The University of Melbourne
School of Computing and Information Systems
COMP30024 Artificial Intelligence

Project Part B:

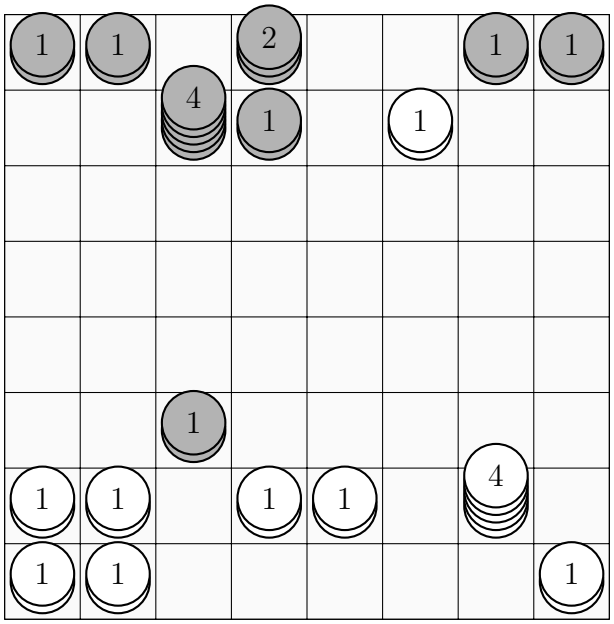
Playing the Game

Last updated March 30, 2020

Overview

In this second part of the project we will play the original, two-player version of *Expendibots* (as introduced in week 1). Before you read this specification you may wish to re-read the 'Rules for the Game of *Expendibots*' document.

The aims for Project Part B are for you and your project partner to (1) practice applying the game-playing techniques discussed in lectures and tutorials, (2) develop your own strategies for playing *Expendibots*, and (3) conduct your own research into more advanced algorithmic game-playing techniques; all for the purpose of creating the best *Expendibots*-playing program the world has ever seen.



The task

Your task is twofold. Firstly, your team will design and implement a program that 'plays' a game of *Expendibots*: given information about the evolving state of a game, your program will decide on an action to take on each of its turns (we provide a driver program to coordinate a game of *Expendibots* between two such programs, so that you can focus on implementing a game-playing strategy).

Secondly, your team will write a report discussing the strategies your program uses to play the game, the algorithms you have implemented, and other techniques you have used in your work.

The program

You must create a program to play the game in the form of a Python 3.6 package. The name of this package must be your team name. When imported, your package must define a class named `Player` conforming to the interface described below. Furthermore, to avoid naming conflicts with modules from other teams' packages, all Python modules within your package must use *absolute imports* when importing from other modules within your package.

Your package will therefore be a directory named with your team name. This directory will contain one or more Python modules (`.py` files) and subpackages (subdirectories). One file called `__init__.py` will expose a class called `Player` (though this class may be imported from another module where it is defined). See the provided skeleton code for a starting point, and make sure to seek clarification if you are unsure about how to structure your program.

Representing actions

Our programs will need a consistent way to represent actions. In this part of the project, all actions will be represented as *tuples with two or more components*; first a string representing the **action type**, followed by one or more values representing the **action arguments** (which may themselves be tuples). We represent the different types of actions as follows:

- To represent a move action, use a tuple of the format

  `("MOVE",` n`, (`x_a`,` y_a`), (`x_b`,` y_b`))`

  where `"MOVE"` is the action type, the first argument n is the number of tokens to move, (x_a, y_a) are the coordinates of the moving tokens before the move, and (x_b, y_b) are the coordinates after the move.

  For example, the tuple `("MOVE", 3, (0, 1), (0, 4))` represents a move action to move 3 tokens from the square indexed $(0, 1)$ to the square indexed $(0, 4)$ in Figure 1.

- To represent a boom action, use a tuple of the format

  `("BOOM", (`x`,` y`))`

  where `"BOOM"` is the action type and (x, y) are the coordinates of the stack initiating the explosion.

  For example, the tuple `("BOOM", (2, 3))` represents a boom action at the square indexed $(2, 3)$ in Figure 1.

While representing actions, we'll index the board's squares with the *same coordinate system* we used in Project Part A. This system uses a two element pair (x, y) to represent the x (horizontal) and y (vertical) position of each square on the board. This is shown in Figure 1. Note that the coordinates are zero-indexed.

$(0,7)$	$(1,7)$	$(2,7)$	$(3,7)$	$(4,7)$	$(5,7)$	$(6,7)$	$(7,7)$
$(0,6)$	$(1,6)$	$(2,6)$	$(3,6)$	$(4,6)$	$(5,6)$	$(6,6)$	$(7,6)$
$(0,5)$	$(1,5)$	$(2,5)$	$(3,5)$	$(4,5)$	$(5,5)$	$(6,5)$	$(7,5)$
$(0,4)$	$(1,4)$	$(2,4)$	$(3,4)$	$(4,4)$	$(5,4)$	$(6,4)$	$(7,4)$
$(0,3)$	$(1,3)$	$(2,3)$	$(3,3)$	$(4,3)$	$(5,3)$	$(6,3)$	$(7,3)$
$(0,2)$	$(1,2)$	$(2,2)$	$(3,2)$	$(4,2)$	$(5,2)$	$(6,2)$	$(7,2)$
$(0,1)$	$(1,1)$	$(2,1)$	$(3,1)$	$(4,1)$	$(5,1)$	$(6,1)$	$(7,1)$
$(0,0)$	$(1,0)$	$(2,0)$	$(3,0)$	$(4,0)$	$(5,0)$	$(6,0)$	$(7,0)$

Figure 1: Coordinate system for *Expendibots*.

The `Player` class

Your player class, named `Player`, must define at least the following three methods:

1. `def __init__(self, colour):` This method is called once at the beginning of the game to initialise your player. You should use this opportunity to set up your own internal representation of the game state, and any other information about the game state you would like to maintain for the duration of the game.

   The parameter `colour` will be a string representing the player your program will play as (White or Black). The value will be one of the strings `"white"` or `"black"` correspondingly.

2. `def action(self):` This method is called at the beginning of each of your turns to request a choice of action from your program.

   Based on the current state of the game, your player should select and return an allowed action to play on this turn. The action must be represented based on the above instructions for representing actions.

3. `def update(self, colour, action):` This method is called at the end of every turn (including your player's turns) to inform your player about the most recent action. You should use this opportunity to maintain your internal representation of the game state and any other information about the game you are storing.

   The parameter `colour` will be a string representing the player whose turn it is (White or Black). The value will be one of the strings `"white"`, `"black"` correspondingly.

   The parameter `action` is a representation of the most recent action conforming to the above instructions for representing actions.

   You may assume that `action` will always correspond to an allowed action for the player `colour` (your method *does not* need to validate the action against the game rules).

Program constraints

Resource limits: The following constraints will be strictly enforced on your program during testing. This is to prevent your programs from gaining an unfair advantage by using a large amount of memory and/or computation time. Please note that these limits apply to each player for an entire game. In particular, they *do not* apply to each turn.

- A maximum computation time limit of **60 seconds per player, per game**.
- A maximum memory usage of **100MB per player, per game** (not including space for imported libraries).

Furthermore, any attempt to circumvent these constraints *will not* be allowed. For example, your program must not use multiple threads and must not attempt to communicate with other programs to access additional resources. If you are not sure as to whether some other technique will be allowed, please seek clarification early.

Allowed tools: Your program will be run with **Python 3.6** on the **Melbourne School of Engineering's student unix machines** (for example, `dimefox`[1]). There, the following Python libraries will be available when we test your program. Beyond these, your program should not require any additional tools or libraries in order to play a game:

- The Python Standard Library.
- The third-party Python libraries NumPy and SciPy.
- The library of algorithms provided by the AIMA textbook website.

While you are developing your player program, however, you may use any tools or third-party Python libraries to help you do so. For example, you may like to use third-party Python libraries such as scikit-learn or TensorFlow to help you conduct machine learning. You may even like to use tools based on other programming languages. This is all allowed *as long as your Player class does not require these tools to be available when it plays a game.* If you use any such external tools to help develop your program then these should be acknowledged in your report.

[1] Note that Python 3.6 is not available on `dimefox` by default. However, it can be used after running the command '`enable-python3`' (once per login).

Running your program

To play a game of *Expendibots* using your program, we provide a driver program (a 'referee') in a Python package called `referee`, available on the LMS. The referee's main program essentially has the following structure:

1. Set up a new *Expendibots* game and initialise a White and Black player (constructing two `Player` classes including running their `.__init__()` methods).
2. Repeat the following until the game ends (starting with White as the current player, then alternating):
   (a) Ask the current player for their next action (calling their `.action()` method).
   (b) Validate this action and apply it to the game if it is allowed (otherwise, end the game with an error message). Display the resulting game state to the user.
   (c) Notify both players (including the current player) of the action (using their `.update()` methods).
3. Display the final result of the game to the user.

To play a game using the referee, invoke it as follows, ensuring that the referee package (the directory `referee`) and your player package (the directory named with your team name) are both within your current directory:

```
python -m referee <white package> <black package>
```

where `python` is the name of a Python 3.6 interpreter[2] and `<white package> <black package>` are the names of packages containing the class `Player` to be used for White and Black, respectively.

The referee offers many additional command-line options, including for inserting a delay between turns (to make games easier to watch); controlling output verbosity; creating an action log; enforcing time constraints and (on linux) space constraints; configuring output style; and using other player classes (not named `Player`) from a package (useful for testing your main player against other player classes). You can find information about these additional options by running:

```
python -m referee --help
```

The report

Finally, you must discuss the strategic and algorithmic aspects of your game-playing program in a separate file called `report.pdf` (to be submitted alongside your program). Your discussion may include some or all of the following points, or may contain anything else you consider relevant:

- Describe the approach your game-playing program uses for deciding on which actions to take throughout the game. Comment on your choice of search algorithm, and on any modifications you have made, and why. Explain your evaluation function and its features, including their strategic motivations.
- If you have applied machine learning, discuss how it fits into your program's overall game-playing approach, and discuss the learning methodology you followed for training and why you followed this methodology.
- Comment on the overall effectiveness of your game-playing program. If you have created multiple game-playing programs using different techniques, compare their relative effectiveness, and explain how you chose which program to submit for performance assessment.
- Include a discussion of any other important creative or technical aspects of your work, such as: algorithmic optimisations, specialised data structures, any other significant efficiency optimisations, alternative or enhanced algorithms beyond those discussed in class, other significant ideas you have incorporated from your independent research, and any supporting work you have completed to assist in the process of developing an *Expendibots*-playing program.

Your report can be written using any means but must be submitted as a PDF document. There is no maximum page limit or word limit. The right length for your report will depend on how much work you have completed. As a rough guide, a report within a single page may not include enough detail, but a 10-page report likely includes *far more detail than is necessary*. You should keep your writing succinct and avoid going in to too much technical detail (for example, there's no need to present detailed code inside your report).

[2] Note that Python 3.6 is not available on `dimefox` by default. However, it can be used after running the command '`enable-python3`' (once per login).

Assessment

Your team's Project Part B submission will be assessed out of 22 marks, and contribute 22% to your final score for the subject. Of these 22 marks:

- **11 marks** will be allocated to the level of performance of your final player.

  Marks are awarded based on the results of testing your player against a suite of hidden 'benchmark' opponents of increasing difficulty, as described below. In each case, the mark will be based on the number of games won by your player, playing in multiple test games as each colour against each type of opponent.

  3 marks available: Opponents who choose at random from their set of allowed actions each turn.

  2 marks available: 'Greedy' opponents who select the most immediately promising action available each turn, without considering your player's responses, for various definitions of 'most promising'.

  2 marks available: Opponents using the adversarial search techniques discussed in class and a simple evaluation function to look an increasing number of turns ahead.

  The remaining **4 marks** are available for a player capable of consistently completing games without syntax/import/runtime errors, without invalid actions, and without violating the time or space constraints.

  Please note once again, all test games will be run with **Python 3.6** on the **Melbourne School of Engineering's student unix machines**. Therefore, we strongly recommended that you test your program in this environment before submission[3]. If your team still has trouble accessing the student unix machines, please seek help *well before* submission.

- **11 marks** will be allocated to the successful application of techniques demonstrated in your work.

  We will review your report (and, on occasion, your code) to assess your application of adversarial game-playing techniques, including (but not limited to): your game-playing strategy, your choice of adversarial search algorithm, and your evaluation function and its development. For top marks, we will also assess your level of exploration beyond techniques discussed in class for enhancing the effectiveness of your player. Marks are awarded based on the following criteria:

  0–5 marks: Work that does not demonstrate a successful application of important techniques discussed in class for playing adversarial games.

  6–7 marks: Work that demonstrates a successful application of the important techniques discussed in class for playing adversarial games, possibly with some theoretical, strategic, or algorithmic enhancements to these techniques.

  8–9 marks: Work that demonstrates a successful application of the important techniques discussed in class for playing adversarial games, along with many theoretical, strategic, or algorithmic enhancements to these techniques, possibly including some significant enhancements based on independent research into algorithmic game-playing or original strategic insights into the game being played.

  10–11 marks: Work that demonstrates a highly successful application of important techniques discussed in class for playing adversarial games, along with many significant theoretical, strategic, or algorithmic enhancements to those techniques, based on independent research into algorithmic game-playing or original strategic insights into the game being played, leading to a program with excellent performance.

  Note that your report will be the primary means for us to assess this component of the project, so please use it as an opportunity to highlight your application of techniques discussed in class and beyond.

  Finally, please note that while we will not assess the quality of your submitted code, we may seek to clarify and verify claims in your report by referring to your implementation. Therefore, it is important for you to submit well-structured, readable, and well-documented code.

[3] Note that Python 3.6 is not available on `dimefox` by default. However, it can be used after running the command '`enable-python3`' (once per login).

Academic integrity

Your submission should be **entirely the work of your team.** We automatically check all submissions for originality. **Submitting work that is not entirely your own is against the university's academic integrity policy, and may lead to formal disciplinary action.** For example, please note the following:

1. You are encouraged to discuss ideas with your fellow students, but **it is not acceptable to share code between teams, nor to use code written by anyone else**. Do not show your code to another team or ask to see another team's code.
2. You are encouraged to use code-sharing/collaboration services, such as GitHub, *within* your team. However, **you must ensure that your code is never visible to students outside your team**. Set your online repository to 'private' mode, so that only your team members can access it.
3. You are encouraged to study additional resources to improve your Python skills. However, **any code adapted from an external source must be clearly acknowledged**. If you use code from a website, you should include a link to the source alongside the code.[4]

Please refer to the 'Academic Integrity' section of the LMS and to the university's academic integrity website (`academicintegrity.unimelb.edu.au`), or ask the teaching team, if you need further clarification.

Submission

One submission is required from each team. That is, one team member is responsible for submitting all of the necessary files that make up your team's solution.

You must submit a single compressed archive file (e.g. a `.zip` or `.tar.gz` file) containing all files making up your solution via the 'Project Part B Submission' item in the 'Assessments' section of the LMS. This compressed file should contain all Python files required to run your program (with the correct directory structure), along with your report. In addition, if you have created any extra files to assist you while working on this project,[5] then all of these files are worth including when you submit your solution.

The submission deadline for Project Part B is 11:00PM on Tuesday the 12th of May.

Late submissions are allowed. A late submission will incur a penalty of two marks per day (or part thereof) late. You may submit multiple times. We will mark the latest submission made by a member of your team, unless we are advised otherwise.

Extensions

If you require an extension, please email `comp30024-team@unimelb.edu.au` at the earliest possible opportunity. We will then assess whether an extension is appropriate. Requests for extensions on medical grounds received after the deadline may be declined.

Teamwork issues

In the unfortunate event that an issue arises between you and your partner (such as a breakdown in communication, a dispute about responsibilities or individual contributions to the project, or any other such issue), and you are unable to resolve this issue within your team, then please email `comp30024-team@unimelb.edu.au`. Note that it is in your best interest that any teamwork issues are identified as early as possible so that we have a chance to mitigate their effect on your completion of the project.

In addition, while completing this project, it is a good idea to keep brief notes, minutes, or chat logs recording topics discussed at each meeting, and other such documents related to your collaboration. In the event of a teamwork-related issue these documents (along with your project plan) will help us to reach a fair resolution.

[4] Note that for the purposes of assessing your successful application of techniques, using substantial amounts of externally sourced code will count for less than an original implementation. However, it's still better to properly acknowledge all external code than to submit it as your own, in breach of the university's policy.

[5] For example you may have created alternative player classes, a modified referee, additional programs to test your player or its strategy, programs to create training data for machine learning, or programs for any other purpose not directly related to implementing your player class. As long as these files are not too large, you are encouraged to include them with your submission (and mention then in your report).