Computer Games (2023-2024): Take 5!

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 *6 Nimmt!*, *Take 5!*, and *Category 5* (confusingly, there are at least three different games with the name *Take 5!*, one of them very similar to the game that we are using.)

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.

Take 5! the Game

Take 5! supports 2 to 10 players; considering the number of students in the course, we will use the 10-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 10-player version is played with a deck of 104 cards, which are numbered 1 to 104 (if there are less than 10 players, the highest-numbered cards are removed – there will always be 10 cards for each player plus 4 cards in the deck). Each card has a number of penalty points:

- Every card has at least 1 penalty point;
- If its number ends in 5, it has 2 penalty points;
- If its number ends in 0, it has 3 penalty points;
- If its number is a multiple of 11, it has 5 penalty points;
- The “worst” card is 55, which both ends in 5 and is a multiple of 11: this one has 7 penalty points.

Each player is dealt 10 random cards. On the table there are 4 rows of cards. The rows are started with the 4 cards that were remaining after the players received their hands.

The game is played over 10 rounds. Each round, each player takes one card from his hand. These cards are shown simultaneously. Then, in increasing order (low to high), they are placed in the rows. The three rules for placement are as follows:

- If the card has a number that is lower than the highest card in each of the rows, then the player who played this card has to take one of the rows and puts it in his personal penalty pool; the card played is then used to start a new row;
- Otherwise the card must be placed in the row of which the highest card is lower than the number of the played card, but closest to it;
- If that row already contains 5 cards, the player has to take the whole row and places it in their personal penalty pool; the card played is used to start a new row.

For example, suppose that these are the 4 rows:

- 2, 5, 9, 23, 34
- 17, 18, 22
- 76
- 10, 11, 12, 103

Suppose that the following 3 cards must be placed in rows: 4, 35, 80. Being the lowest card, the 4 must be placed first. Since 4 is lower than the highest card in each of the rows, the player who played the 4 must take one of the rows. The 4 then replaces that row. Since the third row contains the least penalty points (namely just 1), we assume that the third row is taken. The situation becomes:

- 2, 5, 9, 23, 34
- 17, 18, 22
- 4
- 10, 11, 12, 103

Then the 35 must be placed in one of the rows. 35 is higher than the top card of the first three rows, but since it is closest to 34, it must go to the first row. Since the first row already has 5 cards, the player who played the 35 must take that whole row, and the 35 starts a new row. The situation becomes:

- 35
- 17, 18, 22
- 4
- 10, 11, 12, 103

Finally, the 80 must be placed. Again, this card would fit in the first 3 rows, but since it is closest to 35, it goes in the first row. The situation then is:

- 35, 80
- 17, 18, 22
- 4
- 10, 11, 12, 103

At the end of the game, each player adds up all their penalty points. The lower the penalty points, the better. Winner of the game is the person who achieves the lowest average score after all the games have been played.

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 Take5.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 `Take5Beginners.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 `Take5.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 `['Take5Beginners','MyAI']`, it will load all the AIs from `Take5Beginners.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 `Take5Player.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 two methods, namely `playCard` and `chooseRow`. `playCard` is used to play a card from your hand. `chooseRow` is used to select a row to take when your card is lower than the top cards in any of the rows. If you do not create these methods, they will be inherited from `Player`, which means that they will play a random card and choose a random row.

`playCard` should be defined as follows:

```
def playCard( self, hand, rows, state ):
```

In this definition, `hand` is a list of objects of the type `Card`, which are the cards that you have in hand. `rows` is a list of four `Row` objects, each `Row` object specifying a row currently on the table, containing a list of cards. `state` is a special object of the type `State`, that provides an overview of everything known about the current game, such as the players in it, the cards they played last, the penalties they collected, etc. `playCard` must return one of the `Card` objects from `hand`. If it returns something different, a random card will be selected from the `hand`.

`chooseRow` should be defined as follows:

```
def chooseRow( self, rows, state ):
```

In this definition, `rows` and `state` are the same as specified for the `playCard` method. `chooseRow` must return 0, 1, 2, or 3, indicating the index of the row that it wants to take. If a number outside this range is returned, a random row is selected.

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. Other than making changes to attributes that you create for your own AI, you are only allowed to effectuate changes by returning values from the `playCard` and `chooseRow` methods. Other changes are not allowed. You never need to make changes to cards or rows yourself; that is all handled by the engine. You are not allowed to look at information that is secret to you during the game, i.e., the cards that the other players have in hand.

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 (which you find in the list that represents your hand, and in the rows) contains the following helpful method:

```
def goesToRow( self, rows ):
```

You call it with the `rows` list that is passed to `playCard` and `chooseRow`, and it will return the index of the row (0-3) in which that card would be placed if it is placed right now (in practice, it may end up in a different row, because lower cards played by other players may change the rows). You can find an example of a use of this method in the `AvoidPenalty` AI in the `Take5Beginners.py` file. The `Card` object also contains the attributes `number` and `penalty`, which refer to the card's number and penalty points, respectively.

A `Row` object contains a handy method `penalty()`, which returns the total penalty that this row gives. You can use it to select a row when you have to choose one. It also contains the `size()` method which returns the row's length. An example is found in the `LowPenalty` AI in the `Take5Beginners.py` file.

The `Player` object, which your AI inherits from, has a number of handy attributes and methods (since your AI inherited those, you can access them using `self`). `score()` gives your current average score over the games you have played. `penalty()` gives the total penalty that you have collected in the current game. The attribute `hand` contains the cards you currently have in hand, and the attribute `collection` contains the cards that you currently have in your penalty pool. `totalpenalty` is the sum of all the penalties you have collected over all the games, and `games` contains the number of games that you have played in.

The `State` object that is passed to `playCard` and `chooseRow` contains information on the current game. That information can be accessed through attributes. It entails the following: `round` is the current game round (1-10) and `rows` is the same as the `rows` that are passed to `playCard` and `chooseRow` (so it is redundant information, which I only included for completeness' sake). Moreover, the `State`

object contains a list called `players`. This is a list of `PlayerState` objects. Each of these `PlayerState` objects describes a player, and contains the following attributes: `name` for the player's name, `collection` for the list of the cards that the player collected in his penalty pool, `score` for his current average score, a boolean `ingame` (True/False) that indicates whether the player is in the current game, and `lastcard`, which is a `Card` object that indicates the last card that the player played (if there isn't one, it is `None`). You are allowed to use all the information in the `State` object in your AI.

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 `Take5Beginners.py`. Know, however, that these AIs are rather weak. You should be able to create an AI that plays a better game.

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.