Computational Thinking and Programming

1 Projects

1.1 Maze Solver

Implement a maze solver using Python's Turtle module. The maze will be represented as a grid, with walls and a clear path. The turtle must find its way from a designated start point to an end point.

Deliverables

1. A report explaining the maze representation, the algorithm used for solving the maze, and how the Turtle module was utilized.
2. The implementation of the maze solver, pushed to a GitHub repository, using only Python and the Turtle module. It should be well-commented to explain key sections and logic of the maze.
3. A video showing the turtle solving different mazes, demonstrating the algorithm's effectiveness.

1.2 Snake Game

Implement the classic Snake game. The snake moves across the screen, growing longer as it eats food, and the game ends if the snake runs into the screen border or itself.

Deliverables

1. A report explaining the mechanics of the game, how data representations were used for the snake's movement and growth, and collision detection logic. Likewise, the report should also discuss the faced challenges and how they were overcome.
2. A fully functional Snake game code, uploaded to a GitHub repository, using standard Python libraries and the Turtle module. It should be well-commented to explain key sections and logic of the game.
3. A video of the game being played, showing the snake growing and the game's end conditions.

1.3 Tic-Tac-Toe

Create a two-player Tic-Tac-Toe game. The game should allow players to take turns to mark Xs and Os on a 3×3 grid and determine the winner or a tie.

Deliverables

1. A report detailing the implementation of the game logic, user interaction, and the algorithm to check for a win or tie. Likewise, the report should also discuss the faced challenges and how they were overcome.
2. The Tic-Tac-Toe game code, hosted on GitHub, utilizing only Python and the Turtle module. It should be well-commented to explain key sections and logic of the game.
3. A video showing a sample game between two players, including win/tie scenarios.

1.4 Pong Game

Implement a basic version of the Pong game. Two paddles on either side of the screen hit a ball back and forth. Points are scored when one player misses the ball.

Deliverables

1. A report outlining how the game mechanics were implemented, including paddle movement, ball physics, and score keeping. Likewise, the report should also discuss the faced challenges and how they were overcome.
2. The Pong game source code, available on GitHub, written using Python and the Turtle module. It should be well-commented explaining the key sections and logic of the game
3. A gameplay video showcasing the functioning of the game, including paddle movements and scoring.

1.5 Brick Breaker

Develop a Brick Breaker game where a ball moves across the screen, breaking bricks and bouncing off a paddle controlled by the player. The goal is to break all bricks without letting the ball fall.

Deliverables

1. A comprehensive description of the game's design, including brick layout, ball movement, collision detection, and game progression. Likewise, the report should also discuss the faced challenges and how they were overcome.
2. The complete game code on GitHub, using Python and the Turtle module exclusively. It should be well-commented to explain key logic of the game.
3. A video illustrating the gameplay, brick breaking mechanics, and the game's user interface.

1.6 Memory Puzzle

Create a Memory Puzzle game. The game presents a grid of cards facing down. Players turn over two cards at a time, with the goal of finding matching pairs.

Deliverables

1. A detailed report on the game's logic, including the grid layout, card flipping mechanism, and matching logic. Likewise, the report should also discuss the faced challenges and how they were overcome.
2. The game's source code, uploaded to GitHub, coded with Python and the Turtle module. It should be well-commented to explain key sections and logic of the game.
3. A gameplay video demonstrating the card flipping and matching process.

1.7 Hangman

Implement the word game Hangman. The game selects a word, and the player guesses letters. Incorrect guesses result in a hangman being drawn step by step.

Deliverables

1. A report explanaing how the game chooses words, captures guesses, and the logic for drawing the hangman and checking win/lose conditions. Likewise, the report should also discuss the faced challenges and how they were overcome.
2. The Hangman game code, pushed to a GitHub repository, using only standard Python libraries and the Turtle module. It should be well-commented to explain the key sections and logic of the game
3. A video showing a few rounds of the game, including both winning and losing scenarios.

1.8 Treasure Hunt

Develop a Treasure Hunt game using Python's Turtle module. In this game, the player controls a turtle character navigating through a maze or map to find hidden treasures. The game should include obstacles that the player must avoid and possibly other elements like keys to unlock areas or time limits.

Deliverables

1. A report describing the design of the game, including the maze or map layout, placement of treasures, obstacles, and any additional gameplay elements like keys or time limits. The report should also explain the logic for the turtle's movement, collision detection with obstacles and treasures, and the overall game flow. Likewise, the report should also discuss the faced challenges and how they were overcome.

2. An implementation of the Treasure Hunt game, pushed to a GitHub repository. The code should use only Python and the Turtle module, without relying on external libraries. It should be well-commented to explain key sections and logic.

3. A video showcasing the gameplay of the Treasure Hunt. This should include the start of the game, the turtle navigating through the maze, interacting with various elements like collecting treasures or avoiding obstacles, and any end-game scenario (such as finding all treasures or hitting an obstacle).

1.9 Whack-a-Mole

Develop a Whack-a-Mole game using Python's Turtle module. In this game, "moles" pop up at random positions on the screen for a short period, and the player must "whack" them by clicking on them with the mouse before they disappear. The goal is to whack as many moles as possible within a set time limit.

Deliverables

1. A comprehensive report explaining the game's design, including the algorithm for random mole generation, the timing mechanism for moles appearing and disappearing, and the method for detecting mouse clicks. The report should also discuss the faced challenges and how they were overcome, as well as any interesting features or variations added to the game.

2. The complete source code for the Whack-a-Mole game, uploaded to a GitHub repository. The code should solely use Python and the Turtle module, and be well-documented to explain major sections and logic.

3. A gameplay video that clearly demonstrates how the game is played. It should show moles popping up, being whacked, and the score mechanism. The video can also highlight any special features or levels included in the game.

2 Expected Learning Outcomes

The projects aim to enable everyone to:

1. Enhance their Computational Thinking mindset by developing a stronger understanding of algorithmic thinking, problem-solving, and the logic behind programming.

2. Improve their Python knowledge through a practical application, and hence, solidify their grasp of Python programming concepts.

3. Understand the basics of how games work, including user interaction, game physics, and graphics.

4. Encourage innovation and creativity in how one designs and implements game features.

5. Serve as an excellent portfolio pieces to showcase students' algorithm skills.

3 Due date

- December 8th, 2023

4 Groups

The groups were randomly generated using the code below, considering a list of students ordered by their surnames.

```python
import random

random.seed(42)
students = list(range(1, 39))
random.shuffle(students)

groups = []

for _ in range(7):
    groups.append(students[:4])
    students = students[4:]

for _ in range(2):
    groups.append(students[:5])
    students = students[5:]


projects = [f"Project {i}" for i in range(1, 10)]
random.shuffle(projects)

group_assignments = {projects[i]: groups[i] for i in range(len(groups))}

for project, group in group_assignments.items():
    print(f"{project}: {group}")
```

Project	**Members**
Pong Game	DAS Millan SABOURDIN Alienor CESBRON DARNAUD Toscane ROSTOMYAN Alen
Snake Game	SU Yuhan PASCAL Paul DE LEUSSE Arthur BENNIS Ghali
Maze Solver	MAURAISIN Gabriel PELTIER William MORVAN Arthur BENGUEDOUAR Abderrahmene
Memory Puzzle	CAUDRELIEZ Oscar JAKKLI Arya BENANI DAKHAMA Oulaya MAULBON D'ARBAUMONT Marine
Brick Breaker	LAZARIS Michael MACE Ulysse HADJAL Apolline VERHAEGHE Alexander
Hangman	RIZK Adriana MENKOR Othmane AJBAR Walid VANNSON Alexis
Tic-Tac-Toe	DEYRIS Elise-Calisthe HORLIN Octave BEKHTIAR Aya SURACE GOMEZ Andrea
Treasure Hunt	NOTKIN Daniil MAI Yiwen LE BARS Kerrian BOUEV–DOMBRE Anastasia BRAHAM Fatma Zahra
Whack-a-Mole	ERRAJI Yassine GERARD Ashanty HALLAGE Mia AMAR-ROISENBERG Simon BOURGOGNE Alessia