# COM377 Coursework (Programming III, CRN: 41812)– Maze Game

## 1. Aims

This coursework aims to develop a C# implementation of the interactive Maze game in Microsoft Visual Studio .NET. The topics involved include GUI programming, event handling, animation, and multimedia. The task to be completed will require students to demonstrate use of software engineering tools for configuration management, testing and system deployment.

## 2. Objectives

On completion of this coursework you should be able to:

- construct a user-friendly GUI using various GUI controls provided by Visual Studio
- implement the main features of the event handling model in .NET using C#
- understand how to play sound, draw images and generate animation in C#
- develop a team project using Visual Studio Online
- have a better understanding of software configuration management and version control

## 3. Tasks

You are required to develop a team project for a Maze Game using Visual Studio Team Services (https://www.visualstudio.com/team-services/). There are two actors in this game, i.e. a player and a guard. The player is navigated through the maze by using 4 arrow keys. The guard is under computer control and initially moves randomly around the maze. However, when the distance between the player and guard is less than a threshold defined by a user, the guard enters the intelligent mode, i.e. it will utilize some artificial intelligence techniques to find the shortest path to catch the player. Neither the player or the guard can move through the walls or the edges of the maze. The player will lose a life if it collides with the guard and it is given three lives before the game is over. The game should also have a timer to record the time users have spent playing the game.

## Basic requirements

1) The basic requirement for this assignment is to use version control facilities provided by Visual Studio Team Services to manage a team project for the development of a Maze game which includes at lease one player and one guard.

2) The assignment is a group project expected to be completed in a group of 3. Each member **must** contribute to the code development and demonstrate his/her contribution in the demo – **your submission will not be awarded a mark unless a demo has taken place.**

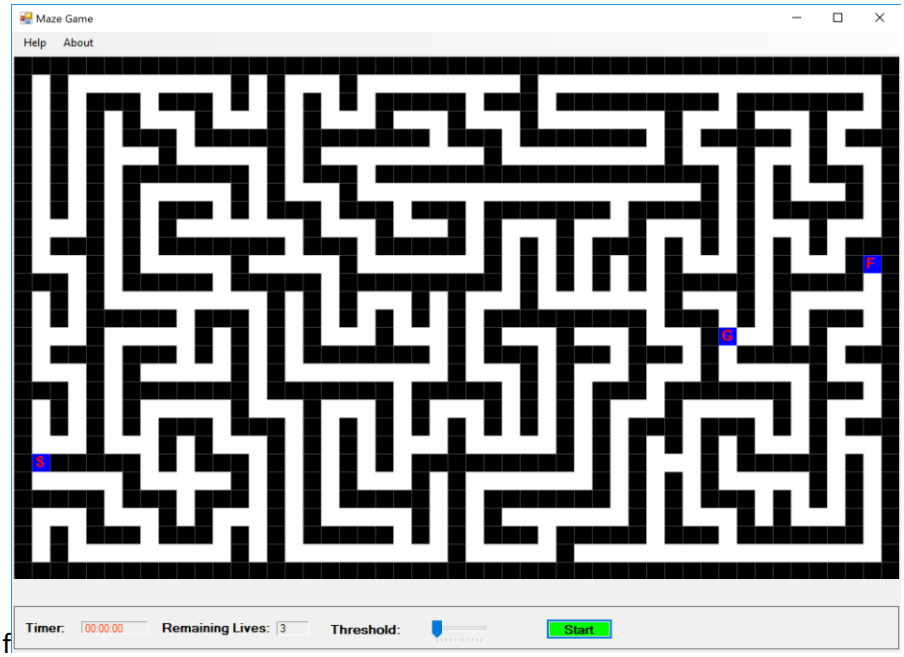
3) The number of player's lives left needs to be updated by using **event handling techniques** in C#. You need to define your own event class deriving from System.EventArgs to represent custom data. An example can be found in the lab exercise in which a **ScoreInfoEventArgs** class is defined to represent score as shown below:

```
public class ScoreInfoEventArgs : EventArgs
{
    private int score;


    public ScoreInfoEventArgs(int score)
    {
        this.score = score;
    }

    public int Score
    {
        get { return score; }
        set { score = value; }
    }
}
```

- An example of layout is illustrated below but the precise user interface design is your choice. However, clearly it will require an area for drawing and redrawing the maze; an area for the slider control, which allows a user to select the threshold; and areas where the time spent and the number of lives left are displayed.




f

## Basic features

### 1) GUI

- The “Help” menu that provides instructions on how to play the game and the “About” menu that must clearly show your group ID, name(s) and student number(s).
- Allow users to start/pause/resume the game by a keyboard or a mouse
- Display both the time spent and the number of player's lives left
- Allow users to set the threshold using a slider to determine when the guard will enter the intelligent mode

### 2) Control

- **The player is moved using keyboard  for example 4 arrow keys.**

- **The guard moves under computer control using both intelligence and random modes**.

    a. The guard starts with random movement, i.e. guard moves in one compass direction (N, S, E or W) until they reach an intersection, at which point it can turn left or right. It will change to another compass direction if it hits an obstruction/walls.

    b. Under intelligence mode: when the distance between the player and the guard is less than a threshold which can be defined by a user, the cat enters the intelligent mode, i.e. it will find the shortest path to catch the player. The distance between the player and guard can be estimated using Euclidean metric.

    Let *pRow* and *pCol* be the coordinate of the player and *gRow* and *gCol* be the coordinate of the guard, the distance between the player and the guard, i.e. $D_{pg}$ can be defined using the following formula:

$$D_{pg} = \sqrt{(pRow - gRow)^2 + (pCol - gCol)^2}$$

**3) Animations:** All characters are animated when they are moving. For example, guard eyes move.

**4) Sound effects and music** – The opening and finishing tune, player dying, etc.

**5) Game rules**

- When a guard touches the player, the player loses a life and the player goes back to the starting point.
- The player is given three lives before the game is over.

## Advanced features

1) Extra levels to the game: when the player gets to the finish line, go on to the next level.
2) Creating unit tests for some methods you developed using Microsoft Visual Studio and detail testing results in your report.
3) The use of advanced Object-Oriented Programming features such as inheritance and polymorphism
4) Any additional features not mentioned above – to exercise your creativity, you are welcomed to add any enhancement to the game.

## 4. Resources

1) Cell.cs – This class is used to construct a maze which can be represented as a 2-dimensional array. Each cell in the maze is addressed by a pair of (row and column) coordinates. The type of cell indicated by a char variable is determined by the input map file (see below). **Don't make any changes to this class** but you can extend it by creating a subclass of it.
2) level1.txt – An example of a map file used to set up the layout of a maze. The encoding is illustrated below. If you want to implement your own version of the maze, the following encoding scheme **must be followed** otherwise **your submission will be assigned a mark of zero**.
    a. 'w'– Wall
    b. 'p' – Path
    c. 's' – Starting position
    d. 'f' – Finishing line
    e. 'g' – Guard initial position

## 5. Marking Criteria

- Implementation of basic function (55%)

    Quality of the user interface (15%)

    Implementation of custom event handlers to update the lives left (5%)

Implementation of game rules (10%)
Movement of the player and guard (15%)
Animation (5%)
Sound effects (5%)

- Implementation of advanced features. You need to detail your implementation in your report (25%)
  Adding an extra level (5 marks)
  Creating a unit test for at least two methods (5 marks)
  The demonstration of using advanced OOP features (5 marks)
  Implementation of additional features (10 marks)
- Report (20%). You are required to include sufficient details for the following parts:
  Demonstration of configuration management and interaction between members supported by for example a check-in/check-out history for version control and a list of tracking Bugs, Tasks and other work items (10%)
  How to update the lives left using event handling (5%)
  The contribution of each member to **code development** (5%).
  - For each member, a brief description of your contribution including the location of code written by yourself should be provided
  - A summary table outlining the code examples of where each requirement can be found in Table 1

**Table 1: Code examples of where each requirement can be found**

| Requirement | Filename | Relevant Line of Code | Contributor |
|---|---|---|---|
| Pause and Resume | | | |
| Help and About menus | | | |
| Display the time spent | | | |
| Display the lives left | | | |
| Allow users to set the threshold using a slider | | | |
| Player's movement | | | |
| Random movement of the guard | | | |
| Intelligent guard | | | |
| Sound effect and music | | | |
| Animations | | | |
| The player loses a life if it collides with the guard | | | |
| The game is over when the player lose all his lives | | | |
| Any advanced features | | | |

## 6. Hand In Instructions

This assignment is expected to be completed in a group of 3. **Each** group must submit a signed 'team-assessment declaration' form, to indicate what percentage contribution each member made to the overall effort. For any submission deviating from an equal split in the effort, marks for this assignment will be scaled according to the declarations made. Only ONE copy of the solution for each group is required with the coversheet clearly showing both student names and student numbers.

**Specific submission guidelines (PLEASE READ CAREFULLY)**

1. The completed assignment including source code, report and any resource files such as images and music and any supporting documents along with the signed Team Declaration form must be zipped and uploaded to Blackboard Learn by **noon 12:00 on Monday 30 April 2018**. Each group is only required to submit ONE copy of the assignment with all names/IDs clearly marked in the coversheet which can be found on Blackboard under Assignments/Docs/ folder. Your submission should be in the form of a single zip file called **YourGroupID.zip,** i.e. "Group1234.zip".
2. A paper copy of the "Team Declaration form" signed and completed by each team member, declaring their contribution to the team effort must be handed to the markers during the demo which will take place after the submission. Please note that each member must be present in the demo and **your submission will not be awarded a mark unless a demo has taken place.**
3. Please ensure that you keep a secure backup electronic copy of your assignment and retain a screenshot of your successful submission.

*This assignment constitutes 60% of the total mark for the module.*

**PLEASE NOTE:**

**Avoid plagiarism: Do not try to pass off any downloaded code as your own work. You must acknowledge all sources. This includes information from the Internet for example the music and the images of player and guard. Plagiarism is a serious offence. Please refer to www.ulster.ac.uk/academicservices/student/plagiarism.pdf for further information.**