**COMP30024 Artificial Intelligence**

# Project Part B:
# *Watch Your Back!*

Sarah Erfani, Matt Farrugia, Chris Leckie
Last updated: 31 March 2018

In this part of the project, you will design and implement a Python program to play a complete game of *Watch Your Back!*. Before you read this specification, please make sure you have carefully read the "Rules of the Game" document.

The aims for Project Part B are for you and your project partner to leverage the game-playing techniques discussed in lectures and tutorials, to develop your own strategies for playing the game, and to conduct your own research into more advanced game-playing algorithms, all for the purpose of creating **the best** *Watch Your Back!* player the world has ever seen.

## Task

Your task is as follows:

1. Create and submit a collection of Python 3.6 programs, including one program that defines a class called `Player`. Your `Player` class (your *player*) must be capable of playing a complete game of *Watch Your Back!*. To do this, it must define the methods described in the following 'Required methods' section. We provide a 'driver' program (`referee.py`) including a main function which coordinates a game of *Watch Your Back!* between two such `Player` classes.
2. Describe your program (including the strategies you have developed, the techniques you have implemented, and any other creative aspects of your solution) in a text file `comments.txt` (case-sensitive) (details below).

### Required methods

Your `Player` class must define *at least* the following three methods:

1. **Set up your player:** `def __init__(self, colour):` …

   This method is called by the referee 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 board, and any other state you would like to maintain for the duration of the game.

   The input parameter `colour` is a string representing the piece colour your program will control for this game. It can take one of only two values: the string `'white'` (if you are the White player for this game) or the string `'black'` (if you are the Black player for this game).

2. **Decide your next action:** `def action(self, turns): ...`

   This method is called by the referee to request an action by your player.

   The input parameter `turns` is an integer representing the number of turns that have taken place since the start of the current game phase. For example, if White player has already made 11 moves in the moving phase, and Black player has made 10 moves (and the referee is asking for its 11th move), then the value of `turns` would be `21`.

   Based on the current state of the board, your player should select its next action and return it. Your player should represent this action based on the instructions below, in the 'Representing actions' section.

3. **Receive the opponent's action:** `def update(self, action): ...`

   This method is called by the referee to inform your player about the opponent's most recent move, so that you can maintain your internal board configuration.

   The input parameter `action` is a representation of the opponent's recent action based on the instructions below, in the 'Representing actions' section.

   This method should not return anything.

   Note: `update()` is only called to notify your player about the opponent's actions. Your player will *not* be notified about its own actions.

**Representing actions**

Depending on the current game phase, the actions either player may take on their turn may involve placing a piece on a square, moving a piece from one square to another square, or forfeiting their turn. For the purposes of the `update()` and `action()` methods, we represent each of these actions as follows:

- To represent the action of placing a piece on square *(x,y)*, use a tuple `(x,y)`.
- To represent the action of moving a piece from square *(a,b)* to square *(c,d)*, use a *nested* tuple `((a,b),(c,d))`.
- To represent a forfeited turn, use the value **`None`**.

**Describing your program**

In addition to implementing a `Player` class, you must write and submit a text file called `comments.txt` (case-sensitive) describing your game playing program:

- Briefly describe the structure of your solution in terms of the major modules and classes you have created and used.
- Describe the approach taken by your game playing program for deciding on which actions to take, in terms of
  - your search strategy,
  - your evaluation function, and
  - any creative techniques that you have applied, for example, machine

learning, search strategy optimisations, specialised data structures, other optimisations, or any search algorithms not discussed in the lectures.
- If you have applied machine learning, you should discuss the learning techniques methodology you followed for training the agent, and the intuition behind using that specific technique.
- Include any other creative aspects of your solution, and any additional comments you want to be considered by the markers.

In addition, while working on your project, you may have built extra files to assist with your project. 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. All of these files are worth including when you submit your work, and should also be described in your `comments.txt` file.

Finally, **if you have implemented multiple `Player` classes, please tell us very clearly the location of the `Player` class you would like us to test and mark** by including a note near the top of your `comments.txt` file. Specifically, please tell us which Python file or module this class is contained within. This will ensure that we can test the correct player while marking your project.

# Playing the game

### Using the referee

The program `referee.py` (the *referee*) is the 'driver' program. It will allow you to actually play a game between two `Player` classes. It has the following structure:

1. Get the player class locations and other options from the command-line arguments
2. Load a new empty board, and initialise a White player and a Black player
3. Set the 'current player' to the White player, and the 'opponent' to the Black player
4. While the game has not ended:
    a. Ask the current player for their next action
    b. Apply this action to the board, if it is legal (otherwise, end the game with an error message)
    c. Update the opponent player with the current player's action
    d. Swap the current player and the opponent, and repeat this loop
5. Display the result of the game

To play a game using the referee, you must invoke it as follows:

```
python referee.py white_module black_module
```

where python is the name of a Python 3.6 interpreter, `white_module` is the full name of the module containing the `Player` class playing as the White player for this game, and `black_module` is the full name of the module containing the `Player` class playing as Black player for this game.

The referee offers some additional flags. These can be viewed by running:

```
python referee.py -h
```

### Performance constraints

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.

- A time limit of 60 seconds per player
- A memory limit of 100MB per player

Please note that these limits apply to each player for an **entire** game—**they do *not* apply on a per-turn basis.**

Any attempt to circumvent these constraints will not be allowed. For example, your program must be single threaded, and must run in isolation (without attempting to connect to any other programs, for example, via the internet, 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 `Player` class implementation may make use of any tools from within the Python Standard Library (without attempting to circumvent the performance constraints). In addition, you may use the library of classes available on the AIMA textbook website (provided you make appropriate acknowledgements that you have used this library). Furthermore, for this part of the project, your program may also use the non-standard Python libraries NumPy and SciPy. Beyond these tools, your `Player` class should **not** require any additional tools or libraries to be available in order to play a game.

However, while you are developing your player you may use *any* third-party tools or libraries you want. For example, you can use libraries to help you conduct machine learning (scikit-learn, TensorFlow, etc.). As long as your `Player` class does not require these tools to be available **when it plays a game*,*** you can use them freely. If you use any third-party tools, these should be acknowledged in your `comments.txt` file.

# Submission

### Assessment

Part B will be marked out of 22 points, and contribute 22% to your final mark for the subject. Of the 22 points:

- 4 points will be for the quality of your code: its structure (including good use of modules, classes, and functions), readability (including good use of code comments, and meaningful variable names), and documentation (including helpful docstrings for modules, classes, and important functions and methods).
- 4 points will be for the reliability and correctness of your program. When playing a game, your `Player` class should not encounter any runtime errors, it should not return any illegal actions, and it should not violate the space or time constraints.
- 7 points will be based on the performance of your `Player` class against a set of other ‘benchmark’ `Player` classes. These benchmark players will range in difficulty from a player who just takes random (legal) actions, to a player using search

techniques discussed in lectures with multiple game-specific optimisations and enhancements.
- 7 points will be for the 'creativity' of your solution. This is a measure of how far beyond the basic techniques discussed in lectures you have taken your implementation (to be discussed further in lectures). Note that your `comments.txt` file will be an important part of how we assess this component of the project, so please use it as an opportunity to highlight any creative aspects of your submission.

Please note that even if you don't have a program that works correctly by the time of the deadline, you should submit anyway. You may be awarded some points for a reasonable attempt at the project.

Please note that questions and answers pertaining to the project will be available on the LMS and will be considered as part of the specification for the project.

**Tournament**

After submission, we will run a tournament between the players of all groups to find out which player is the very best, and the top three groups will receive some awards. Your player's result in the tournament will not affect your project mark.

**Submission instructions**

The submission deadline for Project Part B is **4.00pm Friday 11th May 2018.**

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

You must submit a single compressed archive file (e.g. a `.zip` or `.tar.gz` file) containing all files making up your submission 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 player, **your `comments.txt` file**, and **any additional files** not directly related to implementing your `Player` class.

You can submit as many times as you like before the deadline. The most recent submission will be marked.

**Late submission**

Late submissions will incur a penalty of two marks per working day (or part thereof).

If you cannot submit on time, please contact both Sarah (sarah.erfani@unimelb.edu.au) and Matt (matt.farrugia@unimelb.edu.au) via email (use the subject header "COMP30024 Project B Extension") at the soonest possible opportunity (ideally **before** the deadline). If you have a medical reason for being late, you will be asked to provide a medical certificate. We will then assess whether an extension is appropriate. Requests for extensions on medical grounds received after the deadline may be declined.

Note that computer systems are often heavily loaded near project deadlines, and unexpected network or system downtime can occur. You should plan ahead to avoid leaving things to the last minute, when unexpected problems may occur. Generally, system downtime or failure will not be considered as grounds for an extension.

Late submission will be through the same section of the LMS. All late submissions must be accompanied by an email to let us know that you have made a late submission, and which submission you want us to mark.

### Academic integrity

There should be one submission per group. You are encouraged to discuss ideas with your fellow students, but your program should be entirely the work of your group. It is not acceptable to share code between groups, nor to use the code of someone else. You should not show your code to another group, nor ask another group to look at their code. **If your program is found to be suspiciously similar to someone else's or a third party's software, or if your submission is found to be the work of a third party, you may be subject to investigation and, if necessary, formal disciplinary action.**

Please refer to the 'Academic Integrity' section of the LMS, and to the university's academic honesty website http://academichonesty.unimelb.edu.au/ if you need any further clarification on these points.

# Teamwork

### Project plan

In addition to the tasks described above, as part of your project, you and your partner must create and submit a brief 'project plan' describing how you and your partner plan on working together to complete this part of the project. Specifically, you should include:

- A breakdown of how you plan on sharing the workload of this project between you and your partner.
- A summary of where and how regularly you plan on communicating about the project with your partner (e.g. weekly meetings after tutorials, video chat, email).
- A list of upcoming deadlines for work in other subjects between now and the end of semester.

You must submit this document to the 'Project Plan Submission' item in the 'Assignments' section on the LMS by **4.00pm, Wednesday 18th April 2018**.

### Teamwork reflection

Each group member will have an opportunity to individually comment on their experience after the final submission of Project Part B. If you would like to make any comments on your experience as part of your group in relation to the relative contribution of each group member, you may individually submit these comments. Note that this reflection should be completed individually, and is optional (you may choose not to submit a reflection).

If you want to complete a teamwork reflection, submit your comments to the 'Teamwork reflection submission (optional)' item in the 'Assignments' section of the LMS before **11.59pm, Sunday 20th May 2018** (the end of week 11).

### Teamwork issues

In the unfortunate event that issues arise between you and your partner (such as a breakdown in communication, a dispute about responsibilities or individual contributions to

the project, or other such issues), and you are unable to resolve these issues yourselves first, then please contact both Sarah (sarah.erfani@unimelb.edu.au) and Matt (matt.farrugia@unimelb.edu.au) via email **at the earliest opportunity**. Note that it's in your best interest that any teamwork issues are identified as early as possible, so that we have a chance to mitigate the effect that it has on your project's results.

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