

```
import random
import time
import sys
from enum import Enum

class ClassType(Enum):
    WARRIOR = "Воин"
    MAGE = "Маг"
    ROGUE = "Плут"

class LocationType(Enum):
    ENCOUNTER = "Встреча с врагом"
    CHEST = "Сундук"
    REST = "Место отдыха"

class ItemType(Enum):
    WEAPON = "Оружие"
    ARMOR = "Броня"
    POTION = "Зелье"

class Rarity(Enum):
    COMMON = "Обычный"
    RARE = "Редкий"
    EPIC = "Эпический"
    LEGENDARY = "Легендарный"

class GameState(Enum):
    PLAYING = 0
    VICTORY = 1
    DEFEAT = 2

class Item:
    def __init__(self, name, item_type, rarity, stats):
        self.name = name
        self.type = item_type
        self.rarity = rarity
        self.stats = stats

    def __str__(self):
        return f"{self.rarity.value} {self.type.value}: {self.name}"

class Enemy:
    def __init__(self, name, level, health, attack, defense, exp_reward):
        self.name = name
        self.level = level
        self.health = health
        self.max_health = health
        self.attack = attack
```

```
self.defense = defense
self.exp_reward = exp_reward
```

```
def is_alive(self):
    return self.health > 0
```

```
class Player:
```

```
    def __init__(self, name, class_type):
```

```
        self.name = name
```

```
        self.class_type = class_type
```

```
        self.level = 1
```

```
        self.exp = 0
```

```
        self.exp_to_next_level = 100
```

```
        self.health = 100
```

```
        self.max_health = 100
```

```
        self.attack = 10
```

```
        self.defense = 10
```

```
        self.dodge_chance = 5
```

```
        self.inventory = []
```

```
        self.equipped_weapon = None
```

```
        self.equipped_armor = None
```

```
        self.stat_points = 0
```

```
    # Начальные предметы в зависимости от класса
```

```
    self._initialize_class()
```

```
    def _initialize_class(self):
```

```
        if self.class_type == ClassType.WARRIOR:
```

```
            self.health = 150
```

```
            self.max_health = 150
```

```
            self.attack = 15
```

```
            self.defense = 15
```

```
            self.inventory.append(Item("Стальной меч", ItemType.WEAPON, Rarity.COMMON,
{"attack": 5}))
```

```
        elif self.class_type == ClassType.MAGE:
```

```
            self.health = 80
```

```
            self.max_health = 80
```

```
            self.attack = 25
```

```
            self.defense = 5
```

```
            self.inventory.append(Item("Посох новичка", ItemType.WEAPON, Rarity.COMMON,
{"attack": 8}))
```

```
        elif self.class_type == ClassType.ROGUE:
```

```
            self.health = 100
```

```
            self.max_health = 100
```

```
            self.attack = 20
```

```
            self.defense = 10
```

```
            self.dodge_chance = 15
```

```
self.inventory.append(Item("Кинжалы теней", ItemType.WEAPON, Rarity.COMMON,
{"attack": 6}))
```

```
def add_exp(self, amount):
    self.exp += amount
    print(f"Получено {amount} опыта!")
    while self.exp >= self.exp_to_next_level:
        self.level_up()
```

```
def level_up(self):
    self.level += 1
    self.exp -= self.exp_to_next_level
    self.exp_to_next_level = int(self.exp_to_next_level * 1.5)
    self.stat_points += 5
    self.max_health += 20
    self.health = self.max_health
```

```
print(f"\n=== Достигнут уровень {self.level}! ===")
print("Здоровье увеличено!")
self.show_stats()
self.distribute_stat_points()
```

```
def distribute_stat_points(self):
    while self.stat_points > 0:
        print(f"\nОсталось очков характеристик: {self.stat_points}")
        print("1: +10 к здоровью")
        print("2: +5 к атаке")
        print("3: +5 к защите")
        print("4: +2% к уклонению")
```

```
choice = input("Выберите улучшение: ")
if choice == "1":
    self.max_health += 10
    self.health += 10
    self.stat_points -= 1
elif choice == "2":
    self.attack += 5
    self.stat_points -= 1
elif choice == "3":
    self.defense += 5
    self.stat_points -= 1
elif choice == "4":
    self.dodge_chance += 2
    self.stat_points -= 1
else:
    print("Неверный выбор!")
```

```
def show_stats(self):
```

```

print(f"\n=== Характеристики {self.name} ===")
print(f"Класс: {self.class_type.value}")
print(f"Уровень: {self.level}")
print(f"Опыт: {self.exp}/{self.exp_to_next_level}")
print(f"Здоровье: {self.health}/{self.max_health}")
print(f"Атака: {self.attack}")
print(f"Защита: {self.defense}")
print(f"Уклонение: {self.dodge_chance}%")
if self.equipped_weapon:
    print(f"Оружие: {self.equipped_weapon}")
if self.equipped_armor:
    print(f"Броня: {self.equipped_armor}")

def use_item(self, item):
    if item.type == ItemType.POTION:
        heal_amount = item.stats.get("heal", 0)
        self.health = min(self.max_health, self.health + heal_amount)
        print(f"Использовано {item.name}! Восстановлено {heal_amount} здоровья.")
        self.inventory.remove(item)
    elif item.type == ItemType.WEAPON:
        self.equipped_weapon = item
        print(f"Экипировано оружие: {item.name}")
    elif item.type == ItemType.ARMOR:
        self.equipped_armor = item
        print(f"Экипирована броня: {item.name}")

def calculate_damage(self):
    base_attack = self.attack
    if self.equipped_weapon:
        base_attack += self.equipped_weapon.stats.get("attack", 0)
    return base_attack + random.randint(0, 5)

def calculate_defense(self):
    base_defense = self.defense
    if self.equipped_armor:
        base_defense += self.equipped_armor.stats.get("defense", 0)
    return base_defense

class Battle:
    def __init__(self, player, enemy):
        self.player = player
        self.enemy = enemy

    def start(self):
        print(f"\n=== Бой с {self.enemy.name}! ===")

        while self.player.is_alive() and self.enemy.is_alive():
            self.player_turn()

```

```

        if not self.enemy.is_alive():
            break
        self.enemy_turn()

    if self.player.is_alive():
        print(f"\nПобеда над {self.enemy.name}!")
        self.player.add_exp(self.enemy.exp_reward)
        return True
    else:
        print("\nВы пали в бою...")
        return False

def player_turn(self):
    print(f"\nВаше здоровье: {self.player.health}/{self.player.max_health}")
    print(f"Здоровье {self.enemy.name}: {self.enemy.health}/{self.enemy.max_health}")

    while True:
        print("\nВыберите действие:")
        print("1: Атака")
        print("2: Использовать предмет")
        print("3: Попытаться уклониться")

        choice = input("Ваш выбор: ")

        if choice == "1":
            self.attack()
            break
        elif choice == "2":
            self.use_item()
            break
        elif choice == "3":
            self.dodge_attempt()
            break
        else:
            print("Неверный выбор!")

def attack(self):
    damage = self.player.calculate_damage() - self.enemy.defense // 2
    damage = max(1, damage)
    self.enemy.health -= damage
    print(f"Вы нанесли {damage} урона!")

def use_item(self):
    if not self.player.inventory:
        print("Инвентарь пуст!")
        return self.player_turn()

    print("\nИнвентарь:")

```

```

for i, item in enumerate(self.player.inventory):
    print(f"{i+1}: {item}")

try:
    choice = int(input("Выберите предмет: ")) - 1
    if 0 <= choice < len(self.player.inventory):
        self.player.use_item(self.player.inventory[choice])
    else:
        print("Неверный выбор!")
        self.use_item()
except ValueError:
    print("Введите число!")
    self.use_item()

```

```

def dodge_attempt(self):
    if random.randint(1, 100) <= self.player.dodge_chance:
        print("Вы успешно уклонились от атаки!")
        return True
    else:
        print("Уклонение не удалось!")
        return False

```

```

def enemy_turn(self):
    damage = self.enemy.attack - self.player.calculate_defense() // 2
    damage = max(1, damage)
    self.player.health -= damage
    print(f"{self.enemy.name} наносит вам {damage} урона!")

```

```

class Location:
    def __init__(self, name, description, location_type):
        self.name = name
        self.description = description
        self.type = location_type
        self.connections = []

```

```

def add_connection(self, location):
    self.connections.append(location)

```

```

def explore(self, player):
    print(f"\n=== {self.name} ===")
    print(self.description)

```

```

if self.type == LocationType.ENCOUNTER:
    return self.handle_encounter(player)
elif self.type == LocationType.CHEST:
    return self.handle_chest(player)
elif self.type == LocationType.REST:
    return self.handle_rest(player)

```

```

return GameState.PLAYING

def handle_encounter(self, player):
    enemies = [
        Enemy("Страж Бездны", player.level, 50 + player.level * 10,
              15 + player.level * 2, 10 + player.level, 50),
        Enemy("Небесный Каратель", player.level, 40 + player.level * 8,
              20 + player.level * 2, 5 + player.level, 60),
        Enemy("Хранитель Времени", player.level, 60 + player.level * 12,
              12 + player.level * 2, 15 + player.level, 70)
    ]

    enemy = random.choice(enemies)
    battle = Battle(player, enemy)
    return GameState.PLAYING if battle.start() else GameState.DEFEAT

def handle_chest(self, player):
    items = [
        Item("Зелье здоровья", ItemType.POTION, Rarity.COMMON, {"heal": 50}),
        Item("Меч небес", ItemType.WEAPON, Rarity.RARE, {"attack": 15}),
        Item("Доспех вечности", ItemType.ARMOR, Rarity.EPIC, {"defense": 20}),
        Item("Эликсир боров", ItemType.POTION, Rarity.LEGENDARY, {"heal": 200})
    ]

    item = random.choice(items)
    print(f"Вы нашли: {item}!")
    player.inventory.append(item)

    if item.type == ItemType.POTION:
        use = input("Использовать сейчас? (y/n): ").lower()
        if use == 'y':
            player.use_item(item)

    return GameState.PLAYING

def handle_rest(self, player):
    heal_amount = player.max_health // 2
    player.health = min(player.max_health, player.health + heal_amount)
    print(f"Вы отдохнули и восстановили {heal_amount} здоровья!")
    player.show_stats()
    return GameState.PLAYING

class Game:
    def __init__(self):
        self.player = None
        self.locations = []
        self.current_location = None

```

```

self.game_state = GameState.PLAYING
self._create_locations()

def _create_locations(self):
    # Создание локаций
    locations_data = [
        ("Врата Хроноспирали", "Древние врата, ведущие к богам времени",
LocationType.ENCOUNTER),
        ("Зал Прошлого", "Фрески изображают былые времена", LocationType.CHEST),
        ("Мост Судьбы", "Радужный мост через бездну", LocationType.ENCOUNTER),
        ("Сад Вечности", "Место покоя и восстановления", LocationType.REST),
        ("Чертоги Богов", "Здесь обитают повелители времени",
LocationType.ENCOUNTER),
        ("Сокровищница Времени", "Легендарные артефакты", LocationType.CHEST),
        ("Престол Хроноса", "Финальное испытание", LocationType.ENCOUNTER)
    ]

    for name, desc, loc_type in locations_data:
        self.locations.append(Location(name, desc, loc_type))

    # Создание связей между локациями
    for i in range(len(self.locations)-1):
        self.locations[i].add_connection(self.locations[i+1])

def start(self):
    self.show_intro()
    self.create_character()
    self.current_location = self.locations[0]

    while self.game_state == GameState.PLAYING:
        self.game_state = self.current_location.explore(self.player)

        if self.game_state == GameState.PLAYING:
            if self.current_location == self.locations[-1]:
                self.game_state = GameState.VICTORY
                break

        self.show_travel_options()

    self.show_ending()

def show_intro(self):
    print("=" * 60)
    print("ХРОНОСПИРАЛЬ")
    print("=" * 60)
    print("\nВы - изгой, бросивший вызов богам времени.")
    print("Поднимитесь по Хроноспирали и свергните повелителей вечности!")
    print("Судьба вселенной в ваших руках...")

```

```

print("\n" + "=" * 60)

def create_character(self):
    print("\nСОЗДАНИЕ ПЕРСОНАЖА")
    name = input("Введите имя героя: ")

    print("\nВыберите класс:")
    for i, class_type in enumerate(ClassType, 1):
        print(f"{i}: {class_type.value}")

    while True:
        try:
            choice = int(input("Ваш выбор: "))
            if 1 <= choice <= len(ClassType):
                class_type = list(ClassType)[choice-1]
                break
            else:
                print("Неверный выбор!")
        except ValueError:
            print("Введите число!")

    self.player = Player(name, class_type)
    print(f"\nДобро пожаловать, {name} - {class_type.value}!")
    self.player.show_stats()

def show_travel_options(self):
    print("\nКуда вы хотите отправиться?")
    for i, location in enumerate(self.current_location.connections, 1):
        print(f"{i}: {location.name}")

    try:
        choice = int(input("Ваш выбор: ")) - 1
        if 0 <= choice < len(self.current_location.connections):
            self.current_location = self.current_location.connections[choice]
        else:
            print("Неверный выбор!")
            self.show_travel_options()
    except ValueError:
        print("Введите число!")
        self.show_travel_options()

def show_ending(self):
    if self.game_state == GameState.VICTORY:
        print("\n" + "=" * 60)
        print("ПОБЕДА!")
        print("=" * 60)
        print("\nВы свергли богов времени и стали повелителем Хроноспирали!")
        print("Вселенная обрела нового хранителя...")

```

```
        print(f"\n{self.player.name}, ваше имя будет помнить вечность!")
    else:
        print("\nВаше путешествие окончено...")
        print("Но легенды о вас будут передаваться из уст в уста.")

if __name__ == "__main__":
    game = Game()
    game.start()
```