Department of Computer Science

08101 Programming I

Week 8 Laboratory 2013/2014

C# and Games

In this lab we are going to consider the remaining issues that we need to address to create a complete Hyperspace Cheese Battle game. Please make sure you have completed the previous lab before attempting this one.

Game Program Structure

At the end of last week you should have a program structure that looks a bit like this:

```
class Program
{
   // 2D array that holds the board
    static int[,] board = new int[,] ;

   // 1D array that holds the player information
    static Player[] players = new Player [4];

    // makes a move for the player given in playerNo
    private static void PlayerTurn(int playerNo)
    {
    }

    // reads in the player information for a new game
    static void ResetGame()
    {
    }

    // returns the value of the next dice throw
    static int DiceThrow()
    {
    }

    // returns true if there is a rocket in the specified square
    static bool RocketInSquare(int X, int Y)
    {

    static void Main()
    {
        ResetGame();
    }
}
```

This is not complete, it leaves out the Player structure, the content of the methods and how the board is set up. However it is a good pattern for the program. Each game behaviour is being provided by a method, and the entire program holds the methods and the data that they work on. The Main method will call the game methods in the right order to make the game work. In the code above it just resets the game.

Note that this is just one way of structuring the program. If your design is different this does not mean that it is wrong.

Displaying the Game Status

After each player has moved the game should display the status of all of the players. You will need to write a method that displays this information:

```
static void ShowStatus ()

    // TODO: Show the position of each player on the board
}
```

This could do something like this:

```
Hyperspace Cheese Battle Status Report
======================================

There are 3 players in the game
Fred is on square (1,1)
Jim is on square (3,5)
Ethel is on square (6,2)
```

The best way to do this would be to create a loop that works through the collection of players and prints out each one in turn. This loop will be very similar to the one that you created in the method that reads in the players when the game starts.

1. Add the `ShowStatus` method to your game as above.
2. Create a test in `Main` that you can use to test the method (you might find it useful to run this after you have called `ResetGame` to enter the number of players and their names.

Moving the Players

For each turn in the game you will need to work through each player and move them. In the last lab you created a `PlayerTurn` method that will play a turn for a particular player. Now you are going to use this method to move each player in turn.

The basis of this method will be a loop that goes through each player in turn and moves them. The move will update their position on the board. The key element of this method will be the following code:

```
int i;
for (i = 0; i < noOfPlayers; i++)
{
    PlayerTurn(i);
}
```

The `PlayerTurn` method is supplied with the subscript of the element in the Players array that is being moved.

1. Add a `MakeMoves` method to your game that moves each player as above.

If you notice at this point that you are spending a lot of your time writing loops that work through all the elements in the players array then you are doing the right thing.

Detecting Game End

At the moment the game does not detect when the game ends. You will need to add some code to the `PlayerTurn` method that checks when a player has reached square 64. If a player reaches this square the game will finish at this point. The best type of variable to use to store the game end state is a Boolean variable. These can be either true or false:

```
static bool gameOver = false;
```

When a player reaches square (7,7) this variable can be set to true, which will cause the game loop to end.

1. Add a `gameOver` variable to your class.
2. Add the code to `PlayerTurn` which will test the player position and set the variable to indicate that the game is over when a player reaches the end. `PlayerTurn` should also print a message when someone wins.
3. Add code to `MakeMoves` so that when the game ends it stops making any more moves (you might want to use the break keyword to achieve this)

Dice Management

The first version of the game we created had a dice that just produced the value 1 each time. Now we are going to look at improving this to allow us to easily test the program, and also get the proper gameplay that we want.

Creating a Pre-Set Array of Dice values

C# allows us to create an array that holds a sequence of pre-set values. We can store these in the program along with the Player details and the board design.

```
static int[] diceValues = new int[] { 2,2,3,3 };
```

This array contains 4 elements which are the moves at the start of the sample game. We are going to modify the `DiceThrow` method so that each time the dice is thrown it gives us the next element in the array. To do this the program will need a variable to keep track of the position that has been reached in the array:

```
static int diceValuePos = 0;
```

This value will start at 0 and then each time the dice is thrown it will be moved along to the next position in the array.

```
static int DiceThrow()
{
    int spots = diceValues[diceValuePos];

    diceValuePos = diceValuePos + 1;

    if (diceValuePos == diceValues.Length) diceValuePos = 0;

    return spots;
}
```

This version of `DiceThrow` uses the `Length` property of the array so that when the method runs out of numbers it will go back to the start of the array and repeat.

1. Update the DiceThrow method in your program so that you can easily create any sequence of dice values when the program runs.

Creating a Random dice

For the proper game you will need to create a dice which is random. To get a random number in a C# program we can use the `Random` class that is provided in the `System` library.

This implements a random number generator that gives a very good approximation to a real dice. We can create a variable of type `Random` for our game.

```
static Random diceRandom = new Random();
```

This should be created along with all the other game variables.

If we want to get a random number from this variable there are a number of different methods we can call to get the number. The most useful form for us is one that lets us specify the lowest random number and an *exclusive* upper bound. This means that values with the upper bound are not supplied; instead we get values up to that range. In other words we can use something like this:

```
int spots = diceRandom.Next(1,7);
```

This will set the value of spots to a random number in the range 1-6.

1. Change the name of the `DiceThrow` method to Preset`DiceThrow`.
2. Add a random dice method to provide random dice throws.
3. Now you can quickly switch between the two dice methods just by changing their names. You might want to think about using a flag (perhaps called `TestMode`) which is used to select which dice method is used when the program runs.

Cheese Power Behaviour

The game has a behaviour which is triggered when a player arrives on one of the cheese power squares on the board. When a player arrives on a square they have the following options:

- Absorb the “Cheese Power” from the square and create a “Cheese Deathray” which they can fire at any another player. This causes the engines of that rocket to explode, sending it back to the bottom row of the board. The player being sent to the bottom can choose any unoccupied square on the bottom row.
- Absorb the “Cheese Power” and use it to fire up their engines for another throw of the dice.

Your game will need to implement this gameplay.

Detecting a Cheese Power Square

Your game will need a way of detecting when a player has landed on a cheese power square. There are several ways you could do this:

- You could test the x and y positions of the player looking for the values x=4,y=1 or x=0,y=3 or x=6,y=4 or x=3,y=5. If the rocket is on any of those squares you know it is a cheese square
- You could add an extra set of values to the board array which mark some squares as cheese squares and then detect when the rocket lands on these
- You could make a new array, perhaps called `cheeseBoard`, which is a boolean and holds the value true of there is cheese on that square.

All of these methods will work and each has its advantages and disadvantages. Use the one that you think works best. You can then create a new method that will check for cheese power: The method could have the following signature.

```
static bool CheesePowerSquare(int x, int y )
{
    //TODO: write a method that checks through the
    //cheese square positions and returns true if the square has cheese power
}
```

When you have computed the square that a player is going to land on, you can use this method to determine whether or not they have landed on a cheese power square. If they have landed on a cheese square the game must ask them whether they want to explode the engines another player or throw the dice again.

1. Add the cheese square positions and a `CheesePowerSquare` method to your program.

Making the move

When your game detects that a player has landed on a cheese square it will have to have a conversation with the player about what to do. Then your game will have to update the player (and perhaps ask the player attacked where on the bottom line they want their rocket to be placed). This could all go in the `PlayerTurn` method.

Creating a Game Loop

You now have all the elements that are required to create a complete game. You now need to write code that calls the methods to run the game.

```
static void Main(string[] args)
{
    ResetGame();

    while (!gameOver)
    {
        MakeMoves();
        Console.Write("Press return for next turns");
        Console.ReadLine();
    }
}
```

This `Main` method sets up the game and then repeatedly goes round a loop making moves, pausing before each move for the user to press the return key.

Completing the Game

At the end of this practical you should have methods that read set up a game and then allow each player to have a turn and moves their rocket. If you play the game to completion the program should display a winner.

Note that the code above is simply a starter template. If your design for the game is different from the one above that is absolutely fine.

Rob Miles November 2013