

CpS 111 Program 4: World Builder GUI

Objectives

- Write an event-driven program with a GUI

Overview





In this assignment, you will create a GUI for our world builder project.

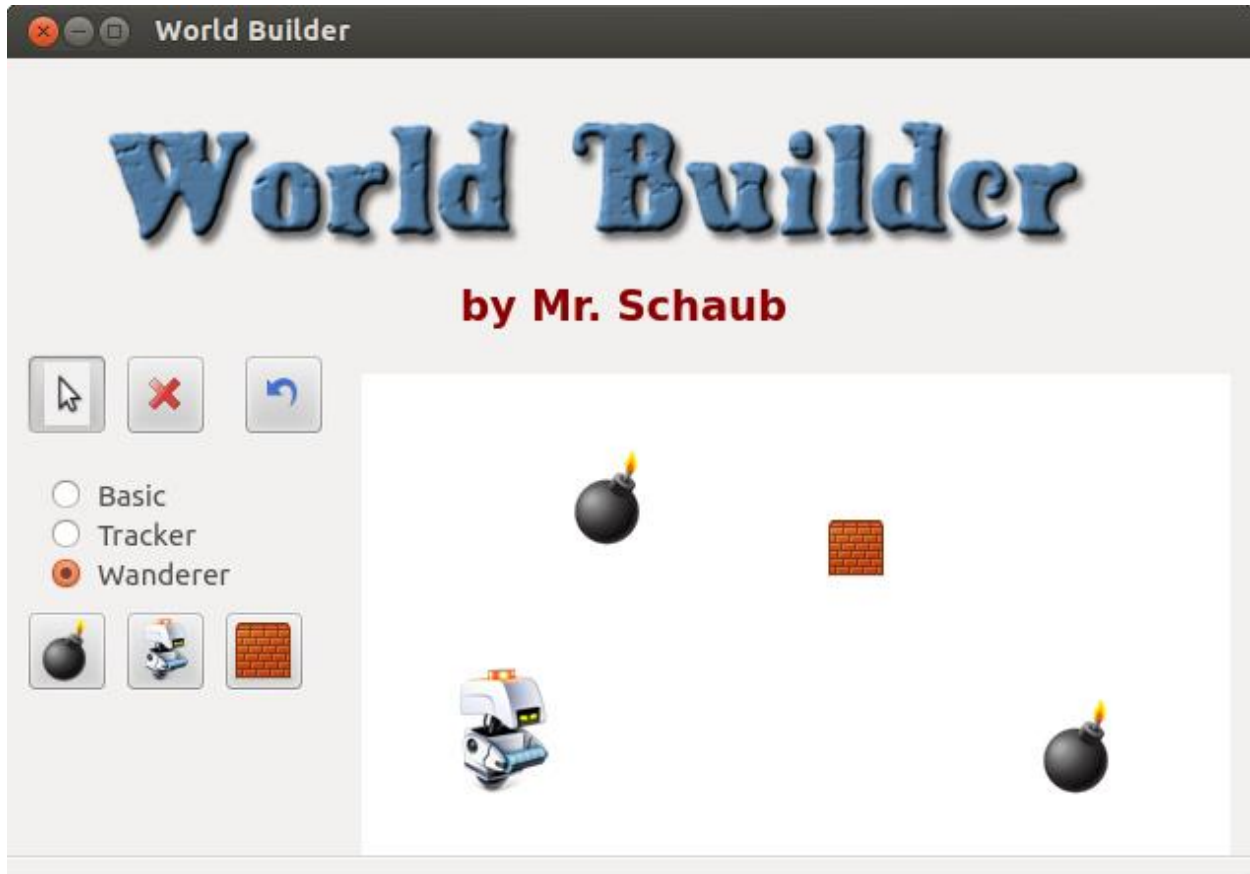
Setup

Following the procedure given in Lab 5, create a Qt GUI application. Name the project prog4, and create a window of type QWidget.

Maximum Grade 80

In this version of the application, you create a basic GUI interface for defining a world.

1. Create a UI that looks like the one below. See Design Requirements below for widget naming requirements. Icons are available in /home/cps111/prog4. Create a nice logo for your program (ex. using **cooltext.com**).
 - To create the white rectangle for the world surface, place a Widget on the window, and set its styleSheet property value to: **background-color: white**
 - Be sure to use the QT image resource system you used in lab 5 for all images, so that the image files do not have to be in the folder with the executable at runtime.
2. When the user clicks one of the entity buttons (robot, bomb, wall), create a new MovableLabel component (see /home/cps111/examples/qt/movablelabel), using the white "world" widget as its parent, and put it at the upper-left-hand corner of the world. Also, create a corresponding WorldObject (or Tracker, or Wanderer), based on the radio button selected, and store a pointer to it in MovableLabel.
3. Set the **checkable** property for the  and  buttons, so that they stay "checked" (pushed in) when the user clicks on them. Write code to manipulate their checked property, so that when one of the pair is pushed in, the other is unchecked. When  is selected, and the user clicks on an entity in the world, the entity should be deleted. When  is selected, the user should be able to drag an entity to a new position in the world, and update the corresponding WorldObject with the new coordinates.
4. In this version the undo functionality is not available. Disable the undo button.
5. After each GUI operation, output the list of entities in the World model using qDebug() or cout.



Design Requirements (All Versions)

- Implement a model/view architecture. The entities displayed visually in the World widget should always reflect the state of the World model.
- Implement model/view separation. As model objects, WorldObjects may not reference widgets or contain code that manipulates the UI, but widgets can reference WorldObjects.
- All GUI widgets that are referenced in C++ code must have appropriate names (ex. **btnDelete** for the Delete button).
- Classes should have only single-line methods defined in .h files. Methods longer than one line must be defined in .cpp files.
- Follow official style guidelines.

Maximum Grade 90

Add a button labeled **Animate** to the form. When the button is clicked, its label should change to **Stop Animation**, and animation should begin as follows: tracker-type objects should begin moving towards the closest wanderer-type entity. Wanderer-type objects should move randomly. The animation should continue until the user clicks **Stop Animation**.

Design Requirements (90 Version)

The logic for updating the x and y coordinates should be encapsulated in the Wanderer and Tracker classes, not in MovableLabel. Define a method named `updatePosition()` in the `WorldObject` class, and override it in the Wanderer and Tracker classes to compute the new locations. Use polymorphism when calling the methods (invoke `updatePosition()` on a variable of type `WorldObject*`).

You can get a list of all of the `MovableLabels` in the World by calling the World widget's `children()` method. Then you can iterate over them and have each label invoke its model object's `updatePosition()` method.

Maximum Grade 100

Add undo functionality by incorporating the code from your program 3 solution, or by requesting the official program 3 solution. If you are using the instructor solution, you should define an `UndoManager` instance variable in the `MainWindow` class. You will use this object to manipulate the world model, and perform undo work.

When the user clicks , undo the last action.

Tips

- It will help if you add a `MovableLabel*` to your `WorldCommand` objects, so that they know which GUI component to operate on when `undo()` is invoked.
- When the user performs an action in the GUI, such as creating, deleting, or moving an Entity, create an appropriate `WorldCommand` instance to do the work on the model, and have the `UndoManager` execute it.

Bonus (+5)

Implement multilevel undo using the undo mechanism implemented in Program 3.

Submission

- Reminder: If you have developed your code on any platform other than the official CSLAB Ubuntu VM, you should compile and test it on that platform before submitting. Submissions that do not work on that platform will not be accepted.
- Make sure your program 4 folder is named prog4. Zip up your prog4 folder and submit it to the Program 4 drop box in BJUOnline as `prog4.zip`.