Program 2 Preview: Pac-Man Minimax

Dr. Demetrios Glinos
University of Central Florida

CAP4630 – Artificial Intelligence

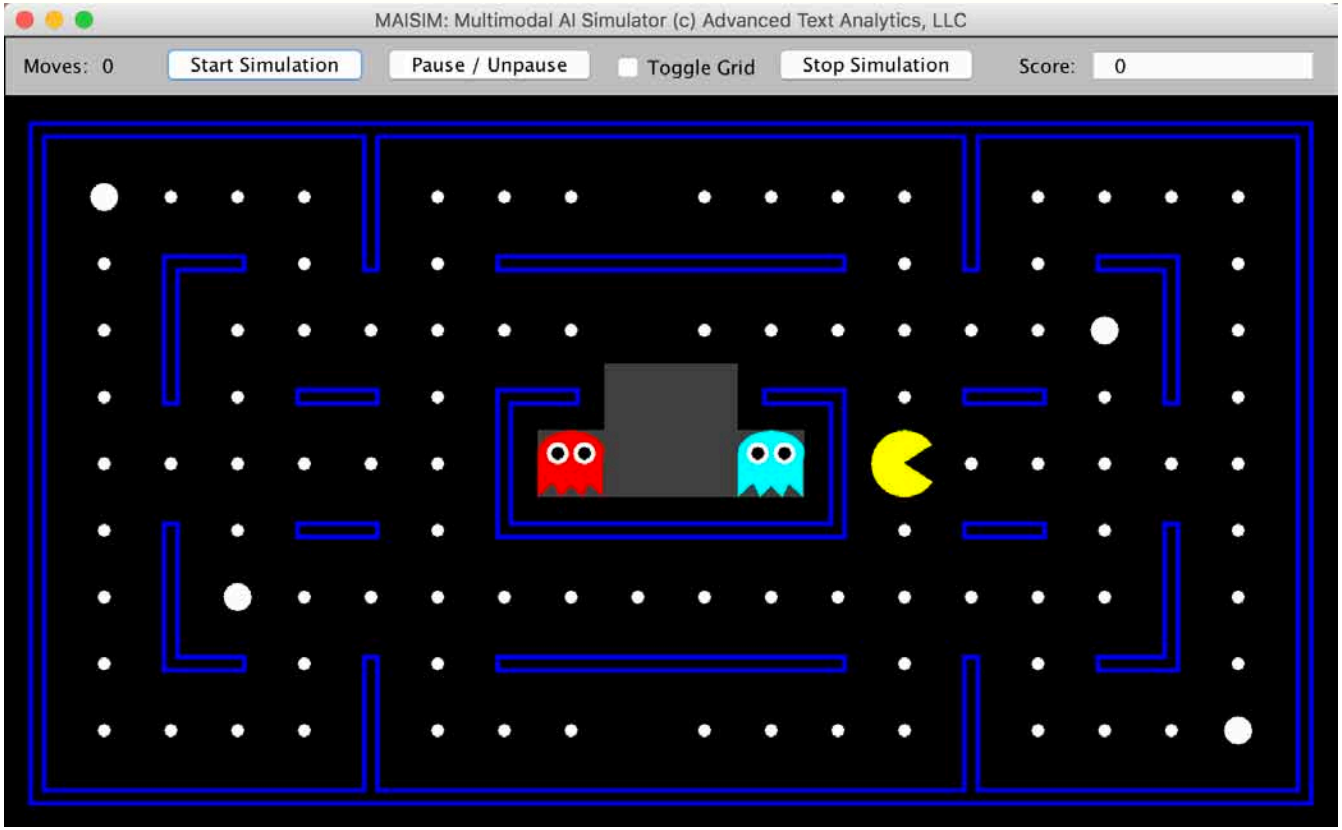
Outline

- Task in a Nutshell
- The Game Board
- Implementing Minimax
- Program Inputs
- Program Template Code
- Required Code Comments
- Sample Output and Demo

The Task in a Nutshell

- Control Pac-Man using the minimax algorithm in a maze that contains two ghosts, food dots, and a few power pellets
- Pac-Man's goal, as always, is to eat all the food dots before he is eaten by a ghost
- Minimax depth must be a program input
- The program must implement the simulation features of the simulation engine, which allow running complete games in the background without the graphics overhead, to accumulate success rate statistics
- Program to be submitted as an executable JAR file (without embedded PacSimLib.jar)

The Game Board: *game-small-new*



- Blinky – goes straight for Pac-Man; home is upper-right
- Inky – aims for spot on other side of Pac-Man at same distance and directly opposite from Blinky; home is lower-right

Implementing Minimax: Depth

- Minimax depth
  - expressed as an integer number of moves
  - we will test depth values of 1 and 2, and maybe 3

- One move
  - an action by Pac-Man
  - an action by Ghost #1 (Blinky)
  - an action by Ghost #2 (Inky)

- Complexity, assuming an average of 2 cells available for each player:
    - 8 board positions must be evaluated for depth 1
    - 64 board positions must be evaluated for depth 2
    - 256 board positions must be evaluated for depth 3

Implementing Minimax: Evaluation Function

- Basic minimax evaluates to end of game
  - all food eaten
  - or, Pac-Man eaten by ghosts
- Depth-limited minimax uses an evaluation function to value states beyond the depth limit
- You may choose any features you wish, for example
  - BFS distance to nearest food
  - BFS distance to nearest ghost
  - between two ghosts
  - in an alley
- You may choose as many or as few features as you wish, provided your program does not take forever to run

Program Inputs

- The program must take the following inputs (in this order) as command-line arguments:
  1. name of game board
  2. minimax depth
  3. number of simulation epochs
  4. granularity for training epoch statistics
  5. maximum number of moves allowed

  } *optional*

- Program may be launched either with or without simulation parameters

  java -jar PacSimMinimax.jar game-small-new 2

  java -jar PacSimMinimax.jar game-small-new 2 10 1 1000

- Please note that the executable JAR file must be in a folder that also contains:
  - a lib folder with PacSimLib.jar in it
  - the game-small-new game board

Program Template Code

- The template is provided with assignment
- Purpose is to aid you in embedding your minimax implementation
- Its essential sections are described on the following slides

```
// import statements

public class PacSimMinimax implements PacAction {

   // optional: class and instance variables

   public PacSimMinimax( int depth, String fname, int te, int gran, int max )
{

         // optional: initialize some variables

      PacSim sim = new PacSim( fname, te, gran, max );
      sim.init(this);
   }

   public static void main( String[] args ) {

      String fname = args[ 0 ;
      int depth = Integer.parseInt(args[ 1 ]);

      int te = 0;
      int gr = 0;
      int ml = 0;

      if( args.length == 5 ) {
         te = Integer.parseInt(args[ 2 ]);
         gr = Integer.parseInt(args[ 3 ]);
         ml = Integer.parseInt(args[ 4 ]);
      }

      new PacSimMinimax( depth, fname, te, gr, ml );

      System.out.println("\nAdversarial Search using Minimax by
<your_name_or_names>:");
      System.out.println("\n   Game board   : " + fname);

      System.out.println("   Search depth : " + depth + "\n");

      if( te > 0 ) {
            System.out.println("   Preliminary runs : " + te
            + "\n   Granularity      : " + gr
            + "\n   Max move limit   : " + ml
            + "\n\nPreliminary run results :\n");
      }
   }

   @Override
   public void init() {}

   @Override
   public PacFace action( Object state ) {

      PacCell[][] grid = (PacCell[][]) state;
      PacFace newFace = null;

         // your code goes here

      return newFace;
   }
}
```

Template Components (1)

- Class must implement PacAction interface

```
public class PacSimMinimax implements PacAction {
```

- Main method must read in all command arguments

```
public static void main( String[] args ) {
    String fname =args[ 0 ];
    int depth = Integer.parseInt(args[ 1 ]);
    int te = 0;
    int gr = 0;
    int ml = 0;
    if( args.length == 5 ) {
        te = Integer.parseInt(args[ 2 ]);
        gr = Integer.parseInt(args[ 3 ]);
        ml = Integer.parseInt(args[ 4 ]);
    }
```

Template Components (2)

- Main method must pass all 5 arguments to the class constructor

```
new PacSimMinimax( depth, fname, te, gr, ml );
```

- Main method must generate output header information

```
System.out.println("\nAdversarial Search using Minimax "
        "by <your name or names>:");
System.out.println("\n   Game board    : " + fname);
System.out.println("   Search depth : " + depth + "\n");
if( te > 0 ) {
      System.out.println("   Preliminary runs : " + te
      + "\n   Granularity      : " + gr
      + "\n   Max move limit   : " + ml
      + "\n\nPreliminary run results :\n");
}
```

Template Components (3)

- Constructor should input all 5 arguments (even zero values, if any) and pass all but the depth argument to the simulation engine

```
public PacSimMinimax( int depth, String fname, int te,
                      int   gran, int max )
{

    // Enter your own code here, if needed

   PacSim sim = new PacSim( fname, te, gran, max );
   sim.init(this);
}
```

- Note: Since depth argument is not passed to the simulation engine, the constructor must do something with it (e.g., save it in a class variable)

Template Components (4)

- Program must have an init() method (which can be stubbed, as in the example below):

```
@Override
public void init() {}
```

- Work of program is done in action() method

```
@Override
public PacFace action( Object state ) {

  PacCell[][] grid = (PacCell[][]) state;
  PacFace newFace = null;
   ...
   return newFace;
}
```

Required Code Comments

- Comment block in code must describe
  - how board positions are evaluated
  - including but not limited to the features used for evaluation and how they are weighted

Sample Output and Demo

```
Adversarial Search using Minimax by Dr. Demetrios Glinos:

   Game board   : game-small-new
   Search depth : 2

   Preliminary runs : 10
   Granularity      : 1
   Max move limit   : 1000

Preliminary run results :

    1 :   1 wins,   0 losses,  149 avg moves
    2 :   1 wins,   0 losses,  180 avg moves
    3 :   0 wins,   1 losses,  193 avg moves
    4 :   1 wins,   0 losses,  177 avg moves
    5 :   1 wins,   0 losses,  199 avg moves
    6 :   1 wins,   0 losses,  298 avg moves
    7 :   1 wins,   0 losses,  209 avg moves
    8 :   1 wins,   0 losses,  353 avg moves
    9 :   1 wins,   0 losses,  247 avg moves
   10 :   0 wins,   1 losses,  107 avg moves

Results for 10 games : 8 wins, 2 losses ( 80.00 % ), 211 avg moves

Game over: Pacman has eaten all the food
```

demo: minimaxsim