CSCI3180 – Principles of Programming Languages – Spring 2013

Course Project — Sokoban Vexed

Phase 1 deadline: Mar 10, 2013 (Sunday) 23:59
Phase 2 deadline: Mar 31, 2013 (Sunday) 23:59

1 Introduction

In this project, we are to develop a puzzle game called **Sokoban Vexed**, which is a combination and evolution of two classical puzzles, i.e., Sokoban and Vexed. Sokoban is a classical sliding/pushing block puzzle, in which various boxes must be pushed to their destination in a twisted warehouse. The boxes cannot be pulled. When a box is pushed into a corner it is stuck, and eventually the puzzle will have to be restarted. Vexed, on the other hand, is a puzzle game in which you are required to move similar blocks together under the effect of gravity, which causes them to annihilate one another. Once all the blocks are gone, you have solved the level. In **Sokoban Vexed**, there is a human character and multiple boxes with various colors in the twisted warehouse. The gravity effect is considered and hence both boxes and human would fall unless they are stepping on something. The goal is to remove all the boxes from the warehouse. There are different versions of the game rules for Sokoban Vexed. In order to avoid ambiguity, we fix the game rules and they are covered in details in a later section. Figure 1 shows the warehouse and other elements of **Sokoban Vexed**.

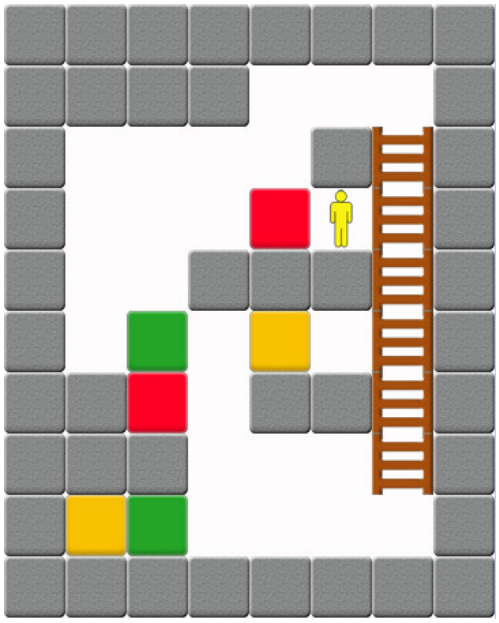


Figure 1: Sokoban Vexed

In this project, your are required to design and implement the Sokoban Vexed 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 Smalltalk in order to get familiar with the language and the development environment. The second task is to create an object-oriented design for your Sokoban Vexed.
2. Implement the Sokoban Vexed using C and Smalltalk by strictly following the game details.
3. Extend your program to suit our extra requirements and features for the Sokoban Vexed. (Details to be announced in Phase 3)
4. (Bonus, optional) Further extend your Sokoban Vexed. (Details to 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 Sokoban Vexed game.

2.1 Goal

In this project, your are required to implement the Sokoban Vexed that allows multiple levels of games. In each level, there are different elements and exactly one human character in the warehouse. A level is finished if all the boxes are removed and the program will load in the next level until all levels are finished. The size of the warehouse and positions of elements in each level is characterized in a text file (map file), which can be read into the program. The specification of input map file is given in a later section. The goal is to complete all levels in the Sokoban Vexed.

2.2 Elements in the Warehouse

The warehouse is built based on a map, which is composed of a customized $M \times N$ grids. There are four basic types of elements, i.e., human (Figure 2a), rock (Figure 2b), ladder (Figure 2c) and box (Figure 2d). Each of them occupies one grid. In each level, there is exactly one human in the warehouse and the human character is controlled by the game user. There are at most seven different colors for boxes, i.e., BLACK, RED, GREEN, YELLOW, BLUE, MAGENTA and CYAN.

(a) Human (b) Rock (c) Ladder (d) Red Box

Figure 2: Elements in the warehouse

2.3 Game Rules

Rocks and ladders are not movable, while human and boxes are movable. The movement of human is controlled by the game user and the movement of box is made by the human character. The following rules govern the movements and effects of the human and boxes.

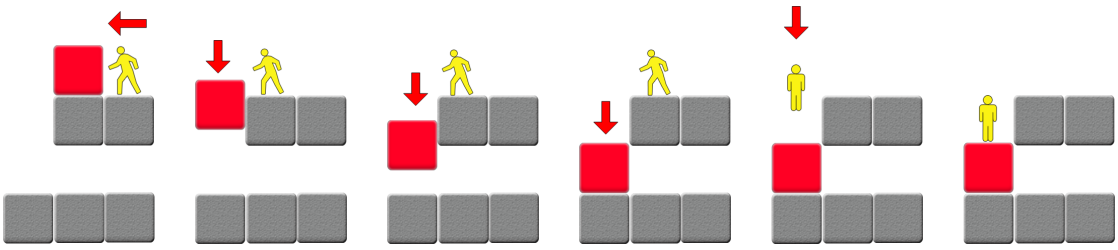
Gravity

The gravity effect applies to the human and boxes along the vertical direction.

- **Both the human and boxes will fall if they are standing on nothing.**
- **Box will fall if it is placed on ladder.**

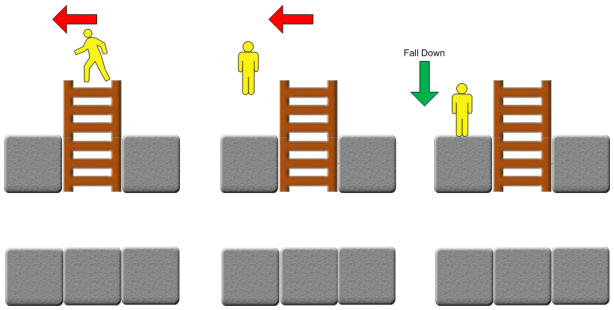
The human and boxes will fall until the above conditions are not satisfied. For example, the human is pushing a box toward a cliff (Figure 3a). The box starts to fall when there are no rocks below it (Figure 3b) and continues to fall (Figure 3c) until it lands on the rock (Figure 3d). During the box falling, the game user cannot control the human to move. After the box lands, if the human continues to go one step forward, he falls due to gravity too (Figure 3e). This time, the human lands on the box that he pushed in the last step (Figure 3f).

It should be noted that gravity has the different effects to boxes and the human when they are on the ladder. The human can stand on a ladder but boxes do not. For example, in Figure 4a, the human does not fall when he is standing on a ladder but he falls when he walks out of the ladder. If the human pushes a box onto a ladder, the box will fall due to the gravity while the human can still stand on the ladder (Figure 4b).

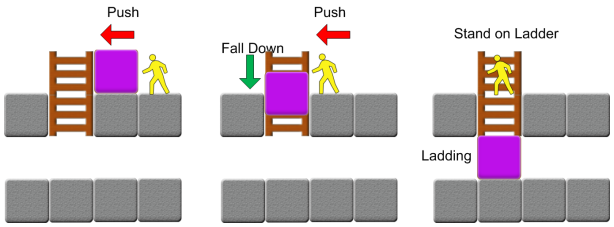


(a) Push the box (b) Box falling (c) Box falling (d) Box landing (e) Human falling (f) Human landing

Figure 3: Gravity effects



(a) Human falls when walking out of the ladder



(b) Box falls off the ladder

Figure 4: Gravity effects on ladder

Box movement

A box can only be moved by gravity or the human but not by the game user directly. The movement of boxes made by human is along the horizontal direction. **A box is *movable* to the left (right) by the human if (a) it is placed on rocks or boxes and (b) there is no rock or box on its left-hand (right-hand) side grid.**

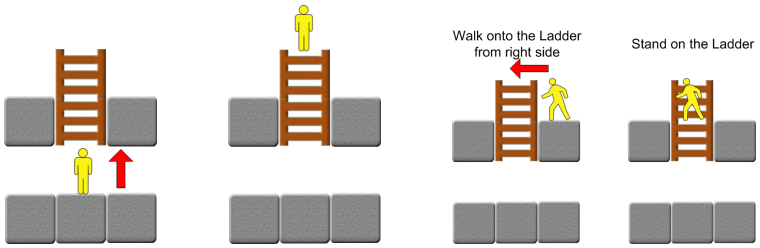
Human Movement

When the human is standing on a rock or box, he can only move horizontally:

- **If there is no rock or box to the left (right) of human, he can move one step left (right).**
- **If there is a rock to the left (right) of human, he cannot move left (right).**
- **If there is a box to the left (right) of human and the box is not movable to the left (right), he cannot move left (right).**
- **If there are boxes or the human himself falling due to gravity, the game user cannot move the human character.**

When the human is standing on or under a ladder, the human also can move up and down vertically. Once the human walks out of the ladder, he cannot move vertically anymore. There are several special cases we need to clarify:

- **If the human stands below a ladder, he can still climb up (Figure 5a).**
- **If the human climb to the top of a ladder, he can still stand on it and move down (Figure 5b).**
- **If the human walks onto a ladder from left side or right side, he can still stand on the ladder and move up/down provided that there are no boxes/rocks blocking the way (Figure 5c).**



(a) Start to climb up the ladder

(b) Stand on the top of the ladder

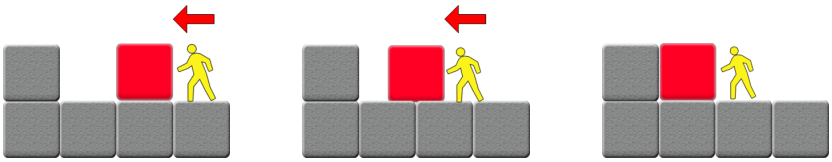
(c) Walk onto the ladder from one side

Figure 5: Human movements on ladder

Pushing the Box

To push a box, the human should stand just next to the box. If the box is movable to the left (right), the human should go to the right (left) grid of the box and move one step left (right) (Figure 6a). Then both the box and human will be moved one step left (right) (Figure 6b). Both box and human stop when they all move one step and reach the next grid respectively (Figure 6c).

Here are some more examples. The box cannot be pushed to a direction if it is blocked by either a rock or another box in that direction (Figure 7). The box can be pushed if there is one or more boxes stacking on it. If the box is pushed one step under this condition, these stacked boxes will stack on the human instead. When the human moves one step forward, these stacked boxes will fall on the ground (Figure 8).

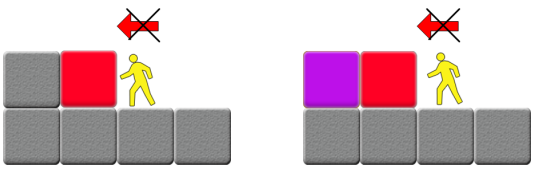


(a) Start pushing the box

(b) Pushing

(c) Finish pushing the box

Figure 6: Pushing boxes



(a) The box is blocked by rock

(b) The box is blocked by another box

Figure 7: Forbidden box movements

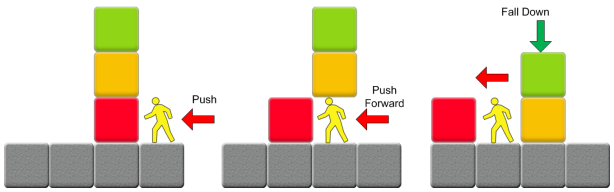


Figure 8: Push a stack of boxes

Box Vanish

Two or more boxes of the same color will vanish at the same time if they are next to each other in either horizontal or vertical direction. The boxes can vanish only under the condition that there is no activity in the warehouse. To be specific, the boxes won't vanish if the human or any box is falling down due to gravity. And all the boxes should be removed at the same time. For example, in Figure 9, the two yellow boxes will not vanish since one of them is still falling down.

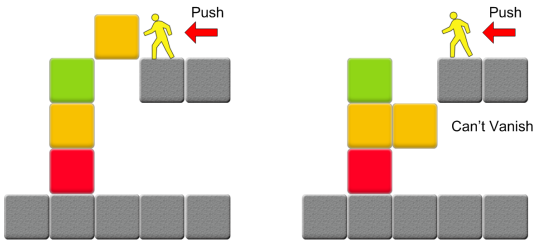


Figure 9: Boxes cannot be annihilated when some boxes are falling due to gravity

3 Phase 1

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

3.1 Programming Exercise

You have to finish the Tic-Tac-Toe game using Smalltalk. The details of the exercise is covered in the 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 the 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 the description of your design. The description must state clearly the purpose of each class in the UML diagram and the purpose of each method. In addition, the description must describe the relationships between classes, and among methods in your UML diagram.

4 Phase 2

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

4.1 Input Map File

The program should read the map files to get the initial elements distributions of all levels. The number of levels N in the game should be input as a command line argument. The filenames of these N levels should be hardcoded as `map1.txt, map2.txt, ..., mapN.txt` and these map files should be put in the folder `maps/` under the directory of your program. The map file shown in Figure 10 characterize the game shown in Figure 1. The first line is the size of the warehouse. In this example, "8 10" means the horizontal length of warehouse is 8 and vertical length of warehouse is 10. We use "`P`", "`#`", "`H`" to represent Human, Rock, Ladder and "`b`", "`r`", "`c`", "`m`", "`g`", "`y`" and "`l`" to represent blue, red, cyan, magenta, green, yellow and black boxes. Each line in an input map file must be ended with '\r\n'.

```
8 10
########
####   #
#    #H#
#   rPH#
#  ###H#
# g y H#
##r ##H#
###   H#
#yg    #
########
```

Figure 10: Map

You are required to design 5 different playable map files for submission. A playable map is composed of the four elements (exactly one human character) and there is at least one possible way to remove all boxes in the warehouse. A game design competition will be held during project phase 1 and 2. Each group can pick one map design for the competition and the three designs that get highest votes will receive bonus marks in the project phase 2. To participate in the competition, you have to choose one out of your five submitted maps and capture the initial state of the chosen warehouse. Your figure should look like Figure 1 and other components in the GUI (such as buttons and message prompting area) should not be captured. The figure must have height less than or equal to 400 pixels and name `GroupXX.jpg` where XX is your group number (e.g. Group 1's figure should be named `Group01.jpg`). Any group that does not submit the above figure, or the submitted figure does not meet the above requirements, will not be considered in the competition. The details of the competition will be posted in the newsgroup.

4.2 Input Control Sequence File

Your program should be able to read a control sequence file for controlling the movements of the human. The control sequence file can be loaded in the program anytime during each level. The file should consist of a sequence of characters "`r`", "`l`", "`u`", "`d`" each separated by a space character, representing the instructions of letting the human to go right, left, up and down respectively. For example, "`l l l l r r r r r u u u l l l r d`" is a sequence of 17 steps. If the level is finished before all the steps in the control sequence file is performed, the remaining steps are discarded.

You have to submit the solutions of your 5 different playable map designs. For each map design, you have to submit one input control sequence file that solves the corresponding map (all boxes disappear after performing all the steps in the control sequence file). The five control sequence files should be named `sol1.txt, sol2.txt, sol3.txt, sol4.txt and sol5.txt`.

4.3 Reset Level

Your program should have the function to let game user to reset the current level anytime during the game. When a level is reset, all elements are positioned according to the original map file.

4.4 Animation Effects

In both C and Smalltalk implementation, the movement of human is grid-by-grid and extra animation effect is not needed in phase 2. However, when the human or boxes fall due to gravity, the movement should be grid-by-grid and a time delay should be added between each move so that the game user is able to see the human or boxes are dropping. A suggested time delay is 0.3 second. Similar time delay should be added to automated movements such as the movements made by input control sequence.

4.5 C Implementation

Please follow the requirements below in the C implementation.

User Interface

To make things simpler, you do not need to make the graphical user interface in the C implementation. However, you MUST use the **Ncurses** (**n**ew **curses**) library to write your text user interface. With Ncurses, you should be able to draw the warehouse and different kinds of elements in the warehouse with proper colors. You must strictly follow the format of each element in the game. Figure 11 shows the same game as in Figure 1 in a C implementation.

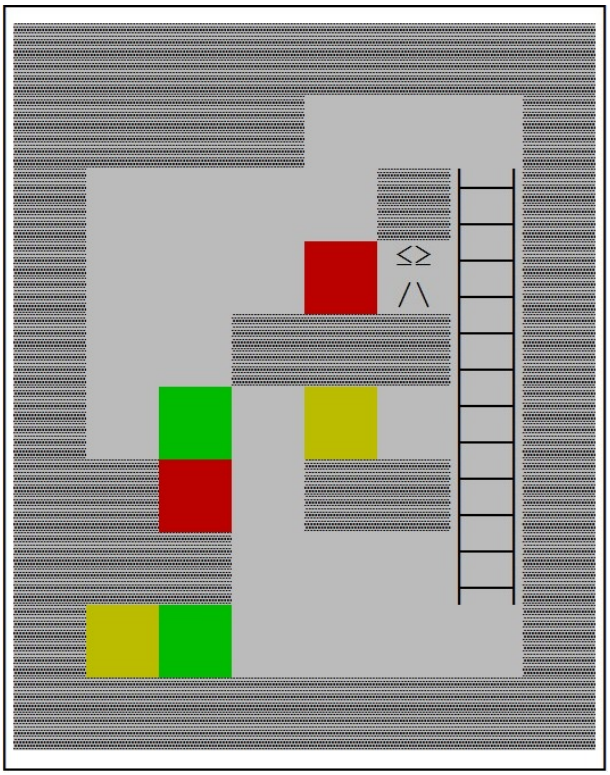


Figure 11: Game Interface for C Implementation

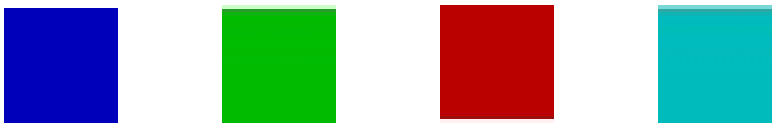
Figure 12 shows the four kinds of elements in the warehouse: **human**, **rock**, **ladder** and **box**. Each grid in the warehouse is composed of 2x4 characters. The characters for human, rock and ladder will be provided to you. We use different colors to represent different kinds of boxes and there are up to seven different colors for boxes. These colors are shown in Figure 13.

Input/Output Methods

Your C program should take one command line argument which specifies the total number of levels we have in the Sokoban Vexed game. During each level, the game user can press 'r' to reset the level and 's' to load input control sequence file by asking for the filename of the control sequence.

(a) Human (b) Rock (c) Ladder (d) Green Box

Figure 12: Four kinds of components in C implementation



(a) Blue (b) Green (c) Red (d) Cyan

(e) Black (f) Magenta (g) Yellow

Figure 13: Seven colors for boxes

Game user uses arrow keys (←↑↓→) to control the movements of the human character. The game board should be shown on screen and should be updated when there are changes.

4.6 Smalltalk Implementation

You MUST utilize the **Model-View-Controller (MVC)** paradigm in the 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.

Graphical User Interface

The program should have the similar GUI as shown in Figure 1. You MUST use the figures that are provided for you to make the warehouse. Since the size of the warehouse in each level may be different, the window size of your program should be dynamic and subject to change according to different input map files. You are **free to design** the other parts of the graphical user interface such as the message prompting area.

Input Methods

The command for starting the Sokoban Vexed game should accept an argument specifying the number of levels in the game. Your program should provide two methods for the game user to control the human character. One should be mouse-based and the other one should be keyboard-based. For keyboard-based input, it is the same as the C implementation and uses arrow keys (←↑↓→) to control the human character. Game user can also use the mouse to control the human character by clicking the destination grid. However, since the human can only move one grid each time, mouse clicking the grids other than those next to the human should not make any effect.

Extra Functional Requirements

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

1. A "Reset" button to let game user reset the level anytime during the game.
2. A "Load Sequence" button to ask for input control sequence file and perform the moving sequence in the file.
3. Provide a message prompting area to show the messages such as winning the level or game.

4.7 Report

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

1. Compare the advantages and disadvantages in implementing the Sokoban Vexed using C and Smalltalk. You may support your points of view regarding to some particular tasks in Sokoban Vexed Game.
2. Explain why you insist on 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 have changed and why.

5 Development Environment

Please test your program under the following environment 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. We suggest you to access you linux account remotely 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 the 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 14, you can export your package in the System Browser in VisualWorks by right-clicking your package, select File Out -> Package and save your package as "SokobanVexed.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 a "SokobanVexedWorkSpace.ws" file.

6 Other Requirements

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

1. **Division of Labour**
   You should divide the works in a way such that both of you participate in all phases, including the OO design and the implementation in 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.

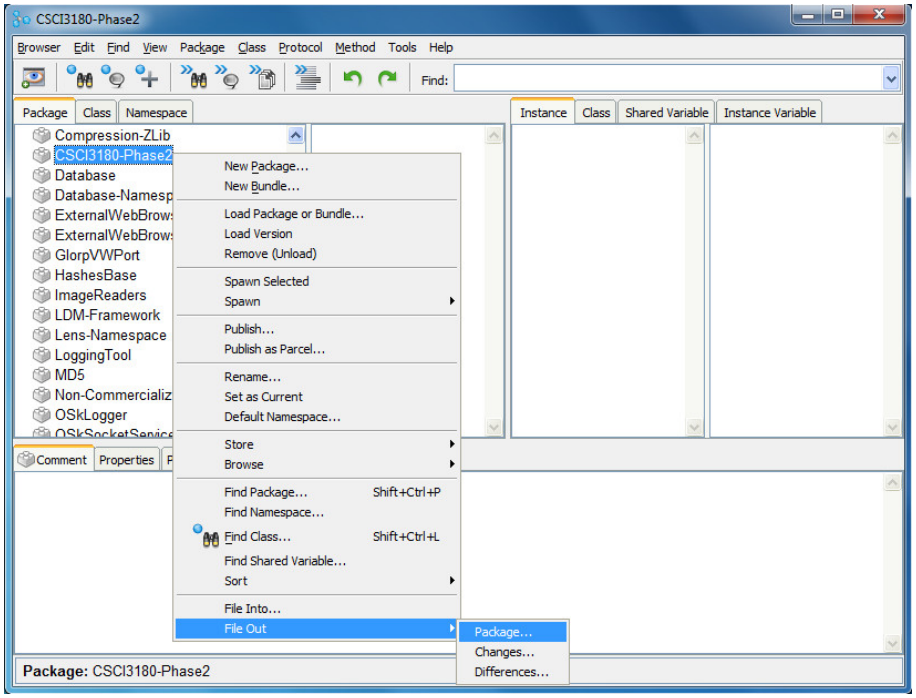


Figure 14: Export your source code from VisualWorks

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 & description and the receipt from VeriGuide. You are required to prepare

your UML class diagrams and the corresponding description in one single PDF file. **In Phases 2 and 3, you are required to submit your code and the report for the two questions.** 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. The pdf file containing your OO design and description should be submitted to VeriGuide, which will generate a submission receipt. The receipt should be submitted together with your OO design diagram and corresponding description, and Smalltalk codes for the exercise in the same ZIP archive.

3. The C source should have the filename "SokobanVexed.c" for Phase 2. The Smalltalk source should have the filename "TicTacToe.st" for exercise in Phase 1, "SokobanVexed.st" and "SokobanVexedWorkSpace.ws" for Phase2. The OO design diagram and description should have the filename "design.pdf" and the report for the two questions with "report.pdf". The map files should have the filenames "map1.txt", "map2.txt", ..., "map5.txt" and the corresponding control sequence files should have filenames "sol1.txt", "sol2.txt", ..., "sol5.txt". The VeriGuide receipt of OO design and report should have the filenames "receipt_design.pdf" and "receipt_report.pdf" respectively. All file naming should be followed strictly and without the quotes. To join the game design competition, you have to submit the figure of your design and it should have the filename "GroupXX.jpg" where XX is your group number (e.g. Group 1's figure should be named Group01.jpg).

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

   ```
   tar cvf group01.tar TicTacToe.st design.pdf receipt_design.pdf
   ```

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

   ```
   tar cvf group01.tar SokobanVexed.c SokobanVexed.st \
   SokobanVexedWorkSpace.ws map1.txt map2.txt map3.txt map4.txt\
   map5.txt sol1.txt sol2.txt sol3.txt sol4.txt sol5.txt report.pdf \
   receipt_report.pdf Group01.jpg
   ```

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

   ```
   tar cvf group01.tar SokobanVexed.c SokobanVexed.st \
   SokobanVexedWorkSpace.ws map1.txt map2.txt map3.txt map4.txt \
   map5.txt sol1.txt sol2.txt sol3.txt sol4.txt sol5.txt report.pdf \
   receipt_report.pdf design.pdf receipt_design.pdf Group01.jpg
   ```

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

   ```
   gzip group01.tar
   ```

6. `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
   ```

7. Please submit your project using your Unix accounts.

8. 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.

9. You can check your submission status at

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

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

11. Enjoy your work :>