Extensive Auction Games

CS404 Agent-based Systems Coursework

1 Introduction

Imagine an auction of paintings by four famous artists: Picasso, Van Gogh, Rembrandt and Da Vinci. In the auction room the auctioneer presents each piece to be sold, and all bidders then write their bids for the item on a secret sealed note that is handed to the auctioneer. The auctioneer then declares the highest bidder the winner, takes a payment (equal to their bid or to that of the second-highest bidder, depending on the auction), and starts the next round with a new item. If there is a target collection of paintings to buy, the first bidder to buy this bundle of goods terminates the auction and wins. Otherwise, the auction continues until there are no more items to sell. If there is no target bundle, the bidder with the paintings of the highest total value is the winner.

The objective is to implement strategies for a Python 3 bidding Bot that will participate in this game.

2 The Game

This is an extensive game with simultaneous moves encoded as an auction played sequentially. The game runs in the following way. An Auctioneer initialises a room full of bots, then sets up the auction. This involves giving each bot the same budget to spend, setting the sequence of items to be sold, and setting the rules of the auction such as how the overall winner of the game will be decided.

The Auctioneer will then announce an item type to be bid upon, and ask the bots for bids. Your Bot will use the information that the Auctioneer gives, including the state of the other players thus far, to determine an amount to bid. Once all bots have bid, the Auctioneer will declare the highest bidder the winner, who will then be charged a payment (not necessarily the amount they bid, see below) and receives the item. If the top bids draw, then the winner is chosen at random from those bidders. For each round there is at most one winner.

The auction will continue until either there are no more items to sell or if there is a set winning condition, e.g., a bidder managed to acquire the target collection of artists' paintings needed, in which case they are declared the winner. If there is no target collection as a winning condition then the auction will end once all the paintings are sold, and the bidder who ends with the highest total value of items is the winner.

Note that, however, whilst the highest bidder will always win, the auction may be set up so the highest bidder does not pay their own bid. It can be set up so that the highest bidder is only charged the second-highest bid (see `winner_pays`).

You will write your strategies in your Bot. Your Bot will be tested in a series of different auctions against Bots of varying difficulty and number. Finally, your Bots will be tested against each other in a tournament.

There are two types of games that will be played:

- Value Game:

  Highest total value at the end wins, the **highest bidder pays the second-highest bid**. Picasso is worth 15, Van Gogh worth 5, Rembrandt worth 3 and Da Vinci worth 1.
- Collection Game:

  First to **collect a given bundle** of painting types, the **highest bidder pays their own bid**. The collection bundle minimal condition will be 3 paintings of any artist, 3 of another artist, 1 of another artist and 1 of another artist. For example, 3 Van Gogh, 3 Picasso, 1 Rembrandt and 1 Da Vinci; or 3 Da Vinci, 3 Rembrandt, 1 Picasso and 1 Van Gogh.

Note that for all games in this tournament, the auction size (number of paintings up for auction) will be 200 and the starting budget for every bidder is 1001.

3 Implementation

Provided to you are two main Python 3 files to run the auctions: `auctioneer.py` and `arena.py`. We also provide you with some example bots in the bots folder, `random_bot.py`, `flat_bot_10.py` and `u1234321.py`.

3.1 auctioneer.py

`auctioneer.py` contains the definition for the Auctioneer class, which sets up and runs the auction. We will use this exact same file while marking your bots, so DO NOT CHANGE ANY OF THE CODE IN THIS FILE.

It has the following arguments:

- `game_type`(string): Either "value" or "collection" for the 2 game types. Value games are won by the bot with the highest value of paintings after all rounds. Collection games are won when a bot manages to collect a full set of paintings as specified in the `target_collection`.
- `room`(list of modules): A "room" is a list of bots that will play the auction. The bots are module objects. There is an example of how to import them at the top of the `auctioneer.py` file, and how to pass them to the Auctioneer as a list at the bottom of the `auctioneer.py` file. There are also examples in the `arena.py` file.
- `painting_order`(list of strings): A list of the sequence of painting types that will be auctioned in each of the rounds. If this is set to None then a random order will be used.
- `target_collection`(list of integers): For collection type games, the winning bot is the first to win a collection of paintings. This is given as a list. For example, [3,3,1,1] means that the winner is the first bot to collect at least 3 paintings of one artist, 3 of another artist, 1 of another artist and 1 of another artist. The specific artist is not important. So 3 Da Vincis, 3 Picassos, 1 Van Gogh and 1 Rembrandt would win, as would 3 Rembrandts, 3 Van Goghs, 1 Picasso and 1 Da Vinci. Note: the strategies will be evaluated against a target collection of [3,3,1,1].
- `slowdown`(float): How long to wait at each round of the auction. If you set this to zero the auctions will run fast.
- `verbose`(boolean): Whether the auctioneer prints updates to the terminal or not.
- `output_csv_file`(string): The auctioneer automatically logs the result of every round in an auction. This defaults to 'data/auctioneer_log.csv', but you can specify a different filename.

When the auctioneer is initialised it will automatically set up the auction and all the bots in the room with the `initialise_bots()` method.

The method `run_auction()` will run the auction until it is completed.

It is not necessary to know how the Auctioneer class works to do well on the coursework. But if you are interested there are more details in the `README.md` file.

3.2 arena.py

The `arena.py` file is provided as a convenient way to run auctions. This includes some methods that show examples of how to run auctions. This is given to you as an example, so please feel free to change any of the code here and experiment.

You might want to run an auction slowly, with a full print out to the terminal of the auction's progress, as shown in `run_basic_auction()`. This will be useful to see how your bot is performing live. Or you might want to run lots of auctions quickly, as shown in `run_lots_of_auctions()`.

3.3 bots

In the bots folder we included a few bots for you to practice with. We have given you 3 bots. `bots/u1234321.py` is a good starting point for your own bot, with everything commented clearly. `bots/flat_bot_10.py` and `bots/random_bot.py` are example bots that should be easy to beat.

3.4 bots/u1234321.py

`u1234321.py` is an example bot that you can use as a starting point to write your own bot.

Each bot is a class, with two main methods, `get_bid_for_collection_game()` and `get_bid_for_value_game()`, that allow you to make bids on the collection and value game types. it is in these methods that you should implement your strategies for the two game types. The comments on the methods explain what each argument is. Currently, the strategies both just return a random integer between 0 and the amount of budget that your bot has left.

Your bot will be initialised at the start of the auction, and then will play until the auction finishes. This means that your bot can hold internal state variables during the auction, which may be useful for some strategies. You can add these variables, or any other code, in the `__init__()` method, if you want to. Please change the self.name="1234321" variable to your own ID number.

Feel free to add any extra methods that you might need, but be careful that the two main methods keep the same input variables and return integer bids without crashing.

If your bot crashes then your bid will be set to zero. If your bot returns a float then this will be rounded down to an integer. If your bot bids more than their budget then their bid will be set to zero. If your bot takes longer than 5 seconds to return a bid, then again your bid will be set to zero.

3.5 bots/random_bot.py and bots/flat_bot_10.py

We've included some simple bots for you to play against.

- `random_bot.py`: Bids a random integer between 0 and the bot's remaining budget.
- `flat_bot_10.py`: Bids 10 on everything.

You can add your own. You can see how to import bots and play against them in the `arena.py` file.

4 Submission

Your coursework submission will consist of a single compressed file (either .zip or .tgz) containing your bot in a python file and your writeup as a pdf. The coursework file should be submitted through Tabula.

4.1 Your Bot

Please save this as `u<YOUR WARWICK ID NUMBER>.py`. The main class should be called Bot, and in the `__init__()` method you should set `self.name = <YOUR WARWICK ID NUMBER>`. Here is an example:

`u1867321.py`

```
class Bot(object):

    def __init__(self):
        self.name = 1867321

    ....
```

You should also include the methods for playing each game type. You can use `bots/u1234321.py` as a starting point.

Here is an example of how your bot will be imported and run:


```
from auctioneer import Auctioneer
from bots import random_bot
from bots import u1867321 # Your id number here
```



```
room = [random_bot, u1867321]
value_game = Auctioneer(room=room, game_type="value", slowdown=0)
winner = value_game.run_auction()
print("The winner is ", winner)
```

Please make sure that your bot will run in this way, replacing the id number here with your own id number. If you run the above script you should see the auction run very fast and then print out "The winner is [1867321]", with your id number shown (if your strategy can beat a `random_bot`).

You can test that your bot runs correctly by using the functions provided in the arena.py file. If your bot crashes you will lose marks!

4.2 Your Writeup

Please save this as <YOUR WARWICK ID NUMBER>.pdf. Alongside the code submission, you are required to write a pdf report, of **at most** three pages including references, of the theory to support your strategies. It should explain the reasoning behind your strategy design including evidence of relevant theory and/or testing. The pdf should be written in IEEE two-column conference format. You are free to design and think about your strategies in your own unique way, and we encourage you to use what you have learnt from the course.

5 Evaluation

The coursework is worth 50% of the module credit. Its marking scheme is structured as follows:

- $\frac{1}{3}$ of the total mark: Strategy performance for the 'Value Game' (against various bots including other submissions).
- $\frac{1}{3}$ of the total mark: Strategy performance for the 'Collection Game' (against various bots including other submissions).
- $\frac{1}{3}$ of the total mark: Quality of the written report (how well you explain and analyse your strategies).

For each game type, your bot will be tested in a range of different auction "rooms", made up of bots of different difficulties, in different room sizes and compositions. For example, you will be tested in a room against one `random_bot`. You will also be tested in a room with a mix of up to 10 bots with different strategies. For collection games, you will also be tested against different `painting_orders`. Your bot's performance in this set of rooms will form the majority of your mark for strategy performance.

We will also play the student bots against each other, and this will contribute to your mark for strategy performance.

6 Cheating/Plagiarism

This coursework is your own work. Group submissions are not allowed. All submissions will be put through plagiarism detection software which compares against a number of sources, including other submissions for CS404, submissions at other universities, web sources, conference papers, journals and books. Please see the student handbook for more information, or ask if you need guidance.

In addition, please note that any attempt to access other bots, modify the auctioneer class or otherwise intervene in the fair running of the auctions will be considered cheating and sanctioned accordingly.

Acknowledgements

The code was developed by Charlie Pilgrim (`charlie.pilgrim@warwick.ac.uk`), Department of Mathematics, University of Warwick. A previous version of the coursework, from which this takes inspiration, was written by Alexander Carver, Department of Computing, Imperial College London. Further precious input for the coursework came from Charlotte Roman, Department of Mathematics, University of Warwick.