Computer Games (2023-2024): No Thanks!

Instructions

On these pages you find an assignment for *Computer Games*. This exercise concerns the writing of an artificial intelligence (AI) for a simple card game. This game is known under different names, such as *Geschenkt... ist noch zu teuer!* and *No Thanks!*

When everybody has submitted their AI, we will run a competition, to see which AI is best able to play the game. The competition will be a simulation of at least 10,000 games, but it will probably be more (depending on how long the competition will take; we may even run 1 million games). It is therefore necessary that your AI runs pretty fast. An AI that is slow will be disqualified. To assess whether an AI is fast enough: if you run it against the Beginner AIs for 1000 matches (the default), it should not take more than a few seconds.

The maximum number of points that you can score for this exercise is 10. To get a high number of points, make sure that you implement an interesting algorithm, and write good Python code. An AI that makes the competition crash is disqualified, so test well. Please make sure that your AI does not produce any printed output. There is no need to write "winning" code (though it cannot hurt, of course). If you create several different AIs, please submit only one for the competition, but you can also submit the others to the instructor, explaining why you prefer the one you submitted over the others (probably because it seems to be the strongest).

When you have completed your AI, submit it via Canvas. Your AI should consist of a single Python file. You can make use of standard Python libraries, but not anything else. If you want to use something beyond this, first discuss it with the instructor. Make sure that you have included your name and student number in your AI's header. The deadline for submission is the Friday before the last lecture.

You submission should be accompanied by a short report (1-2 pages suffices, please use a PDF or Word file) which explains your tactical approach to the game, different approaches you may have considered, and how you tested. If you used particular tools to help the development, or inspiration from literature, please list that also in your report. It is allowed, but needs to be acknowledged.

No Thanks! the Game

No Thanks! supports 2 to 5 players; considering the number of students in the course, we will use the 5-player version (though the software supports fewer players). Your AI will always be pitted against a random selection of opponents in every game, and you will meet every opponent about equally often.

The 5-player version is played with a deck of 33 cards, which are numbered 3 to 35. At the start of each game, 9 random cards are removed from the deck, so 24 cards will be played during the game. Each player starts the game with 11 coins, and the 24 cards are shuffled in a face-down pile.

The players are seated around the table, and one player is the current player (randomly assigned at the start of the game). If there is no card open on the table, then the current player takes the top card from the pile and places it open on the table.

The current player now makes a decision: either they take the card and all coins that are on the card (which is none when the card is initially placed open on the table), or they refuse to take the card. If they refuse to take the card, then they must put one of their coins on the card, and the next player, clockwise around the table, becomes the current player.

Thus a card remains open on the table, and collects coins, until a player is willing to take it with all the coins on it, or the current player has no coins left and thus has no choice but to take the card.

Note that when a player takes the card with the coins on it, they remain the current player and thus turn over the next card from the pile.

Once the pile is empty, the game is over and total penalties are calculated. This goes as follows:

- Each card that a player has bears a penalty equal to the number on the card; the penalties for the cards are added up.
- Exception: A card for which a player ALSO has the card that is 1 point lower, bears no penalty. Thus, if a player has the 33 and the 35, that is a total penalty of 68 points, but if a player has the 33, 34, and 35, that is a total penalty of 33 points.
- The total penalty of a player is lowered by 1 point for each coin that the player has at the end of the game.

Of course, the lower the total penalty that a player has for the game, the better it is.

For example, suppose that a player ends the game with 13 coins and the cards 3, 7, 8, 10, 11, 12, and 25. Their total penalty is 3 + 7 + 10 + 25 – 13 = 32. The 8, 11, and 12 bear no penalty.

Running a Competition

To run a competition, start the file `Take5.py`. If you do that from the command line, you can give command line parameters:

```
python –u NoThanks.py 10000 aifile1 aifile2 ...
```

The first command line parameter (10000 in the example above) is the number of games that should be played in the competition. After that you can specify a list of python files that contain AIs that you want to enter into the competition. If you do not specify any command line parameters, the number of games that will be played is 1000, and the AIs are loaded from the file `NoThanksBeginners.py`.

If you want to run the competition from the editor, you can (temporarily) change the values for `DEFAULTLENGTH` and `DEFAULTAIS` at the top of the `NoThanks.py`

file. DEFAULTLENGTH indicates how many games you want to run, and DEFAULTAIS is a list of all the AI files in the competition. If you change it, for instance, to ['NoThanksBeginners','MyAI'], it will load all the AIs from NoThanksBeginners.py and from MyAI.py.

At the end of the competition, the game prints an ordered list of all the AIs that entered, with a number that indicates in how many games they were, and their average score. The lower the average score, the better it is.

How to write an AI

To write an AI, create a new Python file, and create a new class in it. This class should inherit its features from Player, which is defined in NoThanksPlayer.py. You should also create an __init__ method that calls the __init__ method from Player, and uses the setName method to give itself a name. So, suppose that your new AI is called MyAI, then you will write in your file (probably called MyAI.py) the following code:

```
from Take5Player import Player

class MyAI(Player):
     def __init__( self ):
          Player.__init__( self )
          self.setName( 'MyAI' )
```

What you now have to do is create at least one methods, namely take. take is used to decide whether or not to take a card. If you do not create this method, it will be inherited from Player, which means that it will always take a card.

take should be defined as follows:

```
     def take( self, card, state ):
```

In this definition, card is an object of the type Card, which is the card that is on offer. state is a special object of the type State, that provides an overview of everything known about the current game: in particular, it has a list of the players (<state>.players) in the game. For each player, the state lists the player's name (<player>.name), the coins that they have (<player>.coins), and a list of the cards they have collected (<player>.collection) ordered from low to high. The players are listed in the order in which they are seated around the table, with the current player as the first in the list. The state object also has an attribute round which indicates the round number (1 to 24); thus the size of the hidden pile is 24 minus the round number.

take must return True or False. True means that the player will take the card and all the coins on it. False means that the player will put a coin on the card and will let the next player make a decision. If a player has no coins, take will not be called at all, but the player will automatically take the card and coins on it (note that

the AIs in `NoThanksBeginners.py` check whether the player has coins and return `True` when there are no coins, but this is not actually necessary to include).

For examples, look in the file `Take5Beginners.py`. It contains multiple simple AIs. You are allowed to use those as a starting point for your own AI.

If you want to keep track of some variables over different games, you can do that. The AI is only created once for the competition. You should then initialize the attributes that you want to use in the `__init__` method. Moreover, if you want to keep track of information during a game that needs to be initialized per game, you can use the `startGame` method to do that. This method is called at the start of each game, without parameters. At the end of each game, the `endGame` method is called for every player in the game. As parameter it gets a `State` object. Neither the `startGame` nor the `endGame` method does anything if you do not implement them. These methods are mainly useful if you want to implement a learning AI.

Cheating

If you want to cheat in this game, that is very easy. Since you have access to all the cards, and all the other players if you want, you can just delete your penalty points, adapt your score, adapt the scores from other players, etc. This is, in fact, so easy that I have not taken the trouble of trying to prevent it in code. Such cheating is actually impossible to prevent (I could make it harder, but I can never prevent it, and I want you to be able to see what the code does anyway). Instead, I am going to tell you straight out: you are not allowed to cheat. If you cheat, you are disqualified. I will definitely examine everybody's code, with special attention to the code of those who score well.

What is your AI allowed to do? You can look at your own cards. You can look at your penalty points. You can look at the information in the `State` object, which includes all the cards that the other players have collected and their coins. Other than making changes to attributes that you create for your own AI, you are only allowed to effectuate changes by returning a value from the `take` method. Other changes are not allowed. You never need to make changes to cards or card collections or coins; that is all handled by the engine. You are not allowed to look at information that is secret during the game, such as the cards still hidden in the pile.

Helpful objects and methods

I have provided some helpful objects and methods that you can use when you write your code. If you want even more details on them, look in the source codes provided.

A `Card` object has three attributes: the `number`, the `penalty`, and the `coins`. The number and penalty are always the same (I included both for future variants). The number of coins on the card is indicated by the `coins` attribute. It only has meaning for the card that is given to the `take` method.

The `Player` has a method `penalty()`, which calculates the penalty that the player has with their current collection of cards and current coins. The method `penaltyWhenTake()` gets a card and calculates the penalty that the player gets

when they add that card to their collection. This includes the number of coins that are on the card. Thus you can calculate the change in penalty by using these two methods.

If you can think of other methods or attributes that could provide you with useful information or useful abilities, please let me know; I will see if I can add them.

Testing

To test the capabilities of your AI, you will probably first try it out against the AIs in `NoThanksBeginners.py`. Know, however, that these AIs are rather weak. You should be able to create an AI that plays a better game. Also note that it depends on the tactics of the other players how good your AI is. For instance, if you are playing against cautious players who have a high tendency to put coins on cards, you don't need to immediately take a card that has no penalty for you: letting it go around once more may allow you to collect more coins.

You might use an approach where you create several different AIs, and try to improve them while testing them against earlier incarnations of your AIs. You might also test your AI against those of your fellow students, if they are willing to engage in such activities. I encourage you to do that. For those who like a real challenge, implementing an adaptive AI that learns better behaviour over time might be a cool project (even if the results are meagre, I would really appreciate the attempt, and it may give you a better grade for the assignment).

During your tests, you may want to print out certain objects, such as the `State` objects. Some have suitable `__str__` methods encoded. However, make sure that you remove those prints again before submitting your final code.

Competition Time

I will run a live during the last lecture. You are encouraged to be there. I would like those who score well to explain briefly how they approached the AI to the other students (there is no need to prepare this).

Further Information

If you have more questions, please contact the instructor.