Chapter 3 - Planning in GVGAI

Exercises

The GVGAI Framework is available in a Github repository[1]. Use the release 2.3[2] in order to run the same version presented in this chapter.[3]

1 Monte Carlo Tree Search

1.1 Tweaking MCTS

Execute the sample MCTS agent in the single-player planning setting and tweak its parameters to observe changes in performance.

– The code of this agent is in the package `tracks.singlePlayer.advanced.sampleMCTS`. In this package, `TreeNode.java` has several parameters that can be tweaked, such as the rollout length or the value of C in the UCB1 Equation.
– The function to evaluate states at the end of a Monte Carlo simulation is `value()`. Try to create different value functions that can achieve better results than the default one (spoiler alert: it's not easy!).

1.2 Working with Knowledge-Based Monte Carlo Tree Search

The following ideas could be taken as exercises or research projects.

– Rather than using Euclidean distances for calculating the distances to sprites, implement and use a path-finding technique instead. Distances will be more accurate at the expense of computational cost. How could you maximize the impact of the former and minimize that of the latter?
– Substitute the simple $1 + 1$-EA algorithm used to evolve the weights during the Monte Carlo simulation for a more involved evolutionary technique.
– Can you think of better features (than distances to sprites) that become better guides for KB-MCTS? Maybe some convolutional filters over the space around the avatar or the whole game state?
– Expand the experiments done with KB-MCTS to the other game sets of the GVGAI framework.

[1] `https://github.com/GAIGResearch/GVGAI`
[2] `https://github.com/GAIGResearch/GVGAI/releases/tag/2.3`
[3] These exercises are also available at this book's website: `https://gaigresearch.github.io/gvgaibook/`

1.3 Expanding the Multi-Objective Monte Carlo Tree Search

The following ideas could be taken as exercises or research projects.

- All approaches run in the experiments described in Section ?? use a determined set of fixed weights. You can experiment to see what happens if these weights are different. Could you find a way to dynamically change the weights in response to the state of the game and improve the performance?
- The mixed strategy approach showed some promising results and it is possible that a high performance can be obtained if the policy for switching objectives is optimized. How would you do this? A possibility is to add game and agent related features (see Chapter 4) to this policy for objective decision-making.
- Explore other multi-objective optimization approaches. For instance, the epsilon-constrained approach [?], which considers one of the objective functions as a constraint ($O_i < \epsilon$, in case of maximization) and only the other function is subject to optimization.

2 Rolling Horizon Evolutionary Algorithms

2.1 Tweaking RHEA

Execute the sample RHEA agent in the single-player planning setting and tweak its parameters to observe changes in performance.

- The code of this agent is in the package `tracks.singlePlayer.advanced.sampleRHEA`. In this package, `Agent.java` has several parameters that can be tweaked, such as the population size, individual lengths and evolutionary operators..
- The function to evaluate an individual is `evaluate()`. Try to create different value functions that can achieve better results than the default one.

2.2 Working with Rolling Horizon Evolutionary Algorithms

The following ideas could be taken as exercises or research projects.

- Tweak the parameters of the Dynamic Rollout Length adjustment. Can this be change dynamically in response to features of the game state?
- What other algorithms could provide good initial solutions to seed the initial population of RHEA? What is the effect of changing the budget for the algorithm that seeds the population (i.e. what happens if the allotted budget is less than 50% of the total time for MCTS?

- Also on RHEA seeding, you can investigate how much should the solution provided by seeding algorithms be disturbed in order to form the initial population.
- Try improving the bandit based mutation. This mutation is uni-variate (only one gene is considered at a time), but it is likely that higher-dimensional tuples provide better results if they are able to identify the relationships between the genes in the sequence.