Ms. Pacman AI for .NET

Hi, and welcome to this briefly written manual. I'll try my best to give an explanation on how to get all the essentials running, and luckily with Visual Studio you should be able to work your way through most of the rest of the programs with some help from the example AI's.

If you are completely stuck or just have a question, feel free to contact me at flensbak@itu.dk

Overview

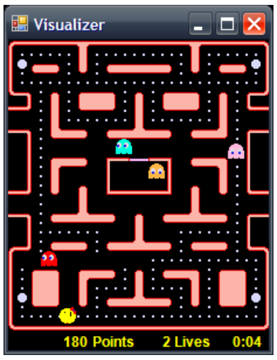
The main purpose of all this code is of course to have an AI controller play Ms. Pacman with some measure of success. To do this there is basic framework for describing controllers and some different features they can use as well as complete freedom to implement your own features – both in the controller or directly in the surrounding code. There is a program for using your controller to play the real Ms. Pacman but there is also a program that will allow you to quickly evaluate your controller against a simulated game. Both will be explained in the next sections.

Please open the solution (Pacman.sln) in Visual Studio 2008 before proceeding. There will be 5 projects in this solution, and the following sections will be about one of these projects each.

Simulator

The simulator is a beast. This project is the basis of all the rest, as it not only contains the simulated Ms. Pacman game, but also the basic description of controllers, information about the map layout and structure, and all the features for computing shortest path and so on. This is why most other projects reference the Simulator project.

The simulator project also contains a way of visualizing the current game state. This can be both a game state in a simulated game, and a game state from the real Ms. Pacman game (covered later). It is also possible to just play the simulated game yourself, to let a controller play it in real time, or even let a captured stream play back.



To play the simulator yourself, view the files in the Simulator project, and right-click on Visualizer.cs to view the code. Under the constructor there is an area marked "`// IMPORTANT SETTINGS AREA //`". This is where you set up how the Visualizer is run. Make sure that all `tryLoadController` and `tryLoadStream` methods are commented out. Then right-click the Simulator project, select Set as StartUp-project and start debugging (F5). Now you should be able to play the simulated Ms. Pacman game by using the arrow keys. You will notice there are no fruits, and only the first level is implemented, as well as other things that could be improved to match the real game more closely. Improving on this would greatly increase the usefulness of the Simulator so feel free to do so.

Now try and comment in the line `tryLoadController("SmartDijkstraPac");`. SmartDijkstraPac can be replaced with any controller type defined under the PacmanAI project, but more on that later. If you start debugging now, the simulator should play by using the SmartDijkstraPac controller. Watch out though – if you change anything in SmartDijkstraPac.cs you will not get that version running as the DLL is loaded directly from the Simulator projects Debug directory. You can either copy over the new DLL from the Debug directory from PacmanAI or simply reroute the path in the `tryLoadController` method.

Now try and comment out the tryLoadController call again, and then comment in `tryLoadStream("best22030");`. This will load the best22030.dat file located in the Debug directory under the Simulator project. If you run the program now it should repeatedly show a recorded game where the score is 22030 points. This is a recorded game from the ConsoleSim project but more on that later.

This is all the things you can directly use the Simulator project for. However, it is likely that you will use it for other purposes when developing your own controller. Either bugfixing a lot of the central code that is placed in this project or hopefully expanding it with some useful features.

PacmanAI

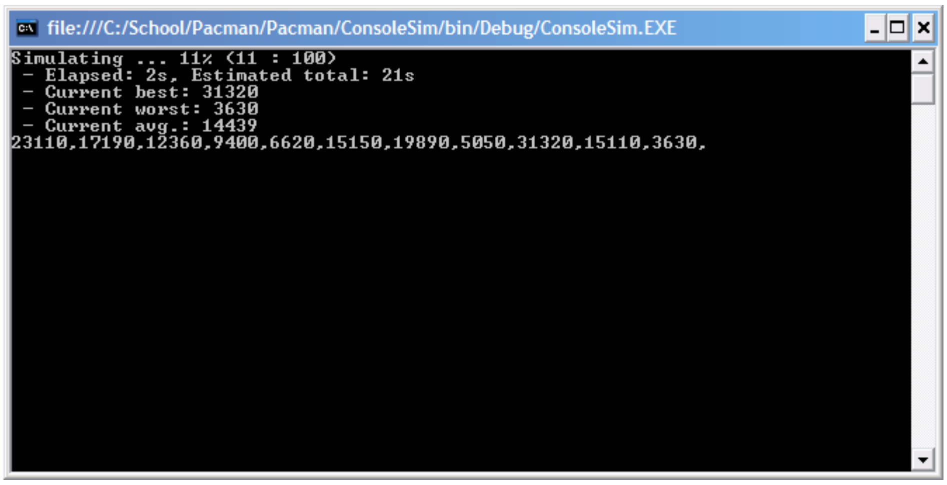
Take a look at the PacmanAI project, which is where you define your custom controllers. If you take a look at TestPac.cs you will see the most basic AI you can define. The Think method will be evaluated and it simply says that Ms. Pacman has to go left no matter what. I suggest you go through the included controllers to get a feel for how they work.

If you want to make your own AI you can just copy TestPac and change the name – simply change everywhere it says TestPac in the file to something else. You can also take a look at the BasePacman.cs class defined under the Simulator, which all controllers must inherit from. It defines some virtual methods you can override as well, but look at the code that calls them to get a feel for how they work.

ConsoleSim

Now go to the ConsoleSim project. By right-clicking and selecting Set as StartUp Project you will be able to run the simulator without visualizing it. This allows you to quickly run through hundreds or thousands of simulated games without the overhead of the visualizer.

Open up program.cs and find the DEFINE CONTROLLER section. Here you can directly reference any controller in the PacmanAI project. Comment one in and start debugging to run through any number of games (defined in the gamesToPlay variable at the top of the file). The Console Window will output the average, best, and worst scores as well as some other statistics. It will also save a data (.dat) file for the worst and best game played in the Debug directory of the ConsoleSim project. You can copy this file into the Debug directory of the Simulator project, reference it from the Visualizer.cs file and run the Simulator project to get a play back the data file.

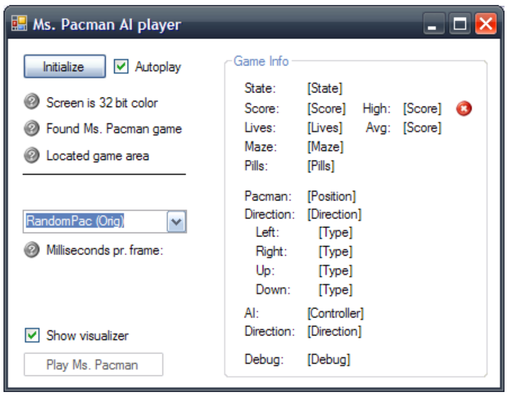

file:///C:/School/Pacman/Pacman/ConsoleSim/bin/Debug/ConsoleSim.EXE

```
Simulating ... 11% (11 : 100)
 - Elapsed: 2s, Estimated total: 21s
 - Current best: 31320
 - Current worst: 3630
 - Current avg.: 14439
23110,17190,12360,9400,6620,15150,19890,5050,31320,15110,3630,
```

The purpose of the ConsoleSim application is of course to enable learning strategies by relatively quickly evaluate a large number of games. The learning depends greatly on how good the simulated game world is, and some improvements could be used there. Also notice that the scores in the simulated game are quite inflated as it only implements something approximating the first two levels in the real game. However, the scores very directly correlate with the scores achieved in the real game, and will also quickly give you an idea of how stable the performance of your controller is. However, evaluating small differences between controllers is probably difficult because of the differences between the simulated game and the real game, just as they are difficult to detect without playing through a large amount of real games.

MsPacmanController

Now we're getting to the interesting part. The MsPacmanController has gotten a nice interface that should be rather easy to use. All you need to do is to right-click on the project, select Set as StartUp Project as usual, and start debugging. This will present you with something like this:

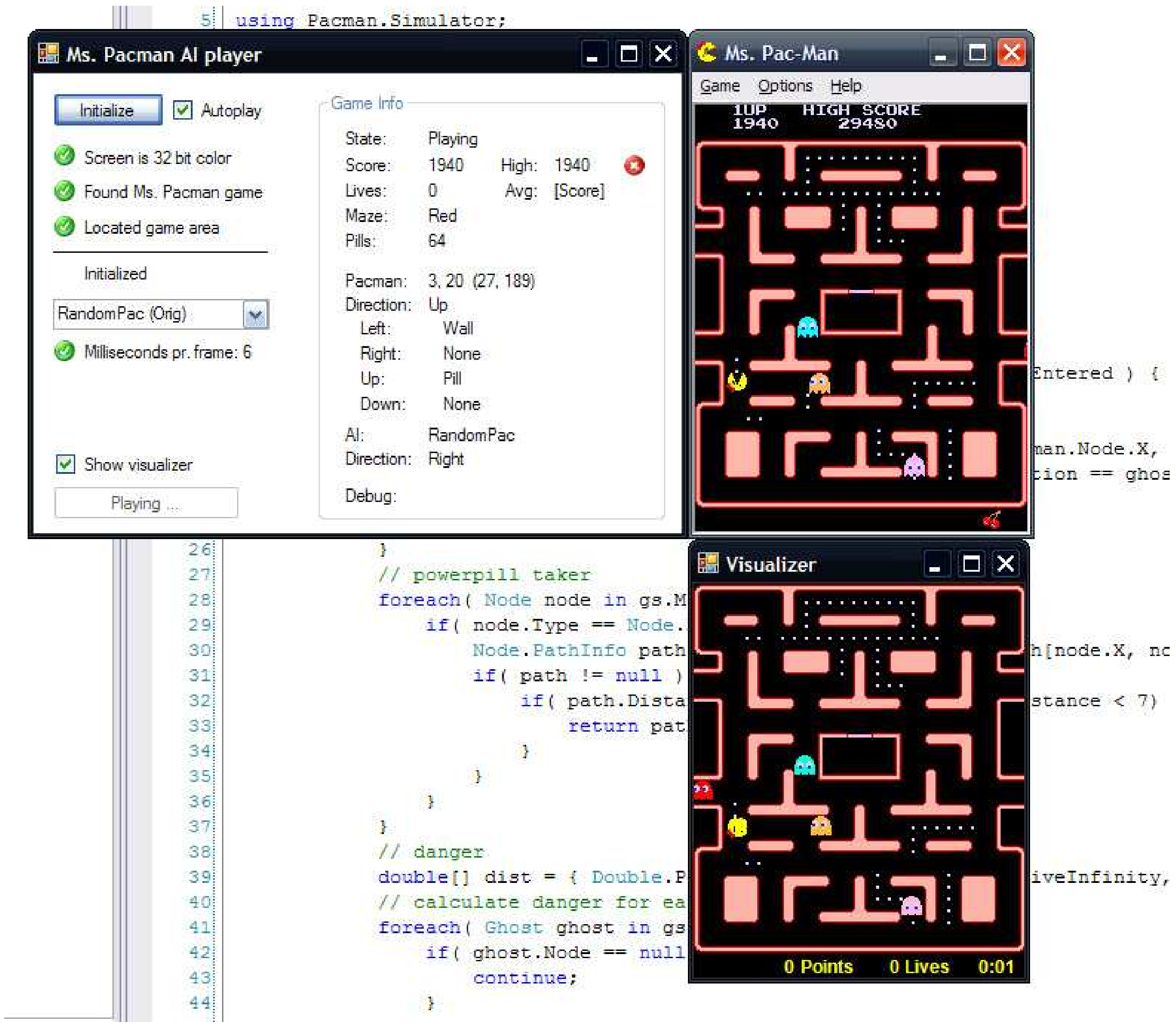


You will need to have Microsoft Ms. Pacman from Return of the Arcade Anniversary Edition running to be able to use your controller. If you have this, and are also running with 32Bit color depth, you should get all green marks in the top part when you click Initialize. If you don't have Microsoft Ms. Pacman and can't get access to it, you will have to change the application to be able to run with something like MAME. As far as I can tell it is exactly the same ROM as the one used by the Microsoft Ms. Pacman game, but you will have to set it up to detect and use it instead then.

Most of the settings should be self-explanatory but here are most of them explained:

- Initialize: This checks if everything is okay to start playing the game. If anything fails you will not be able to test your controller. Be aware that the game window needs focus – so you click or change anything you will have to either re-initialize or click Play Ms. Pacman again.
- Autoplay: This means that the game will start playing with your selected controller immediately after Initialize succeeds.
- Screen is 32bit color: The program needs this, as all color offsets are based on 32bit color values.
- Found Ms. Pacman game: The Microsoft Ms. Pacman game needs to be started and preferably just ready to start playing.
- Located game area: The program will try to find the exact offsets of the game area, subtracting window borders and so. If this fails you will have to look at the code to figure out what is wrong (this is hard, so lets hope it works ok).
- Dropdown box: This is where you select the controller to play. The ones marked with (Orig) are old example controllers that you can try. The rest are the controllers defined in the PacmanAI project, and you can simply change the code in them and start debugging again to get the changed controller running.
- Milliseconds pr. frame: This number should be below 10-15 when playing the game. Otherwise evaluating the controller and the reading the game state takes too long, the computer will lag, and you will risk missing turns and getting caught by the ghosts. This happens either on slow machines or if you put too much processing into your controller.
- Show visualizer: This will show the visualizer from the Simulator project. You don't usually need this when testing your controller, but it is meant as a way to see how the controller sees the game state. If the real game and the visualizer don't agree on the game state, there is a bug in the MsPacmanController code. Debugging this will likely be difficult.
- Play Ms. Pacman: If the game has been initialized probably this will start playing the game with the selected controller.
- Game info: Pretty self-explanatory information about the controller and current game state. Sometimes a wrong highscore will be recorded (if you start playing before the game has finished loading for example), and you can use the red x to clear this.

If everything works out, it should look something like this:



```
State:      Playing
Score:      1940        High:   1940
Lives:      0           Avg:    [Score]
Maze:       Red
Pills:      64

Pacman:     3, 20  (27, 189)
Direction:  Up
  Left:       Wall
  Right:      None
  Up:         Pill
  Down:       None
AI:         RandomPac
Direction:  Right

Debug:
```

```
using Pacman.Simulator;
                                                         Entered ) {
                                                         man.Node.X,
                                                         tion == ghos
                }
                // powerpill taker
                foreach( Node node in gs.M
                    if( node.Type == Node.
                        Node.PathInfo path                h[node.X, nc
                        if( path != null )
                            if( path.Dista                stance < 7)
                                return pat
                            }
                        }
                    }
                }
                // danger
                double[] dist = { Double.P                iveInfinity,
                // calculate danger for ea
                foreach( Ghost ghost in gs
                    if( ghost.Node == null
                        continue;
                    }
```

FeatureExtraction

This is a very small helper project used to get the offset values for Ms. Pacman and the ghosts for the MsPacmanController. You will not likely use this, unless you happen upon some error in MsPacmanController or wish to implement support for recognizing the fruits.