Homework #3 – Duck hunting

DD2380: Artificial Intelligence

Contents

1 Introduction

In this **individual** assignment, you must implement a program that plays a duck hunting game (details in section 2). As in homework 2, you will receive a skeleton that takes care of things like communication with the server and game state representation, so that you can focus on the fun AI stuff: implementing a winning strategy.

> **Warning:** Read all the instructions carefully, since failing to follow them might lead to your submission being rejected

2 The game

"Duck Hunting" is a game for one or more (up to six) players.

The idea of the game is to shoot at certain ducks flying in the sky, and avoid shooting a duck species that is in danger of extinction. Therefore, this game involves three main (not independent!) topics:

- Predicting the movements of the birds in order to shoot them
- Identifying the birds species, to avoid shooting the endangered one (and maybe to help you with the movement prediction)
- Decide if it is beneficial or not to shoot given the confidence of your predictions.

We recommend that you focus first on predicting the movement of a single bird (practice mode), since doing that is a building block for the rest of the assignment, and it is enough for passing the homework (see Section 2.4.)

2.1 The sky

For the purpose of our game, the sky is flat. We can think of it as a wall. Ducks can move west or east along this wall, and up and down, but there is not a third dimension.

2.2 The ducks

There are 6 different species of ducks. All these species look quite similar, and are indistinguishable from a distance. However, they have **specific movement patterns** that will help you identifying them. Moreover, the color of plumage of the head is different for each species, so once you hit a duck, you (only the player who shot the duck) will be able to determine its species with complete certainty. The six colors are blue, yellow, red, green, white and black. Unfortunately, black ducks are critically endangered (to the point of only being one in the sky), and hunting them is forbidden. On the other hand, white ducks are extremely common (more than any other species) and thus not as "valuable". These facts are summarized in the following list, and should help you identifying white and black ducks. At the beginning of the game,

- there will be (in the default settings) 180 birds in the sky, each of them being of one of the 6 species.
- there will be exactly one black duck.
- half of the birds will be white.
- the other half will be divided evenly between the four normal species (blue, yellow, red and green). Because of rounding, some species might end up having one more bird than others

For example, in a game, there might be 1 black duck, 90 white, 23 yellow ducks, 22 red ducks, 22 green ducks and 22 blue ducks, for a total of 180 ducks.

All of the birds will be visible in the sky from the beginning, and until they are hunted down.

At any time, the bird will follow one out of four movement patterns:

migrating a bird that is migrating will always fly in the same direction (east or west), and its height will be more or less stable. Different species may migrate in different directions, but all birds of the same species will always migrate in the same direction.

quacking a bird that is quacking will tend to make erratic, slow moves, both in the horizonatal and vertical direction

panicking a bird that is panicking will tend to fly fast, both in the horizontal and vertical direction

feigning death a bird that is feigning death will tend to fall (go down vertically), and move very little horizontally.

Different species combine these patterns in different ways: there might be methodical species which rarely change their movement pattern and species which change often; some might frequently quack after migrating for a while, while others might panic instead. Unfortunately each hunting occurs in a different region, so you don't know anything about the species movement behavior at the beginning of each new round of hunting. You will have to learn (fast!) about it while hunting.

Each species will have **only 3 of these movement patterns** (which 3 varies for each hunt). The black bird and the white bird will have a different set of patterns from each other and from any other species, while the coloured birds may have the same set of patterns, but with a different probability or a different migrating direction.

In each turn (also called a time step), each bird performs one of the following actions, both for the horizontal and the vertical direction:

accelerate increase its speed. If it was moving east, it will continue moving east, but at a larger speed. If the bird was stopped, it can start moving east or west (up or down if the acceleration action is performed in the vertical direction).

stop completely stop (horizontally or vertically).

keep speed continue flying in the same direction, at the same speed. The bird cannot perform this action when it is stopped.

Any combination of horizontal and vertical actions is possible. For example, at a certain time step, a bird could accelerate vertically and stop horizontally.

The probability of the bird performing each of these actions will depend on what the bird is doing (migrating, quacking, panicking or feigning death) and the species of the bird.

At the beginning of the game, all birds are considered to be stopped, and each bird can start with any of the four movement patterns.

2.3 The players

The game can be played in three modes: Practice, single player and competition. **You should focus first on practice and single-player**, since the grading will mostly depend on how well your system performs in these modes (see section 5), and all the work you do to be good at this mode will be useful when you try to improve your performance in competition mode. A good performance in competition mode is required for full score but, in general, an agent that performs well in one mode will perform well in the other.

2.3.1 Practice mode

In this mode, there is a single bird in the sky, and you receive the actions the bird performed during the last 500 turns. Considering this, you must predict which action the bird will make next, both for the horizontal and vertical directions.

2.3.2 Single-player mode

In this mode, you can choose the number of birds, number of different species, whether an endangered bird appears, time available to perform each action and number of turns.

For the evaluation, fixed settings (as explained in section 5) will be used. However, it might be good for your testing to start with a simple setup that allows you to work in specific aspects of the problem, such as having a single bird (which would help you to get the prediction right).

In each turn, you will have a chance to shoot at one bird. To do that, you must choose a bird, and tell which action the bird will make, both in the horizontal and the vertical direction. If you guess the action, you will kill the duck. The player can also choose not to shoot at all, since missing shots subtracts points from your score (and insufficient knowledge about the species can make you kill the endangered black bird).

Whenever a bird is killed, it immediately disappears from the sky.

2.3.3 Competition mode

In this mode, more than one player "share" the same sky, and try to get as many points as possible. This is the mode that will be used in the online ranking system (and for evaluation of the highest scores), and the defult settings apply.

The game mechanics are very similar to the single-player mode, except that in each turn, only one of the players has a chance to shoot. In the next turn, only the next player can shoot, and so on. Once you are happy with your performance in single-player mode, you can think about how to exploit the differences of this mode.

In their turn, the players have the same options as in single player: shoot at a bird, or not shoot at all, and the same rules apply.

2.4 Scoring

killing the black bird -300 points

missing a shot -1 points

killing a white bird +3 points

killing a colorful bird +5 points

2.5 End of game

The game will end after 500 turns (in the default settings), independently of the number of birds that have been killed.

At game end, your client will be asked to identify the birds remaining in the sky, and you will get 1 extra point for each correct guess. Successful identification of the black bird gives an extra 25 points. If you have not killed any duck of a particular colorful species, it does not matter. You should assign them a color (of your choice) from the ones you have not seen, and the server will correctly interpret your answer. This does not apply to white and black birds.

Of course, the winner will be the one with the highest score. In the event of a tie, the following rules will be applied, in order, until the tie is broken:

- The one who killed the most birds wins.
- The one who guessed the most birds at the end wins.
- The one who killed the first bird wins.

If the tie is not broken by these rules, both players lose.

3 Assignment

Your assignment consists of writing a program that plays the "duck hunting" game.

However, you are not going to start from scratch. You will receive a skeleton in your language of choice, which will take care of communication with the game server, state representation, and calculating the movements each duck made at every time step. This way, you will only have to focus on implementing the strategy for deciding which is the best action to perform, where the core AI lies for this problem.

For playing, your agent will connect to the game server and the game will start. The game will proceed as follows:

Step 1 When it's your turn (it's always your turn in single-player mode), the server will send information about the state of the game (the actions of each bird in all turns until the current one, and the current score).

Step 2 Based on the current state (and all past states), your client will be asked to perform an action (shoot at a bird). Your client will have a limited amount of time (1 second in the default settings) to perform this step)

Step 3 Your client will send the action to the server.

Step 4 You will be notified of the outcome of your action.

Step 5 If the game has not ended (it ends when there are no more birds, or the turns have expired), the other players will perform their actions in the following turns, and then, when it is your turn again, it will continue from **Step 1**.

Step 6 Based on the positions of all birds in all time steps, your client will be asked to identify the birds that have not been killed.

Step 7 The server will report the final score.

In competition mode (which you cannot test by yourself, it will only be used after you submit your program), the server will create a sky according to the default settings mentioned in section 2.

In standalone mode (practice and single-player), on the other hand, you can run the client yourself, and set some options that will be passed to the server to initialize the game. The command line syntax is:

C++ `./client host port "STANDALONE" [options]`

python `python main.py host port "STANDALONE" [options]`

java `java Client host port "STANDALONE" [options]`

where `options` can be a combination of the following arguments (all of them are optional and if omitted are set to the default settings):

practice will generate a single bird, and immediately give you the actions for 500 turns. All other options are ignored.

numbirds=n will generate n birds. n must not be greater than 300. By default it is 180.

numspecies=n will generate n species. It must not be greater than 5. It does not include the black bird, which is generated independently. If $n = 1$, only white birds will be generated. As you increase it, birds of the different colors will appear. Bird generation will still follow the rules in section 2, with half of the birds being white, and the other half divided evenly among the colorful birds.

noblackbird will remove the black bird from the sky. By default, one and only one black bird exists.

numturns=n will make the game end after n turns. The default is 500.

timeperturn=n will allow n seconds to perform each action. The default is 1.

An example command line would be:

- `./client 130.237.218.85 6666 STANDALONE numbirds=50 numspecies=2 noblackbird`

When you receive them, the skeletons will already by able to play against other players, but they will choose their actions using a very poor strategy. This is what you have to do: modify the code that chooses the action so that it maximizes the score, ie kills as many birds as possible (preferably of the right type) while trying to avoid missed shots and killing the bird from the endangered species.

Some basic rules and limits:

- You cannot create any threads.
- The memory limit is 512 Mb.
- Your program cannot read from or write to any files.
- Your program cannot open any network connection (of course, connecting to the game server is an exception, but you will not write the code for this).

4 Provided code

A basic program is provided for you in each of the supported programming languages. The program, as it is provided, is fully functional, but it never shoots, which is not very interesting. You **must** implement a better strategy. The programs (also called skeletons) are written in such a way that they are easy to understand, and provide all the information given by the server in a simple interface.

For this assignment, you can only modify the CPlayer (Player in java) class. The rest of the skeleton must remain the same, and you will only submit the CPlayer class.

Skeletons provide all the code for communication with the server and state representation.

4.1 C++

The documentation for each class is available in each header file in the doxygen format [1].

To implement your solution, you must modify the class **CPlayer** (in files cplayer.h and cplayer.cc). There are three functions which you can modify as a starting point to add your own functionality. Of course, you can, and probably should, add your own functions to this class, or create additional classes.

[1] `http://en.wikipedia.org/wiki/Doxygen`

CAction Shoot(const CState &pState, const CTime &pDue) tells the player to perform an action. pState contains the current state, which includes the actions of all birds in all past turns. It also contains the score for each player. pDue is the time by when the function must return to avoid violating the time limit. You must return a CAction object from this function, containing the predicted action for the bird you are trying to shoot. The special constant cDontShoot can be returned if you choose to pass. *This is your main task; you must change this function*

void Guess(std::vector<CDuck> &pDucks, const CTime &pDue) will be called at the end of the game to ask your player to identify all remaining birds. For each of the birds in the vector which are alive (you can check that by calling the IsAlive() method), you must call the SetSpecies() method, providing the species for the bird. pDue will also contain the time by when your function must have returned. *This is also part of your task; you must change this function*

void Hit(int pDuck,ESpecies pSpecies) will be called whenever one of your shots kills a bird. It will tell you the index of the bird you shot and the species of the bird. Also, the species will be set in the corresponding CDuck object, so you can get it by calling its GetSpecies member function. *It is not necessary to change this function, but it can be helpful to do so*

There are some helper classes that you need to care about. You will probably find useful to use some of their public functions (so **please read their interfaces**), but you cannot modify them:

CAction represents an action that the bird has performed in the past. It is also used to return the predicted action from the Shoot function.

CDuck represents a duck. It contains methods to access the past actions and whether the bird is alive.

CState contains methods to access each of the ducks, and methods to access the current score.

CTime represents a time. It can be used to know the time remaining to perform a move.

4.2 Java

The documentation for each class is available in each header file in the doxygen format [2].

To implement your solution, you must modify the class **Player** (in file Player.java). There are three functions which you can modify as a starting point to add your own functionality. Of course, you can, and probably should, add your own functions to this class, or create additional classes.

[2] `http://en.wikipedia.org/wiki/Doxygen`

Action Shoot(State pState,Date pDue) tells the player to perform an action. pState contains the current state, which includes the actions of all birds in all past turns. It also contains the score for each player. pDue is the time by when the function must return to avoid violating the time limit. You must return an Action object from this function, containing the predicted action for the bird you are trying to shoot. The special constant Action.cDontShoot can be returned if you choose to pass. *This is your main task; you must change this function*

void Guess(Duck[pDucks, Date pDue)] will be called at the end of the game to ask your player to identify all remaining birds. For each of the birds in the vector which are alive (you can check that by calling the IsAlive() method), you must call the SetSpecies() method, providing the species for the bird. pDue will also contain the time by when your function must have returned. *This is also part of your task; you must change this function*

void Hit(int pDuck,int pSpecies) will be called whenever one of your shots kills a bird. It will tell you the index of the bird you shot and the species of the bird. Also, the species will be set in the corresponding Duck object, so you can get it by calling its GetSpecies member function. *It is not necessary to change this function, but it can be helpful to do so*

There are some helper classes that you need to care about. You will probably find useful to use some of their public functions (so **please read their interfaces**), but you cannot modify them:

Action represents an action that the bird has performed in the past. It is also used to return the predicted action from the Shoot function.

Duck represents a duck. It contains methods to access the past actions and whether the bird is alive.

State contains methods to access each of the ducks, and methods to access the current score.

4.3 Python

The documentation for each class is available in comments inside each class file.

To implement your solution, you must modify the class **CPlayer** (in file cplayer.py). There are three functions which you can modify as a starting point to add your own functionality. Of course, you can, and probably should, add your own functions to this class, or create additional classes.

Shoot(pState,pDue) tells the player to perform an action. pState contains the current state, which includes the actions of all birds in all past turns. It also contains the score for each player. pDue is the time by when the function must return to avoid violating the time limit. You must return a CAction object from this function,

containing the predicted action for the bird you are trying to shoot. You can also return None if you choose to pass. *This is your main task; you must change this function*

Guess(pDucks,pDue) will be called at the end of the game to ask your player to identify all remaining birds. For each of the birds in the vector which are alive (you can check that by calling the IsAlive() method), you must call the SetSpecies() method, providing the species for the bird. pDue will also contain the time by when your function must have returned. *This is also part of your task; you must change this function*

Hit(pDuck,pSpecies) will be called whenever one of your shots kills a bird. It will tell you the index of the bird you shot and the species of the bird. Also, the species will be set in the corresponding CDuck object, so you can get it by calling its GetSpecies member function. *It is not necessary to change this function, but it can be helpful to do so*

There are some helper classes that you need to care about. You will probably find useful to use some of their public functions (so **please read their interfaces**), but you cannot modify them:

CAction represents an action that the bird has performed in the past. It is also used to return the predicted action from the Shoot function.

CDuck represents a duck. It contains methods to access the past actions and whether the bird is alive.

CState contains methods to access each of the ducks, and methods to access the current score.

5 Grading

You will have to submit your solution as described in Section 6. Your grade will depend on how your program does in practice and single-player modes, and against our bot in competition mode:

- Consistently predicting the most likely action in practice mode – **12 points**
- Getting 200-600 points average in single-player mode – **0-12 points** (0 for less than 200 points, 12 for more than 600, linear in between)
- Beating `Mr. Smart` – **6 points**

To see how well your checker program does against our bots and against other people in the class, an online ranking system will be available at:

`http://130.237.218.82/ducks/`

6 Submit your solution

You need to submit a `zip` archive *usern_lang*`_hw2.zip`, where *usern* is your KTH username and *lang* is you programming language of choice (either `cpp`,`java` or `py`). For example, `johang_cpp_hw2.zip`.

The server will compile and run your solution under Ubuntu 11.04. Therefore your solution must compile and run in such a system. The skeletons provided do fulfill these requirements. Moreover, they should work in Windows and MacOSX as well. If you work under an environment different from Ubuntu, you should just avoid adding OS specific instructions in your code.

The `zip` archive you submit must contain all the files you received in the skeleton, where only the cplayer.h and cplayer.cc (for C++) and the Player.java (for java) may be different from the ones in the skeleton. It can also contain additional .h and .cc files (for C++) and .java files (for java).

It must not contain anything else.

6.1 C++ notes

The only supported libraries are the libraries which Linux links against by default[3]. Your solution cannot depend on third party libraries to run. Exception: in Windows you must link against sockets library ws2_32.lib.

Our evaluation system will run Ubuntu Linux. We will use the `g++` compiler version `4.5` from the GCC compiler suite to compile your solution.

6.2 Java notes

You can program your solution on Windows, MacOS X or Unix using a reasonably recent version of Java.

Your solution should be self-contained and cannot depend on external `java` files or third party libraries to run. The only supported library is the standard Java library. Windows or MacOS specific libraries or Java extensions are not allowed.

Our evaluation system will run Ubuntu Linux. We will use the official Oracle's `javac` compiler for Java 6.

6.3 Python notes

You can program your solution on Windows, MacOS X or Unix using a resonably recent version of Python 2 (e.g. 2.7) or 3. The skeleton we provide is compatible with (at least) versions 2.7 and 3.2 of Python. Evaluation will be done using Python 3.2, so if you decide to do your development in Python 2 you must be careful not to use any construction that is Python 2-specific[4]

[3] standard C++ library and the POSIX-compatible C library in Linux

[4] There are tools to convert your python2 code to python3: `http://docs.python.org/library/2to3.html`

Apart from the libraries in the standard Python distribution, numpy can also be used. You can only provide .py files, which must all lie in the same folder.

We will use Python version `3.2` to run your solution. Our evaluation system will run Ubuntu Linux.