Evolutionary Computing
COMP 5660-001/6660-001/6666-V01 – Auburn University
Fall 2020 – Assignment Series 2
GPac: A Genetic Programming & Coevolution Approach to the Game of Pac-Man

Daniel Tauritz, Ph.D.

October 7, 2020

Synopsis

The goal of this assignment set is for you to become familiarized with (I) unambiguously formulating complex problems in terms of optimization, (II) implementing an Evolutionary Algorithm (EA) of the Genetic Programming (GP) persuasion, (III) conducting scientific experiments involving EAs, (IV) statistically analyzing experimental results from stochastic algorithms, and (V) writing proper technical reports. The problem you will be solving is to employ GP to first evolve a controller for Pac-Man and subsequently to coevolve controllers for both Pac-Man and the Ghosts. This problem is representative of a large and very important class of problems which require the identification of system models such as controllers, programs, or equations. An example of the latter is symbolic regression which attempts to identify a system model based on a limited number of observations of the system's behavior; classic mathematical techniques for symbolic regression have certain inherent limitations which GP can overcome. Employing GP to evolve a controller for Pac-Man is also a perfect illustration of how GP works, while avoiding many of the complications of evolving full blown computer programs.

These are individual assignments and plagiarism will not be tolerated. You must write your code from scratch in one of the approved programming languages. You are free to use libraries/toolboxes/etc, except problem and search/optimization/EA specific ones.

Problem statement

In this assignment you will implement GPac, the simplified game of Pac-Man outlined in this document, and evolve controllers for it to control Pac-Man and the Ghosts.

GPac

In GPac, the world is a two-dimensional grid and there is no world wrap. There are two types of units: Pac-Man and the Ghosts. Pac-Man always starts at the top left cell and all three the ghosts always start at the bottom right cell. These units are guided by controllers, which is what your GP will evolve. Units move in cardinal directions (up, down, left, right); Pac-Man can choose to hold position, but the Ghosts cannot. They move from one grid cell to another in a discrete fashion (i.e., they move a whole cell at a time). Units cannot move off the edges of the map. Ghosts can occupy the same grid cell as other ghosts. If Pac-Man and a Ghost occupy the same cell, the game is over. If Pac-Man and a Ghost collide (i.e., exchange cells), the game is over.

Walls

To relieve you from the burden of creating Pac-Man maps – which is both a science and an art! – you are provided 100 guaranteed solvable maps that you need to uniform randomly sample from. Note that nothing can be placed in, or move through, a cell with a wall. The format of the map files is as follows: the first line contains two integers, the first being the width and the second the height, both with a minimum value of two; the following lines consist of octothorpes (#) and tildes (∼) where # indicates that a cell contains wall and a ∼ indicates that a cell does not contain wall.

Pills

Before the game begins, cells are chosen at random according to a preset pill-density parameter to contain "pills". The pill-density parameter specifies the percentage chance for any given cell to contain a pill, subject to the constraints (a) at least one cell needs to contain a pill, (b) pills cannot be placed in walls and (c) Pac-Man's starting cell cannot contain a pill. Thus: *E[number of cells containing a pill] = MAX (1 , pill-density · (total number of cells - 1 - number of walls))*

If Pac-Man occupies a cell that contains a pill, the pill is removed, and Pac-Man's score is increased. When all pills have been removed from the world, the game is over.

Fruit

Each turn that the game is running, there is a user-configurable chance for a piece of fruit to spawn. There can only be one piece of fruit on the field at a time and it may not spawn in the same cell as a pill, a wall, or Pac-Man's current cell. If Pac-Man occupies the cell that contains the piece of fruit, the fruit is removed, and Pac-Man's score is increased by the fruit score which is user-configurable.

Time

Each GPac game starts with time equal to the number of grid cells in the world multiplied by the time multiplier. Each turn is one time step. When the time limit reaches zero, the game is over. This prevents games from getting stuck in infinite loops. It also promotes efficient controller evolution.

Game Play

Each turn, the game gives each of the unit's controllers the current game state. This state includes at least: where all of the units are currently located and where all of the pills are located. Each controller will then choose what move to make (up, down, left, right for all controllers, also hold just for Pac-Man). Once all of the units have determined their next move, the game state will update everyone's position and decrease the time remaining by one. Once everyone has moved, the game will check if:

1. Pac-Man and any of the Ghosts are in the same cell, causing game-over.
2. Pac-Man collided with a Ghost, causing game-over.
3. Pac-Man is in a cell with a pill, causing the pill to be removed, and the score to be adjusted.
4. Pac-Man is in a cell with a piece of fruit, causing the fruit to be removed, and the score to be adjusted.
5. All the pills are removed, causing game-over.
6. Time remaining is equal to zero, causing game-over.

Score

Pac-Man's score is equal to the percentage of the total number of pills he has consumed truncated to an integer, *plus* the score for fruit consumed. If the game ends because there are no more pills on the board, Pac-Man's score is increased by the percentage of time remaining truncated to an integer. This score can be used directly for the fitness of the Pac-Man controller. Ghost fitness should be inversely proportional to Pac-Man's fitness (for example, negate his fitness) and if the game ends due to Pac-Man's demise, then the Ghost score is increased by the percentage of time remaining truncated to an integer.

World File Format

You need to write out a sequence of your world states for a single run to facilitate debugging, visualization, and grading. The common file format you are required to use consists of header values for the width and height of the world, followed by, for each snap shot that you are outputting, a list of ordered triples consisting of <key><space><value><space><value>. The origin (0,0) is in the lower-left corner. The valid triples are:

- m Pac-Man; second value is x-coordinate; third value is y-coordinate
- 1 Ghost 1; second value is x-coordinate; third value is y-coordinate
- 2 Ghost 2; second value is x-coordinate; third value is y-coordinate
- 3 Ghost 3; second value is x-coordinate; third value is y-coordinate
- p Pill; second value is x-coordinate; third value is y-coordinate
- w Wall; second value is x-coordinate; third value is y-coordinate
- f Fruit spawned; second value is x-coordinate; third value is y-coordinate
- t End of current turn; second value is remaining time; third value is current score

Note that you only need to write out the pill and wall locations during the first snap shot as the moves of Pac-Man implicitly define the pill locations of all later snap shots. This will make your world file significantly smaller in size. Here is an example file for a world with width 40, height 30, 3 snap shots, and 3 pills:

```
40
30
m 0 29
1 39 0
2 39 0
3 39 0
w 1 1
w 36 20
w 10 10
w 2 29
p 1 29
p 36 19
p 27 8
t 2400 0
m 1 29
1 38 0
2 38 0
3 39 1
```

```
t 2399 33
m 1 28
1 38 1
2 37 0
3 39 0
t 2398 33
```

General implementation requirements

For this assignment you must implement GPac. You will need to implement a method of game-over evaluation that determines if a game-over state has occurred, the ability to send the current game state to a controller and receive an action, and a way to update the state using the actions returned. In theory, the fitness of a controller is its expected performance for an arbitrary game instance (i.e., its performance averaged over all game instances). However, as it is computationally infeasible to evaluate a controller over all possible game instances, for the purpose of this assignment it will be sufficient to play a single game instance to completion to estimate fitness. Thus the game instance has to be uniform randomly reinitialized for each fitness evaluation. We recommend but do not require the addition of a fidelity user parameter to specify the number of game instances to be played to completion to average performance over as an estimate of fitness, in order to allow experimentation with higher-fidelity fitness evaluations.

The GP controllers must be implemented as follows: for each valid action, generate the corresponding new state and apply the state evaluator encoded in the GP tree, returning the valid action with the best state evaluation. The terminal nodes consist of sensor inputs and floating point constants. The function nodes consists of mathematical operators which take as input floating point numbers and provide as output floating point numbers as well. You need to implement at minimum four sensor inputs, where the target location specifically refers to the grid cell where the controller's unit is moving to this turn, namely: (1) the Manhattan distance between the target location and the nearest Ghost, (2) the Manhattan distance between the target location and the nearest pill, (3) the number of walls immediately adjacent to the target location, and (4) the Manhattan distance between the target location and the fruit. Note: When considering a move in which a pill or the fruit is consumed, immediately increasing the Pac-Man to pill or fruit distance input may lead your controllers to think that the new state is worse than the state prior to eating the fruit. You need to implement at minimum five mathematical operators, namely: (1) addition, (2) subtraction, (3) multiplication, (4) division, and (5) rand(a,b) which returns uniform randomly a number between a and b.

Your programs need to by default take as input a configuration file *default.cfg* in which case it should run without user interaction and you must provide a command line override (this may be handy for testing on different configuration files). The configuration file should at minimum:

- the pill density,
- fruit spawning probability,
- fruit score,
- time multiplier,
- either an indicator specifying whether the random number generator should be initialized based on time in microseconds or the seed for the random number generator (to allow your results to be reproduced),
- all black-box search algorithm parameters,
- the number of runs a single experiment consists of,
- the number of fitness evaluations each run is allotted,
- the relative file path+name of the log file,

- the relative file path+name of the highest-score-game-sequence all-time-step world file, and
- the relative file path+name of the solution file(s)

The log file should at minimum include:

- the relative file path + name of the highest-score-game-sequence all-time-step world file,
- the relative file path+name of the solution file(s),
- the pill density,
- the random number generator seed,
- the black box search algorithm parameters (enough detail to recreate the config file from the log), and
- an algorithm specific result log (specified in the assignment specific section).

The highest-score-game-sequence all-time-step world file is a file in the previously specified World File Format containing a sequence of world states at every time step of typically the best run of your experiment (i.e., the run with the global best fitness). In 2a & 2b the solution file should contain the best Pac-Man controller found, in 2c the solution files should contain the best Pac-Man and Ghost controllers found respectively.

The solution file format must be exactly as follows to allow automated format checking: you need to output your parse tree(s) such that each tree node is on a new line in the format <'|' characters corresponding to tree depth><node character>, where the root of the tree is at depth 0 and the following characters represent tree nodes:

- Sensor inputs

  G Manhattan distance to nearest ghost

  P Manhattan distance to nearest pill

  W Number of immediately adjacent walls

  F Manhattan distance to nearest fruit

  #.# Constant value (where the '#' character is replaced with a one or more digits)

- Operators

  + Addition

  - Subtraction

  ***** Multiplication

  / Division

  RAND Random number between two children node values.

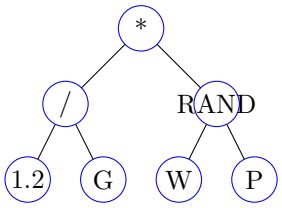
All of the above operators should have strictly two children.

For the sensor inputs P and F, if a move would place Pac-Man on top of a pill or fruit, the sensor value should be 0, rather than considering the item as collected and gone. If there is no fruit, then F should have a constant value.

Note: you are encouraged to implement additional sensor inputs and operators with your own (well documented) single or multiple character representations. These new operators are not bound to the strict two children constraint of the required operators, but make sure you document the degree or range of acceptable degrees for your operators. Additional sensor nodes often lead to interesting agent capabilities and strategies.

Example

Here is an example parse tree:



The parse tree shown above would look as follows in a solution file:

```
*
|/
||1.2
||G
|RAND
||W
||P
```

Version control requirements

For each assignment you will be given a new repository on [`https://classroom.github.com`]. **Please view your repository and the README.md file**. It may clear things up after reading this.

Included in your repository is a script named "finalize.sh", which you will use to indicate which version of your code is the one to be graded. When you are ready to submit your final version, run the command `./finalize.sh` from your local Git directory, then commit and push your code. This will create a text file, "readyToSubmit.txt", that is pre-populated with a known text message for grading purposes. You may commit and push as much as you want, but your submission will be confirmed as "final" if "readyToSubmit.txt" exists and is populated with the text generated by "finalize.sh" at 10:00pm on the due date. If you do not plan to submit before the deadline, then you should NOT run the "finalize.sh" script until your final submission is ready. If you accidentally run "finalize.sh" before you are ready to submit, do not commit or push your repo and delete "readyToSubmit.txt". Once your final submission is ready, run "finalize.sh", commit and push your code, and do not make any further changes to it.

By each assignment deadline, the code currently pushed to Master will be pulled for grading. You are required to include a `run.sh` that needs to be configured to compile and run with the command `./run.sh` using a problem file provided by the TAs and a configuration provided by you, passed in that order through the command line. Specifically, in order to run your submission and grade your output, all the TAs should have to do is execute `./run.sh`. The TAs should not have to worry about external dependencies. Any files created by your assignment must be created in the present working directory or subfolders in the present working directory.

Submissions, penalties, documents, and bonuses

You may commit and push to your repository at anytime. At submission time, your latest, pushed, commit to the master branch will be graded (if there is one). In order to ensure that the correct version of your code will be used for grading, after pushing your code, visit [`https://github.com`] and verify that your files are present. If for any reason you submit late, then **please notify the TAs when you have submitted.** If you do submit late, then your first late submission will be graded.

The penalty for late submission is a 5% deduction for the first 24 hour period and a 10% deduction for every additional 24 hour period. So 1 hour late and 23 hours late both result in a 5% deduction. 25 hours late results in a 15% deduction, etc. Not following submission guidelines can be penalized for up to 5%, which may be in addition to regular deduction due to not following the assignment guidelines.

Your code needs to compile/execute as submitted without syntax errors and without runtime errors. Grading will be based on what can be verified to work correctly as well as on the quality of your source code. You must follow the coding requirements as stated in the syllabus.

Documents are required to be in PDF format; you are encouraged to employ LaTeX for typesetting.

Note that the first two assignments in this series are weighted equally, but the third and last counts double to allow you one extra chance to significantly makeup your assignment grade.

Deliverable Categories

There are three deliverable categories, namely:

GREEN Required for all students in all sections.

YELLOW Required for students in the 6000-level sections, bonus for the students in the 5000-level section.

RED Bonus for all students in all sections.

Note that the max grade for the average of all assignments in Assignment Series 2, including bonus points, is capped at 100%.

Assignment 2a: Random Search

You must implement GPac, a random action generator for the Ghosts which generates valid actions out of the four possible actions for the Ghosts, and a random controller for Pac-Man with the following behavior: assign each new controller a random vector of four weights, generated such that the weights can be positive or negative. On each turn, for each of Pac-Man's valid possible moves, compute the four non-constant sensor inputs (G, P, W, F) defined above. Compute a weighted sum of these using the controller's weight vector to get a score for that move. Then, make the move with the highest score. This is a simplified controller compared to the trees you will implement in assignments 2b and 2c.

This assignment will employ a different solution file format requirements than assignments 2b and 2c to reflect the simplified controller representation. For this assignment only, it is acceptable to format your solution files to contain a single-line mathematical expression using the weights found during your search and the sensor node encoding described in the general implementation requirements.

The result log should be headed by the label "Result Log" and consist of empty-line separated blocks of rows where each block is headed by a run label of the format "Run i" where i indicates the run of the experiment and where each row is tab delimited in the form <evals><tab><highest score> (not including the < and > symbols) with <evals> indicating the number of game sequences executed so far and <highest score> is the score of the game sequence with the highest score found so far in this run. The first row has 1 as value for <evals>. Rows are only added if they improve on the highest score found so far.

The deliverables of this assignment are:

GREEN 1 your source code, configured to compile and run with '*./run.sh <optional configuration filepath>*' (including any necessary support files such as makefiles, project files, etc.).

GREEN 2 four configuration files, using the following world descriptions (given as pill density, fruit spawn probability, fruit score, time multiplier): (50%,1%, 10, 2), (80%,1%, 10, 2), and (50%,100%, 10, 2), and configured for 30 runs of 2000 fitness evals each (i.e., games run to completion), timer initialized random seed, along with the corresponding log files, highest-score-game-sequence all-time-step world files, and solution files (these should go in the repo's *logs*, *worlds*, and *solutions* directories respectively).

GREEN 3 a document headed by your name, AU E-mail address, and the string "COMP x66y Fall 2020 Assignment 2a", where x and y need to reflect the section you are enrolled in, containing for each of the four configuration files, the evals versus fitness plot corresponding to your log file which produced the best solution (this should be a stair-step graph similar to that of Assignment 1a).

RED 1 In order to demonstrate that an EA is a reasonable tool for solving a given problem, it is generally more compelling to compare the EA to a simple optimization algorithm such as a hill climber, rather than random search. Showing that the EA outperforms a hill climber indicates that the problem being solved is probably multimodal, and that evolution allows a more effective exploration of the search space. Up to 15% bonus points can be earned by investigating the use of a hill climber to optimize controller weights by making small changes to the weight vector and accepting changes that improve fitness. This bonus investigation needs to be documented, including result plots, in a separate section of the required document marked as "Hill Climber". You also need to indicate in your source files any code which pertains to this bonus and additionally describe it in your README.md file. Basically, you need to make it as easy as possible for the TAs to see with a glance what part of your submission pertains to this bonus, and which does not.

Edit your README.md file to explain anything you feel necessary. Submit all files via GitHub, by *pushing* your latest commit to the *master* branch. The due date for this assignment is 10:00 PM on Sunday October 25, 2020.

Grading
The point distribution is as follows:

Assessment Rubric \ Deliverable Category	Green	Red
Algorithmic	50	50
Configuration files and parsing	15	5
Logging and output files	15	5
Good programming practices including code reliability and commenting	10	10
Document containing evals versus fitness plots	10	30

Assignment 2b: Genetic Programming Search

You need to evolve a GP tree controller for Pac-Man which generates the most high-quality valid action out of the 5 possible actions for Pac-Man that it can find, employing the approach described in the general specification. The Ghosts should be controlled by the same random action generator as specified in Assignment 2a.

You need at minimum to implement support for the following EA configurations, where operators with multiple options are comma separated, and each operator needs to be implemented in a separate function to increase modularity and code readability:

Representation Tree

Initialization Ramped half-and-half (see Section 6.4 in [1])

Parent Selection Fitness Proportional Selection, Over-Selection (see Section 6.4 in [1])

Recombination Sub-Tree Crossover

Mutation Sub-Tree Mutation

Survival Selection Truncation, k-Tournament Selection without replacement

Bloat Control Parsimony Pressure

Termination Number of evals, no change in best fitness for n generations

Your configuration file should allow you to select which of these configurations to use as well as how many runs a single experiment consists of. Your configurable EA strategy parameters should include all those necessary to support your operators, such as:

- μ
- λ
- D_{max} for ramped half-and-half initialization
- k for survival selection
- p for parsimony pressure penalty coefficient
- Number of evals till termination
- n for termination convergence criterion

The result log should be headed by the label "Result Log" and consist of empty-line separated blocks of rows where each block is headed by a run label of the format "Run i" where i indicates the run of the experiment and where each row is tab delimited in the form <evals><tab><average fitness><tab><best fitness> (not including the < and > symbols) with <evals> indicating the number of evals executed so far, <average fitness> is the average population fitness at that number of evals, and <best fitness> is the fitness of the best individual in the population at that number of evals (so local best, not global best!). The first row has $< \mu >$ as value for <evals>. Rows are added after each generation, so after each λ evaluations. The solution file should consist of the best solution (i.e., controller for Pac-Man) found in any run.

The deliverables of this assignment are:

GREEN 1 your source code configured to compile and run with '*.run.sh <optional configuration filepath>*' (including any necessary support files such as makefiles, project files, etc.)

GREEN 2 four best configuration files you can find, one for each of the following four world descriptions (given as pill density, fruit spawn probability, fruit score, time multiplier): (50%,1%, 10, 2), (80%,1%, 10, 2), and (50%,100%, 10, 2), and configured for 30 runs of 2000 fitness evals each (i.e., games run to completion), the corresponding log files, highest-score-game-sequence all-time-step world files, and solution files

GREEN 3 a document headed by your name, AU E-mail address, and the string "COMP x66y Fall 2020 Assignment 2b", where x and y need to reflect the section you are enrolled in, a methodology section describing your recombination, mutation, and parsimony implementations, and a results section containing for each of the four configuration files, the corresponding plot of the fitness evals versus average local best fitness (box plot preferred) and statistical analysis comparing the final average best GP result with the average final random result from 2a.

YELLOW 1 Investigate the role of parsimony pressure with respect to both parsimony technique (e.g., tree depth versus tree size) and parsimony coefficient (i.e., the amount of penalty incurred for bloat). This bonus investigation needs to be documented, including evals versus a relevant tree complexity metric as well as the usual evals versus best and average fitness plots averaged over all runs, and also including appropriate statistical analysis, in a separate section of the required document marked as "Parsimony Investigation". You also need to indicate in your source files any code which pertains to this bonus and additionally describe it in your README.md file. Basically, you need to make it as easy as possible for the TAs to see with a glance what part of your submission pertains to this deliverable. Students enrolled in COMP 5660 can earn up to 15% bonus for this investigation. For students enrolled in COMP 6660/6666 this will count for 15% regular points (not bonus) beyond the GREEN deliverables.

RED 1 Up to 10% bonus points can be earned by investigating having multiple simultaneous Pac-Man's all employing *identical* controllers, where they all have to die for the game to end, and they share the same score (i.e., there's no competition between the Pac-Man's). You should add appropriate additional terminals, such as "Distance to nearest other Pac-Man". This bonus investigation needs to be documented, including result plots, in a separate section of the required document marked as "Multiple Identical Pac-Man Controllers". You also need to indicate in your source files any code which pertains to this bonus and additionally describe it in your README.md file. Basically, you need to make it as easy as possible for the TAs to see with a glance what part of your submission pertains to this bonus, and which does not.

RED 2 Up to 10% bonus points can be earned by investigating having multiple simultaneous Pac-Man's employing *different* controllers, where they all have to die for the game to end, and they share the same score (i.e., there's no competition between the Pac-Man's). You should add appropriate additional terminals, such as "Distance to nearest other Pac-Man". This bonus investigation needs to be documented in a separate section of the required document marked as "Multiple Unique Pac-Man Controllers". You also need to indicate in your source files any code which pertains to this bonus and additionally describe it in your README.md file. Basically, you need to make it as easy as possible for the TAs to see with a glance what part of your submission pertains to this bonus, and which does not.

Include a readme file to explain anything you feel necessary. The due date for this assignment is 10:00 PM on Sunday November 8, 2020.

Grading
The point distribution per deliverable category is as follows:

Assessment Rubric \ Deliverable Category	Green	Yellow	Red
Algorithmic	50%	55%	70%
Configuration files and parsing	10%	5%	5%
Logging and output/solution files	10%	5%	5%
Good programming practices & robustness	10%	10%	10%
Document including evals versus fitness plots	15%	10%	10%
Statistical analysis	5%	15%	0%

Assignment 2c: Competitive Coevolutionary Search

You need to coevolve within a configurable number of fitness evaluations, where a fitness eval is now defined to be a single game played, a GP controller for Pac-Man which generates the supposedly optimal valid action out of the 5 possible actions for Pac-Man and a GP controller shared by all three Ghosts. The type of coevolution you will be using is competitive coevolution [1, Section 15.3] employing two populations, one for the Pac-Man controllers and one for the Ghost controllers. The recommended GP approach is the same as in Assignment 2b, modified appropriately for the Ghosts. Ghosts require at minimum two different terminal sensor inputs: (1) distance to Pac-Man (M in the solution file tree encoding), and (2) distance to nearest other Ghost (G in the solution file tree encoding).

You need at minimum to implement support for the following EA configurations, where operators with multiple options are comma separated, and operators need to be able to be configured independently for Pac-Man and the Ghosts respectively:

Pac-Man Representation Tree

Pac-Man Initialization Ramped half-and-half (see Section 6.4 in [1])

Pac-Man Parent Selection Fitness Proportional Selection, Over-Selection (see Section 6.4 in [1])

Pac-Man Recombination Sub-Tree Crossover

Pac-Man Mutation Sub-Tree Mutation

Pac-Man Survival Selection Truncation, k-Tournament Selection without replacement

Pac-Man Bloat Control Parsimony Pressure

Ghost Representation Tree

Ghost Initialization Ramped half-and-half (see Section 6.4 in [1])

Ghost Parent Selection Fitness Proportional Selection, Over-Selection (see Section 6.4 in [1])

Ghost Recombination Sub-Tree Crossover

Ghost Mutation Sub-Tree Mutation

Ghost Survival Selection Truncation, k-Tournament Selection without replacement

Ghost Bloat Control Parsimony Pressure

Termination Number of evals

Your configuration file should allow you to select which of these configurations to use as well as how many runs a single experiment consists of. Your configurable EA strategy parameters should include all those necessary to support your operators, such as:

- $\mu_{Pac-Man}$,μ_{Ghost}
- $\lambda_{Pac-Man}$,λ_{Ghost}
- $k_{Pac-Man}$,k_{Ghost} for survival selection
- $p_{Pac-Man}$,p_{Ghost} for parsimony pressure penalty coefficient
- Number of evals till termination

Note: If $\mu_{Pac-Man}$ is not equal to μ_{Ghost} or $\lambda_{Pac-Man}$ is not equal to λ_{Ghost}, some controllers will have to be used more than once to complete all fitness evaluations. If you use a controller more than once, simply take the average of all games the controller played.

The result log should be headed by the label "Result Log" and consist of empty-line separated blocks of rows where each block is headed by a run label of the format "Run i" where i indicates the run of the experiment and where each row is tab delimited in the form <evals><tab><average fitness><tab><best fitness> (not including the < and > symbols) with <evals> indicating the number of evals executed so far, <average fitness> is the average Pac-Man population fitness at that number of evals, and <best fitness> is the fitness of the best individual in the Pac-Man population at that number of evals (so local best, not global best!). Rows are added after each generation. The solution files should consist of the best solutions (i.e., controllers for Pac-Man and the Ghosts) found in the final generation of any run.

For each experiment in this assignment you are asked to submit highest-score-game-sequence all-time-step world files from an exhibition game (i.e., an extra match does not count as an evaluation) played between your best Pac-Man and Ghost controllers from the final generation of the same run. Similarly, you will produce two solution files per experiment: one for the Pac-Man controller and one for the Ghost controller that played in this game.

All experiments in this assignment are to be conducted on a configuration file using the following world description (given as pill density, fruit spawn probability, fruit score, time multiplier): (50,1%, 10, 2) with termination after at least 2,000 fitness evaluations (note: all experiments should use the same number of fitness evaluations). Informally experiment with the sensitivity of the final local best to the EA strategy parameters to determine which parameter seems to make the most difference. Then formally experiment with the sensitivity of the final local best to that parameter by at minimum trying three different values for it and collecting statistics for 30 runs. Use an appropriate statistical test (e.g., t-test) to determine with $\alpha = 0.05$ which combinations are statistically better in terms of final local best. Make three plots, one for each combination, with each plot showing evals vs. population mean fitness averaged over the 30 runs (fitness on the left vertical axis).

The deliverables of this assignment are:

GREEN 1 Your source code (including any necessary support files such as makefiles, project files, etc.)

GREEN 2 The three configuration files and corresponding three log files, three highest-score-game-sequence all-time-step exhibition world files for the best Pac-Man and Ghost controllers in the final generation, and corresponding six solution files.

GREEN 3 a document in PDF format headed by your name, AU E-mail address, and the string "COMP x66y Fall 2020 Assignment 2c", where x and y need to reflect the section you are enrolled in', containing:

- **Methodology** Describe the custom parts of your EA design, such as your function and terminal sets, as well as your approach to performing adversarial evaluations, in sufficient detail to allow someone else to implement a functionally equivalent EA, and explain your EA design decisions.
- **Experimental Setup** Provide your experimental setup in sufficient detail to allow exact duplication of your experiments (i.e., producing the exact same results within statistical margins) and justify your choice of EA strategy parameters.
- **Results** List your experimental results in both tabular and graphical formats (box plots preferred) along with your statistical results, corresponding to the three configuration and log files, and six solution files, referenced above (so you'll have three plots and a table containing your statistical comparison of the three combinations). Note that you need to take into consideration that fitness based on direct comparisons between individuals in opposing populations is relative in coevolution. Make sure to explain how you addressed this in your comparisons.
- **Discussion** Discuss your experimental and statistical results, providing valuable insights such as conjectures you induce from your results. Your choice of what to report on and how you go about rationalizing it is your subjective interpretation.

Conclusion Conclude your report by stating your most important findings and insights in the conclusion section.

Bibliography This is where you provide your citation details, if you cited anything. Only list references here that you actually cite in your report.

Appendices If you have more data you want to show than what you could reasonably fit in the body of your report, this is the place to put it along with a short description.

GREEN 4 The configuration files needed to recreate the results reported in your document; clearly mark them as to which configuration file goes with which reported experiment, document in the README.md file exactly how to use the configuration files (in addition, self-documenting configuration files are strongly encouraged), corresponding log files and solution files.

YELLOW 1 Investigate under what circumstances coevolutionary cycling occurs in Pac-Man and the Ghosts and adding a section to the document to describe all aspects of your investigation, including CIAO plots [2] to visualize your findings. You need to include all your related configuration, log, world, and solution files and you need to indicate in your source files any code which pertains to this bonus and additionally describe this in your readme file. Basically, you need to make it as easy as possible for the TAs to see with a glance what part of your submission pertains to this bonus, and which does not. Students enrolled in COMP 5660 can earn up to 10% bonus for this investigation. For students enrolled in COMP 6660/6666 this will count for 10% regular points (not bonus) beyond the GREEN deliverables.

RED 1 Up to 10% bonus points can be earned by investigating having each Ghost employ a different GP controller and comparing that to the base case where they share the same GP controller. You need to add a section to the document to describe all aspects of your investigation, include all your related configuration/log/world/solution files, and indicate in your source files any code which pertains to this bonus and additionally describe this in your readme file. Basically, you need to make it as easy as possible for the TAs to see with a glance what part of your submission pertains to this bonus, and which does not.

RED 2 Each of the following four sub-bonuses has to do with coevolving Pac-Man and Ghost controllers where you are investigating having multiple simultaneous Pac-Men which both have to die for the game to end and who share the same score (i.e., there's no competition between the Pac-Men). You should add appropriate additional terminals, such as "Distance to nearest other Pac-Man".
You can opt to select one or more of the following four sub-bonuses. If you opt for multiple of these sub-bonuses, then you need to perform appropriate comparisons between them. Either way, you need to add a section to the document to describe all aspects of your investigation, include all your related configuration/log/world/solution files, and indicate in your source files any code which pertains to this bonus and additionally describe this in your readme file. Basically, you need to make it as easy as possible for the TAs to see with a glance what part of your submission pertains to this bonus, and which does not.
Bonus RED 2a
Up to 5% bonus points can be earned by investigating having multiple Pac-Men controlled by the same GP controller and multiple Ghosts having the same GP controller.
Bonus RED 2b
Up to 5% bonus points can be earned by investigating having multiple Pac-Men controlled by a different GP controller and multiple Ghosts having the same GP controller.
Bonus RED 2c
Up to 5% bonus point can be earned by investigating having multiple Pac-Men controlled by the same GP controller and multiple Ghosts having different GP controllers.
Bonus RED 2d
Up to 5% bonus points can be earned by investigating having multiple Pac-Men controlled by a different GP controller and multiple Ghosts having different GP controllers.

RED 3 Up to 10% bonus points can be earned by investigating simultaneously varying the number of Ghosts and the speed ratio between Pac-Man and the Ghosts, where the speed ratio is defined as the number of actions Pac-Man can execute for each Ghost action (so a ratio of 1 is the base case, a ratio of 1.5 would mean that every other turn Pac-Man takes two actions versus the Ghosts just one). You need to add a section to the document to describe all aspects of your investigation, including a three-dimensional win-loss ratio graph with number of Ghosts on the horizontal axis, speed ratio between Pac-Man and the Ghosts on the second axis, and win-loss ratio on the third axis, include all your related configuration/log/world/solution files, and indicate in your source files any code which pertains to this bonus and additionally describe this in your readme file. Basically, you need to make it as easy as possible for the TAs to see with a glance what part of your submission pertains to this bonus, and which does not.

Include a readme file to explain anything you feel necessary. The due date for this assignment is 10:00 PM on Sunday November 29, 2020.

Grading
The point distribution per deliverable category is as follows (note that 2c has twice the points as 2a & 2b):

Assessment Rubric \ Deliverable Category	Green	Yellow	Red
Algorithmic	40%	30%	30%
Configuration files and parsing	5%	5%	5%
Logging and output/solution files	10%	0%	5%
Good programming practices & robustness	10%	10%	10%
Document	25%	30%	25%
Statistical analysis	10%	25%	25%

References

[1] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. Second Edition, Springer-Verlag, Berlin Heidelberg, 2015, ISBN 978-3-662-44873-1.

[2] Dave Cliff and Geoffrey F. Miller, *Tracking the red Queen: Measurements of Adaptive Progress in Co-Evolutionary Simulations*. In Advances in Artificial Life, Lecture Notes in Computer Science, Volume 929, Pages 200-218, Springer-Verlag, Berlin Heidelberg, 1995, ISBN 978-3-540-59496-3. `http://www.cs.uu.nl/docs/vakken/ias/stuff/cm95.pdf`