3) -> None:
        self.parent.fighter.max_hp += amount
        self.parent.fighter.hp += amount

        self.engine.message_log.add_message("Stress builds strength: your body has
taken so many beatings you grow stronger! (+hp)")

        self.increase_level()

    def increase_power(self, amount: int = 1) -> None:
        self.parent.fighter.power += amount

        self.engine.message_log.add_message("You've killed so much you grow better at
it. Is this really nessecary? (+dam)")

        self.increase_level()

    def increase_defense(self, amount: int = 1) -> None:
        self.parent.fighter.defense += amount

        self.engine.message_log.add_message("Your skin callouses and hardens, enemy
blows hurt you less! (+def)")

        self.increase_level()
```

**game_map.py:**

```
#RogueFishing
#Input Handlers File

#This file is responsible for creating and managing tiles and all tile related elements


#credits:
#general TCOD reference: https://python-tcod.readthedocs.io/en/latest/index.html
#roguelike tutorial: rogueliketutorials.com

#Used code from "Roguelike Tutorials" created by Tyler Standridge, website found at
rogueliketutorials.com with addendums/modifications by me

from __future__ import annotations

from typing import Iterable, Iterator, Optional, TYPE_CHECKING

import numpy as np  # type: ignore
from tcod.console import Console

from entity import Actor, Item
import tile_types

if TYPE_CHECKING:
    from engine import Engine
    from entity import Entity

class GameMap:
    #this is doing some sketchy numpy array stuff that is definitely beyond me right now
lol

    def __init__(
        self, engine: Engine, width: int, height: int, entities: Iterable[Entity] = ()
    ):
        self.engine = engine
        self.width, self.height = width, height
        #entity spawning
        self.entities = set(entities)
        self.tiles = np.full((width, height), fill_value=tile_types.wall, order="F")
        #these arrays determine if a tile has been "seen" or "not seen"vvv
        #comments below not by me vvv
        self.visible = np.full(
            (width, height), fill_value=False, order="F"
        )  # Tiles the player can currently see
        self.explored = np.full(
            (width, height), fill_value=False, order="F"
        )  # Tiles the player has seen before
        #comments above not by me ^^^
        self.tiles[30:33, 22] = tile_types.wall
        #default stairwell placement
        self.uppassage_location = (0, 0)

    @property
    def gamemap(self) -> GameMap:
        return self

    @property
    def actors(self) -> Iterator[Actor]:
        """Iterate over this maps living actors."""
        yield from (
            entity
            for entity in self.entities
            if isinstance(entity, Actor) and entity.is_alive
        )

    @property
    def items(self) -> Iterator[Item]:
        yield from (entity for entity in self.entities if isinstance(entity, Item))

    #determines if an entity is on a tile
    def get_blocking_entity_at_location(
        self, location_x: int, location_y: int,
    ) -> Optional[Entity]:
        for entity in self.entities:
            if (
                entity.blocks_movement
                and entity.x == location_x
                and entity.y == location_y
            ):
            return None

    def in_bounds(self, x: int, y: int) -> bool:
        """Return True if x and y are inside of the bounds of this map."""
        return 0 <= x < self.width and 0 <= y < self.height

    def render(self, console: Console) -> None:
        """
        Renders the map.

        If a tile is in the "visible" array, then draw it with the "light" colors.
        If it isn't, but it's in the "explored" array, then draw it with the "dark" colors.
        Otherwise, the default is "SHROUD".
        """
        console.tiles_rgb[0 : self.width, 0 : self.height] = np.select(
            condlist=[self.visible, self.explored],
            choicelist=[self.tiles["light"], self.tiles["dark"]],
            default=tile_types.SHROUD,
        )

        entities_sorted_for_rendering = sorted(
            self.entities, key=lambda x: x.render_order.value
        )

        #entity rendering system, which is now handled by game_map
        for entity in entities_sorted_for_rendering:
            # Only print entities that are in the FOV
            if self.visible[entity.x, entity.y]:
                console.print(
                    x=entity.x, y=entity.y, string=entity.char, fg=entity.color
                )

    def get_actor_at_location(self, x: int, y: int) -> Optional[Actor]:
        for actor in self.actors:
            if actor.x == x and actor.y == y:
                return actor

        return None

#generates new maps
class GameWorld:
    """
    Holds the settings for the GameMap, and generates new maps when moving toward
the surface
    """

    def __init__(
        self,
        *,
        engine: Engine,
        map_width: int,
        map_height: int,
        max_rooms: int,
        room_min_size: int,
        room_max_size: int,
        current_floor: int = 0
    ):
        self.engine = engine

        self.map_width = map_width
        self.map_height = map_height

        self.max_rooms = max_rooms

        self.room_min_size = room_min_size
        self.room_max_size = room_max_size

        self.current_floor = current_floor

    def generate_floor(self) -> None:
        from procgen import generate_dungeon

        self.current_floor += 1

        self.engine.game_map = generate_dungeon(
            max_rooms=self.max_rooms,
            room_min_size=self.room_min_size,
            room_max_size=self.room_max_size,
            map_width=self.map_width,
            map_height=self.map_height,
            engine=self.engine,
        )
```

**exceptions.py:**

```
#RogueFishing exceptions file

#This file allows for special exceptions to be created and managed

#credits:
#general TCOD reference: https://python-tcod.readthedocs.io/en/latest/index.html
#roguelike tutorial: roguelketutorials.com

#Used code by "Roguelike Tutorials", website found at roguelketutorials.com with slight
```

addendums/modifications by me

```
class Impossible(Exception):
    """Exception raised when an action is impossible to be performed.

    The reason is given as the exception message.
    """

class QuitWithoutSaving(SystemExit):
    """Can be raised to exit the game without automatically saving."""
```

**equipment_types.py:**

#equpment types

#This file manages the various equipment types.

#credits:
#general TCOD reference: https://python-tcod.readthedocs.io/en/latest/index.html
#roguelike tutorial: rogueliketutorials.com

#Used code from "Roguelike Tutorials" created by Tyler Standridge, website found at rogueliketutorials.com with addendums/modifications by me

```
#imports
from enum import auto, Enum


class EquipmentType(Enum):
    WEAPON = auto()
    ARMOR = auto()
```

**entity.py:**

#entity controller

#this file controls ALL entities that will be used in RogueFishing

#credits:
#general TCOD reference: https://python-tcod.readthedocs.io/en/latest/index.html
#roguelike tutorial: rogueliketutorials.com

#Used code from "Roguelike Tutorials" created by Tyler Standridge, website found at rogueliketutorials.com with addendums/modifications by me

```
#imports
from __future__ import annotations
import copy
import math
from typing import Optional, Tuple, Type, TypeVar, TYPE_CHECKING, Union

from render_order import RenderOrder

if TYPE_CHECKING:
    from components.ai import BaseAI
    from components.consumable import Consumable
    from components.equipment import Equipment
    from components.equippable import Equippable
    from components.fighter import Fighter
    from components.inventory import Inventory
    from components.level import Level
    from game_map import GameMap

#basic entity template
T = TypeVar("T", bound="Entity")

class Entity:
    """
    A generic object to represent players, enemies, items, etc.
    """

    parent: Union[GameMap, Inventory]

    def __init__(
        self,
        parent: Optional[GameMap] = None,
        x: int = 0,
        y: int = 0,
        char: str = "?",
        color: Tuple[int, int, int] = (255, 255, 255),
        name: str = "<Unnamed>",
        blocks_movement: bool = False,
        render_order: RenderOrder = RenderOrder.CORPSE,
    ):
        self.x = x
        self.y = y
        self.char = char
        self.color = color
        self.name = name
        #allows entity to be moved over/not count as a block
        self.blocks_movement = blocks_movement
        self.render_order = render_order
        if parent:
            # If parent isn't provided now then it will be set later.
            self.parent = parent
            parent.entities.add(self)

    @property
    def gamemap(self) -> GameMap:
        return self.parent.gamemap

    #spawn entities
    def spawn(self: T, gamemap: GameMap, x: int, y: int) -> T:
        """Spawn a copy of this instance at the given location."""
        clone = copy.deepcopy(self)
        clone.x = x
        clone.y = y
        clone.parent = gamemap
        gamemap.entities.add(clone)
        return clone

    def place(self, x: int, y: int, gamemap: Optional[GameMap] = None) -> None:
        """Place this entity at a new location.  Handles moving across GameMaps."""
        self.x = x
        self.y = y
        if gamemap:
            if hasattr(self, "parent"):  # Possibly uninitialized.
                if self.parent is self.gamemap:
                    self.gamemap.entities.remove(self)
            self.parent = gamemap
            gamemap.entities.add(self)

    def distance(self, x: int, y: int) -> float:
        """
        Return the distance between the current entity and the given (x, y) coordinate.
        """
        return math.sqrt((x - self.x) ** 2 + (y - self.y) ** 2)


    def move(self, dx: int, dy: int) -> None:
        # Move the entity by a given amount
        self.x += dx
        self.y += dy

class Actor(Entity):
    def __init__(
        self,
        *,
        x: int = 0,
        y: int = 0,
        char: str = "?",
        color: Tuple[int, int, int] = (255, 255, 255),
        name: str = "<Unnamed>",
        ai_cls: Type[BaseAI],
        equipment: Equipment,
        fighter: Fighter,
        inventory: Inventory,
        level: Level,
    ):
        super().__init__(
            x=x,
            y=y,
            char=char,
            color=color,
            name=name,
            blocks_movement=True,
            render_order=RenderOrder.ACTOR,
        )

        self.ai: Optional[BaseAI] = ai_cls(self)
        #import equippables
        self.equipment: Equipment = equipment
        self.equipment.parent = self
        #designate as player
        self.fighter = fighter
        self.fighter.parent = self

        self.inventory = inventory
        self.inventory.parent = self

        self.level = level
        self.level.parent = self

    @property
    def is_alive(self) -> bool:
        """Returns True as long as this actor can perform actions."""
        return bool(self.ai)

class Item(Entity):
    def __init__(
        self,
        *,
        x: int = 0,
        y: int = 0,
        char: str = "?",
        color: Tuple[int, int, int] = (255, 255, 255),
        name: str = "<Unnamed>",
        consumable: Optional[Consumable] = None,
        equippable: Optional[Equippable] = None,
    ):
        super().__init__(
            x=x,
            y=y,
            char=char,
            color=color,
            name=name,
            blocks_movement=False,
            render_order=RenderOrder.ITEM,
        )

        self.consumable = consumable
        if self.consumable:
            self.consumable.parent = self

        self.equippable = equippable

        if self.equippable:
            self.equippable.parent = self

class Static(Entity):
    def __init__(
        self,
        *,
        x: int = 0,
        y: int = 0,
        char: str = "?",
        color: Tuple[int, int, int] = (255, 255, 255),
        name: str = "<Unnamed>",
    ):
        super().__init__(
            x=x,
            y=y,
            char=char,
            color=color,
            name=name,
            blocks_movement=False,
            render_order=RenderOrder.ITEM,
        )
```

**entity_factories.py**

```
#additional entity controller


#This file manages entities, specifically when they are cloned and transferred to
game_map.py, additionally it allows the various entities to be defined!

#credits:
#general TCOD reference: https://python-tcod.readthedocs.io/en/latest/index.html
#roguelike tutorial: rogueliketutorials.com

#Used code from "Roguelike Tutorials" created by Tyler Standridge, website found at
rogueliketutorials.com with addendums/modifications by me

#imports

from components.ai import HostileEnemy
from components.ai import Static
from components import consumable, equippable
from components.equipment import Equipment
from components.fighter import Fighter
from components.inventory import Inventory
from components.level import Level
from entity import Actor, Item, Static

#These options allow entities to be created and customized!
#entitiy visual appearance and gameplay attributes customizable! yipeeee!

#Player
player = Actor(
    char="@",
    color=(255, 255, 255),
    name="Player",
    #because player doesn't use ai this is useless vvv
    ai_cls=HostileEnemy,
    equipment=Equipment(),
    fighter=Fighter(hp=30, base_defense=1, base_power=2),
    inventory=Inventory(capacity=26),
    level=Level(level_up_base=200),
)

#NPC - Enemy
orc = Actor(
    char="}",
    color=(63, 127, 63),
    name="Feral Human",
    ai_cls=HostileEnemy,
    equipment=Equipment(),
    fighter=Fighter(hp=10, base_defense=0, base_power=3),
    inventory=Inventory(capacity=0),
    level=Level(xp_given=35),
)
troll = Actor(
    char="~",
    color=(0, 127, 0),
    name="Cave Horror",
    ai_cls=HostileEnemy,
    equipment=Equipment(),
    fighter=Fighter(hp=16, base_defense=1, base_power=4),
    inventory=Inventory(capacity=0),
    level=Level(xp_given=100),
)

#consumable - health
health_potion = Item(
    char="I",
    color=(255, 255, 255),
    name="Bandages",
    consumable=consumable.HealingConsumable(amount=4),
)

#consumable - weapon
Single_Shot_Musket = Item(
    char="↔",
    color=(165, 42, 42),
    name="Single Shot Handgun",
    consumable=consumable. SingleShotMusket(damage=12, maximum_range=5),
)

Single_Shot_Blunderbuss = Item(
    char="`",
    color=(255, 0, 0),
    name="Single Shot Blunderbuss",
    consumable=consumable.SingleShotBlunderbuss(damage=10, radius=3),
)

#equippables
Shortsword = Item(
    char="◄", color=(184, 115, 51), name="Shortsword",
equippable=equippable.Shortsword()
)

Cleaver = Item(char="▼", color=(0, 191, 255), name="Cleaver",
equippable=equippable.Cleaver())
```

```
Gambeson = Item(
    char="▶",
    color=(128, 128, 128),
    name="Gambeson",
    equippable=equippable.Gambeson(),
)

FlakVest = Item(
    char="[", color=(139, 69, 19), name="Flak Vest", equippable=equippable.FlakVest()
)

#objects
water = Static(
    char="%",
    color=(0, 0, 255),
    name="Water",
)
```

**engine.py:**

```
#RogueFishing engine


#This engine file is responsible for creating the map and managing entities

#credits:
#general TCOD reference: https://python-tcod.readthedocs.io/en/latest/index.html
#roguelike tutorial: rogueliketutorials.com

#Used code from "Roguelike Tutorials" created by Tyler Standridge, website found at
rogueliketutorials.com with addendums/modifications by me

#imports

from __future__ import annotations

import lzma
import pickle

from typing import TYPE_CHECKING


from tcod.console import Console
from tcod.map import compute_fov
import exceptions

from message_log import MessageLog
```

```
import render_functions

if TYPE_CHECKING:
    from entity import Actor
    from game_map import GameMap, GameWorld


#This first bit manages the player and creates a pseudo list exclusive to TCOD that
handles entities.
#the pseudo list ensures one entity cannot be added to it multiple times
class Engine:
    game_map: GameMap
    #multiple maps achieved through caving
    game_world: GameWorld

    def __init__(self, player: Actor):
        self.message_log = MessageLog()
        self.mouse_location = (0, 0)
        self.player = player

    def handle_enemy_turns(self) -> None:
        for entity in set(self.game_map.actors) - {self.player}:
            if entity.ai:
                try:
                    entity.ai.perform()
                except exceptions.Impossible:
                    pass  # Ignore impossible action exceptions from AI.


    #This iterates through events

    def update_fov(self) -> None:
        """Recompute the visible area based on the players point of view."""
        self.game_map.visible[:] = compute_fov(
            self.game_map.tiles["transparent"],
            (self.player.x, self.player.y),
            radius=8,
        )
        # If a tile is "visible" it should be added to "explored".
        self.game_map.explored |= self.game_map.visible

    #draws entities, tiles and ui
    #VERY CRUCIAL/useful!!!
    def render(self, console: Console) -> None:
        self.game_map.render(console)

        self.message_log.render(console=console, x=21, y=45, width=40, height=5)
        console.print(
            x=1,
            y=46,
            string=f"HP: {self.player.fighter.hp}/{self.player.fighter.max_hp}",
        )
#shows cave level and entity names
        render_functions.render_cave_level(
            console=console,
            cave_level=self.game_world.current_floor,
            location=(0, 47),
        )

        render_functions.render_names_at_mouse_location(
            console=console, x=21, y=44, engine=self
        )

#this function manages savefiles
    def save_as(self, filename: str) -> None:
        """Save this Engine instance as a compressed file."""
        save_data = lzma.compress(pickle.dumps(self))
        with open(filename, "wb") as f:
            f.write(save_data)
```

**actions.py:**

```
#RogueFishing actions file



#This file allows for actions to be called from the main file

#credits:
#general TCOD reference: https://python-tcod.readthedocs.io/en/latest/index.html
#roguelike tutorial: rogueliketutorials.com

#Used code by "Roguelike Tutorials", website found at rogueliketutorials.com with slight
addendums/modifications by me

#allows actions to take some weight from engine

#imports

from __future__ import annotations

from typing import Optional, Tuple, TYPE_CHECKING
import color
import exceptions

if TYPE_CHECKING:
   from engine import Engine
   from entity import Actor, Entity, Item

class Action:
    def __init__(self, entity: Actor) -> None:
        super().__init__()
        self.entity = entity

    @property
    def engine(self) -> Engine:
        """Return the engine this action belongs to."""
        return self.entity.gamemap.engine

    def perform(self) -> None:
        """Perform this action with the objects needed to determine its scope.

        `self.engine` is the scope this action is being performed in.

        `self.entity` is the object performing the action.

        This method must be overridden by Action subclasses.
        """
        raise NotImplementedError()

class PickupAction(Action):
    """Pickup an item and add it to the inventory, if there is room for it."""

    def __init__(self, entity: Actor):
        super().__init__(entity)

    def perform(self) -> None:
        actor_location_x = self.entity.x
        actor_location_y = self.entity.y
        inventory = self.entity.inventory

        for item in self.engine.game_map.items:
            if actor_location_x == item.x and actor_location_y == item.y:
                if len(inventory.items) >= inventory.capacity:
                    raise exceptions.Impossible("No free slots in suit inventory.")

                self.engine.game_map.entities.remove(item)
                item.parent = self.entity.inventory
                inventory.items.append(item)
                self.engine.message_log.add_message(f"Acquired {item.name}!")
                return

        raise exceptions.Impossible("Your arms flail around foolishly as you grasp at thin
air.")

#consumable manager
class ItemAction(Action):
    def __init__(
        self, entity: Actor, item: Item, target_xy: Optional[Tuple[int, int]] = None
    ):
        super().__init__(entity)
        self.item = item
        if not target_xy:
            target_xy = entity.x, entity.y
        self.target_xy = target_xy

    @property
    def target_actor(self) -> Optional[Actor]:
        """Return the actor at this actions destination."""
        return self.engine.game_map.get_actor_at_location(*self.target_xy)

    def perform(self) -> None:
        """Invoke the items ability, this action will be given to provide context."""
        if self.item.consumable:
            self.item.consumable.activate(self)

class DropItem(ItemAction):
    def perform(self) -> None:
        if self.entity.equipment.item_is_equipped(self.item):
            self.entity.equipment.toggle_equip(self.item)
        self.entity.inventory.drop(self.item)

class EquipAction(Action):
    def __init__(self, entity: Actor, item: Item):
        super().__init__(entity)

        self.item = item

    def perform(self) -> None:
        self.entity.equipment.toggle_equip(self.item)

class WaitAction(Action):
    def perform(self) -> None:
        pass

class TakePassageAction(Action):
    def perform(self) -> None:
        """
        Take the passage, if any exist at the entity's location.
        """
        if (self.entity.x, self.entity.y) == self.engine.game_map.downstairs_location:
            self.engine.game_world.generate_floor()
            self.engine.message_log.add_message(
                "You crawl through the tunnel to find more caverns.", color.ascend
            )
        else:
            raise exceptions.Impossible("You can't crawl through there.")

class ActionWithDirection(Action):
    def __init__(self, entity: Actor, dx: int, dy: int):
        super().__init__(entity)

        self.dx = dx
        self.dy = dy

    @property
    def dest_xy(self) -> Tuple[int, int]:
        """Returns this actions destination."""
        return self.entity.x + self.dx, self.entity.y + self.dy

    @property
    def blocking_entity(self) -> Optional[Entity]:
        """Return the blocking entity at this actions destination.."""
        return self.engine.game_map.get_blocking_entity_at_location(*self.dest_xy)

    @property
    def target_actor(self) -> Optional[Actor]:
        """Return the actor at this actions destination."""
        return self.engine.game_map.get_actor_at_location(*self.dest_xy)

    def perform(self) -> None:
        raise NotImplementedError()

# basic melee attack
class MeleeAction(ActionWithDirection):
    def perform(self) -> None:
        target = self.target_actor
        if not target:
            raise exceptions.Impossible("Nothing to attack.")

        damage = self.entity.fighter.power - target.fighter.defense

        attack_desc = f"{self.entity.name.capitalize()} attacks {target.name}"
        if self.entity is self.engine.player:
            attack_color = color.player_atk
        else:
            attack_color = color.enemy_atk

        if damage > 0:
            self.engine.message_log.add_message(
                f"{attack_desc} for {damage} hit points.", attack_color
            )
            target.fighter.hp -= damage

        else:
            self.engine.message_log.add_message(
                f"{attack_desc} but does no damage.", attack_color
            )



class MovementAction(ActionWithDirection):
    def perform(self) -> None:
        dest_x, dest_y = self.dest_xy

        if not self.engine.game_map.in_bounds(dest_x, dest_y):
            # Destination is out of bounds.
            raise exceptions.Impossible("You bump into something, the cave is as 
disorienting as ever.")
        if not self.engine.game_map.tiles["walkable"][dest_x, dest_y]:
            # Destination is blocked by a tile.
            raise exceptions.Impossible("You bump into something, the cave is as 
disorienting as ever.")
        if self.engine.game_map.get_blocking_entity_at_location(dest_x, dest_y):
            # Destination is blocked by an entity.
            raise exceptions.Impossible("You bump into something, the cave is as 
disorienting as ever.")

        self.entity.move(self.dx, self.dy)

class BumpAction(ActionWithDirection):
    def perform(self) -> None:
        if self.target_actor:
            return MeleeAction(self.entity, self.dx, self.dy).perform()

        else:
            return MovementAction(self.entity, self.dx, self.dy).perform()
```

**color.py:**

```
#RogueFishing color manager file


#This file manages colors used in ui elements

#credits:
#general TCOD reference: https://python-tcod.readthedocs.io/en/latest/index.html
#roguelike tutorial: rogueliketutorials.com

#Used code from "Roguelike Tutorials" created by Tyler Standridge, website found at
rogueliketutorials.com with addendums/modifications by me


#these are color presets which are used in various UI elements
#these are called by different functions


white = (0xFF, 0xFF, 0xFF)
black = (0x0, 0x0, 0x0)
red = (0xFF, 0x0, 0x0)

player_atk = (0xE0, 0xE0, 0xE0)
enemy_atk = (0xFF, 0xC0, 0xC0)
needs_target = (0x3F, 0xFF, 0xFF)
status_effect_applied = (0x3F, 0xFF, 0x3F)
ascend = (0x9F, 0x3F, 0xFF)

player_die = (0xFF, 0x30, 0x30)
enemy_die = (0xFF, 0xA0, 0x30)

invalid = (0xFF, 0xFF, 0x00)
impossible = (0x80, 0x80, 0x80)
error = (0xFF, 0x40, 0x40)

welcome_text = (0x20, 0xA0, 0xFF)
health_recovered = (0x0, 0xFF, 0x0)


bar_text = white
bar_filled = (0x0, 0x60, 0x0)
bar_empty = (0x40, 0x10, 0x10)

menu_title = (0, 0, 0)
menu_text = white
```

**level.py:**

```
#RogueFishing
#Leveling system file

#This file manages the exp and leveling up systems

#credits:
#general TCOD reference: https://python-tcod.readthedocs.io/en/latest/index.html
#roguelike tutorial: rogueliketutorials.com

#Used code from "Roguelike Tutorials" created by Tyler Standridge, website found at
rogueliketutorials.com with addendums/modifications by me



#imports
from __future__ import annotations

from typing import TYPE_CHECKING

from components.base_component import BaseComponent

if TYPE_CHECKING:
   from entity import Actor

#default leveling info
class Level(BaseComponent):
   parent: Actor

   def __init__(
      self,
      current_level: int = 1,
      current_xp: int = 0,
      level_up_base: int = 0,
      level_up_factor: int = 150,
      xp_given: int = 0,
   ):
      self.current_level = current_level
      self.current_xp = current_xp
      self.level_up_base = level_up_base
      self.level_up_factor = level_up_factor
      self.xp_given = xp_given

   @property
   def experience_to_next_level(self) -> int:
      return self.level_up_base + self.current_level * self.level_up_factor

   @property
```

```
    def requires_level_up(self) -> bool:
        return self.current_xp > self.experience_to_next_level

    def add_xp(self, xp: int) -> None:
        if xp == 0 or self.level_up_base == 0:
            return

        self.current_xp += xp

        self.engine.message_log.add_message(f"You learn more from the caves and gain
{xp} experience points.")

        if self.requires_level_up:
            self.engine.message_log.add_message(
                f"You reach a breakthrough in your knowledge and achieve level
{self.current_level + 1}!"
            )

    def increase_level(self) -> None:
        self.current_xp -= self.experience_to_next_level

        self.current_level += 1

    def increase_max_hp(self, amount: int = 20) -> None:
        self.parent.fighter.max_hp += amount
        self.parent.fighter.hp += amount

        self.engine.message_log.add_message("Stress builds strength: your body has
taken so many beatings you grow more resilient! (+hp)")

        self.increase_level()

    def increase_power(self, amount: int = 1) -> None:
        self.parent.fighter.base_power += amount

        self.engine.message_log.add_message("You've killed so much you grow better at
it. Is this really nessecary? (+dam)")

        self.increase_level()

    def increase_defense(self, amount: int = 1) -> None:
        self.parent.fighter.base_defense += amount

        self.engine.message_log.add_message("Your skin callouses and hardens, enemy
blows hurt you less! (+def)")

        self.increase_level()
```

**inventory.py:**

```
#inventory manager file
#ALL comments by me

#This file manages the player's inventory

#credits:
#general TCOD reference: https://python-tcod.readthedocs.io/en/latest/index.html
#roguelike tutorial: rogueliketutorials.com

#Used code from "Roguelike Tutorials" created by Tyler Standridge, website found at
rogueliketutorials.com with addendums/modifications by me

#imports
from __future__ import annotations

from typing import List, TYPE_CHECKING

from components.base_component import BaseComponent

if TYPE_CHECKING:
    from entity import Actor, Item


class Inventory(BaseComponent):
    parent: Actor

    def __init__(self, capacity: int):
        self.capacity = capacity
        self.items: List[Item] = []

    def drop(self, item: Item) -> None:
        """
        Removes an item from the inventory and restores it to the game map, at the
player's current location.
        """
        self.items.remove(item)
        item.place(self.parent.x, self.parent.y, self.gamemap)

        self.engine.message_log.add_message(f"You dropped {item.name}.")
```

**fighter.py:**

```
#RogueFishing player stats and statuses management file
```

```
#This file controls the player and their attributes

#credits:
#general TCOD reference: https://python-tcod.readthedocs.io/en/latest/index.html
#roguelike tutorial: rogueliketutorials.com

#Used code from "Roguelike Tutorials" created by Tyler Standridge, website found at
rogueliketutorials.com with addendums/modifications by me

#imports
from __future__ import annotations

from typing import TYPE_CHECKING
import color

#added
import random

from components.base_component import BaseComponent

from render_order import RenderOrder

if TYPE_CHECKING:
    from entity import Actor

#this allows "classes" to be created and player/actor attributes to be managed


class Fighter(BaseComponent):
    parent: Actor
    def __init__(self, hp: int, base_defense: int, base_power: int):
        self.max_hp = hp
        self._hp = hp
        self.base_defense = base_defense
        self.base_power = base_power

    @property
    def hp(self) -> int:
        return self._hp

    @hp.setter
    def hp(self, value: int) -> None:
        self._hp = max(0, min(value, self.max_hp))
        if self._hp == 0 and self.parent.ai:
            self.die()

    @property
    def defense(self) -> int:
        return self.base_defense + self.defense_bonus

    @property
    def power(self) -> int:
        return self.base_power + self.power_bonus

    @property
    def defense_bonus(self) -> int:
        if self.parent.equipment:
            return self.parent.equipment.defense_bonus
        else:
            return 0

    @property
    def power_bonus(self) -> int:
        if self.parent.equipment:
            return self.parent.equipment.power_bonus
        else:
            return 0

    #allows players and npcs to die, chooses from multiple possible messages and prints
them to the log
    def die(self) -> None:
        #by me :0
        RandInt1 = random.randint(1, 12)
        RandInt2 = random.randint(1, 11)

        if self.engine.player is self.parent:
            if RandInt2 == 1:
                death_message = "As your last breath escapes your mangled body, only one
light remains in the void that engulfs you: hope."
                death_message_color = color.player_die

            elif RandInt2 == 2:
                death_message = "You choked on fear and made a grave error. There is
always a next chance!"
                death_message_color = color.player_die

            elif RandInt2 == 3:
                death_message = "They just got lucky, you'll be better next time."
                death_message_color = color.player_die

            elif RandInt2 == 4:
                death_message = "The pain is so intense that you lose consiousness for the
last time, at least in this life."
                death_message_color = color.player_die
```

```
          elif RandInt2 == 5:
             death_message = "Final memories of the surface flash before your eyes,
hopefully when you return the radiation will be gone."
             death_message_color = color.player_die

          elif RandInt2 == 6:
             death_message = "You did so well, I'm proud of you. I'll see you next time."
             death_message_color = color.player_die

          elif RandInt2 == 7:
             death_message = "No one said it would be easy, you need to try harder, you
can do it!"
             death_message_color = color.player_die

          elif RandInt2 == 8:
             death_message = "The human body can only take so much: what's left of you
lifelessly slumps to the ground. You'll be back though. "
             death_message_color = color.player_die

          elif RandInt2 == 9:
             death_message = "Time has Little to do With Infinity and Jelly Dougnuts."
             death_message_color = color.player_die

          elif RandInt2 == 10:
             death_message = "You see your loved ones in the beautiful sunlight. One day
you'll get to the surface for real."
             death_message_color = color.player_die

          elif RandInt2 == 11:
             death_message = "Death smiles at us all. All a man can do is smile back."
             death_message_color = color.player_die




       else:
          if RandInt1 == 1:
             death_message = f"{self.parent.name} won't be able to fulfill their dreams."
             death_message_color = color.enemy_die
          elif RandInt1 == 2:
             death_message = f"{self.parent.name} is dead. You did this."
             death_message_color = color.enemy_die
          elif RandInt1 == 3:
             death_message = f"You didn't let {self.parent.name} run."
             death_message_color = color.enemy_die
          elif RandInt1 == 4:
             death_message = f"{self.parent.name} was alive just a moment ago, not
anymore thanks to you."
             death_message_color = color.enemy_die
          elif RandInt1 == 5:
             death_message = f"{self.parent.name} tried to stop you, it was self defense,
right?"
             death_message_color = color.enemy_die
          elif RandInt1 == 6:
             death_message = f"{self.parent.name}'s flailing about finally stops. Did you
make the right choice?"
             death_message_color = color.enemy_die
          elif RandInt1 == 7:
             death_message = f"You murdered {self.parent.name}, they'll haunt you for the
rest of your lives."
             death_message_color = color.enemy_die
          elif RandInt1 == 8:
             death_message = f"You are covered with {self.parent.name}'s remains,
disgusting."
             death_message_color = color.enemy_die
          elif RandInt1 == 9:
             death_message = f"It was you or {self.parent.name}. You were stronger."
             death_message_color = color.enemy_die
          elif RandInt1 == 10:
             death_message = f"You kill {self.parent.name} for the empire."
             death_message_color = color.enemy_die
          elif RandInt1 == 11:
             death_message = f"You thought {self.parent.name} was stronger."
             death_message_color = color.enemy_die
          elif RandInt1 == 12:
             death_message = f'"How many more will I have to kill" you think to yourself as
you finish off {self.parent.name}.'
             death_message_color = color.enemy_die

       #creates "corpse"
       self.parent.char = "%"
       self.parent.color = (191, 0, 0)
       self.parent.blocks_movement = False
       self.parent.ai = None
       self.parent.name = f"What remains of {self.parent.name}, do you feel guilt?"
       self.parent.render_order = RenderOrder.CORPSE

       self.engine.message_log.add_message(death_message, death_message_color)
       self.engine.player.level.add_xp(self.parent.level.xp_given)

    #this function heals an entity!
    def heal(self, amount: int) -> int:
       if self.hp == self.max_hp:
          return 0

       new_hp_value = self.hp + amount

       if new_hp_value > self.max_hp:
          new_hp_value = self.max_hp

       amount_recovered = new_hp_value - self.hp

       self.hp = new_hp_value

       return amount_recovered

    def take_damage(self, amount: int) -> None:
       self.hp -= amount
```

**equippable.py:**

```
#Euippable file


#This file manages the ability of items to be equipped

#credits:
#general TCOD reference: https://python-tcod.readthedocs.io/en/latest/index.html
#roguelike tutorial: rogueliketutorials.com

#Used code from "Roguelike Tutorials" created by Tyler Standridge, website found at
rogueliketutorials.com with addendums/modifications by me



#imports
from __future__ import annotations

from typing import TYPE_CHECKING

from components.base_component import BaseComponent
from equipment_types import EquipmentType

if TYPE_CHECKING:
    from entity import Item


class Equippable(BaseComponent):
    parent: Item
    def __init__(
        self,
        equipment_type: EquipmentType,
        power_bonus: int = 0,
        defense_bonus: int = 0,
    ):
        self.equipment_type = equipment_type

        self.power_bonus = power_bonus
        self.defense_bonus = defense_bonus

#manages equipment
class Shortsword(Equippable):
    def __init__(self) -> None:
        super().__init__(equipment_type=EquipmentType.WEAPON, power_bonus=2)


class Cleaver(Equippable):
    def __init__(self) -> None:
        super().__init__(equipment_type=EquipmentType.WEAPON, power_bonus=4)


class Gambeson(Equippable):
    def __init__(self) -> None:
        super().__init__(equipment_type=EquipmentType.ARMOR, defense_bonus=1)


class FlakVest(Equippable):
    def __init__(self) -> None:
        super().__init__(equipment_type=EquipmentType.ARMOR, defense_bonus=3)
```

**equipment.py:**

```
#equpment

#This file manages equipment

#credits:
#general TCOD reference: https://python-tcod.readthedocs.io/en/latest/index.html
#roguelike tutorial: rogueliketutorials.com

#Used code from "Roguelike Tutorials" created by Tyler Standridge, website found at
rogueliketutorials.com with addendums/modifications by me


#imports
from __future__ import annotations
from typing import Optional, TYPE_CHECKING

from components.base_component import BaseComponent
from equipment_types import EquipmentType

if TYPE_CHECKING:
   from entity import Actor, Item

#Slots
#more things can be added here like off hand/detailed armor
class Equipment(BaseComponent):
   parent: Actor

   def __init__(self, weapon: Optional[Item] = None, armor: Optional[Item] = None):
      self.weapon = weapon
      self.armor = armor

#these calculate stat bonuses granted by equipment
   @property
   def defense_bonus(self) -> int:
      bonus = 0

      if self.weapon is not None and self.weapon.equippable is not None:
         bonus += self.weapon.equippable.defense_bonus

      if self.armor is not None and self.armor.equippable is not None:
         bonus += self.armor.equippable.defense_bonus

      return bonus

   @property
   def power_bonus(self) -> int:
      bonus = 0

      if self.weapon is not None and self.weapon.equippable is not None:
         bonus += self.weapon.equippable.power_bonus

      if self.armor is not None and self.armor.equippable is not None:
         bonus += self.armor.equippable.power_bonus

      return bonus

#Check if something is equipped
   def item_is_equipped(self, item: Item) -> bool:
      return self.weapon == item or self.armor == item
```

```
#messages for equipping/unequipping
    def unequip_message(self, item_name: str) -> None:
        self.parent.gamemap.engine.message_log.add_message(
            f"You unequip the {item_name}."
        )

    def equip_message(self, item_name: str) -> None:
        self.parent.gamemap.engine.message_log.add_message(
            f"You equip the {item_name}."
        )

    def equip_to_slot(self, slot: str, item: Item, add_message: bool) -> None:
        current_item = getattr(self, slot)

        if current_item is not None:
            self.unequip_from_slot(slot, add_message)

        setattr(self, slot, item)

        if add_message:
            self.equip_message(item.name)

    def unequip_from_slot(self, slot: str, add_message: bool) -> None:
        current_item = getattr(self, slot)

        if add_message:
            self.unequip_message(current_item.name)

        setattr(self, slot, None)

#equip things in slots created in above part
    def toggle_equip(self, equippable_item: Item, add_message: bool = True) -> None:
        if (
            equippable_item.equippable
            and equippable_item.equippable.equipment_type == EquipmentType.WEAPON
        ):
            slot = "weapon"
        else:
            slot = "armor"

        if getattr(self, slot) == equippable_item:
            self.unequip_from_slot(slot, add_message)
        else:
            self.equip_to_slot(slot, equippable_item, add_message)
```

**Changelog.txt:**

RogueFishing
——————————————————————————————————————————————

Build naming convention:
(Month)(Day)(Year)(Build designation)
Ex: 010203a
“i” is representative of an initial build of a NEW version.
After “i” comes a-h then j-z
After z comes aa, bb, cc etc. (including ii)

- Versions, starting from v0.1, denote major change.
- Multiple builds released in one day will get more letters, ex: 010203a, 010203b.
- The date is in relation to when the build was first started!

——————————————————————————————————————————————

Changelog:

v0.1, 030525i :
Laid down the basic foundations for creating the game:
- created the player character, represented with @
- spawned the player
- made the player controllable
- made multiple specialized files to perform tasks such as drawing the player character to controlling it

v0.1, 030525a :
Major overhaul to EVERYTHING!!!
- created new files, entity.py and engine.py responsible for setting up entities and managing them as well as creating maps respectively.
- modified main.py to be more lightweight and delegate tasks
- created tile_types.py, this file manages tiles and tile properties
- created game_map.py which manages the game map (doesn’t draw it, that’s engine’s job)
- modified main.py to support the newly created engine and map
    -main.py’s modifications allow for a map to be created with certain parameters, like length/width!
- modified the newly created engine to support the gamemap file
- engine handles the map like this: handle_events is called to determine details about a tile and render draws it

v0.1, 030625a :
- changed changelog from rtf to txt
- updated changeling build convention info

v0.1, 030725a :
- Modified "dejavu" font elements, characters changed:
    -@, changed to a character

    -$, changed to fish

v0.11, 030825i :
- Functional procedural map generation is implemented!
- addeded procgen.py to handle procedural generation
- modified main.py to accomidate procgen
- modified game_map.py to accomidate procgen
- The game will now procedurally generate a map when main.py is run. The player will be intellignetly placed on a valid tile.

v0.11, 030925a :
- modified game_map.py (using arrays) calculates FOV and stores discovered portions of the caves, this heavily utilizes TCOD features!
- modified tyle_types.py to add new "light" and "dark" tiles, these tiles are for parts of the dungeon that have been discovered or are yet to be discovered/ in sight and out of sight
- added shroud which acts as a fog of war for undiscovered parts
- modified the game engine to support the new light and dark tiles as well as the shroud mechanic

v0.11, 031025a -Intelligent entity spawning :
- Entities now spawn inside rooms!
- Basic framework for attacks and enemy moves set up!
- Modified engine.py to accomidate new entities
- modified main.py to remove placeholder entities and old spawning system
- modified engine.py to remove entity rendering
    -gamemap.py now is responisble for this
- modified procgen.py to associate the player with entities
- added new parameters to the procgen system in main.py (max entities per room)
- Modified entity.py to support the new entity spawning system
- created entity_factories.py, which acts as a conduit for entities and allows for the various entities to be defined and customized!
- added basic melee attack abilities in actions.py
- changed action type in input_handlers.py from "MovementAction" to "BumpAction" to support more potential interactions such as attacks

v0.12, 031225i - Rework and attack basics :
- The tutorial used some outdated syntax, these fixes should stop non fatal futureproofing
errors
- This should also optimize the game and make it run significantly better
- Set up groundwork for combat
- Reworked/futureproofed actions.py
- Reworked/futureproofed input_handlers.py
- Reworked/futureproofed game_map.py
- Reworked/futureproofed main.py
- Reworked/futureproofed entity.py
- Reworked/futureproofed procgen.py

- Reworked/futureproofed engine.py
- Created new components folder in the root, this contains information relating to combat and enemy ai
- Modified entity.py to accomidate new AI
- added new hostile ai framework
- modified entity_factories to add new parameters for NPC types such as ai type
- modified npc ai to only attack in 4 cardinal directions
- made attacks deal damage
- made the player and enemies able to die
- added render_order.py which allows us to determine the order in which npcs/entities are drawn
- added basic health bar
- Death config in fighter.py
- Player attribute framework in fighter.py

v0.12, 031325a
- Modified credits across most files to be more descriptive
- Commented up the files in the folder "components" which were added in v.12 031225i
- Commented up entity_factories.py to make it's new capabilities more clear
- Modified fontsheet to add new custom entity sprites

v0.12, 031325b - UI and modernization changes
- added color.py which manages color for ui elements
- added message_log.py which outputs terminal messages ingame to emulate classic roguelikes
- modified engine.py to support the new ingame log
- modified main.py to support new log
- modified fighter.py to support new log
- modified actions.py to support new log
- updated syntax in engine.py
- added the ability to store mouse location in engine.py
- mouse support and function to display info about entities in the log when the mouse is over them!
- most of mouse code is in input_handlers.py
- scroll through log possible!
- press "v" to see full log!

v0.12, 031425a - Inventory and items!
-NOTE TO ME: GO HERE TO ADD MORE ENTITIES!
- modified (modernized i guess) ai.py to remove errors (in components folder)
- modified message_log.py to remove errors
- modified game_map.py to remove errors
- modified entity.py
- modified base_component.py (in components folder)
- modified fighter.py (in components folder)
- addded new colors to color.py for UI
- created healing function in fighter.py

- created execeptions.py to manage special/unusual events...or exceptions!
- modified main.py to accomidate new exceptions
- modified input_handlers.py to accomidate new exceptions
- modified engine.py to accomidate new exceptions
- modified entity.py to support a new entity subclass for items
    - I will probably do something similar for making water!
- added consumable support to actions.py
- added health potion to entity_factories.py
- modified PROCGEN.PY to now spawn health potions!
- modified main.py to add new item spawning parameters
- created inventory.py to manage inventory
- modified entity.py to support inventory
- added new inventory parameter to entity_factories.py
- modified actions.py to allow items to be picked up
- G key picks stuff up!
- D key drops items!
- I opens inventory

v0.12, 031725a
- added credits.txt to make credited authors more clear
- modified readme.md to update copyright information

v0.12, 031725b
- changed placeholder "troll" and "orc" into "Cave Horror" and "Fallen Fisher"
- Tweaked world-gen settings, the caves should now have far more diversity and complexity in generation

v0.12, 031725c
- added randomly generated messages!
    - these messages are currently used for kills and deaths
- created a general use function based on random that prints random messages, these can be used for anything!

v0.12, 031725d
- modified line of code in fighter.py which would produce an error when a certain number was generated for a death message

v0.12, 031725e - ranged weapons
- added new UI colors to color.py
- Added single shot muskets that deal massive damage
- added new musket sprite to the spritesheet
- added "enter" keybind class, these keys confirm actions
- added targeting controls for ranged weapons
- new "look" mode controlled by mouse which is activated by pressing "/"
- added new single shot blunderbuss with an AOE attack, be careful, due to the nature of the caves if you shoot close to yourself your bound to be hit by ricochets!

v0.12, 031925a - Save Files
- added new colors to color.py
- added new action handler system to input_handlers.py, this part is completely above me but it should allow for save file stuff to work.
- modified main.py to support new menus/saves n such
- added new file: setup_game.py
    - this file manages the main menu!
- modified engine.py to support new save system
- removed parameters from main.py
    - generally trimmed main.py down significantly
- added save and load functions to setup_game.py
- added lzma, pickle anddddddd traceback to support file saving/loading
- created RogueFishing.gitignore for my own dev uses
- Game setup parameters moved to setup_game.py
- TO DO:
    - make menu nicer
    - random splash text!
    - make the menu screen art not look god awful
    - make more sprites
    - make placeholder friendly NPC using confused AI
    - make musket use different mechanics (current ones are abysmal lol)

v0.13, 032025i - Exploration and Progression!
- added new title screen that replaces the WIP version
- added new colors
- modified old colors
- > is set as the new passageway
- push ">" to travel through the passage (you may have to hold shift)
- this new passageway is in tile_types.py
- modified both procgen.py and game_map.py to spawn the new passages
- modifed engine.py to add the new cave section
- actions.py was modifed to enable passage system
- render_functions.py modified
- the current floor is currently displayed
- added new level.py file
    - this manages exp and the leveling system
- modified entity.py to use new leveling system
- modified entity_factories.py to use new leveling system
- added character information to input_handlers.py
    -press "c" to see this

v0.13, 032025a - minor changes

v0.13, 032725a - increasing difficulty (Based on part 12 of roguelike tutorials)
- modified procgen.py
    - used tuples to change entity spawns on each "floor"
- new system makes previous spawning systems outdated

  - removed traces of old spawning systems:
    - gamemap.py
    - procgen.py
    - setup_game.py
- added spawining parameters to procgen.py
- added spawning increase, ex more of a certain thing spawns the higher you go in the caves
- will tweak and customize these in next update, for now they're the default numbers from the tutoral.

v0.13, 032825 - equipment framework
- added basic armor framework
- created equipment_types.py to manage new equipment
- created equippable.py to manage gear
- modified entity.py and actions.py to accomidate new equippables
- equippable is now used with equippable for entity_factories.py
- created equipment.py
  - handles entitiy equipment and all equipment details
- modified entity.py to support equipment.py
- modified fighter.py to accomidate equippables
- defense and attack are calculated differently
- updated level.py to use new args
- modified procgen.py to spawn new equipment
- updated actions.py to allow equipping/unequipping
- actual equipment is in equipable.py
- updated sprites
- low tier equipment is automatically granted on spawn
- graphics update!
- New Static entity type
  - this is the framework for water and other immovable things like chests, doors etc!
- Updated random system
  - used for splash texts in setup_game.py
  - this is my main student created function!
- Updated credits.txt
- Updated credits/opening in all files to be compliant.

#end of line

dejavu10x10_gs_tc.png:

! " & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : < = > ?
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z

New Piskel.png: