Evolutionary Computing
COMP 5660-001/6660-001/6660-D01 – Auburn University
Fall 2021 – Assignment Series 1
Automated Map Design for GPac

Deacon Seals
Braden Tisdale
Daniel Tauritz, Ph.D.

July 15, 2022

Synopsis

The goal of this assignment set is for you to become familiarized with (I) representing problems in mathematically precise terms, (II) implementing an Evolutionary Algorithm (EA) to solve a problem, (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 the design of maps capable of challenging an AI agent. This problem is representative of problems which require the selection or generation of complicated solutions that optimize an objective function which may not be easily solved with a heuristic-based approach, and where an exhaustive search is infeasible. This is a large class of problems which comes up frequently in the real world, and EAs are one of the most commonly-used optimizers for these problems.

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.

Problem statement

In this assignment you will implement map generation for the game GPac. Automated design is a hallmark application of evolutionary computing that can generate art, solve engineering tasks, and even write code. For this assignment series, your task is to generate difficult maps to assess the performance of AI agents and eventually creating better AI agents that play a game.

GPac

In GPac, the world is a two-dimensional rectangular grid with the origin (0, 0) in the bottom-left corner. There are two types of units: Pac-Man and the Ghosts. Pac-Man always starts at the top left cell and all of the ghosts always start at the bottom right cell. These units are guided by controllers, which will be provided for you for the first assignment series. Units move in cardinal directions (up, down, left, right). 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, and there is no world wrap. 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.

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

GPac generates a sequence of your world states for a single run to facilitate debugging, visualization, and grading. The file format 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 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). Your code needs to be compatible with the provided GPac implementation and adhere to the specifications of

the individual assignments in this series. Your submission should also compile and execute within the base Conda Linux environment as prescribed in the environment setup assignment. If you require a package not present in that base environment, please contact a TA for approval. In all assignments that contain Jupyter notebooks: those notebooks must be completed and submitted with results from running the full notebook.

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 commands `chmod 755 finalize.sh` then `./finalize.sh` from your repository, 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 many times as you like, 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.

The code currently pushed to the default branch will be pulled for grading. 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. 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, 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.

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 1, including bonus points, is capped at 100%.

Assignment 1a: Random Search

You must implement a random search algorithm which generates uniform random placement of walls within a GPac map. To simplify this problem and guarantee the validity of a generated map, you will be provided with a repair function that ensures that a path always exists between the Pac-Man and Ghost starting locations and then places walls in all unreachable cells. Maps will be described as a fixed-length list of integer 0 and 1 values where 1 indicates the presence of a wall and 0 indicates the absence of a wall. This representation will be converted to a 2-dimensional map, repaired, and evaluated by a provided fitness function.

In this assignment you are asked to complete the Jupyter notebook `1a_notebook.ipynb`, part of a Python class, and a report. The notebook will guide you through implementation where you will perform the experiments necessary to create the report. While implementing the changes listed in the notebook, think about what data you will need to record in order to write the report described below.

Once you've finished the notebook, you need to write a report. This report should include a stairstep plot showing evals vs fitness of the run which resulted in the most-fit solution. This report should also include statistical analysis (F-test and t-test) comparing the fitness obtained by each run to the sample data provided in your repository, and a brief discussion interpreting the results of the statistical tests. This sample data can be found in `data/mysteryAlgorithmResults.txt`.

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 x66y Fall 2021 Assignment 1a", where x and y need 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!)

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 August 29, 2021.

Grading
The point distribution is as follows:

Assessment Rubric \ Deliverable Category	Green
Algorithmic	30%
Logging and output files	20%
Good programming practices including code reliability and commenting	10%
Document containing evals versus fitness plots	20%
Statistical analysis	20%

Assignment 1b: Evolutionary Algorithm Search

You must implement an EA which generates map layouts for a game of GPac. This assignment will utilize the same framework that was utilized for Assignment 1a, and builds on some of the code you produced in that assignment. Treating the game of GPac as a black-box problem, your EA must generate maps and use their evaluated fitness to search for higher-fitness solutions.

In this assignment you are asked to complete the Jupyter notebook `1b_notebook.ipynb`, several functions outside the notebook which will be reused in later assignments, and a report. The notebook will guide you through implementation of your EA, and will explain how to perform the experiments necessary to create the report. While completing the notebook, think about what data you will need to record in order to write the report described below.

Once you've finished the notebook, you need to write a report. This report should include an evals-vs-fitness plot showing the progress of evolution averaged over 30 runs. This report should also include statistical analysis (F-test and t-test) comparing the best fitness obtained by each run to data generated by the algorithm you implemented during 1a, and a brief discussion interpreting the results of the statistical tests. The report must also include every configured parameter used in your experiment.

This assignment also includes a unit testing suite provided for your benefit. The unit tests can be run by executing *pytest* from a command line from the top-level directory of your repository (i.e., not inside a folder in the repository, and not outside the repository). When a unit test fails, you are expected to perform basic troubleshooting before trying to contact a TA. The unit tests are designed so you can determine the cause of failure and remedy it on your own. Note that, due to the stochastic nature of EAs, the unit tests have a small chance of creating false failures or false passes. When in doubt, run them several times to be certain of their accuracy. **Passing these unit tests is part of your assignment grade** – your submission must pass all tests for full points (we will run them several times to ensure accurate testing). You may not make any changes to the unit tests that changes their outcome (pass/fail). You may, however, include things such as *print* statements in them (then run *pytest -rx* to see their output), **but you must mention this in your report**.

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 x66y Fall 2021 Assignment 1b", where x and y need 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 should include instructions on how to interpret your data, then you should!)

YELLOW 1 Up to 10% (bonus points for COMP 5660 students) for an implementation of Stochastic Universal Sampling (see Figure 5.2, page 84 in the textbook) with accompanying report and statistical analysis comparing the impact on performance against another parent selection technique.

RED 1 an implementation of a recombination algorithm which takes the geometry/shape of the board into consideration. What does that mean? You tell us! Any correct algorithm that reasonably fits this definition can be submitted. Describe your design in your report and provide statistical analysis comparing performance against another recombination operator. Students can earn up to 15% for this investigation.

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 September 12, 2021.

Grading
The point distribution is as follows:

Assessment Rubric \ Deliverable Category	Green	Yellow	Red
Algorithmic	40%	60%	60%
Passing unit tests	15%	0%	0%
Logging and output files	5%	5%	5%
Good programming practices including code reliability and commenting	10%	10%	10%
Report	15%	10%	10%
Statistical analysis	15%	15%	15%

Assignment 1c: Constraint Satisfaction and Self-Adaptation

Using the EA you created in Assignment 1b, you must implement two new EAs. The first new EA is a constraint satisfaction EA, and will add a penalty function to the fitness of each individual. The second new EA uses self-adaptivity to evolve the mutation rate of each individual.

In this assignment you are asked to complete the Jupyter notebooks `1c0_notebook.ipynb` and `1c1_notebook.ipynb`, several functions outside the notebooks, and a report. The notebooks will guide you through implementation of your EAs, and will explain how to perform the experiments necessary to create the report. While completing the notebooks, think about what data you will need to record in order to write the report described below.

Once you've finished the notebooks, you need to write a report. This report should include, for each of the new EAs, an evals-vs-fitness plot showing the progress of evolution averaged over 30 runs. This report should also include statistical analysis (F-test and t-test) comparing the best fitness obtained by each run to data generated by your EA from 1b, as well as to each other (for a total of 3 comparisons), and a brief discussion interpreting the results of the statistical tests. Note that for statistical analysis with the constraint satisfaction EA, you should use the best raw fitness. The report must also include every configured parameter used in your experiment.

This assignment also includes a unit testing suite provided for your benefit. The unit tests can be run by executing *pytest* from a command line from the top-level directory of your repository (i.e., not inside a folder in the repository, and not outside the repository). When a unit test fails, you are expected to perform basic troubleshooting before contacting a TA. The unit tests are designed so that you can determine the cause of failure and remedy it on your own. Note that, due to the stochastic nature of EAs, the unit tests have a small chance of creating false failures or false passes. When in doubt, run them several times to be more certain of their accuracy. **Passing these unit tests is part of your assignment grade** – your submission must pass all tests for full points (we will run them several times to ensure accurate testing). You may not make any changes to the unit tests that changes their outcome (pass/fail). You may, however, include things such as *print* statements in them (then run *pytest -rx* to see their output), **but you must mention this in your report**.

The deliverables of this assignment are:

GREEN 1 your source code and completed notebooks

GREEN 2 a PDF document headed by your name, AU E-mail address, and the string "COMP x66y Fall 2021 Assignment 1c", where x and y need 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!)

YELLOW 1 Up to 10% (bonus points for COMP 5660 students) for making the penalty coefficient self-adaptive. Using notebook `1c0_notebook.ipynb` and the self-adaptive genotype class implemented during the completion of `1c1_notebook.ipynb`, create an updated population evaluation function that considers a self-adaptive penalty coefficient. Evaluate your implementation using the multi-run experiment cells of notebook `1c0_notebook.ipynb` and the provided config file `yellow1c_config.txt`. Implement the full EA, then create a plot similar to the plots from the GREEN deliverables. Finally, perform statistical analysis comparing the best raw fitness of each run from this EA to the GREEN constraint satisfaction EA. Place these in a separate section in your report and be sure not to overwrite the notebook functionality required for your GREEN deliverables!

RED 1 Up to 10% bonus for implementing an adaptive mutation rate using a heuristic of your choice. Using the code in the notebooks as a guide, implement this new EA in a new file named `adaptiveEA.py`. Implement the full EA, then create a plot similar to the plots from the GREEN deliverables. Finally,

perform statistical analysis comparing the best fitness of each run from this EA to the GREEN self-adaptive EA. Place these in a separate section in your report.

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 September 26, 2021.

Grading
The point distribution is as follows:

Assessment Rubric \ Deliverable Category	Green	Yellow	Red
Algorithmic comprehension	45%	60%	60%
Logging and output files	5%	5%	5%
Good programming including code reliability, commenting, correctness, & practices	20%	10%	10%
Report	15%	10%	10%
Statistical analysis	15%	15%	15%

Assignment 1d: Multi-Objective EA

Using the EA you created in assignments 1b and 1c, you must implement a multi-objective EA.

In this assignment you are asked to complete the Jupyter notebook `1d_notebook.ipynb`, several functions outside the notebook, and a report. The notebook will guide you through implementation of your EA, and will explain how to perform the experiments necessary to create the report. While completing the notebook, think about what data you will need to record in order to write the report described below.

Once you've finished the notebook, you need to write a report. This report should include an evals-vs-fitness plot showing the progress of evolution for each objective averaged over 30 runs. This report should also include a plot of the best Pareto front found in any run, where we count Pareto front P1 as better than Pareto front P2 if the proportion of solutions in P1 which dominate at least one solution in P2 is larger than the proportion of solutions in P2 which dominate at least one solution in P1. Visualize the maps of the individuals within the best Pareto front and comment on how they differ. The report must also include every configured parameter used in your experiment.

The deliverables of this assignment are:

GREEN 1 your source code and completed notebooks

GREEN 2 a PDF document headed by your name, AU E-mail address, and the string "COMP x66y Fall 2021 Assignment 1d", where x and y need 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!)

YELLOW 1 Up to 10% (bonus points for COMP 5660 students) for implementing crowding as a diversity preservation mechanism. In order to maintain compatibility with the MOEA implementation prescribed within the notebook, crowding should be implemented such that a value [0,1) is added to all individuals in a level of non-domination proportional to that individual's crowding distance relative to others in their non-domination level. Using a diversity metric of your choice, record the diversity of the Pareto front from the last generation of each run and perform statistical analysis (including f-tests and t-tests) to compare against diversity observed in your GREEN deliverables. Plot the best Pareto front with crowding and compare with the front plotted from the GREEN deliverables to informally identify which algorithm performed better.

RED 1 Up to 10% bonus for implementing constraint satisfaction by casting it as a multi-objective problem that considers negative average pac-man score and negative repair count as the two maximization objectives. Perform statistical analysis (using f-tests and t-tests) to compare performance with your Assignment 1c constraint satisfaction implementation using negative average pac-man score (a.k.a. raw fitness). Report on your findings as appropriate.

RED 2 Up to 10% bonus for implementing and experimenting with 3-objective search using negative average pac-man score, negative dead end count, and negative repair count. Find the best 3-objective Pareto front generated by a 30-run experiment using the criteria described earlier. Include in your report a 3D plot of this Pareto front. Using the individuals in this Pareto front and non-domination sort, form 2-objective Pareto fronts considering only the objectives shared with GREEN/YELLOW and RED 1, respectively, and compare against the best Pareto fronts from those deliverables through informal analysis of Pareto front plots. Include these plots and your comparison in your report.

RED 3 We alluded in the notebook that rigorous analysis of multi-objective results is complicated! For up to 15% bonus, perform a rigorous comparison of the two MOEA configurations of your choice (that use the same objectives) using the hypervolume indicator metric as calculated with software available at the following link: `http://lopez-ibanez.eu/hypervolume` . Include in your report a detailed description of the analysis technique as well as your results.

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 10, 2021.

Grading
The point distribution is as follows:

Assessment Rubric \ Deliverable Category	Green	Yellow	Red 1	Red 2	Red 3
Algorithmic comprehension	45%	50%	50%	20%	0%
Logging and output files	10%	5%	5%	5%	5%
Good programming including code reliability, commenting, correctness, & practices	20%	10%	10%	10%	10%
Report	25%	20%	15%	65%	55%
Statistical analysis	0%	15%	20%	0%	30%

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`