**Turtle Challenge Solution Description**

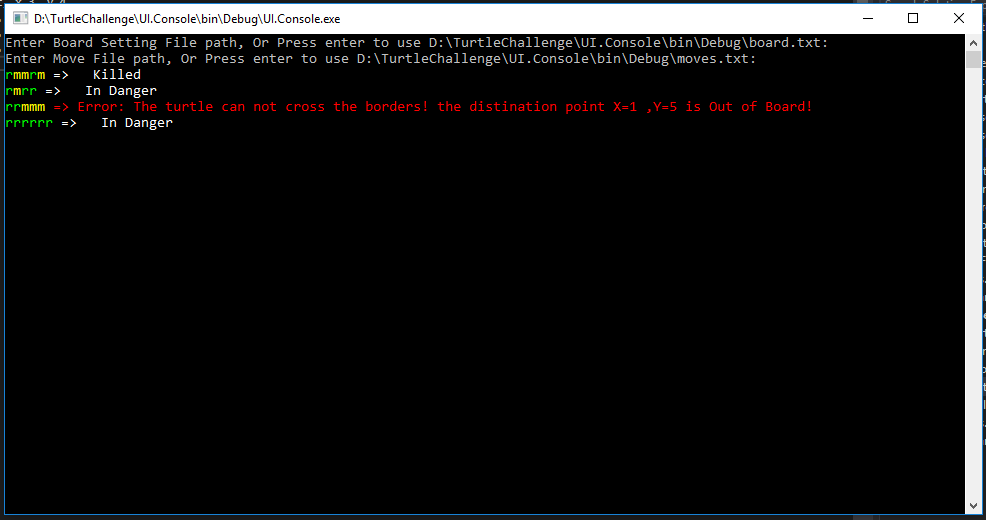
This document contains a brief description of the implemented solution for “Turtle Challenge” as the following sections:

* Capabilities
* Modules
* Class design and Algorithm

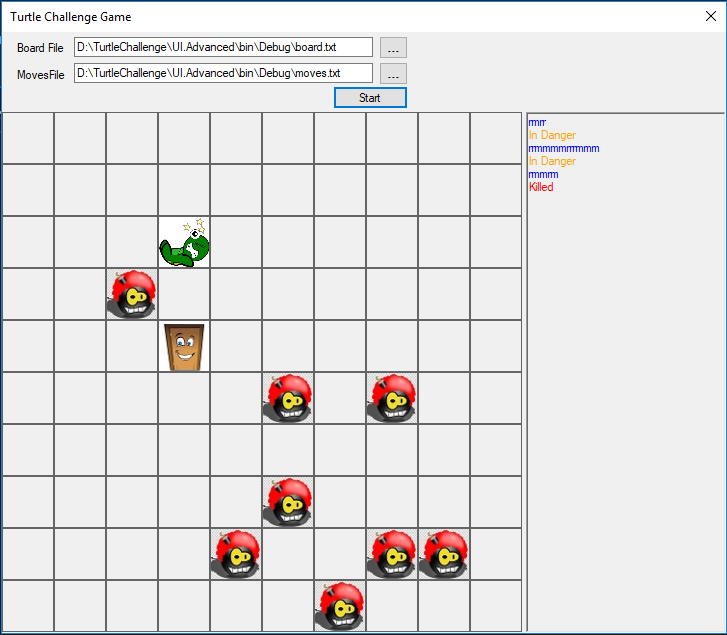
**Capabilities:**

 The implemented solution Accept tow files which one of them contains the Mine Filed definition, and the other one contains the start position and directions of the Turtle besides its movements;

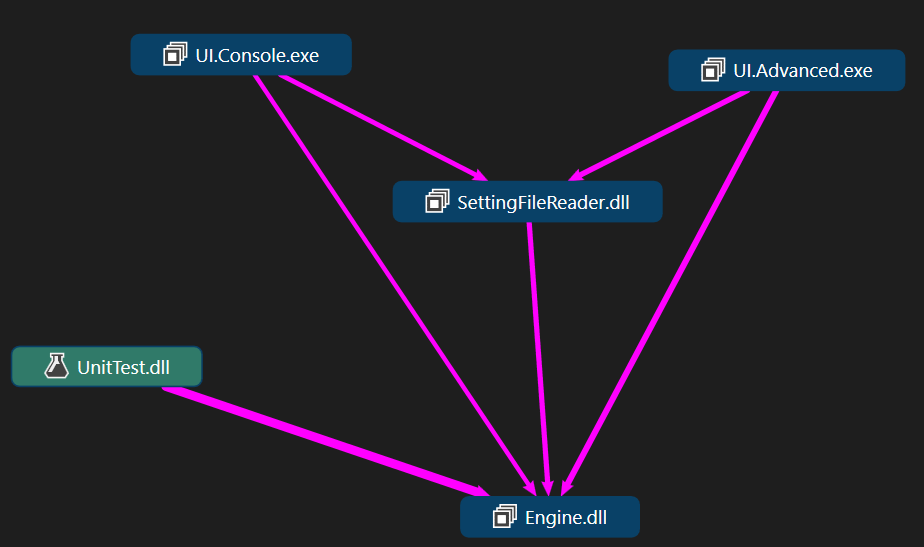
By applying the configurations in these input files, the application can determine what would happen to the Turtle per each movement set. Following image shows the console application result.



There are two UI for the application; the first one is created based on the assignment’s instruction as a Console application, the other one is a developed for fun as a windows application which represents the board and movements as a GUI.



**Modules:**

****

|  |  |
| --- | --- |
| **Module Name** | **Description** |
| **Engine.dll** | This assembly contains the classes and interfaces of the game which implement the business of the application. There are no dependency on any other resources rather than the .Net Framework here. |
| **SettingFileReader.dll** | This assembly contains some classes which facilitate the reading of the input file and represent the contents of the file as meaning full objects. If there were any format error in the files, the related exception with sophisticated data is thrown here. |
| **UI.Console.exe** | This is the Console based version of the applications UI. |
| **UI.Advanced.exe** | This is a desktop application which shows the board and movements as some graphical elements. |
| **UnitTest.dll** | There are unit tests created by MsTest for the Engine. 94% of the Engine’s codebase is covered by these tests. |

**Object-Oriented Design and Algorithms:**

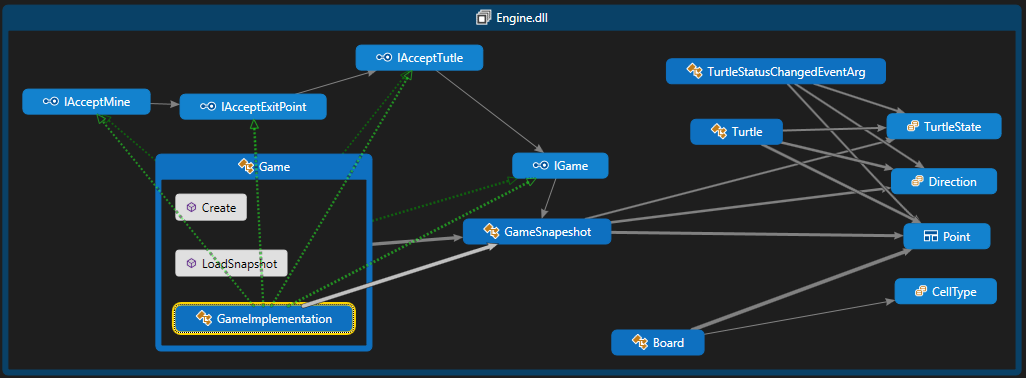
1. **Using Fluent Interface coding Style**

To facilitate the usage of the game engine exists in the “Engine” module there are a Factory class and some interfaces to apply fluent code style which are responsible for preventing the creation of the game with least potential error because of unordered setting up the game object. The Program class in the UI modules use these APIs to create a game.

1. **Special Design decisions**

Design patterns are used for implementing the engine:

* **Observer**: This pattern is used to notify the turtle changes to outside of the engine.
* **Momentum:** Thisdesign pattern is used in “**TakeSnapshot”**toenable the clients (those components using the engine) to save the internal state of the game as a serializable object; There is a “**Loadsnapshot** “ method which enables the Game class to reconstruct the game from a snapshot.



1. **Algorithm for finding Mines Position**

Laid mine in the game board are stored in a Hash set of “Point” objects. “Point” is a struct which overrides compression operators and “GetHashCode” Method.

This approach, the time complexity of checking coordination to see if it contains a mine would be O (1) although the space complexity will be O(n) while n is the number of mines.

The checking for meeting the exit point is just a simple compression between the board’s exit point and the turtle's