Evolutionary Computing
COMP 5660-001/6660-001/6660-D01 – Auburn University
Fall 2022 – Assignment Series 2
GPac: A Genetic Programming & Co-evolution Approach to the Game of Pac-Man

Deacon Seals
Braden Tisdale
Daniel Tauritz, Ph.D.

January 10, 2023

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 (also referred to as a Pac-Man agent) and subsequently to co-evolve controllers for Pac-Man with controllers for 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 in Python using the provided assignment framework. You are free to use libraries/toolboxes/etc, except for problem-specific or search/optimization/EA-specific ones. We will allow any standard Python library (e.g., `random` and `json`), in addition to well-known libraries for generic data processing (e.g., `numpy`) or visualization (e.g., `matplotlib`). If you want to use something outside these categories, or anything not provided in the base Conda Linux environment, ask a TA for permission.

Problem statement

In this assignment you will implement and evolve controllers for Pac-Man and co-evolve against 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. 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 a wall and a ∼ indicates that a cell does not contain a wall. We are also providing a function to parse these map files for you.

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:

$$E[number\ of\ pills] = MAX(1,\ pill\ density \cdot (number\ of\ cells - number\ of\ walls - 1))$$

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 enters the cell that contains the 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, and no movement for Pac-Man only). 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 the game score) 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

GPac generates 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

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 series you must implement GPac controllers for Pac-Man. You are provided with an implementation of GPac with proper score calculation, spawn mechanics, game-over identification, and world file generation (called a game log in the code). 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. Your code needs to be compatible with the provided GPac implementation and adhere to the specifications of the individual assignments in this series.

The GP controllers must be implemented as follows: for each valid action and the corresponding new state from executing the action, 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.

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.

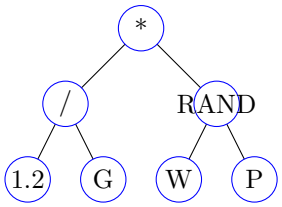
All of the above operators should have strictly two children.

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 "`chmod 755 finalize.sh && ./finalize.sh`" from your repository then type in your Auburn username. This will create a text file `readyToSubmit.txt` which lets us know your submission is finished. Commit and push this file to your default branch to submit your assignment. You may commit and push as many times as you like, but your submission will be considered finalized if `readyToSubmit.txt` exists in the default branch after 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, make sure to delete `readyToSubmit.txt` before pushing. Similarly, if it is past the due date and you have already pushed `readyToSubmit.txt`, do not make any further pushes to your repo.

After submission, your latest, pushed, commit to the default branch will be graded if it contains `readyToSubmit.txt`. In order to ensure that the correct version of your code will be used for grading, after pushing your code, examine your repo [`https://github.com`] and verify that you have submitted what you intended to. If for any reason you submit late, then **please notify the TAs when you have submitted.**

Submission, penalties, documents, and bonuses

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.

The code pushed to the default branch after submission will be pulled for grading. Any files created by your assignment must be created in the present working directory or subdirectories within it. All Jupyter notebooks must be completed and submitted with results from running the full notebook. Your submitted code needs to execute as expected, within the base Conda Linux environment, without error. The TAs should not have to worry about any external dependencies or environments. 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. Always remember that the TAs will thoroughly examine everything by hand, and that your code being easy to read and understand is a substantial part of your grade (*and their sanity*).

Documents are required to be in PDF format; you are encouraged (but not required) to employ LaTeX for typesetting.

Note that the first two assignments in this series are weighted equally, but the third, final assignment counts double to allow you one extra chance to significantly improve 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%. That is, if you received 100%, 90%, and 120% on the individual assignments, you will receive a 100% for Assignment Series 2.

Assignment 2a: Random Search

You must implement a random search through valid parse tree space for Pac-Man controllers in GPac. In this assignment, you are asked to complete the Jupyter notebook `2a_notebook.ipynb` and several other Python files as directed by the notebook. Your submission should also contain a report to document the findings of a 30-run experiment as well as files containing the world file (log) and tree from the solution with the highest fitness from all runs. In your report, include a stair-step plot showing the progression of number of fitness evaluations versus local best fitness for the run that produced the highest fitness overall, the standard deviation and mean of the best fitness found from each run, and an informal analysis of your agent's performance from watching the visualization of the highest-fitness solution. In this informal analysis, we want you to comment on whether or not you think the agent performs well, as well as any notable behavioral quirks.

The deliverables of this assignment are:

GREEN 1 your source code and completed notebook

GREEN 2 a PDF document headed by your name, AU E-mail address, and the string "COMP x660 Fall 2022 Assignment 2a", where x needs to reflect the section you are enrolled in, containing your report, including statistical analysis and plot(s)

GREEN 3 files containing any data you analyzed to write your report or generate your plot(s), in a format that can be easily understood by the TAs (if you think you might should include instructions on how to interpret your data, then you should!), as well as the highest-score-game-sequence all-time-step world files (the game logs) and solution files (these should go in the *worldFiles* and *solutions* subdirectories respectively).

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 controllers by making small changes to the controller and accepting changes that improve fitness. This bonus investigation needs to be documented, including result plots and a new config file, 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 report. Basically, you need to make it as easy as possible for the TAs to see at a glance what parts of your submission pertain to this bonus, and what parts do not.

Submit all files via GitHub, by *pushing* your latest commit to the default branch, including `readyToSubmit.txt`. The due date for this assignment is 10:00 PM on Sunday October 30, 2022.

Grading
The point distribution is as follows:

Assessment Rubric \ Deliverable Category	Green	Red
Algorithmic	50	55
Logging and output files	15	5
Programming practices, readability, and implementation	10	10
Report and plot(s)	25	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 map should be the same static map as specified in Assignment 2a. In this assignment, you are asked to complete the Jupyter notebook `2b_notebook.ipynb` and several other Python files as directed by the notebook.

You need at minimum to implement support for the following EA configurations, as described in the notebook:

Representation Tree

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

Parent Selection Fitness Proportional Selection, k-Tournament Selection without replacement, uniform random selection

Recombination Sub-Tree Crossover

Mutation Sub-Tree Mutation

Survival Selection Truncation, k-Tournament Selection without replacement

Bloat Control Parsimony Pressure

Termination Number of fitness evaluations

Your submission should also contain a report to document the findings of a 30-run experiment as well as files containing the world file (log) and tree from the solution with the highest fitness from all runs. In your report, include the following:

- a plot showing fitness evaluations versus best and average population fitness averaged over 30 runs (like Assignments 1b-1d)
- statistical analysis (F-test and t-test) comparing the best fitness obtained by each run to the data generated by the search algorithm in Assignment 2a
- a brief discussion interpreting the results of the statistical tests
- an informal comparison of the behavior of the best Assignment 2a agent and the best GP agent from watching the visualization of the highest-scoring solutions from each algorithm

The deliverables of this assignment are:

GREEN 1 your source code and completed notebook

GREEN 2 a PDF document headed by your name, AU E-mail address, and the string "COMP x660 Fall 2022 Assignment 2b", where x needs to reflect the section you are enrolled in, containing your report, including statistical analysis and plot(s)

GREEN 3 files containing any data you analyzed to write your report or generate your plot(s), in a format that can be easily understood by the TAs (if you think you might should include instructions on how to interpret your data, then you should!), as well as the highest-score-game-sequence all-time-step world files, and solution files (these should go in the repo's *worldFiles*, and *solutions* directories respectively).

YELLOW 1 Investigate the role of parsimony pressure with respect to two distinct parsimony techniques (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 fitness evaluations versus a relevant tree complexity metric as well as the usual fitness evaluations versus best and average fitness plots averaged over all runs, and also including appropriate statistical analysis, in a separate section of your report 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 report. 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 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 characters 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 players). You should add appropriate additional terminals, such as "Distance to nearest other Pac-Man". This bonus investigation needs to be documented, including result plots and informal commentary of behaviors observed in the highest-score log of a 30-run experiment, in a separate section of the required document marked as "Multiple Identical Pac-Man Controllers". 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 In addition to completing RED 1, up to 15% bonus points can be earned by investigating having multiple simultaneous Pac-Man characters all 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 players). In addition to the new primitives implemented in RED 1, you should implement an alternative version of the *play_GPac* function called *play_GPac_multicontroller* that accepts a list of Pac-Man controllers (in addition to the typical parameters) and uses each controller to determine moves for a particular Pac-Man player in the game. Evolution should utilize a single population of controllers and fitness should be assessed each generation with stochastic controller paring such that all controllers play an equal number of games in that generation, each controller plays at least one game per generation, and the fitness of an individual is determined by averaging the fitness obtained in each game in a generation. For the sake of calculating fitness evaluations, each individual game played should be counted as a fitness evaluation. This bonus investigation needs to be documented in a separate section of the required document marked as "Multiple Distinct Pac-Man Controllers," including result plots, an informal commentary of behaviors observed in the highest-score log of a 30-run experiment, and a comparison of these behaviors with those observed in RED 1.

RED 3 Up to 10% bonus points can be earned by getting a head start on Assignment 2c and investigating the evolution of a controller that controls all ghosts and plays against a static Pac-Man strategy. This investigation should use negative game score plus remaining time as the fitness metric. You should add appropriate additional terminals, such as "Distance to nearest Pac-Man" and "Distance to nearest other ghost". This bonus investigation needs to be documented in a separate section of the required document marked as "Multiple Identical Ghost Controllers". You also need to indicate in your source files any code which pertains to this bonus and additionally describe it in your report. 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 4 In addition to completing RED 3, up to 15% bonus points can be earned by investigating having multiple simultaneous ghosts all employing *different* controllers, where a static Pac-Man strategy is used and all ghosts share the same fitness for a particular game (i.e., there's no competition between the ghost players). In addition to the components implemented in RED 3, you should implement an alternative version of the *play_GPac* function called *play_GPac_multicontroller* that accepts a list of ghost controllers (in addition to the typical parameters) and uses each controller to determine moves

for a particular ghost player in the game. Evolution should utilize a single population of controllers and fitness should be assessed each generation with stochastic controller pairing such that all controllers play an equal number of games in that generation, each controller plays at least one game per generation, and the fitness of an individual is determined by averaging the fitness obtained in each game in a generation. For the sake of calculating fitness evaluations, each individual game played should be counted as a fitness evaluation. This bonus investigation needs to be documented in a separate section of the required document marked as "Multiple Distinct Ghost Controllers," including result plots, an informal commentary of behaviors observed in the highest-score log of a 30-run experiment, and a comparison of these behaviors with those observed in RED 3.

Submit all files via GitHub, by *pushing* your latest commit to the default branch, including `readyToSubmit.txt`. The due date for this assignment is 10:00 PM on Sunday November 13, 2022.

Grading
The point distribution per deliverable category is as follows:

Assessment Rubric \ Deliverable Category	Green	Yellow	Red
Algorithmic and tuning	50%	55%	65%
Logging and output files	10%	5%	5%
Programming practices, readability, and implementation	10%	10%	10%
Report and plot(s)	15%	15%	20%
Statistical analysis	15%	15%	0%

Assignment 2c: Competitive Co-evolutionary Search

You need to co-evolve within a configurable number of fitness evaluations, a GP controller for Pac-Man and a GP controller for all Ghosts where a single fitness evaluation is counted as a single full game played between competing controllers. In this assignment, Ghosts within a given game of GPac will share a singular controller to determine their individual actions. The fitness of a Ghost controller should be calculated as negative game score plus remaining time. The type of co-evolution you will be using is competitive co-evolution [1, Section 15.3] employing two populations, one for the Pac-Man controllers and one for the Ghost controllers. The recommended GP approaches are the same for Pac-Man and Ghost controllers as in Assignment 2b, though each species of controller will use different sets of GP primitives. Specifically, Ghosts require the same primitives as Pac-Man except for the following new or modified sensor inputs:

G Manhattan distance to nearest other ghost

M Manhattan distance to nearest Pac-Man

In practice, it is necessary to evaluate individuals against a sample of opponents in the competing population, though it is sufficient in this assignment to evaluate an individual against a single opponent to manage computational cost. However, you must ensure that individuals are evaluated each generation and that fitness is updated to accommodate for evolving opponents. In this assignment, you are asked to complete the Jupyter notebook `2c_notebook.ipynb` and several other Python files as directed by the notebook.

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 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, k-Tournament Selection without replacement, uniform random selection

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, k-Tournament Selection without replacement, uniform random selection

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 fitness evaluations

Note: If μ of the Pac-Man population is not equal to μ of the Ghost population or λ of the Pac-Man population is not equal to λ of the Ghost population, 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 for fitness and store the game log of the game that was better for the individual in question (e.g., Pac-Man uses the log with the higher score and a Ghost uses the log with the lower score).

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 controller and Ghost 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 used in this game. Note that for all deliverables except RED 1, acceptable statistical analysis consists of reporting the mean and standard deviation of the best fitness of the final generation of each species across all runs in your experiment.

The deliverables of this assignment are:

GREEN 1 your source code and completed notebook

GREEN 2 a document in PDF format headed by your name, AU E-mail address, and the string "COMP x660 Fall 2022 Assignment 2c", where x needs 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 and justify your choice of EA strategy parameters.
- **Results** List your experimental results in both tabular (mean and standard deviation) and graphical formats (box plots preferred).
- **Discussion** Discuss any notable behavioral characteristics of the best controllers when compared to the controllers found in Assignment 2b. In particular, we want you to report on how and if Pac-Man agent behavior changed in the presence of co-evolving Ghosts.
- **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 3 files containing any data you analyzed to write your report or generate your plot(s), in a format that can be easily understood by the TAs (if you think you might should include instructions on how to interpret your data, then you should!), as well as the highest-score-game-sequence all-time-step world files, and solution files (these should go in the repo's *worldFiles*, and *solutions* directories respectively).

YELLOW 1 Investigate under what circumstances co-evolutionary cycling occurs in Pac-Man and Ghosts and add 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, data, world, and solution files and clearly indicate in your source files any code which pertains to this deliverable. 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, and which does not. Students enrolled in COMP 5660 can earn up to 10% bonus for this investigation. For students enrolled in COMP 6660 this will count for 10% regular points (not bonus) beyond the GREEN deliverables.

RED 1 You may have noticed that this assignment doesn't contain any request for rigorous statistical analysis. This is because performing meaningful statistical analysis is often nuanced (and computationally expensive) for problems without an inherent absolute fitness metric (like GPac). For up to 15% bonus points, design and implement a method for performing a meaningful, fair, and consistent comparison between solutions from any two co-evolution configurations/experiments co-evolving the same species, then perform statistical analysis using metrics obtained from your comparison method. Your method and statistical analysis results (with your interpretation of results) should be documented in a separate section of the required document marked as "Statistical Analysis of Co-Evolution". You also need to indicate in your source files any code which pertains to this bonus and additionally describe it in your report. 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.

Note on RED 2-6 Deliverables RED 2-6 expand upon the RED deliverables of Assignment 2b and involve exploring permutations of co-evolution with one or more Pac-Man players employing shared or unique controllers and Ghost players employing shared or unique controllers. If exploring scenarios involving agents of the same type employing unique controllers for each player, you should implement the *play_GPac_multicontroller* function described in Assignment 2b. In such scenarios, you have the option of co-evolving a population for player, a single population that where controllers are sampled for each player of a given type, or by evolving multi-controller individuals. In any event, you must ensure that each individual of a population is evaluated at least once per generation.

In scenarios involving multiple Pac-Man players in a game, you should add appropriate additional Pac-Man terminals, such as "Distance to nearest other Pac-Man" and ensure that Ghost terminals behave correctly if multiple Pac-Man characters exist. All Pac-Man characters have to die for the game to end, and they share the same score (i.e., there's no competition between the Pac-Man players). Additionally, all ghosts share the same fitness for a particular game (i.e., there's no competition between the ghost players).

RED 2 Up to 5% bonus points can be earned by investigating competitive co-evolution using multiple simultaneous Pac-Man characters all employing *identical* controllers and Ghost characters sharing a common controller. This bonus investigation needs to be documented, including an informal commentary of novel behaviors observed in the highest-score log of a 30-run experiment, in a separate section of the required document marked as "Multiple Identical Pac-Man Controllers vs Multiple Identical Ghost Controllers". 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 3 Up to 10% bonus points can be earned by investigating competitive co-evolution of multiple simultaneous Pac-Man characters all employing *different* controllers and Ghost characters sharing a common controller. This bonus investigation needs to be documented, including an informal commentary of novel behaviors observed in the highest-score log of a 30-run experiment, in a separate section of the required document marked as "Multiple Distinct Pac-Man Controllers vs Multiple Identical Ghost Controllers". 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 4 Up to 10% bonus points can be earned by investigating competitive co-evolution of a controller for a single Pac-Man and multiple simultaneous ghosts all employing *different* controllers. This bonus investigation needs to be documented, including an informal commentary of novel behaviors observed in the highest-score log of a 30-run experiment, in a separate section of the required document marked as "Single Pac-Man Controllers vs Multiple Distinct Ghost Controllers". 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 5 Up to 10% bonus points can be earned by investigating competitive co-evolution of multiple simultaneous Pac-Man characters all employing *identical* controllers and multiple simultaneous ghosts

all employing *different* controllers. This bonus investigation needs to be documented, including an informal commentary of novel behaviors observed in the highest-score log of a 30-run experiment, in a separate section of the required document marked as "Multiple Identical Pac-Man Controllers vs Multiple Distinct Ghost Controllers". 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 6 Up to 10% bonus points can be earned by investigating competitive co-evolution of multiple simultaneous Pac-Man characters all employing *different* controllers and multiple simultaneous ghosts all employing *different* controllers. This bonus investigation needs to be documented, including an informal commentary of novel behaviors observed in the highest-score log of a 30-run experiment, in a separate section of the required document marked as "Multiple Distinct Pac-Man Controllers vs Multiple Distinct Ghost Controllers". 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.

The due date for this assignment is 10:00 PM on Sunday November 27, 2022. This assignment also has a unique late policy. The cumulative late penalty for submitting each day before 10:00 PM is:

- Monday the 28th: -1%
- Tuesday the 29th: -5%
- Wednesday the 30th: -25%
- Thursday the 1st: -50%
- Friday the 2nd: -100%
- Any later date: -170%

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 1	Red 2-6
Algorithmic and tuning	45%	35%	40%	25%
Logging and output/solution files	10%	5%	5%	5%
Programming practices, readability, and implementation	10%	10%	10%	10%
Report and plot(s)	25%	40%	30%	35%
Statistical analysis	10%	10%	30%	10%

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`