SELF-EVOLVING MULTI-AGENT COLLABORATION NETWORKS FOR SOFTWARE DEVELOPMENT

Anonymous authors
Paper under double-blind review

APPENDIX

1 BENCHMARK DETAILS

Table 1: Basic statistics for website and game domains, including the amount of samples, prompt length (mean/max), and number of test cases at both Basic and Advanced levels.

Benchmark	Software		Test Case	
	Amount	Length	Basic	Advanced
Website	45	1011/1553	292	247
Game	8	507/788	46	31

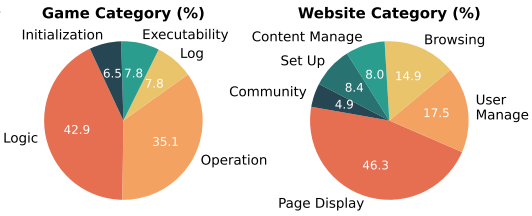


Figure 1: Statistics of Game and Website tasks.

Step 1: Software requirement generation. Each task instance begins with the generation of clear, measurable software requirements. Given the inherent differences across various types of software, we adopt distinct approaches for their formulation. For game-related software, we focuses on common real-world games, capturing detailed task requirements such as GUI layout initialization, interaction methods, and game rules. To align more closely with actual game development practices, we also include game state logging as part of the software requirements. Due to the complexity of logic in game software, these requirements are manually crafted by human. In contrast, for website-related software, we begin with a concise website name, and then leverage the large language model (`gpt-4o-mini`) to enrich the requirements according to predefined patterns. This approach ensures both efficiency and scalability in the creation of benchmarks for websites. By tailoring the process to the distinct characteristics of each software domain, we maintain precision in requirement formulation while addressing the unique challenges posed by each context.

Step 2: Requirement-based test cases generation.

As illustrated in Fig 2 and Fig 3, unit tests offer a precise evaluation of software completion. Each task instance includes black-box unit test cases that correspond directly to the software requirements, allowing for a quantitative assessment of requirement fulfillment. To further assess the model's code generation capabilities, we categorize test cases into two levels of difficulty—basic and advanced, as outlined in Tab. 1. We also provide an overview of all websites and games in Tab. 2 and Tab. 3 respectively. As shown in Fig. 1, test cases for website and game software exhibit structural differences, reflecting the distinct nature of each software type. They enable more targeted evaluation of code generation capabilities. Thus, similar to software requirements, the test cases are constructed differently based on the software type. For game-related tests, we manually create test cases, akin to the HumanEval Chen et al. (2021) benchmark, which tracks state changes in response to specific inputs. In the game environment, we assess how game states evolve in response to GUI interactions. For website-related tests, large language model (`gpt-4o-mini`) generates Selenium-based test cases aligned with the software requirements, followed by manual corrections to resolve any ambiguities. This structured approach ensures rigorous evaluation across diverse software domains.

Basic and advanced requirements definition. For the games, basic requirements involve straightforward user interactions that do not require complex logic, such as character movement or interacting with simple GUI elements. Advanced requirements incorporate more intricate logic, such as managing game state transitions based on user actions or handling conditional game events. These cases focus on ensuring the correct execution of basic actions. In contrast, advanced cases incorporate more intricate logic, such as managing game state transitions based on user actions or handling

Table 2: Overview of Websites in rSDE-Bench.

Websites		
CharitableGivingPlatform	DailyHealthTips	DailyJournalApp
EcoFriendlyLivingTips	ElderCareResources	EventPlanner
FitnessEquipmentRental	FitnessTracker	FreelancerMarketplace
GreenLivingGuide	HealthConsultationPlatform	MotivationalQuotesApp
MusicFestivalDirectory	NoteTakingApp	NutritionInformationHub
OnlineLibraryManagementSystem	OnlineTherapeuticJournaling	OnlineThriftStore
PeerTutoringNetwork	PersonalBlog	PersonalFinanceBlog
RecipeHub	RemoteInternshipMarketplace	RemoteJobBoard
TravelDiary	VirtualBookPublishing	VirtualWellnessRetreats
DigitalArtworkGallery	DigitalStorytellingPlatform	ExpenseTracker
FitnessChallenges	GardeningForBeginners	GourmetFoodSubscription
MovieRecommendationSystem	MusicCollaborator	OnlineCulturalExchange
OnlineCulturalFestivals	OnlineVintageMarket	ParentingAdviceForum
PetCareCommunity	PortfolioSite	SkillShare
TaskManager	VolunteerMatch	OnlineShoppingCenter

Table 3: Overview of Games in rSDE-Bench.

Games			
Balls	Tank	Racing	Ghostly
Mario	Bomberman	Sokoban	Brick

Software description

```
Task: Develop a simple Sokoban game. You must design a GUI.
Requirements:
1. The game board should be divided into grid squares.
2. Players will control the game using the arrow keys on the keyboard.
3. As the game starts, a log file named 'game.log' should be created to record the game's progress. The content of the
game.log file should be appended with a new entry after each player action.The content of the game.log file should be
cleared (if any) at the start of each game session.
Each log entry should follow this format:
{
"timestamp": timestamp,
"EVENT_TYPE": "MOVE_RIGHT" | "MOVE_LEFT" | "MOVE_UP" | "MOVE_DOWN" | "INVALID_MOVE",
"player_position": [x, y],
"box_positions": [[x1, y1], [x2, y2], ...],
"game_status": "ONGOING" | "COMPLETE"
}
4. The victory conditions for the game is: All boxes are pushed onto their corresponding coordinate point.
5. The initial positions of each element are required as follows:
player_position = [1, 1]
box_positions = [[3, 3], [4, 2]]
goal_positions = [[5, 5], [6, 3]]
([3, 3] is the initial position of the first box whose target position is [5, 5].  [4, 2] is the initial position of
the second box whose target position is [6, 3].)
wall_positions = [[0, 4], [1, 4], [2, 4],[3, 4],[4, 4]]
(the first numnber in each pair is the x-coordinate and the second number is the y-coordinate)
```

Evaluation functions

```
check_Excutablity  check_log  check_move_right  check_move_left  check_move_box
check_move_wall  check_seqbox  check_end  check_wrong_end
```

Figure 2: Test cases of Game in rSDE-Bench.

conditional game events. These cases challenge the model's ability to generate code that integrates dynamic decision-making and interaction within the game environment. For websites, basic cases focus on ensuring that the necessary page elements—such as input fields, buttons, and layouts—are present correctly. These cases assess the completeness of the webpage's structure. On the other hand, advanced cases evaluate more complex functionality, such as handling user authentication, managing dynamic content, or executing specific operations within a content management system. These cases require the model to generate code that performs backend logic and manages user interactions at a deeper level.

Software description

```
# Requirement Document for DailyHealthTips Web Application

## 1. Objective
Develop a web application named 'DailyHealthTips' that provides users with daily health tips, allowing them to receive
advice and information about maintaining a healthy lifestyle, using Python as the development language. Note that the
website should start from the login page.

## 2. Language
The required development language for the DailyHealthTips web application is Python.

## 3. Page Design

### Page 1: Login Page
- **Page Title**: User Login
- **Overview**: This page allows users to log in to their accounts.
- **Elements**:
  - **Username Field**:
    - **ID**: `username_field`
  - **Password Field**:
    - **ID**: `password_field`
  - **Login Button**:
    - **ID**: `login_button`
……
```

Evaluation functions

```
test_login_page_elements  test_login_page_functionality  test_daily_tips_page_elements
test_daily_tips_page_functionality  test_tips_archive_page_elements  test_tips_archive_page_functionality
```

Figure 3: Test cases of Website in rSDE-Bench.

Algorithm 1 Self-Evolving Paradigm

Require: $\mathbf{X}$ ▷ Task input
Require: $\mathcal{A}_g^{(0)}$ ▷ Initialized MAC network: agent prompts and pipeline
Require: $\mathcal{A}_t$ ▷ Designed MAC network to generate target proxy
Require: $\mathcal{G}$ ▷ Agent-based gradient function
Require: $\mathcal{U}$ ▷ Agent-based update function
Require: E ▷ Environment tool to generate loss

Define K as the number of self-evolving iterations, Φ as MACN generation process
Target Proxy
$\mathbf{T} = \Phi(\mathbf{X}, \mathcal{A}_t)$
# Self-Evolving Procedure
for $k = 0, 1, \ldots, K-1$ **do**
    # Forward Pass
    $\mathbf{G}^{(k)} = \Phi(\mathbf{X}, \mathcal{A}_g^{(k)})$
    # Loss Computation
    $\mathbf{L}^{(k)} = \langle \mathbf{G}^{(k)}, \mathbf{T} \rangle_E$ ▷ Use environment feedback as textual loss
    # Textual Backpropagation
    $\nabla \mathbf{L}^{(k)} = \mathcal{G}(\mathbf{L}^{(k)}, \mathcal{A}_g^{(k)})$ ▷ Summarize textual gradient
    $\mathcal{A}_g^{(k+1)} = \mathcal{U}(\mathcal{A}_g^{(k)}, \nabla \mathbf{L}^{(k)})$ ▷ Update agent prompts and pipeline
end for
return $\mathcal{A}_g^{(K)}, \mathbf{G}^{(K)}$

2 Algorithm

In this section, we present the algorithm of EvoMAC in Alg. 1. For more details, please refer to Section 3.

3 Case Study

3.1 Complete EvoMAC process

In this section, we show a complete process of EvoMAC on RSD-Bench. Please refer to Tab. 4.

Table 4: A complete iteration process of EvoMAC on RSD-Bench

Notation	Meaning	Example	Real example
$\mathbf{X}$	Textual description of the task to be completed.	A coding task such as: "Implement a code that simulates keyboard input processing via Python."	See Tab. 5
$\mathcal{A}_g$	MAC network representing the team responsible for generating the code.	The coding team consists of coding agents completing subtasks in sequence.	See Tab. 6
$\mathbf{G} = \Phi(\mathbf{X}, \mathcal{A}_g)$	Generated output produced by the coding team as a result of the feed-forward pass.	The generated code: '...def process_input(keyboard_input): ...'	See Tab. 7
$\mathcal{A}_t$	MAC network representing the team responsible for generating the target proxy (unit tests).	The testing team generates unit tests for the task.	See Tab. 8
$\mathbf{T} = \Phi(\mathbf{X}, \mathcal{A}_t)$	Target proxy (unit tests) generated by the testing team based on the task description.	Unit tests like: '...def test_press_input(): assert process_input('Enter') == 'Processed Enter''	See Tab. 9
$< \mathbf{G}, \mathbf{T} >_E$	Environmental feedback comparing the generated output $\mathbf{G}$ with the target proxy $\mathbf{T}$ using an objective environment (e.g., compiler or test results).	The environment executes the generated code against the unit tests, providing feedback like: 'Failure: test_press_input'	The execution outcome of the unit test from the terminal. If the execution is successful, the outcome is 'The software run successfully without errors.'
$\min < \mathbf{G}, \mathbf{T} >_E$	The optimization objective aiming to minimize the difference between the generated output and the target proxy using the environmental feedback.	Based on feedback, the system iteratively refines the coding team to generate code that better meets the task.	See Tab. 10

3.2 Unit test case

In this section, we show more unit test cases written by coder on RSD-Bench, please refer to Tab. 11 and Tab. 14.

3.3 Updating process

In this section, we show additional examples of the updating process on RSD-Bench and HumanEval dataset. Please refer to Tab. 17 and Tab. 20 respectively. For RSD-Bench, due to the code length, we only show the texture updating process(codes are available at Sec. 3.1). We can see that the updating agent will adjust the job of each coder dynamically according to the result of test team.

4 Software Presentation

In this section, we show some games and websites written by EvoMAC. Fig. 4 and Fig. 5 present the games and websites respectively. We see that: i) EvoMAC outputs games with well-written GUI and

Table 5: **Textual description of the task to be completed.**

Task: Design a Single-Player Tank Battle Game
Requirements:
1. The interface should be divided into a 20x20 grid, though grid lines are not necessary. Each tank occupies one grid space, while obstacles may occupy multiple grid spaces. The background should be black, obstacles should be brown, enemy tanks should be silver, and the player's tank should be yellow.
2. The player can control the tank's movement using the arrow keys on the keyboard, allowing for movement one grid space at a time. The 'enter' key is used to fire bullets.
3. In the game, there are two enemies fixed at a certain position on the game interface, constantly firing bullets in four directions: up, down, left, and right. Two enemies and players cannot be initialized in the same row.
4. Both the player and the enemies have their own health points, which are initialized to 200. When hit by a bullet, the player's health decreases by 10 and the enemy's health decreases by 100. When health points drop to zero, the corresponding tank is destroyed. But the log still records information about the destroyed tank, with health points of 0.
5. Destroying an enemy tank earns the player 200 points. The game ends when the player's tank is destroyed or all enemy tanks are destroyed, at which point the player's score will be displayed on the screen.
6. As the game start, a new log file named 'game.log' should be created to record the game's progress. The first log entry should capture the initial state of the game. Each time a new event occurs to the player, a new log entry should be written in real-time. The logs should follow the format below. The EVENT_TYPE can only be one of the following: "INIT", "MOVE_LEFT", "MOVE_RIGHT", "MOVE_UP", "MOVE_DOWN", "FIRE" or "INJURED". The game_state should capture the current state of the game. In this setup, the position coordinates are defined with the top-left grid as [0, 0], where the x-coordinate increases by one unit for each grid space moved to the right, and the y-coordinate increases by one unit for each grid space moved downward. The player is initialized at position [0,0].

```
{
    "timestamp": timestamp,
    "EVENT_TYPE": "INIT" | "MOVE_LEFT" | "MOVE_RIGHT" | "MOVE_UP"
    | "MOVE_DOWN" | "FIRE" | "INJURED",
    "game_state": {
        "player": {
            "position": [xp, yp],
            "health": health_player,
            "score": score
        },
        "enemies": [
            {
                "position": [xe1, ye1],
                "health": health_enemy1
            },
            {
                "position": [xe2, ye2],
                "health": health_enemy2
            }
        ],
        "obstacle_position": [(xo1, yo1), ..., (xon, yon)]
    },
}
```

game rules. It can handle different kinds of GUI and game rule requirements from diverse games. ii) EvoMAC outputs websites with beautified, user-friendly web pages and correct transition logic. It can handle the requirements of different websites.

Table 6: **MAC network representing the team responsible for generating the code.**


```
{
    "composition": {
        "Programmer 1": "Implement bullet firing logic that decreases
        health points for both the player and enemies when hit. This
        includes creating a function to handle bullet movement and
        collision detection, updating health points accordingly, and
        logging the events in the game.log file.",
        "Programmer 2": "Implement health management to check for
        health points dropping to zero and handle tank destruction.
        This includes creating a function to check health status
        after each hit, updating the game state, and logging the
        destruction event in the game.log file.",
        "Programmer 3": "Implement scoring logic to update the player's
        score when an enemy tank is destroyed and display the final
        score when the game ends. This includes creating a function
        to manage scoring and game-over conditions, and logging the
        final score in the game.log file."
    },
    "workflow": {
        "Programmer 1": [],
        "Programmer 2": [
            "Programmer 1"
        ],
        "Programmer 3": [
            "Programmer 2"
        ]
    }
}
```

Table 7: **Generated output produced by the coding team as a result of the feed-forward pass. Due to the limit of page length, we only show partial of the codes.**

```
'''
This is the main file for the Single-Player Tank Battle Game.
It initializes the game, sets up the GUI, and manages the game
loop, including player movement, enemy behavior, and logging
events.
'''
import pygame
import json
import time
# Constants
GRID_SIZE = 20
CELL_SIZE = 30
WINDOW_SIZE = GRID_SIZE * CELL_SIZE
PLAYER_COLOR = (255, 255, 0)  # Yellow
ENEMY_COLOR = (192, 192, 192)  # Silver
OBSTACLE_COLOR = (139, 69, 19)  # Brown
BACKGROUND_COLOR = (0, 0, 0)  # Black
FPS = 60
# Game state
player_position = [0, 0]
player_health = 200
player_score = 0
enemies = [
    {"position": [5, 5], "health": 200},
    {"position": [10, 10], "health": 200}
]
obstacles = [(3, 3), (7, 7), (12, 12)]
# Log file
log_file = open('game.log', 'w')
initial_state = {
    "timestamp": time.time(),
    "EVENT_TYPE": "INIT",
    "game_state": {
        "player": {
            "position": player_position,
            "health": player_health,
            "score": player_score
        },
        "enemies": enemies,
        "obstacle_position": obstacles
    }
}
log_file.write(json.dumps(initial_state) + '\n')
# Initialize Pygame
pygame.init()
screen = pygame.display.set_mode((WINDOW_SIZE, WINDOW_SIZE))
pygame.display.set_caption("Tank Battle Game")
clock = pygame.time.Clock()
class Bullet:
    '''
    This class represents a bullet fired by the player.
    '''
    def __init__(self, position, direction):
        self.position = position
        self.direction = direction
    def move(self):
        if self.direction == 'UP':
            self.position[1] -= 1
        elif self.direction == 'DOWN':
            self.position[1] += 1
        elif self.direction == 'LEFT':
            self.position[0] -= 1
        elif self.direction == 'RIGHT':
            self.position[0] += 1
```

Table 8: **MAC network representing the team responsible for generating the target proxy (unit tests).**

```
{
    "composition": {
        "Programmer 1": "Carefully test the logging mechanism
        according to the task requirements. Ensure that logs
        happen immediately after an action is taken, recording
        the most recent state. Verify the logging order, ensuring
        basic operations are recorded first, followed by subsequent
        events. Check that the data format, keys, and values are
        accurate, paying attention to the nested data types and
        carefully checking each element.",
        "Programmer 2": "Test the logging mechanism for special
        triggered conditions, such as when the player's health drops
        to zero or when an enemy tank is destroyed. Ensure that these
        events are logged correctly and that the game state reflects
        the changes accurately.",
        "Programmer 3": "Test the value initialization required by
        the task to ensure they are correctly achieved. Pay attention
        to the coordinates of the player and enemies, ensuring they
        are initialized in the correct positions and that no two tanks
        occupy the same row.",
        "Programmer 4": "Test the function inputs and the global
        variables imported in each function. Ensure that the input
        values and global variables used in the functions are valid
        and involved when the function is called.",
        "Programmer 5": "Test that each event in the task is
        implemented correctly and that the logic triggered matches
        the conditions described in the task. This includes movement,
        firing bullets, and health reduction for both the player
        and enemy tanks."
    },
    "workflow": {
        "Programmer 1": [],
        "Programmer 2": [
            "Programmer 1"
        ],
        "Programmer 3": [
            "Programmer 1",
            "Programmer 2"
        ],
        "Programmer 4": [
            "Programmer 1",
            "Programmer 2",
            "Programmer 3"
        ],
        "Programmer 5": [
            "Programmer 1",
            "Programmer 2",
            "Programmer 3",
            "Programmer 4"
        ]
    }
}
```

Table 9: **Target proxy (unit tests) generated by the testing team based on the task description. Due to the limit of page length, we only show one test case written by one of the five test coders.**

```
import unittest
import json
import time
from main import player_position, player_health, player_score,
enemies, log_file, log_event, fire_bullet, check_health
class TestTankBattleGame(unittest.TestCase):
    def setUp(self):
        global player_position, player_health, player_score, enemies
        player_position = [0, 0]
        player_health = 200
        player_score = 0
        enemies = [
            {"position": [5, 5], "health": 200},
            {"position": [10, 10], "health": 200}
        ]
        log_file = open('game.log', 'w')
    def tearDown(self):
        log_file.close()
    def test_move_left(self):
        global player_position
        player_position[0] = 1  # Move to the right first
        log_event("MOVE_LEFT")
        player_position[0] -= 1
        self.assertEqual(player_position, [0, 0])
    def test_move_right(self):
        global player_position
        log_event("MOVE_RIGHT")
        player_position[0] += 1
        self.assertEqual(player_position, [1, 0])
    def test_move_up(self):
        global player_position
        player_position[1] = 1  # Move down first
        log_event("MOVE_UP")
        player_position[1] -= 1
        self.assertEqual(player_position, [0, 0])
    def test_move_down(self):
        global player_position
        log_event("MOVE_DOWN")
        player_position[1] += 1
        self.assertEqual(player_position, [0, 1])
    def test_fire_bullet_hit_enemy(self):
        global player_score
        initial_health = enemies[0]["health"]
        fire_bullet()
        self.assertEqual(enemies[0]["health"], initial_health - 100)
        self.assertEqual(player_score, 200)
    def test_fire_bullet_miss_enemy(self):
        global player_position
        player_position = [0, 0]
        fire_bullet()
        self.assertEqual(enemies[0]["health"], 200)
    def test_player_injury(self):
        global player_health
        player_health -= 10
        log_event("INJURED")
        check_health()
        self.assertEqual(player_health, 190)
    def test_enemy_destruction(self):
        global enemies
        enemies[0]["health"] = 0
        check_health()
        self.assertEqual(enemies[0]["health"], 0)
if __name__ == '__main__':
    unittest.main()
```

Table 10: **The optimization objective aiming to minimize the difference between the generated output and the target proxy using the environmental feedback. According to the unit test results, Updating agent add more notes for the sub-task for Programmer 2 and Programmer 3. To see a complete updating process, please refer to Sec. 3.3**

```
{
    "composition": {
        "Programmer 1": "Implement bullet firing logic that decreases
        health points for both the player and enemies when hit. This
        includes creating a function to handle bullet movement and
        collision detection, updating health points accordingly, and
        logging the events in the game.log file.",
        "Programmer 2": "Implement health management to check for
        health points dropping to zero and handle tank destruction.
        This includes creating a function to check health status after
        each hit, updating the game state, and logging the destruction
        event in the game.log file. Additionally, ensure the game loop
        terminates properly when the player's tank is destroyed.",
        "Programmer 3": "Implement scoring logic to update the player's
        score when an enemy tank is destroyed and display the final
        score when the game ends. This includes creating a function to
        manage scoring and game-over conditions, and logging the final
        score in the game.log file. Ensure the final score is
        displayed correctly when the game ends."
    },
    "workflow": {
        "Programmer 1": [],
        "Programmer 2": [
            "Programmer 1"
        ],
        "Programmer 3": [
            "Programmer 2"
        ]
    }
}
```

Table 11: Unit test case: Game

Example 1	**Game Ghostly**
Requirement(partial)	3. If the ghost controlled by the player eats a superpellet (the big pellets), it gains the ability to eat other ghosts.
Subtask given by Test Organizer(partial)	"Programmer 5": "Test that each event in the task is implemented and that the logic triggered matches the conditions in the task description."
Generated test case(partial)	please see Tab. 12
Evaluation test case(partial)	please see Tab. 13

Table 12: **The partial generated test case for Game Ghostly.**

```
def test_eat_other_ghost(self):
    main.superpellet_active = True  # Activate superpellet
    main.move_ghost("up")           # Move to [1, 0]
    # more move_ghost function call omitted here
    with open('game.log', 'r') as log_file:
        logs = log_file.readlines()
        last_log = json.loads(logs[-1])
        self.assertEqual(last_log["event_type"], "eat_other_ghost")
        self.assertEqual(last_log["ghost_position"], [6, 5])
        self.assertNotIn([0, 5], main.other_ghost_positions)
```

Table 13: **The partial evaluation test case for Game Ghostly.**

```
def check_eat_other(self): # Check if Character can eat others
with superpellet
    try:
        print("==============Test eat other=============")
        process = self.start_game()
        time.sleep(0.1)
        pg.press('right',presses=1,interval=0)
        time.sleep(0.1)
        # more press omitted here
        process.terminate()
        print("terminated")
        # Read the log
        logs = self.read_log()
        print(logs)
        cnt=0
        for log in logs:
            if log["event_type"]=="eat_other_ghost":
                cnt+=1
        if cnt==2:
            print("Character can eat other with superpellet\n\n")
            return 1
        print("Character can eat other with superpellet\n\n")
        return 0
    except:
        print("eat_other error")
        return 0
```

Table 14: Unit test case: Website

Example 2	**Website DailyHealthTips**
Requirement(partial)	please see Fig. 3
Subtask given by Test Organizer(partial)	"Programmer 2": "Test whether the first page of the website is the login page and whether it can log in correctly based on the example data provided in the Task."
Generated test case(partial)	please see Tab. 15
Evaluation test case(partial)	please see Tab. 16

Table 15: **The partial generated test case for Website DailyHealthTips.**

```
'''
Test whether the first page of the website is the login page and
whether it can log in correctly based on the example data
provided in the Task.
'''
import unittest
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
import time
class TestDailyHealthTipsLogin(unittest.TestCase):
    def setUp(self):
        self.driver = webdriver.Chrome()
        self.driver.get("http://localhost:5000")
    def test_login_page(self):
        driver = self.driver
        # Check if the current page is the login page
        self.assertIn("User Login", driver.title)
        print("Login page title is correct.")
        # Locate username and password fields
        username_field = driver.find_element(By.ID, "username_field")
        password_field = driver.find_element(By.ID, "password_field")
        login_button = driver.find_element(By.ID, "login_button")
        # Input example data
        username_field.send_keys("john_doe")
        password_field.send_keys("securepassword")
        login_button.click()
        # Wait for the page to load
        time.sleep(2)
        # Check if redirected to Daily Health Tips page
        self.assertIn("Daily Health Tips", driver.title)
        print("Successfully logged in and redirected to
        Daily Health Tips page.")
    def tearDown(self):
        self.driver.quit()
if __name__ == "__main__":
    unittest.main()
```

Table 16: **The partial evaluation test case for Website DailyHealthTips.**

```
class TestCase(unittest.TestCase):
    def setUp(self):
        """Set up the Selenium WebDriver before each test."""
        self.driver = webdriver.Chrome()
        self.driver.get("http://localhost:5000")

    def tearDown(self):
        """Tear down the WebDriver after each test."""
        self.driver.quit()

    def login(self):
        self.driver.delete_all_cookies()
        username = "john_doe"
        password = "securepassword"
        # Performing login
        self.driver.get("http://localhost:5000")
        self.driver.find_element(By.ID, "username_field")
        .send_keys(username)
        self.driver.find_element(By.ID, "password_field")
        .send_keys(password)
        self.driver.find_element(By.ID, "login_button").click()

    # ===== Login Page Tests =====
    def test_login_page_elements(self):
        # Go back to the login page and check elements
        self.assertTrue(self.driver.find_element
        (By.ID, 'username_field').is_displayed())
        self.assertTrue(self.driver.find_element
        (By.ID, 'password_field').is_displayed())
        self.assertTrue(self.driver.find_element
        (By.ID, 'login_button').is_displayed())

    def test_login_page_functionality(self):
        # Assume user login is verified by the presence
        of daily tips page elements
        self.assertIn("User Login", self.driver.title)
        username = "john_doe"
        password = "securepassword"
        self.driver.find_element(By.ID, 'username_field')
        .send_keys(username)
        self.driver.find_element(By.ID, 'password_field')
        .send_keys(password)
        self.driver.find_element(By.ID, 'login_button').click()

        # Check if we are redirected to the daily tips page
        self.assertIn("Daily Health Tips", self.driver.title)
    # more test case omitted
```

Table 17: Updating process on RSD-Bench

RSD-Bench Game Tank	**Example**
Requirement(partial)	please see Tab. 5
Initial Code Organizer(iteration 0)	please see Tab. 6
Updating agent(iteration 1)	please see Tab. 10
Updating agent(iteration 2)	please see Tab. 18
Updating agent(iteration 3)	please see Tab. 19

Table 18: **Updating process on RSD-Bench Game Tank, iteration 2**

```
{
    "composition": {
        "Programmer 1": "Fix the health management logic to ensure
        that when the player's health drops to zero, the destruction
        event is logged correctly, and the game loop terminates
        properly. This includes modifying the `check_health` function
        to accurately log destruction events for both the player and
        enemies.",
        "Programmer 2": "Implement the game-over conditions to ensure
        that the game ends when the player's tank is destroyed or all
        enemy tanks are destroyed. This includes creating a function
        to manage the game-over state and logging the final score
        in the game.log file."
    },
    "workflow": {
        "Programmer 1": [],
        "Programmer 2": [
            "Programmer 1"
        ]
    }
}
```

Table 19: **Updating process on RSD-Bench Game Tank, iteration 3**

```
{
    "composition": {
        "Programmer 1": "Implement constant firing behavior for enemy
        tanks in four directions. This includes creating a mechanism
        for enemies to fire bullets at the player and logging the
        corresponding events.",
        "Programmer 2": "Implement the display of the player's score
        at the end of the game when the game is over. This includes
        modifying the game-over logic to ensure the final score is
        logged and displayed correctly.",
        "Programmer 3": "Fix the logging mechanism to ensure that
        the log file remains open during gameplay and logs events
        in real-time without closing prematurely. This includes
        reviewing the log_event function and ensuring it is called
        correctly throughout the game loop."
    },
    "workflow": {
        "Programmer 1": [],
        "Programmer 2": [
            "Programmer 1"
        ],
        "Programmer 3": [
            "Programmer 1",
            "Programmer 2"
        ]
    }
}
```

Table 20: Updating process on HumanEval

HumanEval	**Example**
Requirement	Write a function that takes a message, and encodes in such a way that it swaps case of all letters, replaces all vowels in the message with the letter that appears 2 places ahead of that vowel in the english alphabet. Assume only letters.
Initial Code Organizer(iteration 0)	"Programmer 1": "Identify and implement the logic to swap the case of all letters in the message.", "Programmer 2": "Implement the logic to replace all vowels with the letter that appears 2 places ahead in the English alphabet."
Generated test case(iteration 0)	please see Tab. 21
Main code(iteration 0)	please see Tab. 22
Updating agent(iteration 1)	"Programmer 1": AssertionError: encode('test') != 'TGST'. The actual output is: 'TgST'., "Programmer 2": AssertionError: encode('This is a message') != 'tHKS KS C MGSSCGG'. The actual output is: 'tHkS kS c MgSScGg'.
Final code(iteration 1)	please see Tab. 23

Table 21: **Updating process on HumanEval, Generated test case. iteration 0**

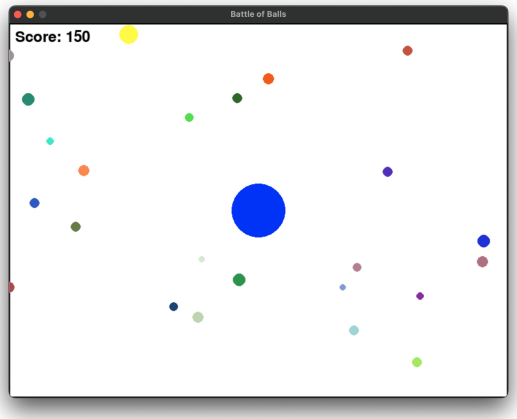
```
'''
Write a simple test case to verify the logic of the encode function.
'''
import unittest
from encode_function import encode
class TestEncodeFunction(unittest.TestCase):
    def test_example_case_1(self):
        self.assertEqual(encode('test'), 'TGST')
    def test_example_case_2(self):
        self.assertEqual(encode('This is a message'),
        'tHKS KS C MGSSCGG')
if __name__ == '__main__':
    unittest.main()
```

Table 22: **Updating process on HumanEval, Main code, iteration 0**

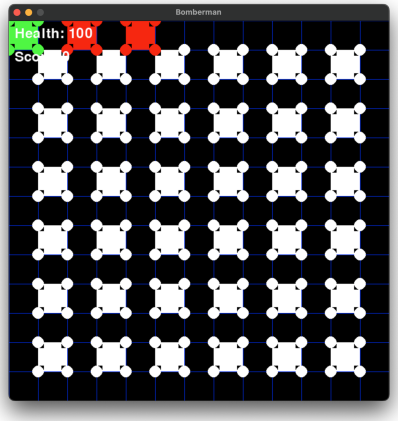
```
'''
def encode(message):
    vowels = 'aeiouAEIOU'
    encoded_message = []
    for char in message:
        if char.isalpha():
            # Swap case
            if char in vowels:
                # Replace vowel with the letter 2 places ahead
                if char.islower():
                    new_char = chr(((ord(char) - ord('a') + 2) % 26)
                    + ord('a'))
                else:
                    new_char = chr(((ord(char) - ord('A') + 2) % 26)
                    + ord('A'))
                encoded_message.append(new_char)
            else:
                # Just swap case for consonants
                encoded_message.append(char.swapcase())
        else:
            # If it's not a letter, we can just append it as is
            encoded_message.append(char)
    return ''.join(encoded_message)
```

Table 23: **Updating process on HumanEval, Final code, iteration 1**

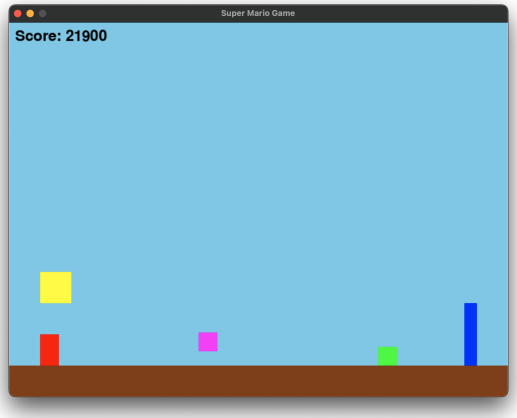
```
'''
def encode(message):
    vowels = 'aeiouAEIOU'
    encoded_message = []
    for char in message:
        if char.isalpha():
            # Swap case
            if char in vowels:
                # Replace vowel with the letter 2 places ahead
                and swap case
                if char.islower():
                    new_char = chr(((ord(char) - ord('a') + 2) % 26)
                    + ord('a')).upper()
                else:
                    new_char = chr(((ord(char) - ord('A') + 2) % 26)
                    + ord('A')).lower()
                encoded_message.append(new_char)
            else:
                # Just swap case for consonants
                encoded_message.append(char.swapcase())
        else:
            # Preserve spaces
            encoded_message.append(char)
    return ''.join(encoded_message)
```



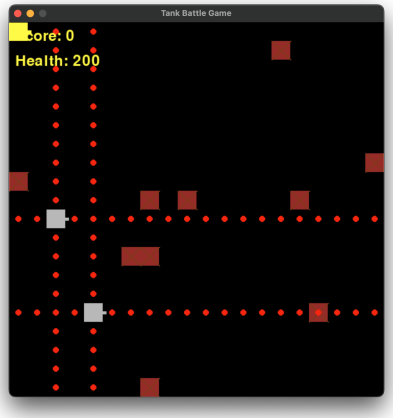
(a) Balls



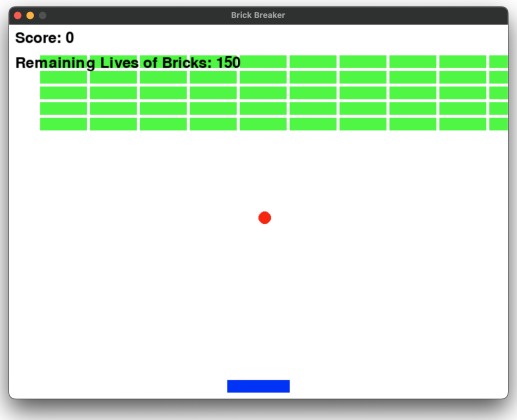
(b) Bomberman



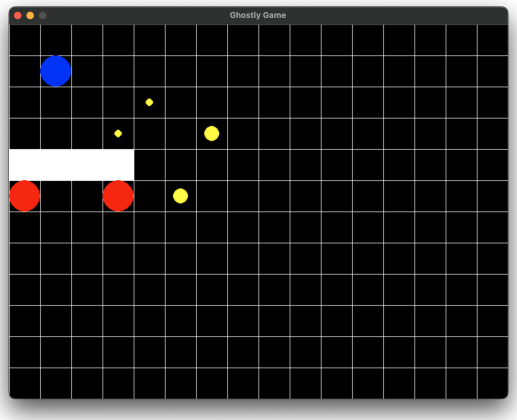
(c) Mario



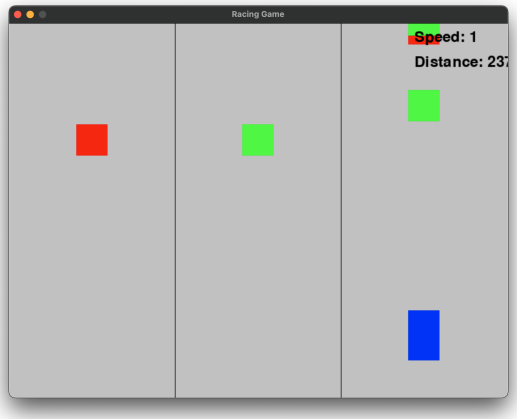
(d) Tank



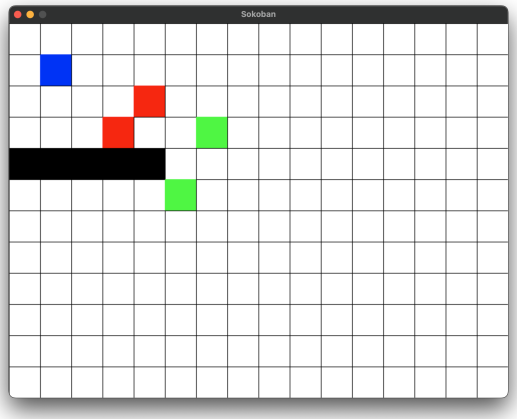
(e) Brick



(f) Ghostly

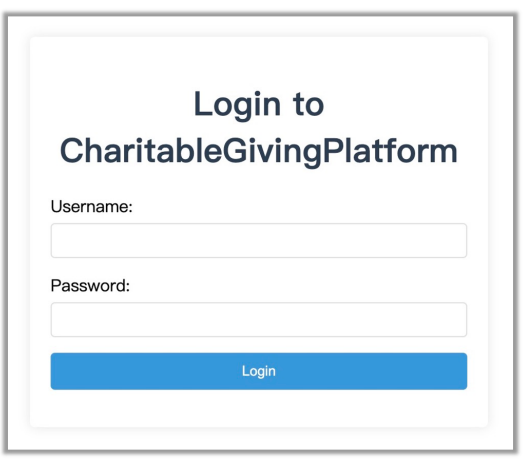


(g) Racing

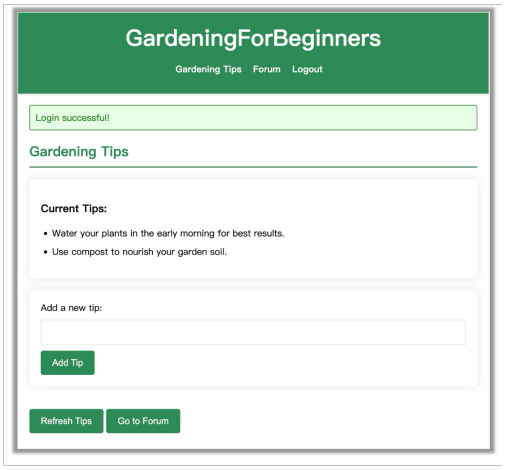


(h) Sokoban

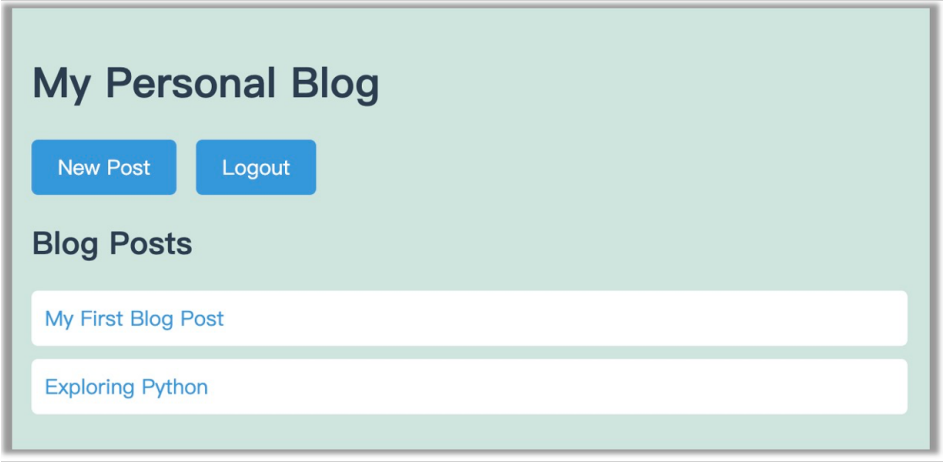
Figure 4: Games generated by EvoMAC.



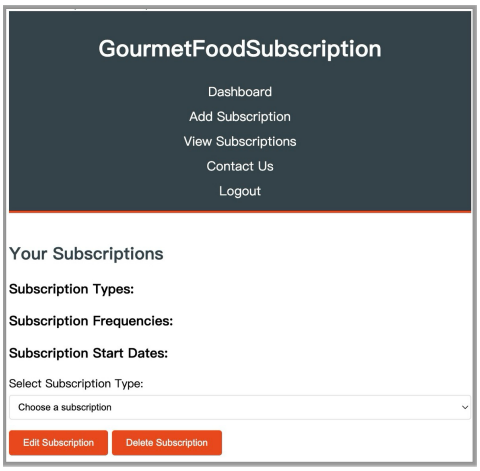
(a) CharitableGivingPlatform



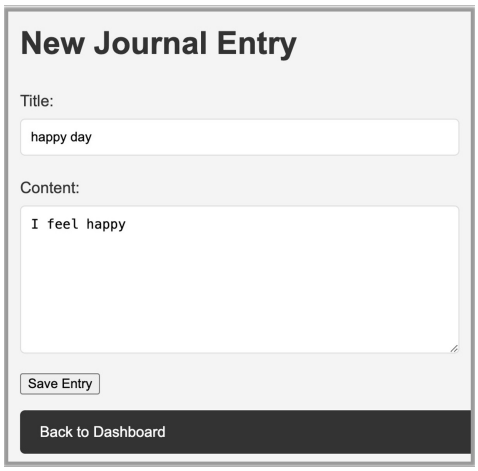
(b) GardeningForBeginners



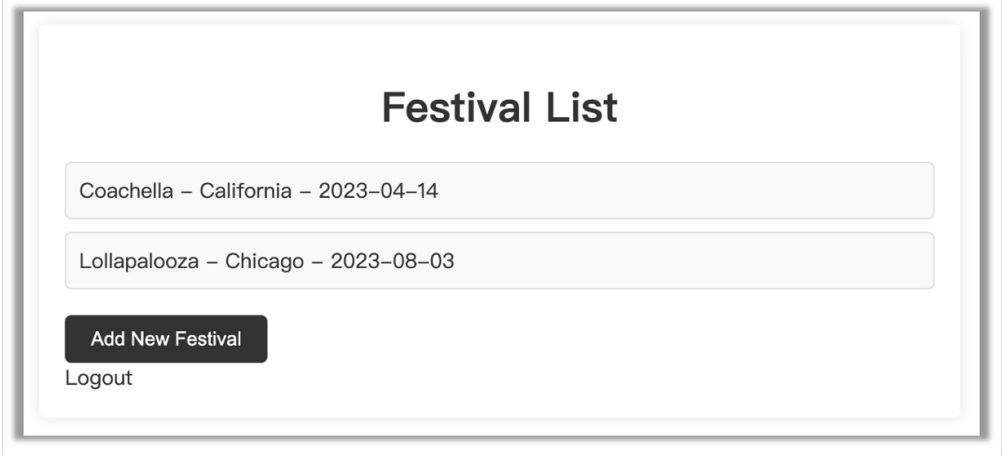
(c) PersonalBlog



(d) GourmetFoodSubscription



(e) OnlineTherapeuticJournaling



(f) MusicFestivalDirectory

Figure 5: Websites generated by EvoMAC.

REFERENCES

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. 2021.