The University of Queenland, School of ITEE
CSSE2002/7023 — 2018
Assignment 3 (15%)
Due: 19 October, 12 **Noon**
Revision: 1.0 (as at October 5, 2018)

Introduction

The goal of this assignment is to create a GUI which utlises classes developed in assignment 1 and assignment 2. The GUI will display the game world and have options for users to interact with the world. For this assignment, the only supplied code will be the classes implemented in assignment 1 and 2, no other code will be provided to create the interface.

Unlike previous assignments however, you are allowed to create public classes with public methods as required. Keep in mind that any methods that dont need to be public should be made private and all good OO practices must be followed.

Language requirements: Java version 1.8, JavaFX, JUnit 4
Restrictions: use of FXML is not permitted
Helpful Resources:

- Week 10 JavaFX prac
- Getting started with JavaFX:
  `https://docs.oracle.com/javafx/2/get_started/jfxpub-get_started.htm`
- Working with JavaFX layouts:
  `https://docs.oracle.com/javafx/2/layout/builtin_layouts.htm`

Context

Now that we have developed a set of primitives to represent Tile, Block, and Builder and a way to create worlds and interact with it, we want to be able to:

- launch a game window,
- load world maps from the game window,
- display the loaded map,
- perform actions on the world by using control elements in the game window,
- provide helpful messages to users on any failures,
- display the updated world map once an action is performed, and
- save the updated world map.

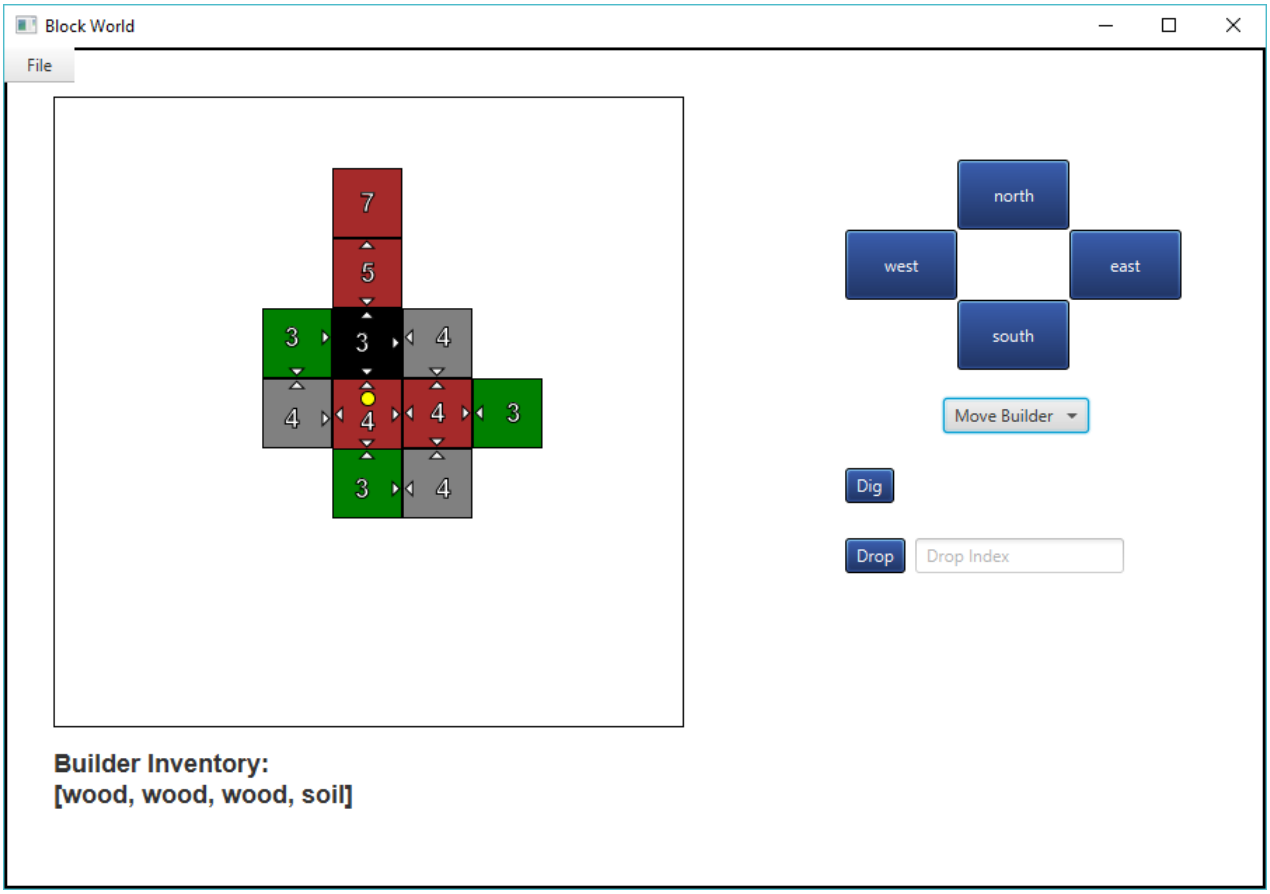


Figure 1: Game window of block world, with a small map already loaded

In this assignment you will create something similar to the above GUI, **it does not need to be exactly same as shown above** but needs to have the same functionality.

The required features of your GUI application are:

File Menu

A file menu which allows users to load and save world map files. Selecting one of “load” or “save” option opens a file dialog box and allows users to select a file or save a file. If a map is already loaded into the application, users should be able to load new maps without restarting the program.

Display Map

There should be some area in the window which displays the current tiles. The **minimum** grid dimensions for the display are 9x9 (at least 81 tiles should be visible, if a map has that many). The above example GUI is a 9x9 display, with the builder(yellow circle) always center in the grid. You can create the dispaly using any method you like (canvas, grid layout, etc.) however the following must always being displayed:

- at least 9x9 tiles, with **Builder** on one of those tiles,

- total blocks on the tiles,
- exits on the tiles,
- builder's current inventory, and
- disply at least the top block on the tiles. You can use images or different colors to show what the top block is.

Action buttons

There should be some way to perform actions on the map. You can use buttons or any other JavaFX controls as you please to allow users to perform the actions from Assignment 2.
"DIG", "DROP", "MOVE_BUILDER", "MOVE_BLOCK"
There should also be some way to provide directions for the move actions and a way to select the index to drop from builder's inventory.
If no map is loaded then these controls should either be disabled or when used, should let the user know that no map has been loaded (See **AlertBoxes on errors**).

You are free to create the controls however you like, but everything must be documented (see **Example Description**)

AlertBoxes on errors

Users may try to undertake actions which are impossible, or which fail. Your program should respond accordingly. In some cases, you will pre-empt the user; for example, you may disable direction buttons if there is no exit in that direction. e.g. If builder's current tile does not have a "north" exit, then the north button can be disabled. If a user tries to perform an impossible action, or there is some failure in your program, you should alert the user with alert boxes.

Relevant alert boxes may include, but are not limited to:

- map is successfully loaded
- map cannot be loaded
- cannot move builder in the intended direction
- cannot move block in the intended direction
- cannot place chosen block

Doocumentation

You must provide documentation for your GUI. This should include an annotated screenshot of your GUI, indicating what each section of each screen does. You should also provide a short description of how each feature has been achieved (e.g. pressing the button that says "DIG" makes the Builder dig on their current tile)

Documentation of the example GUI, shown in Figure 1, is provided in section **Example Description**.

Summary

For Assignment 3, you are creating an interactive GUI which will allow a user to control the Builder and explore the world, removing and placing blocks on tiles as they travel. You have creative freedom as to what the GUI looks like, but it must meet the minimum functional requirements described above. Users should be able to load new maps without restarting the program.

Ethical obligations

All work on this assignment is to be your own individual work. As detailed in Lecture 1, code supplied by course staff is acceptable but there are no other exceptions.

You are expected to be familiar with "What not to do" from Lecture 1 and `http://www.itee.uq.edu.au/itee-student-misconduct-including-plagiarism`.

If material is found to be "lacking academic merit", that material may be removed from your submission prior to marking. No attempt will be made to repair any code breakage caused by doing this.

If you have questions about what is acceptable, please ask.

Supplied material

- This task sheet
- A `.zip` file containing implementations of the Assignment 1 and Assignment 2 classes.

Tasks

1. Create a GUI with all the required functionality
    - Create a "MainApplication" class in **game** package which extends from **javafx.application.Application**

    MainApplication class is the entry point of your JavaFX application.
2. Create a document which explains how the created applications works. See **Example Description**.

Marking

The 100 marks available for the assignment will be divided as follows:

Symbol	*Marks*	*Marked*	*Description*
F	55	By "humans"	Implementation and functionality: Does the submission conform to the requirements? How well does the GUI represent information and allow interactions for a user familiar with the documentation
S	25	By "humans"	Style and clarity.
D	20	By "humans"	Documentation: Does the documentation provided adequately describe: • The behaviour of all aspects of the GUI? and • How the GUI meets the requirements?

The overall assignment mark will be $A_3 = F + D + S$ with the following adjustments:

1. If $F < 5$, then $S = 0$ and "style" will not be marked.
2. If $D > F$, then $D = F$.
3. If $S > F$, then $S = F$.

For example: $F = 22, D = 17, S = 25 \Rightarrow A_3 = 22 + 17 + 22$.

The reasoning here is not to give marks to cleanly laid out classes which do not follow the specification.

Functionality marking

The number of functionality marks earned will be awarded in three broad categories:

- Does the representation of the world / world map render correctly and appropriately? [20 marks]
- Appropriateness of the visual aspects of the rest of the GUI (Layout, titles, interface elements, etc). [15 marks]
- Do user interactions function as described in your documentation? Are these interactions usable? [20 marks]

Documentation marking

The number of documentation marks earned will be awarded in two broad categories:

- Do the annotated images of each screen adequately describe the functionality? [10 marks]
- Does the written description address each required aspect of functionality? [10 marks]

Note that documentation which is illegible or unintelligible will earn 0 marks.

Style marking

As a style guide, we are adopting[1] the Google Java Style Guide
`https://google.github.io/styleguide/javaguide.html` with some modifications:

4.2 Indenting is to be **+4** chars not +2.

4.4 Column limit for us will be 80 columns.

4.5.2 First continuation is to be +8 chars.

- All private/"package private" members must be commented (you may use Javadoc comments but are not required to).
- Java source files must be encoded as either ASCII or UTF-8.
- All public and protected comments are expected to use Javadoc markup (e.g. @param, etc).

There is quite a lot in the guide and not all of it applies to this course (eg no copyright notices).

The marks are broadly divided as follows:

Naming	4
Commenting	8
Readability and layout	5
Implementation practices (including OO)	8

Note that this category does involve some aesthetic judgement (and the marker's aesthetic judgement is final).

Commenting

All methods and member variables need to be commented for this assignment. Comments for methods need to explain how the method is used, what the method returns, and what changes the method makes to its calling instance.

All public and protected member methods and variables must have Javadoc comments.

Additionally, any section of code (inside a method) that would be difficult for another reader to understand needs comments that explain what the code does.

Code Compilation

It is critical that your code compiles. If one of your classes does not compile, you will recieve zero. No corrections of typos or incorrect imports will be made by staff. It is your responsibility to ensure that your submission is error-free and meets with the specification.

Submission

Submission is via the course blackboard area `Assessment/`.

Your submission is to consist of a single `.zip` file with the following internal structure:

[1] There is no guarantee that code from lectures complies.

`src/game`	`.java` files for classes described in the Javadoc
`src/images`	optional folder to contain images if required by your program
`doc/`	which contains .pdf file which documents your application

A complete submission would look like:

```
src/game/MainApplication.java
src/game/* (any other classes you have created)
src/images/* (if any images were used)
doc/*.pdf
```

Use relative paths for images, e.g. `new Image("images/name.png")`. Do not enter a full path. For example on windows `C://Users/more/folders/CSSE2002/src/images/name.png`.

Any submission which does not comply with this structure will receive a penalty of 10% of the maximum marks available. Your classes must declare themselves to be members of the `game` package. Do not submit any other files (eg no `.class` files). Remember that java filenames are case sensitive when your filesystem isn't.

Late submission

As stated in the ECP:

- No late submissions will be accepted.

Example Description

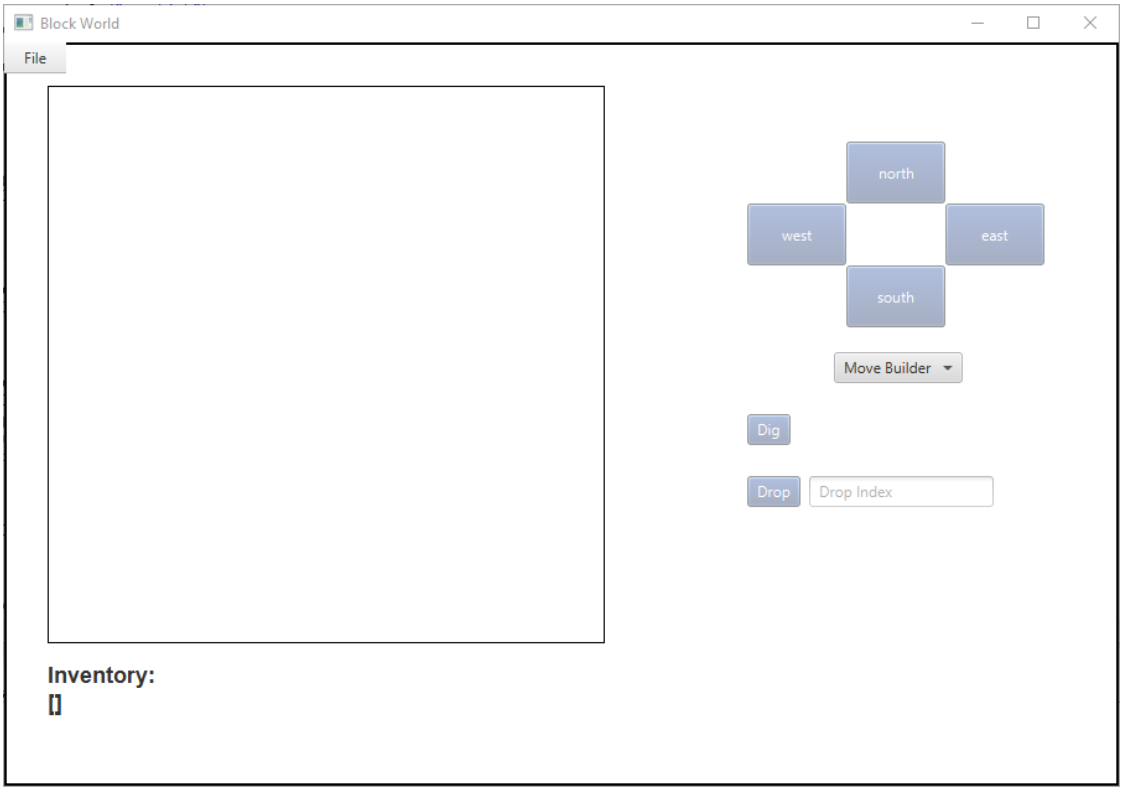


Figure 2: Block world program on start-up

When the game is launched all the action buttons are disabled as shown in figure 2. This ensures that users are not able to perform actions, as no map is loaded yet.

File Menu

To load a map user can use the “File” menu located at the top left. This menu includes two options, “load” and “save”, which can be seen in Figure 3.

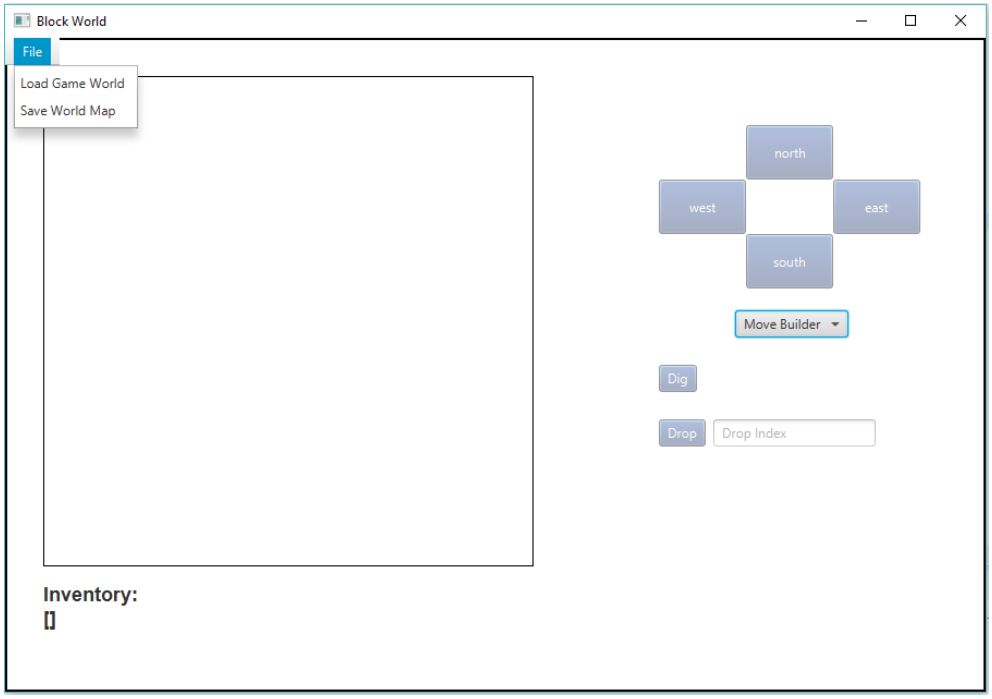


Figure 3: File menu options

This feature satisfies the **File Menu** requirement of the program.

Clicking the 'Load Game World" will open a file dialog which can be used to select a file to read the game world from.

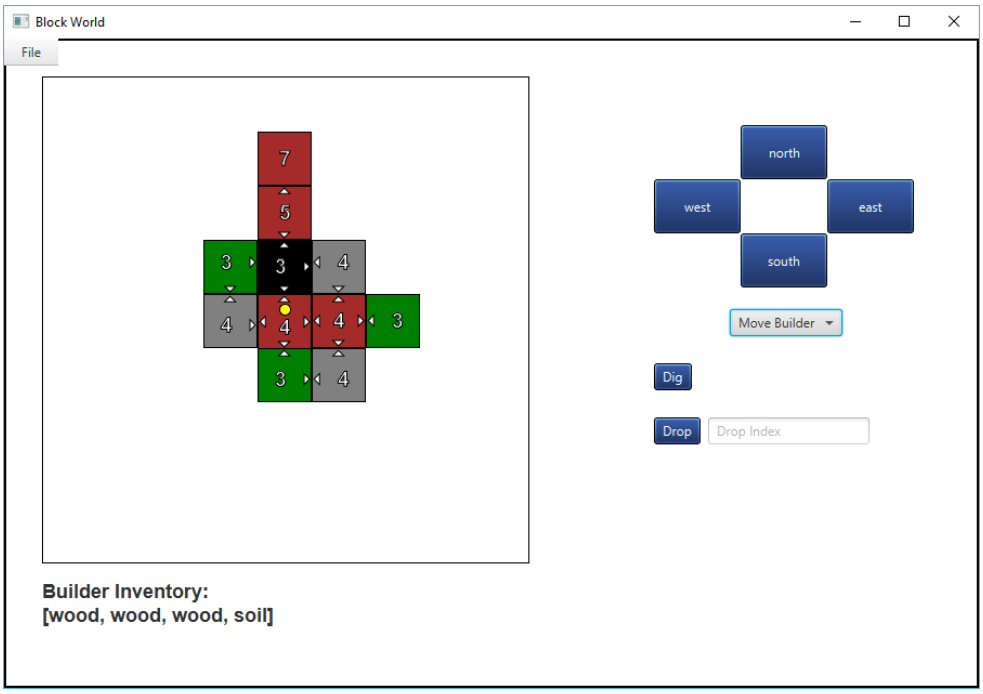


Figure 4: GUI when a map is loaded

Once a map is loaded, the GUI is updated so that the buttons are no longer disabled, and the game world is visualised in the left area.

Display Map

- The builder is shown in yellow
- Area will display 9x9 tiles. If a map contains more tiles, they will not be displayed until those tiles are in builder's view.
- Numbers on tiles are total blocks on the tile.
- Colour of the tile represents the colour of the top block on that tile.
- White triangles show the exits on the tile
- Text at the bottom is the builder's inventory.

Block colours:

- Brown → Wood Block
- Green → Grass Block
- Black → Soil Block
- Grey → Stone Block

Action buttons

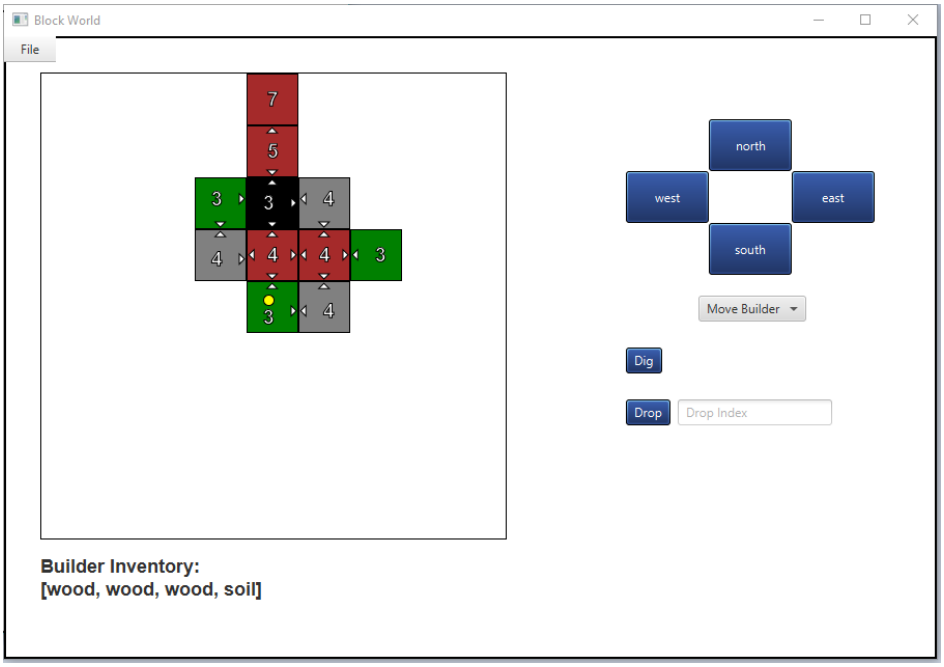


Figure 5: Builder moving south

The builder is always placed in the centre of the grid. Clicking "south" will move all the tiles up by 1. This can be seen in figure 5. Similarly moving north, east or west will move the tiles and not the builder.

Buttons are not disabled if the builder cannot move in a direction, AlertBoxes are used instead.

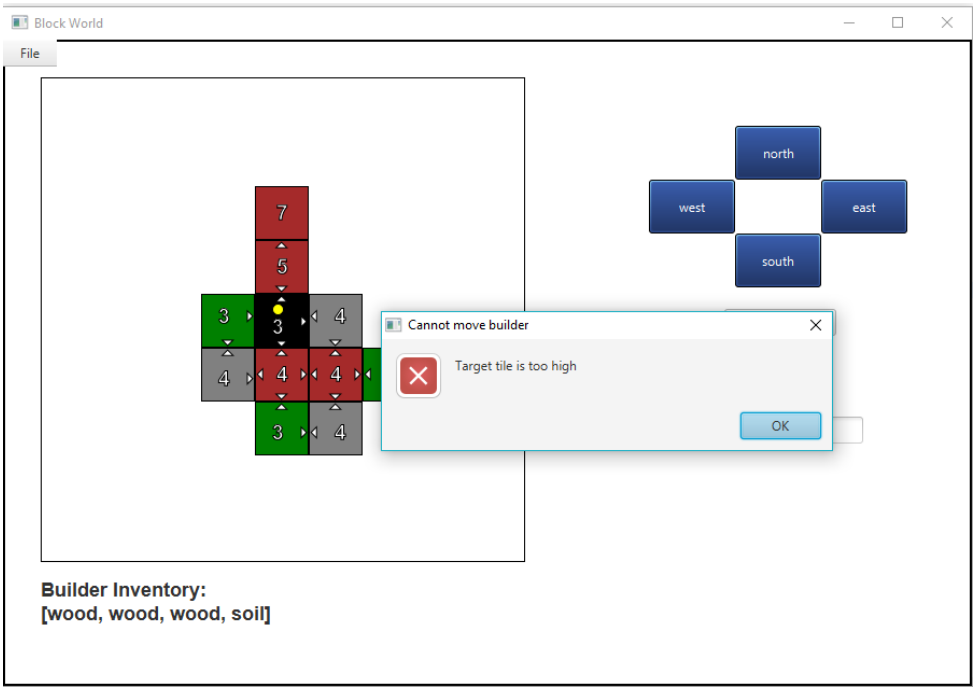


Figure 6: Target tile is too high when moving north

Figure 6 shows that builder cannot go north, as the target tile is too high. An AlertBox is used to display the message to the user. Similar message is also displayed when current Tile does not have an exit.

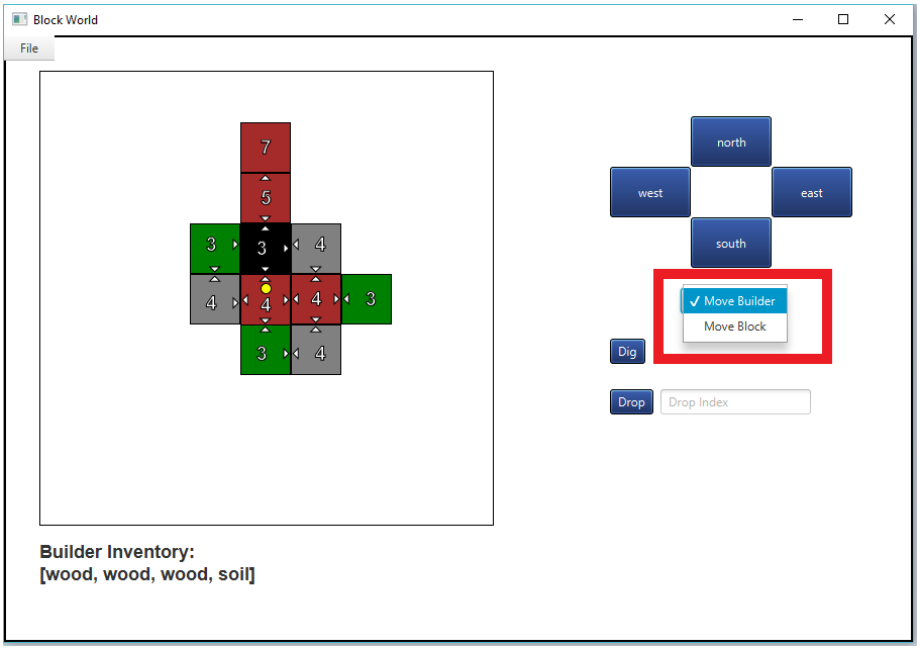


Figure 7: Options for movement

The choice list marked in Figure 7 can be used to switch between moving the builder or block.

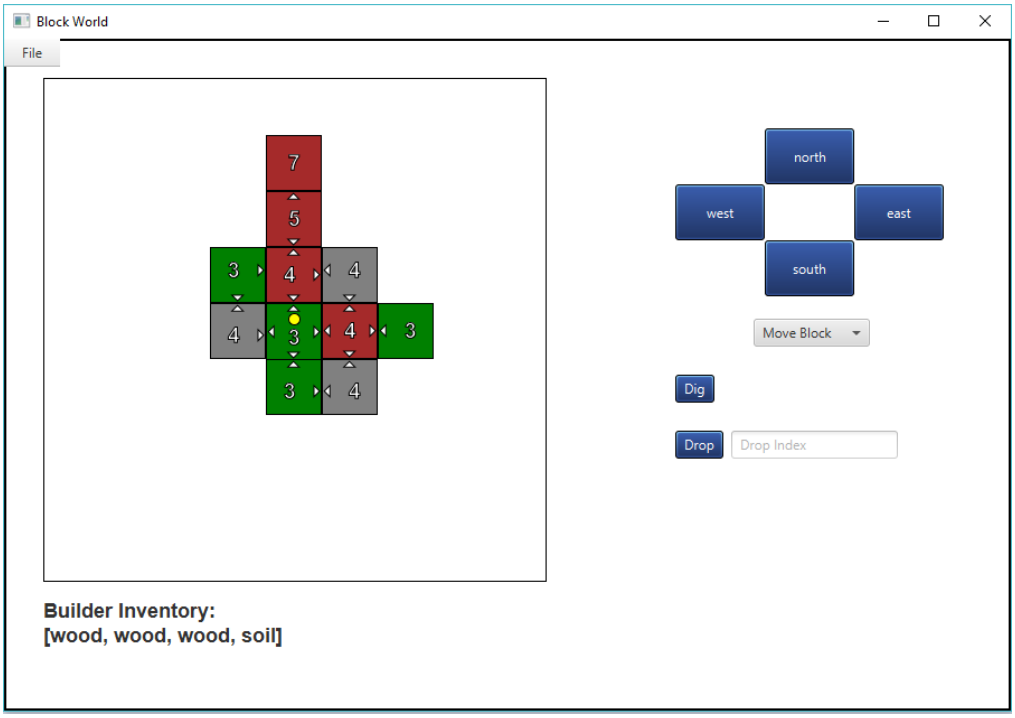


Figure 8: Moved block north

Refer to Figure 7, builder's current tile has 4 blocks and top block is a wooden block. When "Move Block" is selected and north direction is pressed, the top block is moved to the target tile. (See Figure 8)

If target tile is too high, there is no exit from current tile or block is not movable, an alert Boxes pop us with an appropriate message. See Figure 9

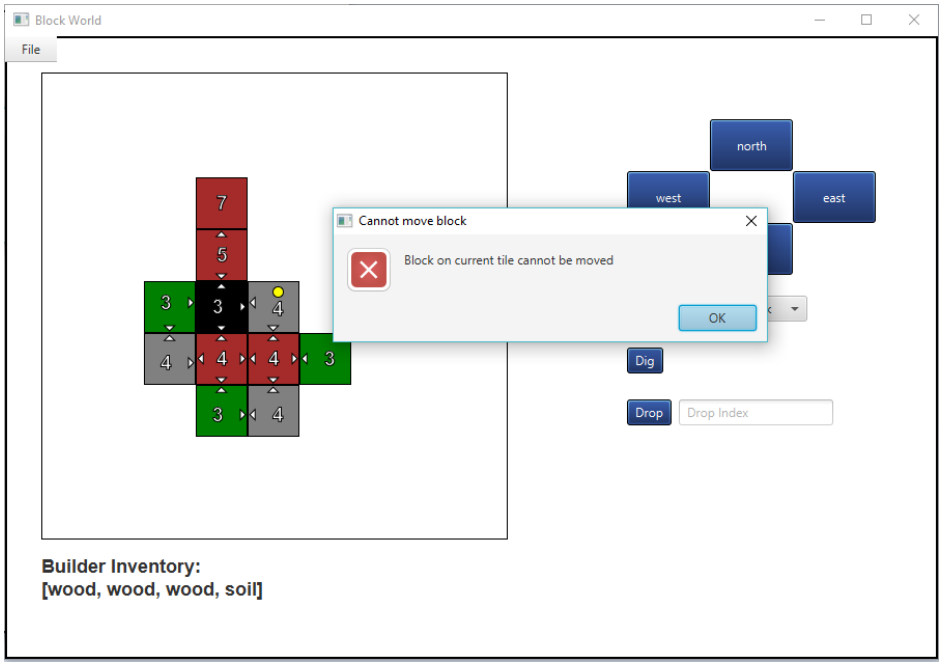


Figure 9: Stone block is not movable

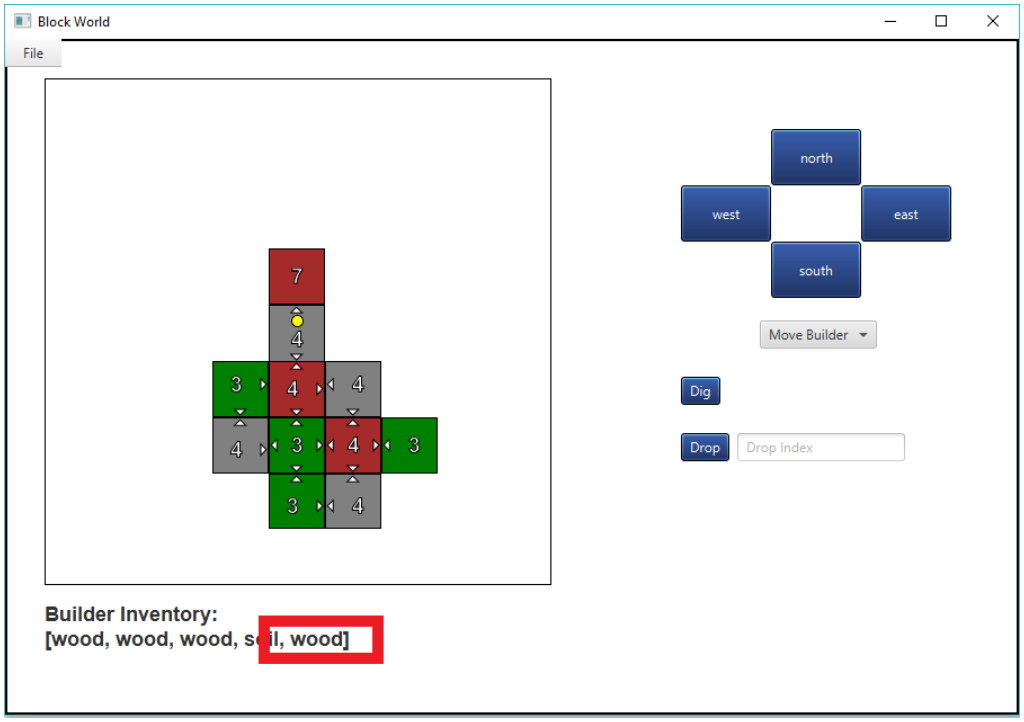


Figure 10: Dig on current tile

Pressing the Dig button will dig on current tile and if the block can be carried, it will add it to builder's inventory. Figure 10 shows that wood block is now added to builder's inventory and in the display area, the total number of blocks on the tile has changed, and top block colours has also changed. (Refer to Figure 8 to see previous state of the current tile).

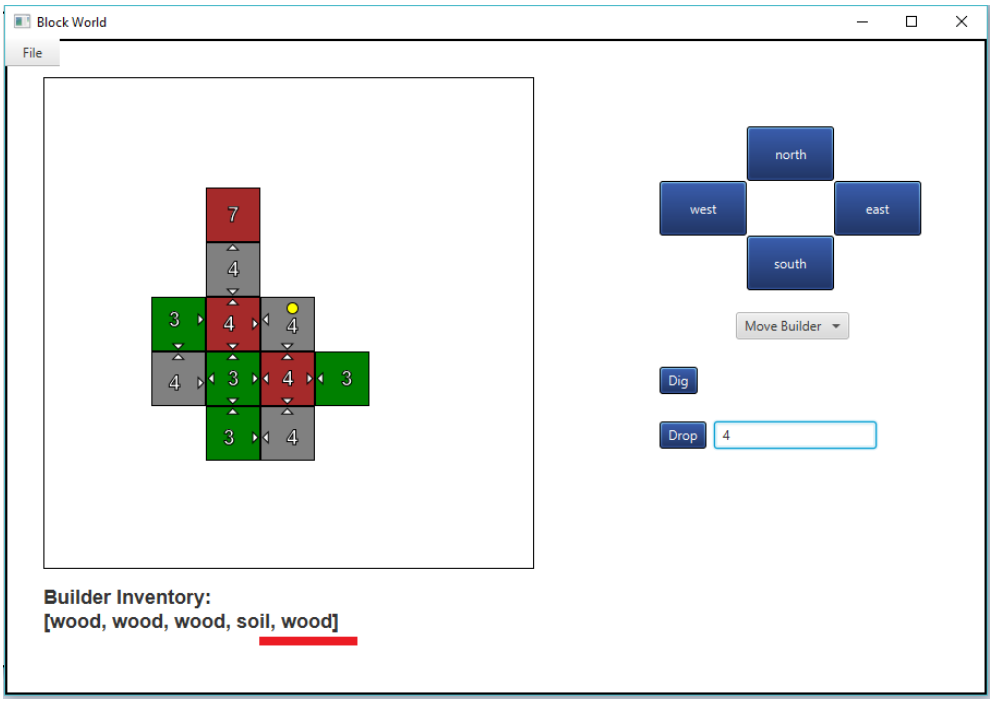


Figure 11: Enter text "4" for DROP action

Figure 11 shows how the index can be entered for drop action. When Drop button is pressed, if the entered string is invalid (not a number or index out of bounds), an error message will be displayed.

If the entered text is valid, the block will be dropped, and the display will be updated as shown in Figure 12.

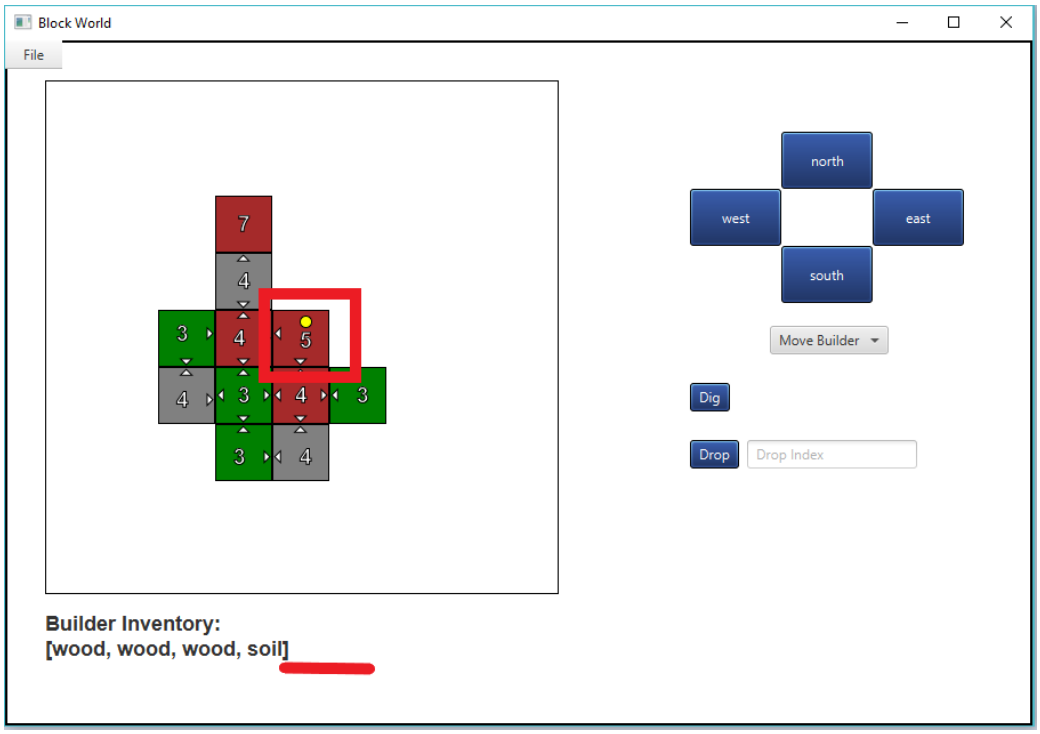


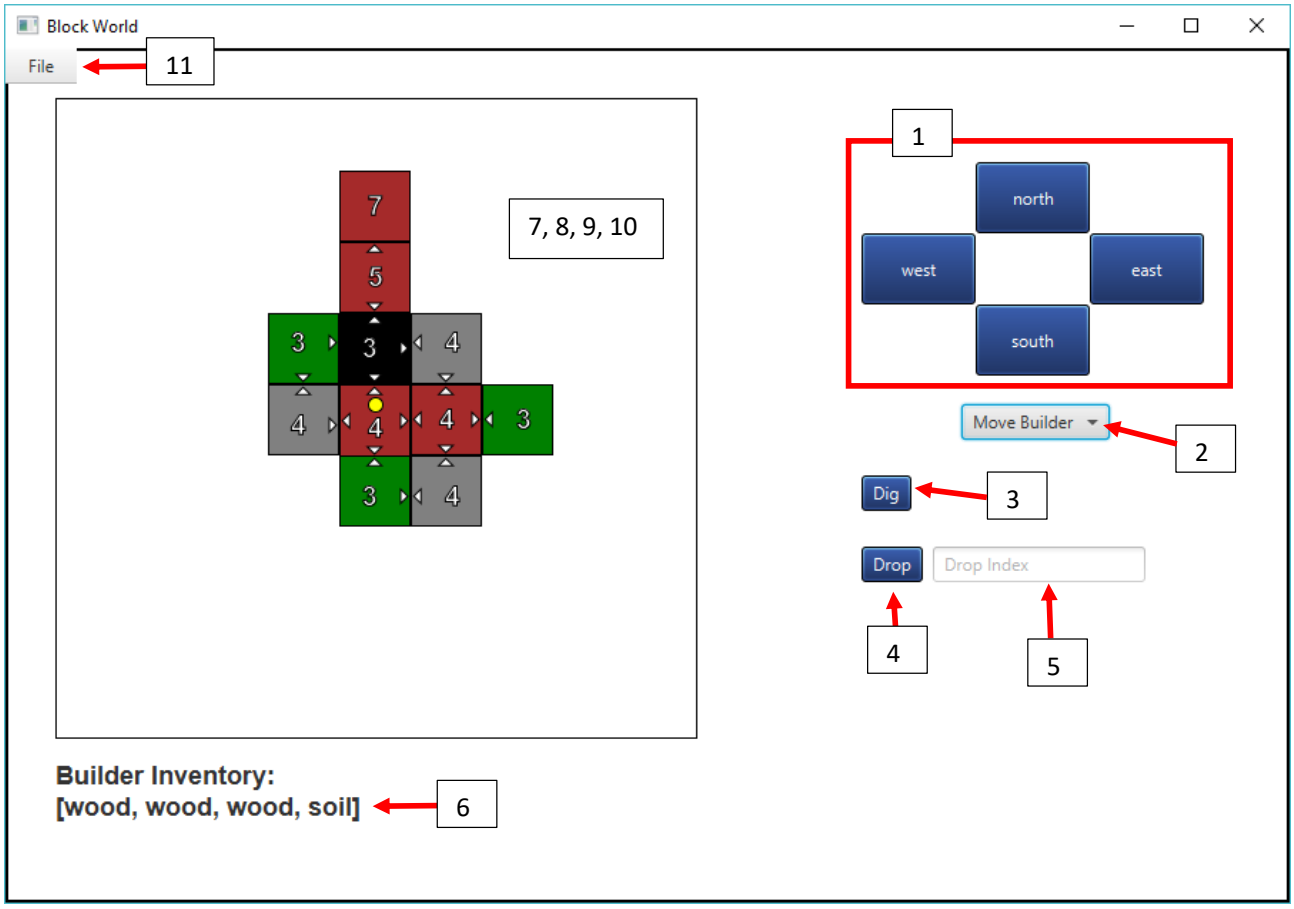
Figure 12: Block dropped at index 4 on current tile

Alert Boxes

Alert Boxes are used for the following scenarios:

- Moving builder
  - No exit in the intended direction
  - Target tile is too high
- Moving block
  - No exit in the intended direction
  - Target tile is too high
  - Top block is not movable
- Drop
  - Invalid text is entered in the “Drop index” field
- File Menu
  - When map is successfully loaded
  - When a map cannot be loaded

Annotated screen sample



1. Buttons to move the Builder around the map. Buttons to move the block on current tile in a direction.
2. Drop-down list to select whether to move Builder or Block.
3. Dig button to make Builder dig on current tile
4. Drop button to drop block on builder's inventory
5. TextField to specify which index to drop block at.
6. Display builder's current inventory
7. Builder (in yellow)
8. White triangles are exits on tiles
9. Numbers are total number of blocks on the tiles
10. Tile colours
    - green → grass
    - brown → wood
    - black → soil
    - grey → stone
11. File menu which has load and save options