codesprint

NATIONAL CODING COMPETITION

Qualifiers Task

Secondary Category 2026

POWERED BY mdia MALTA DIGITAL INNOVATION AUTHORITY

GOVERNMENT OF MALTA
MINISTRY FOR EDUCATION AND SPORTS
DIRECTORATE FOR STEM AND VET PROGRAMMES

× icecampus

Qualifiers Round Schedule

- Task Analysis (15 min)
- Task Development (60 min)
- Interval (15 min)
- Continuation of Task Development (60 min)
- Task Submission (15 min)
- **Top 10 announced for the Final round!**

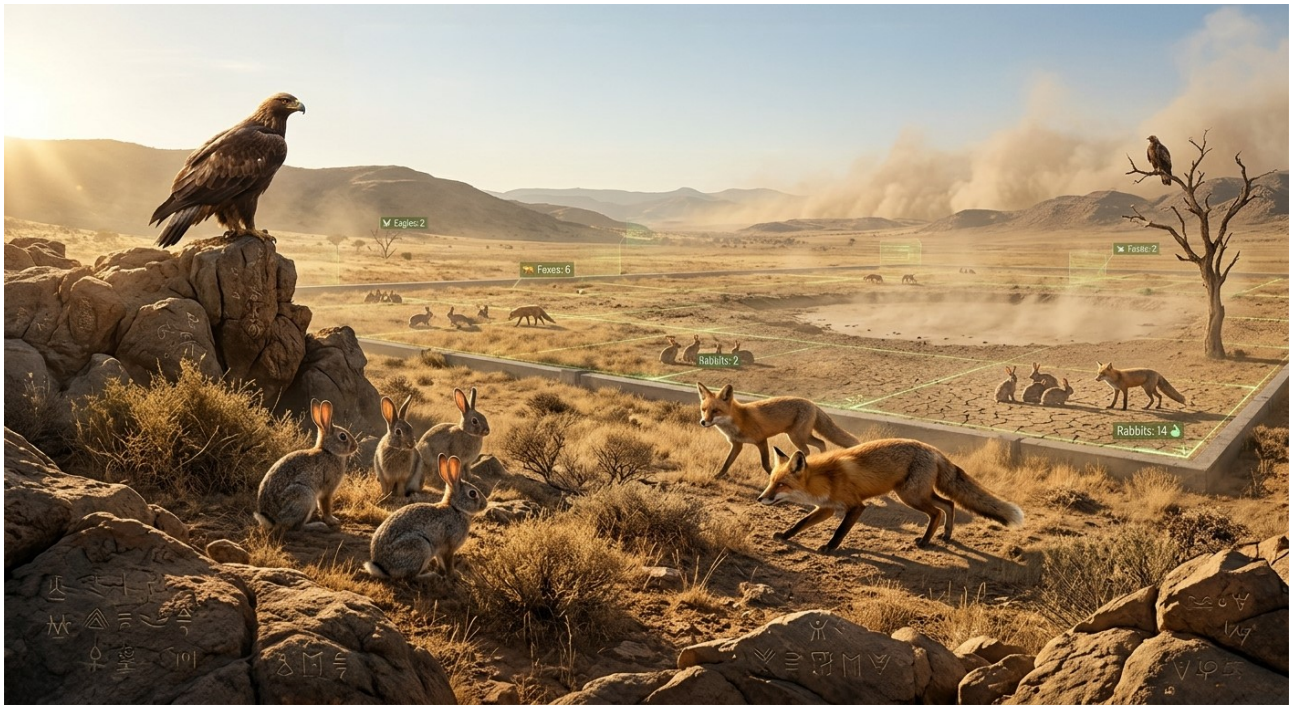
The Wild Dominion

The wilderness is no longer wild. It is controlled and planned, and it is highly competitive.

In Wild Dominion, two opponents enter a simulated ecosystem where survival depends on smart decisions. In a 6-by-6 grid, Eagles, Foxes and Rabbits compete for control. Every placement matters. Every move can shift the balance of power.

Predators hunt. Prey try to survive. At times everything seems stable, but unexpected environmental events can suddenly change the situation.

Will you think better than your opponent and build the strongest population, or will your animals disappear over time?



Your task is to build a Python program that runs a turn-based ecosystem battle between a user and the computer. The program may use either a text-based interface, OR a graphical user interface. The system must manage animal placements, movements, interactions, and environmental events across two separate grids.

Think carefully about how you structure your solution so every decision shapes what happens next.

Functionality #1: Program initialisation

When the program starts, it should:

- Display the title: 'The Wild Dominion Challenge'.
- Ask the user to enter a username.
- Create a 6 × 6 grid for the user, and another 6 × 6 grid for the computer.
- For the first game turn, each player (user and computer) must place x1 Eagle (E), x3 Foxes (F), and x6 Rabbits (R) on their respective grid.
- The user must select a valid position on the grid for each animal placement.

  💡 A position represents a single cell in the 6 x 6 grid.
- Input method may vary depending on interface. Users may choose animals placement by specifying the row and column, or by a mouse selection.
- The computer places animals randomly within the grid.

🔍 **Validation rules:**

- Username must contain letters only.
- Username must be at least 2 characters long, excluding spaces.
- Animals must be placed inside the grid. Invalid grid positions must not be accepted.
- Animals of the same player cannot be placed in the same position (row & column).

⚠️ **Important:** Do not display grids until all animals for both players have been placed.

💡 **A text-based sample visual is shown in Screenshot 1.**

```
TURN 1

🎮 John's animals placement:
Eagle placed at (x,y): 3,2
Fox #1 placed at (x,y): 2,1
Fox #2 placed at (x,y): 3,7
❌ INVALID POSITION

Fox #2 placed at (x,y): 3,5
Fox #3 placed at (x,y): 5,4
Rabbit #1 placed at (x,y): 1,1
Rabbit #2 placed at (x,y): 6,1
Rabbit #3 placed at (x,y): 2,1
🚫 Cell occupied by a Fox
🔁 Please choose another position

Position (1, 2) already occupied by a Fox!!

Rabbit #3 placed at (x,y): 5,2
Rabbit #4 placed at (x,y): 4,3
Rabbit #5 placed at (x,y): 1,4
Rabbit #6 placed at (x,y): 6,6

💻 Computer placement complete
```

Screenshot 1: 1st turn sample visual

Functionality 2: Gameplay Sequence

- The game runs for a maximum of 3 turns, including the first turn upon program initialisation.
- Each turn consists of:
  - User move, as per functionality 3 below.
  - Computer move, as per functionality 3 below.
  - Possible predator interactions, as per functionality 4 below.
  - A possible environmental event, as per functionality 5 below.
  - Display the updated user and computer grids.
  - Display the status of the turn, including:
    - any predator interaction/s
    - any environment event
    - The number of animals remaining for each player
- Gameplay ends after 3 turns have been completed, or when at least one player has no Foxes and no Rabbits remaining.

A text-based sample visual is shown in Screenshot 2.

```
   -----------        -------------
    USER GRID         COMPUTER GRID
   -----------        -------------

   1 2 3 4 5 6         1 2 3 4 5 6
 ┌─────────────┐     ┌─────────────┐
1│ R F . . . R │   1 │ . F . . R . │
2│ . . E . R . │   2 │ . . E . . . │
3│ . . . R . . │   3 │ R . . F . . │
4│ R . . . F . │   4 │ . . . . . F │
5│ . . F . . . │   5 │ . R . . . . │
6│ . . . . . R │   6 │ . . . . R . │
 └─────────────┘     └─────────────┘

------------------
  TURN 1 SUMMARY
------------------

 USER:
 Eagles: 1   Foxes: 3   Rabbits: 6

 Animals eaten → Foxes: 0   Rabbits: 2
----------------------------------------

 COMPUTER
 Eagles: 1   Foxes: 3   Rabbits: 4

 Animals eaten → Foxes: 0   Rabbits: 0
```

Screenshot 2: 1st Turn Results Summary

Functionality #3: Animal Movement

- Starting from the second turn, players may move their animals.
- Both players (user and computer) may move, or may not, move one or more than one animal.
- Each animal can move only **ONE** cell per turn in any direction.

🔍 **Movement Validation Rules:**

- Animals must stay within the grid. Invalid grid positions must not be accepted.
- Animals of the same player cannot move onto a cell occupied by another animal.
- Animals can only be moved once per turn.

⚠️ **Important:**

- Computer must select which animals to move at random;
- Computer chosen cell destination must be random;
- It must not react to or depend on the user's moves.

💡 **A text-based sample visual is shown in Screenshot 3.**

```
TURN 2

Do you want to move an animal? [Y/N]: y
Select animal to move (row,column): 3,1
⚠ There is no animal placed in this position

Select animal to move (row,column): 2,1
Select new position (row,column): 7,1
✘ POSITION NOT IN GRID

Select new position (row,column): 3,2
⦸ Cell already occupied by an Eagle

Select new position (row,column): 4,1
⚠ Cannot move more than one cell

Select new position (row,column): 2,2
✔ Fox moved from (2,1) → (2,2)
----------------------------------------

Move another animal [Y/N]: n
```

Screenshot 3: Visual sample of animal movement

Functionality 4: Predator Interactions

- When animals from both players occupy the same cell position (same row and column), interactions occur.
- If the animals are of the same species, both animals survive.
- If animals are not of the same species:
  - Eagle eats Fox and Rabbit;
  - Fox eats Rabbit;
  - Rabbit does not eat.
- Any animal that is eaten must be removed from the grid.

💡 These rules are applied after every turn.

Functionality #5: Environmental Events

On the third turn, one random event may occur:

- Pest Attack (25% probability): Removes 50% of one random species for both players *(Rounded to the nearest unit).*
- Foreign Predator (10% probability): Removes one random Eagle of one random player.
- Favourable Weather (20% probability): Adds two (x2) animals of one random species to one random player.
- Otherwise, no event occurs.

💡 **A text-based sample visual is shown in Screenshot 4.**

```
5 |  . . F . . .  |    5 |  . R . . . .  |
6 |  . . . . . R  |    6 |  . . . . R .  |

------------------
  TURN 3 SUMMARY
------------------

 JOHN:
 Eagles: 1   Foxes: 1   Rabbits: 3

 Animals eaten →  Foxes: 0   Rabbits: 1
-------------------------------------------------

 COMPUTER
 Eagles: 1   Foxes: 3   Rabbits: 2

 Animals eaten →  Foxes: 2   Rabbits: 0
-------------------------------------------------

 ENVIRONMENT EVENT
Pest Attack killed 50% of the Rabbits
-------------------------------------------------

Press Enter to check final score...
```

Screenshot 4: Visual sample of turn 3 summary results

Functionality #7: Scoring System

- At the end of the game, each player's score is calculated as follows:
  - Add 2 points for each remaining fox; and
  - Add 1 point for each remaining rabbit;

  Eagles equate to 0 points!

Game Result:

- The player with the highest points wins, OR
- If both players have the same number of points, the game ends in a tie.

A text-based sample screenshot is shown in Screenshot 5.

```
------------------
  FINAL SCORE
------------------

 JOHN:
 Eagles: 1   Foxes: 1   Rabbits: 1
SCORE: 3 points

 COMPUTER
 Eagles: 1   Foxes: 3   Rabbits: 1
SCORE: 7 points

    WINNER
   COMPUTER
```

Screenshot 5: Visual sample of the game ending

Functionality #8: Validation Rules

Besides the validation rules mentioned above, the program must:

- Prevent invalid input, and re-prompt when input is invalid.
- Avoid runtime errors.

Functionality #9: Modular Programming

- Structure your code using user-defined functions.
- Use dictionaries, lists, and/or tuples where appropriate.
- Include comments and apply Python naming conventions.

Name your program **sec_wildDominion.py**,
and submit as a Zip file named name_surname

Assessment Rubric

Core Functionality (30 points)	
Program Initialization	Displays title, accepts username, creates two 6 × 6 grids.
Game Sequence	Stops when turn limit is reached OR no Foxes/Rabbits remain.
Animal Setup Completion (User)	Player places all animals using valid empty cells.
Animal Setup Completion (Computer)	Computer places all animals at random using valid empty cells.
Movement Execution (User)	Applies animal movement correctly when valid (many, one or none).
Movement Execution (Computer)	Executes valid animal movement at random (many, one or none).
Predator Interactions	Correctly resolves all interactions based on species hierarchy.
Environmental Event Probability	Correct event probability mechanism applied on turn 3 only.
Environmental Event Application	Applies correct effect according to event type.
Turn Grid Update	Correctly updates grid state after every turn.
Turn Summary Update	Correctly updates movement, interaction, and events.
Scoring Calculation	Computes final scores correctly based on remaining animals.
Result Determination	Correctly identifies winner or tie.
Input Handling	Manages all the required user input.
Other Functionalities	Other possible 'hidden' functionalities work correctly.
Validation Checks (14 Points)	
Username	Enforces letters only and minimum length.
Grid Boundary	Rejects positions outside grid.
Animal Overlap	Prevents multiple animals of same player in same cell.
Animal Placement Completion on 1st turn.	All required animals are placed before proceeding.
Movement Distance	Enforces movement of exactly one cell.
Movement Frequency	Enforces each animal moves at most once per turn.

Avoidance of Run Time Errors	Program does not crash unexpectedly during runtime.
Technical Implementation (6 Points)	
Data Structures Usage	Appropriate and effective use of lists and/or dictionaries and/or tuples.
Modular Programming	Uses functions for key components (placement, movement, interaction, events, scoring) or any other modular approach.
Code Efficiency & Readability	Code is efficient, avoids unnecessary complexity, and is easy to understand.
User Experience & Good Programming Practices (10 Points)	
Interface Clarity	Output is clear and user-friendly (text or GUI).
User Prompts	Instructions and prompts are clear and understandable.
Python Conventions	Adheres to Python naming conventions (e.g., snake_case), uses meaningful variable names, and maintains consistent indentation.
Code Comments	Includes clear and concise comments to explain complex logic, functions, and important sections of the code.
File Naming	The program file is correctly named according to the specified format.
0 – Not Satisfactory | 1- Partially Satisfactory | 2- Entirely Satisfactory	
Maximum Score: 60 + 2 for every extra feature*	

* Extra features should extend the task functionality or demonstrate technical creativity WITHOUT altering or interfering with the core rules or required functionality of the task.

codesprint

ORGANISED BY

GOVERNMENT OF MALTA
MINISTRY FOR EDUCATION AND SPORTS
DIRECTORATE FOR STEM AND VET PROGRAMMES

×

icecampus

POWERED BY TOP BRANDS

MAIN TITLE SPONSORS

mdia MALTA DIGITAL INNOVATION AUTHORITY

me direct

COMMUNITY & EXPERIENCE SPONSORS

pwc

melita

Bolt

KNIGHTS COLLEGE

JOli

XJENZA MALTA

SUPPORTING SPONSORS

mita

edQuanta RESEARCH FOR A BETTER TOMORROW

computime BUSINESS SYSTEMS ENGINEERING

9H

MEDIA PARTNERS

Lovin Malta

TIMES OF MALTA

meet INC.