

Run ID,Agent Name,Task,Result,Timestamp

8edef031-5f31-4584-a019-f56005f9a998,Pattern Exploration Agent 1,Create 10 unique games in Python using novel game design patterns. Each agent should output the Python code for their game and only their game.,```\python

```
import random
```

```
import time
```

```
class GameObject:
```

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

```
        self.name = name
```

```
        self.position = position
```

```
    def move(self, new_position):
```

```
        self.position = new_position
```

```
class Environment:
```

```
    def __init__(self, size):
```

```
        self.size = size
```

```
        self.objects = []
```

```
    def add_object(self, game_object):
```

```
        self.objects.append(game_object)
```

```
    def update(self):
```

```
        for obj in self.objects:
```

```
            if random.random() < 0.5: # Randomly move objects
```

```
new_position = (random.randint(0, self.size - 1), random.randint(0, self.size - 1))
```

```
obj.move(new_position)
```

```
def display(self):
```

```
    grid = [['.' for _ in range(self.size)] for _ in range(self.size)]
```

```
    for obj in self.objects:
```

```
        x, y = obj.position
```

```
        grid[x][y] = obj.name[0] # Use first letter of the object's name
```

```
    for row in grid:
```

```
        print(' '.join(row))
```

```
    print()
```

```
class Game:
```

```
    def __init__(self, size):
```

```
        self.environment = Environment(size)
```

```
        self.score = 0
```

```
    def add_game_object(self, name):
```

```
        position = (random.randint(0, self.environment.size - 1), random.randint(0, self.environment.size - 1))
```

```
        game_object = GameObject(name, position)
```

```
        self.environment.add_object(game_object)
```

```
    def play(self):
```

```
        print("""Welcome to the Dynamic Environment Game!""")
```

```
        for _ in range(10): # Play for 10 rounds
```

```
self.environment.update()

self.environment.display()

self.score += 1

time.sleep(1) # Pause for a moment to see the changes

print(f"Game Over! Your score: {self.score}")
```

```
if __name__ == "__main__":
```

```
    game = Game(size=5)
```

```
    game.add_game_object("Player")
```

```
    game.add_game_object("Enemy")
```

```
    game.add_game_object("Treasure")
```

```
    game.play()
```

```
```,2024-08-24T18:39:39.929561
```

8edef031-5f31-4584-a019-f56005f9a998,Pattern Exploration Agent 2,Create 10 unique games in Python using novel game design patterns. Each agent should output the Python code for their game and only their game.,``python

```
import random
```

```
class Game:
```

```
 def __init__(self):
```

```
 self.story = {
```

```
 "start": {
```

```
 "text": "You wake up in a mysterious forest. What do you want to do?",
```

```
 "options": {
```

```
 "explore": "explore_forest",
```

```
 "build_shelter": "build_shelter"
```

```
}

,

 ""explore_forest"": {

 ""text"": ""You find a hidden cave. Do you want to enter?"",

 ""options"": {

 ""yes"": ""enter_cave"",

 ""no"": ""return_to_forest""

 }

 },

 ""build_shelter"": {

 ""text"": ""You gather materials and build a shelter. What next?"",

 ""options"": {

 ""explore"": ""explore_forest"",

 ""wait"": ""wait_for_help""

 }

 },

 ""enter_cave"": {

 ""text"": ""Inside the cave, you find a treasure chest! Open it?"",

 ""options"": {

 ""open"": ""treasure_found"",

 ""leave"": ""leave_cave""

 }

 },

 ""return_to_forest"": {

 ""text"": ""You return to the forest. What do you want to do?"",

 ""options"": {
```

```

 ""explore"": ""explore_forest"",

 ""build_shelter"": ""build_shelter""

 }

},

""wait_for_help"": {

 ""text"": ""You wait for hours, but no one comes. What now?:"",

 ""options"": {

 ""explore"": ""explore_forest"",

 ""build_shelter"": ""build_shelter""

 }

},

""treasure_found"": {

 ""text"": ""You found gold and jewels! You win!:"",

 ""options"": {}

},

""leave_cave"": {

 ""text"": ""You leave the cave and find yourself back in the forest."",

 ""options"": {

 ""explore"": ""explore_forest"",

 ""build_shelter"": ""build_shelter""

 }

}

}

self.current_node = ""start""

```

```

def display_current_story(self):

```

```

current_story = self.story[self.current_node]

print(current_story["text"])

if current_story["options"]:

 for option in current_story["options"]:

 print(f"- {option}")

def make_choice(self, choice):

 if choice in self.story[self.current_node]["options"]:

 self.current_node = self.story[self.current_node]["options"][choice]

 else:

 print("Invalid choice. Try again.")

def play(self):

 while True:

 self.display_current_story()

 if not self.story[self.current_node]["options"]:

 break

 choice = input("What do you want to do? ").strip().lower()

 self.make_choice(choice)

 print("Game Over.")

if __name__ == "__main__":

 game = Game()

 game.play()

```

```,2024-08-24T18:39:39.929561

8edef031-5f31-4584-a019-f56005f9a998,Pattern Exploration Agent 3,Create 10 unique games in

Python using novel game design patterns. Each agent should output the Python code for their game and only their game.,```\python

```
import random
```

```
class Entity:
```

```
    def __init__(self, name, health, damage):
```

```
        self.name = name
```

```
        self.health = health
```

```
        self.damage = damage
```

```
    def is_alive(self):
```

```
        return self.health > 0
```

```
    def attack(self, target):
```

```
        if self.is_alive():
```

```
            target.health -= self.damage
```

```
            print(f"{self.name} attacks {target.name} for {self.damage} damage!")
```

```
class Player(Entity):
```

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

```
        super().__init__(name, health=100, damage=10)
```

```
    def heal(self):
```

```
        self.health += 20
```

```
        print(f"{self.name} heals for 20 health!")
```

```
class Enemy(Entity):

    def __init__(self, name):

        super().__init__(name, health=random.randint(50, 100), damage=random.randint(5, 15))
```

```
class Game:
```

```
    def __init__(self):

        self.player = Player("Hero")

        self.enemies = [Enemy(f"Enemy {i}") for i in range(1, 5)]

        self.turn = 0

    def player_turn(self):

        action = input("Choose action (attack/heal): ").strip().lower()

        if action == "attack":

            target = self.choose_enemy()

            self.player.attack(target)

        elif action == "heal":

            self.player.heal()

        else:

            print("Invalid action. Try again.")
```

```
    def choose_enemy(self):

        alive_enemies = [enemy for enemy in self.enemies if enemy.is_alive()]

        if not alive_enemies:

            print("All enemies defeated!")

            return None

        print("Choose an enemy to attack:")
```



```
for idx, enemy in enumerate(alive_enemies):

    print(f""{idx + 1}: {enemy.name} (Health: {enemy.health})"")

choice = int(input("Enter enemy number: ")) - 1

return alive_enemies[choice]
```

```
def enemy_turn(self):

    for enemy in self.enemies:

        if enemy.is_alive():

            enemy.attack(self.player)
```

```
def check_game_over(self):

    if not self.player.is_alive():

        print("You have been defeated!")

        return True

    if all(not enemy.is_alive() for enemy in self.enemies):

        print("All enemies defeated! You win!")

        return True

    return False
```

```
def play(self):

    while not self.check_game_over():

        print("\n--- Player's Turn ---")

        self.player_turn()

        self.enemy_turn()

        self.turn += 1

    print(f"Turn {self.turn} complete.")
```

```
print(f"{self.player.name} Health: {self.player.health}")
```

```
if __name__ == "__main__":
```

```
    game = Game()
```

```
    game.play()
```

```
```,2024-08-24T18:39:39.929561
```

8edef031-5f31-4584-a019-f56005f9a998,Pattern Exploration Agent 4,Create 10 unique games in Python using novel game design patterns. Each agent should output the Python code for their game and only their game.,``python

```
import random
```

```
class Character:
```

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

```
 self.name = name
```

```
 self.abilities = self.generate_abilities()
```

```
 def generate_abilities(self):
```

```
 abilities = []
```

```
 for _ in range(3): # Each character gets 3 abilities
```

```
 ability = self.create_ability()
```

```
 abilities.append(ability)
```

```
 return abilities
```

```
 def create_ability(self):
```

```
 ability_types = ['Fire', 'Water', 'Earth', 'Air', 'Lightning']
```

```
 power_levels = ['Weak', 'Moderate', 'Strong']
```

```
ability_type = random.choice(ability_types)

power_level = random.choice(power_levels)

return f"""{power_level} {ability_type} Attack"""
```

```
def display_character(self):

 print(f"""Character: {self.name}""")

 print("""Abilities: """)

 for ability in self.abilities:

 print(f"""- {ability}""")
```

```
def create_characters(num_characters):

 characters = []

 for i in range(num_characters):

 name = f"""Hero_{i + 1}"""

 character = Character(name)

 characters.append(character)

 return characters
```

```
def main():

 num_characters = 5

 characters = create_characters(num_characters)

 for character in characters:

 character.display_character()

 print()
```

```
if __name__ == "__main__":
```

```
 main()
```

```
```",2024-08-24T18:39:39.929561
```

8edef031-5f31-4584-a019-f56005f9a998,Pattern Exploration Agent 5,Create 10 unique games in Python using novel game design patterns. Each agent should output the Python code for their game and only their game.,``python

```
import random
```

```
class Game:
```

```
    def __init__(self):
```

```
        self.locations = {
```

```
            'Forest': {'description': 'A dark and dense forest.', 'items': ['Mushroom'], 'exits': ['Cave',
```

```
'Village']},
```

```
            'Cave': {'description': 'A damp cave echoing with sounds.', 'items': ['Gem', 'Lantern'], 'exits':
```

```
['Forest']},
```

```
            'Village': {'description': 'A small village with friendly faces.', 'items': ['Bread', 'Potion'], 'exits':
```

```
['Forest', 'Castle']},
```

```
            'Castle': {'description': 'An ancient castle, looming in the distance.', 'items': ['Sword', 'Shield'],
```

```
'exits': ['Village']}
```

```
        }
```

```
        self.current_location = 'Forest'
```

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

```
    def play(self):
```

```
        while True:
```

```
            self.describe_location()
```

```
command = input("What do you want to do? (explore, take, use, exit): ").strip().lower()

if command == 'explore':
    self.explore()
elif command == 'take':
    self.take_item()
elif command == 'use':
    self.use_item()
elif command == 'exit':
    print("Thanks for playing!")
    break
else:
    print("Invalid command.")
```

```
def describe_location(self):
    location = self.locations[self.current_location]
    print(f"\nYou are at the {self.current_location}.")
    print(location['description'])
    print(f"You see: {', '.join(location['items'])}")
    print(f"Exits: {', '.join(location['exits'])}")
```

```
def explore(self):
    exit_choice = input("Which exit do you want to take? ").strip()

    if exit_choice in self.locations[self.current_location]['exits']:
        self.current_location = exit_choice
    else:
        print("You can't go that way!")
```

```
def take_item(self):

    location = self.locations[self.current_location]

    item_choice = input("Which item do you want to take? ").strip()

    if item_choice in location['items']:

        self.inventory.append(item_choice)

        location['items'].remove(item_choice)

        print(f"You took the {item_choice}.")

    else:

        print("That item is not here!")
```

```
def use_item(self):

    if not self.inventory:

        print("Your inventory is empty!")

        return

    item_choice = input("Which item do you want to use? ").strip()

    if item_choice in self.inventory:

        print(f"You used the {item_choice}.")

        # Here, you might want to define effects of using items

        self.inventory.remove(item_choice)

    else:

        print("You don't have that item!")
```

```
if __name__ == "__main__":
```

```
    game = Game()
```

```
    game.play()
```

```
```",2024-08-24T18:39:39.929561
```

8edef031-5f31-4584-a019-f56005f9a998,Pattern Exploration Agent 6,Create 10 unique games in Python using novel game design patterns. Each agent should output the Python code for their game and only their game.,```python

```
import random
```

```
class Agent:
```

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

```
 self.name = name
```

```
 self.health = 100
```

```
 self.power = random.randint(5, 15)
```

```
 def attack(self, target):
```

```
 damage = self.power
```

```
 target.health -= damage
```

```
 print(f"{self.name} attacks {target.name} for {damage} damage!")
```

```
 if target.health <= 0:
```

```
 print(f"{target.name} has been defeated!")
```

```
class Player(Agent):
```

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

```
 super().__init__(name)
```

```
 def heal(self):
```

```
 heal_amount = random.randint(10, 20)
```

```
 self.health += heal_amount
```

```
print(f"{self.name} heals for {heal_amount} health!")
```

```
class AI(Agent):
```

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

```
 super().__init__(name)
```

```
 def choose_action(self, target):
```

```
 if self.health < 20:
```

```
 self.heal()
```

```
 else:
```

```
 self.attack(target)
```

```
class Game:
```

```
 def __init__(self):
```

```
 self.player = Player("Hero")
```

```
 self.ai_partners = [AI(f"AI-{i}") for i in range(1, 4)]
```

```
 self.enemy = Agent("Monster")
```

```
 def player_turn(self):
```

```
 action = input("Do you want to (A)ttack or (H)eal? ").strip().lower()
```

```
 if action == 'a':
```

```
 self.player.attack(self.enemy)
```

```
 elif action == 'h':
```

```
 self.player.heal()
```

```
 def ai_turns(self):
```



```

 for ai in self.ai_partners:

 if self.enemy.health > 0:

 ai.choose_action(self.enemy)

def game_loop(self):

 while self.player.health > 0 and self.enemy.health > 0:

 print(f"\nPlayer Health: {self.player.health}, Enemy Health: {self.enemy.health}")

 self.player_turn()

 if self.enemy.health > 0:

 self.ai_turns()

 if self.enemy.health <= 0:

 print("The Monster has been defeated! You win!")

 break

 self.enemy.attack(self.player)

 if self.player.health <= 0:

 print("You have been defeated! Game over.")

if __name__ == "__main__":

 game = Game()

 game.game_loop()

```",2024-08-24T18:39:39.929561

8edef031-5f31-4584-a019-f56005f9a998,Pattern Exploration Agent 7,Create 10 unique games in
Python using novel game design patterns. Each agent should output the Python code for their game
and only their game.,```python

import cv2

import numpy as np

```

```
import random
```

```
class AugmentedRealityGame:
```

```
    def __init__(self):
```

```
        self.cap = cv2.VideoCapture(0) # Open the camera
```

```
        self.target_position = self.generate_target_position()
```

```
        self.score = 0
```

```
    def generate_target_position(self):
```

```
        # Generate random target position on screen
```

```
        return (random.randint(100, 500), random.randint(100, 400))
```

```
    def draw_target(self, frame):
```

```
        # Draw target on the frame
```

```
        target_color = (0, 255, 0) # Green target
```

```
        cv2.circle(frame, self.target_position, 20, target_color, -1)
```

```
    def check_hit(self, x, y):
```

```
        target_x, target_y = self.target_position
```

```
        if (target_x - 20) < x < (target_x + 20) and (target_y - 20) < y < (target_y + 20):
```

```
            self.score += 1
```

```
            self.target_position = self.generate_target_position() # Move target
```

```
    def run(self):
```

```
        while True:
```

```
            ret, frame = self.cap.read()
```

```
if not ret:
```

```
    break
```

```
self.draw_target(frame)
```

```
    cv2.putText(frame, f'Score: {self.score}', (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2)
```

```
cv2.imshow("Augmented Reality Game", frame)
```

```
key = cv2.waitKey(1)
```

```
if key & 0xFF == 27: # Escape key to exit
```

```
    break
```

```
elif key & 0xFF == ord('c'): # Simulate clicking
```

```
    x, y = random.randint(100, 600), random.randint(100, 400) # Simulated cursor position
```

```
    self.check_hit(x, y)
```

```
self.cap.release()
```

```
cv2.destroyAllWindows()
```

```
if __name__ == "__main__":
```

```
    game = AugmentedRealityGame()
```

```
    game.run()
```

```
```,2024-08-24T18:39:39.929561
```

8edef031-5f31-4584-a019-f56005f9a998,Pattern Exploration Agent 8,Create 10 unique games in Python using novel game design patterns. Each agent should output the Python code for their game

and only their game.,``python

```
import pygame
```

```
import sys
```

```
import random
```

```
Initialize Pygame
```

```
pygame.init()
```

```
Constants
```

```
WIDTH, HEIGHT = 800, 600
```

```
FPS = 60
```

```
TIME_SCALE = 0.5 # Time manipulation factor
```

```
Colors
```

```
WHITE = (255, 255, 255)
```

```
BLACK = (0, 0, 0)
```

```
RED = (255, 0, 0)
```

```
Game Objects
```

```
class Player(pygame.sprite.Sprite):
```

```
 def __init__(self):
```

```
 super().__init__()
```

```
 self.image = pygame.Surface((50, 50))
```

```
 self.image.fill(WHITE)
```

```
 self.rect = self.image.get_rect(center=(WIDTH // 2, HEIGHT // 2))
```

```
 self.speed = 5
```

```
def update(self):
 keys = pygame.key.get_pressed()

 if keys[pygame.K_LEFT]:
 self.rect.x -= self.speed

 if keys[pygame.K_RIGHT]:
 self.rect.x += self.speed

 if keys[pygame.K_UP]:
 self.rect.y -= self.speed

 if keys[pygame.K_DOWN]:
 self.rect.y += self.speed
```

```
class TimeBubble(pygame.sprite.Sprite):
```

```
 def __init__(self, x, y):
 super().__init__()

 self.image = pygame.Surface((30, 30))

 self.image.fill(RED)

 self.rect = self.image.get_rect(center=(x, y))
```

```
 def update(self):
 self.rect.y += 2 # Move downwards
```

```
Game Class
```

```
class TimeManipulationGame:
```

```
 def __init__(self):
 self.screen = pygame.display.set_mode((WIDTH, HEIGHT))
```

```
pygame.display.set_caption("Time Manipulation Game")

self.clock = pygame.time.Clock()

self.player = Player()

self.all_sprites = pygame.sprite.Group(self.player)

self.time_bubbles = pygame.sprite.Group()

self.spawn_time_bubble()
```

```
def spawn_time_bubble(self):

 x = random.randint(0, WIDTH)

 bubble = TimeBubble(x, 0)

 self.time_bubbles.add(bubble)

 self.all_sprites.add(bubble)
```

```
def run(self):

 while True:

 for event in pygame.event.get():

 if event.type == pygame.QUIT:

 pygame.quit()

 sys.exit()

 self.all_sprites.update()

 # Check for collisions with time bubbles

 if pygame.sprite.spritecollide(self.player, self.time_bubbles, True):

 self.player.speed *= TIME_SCALE # Slow down player speed

 self.spawn_time_bubble() # Spawn new bubble
```

```
self.screen.fill(BLACK)

self.all_sprites.draw(self.screen)

pygame.display.flip()

self.clock.tick(FPS)
```

```
if __name__ == "__main__":

 game = TimeManipulationGame()

 game.run()
```

```
```,2024-08-24T18:39:39.929561
```

8edef031-5f31-4584-a019-f56005f9a998,Pattern Exploration Agent 9,Create 10 unique games in Python using novel game design patterns. Each agent should output the Python code for their game and only their game.,```\python

```
import random

import time
```

```
class Game:

    def __init__(self):

        self.players = []

        self.resources = 100

        self.rounds = 5

    def add_player(self, player):

        self.players.append(player)

    def start_game(self):
```

```
print("""Game Start!""")

for round_number in range(1, self.rounds + 1):

    print(f"""\nRound {round_number}""")

    self.play_round()

    time.sleep(1)

self.end_game()
```

```
def play_round(self):

    for player in self.players:

        player.take_turn(self.resources)
```

```
def end_game(self):

    print("""\nGame Over!""")

    for player in self.players:

        print(f""{player.name} collected {player.resources_collected} resources.""")

    winner = max(self.players, key=lambda p: p.resources_collected)

    print(f""The winner is {winner.name}!""")
```

```
class Player:

    def __init__(self, name, role):

        self.name = name

        self.role = role

        self.resources_collected = 0

    def take_turn(self, total_resources):

        if self.role == ""Collector"":
```



```

        collected = random.randint(1, 10)

        self.resources_collected += collected

        print(f"{self.name} (Collector) collected {collected} resources.")

    elif self.role == "Saboteur":

        sabotage = random.randint(1, 5)

        total_resources -= sabotage

        print(f"{self.name} (Saboteur) sabotaged {sabotage} resources.")

    else:

        print(f"{self.name} has an unknown role.")

def main():

    game = Game()

    num_collectors = 2

    num_saboteurs = 1

    for i in range(num_collectors):

        game.add_player(Player(f"Collector {i + 1}", "Collector"))

    for i in range(num_saboteurs):

        game.add_player(Player(f"Saboteur {i + 1}", "Saboteur"))

    game.start_game()

if __name__ == "__main__":

    main()

```",2024-08-24T18:39:39.929561

```

8edef031-5f31-4584-a019-f56005f9a998,Pattern Exploration Agent 10,Create 10 unique games in Python using novel game design patterns. Each agent should output the Python code for their game and only their game.,```\python

```
import random
```

```
class Game:
```

```
 def __init__(self):
```

```
 self.players = {}
```

```
 self.levels = []
```

```
 self.current_level = 0
```

```
 self.player_content = []
```

```
 def add_player(self, player_name):
```

```
 self.players[player_name] = []
```

```
 def add_level(self, level_content):
```

```
 self.levels.append(level_content)
```

```
 def generate_level(self):
```

```
 if self.player_content:
```

```
 level_content = random.choice(self.player_content)
```

```
 else:
```

```
 level_content = f"Default Level {len(self.levels) + 1}"
```

```
 self.add_level(level_content)
```

```
 def submit_content(self, player_name, content):
```

```
if player_name in self.players:

 self.player_content.append(content)

 self.players[player_name].append(content)

 print(f"""{player_name} submitted: {content}""")

else:

 print(f"""Player {player_name} not found.""")
```

```
def play_level(self):

 if self.current_level < len(self.levels):

 print(f"""Playing Level: {self.levels[self.current_level]}""")

 self.current_level += 1

 else:

 print("""No more levels to play!""")
```

```
def show_player_content(self, player_name):

 if player_name in self.players:

 print(f"""{player_name}'s submissions: {self.players[player_name]}""")

 else:

 print(f"""Player {player_name} not found.""")
```

```
def main():

 game = Game()

 game.add_player("""Alice""")

 game.add_player("""Bob""")

 game.submit_content("""Alice""", """A mysterious cave filled with crystals.""")
```

```
game.submit_content("Bob", "A raging river with dangerous currents.")
```

```
game.generate_level()
```

```
game.play_level()
```

```
game.show_player_content("Alice")
```

```
game.show_player_content("Bob")
```

```
game.generate_level()
```

```
game.play_level()
```

```
if __name__ == "__main__":
```

```
 main()
```

```
```,2024-08-24T18:39:39.929561
```