**Project end of module: Advanced Algo & Design Patterns,** **Graph Theory**.

**Development of a Dynamically Generated Maze Game:**



Link Git Hub: https://github.com/safae97/unity-2d-dynamic-maze-game.git

Created by: Fatima Zahra Ait Hssaine.

Safae Hammouch.

Mentored by: M.Lotfi EL AACHAK.

M.Abderrahim GHADI.

**Acknowledgments:**

We would like to express our heartfelt gratitude to our professor for their invaluable guidance and support throughout this semester. Their expertise, dedication, and encouragement have been instrumental in our learning journey. Thank you for your tireless efforts in imparting knowledge and fostering our growth. Your mentorship has truly made a difference, and I am grateful for the opportunity to learn from you.

## Introduction:

This report details the comprehensive development process of a dynamically generated maze game, which I developed in collaboration with my colleague. The project leverages graph theory algorithms, Singleton and inheritance design patterns, and the Unity game engine to create a unique and engaging gaming experience. Our primary objective was to build a system capable of generating dynamic game levels, incorporating effective pathfinding and strategic elements.

### Project Planning and Objectives:

1-Random Generation of Game Map.

2-Game concept.

3-Agent Pathfinding Algorithms.

4-Enemy and Coin Generation.

5-Design Patterns and Programming Paradigms.

6-Challenges Faced.

**1-Random Generation of Game Map**

- Approach: Implemented algorithms based on graph theory principles to generate random game maps.

We implemented algorithms to generate random game maps using principles of Graph Theory. We explored different graph structures to represent the game world and decided to create a maze that is generated each time using the DFS (Depth-First Search) algorithm.

The DFS algorithm for maze generation works as follows:

1. We start at an initial position in an empty grid.
2. At each step, we mark the current cell as visited.
3. We randomly select an unvisited neighboring cell.
4. If an unvisited neighboring cell exists, we choose it as the next cell, create a path between the current cell and the new cell, and recursively proceed from the new cell.
5. If no unvisited neighboring cell exists, we backtrack to the previous cell and continue the search.
6. The process repeats until all cells have been visited, thus generating a complete maze.

This process ensures that each maze is unique with every execution of the algorithm, ensuring that every cell is accessible and forming complex and random paths, which is ideal for game maps.

We created four rooms, to manage transitions between different game rooms, ensuring varied experiences in each room. we start by generating a random initial room and construct a complete undirected graph with rooms. Using BFS, it creates a random sequence of rooms to visit. Each room class inherits common maze generation logic and implements unique features such as different corners and enemies. When the player triggers a transition, the script loads the next piece in the sequence. This approach ensures reusable maze logic while providing unique gameplay experiences in each room.

**2-Game concept:**

In the third scene, the player is faced with the challenge of finding the solution to the maze within a set amount of time. If the timer runs out before the player finds the exit from the maze, he loses the game. This mechanic adds a dimension of time pressure to the game, challenging players to quickly navigate through the maze while remaining focused to avoid defeat.

In addition to navigating the maze and avoiding enemies, the player has the option to fire projectiles to eliminate enemies. When the player comes into contact with an enemy, their health decreases, mimicking the damage taken. However, the player also has the option to wait a brief period until enemies disappear, allowing them to continue without engaging in combat. Throughout the game, the player collects coins, which serve as both a reward for exploration and a way to improve their score. This combination of game mechanics adds depth and strategy to the player experience, providing various paths to success while navigating the maze and completing challenges.

### 3-Agent Pathfinding Algorithms:

- Algorithms Used: DFS, A\*, and Dijkstra algorithms.

In our second scene, we implemented three pathfinding algorithms DFS, Dijkstra, and A\* to navigate through the maze. When entering this room, a panel allows players to select their preferred algorithm. Once an algorithm is chosen and the path is found, the "auto-play" feature automatically guides the player along the determined path. The maze's rooms are stored in a graph structure, with each room represented as a node. Algorithms like DFS, Dijkstra, and A\* utilize this graph to traverse the maze efficiently, considering connections between adjacent rooms to find the optimal path from start to finish.

Each pathfinding algorithm operates differently to find the optimal path through the maze.

**DFS (Depth-First Search):** DFS explores as far as possible along a branch of the tree before backtracking. In the context of pathfinding, this means that it thoroughly explores different routes until it reaches the destination point or there are no more paths to explore. DFS is fast because it explores paths quickly without guaranteeing optimality but is memory efficient, but it does not guarantee the shortest path.

**Dijkstra:** Dijkstra travers nodes in the maze in order of increasing cost from the start node, updating the costs of all neighboring nodes. It guarantees finding the shortest path between two nodes, but can be costly in terms of time and memory for large graphs due to the repeated cost updates.

**A\***: uses a heuristic approach to estimate the cost of moving from each node to the destination. It combines the actual cost already traveled with an estimate of the remaining cost. A\* utilizes this information to prioritize exploring nodes that are most promising in terms of total estimated cost. A\* is generally faster than Dijkstra as it does not traverse all nodes, but it requires an admissible heuristic to guarantee optimality.

### 4-Enemy and Coin Generation:

- Objective: Develop a system for random generation and placement of enemies and coins within the game map.

we implement the random generation and placement of enemies and rooms on the game map. When creating the maze, existing rooms and enemies are eliminated before new ones appear. Using the SpawnEnemiesAndCoins method, enemies and coins are strategically placed in random positions within the maze, ensuring a diverse and engaging gameplay experience. The script cleverly avoids placing enemies and pieces in the same position and avoids overlap by excluding positions where enemies have already spawned. This approach allows for a balanced distribution of challenges and rewards within the maze, thereby enhancing player immersion and enjoyment.

### 5-Design Patterns and Programming Paradigms:

In this part of the game development process, we implemented the Singleton design pattern to ensure that only one instance of the Player's Health and Coin Collection is created and maintained throughout the game. By using the Singleton pattern, we guarantee that there is only a single instance of these important game elements, preventing multiple instances from being created and ensuring data consistency across different scenes.

## 6-Challenges Faced:

As beginners in game development, we faced several challenges while working with Unity. We dedicated a considerable amount of time to learning how to use Unity's interface and understand its features.

At the outset, we encountered a challenge in ensuring consistency across scenes while facilitating diverse experiences. However, we resolved this by implementing inheritance with the maze class, allowing for scene-specific variations while maintaining a common base structure, thus promoting code reusability and facilitating scalability in our game development framework.

## Conclusion:

The project was a significant learning experience in game development, graph theory, and the application of design patterns. Despite the challenges faced, the final product met our objectives, providing a unique and engaging game experience. The knowledge and skills gained during this project will undoubtedly be valuable for future endeavors in game development.

## References:

- Procedural World Generation Using Graph Theory.  
- Unity Documentation and Tutorials.  
- Various online resources on graph theory and pathfinding algorithms.