# Program 2

Submit Assignment

**Due** Feb 24 by 11:59pm **Points** 100 **Submitting** a file upload **File Types** jar
**Available** after Jan 29 at 12am

## Minimax Adversarial Search for Pac-Man

### Introduction

For this assignment you will use Java to implement a depth-limited minimax adversarial search algorithm as discussed in lecture to drive Pac-Man through a maze to collect food dots while avoiding two ghosts in the maze who are trying to eat Pac-Man. You will use the same simulation infrastructure in the form of a JAR file that you used for Program 1, so you will not need to develop any graphics or simulation code. You may also wish to consult the same API documentation for the simulator that was furnished to you for Program 1.

You will be provided with skeleton code to which you can add your implementation of depth-limited minimax.

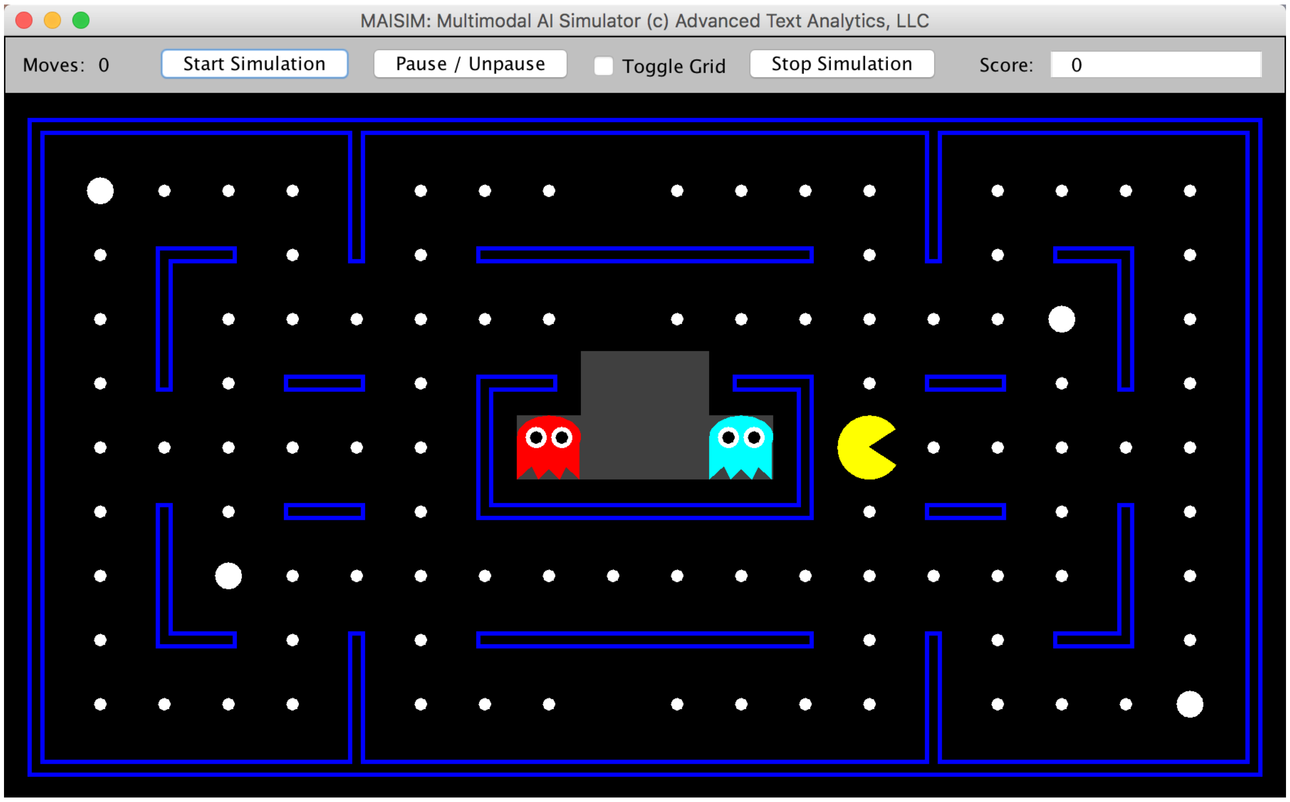
The skeleton code allows the program to accept either two or five command-line arguments, as described in a section below. In particular, the depth for the search must be a variable that is set by one of the input arguments and should not be hard coded in the program.

For searching beyond the depth limit, your program must implement an evaluation function to estimate state values.

Program output must conform to the sample output that is presented in this assignment description.

### The Game Board

You can download the game board for this assignment **here**. This is what it looks like:



The image above shows the starting position for the game. As you can see, the maze consists of many food dots, four power pellets, Pac-Man, and two ghosts. The dark gray cells are called "ghost house" cells and Pac-Man is forbidden from entering them.

The red ghost is named Blinky. The cyan ghost is named Inky. Basic ghost motion alternates between "scatter" mode (for 7 turns) and "chase" mode (for 20 turns). In scatter mode, each ghost heads for its home corner, upper right for Blinky and lower right for Inky. In chase mode, Blinky heads right for Pack-Man, while Inky heads for the spot that is located directly opposite from Blinky on a line from Blinky through Pac-Man. When Pac-Man eats a power pellet (large white dot), the ghosts enter into "frightened" mode and move randomly for 20 moves.

Pac-Man's goal in this game is to eat all of the food dots before he is eaten by one of the ghosts.

## Implementing Minimax

The basics of depth-limited minimax have been discussed in lecture. However, to understand how to implement it for Pac-Man, we must keep in mind two things.

First, we must be clear about what we mean by "depth" for the search, since this is a key feature of the program. A single move in a Pac-Man game with our two ghost involves a "move" by Pac-Man, a "move" by Blinky, and a move by Inky. As a result, if we assume that on average each of these players can move to two available cells, then your minimax algorithm must evaluate 8 board positions for a depth level of one, and 64 board positions for a depth level of two.

Second, because the depth of the search is limited, you must implement an evaluation function to compute values for states beyond the depth limit. You may choose any features you wish. For example, you may wish to consider the distance to the nearest food, or to the nearest ghost, or whether Pac-Man is in a long corridor or alley, and so on. You may use as many or as few features as you wish. However, your program must spend so much time evaluating positions that the game is unplayable, that is, the visible motion on the board is so stuttered that it does not proceed smoothly.

## Program Inputs

The program must accept the following five command-line arguments:

1. name of game board
2. minimax depth (complete moves)
3. number of simulation epochs
4. granularity for training epoch statistics
5. maximum number of moves allowed

Inputs 3, 4, and 5 are optional. They invoke the simulation features of the simulation engine, which allow running complete games in the background without the graphics overhead, to accumulate success rate statistics. When specified, the simulation will first run your program in the background for the specified number of simulation epochs and will record the number of wins and losses in groups specified by the granularity argument. For these background runs, if Pac-Man is still alive but has not eaten all the food pellets by the maximum move limit, the game is stopped and is recorded as a loss for Pac-Man.

The program template code described in the next section allows the program to be launched either with or without invoking the background simulation mode:

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

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

Note: for Windows users, the launch command should use a semicolon instead of a colon before the library jar file reference.

Please also note that the 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 following code is furnished as a template for inserting your own code for this assignment:

```
// 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;
   }
}
```

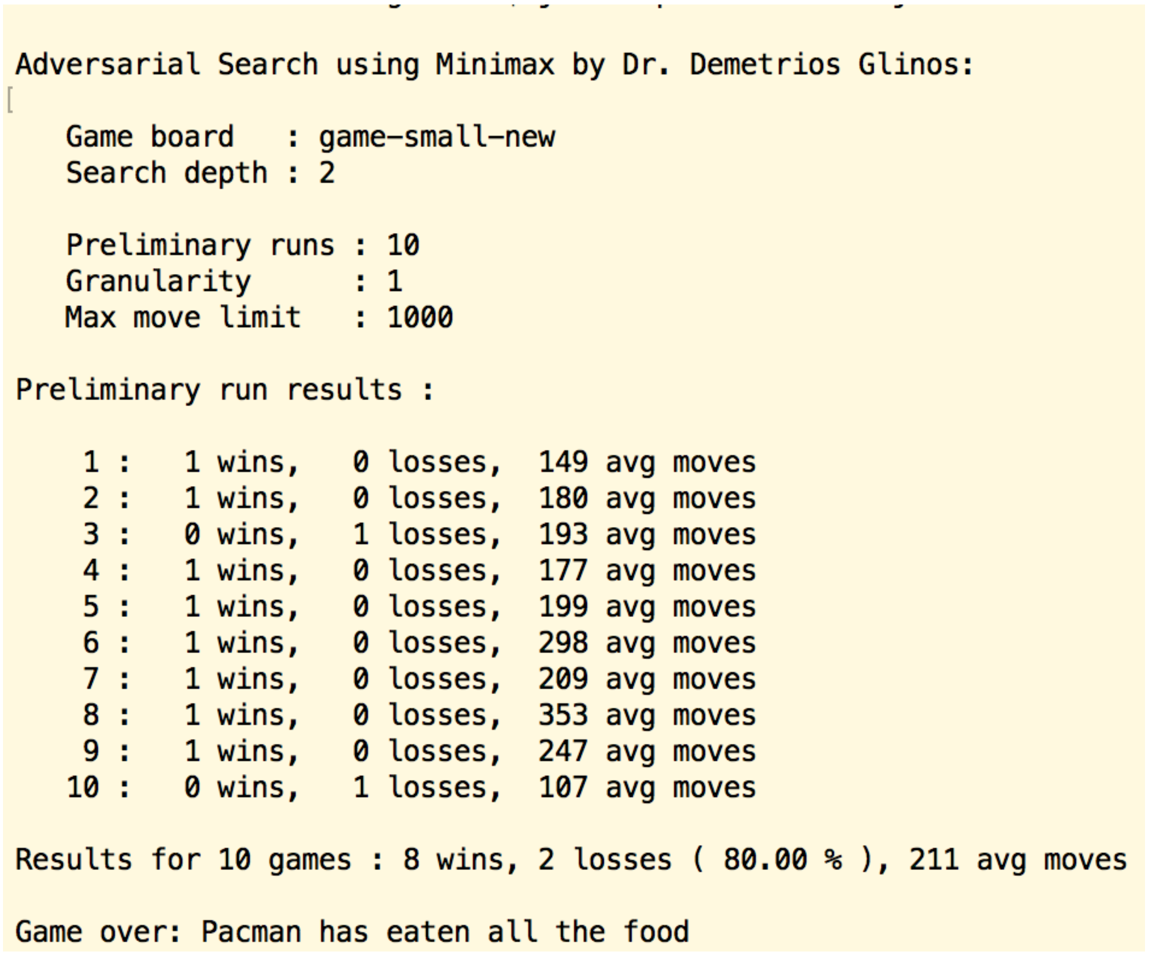
Please note that the constructor should make some use of the search depth argument since it receives it as input but does not pass it to the simulation engine. How you use it is, of course, up to you.

## Required Code Comments

The source code should contain a comment block that fully describes how board positions are evaluated, including but not limited to the features used for evaluation and how they are weighted.

## Sample Output

Except for including your name (or names, if teaming), the required output statements are already provided by the template code that is provided to you.  The following is a sample output of a run that includes background simulation runs:



```
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
```

**Note:**  When background simulations are not specified, the output will show only the title header, game board, and depth limit.  Also, if in the live run that follows the background simulations, a message about Pac-Man winning will be presented, as here. If Pac-Man is eaten by a ghost, no message will be presented.

## Development Teams

Teams of two students in the class are allowed, but are not required. You may develop the program by yourself if you like, with no reduction in requirements.

If you choose to work as a team, both team members must be students in the class, and:

(a) both team members must submit the same program on Webcourses for grading;

(b) both submitted programs must include the names of both students in a code comment at the top of the program.

If submitting as a team, both team members will receive the same grade. If only one member of the team submits, then the team member who does not submit will receive a 20-point deduction for the assignment.

## Program Testing

We will test your program by running your executable JAR from the command line against the test maze, so we suggest that you do the same to verify proper program operation and output prior to submitting. We will test depths 1 and 2, and possibly 3. We will also test both with and without background simulations.

You may wish to use the background simulations as a quick way to evaluate how well your program works. Any average over 40% wins should be considered successful.

## What to Submit

You should submit a single Java executable JAR file containing your PacMan minimax agent. The JAR file must contain both the source (.java) and the corresponding compiled (.class) files for all Java source files in your implementation. Of course, it is possible to implement the agent using only one Java source file.

All of your source files should include a header identifying UCF, this course and semester, and identifying the program author or authors, in the following format:

```
/*
 * University of Central Florida
 * CAP4630 - Spring 2019
 * Author(s):  <your name or names>
 */
```

If you are working as a team and only one team member mentions both authors' names in the source file headers, then the other team member will receive a 20-point deduction.

The program must be submitted on Webcourses by all authors. Email submissions will not be graded. The submission will be graded using the grading rubric below. Additional deductions will be applied as appropriate for late submissions, teaming, and header irregularities, as specified in the course syllabus.

**CAP4630-Prog2-Minimax**

| Criteria | Ratings | | Pts |
|---|---|---|---|
| The submission is an executable JAR file that contains source code that identifies the author or authors in the required format. | **20.0 pts Full Marks** | **0.0 pts No Marks** | 20.0 pts |
| The program runs without crashing, accepts the five (5) required command arguments, and reports them in the required format. | **20.0 pts Full Marks** | **0.0 pts No Marks** | 20.0 pts |
| A comment block in the program fully describes how board positions are evaluated, including but not limited to the features used for evaluation and how they are weighted, | **20.0 pts Full Marks** | **0.0 pts No Marks** | 20.0 pts |
| Pac-Man behaves reasonably for both depth level 1 and depth level 2 | **40.0 pts Full Marks** | **0.0 pts No Marks** | 40.0 pts |
| | | | Total Points: 100.0 |