CMP2020M Artificial Intelligence

Games Workshop 1: Introduction to Path Finding

Objectives

1. Familiarisation with the Pathfinder project
2. Create a simple AI "bot" which moves towards the player
3. Improve the bot so that it can move around simple obstructions

Duration: 1 week

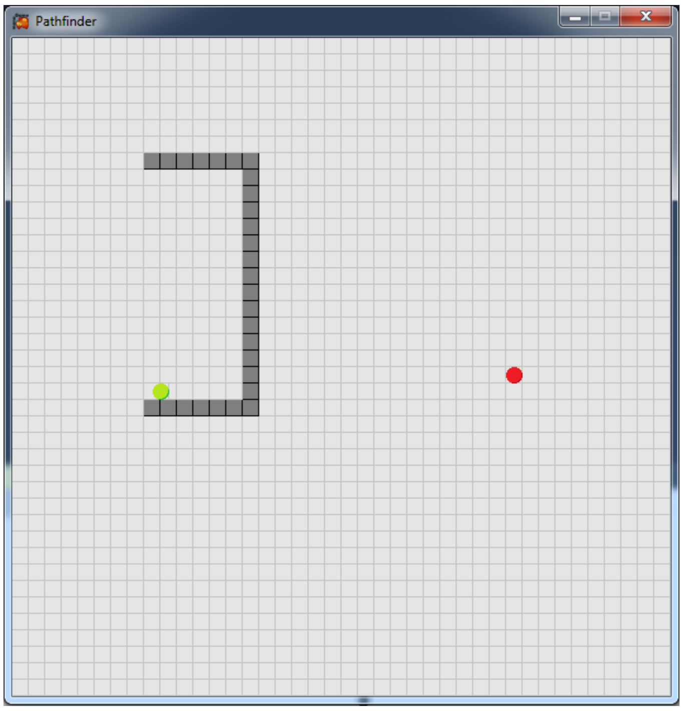
Tools / Libraries

1. Monogame v3.6
2. Microsoft Visual Studio 2015 (including Community Edition)

This workshop is not directly assessed, and will not count to your final mark for this module. ***However, you may asked questions about it on your examination***. You should therefore makes sure that you complete it and discuss any problems with a workshop demonstrator.

Introduction

Download the Pathfinder project from blackboard, and open it using Visual Studio. Build and run the project. You should see something like this:



The project creates a 40x40 2D grid-based map, with 2 objects. The red circle represents the player, and can be moved using the cursor keys. The green objects is an AI "bot". Currently the bot moves around randomly. The workshop tasks you will undertake in the next few weeks will involve implementing increasingly sophisticated algorithms which will help the bot to catch the player.

Project Structure

Quit the Pathfinder programme, and examine the project code files. You should locate the following files in the "Solution Explorer" pane, in the Visual C#/Studio IDE. There are other files also included, but these are the ones you should pay particular attention to:

Game1.cs
Level.cs
Player.cs
Coord2.cs
AIBotBase.cs
AIBotRandom.cs : These are source code files, each defines a game object class

Also in Solution Explorer: under the "Content " subdirectory you should also see the following:

0.txt
1.txt
2.txt
3.txt
4.txt: These are map layouts, coded as ascii text files

ai.xnb
target. xnb
tile1. xnb
tile2. xnb: These are the graphics used to draw the game

Main Features of the Code

Game1.cs defines the Game1 class: an object of this type is created automatically and represents the whole game. The *constructor* function instantiates the main objects in the game – a level, player, and bot. Note that:

- On line 57 the bot class is "AiBotRandom". Later, you will create new classes of AI bots, and instantiate them here.

- Line 55 loads the map from a txt file definition. The text file is specified as "2.txt".

TASK: Try changing this to "0.txt" and running the programme. Then try changing it to "1.txt", up to "4.txt".These levels are defined by the corresponding txt files under "Content". Finally, change the map back to "0.txt".

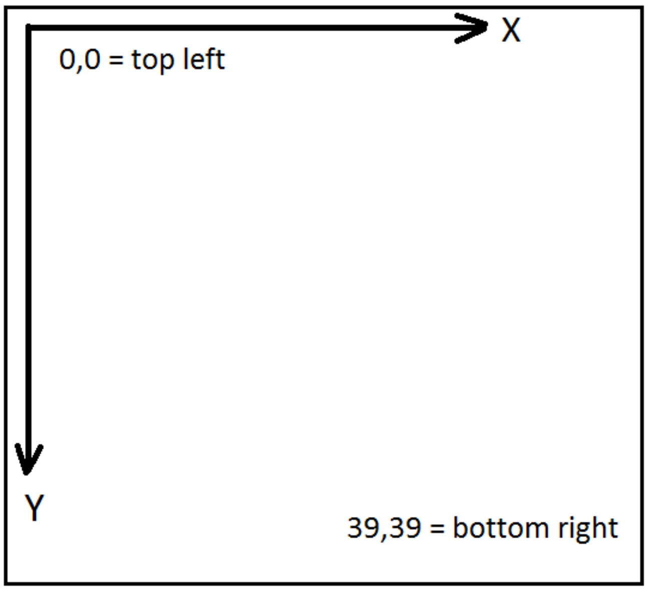
Also note that the main game loop is defined by the *Update*() function (starting on line 81). This processes the keyboard input, and updates the individual objects. You will never need to change this function!

The *Draw*() function draws the game map, and player and bot objects. You will not need to change anything in this function for the moment.

Level.cs
This contains the code for the Level class, which loads the txt file map and turns it into a 2D grid representing the level. You will never need to change anything in this class.

Player.cs
This contains the code for the player class, which stores and updates the player's position on the grid (the *gridPosition* member), and controls the smooth transition between grid squares. You will never need to change anything in this class.

Coord2.cs
This contains the code for the Coord2 struct, which is used by other classes to represent the positions of objects on the grid map. For example, the *gridPosition* member of the Player class is an object of type Coord2. It essentially contains a pair of integers, named X and Y respectively. Note that grid positions in the game map are represented like this:



AiBotBase.cs
This class is a ***base class*** for AI bots. As a base class, you cannot create an object of this type directly - it is a template for basic bot functions and attributes. For example, it contains a grid location member variable (*gridPosition*) to store the position of the bot, and manages the smooth transitions from one location to another.

The key part of this class is the function *ChooseNextGridLocation*(), which selects the next grid location for the bot to move to. This function is declared in AiBotBase, but is not implemented here (it is abstract): the idea is that to implement a new bot type, with a specific behaviour, we can create a new class which inherits from AiBotBase. The derived class then implements the *ChooseNextGridLocation* function to define the specific behaviour we want for that bot type. This simplifies the process of defining a new bot behaviours. For example, look at the next class...

AiBotRandom
This class inherits from AiBotBase (see the line `class AiBotRandom: AiBotBase`). The random walk behaviour is defined within the override function *ChooseNextGridLocation*(): this function merely generates a random number, and uses that to decided whether the next move is up, down, left, or right.
TASK: make sure you understand how this function works.

Also note that because the AiBotBase constructor takes 2 parameters, classes which inherit from it should also take 2 parameters for their constructors. For example, AiBotRandom has the following constructor:

```
public AiBotRandom(int x, int y): base(x,y)
{
        ...
}
```

You will need to use the same format for constructors in your own AiBot classes.

Main Tasks

The main tasks for this workshop are as follows:

TASK A: Create a new AiBotClass that implements a bot which moves towards the player.
To do this, first create a new .cs source file and add it to the project (right click on Pathfinder in solution explorer, and select Add>>New Item>>Class) call the new source file something like AiBotSimple. You should get a new source file which looks something like this:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Pathfinder
{
    class AiBotSimple
    {
    }
}
```

Modify this class definition so that it inherits from AiBotBase, adds a constructor function similar to the one for AiBotRandom, and implements an empty version of *ChooseNextGridLocation*(). You will also need to add some more namespaces so that the class definition looks like this...

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Diagnostics;
using System.Text;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Audio;
using System.IO;

namespace Pathfinder
{
    class AiBotSimple: AiBotBase
    {
      public AiBotSimple(int x, int y): base(x,y)
      {
      }

      protected override void ChooseNextGridLocation(Level level,
Player plr)
      {
                  //YOUR CODE WILL GO HERE
      }
    }
}
```

The behaviour code for your new AI bot class goes inside the *ChooseNextGridLocation*() function. **The first task is to add some code that chooses to move either left/right/up/down such that the bot moves closer to the player...**

Hints:

1. The coordinates of your new bot are given by:
   ```
   GridPosition.X
   GridPosition.Y
   ```

2. You will need to examine the current player position, given by:
   ```
   plr.GridPosition.X
   plr.GridPosition.Y
   ```

3. The function needs to choose a square adjacent to the current bot position, either left, right, up or down. You therefore need to choose whether the bot is going to move in the X or Y direction. **Remember, the bot can only move one square in one direction at a time!** Create a new Coord2 variable and initialise it to the current bot grid position:

   ```
   Coord2 newPos;
   ```

```
newPos = GridPosition;
```

4. Either add or subtract 1 to the X or Y component of newPos, depending on which way you want to move. For example:
   ```
   newPos.X += 1;
   ```

5. Finally, call *SetNextGridPosition(newPos,level)* to set this as the next grid position to move to. Look at the AiRandom class to see how to do this.

6. ***Important: You can detect if the new position is blocked in one of two ways:***

   a. *SetNewGridPosition*() will return false if the new grid location is blocked, or...
   b. You can call *level.ValidPosition(newPos)* to test the new grid position (it will return false if blocked).

To tell the programme to use your new class instead of the AiRandom Class, change line 57 in Game1.cs from:

```
bot = new AiBotRandom(10, 20);
```

To

```
bot = new AiBotSimple(10, 20);
```

Run the programme to check that your code works. The bot should move towards the player. Try moving the player around, to check that all is working correctly.

TASK B: Create a better Bot Class

The implementation for task A should be very simple, and unable to deal with obstructions. Change line 55 of Game1.cs so that the map file is specified as "1.txt". Run your programme. You should find that the wall prevents the bot from reaching the player.

The idea of this task is to make a better bot, which **can** move around the wall. Start by creating a new bot class "AiBotSimple2", as before, and make sure that you change line 52 in Game1.cs to create an object of this new class.

There are a number of ways to approach this task. One way is as follows:

- Set an initial "current direction" of movement towards the player in either the X or Y direction..

- Instead of picking a new direction on each iteration of *ChooseNextGridLocation*(), continue to move one square at a time in this current direction (make a new Coord2 variable, set it to the current position, add 1 in the current direction, and call *SetNewGridPosition*).

- When the bot gets stuck (ie it hits a wall or the map edge), then pick a new direction at 90 degrees to the current direction (which direction is best depends on the player position). Remember, you can detect when a bot gets stuck in two ways:

    - *SetNewGridPosition*() will return false if the new grid location is blocked, or...
    - You can call *level.ValidPosition(newPos)* to test the new grid position (it will return false if blocked).

- Follow the new direction until the bot gets blocked again.

- You will also want to turn when the bot is in line with the player. For example, if the bot is moving in the x direction, and moves to a point where it has the same x value as the player, then it will need to start moving in the y direction instead. You may also want to change direction if the bot is moving away from the player.

Using this approach you should be able to create a bot which is able to move around the wall in "1.txt".

OPTIONAL TASK C

If you have time, try out the improved bot with the other maps (2,3,4). Try to code a bot which can move around these more complicated obstructions. After some time you should be able to convince yourself that these simple path finding strategies are not sophisticated enough to cope with complex maps. Negotiating a map like the one in "4.txt" requires an approach that considers the layout of the whole map, not just the area immediately around the bot.