Python Project 2
Programming board games in Python

Alexander Konovalov
CS2006, 2014/15

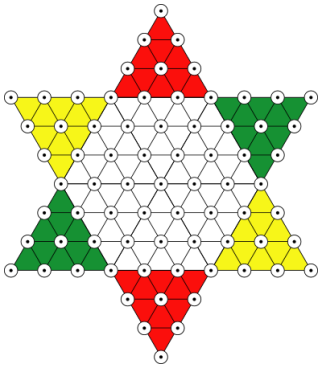
Due date: Tuesday 21st April, 21:00
33.3% of Overall Mark for the Module

Overview

The objective of this project is to learn how to implement a board game in Python. By completing this project, you should further enhance your Python programming skills and get more detailed insights into various aspects of Python's applications, such as, for example, computer graphics and AI.

The Game

The game chosen for this practical is called **Diamond**. It is played by two or three players on a hexagram-shaped board with 73 spaces [1]:



Each player has 10 pieces, placed in the corner of the star The aim is to move all pieces to the opposite corner faster than the opponent(s). The pieces may be moved either by making one step to an adjacent space in any direction or by jumping one or more times over any other pieces (irrespective of their owner). See `http://en.wikipedia.org/wiki/Chinese_checkers#Diamond_game` for further details.

A related game is **Chinese Chequers**, played on a larger hexagram-shaped board by two, three, four or six players (`http://en.wikipedia.org/wiki/Chinese_checkers`). In this case, the board may have 121 spaces, and each player would have 10 or, in some variations, 15 pieces.

Chinese Chequers, in its turn, is considered to be a variation of **Halma**, a game played by two or four players on a chequered square board of several possible sizes (`http://en.wikipedia.org/wiki/Halma`). The standard size of the board is 16x16, but it is also played on 8x8 and 10x10 boards. It is considered to be more difficult than the two previous games as a piece may move in up to eight different directions instead of six.

[1] "Diamond Game" by Dmthoth - Own work. Licensed under CC BY-SA 3.0 via Wikimedia Commons - `http://commons.wikimedia.org/wiki/File:Diamond_Game.svg#mediaviewer/File:Diamond_Game.svg`

Basic Requirements

In this project, you have to implement the **Diamond** game with the following characteristics:

- A standard board with 73 spaces and 10 pieces for each player;
- Playable by two players starting in the opposite corners of the star;
- Offering a single-player mode with a computer opponent.

To achieve this, you have to:

1. choose and implement data structures used to represent the game state;
2. set up all the necessary graphical details;
3. implement the main game loop which handles events, updates the game state and then redraws the screen or quits when the game is over;
4. implement algorithms to calculate the next move of the computer following the strategy of finding the longest hopping path that leads closest to the destination.

You may start by looking at the collection of resources on game programming with Python at the Python Wiki: `https://wiki.python.org/moin/GameProgramming`. In particular, it mentions the PyGame library (`http://www.pygame.org/`), which should allow you to implement everything for this practical. PyGame comes with the documentation too, and it is also available online at `http://www.pygame.org/docs/`.

Another very useful resource is the book **"Making Games with Python & Pygame"** by Al Sweigart (`http://inventwithpython.com/`) which is a free, Creative Commons-licensed book supplemented by the well-documented source code, which gives you an excellent starting point. You may start your work on the project by reading this book and playing with supplementary codes as you read.

PyGame 1.9.1 has been installed on all Linux lab clients and the host servers. You may test its availability using the following commands to run the first example from the **"Making Games with Python & Pygame"** book:

```
wget https://inventwithpython.com/pygameHelloWorld.py
/usr/bin/python pygameHelloWorld.py
```

This should open a new window (for remote connection, you should enable X11 forwarding).

Note that there are four versions of Python on the lab Linux systems: two system ones (v2 and v3) and two locally built ones (v2 and v3). You can also use venv or virtualenv to have your own setup. PyGame has been installed from the Fedora repository and is for the system Python 2.7 only. Of course, you're not limited to the PyGame and may consider other options as well.

The minimal requirement is to implement the game machinery by addressing items (1)–(3) above to allow two human players to play the the game on the computer by making moves in turns. Alternatively, there may be one human player, and the computer would just make some random moves.

A basic implementation of all of the four items above, which demonstrates a reasonable choice of data structures, has accurate graphics, and uses some reasonable algorithm to search for the best possible move would be marked with the highest grade 13. In order to achieve a grade higher than 13, you should implement some of the additional requirements below.

Additional Requirements

Note: It is strongly recommended to ensure that you have completed the Basic Requirements and have something to submit before you attempt to deal with the Additional Requirements.

In order to achieve a grade higher than 13, you should implement some of the additional requirements below. In particular, for a grade higher than 17, you should implement more than three of the requirements marked as **Easy** and three of the requirements marked as **Medium**.

You should not be limited by these and should feel free to discuss and implement any other feature that you consider useful (please make sure that it will be described in the report!)

- **Easy:** Add playing sounds during the moves
- **Easy:** Implement animation for the moves to show each jump
- **Easy:** Use some graphical images for pieces instead of drawing circles
- **Easy:** Make some other improvements of the game graphics, e.g. use anti-aliasing
- **Easy:** Implement rollback of the last move
- **Medium:** Implement saving and loading the game
- **Medium:** Introduce time limits for the moves and the "Pause" command
- **Medium:** Add functionality to configure optional parameters of the game (for example, colours and initial locations of pieces; turning sound on/off, etc.)
- **Medium:** Add possibility of displaying hints to the player
- **Medium:** After the game, count the number of the remaining moves for the losing side
- **Medium to Hard:** Diversify the strategies used by the computer by introducing game levels and adding various AI "personalities"
- **Medium to Hard:** How would the player would make moves consisting of several jumps? A basic implementation might require the player to confirm that the move is finished. An improved version may only require to click on the piece and its final destination.
- **Hard:** Improve algorithms to calculate the next move by, for example, trying to build "chains" (also known as "paths", "bridges", "ladders") or hindering opponent's moves.
- **Hard:** Implement logging to allow reproducing the whole game
- **Hard:** Implement multi-player game specifying for each player whether it is a human or a machine
- **Hard:** Implement **Chinese Chequers**
- **Hard:** Implement **Halma** on a square board with variable board sizes and numbers of pieces

Deliverables

Hand in via MMS, by the deadline of 9pm on Tuesday of Week 11, a single `.zip` or `.tar.gz` file containing two top level subdirectories called `Code` and `Report`, as follows:

The `Code` directory should contain your group's code, and should be the same for everyone in the group. There may be further subdirectories if you wish, containing the source code in well-commented `.py` files, and any other files e.g. text and/or images that you may need to run the game. Everything that is needed to run your application should be in that directory or part of the standard CS lab machines configuration. The marker should be able to run your code on the CS lab machines, and you should document how to run it.

The `Report` directory should contain an individual report (of around 1500–2000 words), in PDF format, describing your design and implementation and any difficulties you encountered. In particular, it should include:

- A summary of the functionality of your program indicating the level of completeness with respect to the Basic Requirements, and any Additional Requirements.
- Any known problems with your code, e.g. any Basic or Additional Requirements that are not met, but were attempted to be implemented.
- Any specific problems you encountered which you were able to solve, and how you solved them.

- The level of adaptability of your code to Chequers and Halma (in case you haven't attempted this in the project).
- A section (**common for both Practicals**) containing your comments on the comparison of the suitability of Python and Haskell for the implementation of board games. Which parts of the implementation were easier to develop in Haskell and which – in Python? Which features from one language were you missing while programming the game using the other one? Which of the two languages would be your first implementation choice for each of the two games?
- An accurate summary of provenance, i.e. stating which files or code fragments were:
  1. written by you;
  2. modified by you from the source files provided for this assignment;
  3. sourced from elsewhere and who wrote them.
- A description of your own contribution to the group work.

Marking Guidelines

This practical will be marked according to the guidelines at `https://studres.cs.st-andrews.ac.uk/Library/Handbook/academic.html#/Mark_Descriptors`. To give an idea of how the guidelines will be applied to this project:

- A simple prototype implementation which supports two human players using the computer to take turns, accompanied by a report, should get you a grade 7.
- A solid basic implementation which demonstrates a reasonable choice of data structures, has accurate graphics, and uses some reasonable algorithm to search for the best possible move, with a well-commented/documented fully-working code, accompanied by a clear and informative report, should get you a grade 13. The report should address all the Basic Requirements, and should make clear which of these you have completed.
- To achieve a grade between 13 and 17, you have to implement between one and three requirements marked as **Easy** and between one and three requirements marked as **Medium**.
- To achieve a grade higher than 17 you have to go beyond the requirements described above by either implementing extra Easy and Medium additional requirements, or by implementing some of the Hard ones, providing well-commented and documented code, accompanied by a clear and informative report.

Finally, remember that:

- Standard lateness penalties apply as outlined in the student handbook at `https://studres.cs.st-andrews.ac.uk/Library/Handbook/academic.html#/lateness-penalties`
- Guidelines for good academic practice are outlined in the student handbook at `https://studres.cs.st-andrews.ac.uk/Library/Handbook/academic.html#/Good_Academic_Practice`

Finally

The project is very much open-ended, so please feel free to discuss your own agenda in addition to implementing Easy and Medium requirements in order to score a high grade, like optional variations of the rules; network multiplayer tournaments between humans and computers; extending your project to play Chinese Chequers; extending it to play Halma on a square board with variable board sizes and numbers of pieces, etc. Be creative, have fun, and produce something you would be proud of!

Finally, please let me or your tutor know if you have any questions or problems! There will be four Tutorials during your work on this project: March 30th, April 6th, April 13th and April 20th. Please be prepared to discuss the following questions:

1. How did you divide responsibilities within your team?
2. Which requirements have you implemented so far?
3. Any requirements or language features you are having difficulty with?

A rough guideline is that you familiarise yourself with the documentation and the literature, sort out the necessary technical details, and have an implementation plan to discuss it at the Tutorial on March 30th. You should have a prototype by April 6th and complete the the basic implementation and Easy requirements by April 13th. That gives you roughly a week to work on Medium and Hard ones. Of course, these are just guidelines, and you may achieve much faster progress!

You may contact me by email `alexander.konovalov@st-andrews.ac.uk` or find me in my office JC 1.02, and we will be also meeting at the Tutorials at 11am on Mondays and in the JH lab at 11am on Fridays.

Alexander Konovalov
March 10th, 2015