CSCI3180 – Principles of Programming Languages – Spring 2013

Course Project — Magical Sokoban Vexed

Phase 3 deadline: April 14, 2013 (Sunday) 23:59

1 Introduction

Sokoban Vexed is a puzzle game in which a human character tries to move boxes with the same colors in a warehouse together, thereby making them disappear. To make this game more interesting, in Phase 3, you will extend the **Sokoban Vexed** you have implemented in Phase 2 to the **Magic Sokoban Vexed**, with extra requirements and features using C and Smalltalk.

(a) Harry rides a magic broom.

(b) bludger can hit Harry off the broom.

(c) Harry opens a wall with bricks using his wand.

Figure 1: Background images from the Harry Potter story.

The extended game is inspired from the famous novel and movie series "Harry Potter". In this magical world, there is a popular game called "Quidditch", in which the wizards ride on a flying broom and try to win the game by catching a "golden snitch" (see Figure 1a). There are also two jet-black iron balls, namely "bludgers" (see Figure 1b), in Quidditch. They are bewitched to fly around and try to knock players off their brooms. Harry also has a powerful wand, which can lift objects, or open a wall made of bricks (see Figure 1c).

In Phase 3, the human player becomes Harry Potter, who can fly with a magic broom and can cast a spell to lift boxes. Although the space of warehouse is filled with bricks, Harry can remove these bricks with his wand. We also have bludgers stacked inside the warehouse. The bludgers are not bewitched but may drop to knock Harry off and kill him if they are not stably placed. Please note that unless otherwise specified, every requirement in Phase 2 are still valid and you should follow strictly in developing the Magical Sokoban Vexed. Details of the **Magical Sokoban Vexed** is covered in later sections. In addition, you may optionally choose to implement the bonus part to get extra marks.

2 Game Details

Some game details are changed or added on top of those defined in the Phases 1 and 2 specification. Please note that except the changes we mentioned in this section, other rules or details remain the same as before. Here is the outline of the major extensions:

- The player becomes Harry with a magical wand.
- A new element, brick, that behaves like a rock, but can be made to disappear by Harry's magic.
- A new element, bludger, that vanishes when it has neighboring bludgers with the same color. Bludgers stacked on top of other bludgers are not stable and may fall.
- A new element, magic broom, with which Harry can get rid of the gravity and fly;
- Two new elements, i.e., lifting spells and magnets, with which Harry can cast a spell to make the magnet generate magnetic force to lift either a box or a bludger. There can only be at most three pairs of spell scroll and magnet in one level.

Following are optional features for bonus. You are encouraged to design new features using your own imagination. If these new features are interesting, you will be given extra bonus too.

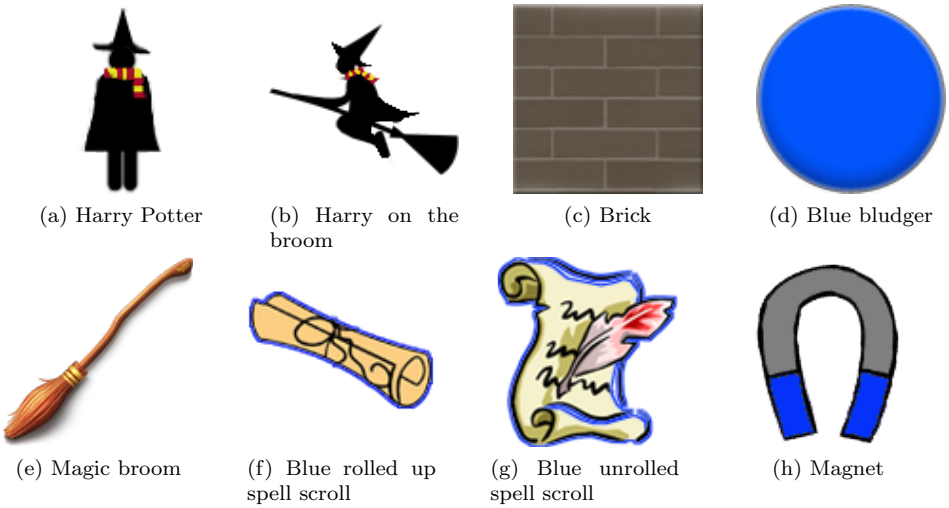
- A new element, portkey, on which Harry can be transmitted to another specific location;
- A new element, elevator, that can go up and down loaded with an object;

2.1 Goal

The goal is to survive and make all boxes and bludgers disappear. Note that, the unstable bludger may roll down and kill Harry. We will demonstrate the stability of bludger in following section.

2.2 Elements in the warehouse

The warehouse is similar to that in Phases 2 but with the following new elements:



(a) Harry Potter (b) Harry on the broom (c) Brick (d) Blue bludger

(e) Magic broom (f) Blue rolled up spell scroll (g) Blue unrolled spell scroll (h) Magnet

Figure 2: New Elements that are required in the warehouse.

As can be seen in Figure 2, each of these new elements occupies exactly one grid. For example, Figure 2c shows a brick. There are at most seven different colors for bludgers, including BLACK, RED, GREEN, YELLOW, BLUE, MAGENTA and CYAN. The example bludger in Figure 2d is in blue. There is only one magic broom in the game as shown in Figure 2e. The lifting spell is composed of two objects, one of which is a scroll containing the spell. The other one, as shown in 2h, is the magnet that generates magnetic force to pull other objects up.

Some elements are movable while some are not. We classify as follows:

- Movable elements: Harry Potter (with/without broom), box and bludger
- Immovable elements: rock, box, ladder, brick, bludger, broom, spell scroll (rolled-up or unrolled) and magnet

At the beginning of each level, each grid only has one of the following elements: Harry Potter without broom, rock, box, ladder, brick, bludger, broom, rolled-up spell scroll and magnet. That means, it is not possible to have two immovable elements in the same grid. After the level is started, the movable elements may move into a grid which already has another element. However, it should be noted that some elements can be passed through by other objects, while other elements cannot. To be specific, we clarify as follows:

- Harry Potter (with/without broom), box and bludger can pass through ladder and spell scroll.
- Box and Bludeger cannot pass through rock, magnet, brick and broom.

- Specifically, Harry can go into a brick and make it disappear (more details will be covered later in this section) and go into a broom and ride on it (so the broom disappears).
- Gravity only has affects on box, bludger and Harry Potter without broom.

2.3 Player

The human in Phase 2 becomes Harry Potter in Phase 3. At the beginning of each level, Harry Potter is not riding on the magic broom. Harry can deny gravity and fly after he gets hold of the magic broom. Without the broom, Harry can only walk and climb the ladder like the human in Phases 2. Harry (and Harry on the broom) can also make the bricks disappear. The rules of flying and breaking the bricks are demonstrated in following sections.

2.4 Game Rules

The bludgers are movable, while the rest of the new elements are not. Gravity only has affects on box, bludger and Harry Potter without broom. The movement of Harry is controlled by the game user. The following rules govern the effects of the new elements and features.

To make things clear, we prioritize different effects as follows. If any two or more effects happen at the same time, the one with the highest priority happens first. The details of these effects will be described in later sections.

```
magnet effect > gravity effect > bludger rolling-down effect > vanishing effect
```

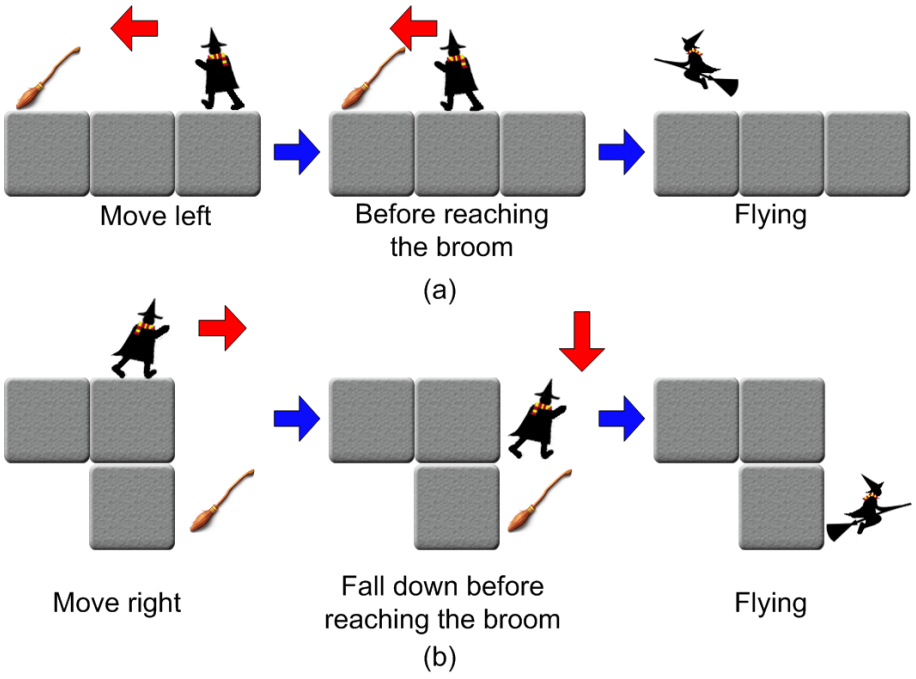


Figure 3: How Harry fetches the broom and flies.

2.4.1 Bricks

Bricks are immovable and they can be made to disappear. When Harry moves to a grid containing a brick, the brick element disappears. However, when other object tries to move to a grid containing a brick, the brick will not disappear and the object cannot move to the grid (that means, a brick is like a rock to objects other than Harry (and Harry on the broom). It is also important to note that, the brick only disappear when Harry moves to the grid containing a brick. Therefore, if Harry

can't make movements to the grid containing a brick, the brick won't disappear. For example, Harry without broom can stand on a brick and can't go down to break the brick. Also even when Harry falls onto a brick, the brick won't disappear since Harry doesn't make a movements to the grid.

2.4.2 Magic Broom

The magic broom floats in a specific grid before Harry gets it. The gravity effect does not apply to the broom. To fetch the magic broom, Harry needs to arrive to the grid that contains the broom. After Harry arrives, the broom element disappears since it is now ridden by Harry. Figure 3a shows such a process. After that, Harry rides the broom all the time until this level of game is over. It should be noted that, Harry can ride on the broom during he is falling down as long as he reaches the broom, as can be seen in Figure 3b.

For example, in Figure 4a, the broom is placed above a box when the game starts. The broom does not fall down even after the bottom box is moved away. Another example is shown in Figure 4b, wherein the broom is surrounded by bricks. After Harry removes the bottom brick, the broom still floats in the original grid. Based on this setting, you are encouraged to design some challenging maps for Harry to fetch the broom.

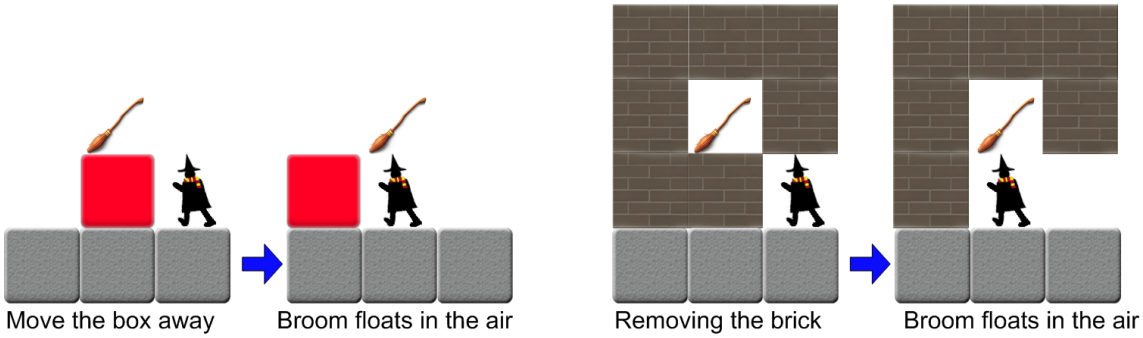


Figure 4: The broom floats in the air even when there is nothing below it.

With the broom, Harry can move freely without gravity constraint. The rules of movement defined in Phase 2 still apply to Harry except:

- Harry can move one grid at a time in all four directions: up, down, left and right.
- When Harry flies through a ladder, he still keeps flying, instead of climbing.
- Harry cannot fly into a grid containing a rock.
- Harry can still push boxes when he is flying. The rules of pushing boxes defined in Phase 2 still apply.

2.4.3 Bludger

In the beginning of the game, all the bludgers must be stable. Thus, Harry must have done something to trigger the bludgers to be unstable. These actions include but not limited to:

- Harry pushes a bludger to a position that is not stable.
- Harry pushes a box away or removes a brick that makes some bludgers unstable.
- Some boxes or bludgers vanish, which makes some bludgers unstable.

The movement rule, pushing rule, vanish rule and gravity effect of a bludger are the same as a box except that bludgers are round and hence unstable if we do not put it on a plane, rendering the bludger to possibly roll down. We define the stability condition and how a bludger can roll down as follows:

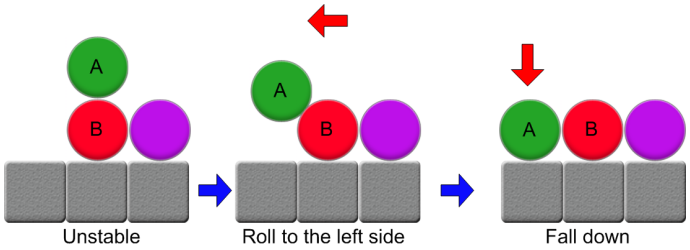


Figure 5: The bludger is unstable and rolls down.

- A bludger is stable when it is on top of a box, a rock, a brick or even on top of Harry.
- A bludger (e.g., A) is unstable when 1) it is on top of another bludger (e.g., B), and 2) both A and B have empty neighbouring grids on one (e.g., left) side. Consequently, A will roll down towards the left side. As shown in Figure 5, since B's left neighbouring grid is empty, bludger A is not stable and rolls towards the grid on the left side of B. If it happens that both A and B have empty neighbouring grids on left and right sides, A rolls down towards the left side.
- It is possible that more than one bludger is unstable. In that circumstance, the lower bludger rolls down before the higher bludger. As can be seen in Figure 6, both bludger A and B are unstable after the boxes vanish. However, B rolls down earlier than A due to its lower position. After a bludger (e.g., B) rolls down, it can alter the stability of other bludgers. In the above example, bludger A becomes stable after bludger A drops down. On the contrary, it can also change other stable bludgers to be unstable and trigger them to fall down successively, forming a chain reaction. Figure 7 shows such an example. Bludger A rolls down first due to the vanishing boxes, which changes another bludger (e.g., B) to become unstable. Thus bludger B starts to roll down too. The same effect applies to bludger C, causing a series of rolling down effects.

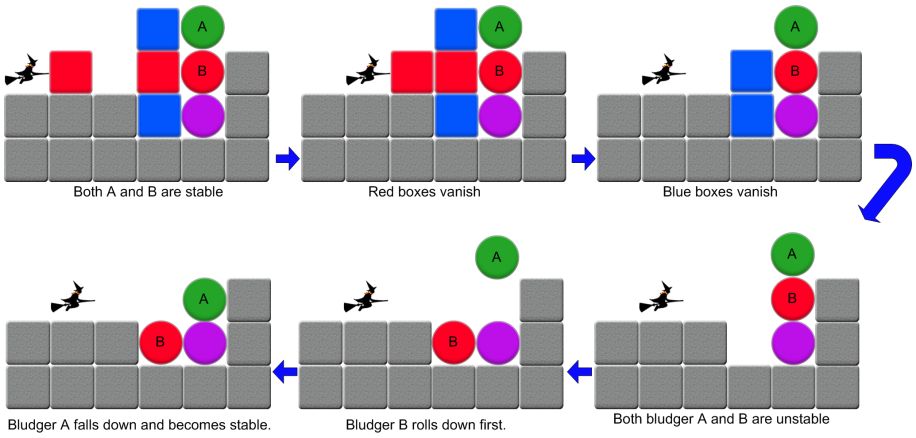


Figure 6: The lower bludger rolls down first and possibly changes other bludgers' stability.

- Every time a bludger (e.g., A) rolls down, it keeps rolling down until it is back to a stable condition again. As shown in Figure 8, Harry pushes bludger A to the right, rendering bludger A unstable. As A rolls down towards right and then falls on top of bludger C, it is still unstable since the grid on the right of C is empty. Hence, A keeps rolling down until it reaches to the bottom.
- In some special cases, two bludgers may try to roll down towards the same grid. Taking bludgers A and B in Figure 9 as an example, we compel the bludger on the left to roll first. In that case, A rolls down first and fill the grid, rendering bludger B stable again.

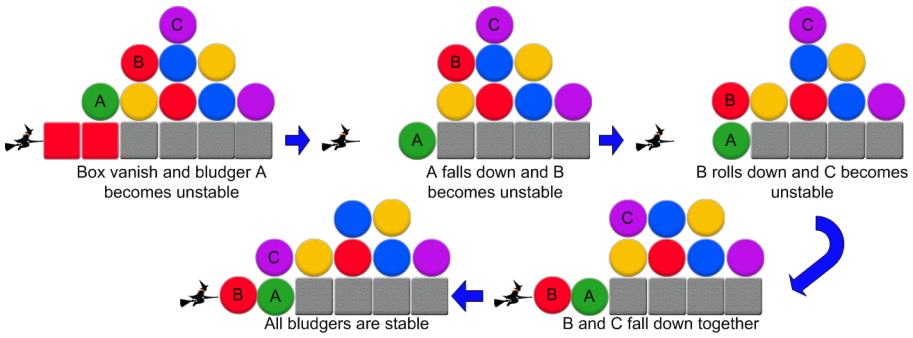


Figure 7: The chain reaction of rolling down effects.

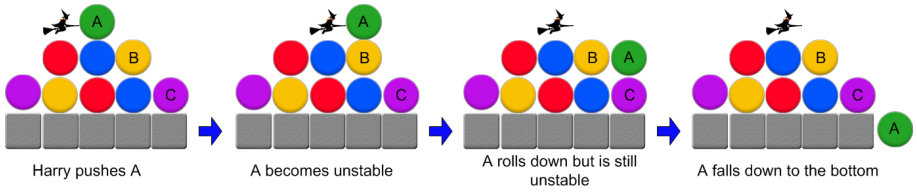


Figure 8: The chain reaction of rolling down effects.

- In some special case, some bludgers are about to fall down due to the gravity effect while other bludgers are about to roll down due to unstable conditions. Under this circumstance, gravity effect has the higher priority, which means the falling down action is taken before the rolling down action. Figure 10 shows such an example.

A falling/rolling bludger can kill Harry if he is hit by a falling bludger from above. It should be noted that, on one hand, Harry is killed only after the bludger begins to fall. For example, in Figure 11, Harry is floating besides bludger B and bludger A is stable. Once Harry leaves that position, the bludger A becomes unstable and begin to roll down. If Harry moves downwards, unfortunately, he will be knocked by the falling bludger and get killed. On the other hand, however, if the bludger is on top of Harry, the bludger is stable and will not fall, neither will kill Harry. When Harry moves aside, the bludger rolls down but won't hit Harry. The game is over as long as Harry is killed. You can show a "GAME OVER" message and let game user to reset this level.

2.4.4 Lifting Spell

A lifting spell has two components, i.e., the spell scroll and the magnet, each of which is immovable and occupies one grid. In each level, there can be at most three pairs of spell scrolls and magnets, with each pair colored in red, blue or green. A spell scroll controls the magnet in the same color (e.g. red spell scroll controls red magnet). As shown in Figure 12, initially, the spell scroll is rolled up and the magnet is turned off. When Harry walks or flies to a spell scroll, the scroll is unrolled and the lifting spell is cast, resulting in the magnet generating a magnetic force to lift the first box/bludger below it up. The effectiveness of the lifting spell is preserved even when Harry leaves the spell scroll. However, when Harry walks/flies back to the unrolled spell scroll again, the spell is uncast, rendering the spell scroll to be rolled up and the lifted object falls down again. It should be noted that only Harry can trigger the spell scroll effect. Moreover, as long as Harry can reach the grid containing the spell scroll, no matter by walking, flying or falling, the lift spell can be trigged.

The movement rules of an object that are lifted by the magnet are listed as follow:

- If there is something (e.g. rock, brick) between the magnet and the box/bludger, the box/bludger will not be lifted by the magnet.
- If there is nothing between them, the box/bludger is lifted and moves up until it reaches the magnet.

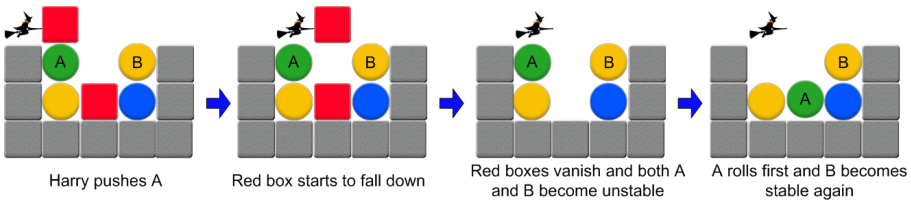


Figure 9: We compel the left bludger has a higher priority to roll down.

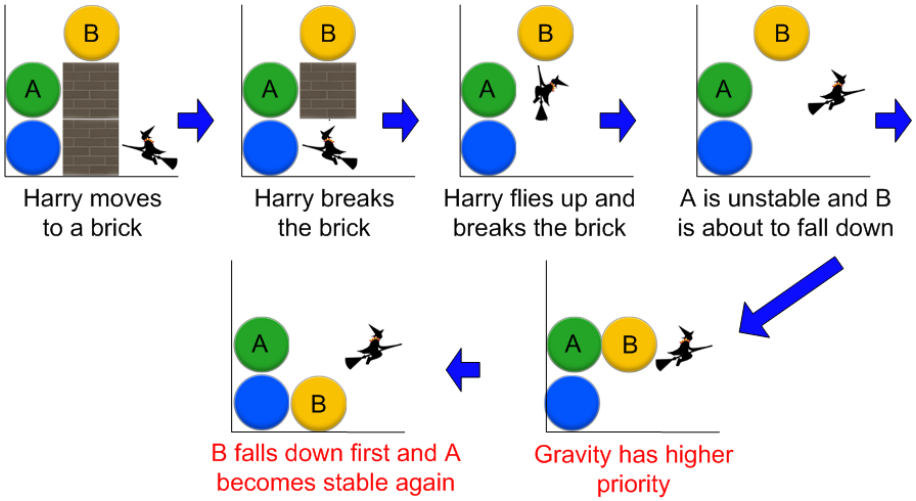


Figure 10: Falling precedes Rolling.

- If the box/bludger reaches the magnet, it becomes stable, which suggests that vanishing can happen if there is a box/bludger with same color next to it.

3 Miscellaneous Requirements

This section specifies the requirements in animation and the details of C and Smalltalk implementations.

3.1 Input Map File

In addition to the characters for input map file that are defined in Phase 2, here is the set of characters representing new elements of phase 3. We use 'P' to represent Harry Potter (instead of Human) and 'B', 'R', 'C', 'M', 'G', 'Y', 'L' to represent BLUE, RED, CYAN, MAGENTA, GREEN, YELLOW and BLACK bludgers.

Element	Character
Broom	'-'
Brick	'D'
Red Scroll	a
Red Magnet	A
Blue Scroll	v
Blue Magnet	V
Green Scroll	o
Green Magnet	O

Same as in Phase 2, you have to provide 5 map files and the corresponding control sequence files as the solutions of the 5 provided maps. The provided games must be playable. Please name the map files `map1.txt, map2.txt, ..., map5.txt` and the corresponding control sequence files `sol1.txt, sol2.txt, ..., sol5.txt`.

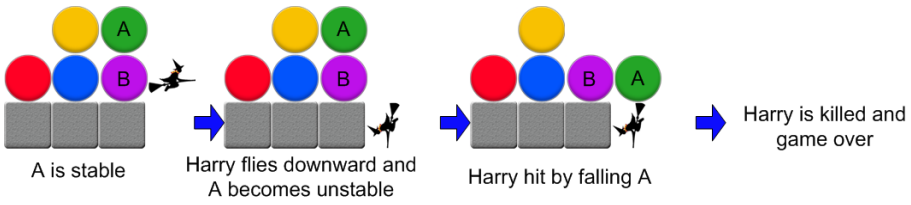


Figure 11: Harry killed by a falling bludger.

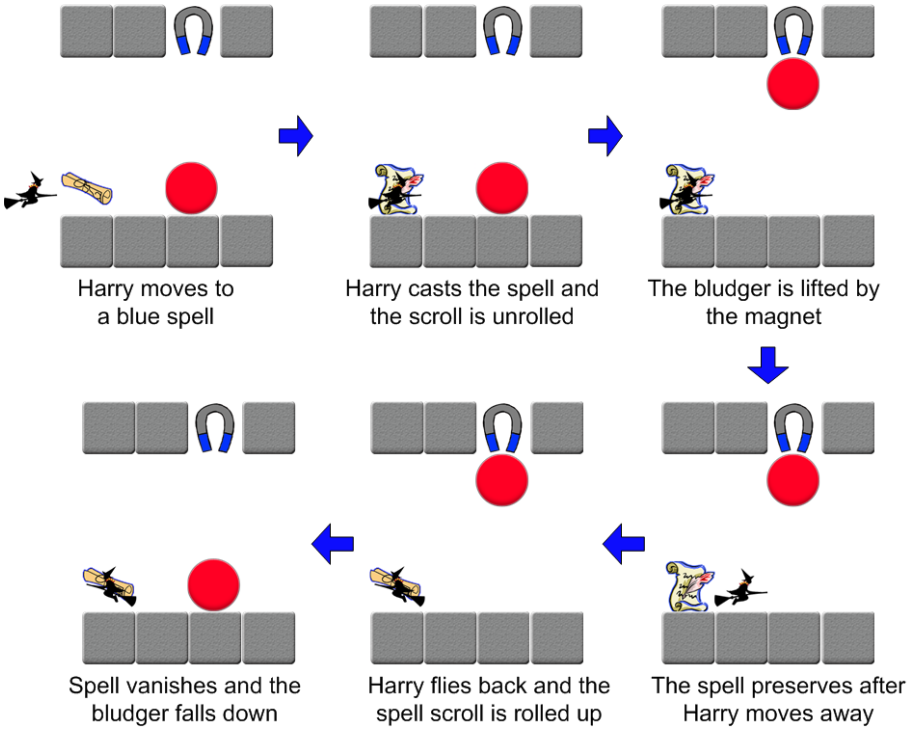


Figure 12: How the lifting spell is cast and vanish.

We provide you 6 sample maps and the corresponding control sequence files (may have some redundant control for testing purpose). You only need to consider the cases that are stated in the specification or in the six sample games. The sample games give you the idea of what we are going to test (Certainly, we will have different test cases during project demonstration). Thus, you do not need to think of every possible situations which are not described in the specification or the sample games. If you find a case which is not described in the specification or the sample games, you can just find a reasonable way to handle it. The maps and control sequence files can be found in the course homepage. We also provide you the corresponding videos for each sample game. You may refer to the videos and see what the games should look like. We specify the possible purpose of testing and the priority issues for each map as follows:

- map 6: Test the properties of the broom and the brick.
- map 7: Test the property of lifting spell. Note that the magnet effect has higher priority than gravity effect.
- map 8: Test the property of bludgers with cases that are similar to what are described in figure 6, 9, 10. Note that the gravity effect has higher priority than rolling down of unstable bludgers.
- map 9: Test the property of bludgers with cases that are similar to what are described in figure 8. Note that the rolling down of bludgers has higher priority than the vanishing effect.

- map 10: Test the property of bludgers with cases that are similar to what are described in figure 7. Note that the rolling down of bludgers has higher priority than vanishing.
- map 11: Test the property of bludgers with cases that are similar to what are described in figure 11.

3.2 Animation Effects

Concerning the quality of the game, we expect you to display the object movement smoothly in the Smalltalk implementation. To fulfil this requirement, you should display the object multiple times during its movement from one grid to the next. Moreover, you have to use the pictures we provide to display animation of Harry walking, climbing and flying. The animation of Harry's movements work as follows. For simplicity, we take the walking animation in Figure 13a as an example. When Harry wants to walk one grid to the left, he is standing on the original grid initially. This one grid movement should go through two intermediate actions. Finally, the human should be standing on the destination. Walking one step right is similar. As shown in Figure 13b, when Harry is climbing up or down the ladder, there are also two intermediate climbing actions between two grids. Similar principle applies to the animation of flying on the broom, as can be seen in Figure 13c. Moreover, there should be some time delay between some automated movements such as dropping of elements due to gravity and movements made by input control sequence so that the game user can see the movements clearly. A suggested time delay is 0.3 second. In addition to Phases 2, a vanishing animation is required in phase 3 as shown in Figure 14.

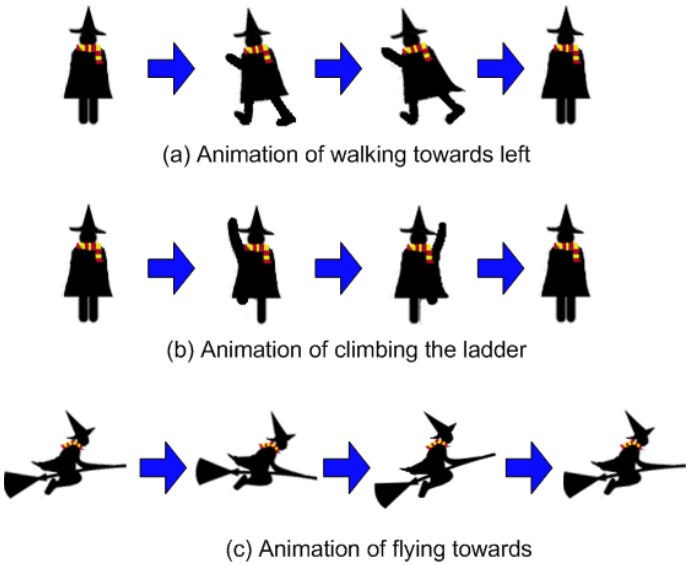


Figure 13: Illustration of implementing the animation effect.

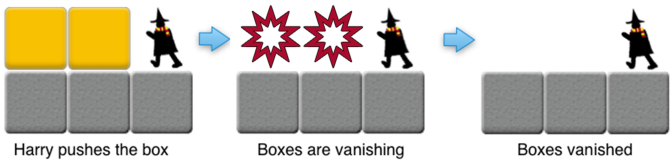


Figure 14: Illustration of the vanishing animation.

3.3 C Implementation

You have to extend your C program in Phase 2 in order to include the extra requirements in Phase 3. The I/O methods are the same as specified in the Phase 1 and 2 specification. You have to use

Ncurses to make the text user interface. Although the I/O methods are not changed, the game board of this game is enhanced to catch for the new elements, as shown in Figure 15.

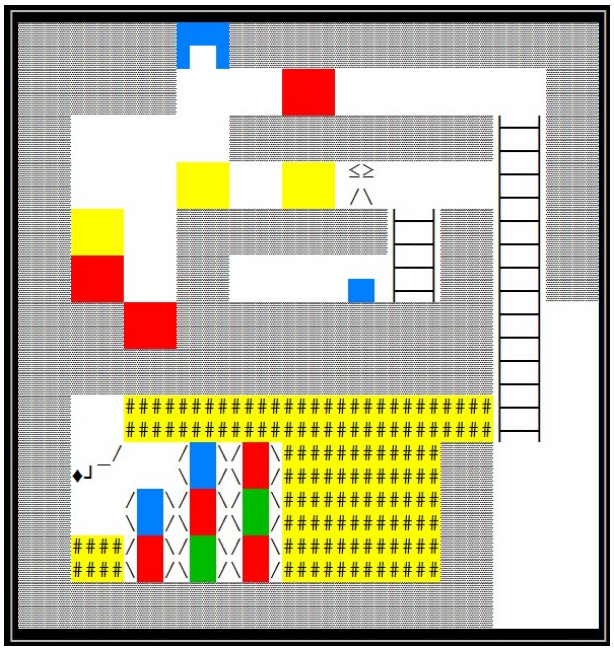


Figure 15: Game Interface for C Implementation

Besides the four common elements from Phase 2 (human/Harry without broom, rock, ladder and box), some new elements are added: Harry with broom, magic broom, brick, bludger, spell scroll and magnet. Figure 16 shows these new elements. Each grid in the warehouse is composed of 2x4 characters. Figure 17 shows the status when Harry (or Harry on the broom) moves onto some other elements.

(a) Harry on the broom (b) Magic broom (c) Brick (d) Red bludger (e) Blue rolled up spell scroll (f) Blue unrolled spell scroll (g) Blue magnet

Figure 16: New elements in the C implementation

(a) Harry on unrolled spell scroll (b) Flying Harry on unrolled spell scroll (c) Harry on rolled up spell scroll (d) Flying Harry on rolled up spell scroll (e) Harry on ladder (f) Flying Harry on ladder

Figure 17: Other status in the C implementation

3.4 Smalltalk Implementation

Similar to the C implementation, the I/O methods remain unchanged in the Smalltalk implementation. You have to draw the game board and the initial element by strictly following the new features.

4 OO Design and Report

Since you must enhance your OO design in order to incorporate the extensions in Phase 3, you have to update your UML class diagram of Phase 2 and submit the class diagram for the Magical Sokoban Vexed. This time, you only need to write up a much simpler report to answer the following questions:

- Compare the advantages and disadvantages in extending the Sokoban Vexed in Phase 3 using C and Smalltalk. You may discuss it with respect to the specified extension tasks.
- Briefly explain what changes you made to your OO design of Phase 2 to incorporate the extensions in Phase 3 and why.

5 Bonus

You are free to design and expand your Magical Sokoban Vexed implemented in Phase 3. **Please note that the bonus part is only needed for the Smalltalk implementation.** You do not have to implement your bonus part in C. You can get **at most 20 extra points** on top of Phase 3. **If you decide to implement the bonus part, please do it on a new copy of your Phase 3 implementation so that you can keep your Phase 3 implementation clean for submission.** Besides, you do not need to draw the OO design for your bonus part. The marks will be based on creativity, rationality, difficulties and completeness of the added functionalities. We provide two possible features as options below. You are encouraged to design some new and interesting features for the bonus part.

- **Portkeys:** a portkey contains a pair of two identical objects, each of which occupies a grid. A portkey can only transmit Harry, a box or a bludger from one portkey to the other. If an object moves to a portkey (for simplicity, we name this portkey object as the entrance), the object disappears from the current position and appears again in the grid that the other portkey of the pair occupies (we name it the exit), provided that there is nothing occupying the exit grid. In addition, a pair of portkeys is bidirectional, which means each portkey can serve as both entrance and exit. After transmitting, an object will just remain in the exit unless some other effects are triggered. For your convenience, we provide some figures as portkeys (as shown in Figure 18). You are free to use other figures as portkeys.
- **Elevator:** the elevator can move vertically, i.e., ascend and descend between two grids that are aligned horizontally, named "Top grid" and "Bottom grid" respectively. The elevator is in the "Bottom grid" by default. Harry can move into the elevator, or pushes a box/bludger into the elevator. As long as there is an object inside the elevator, it begins to ascend until it reaches the "Top grid". It continues staying in the "Top grid" as long as it contains an object. In other words, only when there is nothing inside of the elevator, for example, Harry moving out of the elevator, or pushing the object out, or a box/bludger vanishing, the elevator descend to the "Bottom grid" again. For your convenience, we provide a figure as elevator (as shown in Figure 18d). You are free to use other figure as elevator.

(a) portkey 1 (b) portkey 2 (c) portkey 3 (d) elevator

Figure 18: Suggested new elements for bonus part

6 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 in other development environments.

6.1 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.

6.2 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. 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. If you decided to implement the bonus part, please create a new package for it and keep a clean copy for your extended Sokoban Vexed. Please name the files of the package and workspace of the bonus part by "BonusSokobanVexed.st" and "BonusSokobanVexed.ws" respectively.

It should be noted that we allow at most **three** the portkey pairs in each level. If there are multiple pairs of portkey, the object can only be transmitted between the same portkey.

7 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.

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.

8 Submission Guidelines

In Phase 3, you have to submit the files listed in the following. Note that this time you have to put the object-oriented design & description and report for the two questions in one single pdf file. Please submit this pdf file to **VeriGuide Assignment 6** and sign the receipt for submission.

- Smalltalk source code files: **MagicalSokobanVexed.st** and **MagicalSokobanVexedWorkSpace.ws**
- C source code: **MagicalSokobanVexed.c**
- The OO design & description and report for the two questions in one file: **report.pdf**
- The VeriGuide receipt file of report.pdf: **receipt.pdf**
- Input map files: **map1.txt, map2.txt, map3.txt, map4.txt and map5.txt**
- Control Sequence files: **sol1.txt, sol2.txt, sol3.txt, sol4.txt and sol5.txt**
- Any extra files that you will need during project demonstration

If you have done the bonus part, please also submit the following files:

- Smalltalk source code files: **MagicalSokobanVexedBonus.st** and **MagicalSokobanVexedBonusWS.ws**
- The extra figures files needed in your extended program. You do not have to submit the provided portkeys and elevator figures if you use them.

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

   ```
   tar cvf group01.tar MagicalSokobanVexed.c MagicalSokobanVexed.st \
   MagicalSokobanVexedWorkSpace.ws map1.txt map2.txt map3.txt map4.txt \
   map5.txt sol1.txt sol2.txt sol3.txt sol4.txt sol5.txt report.pdf receipt.pdf
   ```

   If you have extra files or have done the bonus part, please include also the needed files.

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

   ```
   gzip group01.tar
   ```

4. `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
```

5. Please submit your project using your Unix accounts.

6. 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.

7. You can check your submission status at

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

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

9. Enjoy your work :>