```
class BulldozePopup : public QWidget
  Q_OBJECT

public:

  BulldozePopup(QWidget * parent=0, const char * name=0, Building* b);
  virtual ~BulldozePopup();

public slots:
  void clickedYes();
  void clickedNo();

signals:
  sendYes(LogicPoint* p);
```

BulldozePopup: Creates a popup window querying the user if they want to destroy the building they left clicked on while the bulldoze button was depressed.

clickedYes(): Is a slot connected to the button labeled “Yes”. Emits sendYes.

clickedNo(): Is a slot connected to the button labeled “No”. Closes window.

sendYes(): Is a signal sending to GUI the LogicPoint of the Building to be deleted. This signal is connected to a slot in GUI.

```
class ImageTextLibrary {

public:

  ImageTextLibrary();
  virtual ~ImageTextLibrary();
  void load();
  QPixmap& getImage(std::string imageName);
  QString& getDescription(std::string name);
```

ImageTextLibaray: This class contains all of the images to decorate GUI with. It's probably best to do it this way so there aren't hard coded filenames and paths in GUI. Plus it's easily extendible if we wish to add more building and need to add corresponding pictures. The text part corresponds to funny descriptions I (or another group member) will write for each building, just like the Sims!

getImage(std::string imageName): Gets the QPixmap mapped under the name passed in.

getDescription(std::string imageName): Gets the QString mapped under the name passed in.

```
class StatsPopup : public QWidget{
  Q_OBJECT

public:

   StatsPopup(GUI* gui, const char * name=0, Building* b);
  virtual ~StatsPopup();
  void listUpgrades();

public slots:
  void okPressed();
  void buyPressed();
```

StatsPopup: This class is a popup window that displays the stats of the building left clicked on when in “normal mode” (no non-view, non-speed buttons pressed). You can also buy available upgrades to the buildings from this window.

void okPressed(): Linked to the ok button. Tells the Building pointer to update the name if the user has changed the name text box. Closes the window.

void buyPressed(): Linked to the buy button. Building told to upgrade.

```
class VolumePopup : public QWidget{
  Q_OBJECT

public:

  VolumePopup(int currentSet, bool isMute);
  virtual ~VolumePopup();

signals:
  void sendVolume(int a);
  void sendMute(bool);

public slots:
  void changeVolume(int value);
  void mutePressed();
  void okPressed();
```

VolumePopup: This is the window from which you can mute all sound effects or change the volume with a volume slider.

void changeVolume(int value): This is called when the slider is moved. It emits sendVolume, which is connected to a slot in GUI.

void mutePressed(): This is called when the mute button is pressed. Emits sendMute, also connected to a slot in GUI.

void okPressed(): This is connected to the ok button. Closes window.

```
class GraphicsSubclass : public QGLWidget{
  Q_OBJECT

public:

  GraphicsSubclass(QWidget * parent=0, const char * name=0);
  virtual ~GraphicsSubclass();
  void mouseReleaseEvent(QMouseEvent* e);
  void mouseMoveEvent(QMouseEvent* e);
  void mousePressEvent(QMouseEvent* e);

signals:
  void mouseReleased(QMouseEvent* e);
  void mouseMoved(QMouseEvent* e);
  void mousePressed(QMouseEvent* e);
```

GraphicsSubclass: This class extends a QGLWidget, but the only functions I touch are QWidget ones. Graphics will add functions later that will be the "meat" of this class. Reimplemented to control mouseEvents.

mouseReleaseEvent(QMouseEvent* e): This emits a mouseRelease signal which is picked up by GUI. This is needed to place buildings, paths, and props as well as to destroy buildings and find their stats.

mousePressEvent(QMouseEvent* e): This emits a mousePressed signal which is picked up by GUI. This is needed to find the button that is being held down during a mouseMove event.

mouseMove(QMouseEvent* e): This emits a mousePressed signal which is picked up by GUI. This is needed to preview building, path, and prop placement.

```
class BudgetStats : public QWidget{

public:
  BudgetStats(GUI* gui, char, const char * name=0);
  virtual ~BudgetStats();

public slots:
  void setBudget();
  void hireProfessor();
  void fireProfessor();
```

BudgetStats: This class creates a window that displays the budget and from where you can change the budget. This class has a QWidgetStack that allows you to pick the displayed top widget from a stack of them. By clicking on a button, school stats are displayed. By clicking on another button, student body stats are displayed. By clicking on a final button, the faculty are displayed and you can hire and fire professors from that window. For Professor and Budget information, BudgetStats asks GUI which in turn asks Logic.

setBudget(): Takes the changed budget stats and sends it to the GUI which sends it to Logic. Does error checking to ensure entered texts are positive numbers. Pops up a warning if not and resets faulty entry box.

hireProfessor(): Takes the professor highlighted in the QListBox and tells GUI to hire professor. Called when hire button is pressed.

fireProfessor():Takes the professor highlighted in the QListBox and tells GUI to hire professor. Called when hire button is pressed.

```
class GUI: public QWidget
{
public:
  GUI(QWidget *parent = 0, const char *name = 0);
  virtual ~GUI();

//updates the QLineEdit displays at every nextCycle call.
//gets information from Logic.
  void setUniversityName();
  void setGameDate();
  void setCurrentEnrollment();
  void setTotalMoney();

public slots:
  //right column button slots
  void addBuildingPressed();
  void addPathPressed();
  void addPropPressed();
  void bulldozePressed();
  void statsBudgetPressed();
  void volumePressed();
  void loadSavePressed();

  //time button slots changes the QTimer's timeout signal frequencies
  void pauseTimePressed();
  void slothTimePressed();
  void lamaTimePressed();
  void cheetahTimePressed();

  //view button slots. GUI calls Graphics's corresponding methods with the
//mouse location
  void zoom();
  void pan();
  void rotate();

  //graphics mouse event slots
  void graphicsMouseMoved(QMouseEvent* e);
  void graphicsMouseReleased(QMouseEvent* e);
  void graphicsMousePressed(QMouseEvent* e);

  //Called when QTimer's timeout signal is emitted.  The main game loop.
//Calls nextState in Logic.
  void nextCycle();
```

```
//The slot connected to the bulldoze popup. Tells Logic to get rid of
//building at point p.
  void bulldoze(LogicPointer* p);

  //Sends this to sound.
  void mute(bool a);
  void volume(int vol);

  // Receives which building was picked by user in
//BuildingSelectPopup
 void buildingSelected(int buildingType);

// Receives which building was picked by user in
//BuildingSelectPopup

void propSelected(int propType);

//for budget stats popup.  Passes changes to Logic.
void hireProfessor(Professor* p);
void fireProfessor(Professor* p);
void setBudget();

//for budget stats popup. Retrieves the info to display
Budget* getBudget();
StudentBody* getStudentBody();
vector<Professor*> getProfessorsToHire();
vector<Professor*> getCurrentProfessors();
GameState* getGameState();
```

GUI: Okay, okay, going to elide tons of functions here! This is the main class which creates the main window, handles most interactions between Logic and the popup windows, communicates with Graphics and holds the timer/loop. The most important functions are below, the others have short descriptions above them.

void addBuildingPressed(): Opens up a BuildingSelectPopup and when a type is selected from the popup, a preview of the building will appear when the mouse is moved.

void addPropPressed(): Opens the PropSelectPopup and when a type is selected from the popup, a preview of the building will appear when the mouse is moved.

void statsBudgetPressed(): Creates a BudgetStatsPopup.

void volumePressed(): Opens VolumePopup.

void loadSavePressed(): Opens file dialogue. Straight Qt class.

void graphicsMouseMoved(QMouseEvent* e): The mouse movements from the graphics window. If the building, path, or prop button is pressed then moving the mouse causes a preview graphic to be dragged along with the pointer. This method calls on Graphics to do that. Moving while clicking on the middle button causes the view to change, and here this class calls on Graphics to do that.

void graphicsMouseReleased(QMouseEvent* e): The mouse release events from the graphics window. This places the building, path, or prop button if the left mouse button is clicked –supposing that the appropriate addBuilding, addProp, addPath button is toggled (a call to logic). Opens a BulldozePopup if left clicked on a building. Right clicking releases all buttons and clears all chosen building and prop types.

void graphicsMousePressed(QMouseEvent* e): Saves which mouse button was last pressed for reference during a mouse move event.

```
class BuildingSelectPopup {

public:

  BuildingSelectPopup();
  virtual ~BuildingSelectPopup();

public slots:
  void buildingSelected(QMouseEvent* e, int indexNumber);

signals:
  void sendBuildingSelected(int indexNumber)
```

<Note: PropSelectPopup will have the exact same functionality, except the Qt window may look a bit different, and therefore I need two different classes.>

BuildingSelectPopup: The window that holds thumbnails, stats, and descriptions of the buildings it is possible for the user to buy. Users click on the thumbnails to bring up descriptions and to activate that building as their "selection". This choice is passed on to the GUI which will place the building the next time the user clicks on the graphics window.

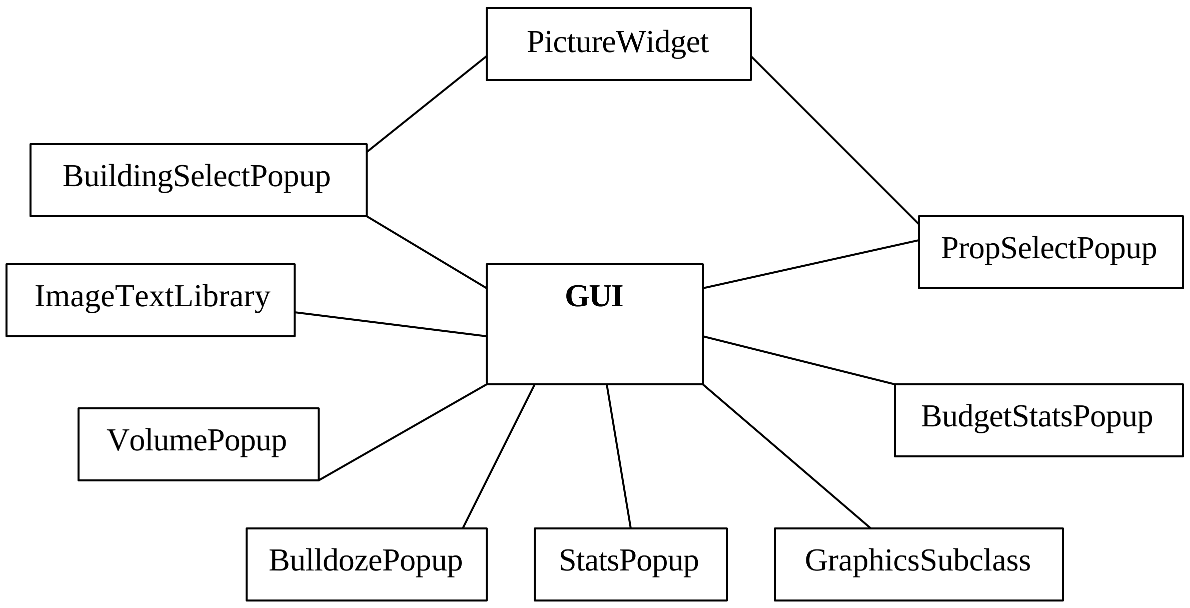
void buildingSelected(QMouseEvent* e, int indexNumber): Connected to each PictureWidget representing a building.

void sendBuildingSelected(int indexNumber): Connected to GUI. Tells GUI which of the buildings were selected.

```
class PictureWidget: public QWidget{

public:
//index number is the enumeration of a building type.
PictureWidget(QWidget* parent, const char *name = 0, int indexNumber);
~PictureWidget();

public slots:
  void mouseReleaseEvent(QMouseEvent* e);

signals:
  void mouseReleased(QMouseEvent* e, int indexNumber);
```

Containment Diagram

GUI instantiates all window popups in function calls, then relies on the windows to close (delete) themselves. ImageTextLibrary and GraphicsSubclass are member variables.


PictureWidget
BuildingSelectPopup
PropSelectPopup
ImageTextLibrary
GUI
BudgetStatsPopup
VolumePopup
BulldozePopup
StatsPopup
GraphicsSubclass