CSCI3180 – Principles of Programming Languages – Spring 2012

Course Project — Jungle Chess

Phase 1 Deadline: Mar 4, 2012 (Sunday) 23:59
Phase 2 Deadline: Mar 25, 2012 (Sunday) 23:59

1 Introduction

Jungle Chess or Dou Shou Qi is a traditional Chinese board game with a 9x7 game board. It is a 2-player game where each player has eight chess pieces, each representing a different animal. The goal is to either get into the den of the opponent's side or capture all of enemy's pieces. There are different versions of the game rules for Jungle Chess. In order to avoid ambiguity, we fix the game rules and they are covered in details in a later section. Figure 1 shows the chess board of Jungle Chess.

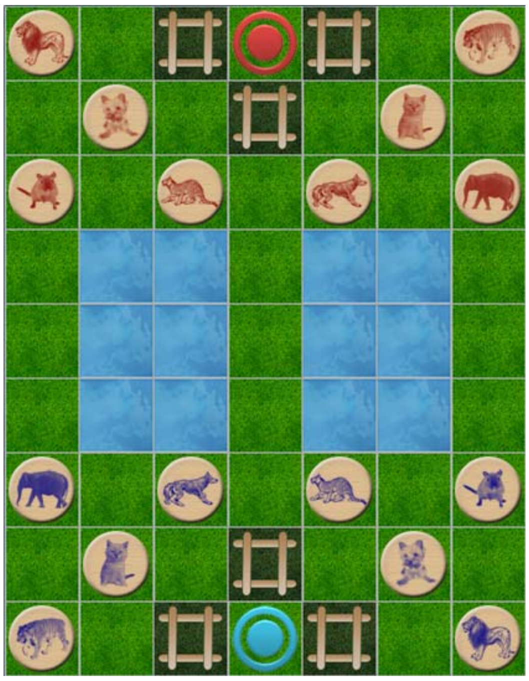


Figure 1: Jungle Chess

In this project, your are required to design and implement the Jungle Chess game using C and Smalltalk. You will do the project **in a group of two** in three phases, plus an extra bonus part if you are interested:

1. Finish a small exercise using C and Smalltalk in order to get familiar with the languages and the development environment. The second task is to create an object-oriented design for your Jungle Chess.
2. Implement Jungle Chess using C and Smalltalk by strictly following the defined game rules.
3. Extend your program to suit our extra requirements and features for an enhanced version of Jungle Chess. Details will be announced in Phase 3.
4. (Bonus, optional) Further extend your implementation. Details will be announced in Phase 3.

2 Game Details

The details of the game are covered in this section. Please follow strictly when designing and implementing your Jungle Chess game.

2.1 Players

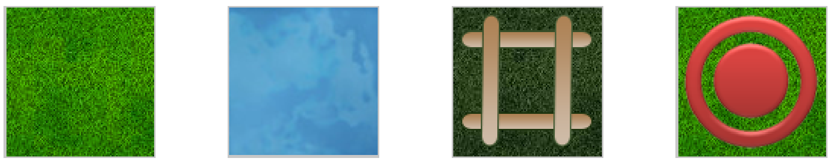
There are two players in the game: **Player Red** and **Player Blue**. Player Red owns the red chess pieces and occupies the upper part of the chess board. Player Blue owns the blue chess pieces and occupies the lower part of the chess board. Your program should always let **Player Blue play first**.

2.2 Goal

There are two ways to win the game: 1) move any one of your pieces into the den of the opponent's side; or 2) capture all opponent pieces.

2.3 Chess Board

The chess board consists of 9 rows and 7 columns of squares. There are two **rivers**, each of size 3x2 located in the middle, in the chess board and the other squares are the earthly **grounds**. The two sides are separated by the rivers and there are three **traps** and one **den** in each side. Figure 2 shows the different types of squares on the chess board.



(a) Ground (b) River (c) Trap (d) Den

Figure 2: Four possible types of squares of the chess board

2.4 Chess Pieces

There are 8 chess pieces in each side, each representing a different animal. Figure 3 shows the eight chess pieces and their representing animals. Before the game starts, the eight pieces of each side are placed on their **predefined position**, as shown in **Figure 1**. The **ranking** among the eight pieces (animals) is defined as follows (from strongest to weakest):

`Elephant` > `Lion` > `Tiger` > `Jaguar` > `Wolf` > `Dog` > `Cat` > `Rat`

and `Rat` > `Elephant`

2.5 Game Rules

The two players take turns to move their own chess pieces and they can move one chess piece in each turn. The game ends when one of the players wins. The following rules govern capturing and movement.

Capturing

Capturing happens when an attacking animal *A* moves to a square which is occupied by another animal *B* of the opponent, and *A* is able to capture *B*. In such a case, *B* will be removed from the chess board and *A* will occupy the square originally occupied by *B*. **Animals can capture any opponent's animals with the same or lower ranking.** For example, Lion can capture Lion or Wolf, but Wolf cannot capture Lion. However, there are some exceptions.

(a) Elephant

(b) Lion

(c) Tiger

(d) Jaguar

(e) Wolf

(f) Dog

(g) Cat

(h) Rat

Figure 3: Eight chess pieces representing eight different animals

- Rats can capture Elephants because they are so small and are able to get into the noses or ears of the Elephants to bite them. So, **Rats can capture Elephants but Elephants cannot capture Rats**.
- **When a chess piece moves into the trap of the opponent's side, it becomes super weak and any opponent's animal can capture the piece in the trap.**
- **When a chess piece moves into their own trap, it becomes super strong and no opponent's animals can capture the piece in the trap.**

Movement

In each turn, a chess piece can be moved **one step vertically (up/down) or horizontally (left/right)** but not diagonally. No chess pieces **can be moved into their own den**. When the target square of the movement is occupied by another chess piece (target animal), the moving chess piece (offensive animal):

- **cannot move to the target square if the offensive animal and the target animal are comrades (belonging to the same player), or the offensive animal is not able to capture the target animal.**
- **can move to the target place if the target square is empty or the offensive animal is able to capture the target animal. In such a case, the target animal will be removed from the chess board.**

In general, no chess pieces can move across rivers, except: Lion, Tiger and Rat. They can move across the rivers in different ways:

- Rats can move into a river square as if it is a ground square. **Any other animal cannot capture the Rat in the river**. However, **two Rats can capture each other in all these cases: 1) both on the ground, 2) both in the river, 3) one on the ground and one in the river**. **However, a Rat in the river cannot capture an Elephant on the ground**.
- **Lions and Tigers can jump over a river horizontally or vertically**. They jump from a square just beside a river square to the first ground square on the other side of the river vertically or horizontally. They can capture the animal on the target square if they are able to capture that kind of animals. Otherwise, they cannot jump over the river. In addition, **they cannot jump over the river if a Rat is on their intended path for jumping in the river**.

3 Phase 1

In this phase, you are required to finish a small and simple programming exercise using C and Smalltalk to get familiar with the languages and development environments. In addition, you need to make an object-oriented design for your Jungle Chess Smalltalk implementation using UML.

3.1 Programming Exercise

You have to finish the Vigenère Cipher using C and Smalltalk. The details of the exercise is covered in a separate exercise specification.

3.2 Object-Oriented Design

In addition to the programming exercise, you are required to create an object-oriented design in UML for your Smalltalk implementation in Phase 2. As part of this phase, you should read Chapters 1 and 2 of Booch's book (Grady Booch, Object-Oriented Analysis and Design with Applications, 2nd Edition, Benjamin/Cummings).

Your design should be as object-oriented as possible, and capable of allowing enhancements required in Phase 3. You have to write up a **report** of your design. The report must state clearly the purpose of each class in the UML diagram and the purpose of each method. In addition, your report must describe the relationships between classes, and among methods in your UML diagram.

4 Phase 2

In Phase 2, you have to implement Jungle Chess using two different programming languages: C and Smalltalk. The programs should strictly follow the game rules defined in Section 2 and be able to handle all possible situations correctly.

4.1 C Implementation

Please follow the requirements below in the C implementation.

User Interface

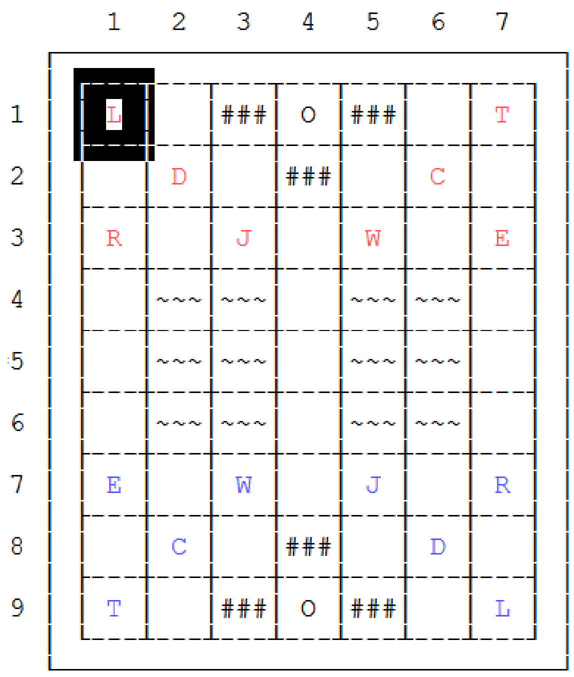
To make things simpler, you do not need to make a graphical user interface for the C implementation. However, you MUST use the **Ncurses** (**n**ew **curses**) library to create a text user interface. With Ncurses, you should be able to paint colors on your chess pieces, highlight the chosen chess piece, move the chess pieces with arrow and space buttons, etc. Your program should have the **same game board** as shown in Figure 4.

Figure 5 shows the four types of squares on the game board: **ground**, **river**, **trap** and **den**. We use the **first letter** of the animal's name to represent the chess piece as defined in Table 1. If a chess piece is on a particular square on the chess board, the centering character is replaced by the animal's letter as shown in Figure 4.

Input and Output Method

The two players play by moving their chess pieces in turn. Players can move around the board using **arrow** buttons (←, ↑, ↓, →). Chess pieces can be picked up and put down using the **space** button.

We **highlight** important information using **reverse video view**: the colors of the background and the text in ordinary video are switched/inverted. There are two things to be highlighted: the selected square and the selected chess piece.



```
      1     2     3     4     5     6     7

   +-----+-----+-----+-----+-----+-----+-----+
1  |  L  |     | ### |  O  | ### |     |  T  |
   +-----+-----+-----+-----+-----+-----+-----+
2  |     |  D  |     | ### |     |  C  |     |
   +-----+-----+-----+-----+-----+-----+-----+
3  |  R  |     |  J  |     |  W  |     |  E  |
   +-----+-----+-----+-----+-----+-----+-----+
4  |     | ~~~ | ~~~ |     | ~~~ | ~~~ |     |
   +-----+-----+-----+-----+-----+-----+-----+
5  |     | ~~~ | ~~~ |     | ~~~ | ~~~ |     |
   +-----+-----+-----+-----+-----+-----+-----+
6  |     | ~~~ | ~~~ |     | ~~~ | ~~~ |     |
   +-----+-----+-----+-----+-----+-----+-----+
7  |  E  |     |  W  |     |  J  |     |  R  |
   +-----+-----+-----+-----+-----+-----+-----+
8  |     |  C  |     | ### |     |  D  |     |
   +-----+-----+-----+-----+-----+-----+-----+
9  |  T  |     | ### |  O  | ### |     |  L  |
   +-----+-----+-----+-----+-----+-----+-----+
```

Figure 4: Game Board for C Implementation

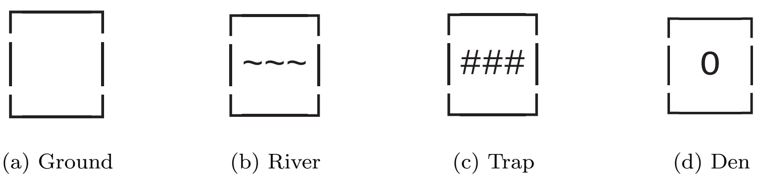


Figure 5: Four possible components of the chess board in C implementation

The **selected square** is the current square where the player moves to and should be highlighted. The corresponding selected square is changed when the player moves around using the arrow buttons. A highlighted river square is shown in Figure 6a. The initial selected square is the one at the upper-left corner, as shown in Figure 4.

When a player press the space button to pick up a chess piece, this **selected chess piece** should be highlighted, as shown in Figure 6b. Even after the player moves away from that square, the selected chess piece should still be highlighted as shown in Figure 6c. As soon as the player puts down the chess piece, it is changed to ordinary video again.

Imagine this is a real game board and a chess piece is placed ON a square so the view of a chess piece covers that of a square. Figure 6d shows a view of an ordinary chess piece on a selected river square. Figure 6e shows a view of a selected chess piece on a selected river square.

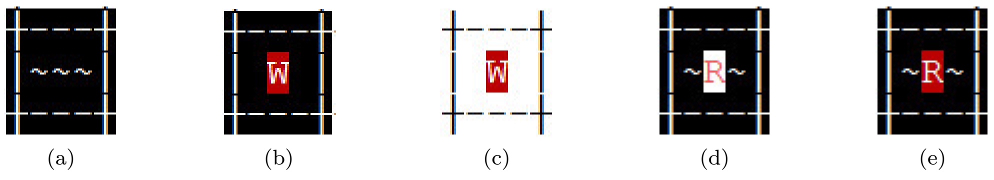


Figure 6: Highlighting examples

Animal	Chess Piece
Elephant	E
Lion	L
Tiger	T
Jaguar	J
Wolf	W
Dog	D
Cat	C
Rat	R

Table 1: Chess Pieces Representation in C implementation

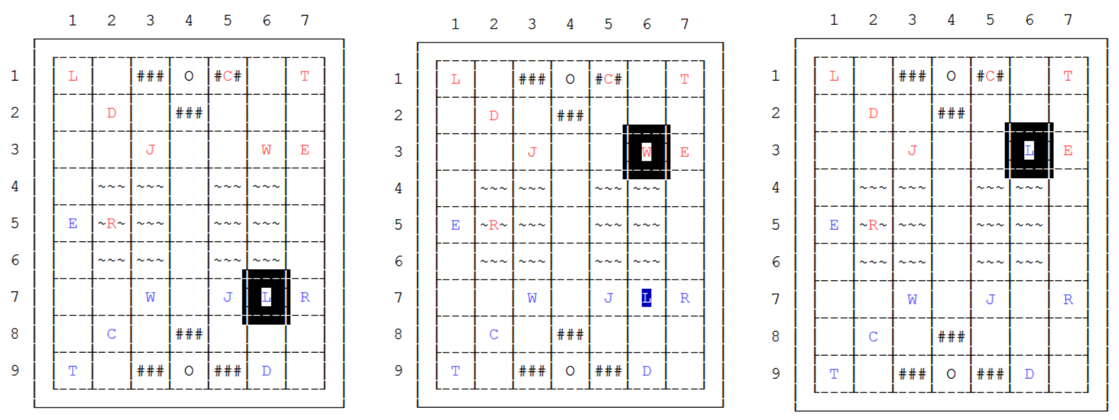
There are some rules to be followed:

- A player cannot select an opponent's chess piece. Only a successfully selected chess piece is highlighted. Otherwise, a warning message should be prompted.
- A selected chess piece is changed to ordinary video view ONLY when it is successfully put down. Therefore it is still highlighted under an invalid move.
- If the player wants to re-select a chess piece, the current selected chess piece should be put down on the original square first.

The movement of a chess piece is given as follows:

1. Move to the square where the chess piece is located.
2. Press space button to pick up the chess piece, and the selected chess piece is highlighted.
3. Moves to the intended destination square.
4. Press space button to put down the chess piece, and the chess piece is changed to ordinary video again.

Figure 7 shows sample moves in the C implementation. You should indicate clearly if a chess piece is picked up by highlighting it, as shown in Figure 7b. In addition to the movement of chess pieces, if a player inputs a single character '`q`' or '`Q`', the program is **terminated**. And if a player inputs a single character '`n`' or '`N`', the game is **restarted**. Your program should only take the movement input, quit input and new game input from the standard input terminal.



```
    1    2    3    4    5    6    7

1   L        ###   O   #C#        T
2        D        ###
3             J             W     E
4       ~~~  ~~~       ~~~  ~~~
5   E   ~R~  ~~~       ~~~  ~~~
6       ~~~  ~~~       ~~~  ~~~
7             W         J   L     R
8        C        ###
9   T        ###   O   ###  D
```

(a) Move to Blue-Lion

```
    1    2    3    4    5    6    7

1   L        ###   O   #C#        T
2        D        ###
3             J             W     E
4       ~~~  ~~~       ~~~  ~~~
5   E   ~R~  ~~~       ~~~  ~~~
6       ~~~  ~~~       ~~~  ~~~
7             W         J   L     R
8        C        ###
9   T        ###   O   ###  D
```

(b) Pick up Blue-Lion and move to Red-Wolf at (3,6)

```
    1    2    3    4    5    6    7

1   L        ###   O   #C#        T
2        D        ###
3             J             L     E
4       ~~~  ~~~       ~~~  ~~~
5   E   ~R~  ~~~       ~~~  ~~~
6       ~~~  ~~~       ~~~  ~~~
7             W         J         R
8        C        ###
9   T        ###   O   ###  D
```

(c) Put down Blue-Lion at (3,6) and Red-Wolf is captured

Figure 7: Sample moves in C implementation

You should print the initial game board with every chess pieces on their **initial position** as defined in **Figure 4** and change the game state whenever there is a change. In addition, your program should be interactive and you should provide **necessary guidelines, prompts and alerts** during the game. You **MUST** clearly show which player should take the current turn and show warning messages if the players attempt to make invalid moves.

4.2 Smalltalk Implementation

You MUST utilize the **Model-View-Controller (MVC)** paradigm in your Smalltalk implementation and you can use either one of the following object-oriented designs:

1. The design you submitted in Phase 1;
2. A sample design we shall give you after the Phase 1 deadline;
3. An adaptation of our sample design as you see fit.

The choice must be stated when you come to do the project demonstration in Phase 2. If you choose option 3, you are required to submit a new design together with your program codes.

Extra Functional Requirements

Besides the correct game functionality, your program should also have the following functions:

1. A "New Game" button to let players start a new game anytime (during/after a game).
2. Highlight the chess piece which is selected by the player in the current turn.
3. Clearly indicate which player should take the current turn.
4. Provide a message prompting area to show the warning message if there is an attempt to make invalid moves.

Graphical User Interface

The program should have the **same game board** as shown in Figure 1. The figures are provided for you in the course homepage. You are **free to design** the other part of the graphical user interface such as the message prompting area. In addition, you are free to design how players move their chess pieces but it must be a **mouse-based** operation (keyboard-based movement is not allowed). The interface should also clearly indicate and highlight selected chess pieces as in the C implementation. Although you have the flexibility to design, your user interface should be user-friendly and this is one of the marking criteria.

4.3 Report

You have to write up a report to answer the following questions:

1. Compare the advantages and disadvantages in implementing the Jungle Chess using C and Smalltalk. You may support your points of view regarding to some particular tasks in Jungle Chess Game.
2. Specify why you insist in using your own design or switch to our sample design. If you have improved your own design or sample design in Phase 1, please briefly explain what you changed and why.

5 Development Environment

Please test your programs under the following environments before you submit it. You take your own risk if you just test your program within other development environments.

5.1 C Implementation

Make sure you can compile and run your C program without any problem with the gcc compiler on CSE **linux** machines. You should be able to print the borders of the game board with the use of Ncurses. One point to note is about displaying the extended characters (for example, the corners of the game board). We suggest you to remote access you linux account using PuTTY which can display the extended characters without any problems.

5.2 Smalltalk Implementation

You are required to use **VisualWorks version 7.8** to develop your Smalltalk implementation. The download link for VisualWorks is provided in the course homepage. You should create a new package to do your project and export the whole package for submission. As in Figure 8, you can export your package in the System Browser in VisualWorks by right-clicking your package, selecting "File Out -> Package" and saving your package as "JungleChess.st".

In addition, you have to export your workspace that is used to start up your program. In your workspace, click "File -> Save As" and save it as "JungleChessWorkSpace.ws".

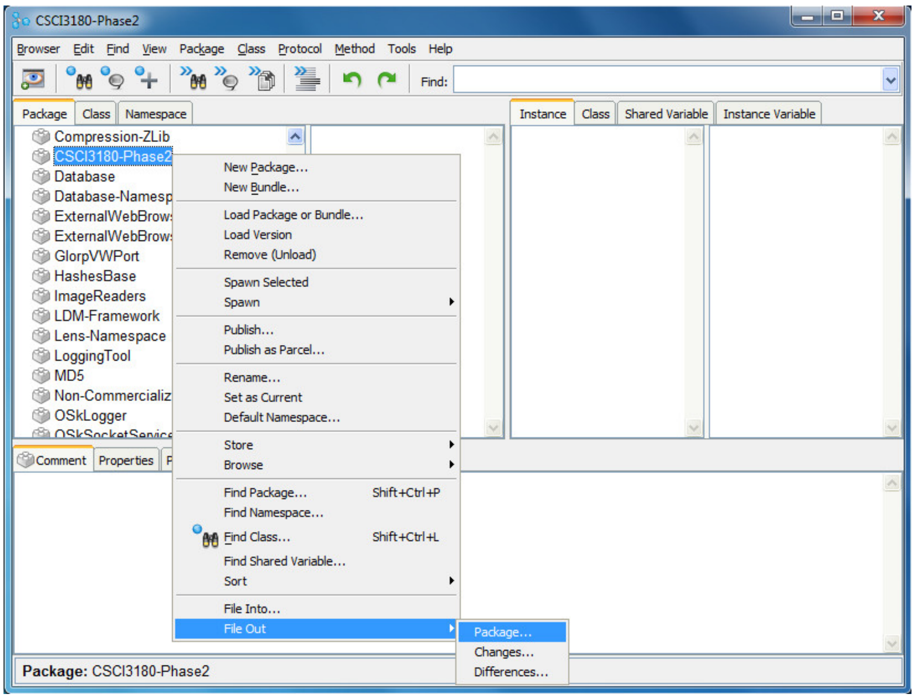


Figure 8: Export your source code from VisualWorks

6 Other Requirements

You are required to follow the requirements below throughout the project.

1. **Division of Labour**
   You should divide the work in a way such that both of you participate in all phases, including the OO design and the implementation of both C and Smalltalk. In this way, both of you can learn and participate in different phases of constructing a real application and be able to tell the differences and difficulties in developing the game with the two languages.

2. **Error Handling**
   The programs should handle possible errors gracefully by printing meaningful error messages

to standard output. For example, failure to open a non-existing file or input with wrong format.

3. **Good Programming Style**
A good programming style not only improves your grade but also helps you a lot when debugging. Poor programming style will receive marks deduction. Construct your program with good readability and modularity. Provide enough documentation in your codes by commenting your codes properly but not redundantly. Divide up your programs into subroutines instead of clogging the main program. The main section of your program should only handle the basic file manipulation such as file opening and closing, and subroutine calling. The main purpose of programming is not just to make it right but also make it good.

4. **Other Notes**
You are **NOT** allowed to implement your program in another language (e.g. Assembly/C++/Java) and then initiate system calls or external library calls in C and Smalltalk. Your source codes will be compiled and PERUSED, and the object code tested!

   In the C implementation, DO NOT implement your programs in multiple source files. Although C does allow you to build a project with subroutines scattered among multiple source files, you should only submit one source file for the C language.

   NO PLAGIARISM! You are free to design your own algorithm and code your own implementation, but you should not "steal" codes from your classmates or any other people. If you use an algorithm or code snippet that is publicly available or use codes from your classmates or friends, be sure to cite it in the comments of your program. Failure to comply will be considered as plagiarism.

7 Submission Guidelines

In Phase 1, you are required to submit your programs for the exercise, the object-oriented design, the report for the OO design and the receipt from VeriGuide. You are required to prepare your UML class diagrams and your report in PDF files. **In Phases 2 and 3, you are required to submit your code and report.** In addition, if you use your design with any modifications, you are required to submit your modified design also. Please read the guidelines CAREFULLY. If you fail to meet the deadline because of submission problem on your side, marks will still be deducted. So please start your work early! **Only one submission is needed for each group.**

1. In the following, **SUPPOSE**

   your group number is 1,
   your name is *Chan Tai Man*,
   your student ID is *1155234567*,
   your username is *tmchan*, and
   your email address is *tmchan@cse.cuhk.edu.hk*.

2. In each phase, the report should be submitted to VeriGuide, which will generate a submission receipt. The report and receipt should be submitted together with your source files in the same ZIP archive.

3. The OO design diagram should have the filename "design.pdf" and the report with "report.pdf". The VeriGuide receipt of report should have the filename "receipt.pdf". All file naming should be followed strictly and without the quotes.

4. In Phase 1, the C source should have the filename "VigenereCipher.c" and the Smalltalk source should have the filename "VigenereCipher.st". In Phase 2 and Phase 3, the C source should have the filename "JungleChess.c" and the Smalltalk source with filename "JungleChess.st" for your package and "JungleChessWS.ws" for the workspace. If you have

implemented the bonus part in Phase 3, you should name your source files with "BonusJungleChess.st" and "BonusJungleChessWS.ws".

5. For Phase 1, `tar` your source files to `group[num].tar` by

   ```
   tar cvf group01.tar VigenereCipher.c VigenereCipher.st \
   design.pdf report.pdf receipt.pdf
   ```

   For Phase 2, if you have not changed your design, `tar` your source files to `group[num].tar` by

   ```
   tar cvf group01.tar JungleChess.c JungleChess.st JungleChessWS.ws \
   report.pdf receipt.pdf
   ```

   Otherwise, `tar` your source files to `group[num].tar` by

   ```
   tar cvf group01.tar JungleChess.c JungleChess.st JungleChessWS.ws \
   design.pdf report.pdf receipt.pdf
   ```

   For Phase 3, if you have done the bonus part, `tar` your source files to `group[num].tar` by

   ```
   tar cvf group01.tar JungleChess.c JungleChess.st JungleChessWS.ws \
   BonusJungleChess.st BonusJungleChessWS.ws design.pdf report.pdf receipt.pdf
   ```

   Otherwise, `tar` your source files to `group[num].tar` by

   ```
   tar cvf group01.tar JungleChess.c JungleChess.st JungleChessWS.ws \
   design.pdf report.pdf receipt.pdf
   ```

6. `Gzip` the `tarred` file to `group[num].tar.gz` by

   ```
   gzip group01.tar
   ```

7. `Uuencode` the `gzipped` file and send it to the course account with the email title "Project Group *your Group Number*" by

   ```
   uuencode group01.tar.gz group01.tar.gz \
   | mailx -s "Project Group01" csci3180@cse.cuhk.edu.hk
   ```

8. Please submit your project using your Unix accounts.

9. An acknowledgement email will be sent to you if your project is received. **DO NOT** delete or modify the acknowledgement email. You should contact your TAs for help if you do not receive the acknowledgement email within 5 minutes after your submission. **DO NOT** re-submit just because you do not receive the acknowledgement email.

10. You can check your submission status at

    `http://www.cse.cuhk.edu.hk/csci3180/submit/project.html`.

11. You can re-submit your project, but we will only grade the latest submission.

12. Enjoy your work :>