Richard Hsieh
Michelle Lin
Louisa Rosenheck
Bernard Peng
Daniel Shue
Ning Ge
Heather Yazawa
Ross Fabricant

Requirements

Overview

The CS190 version of SimsU is a stripped-down combination of the popular computer games The Sims by Maxis (see http://www.thesims.com/) and SimCity 4 by Maxis (see http://www.simcity.com/). However, instead of allowing the user to create and run a neighborhood with people performing the daily tasks of life, SimsU will allow the user to create and run a university with students and faculty who will go to classes, run errands, and socialize.

The object of The Sims and SimCity is to create a world and to ensure the well-being of its populace. In SimsU, the object will be to create a university that students will be interested in enrolling. When students enroll, they will pay tuition, which will give the player money to spend in upgrading the school. As the player builds better facilities and acquires better professors (which the player can also control), more people will want to attend. When students finally graduate, students which who are happy with their experiences at the university will give back to the university, giving the player even more money to spend.

To build a university, the player will be given the option to purchase building contracts to build dorms, libraries, greens, lecture halls, offices, dining halls, frat houses, athletic facilities, and campus bars. Because this is the Brown University version, these buildings will look similar to ones on the Brown University campus. The player can place these buildings (which will be pre-made so the user does not need to build one) anywhere he would like. Students will begin enrolling in the college once the user "opens" the university to the public.

On the more detailed level, the player will be able to control what type of students he wants to accept into the university, the general priorities of the students. He will have the ability to choose how much they study, how much they have fun, and how much they sleep. The player also chooses which professors to hire, which affects how the students learn. These factors combine to determine how the student body is doing.

Features (low priority in grey)

- **Game Flow**
  - Game starts with a preset map template with roads
  - User has $1,000,000 dollars to buy property, build buildings and hire professors
  - Buildings can be upgraded to provide better services
  - Simulation runs to determine the effects of the user's actions
- **Simulation**
  - Game speed can be changed (sloth, llama, cheetah)
  - Game starts in 1900
  - University Statistics
    - Enrollment
    - Safety
    - Food quality
    - Total capacity
    - Budget
    - Average Student Statistics
    - Graduation Rates
    - College Ranking
    - School Spirit
  - Professor Statistics
    - Subject taught
    - Experience
    - Likeability
    - Salary
  - Student Body Statistics
    - GPA
    - Social Life
    - Safety
    - Area of study / type (jock, nerd, arts, political activist, cs geek, etc)
    - General Well Being
  - Building Statistics
    - Subject
    - Capacity
    - Upkeep
    - Type
      - Living spaces
        - Dorms (upgradeable to singles / suites)
        - Theme Houses
        - Greek House
      - Classrooms
      - Cafeteria
      - Library
      - Park / Main Green
      - Student Center
      - Gym
      - Health Services
      - Police
      - Other (Concert hall, museum)
  - Calculation Updates
    - Formulas to determine how all these statistics affect each other
      - Enrollment increases when:
        - Student – Teacher ratio is low
        - Highly ranked professors
        - Highly ranked university

Formatted: Font color: Gray-25%

Formatted: Font color: Gray-25%

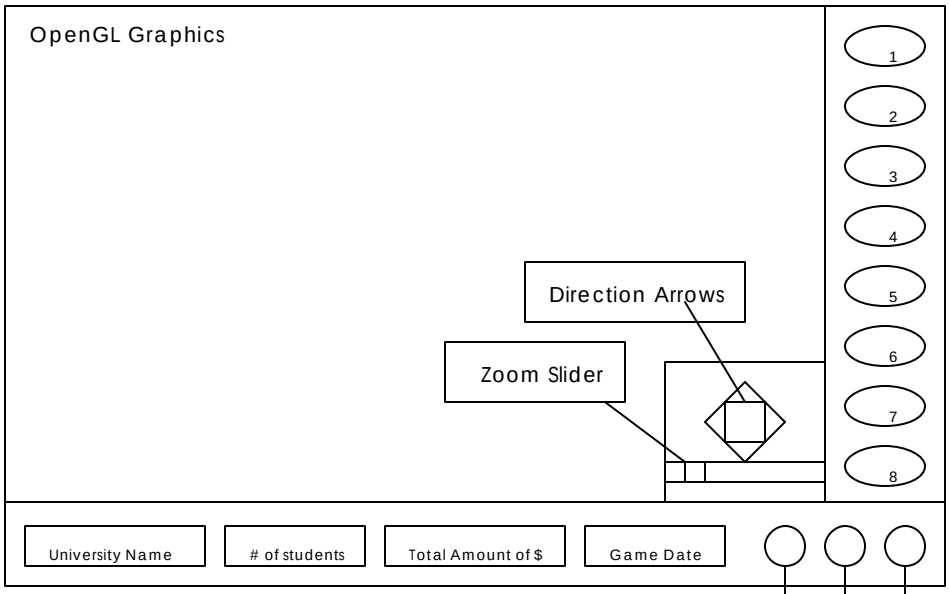
Deleted: <#>Maintenance ¶ Quality

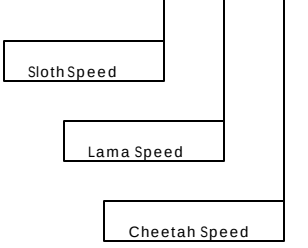
        - Happy students
        - School Spirit
        - Low Tuition
      - Enrollment decreases when:
        - Students unhappy
        - High Tuition
        - Student-Teacher ratio is high
      - Students enroll at the beginning of the year
      - Graduated students are added to an alumni pool
      - Students stats are affected by the types of buildings present (Library, Gym, etc)
  - Budget
    - Income
      - Tuition
      - Alumni Donations
    - Expenditures
      - Professor Salary
      - Building Maintenance / Staff
      - Police
      - Student Recruiting
      - Food / Health / Other
- **Player Management**
  - Name buildings / university
  - Buy land
  - Build buildings
  - Upgrade buildings
  - Hire / fire professors
  - Set budget
  - Set general student priorities
    - Study
    - Play
    - Sleep
- **Map (Logical)**
  - Overhead map similar to Simcity series
  - Buildings take up fixed amount of space on the map
  - User can build walkways and trees
  - Keeps track of locations / actions of student representative characters
    - "Representative Characters" show what each of student is doing (for example, a jock character would be shown going to the gym, going to play in sports games)
    - What these students do reflect the student priorities that the player sets
- **UI**
  - Menus to access configuration / load / save
  - Panel for choosing what kind of building/paths/trees the user will build
  - Clicking on the map will place the type of structure to build
  - Selecting a building will bring up information about it
    - Panel will be available for upgrading a building
  - Selecting a student will bring up information about them
  - Pop-up windows to warn player about what's going on
  - Menu for changing the school budget / general student behavior
  - Student newspaper to show what the people want
  - Menu for hiring professors
- **Sound**
  - Background music (mp3 playback)
  - Sound effects for game ev ents

Formatted: Font color: Gray-25%

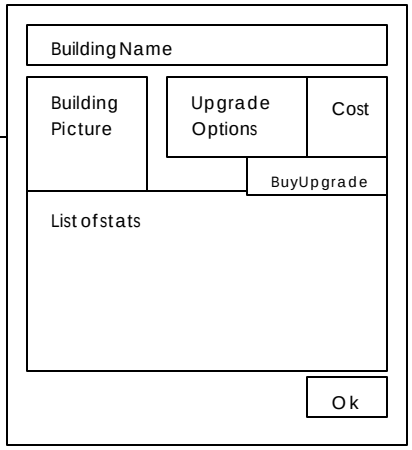
- **Graphics**
  - OpenGL display of buildings / streets / trees
  - Display of a few characters to represent what's going on in the school
    - Simple animation of characters (walk, etc)
  - User can move the camera around

GUI Layout



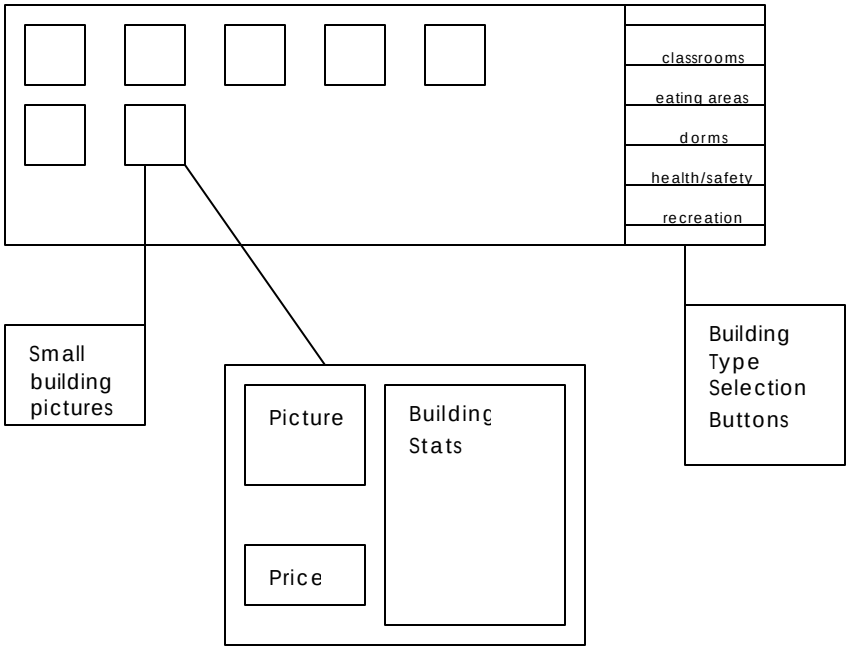


Left click to select buttons. Left click on buildings to bring up building information (occupancy, max occupancy, name, maintenance, department using it, description, upgrades done on it).



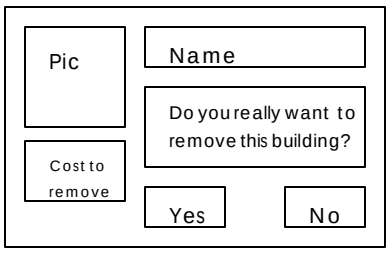
Buttons
1: Lay Buildings (pops up screen to choose building)
2. Lay Paths
3. Lay Trees
4. Buy Land
5. Destroy (Bulldoze)
6. Statistics/Budget (pops up screen)
6. Volume Control
7. Load/Save (Qt file i/o)

Building Popup



Building Popup shows thumbnails of buildings on the left. Left clicking on the picture brings up the stats for the building and price. Then left clicking on the map will place the object if possible (no obstructions). Right clicking on the map will cancel placement. Placing walkways and trees are similar, except that no diversity in trees and pathways are planned, so after selection either on the main menu, you can directly left click on the map to place.

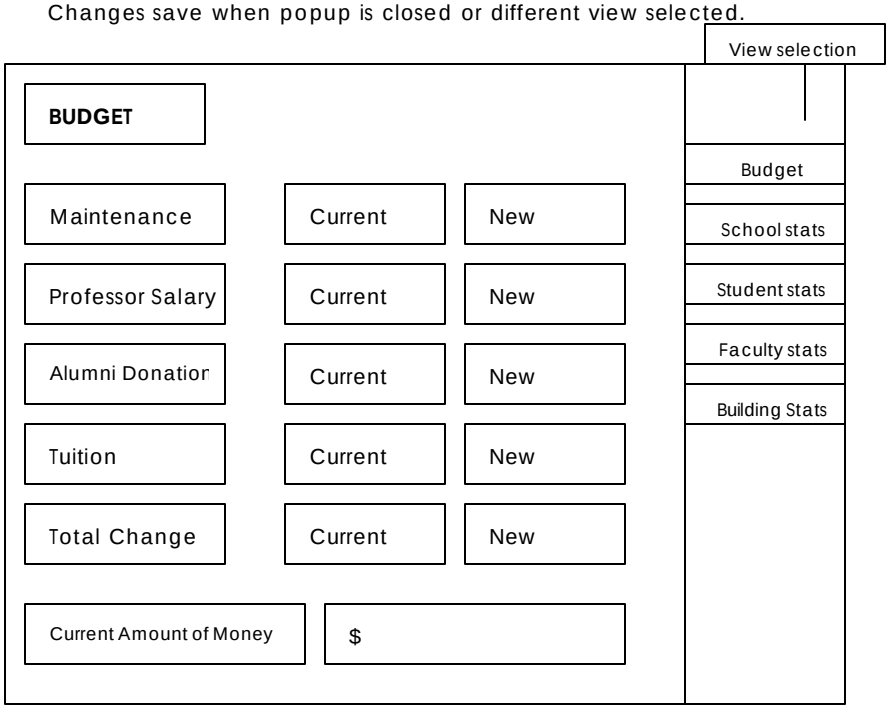
Bulldoze Popup



Left clicking on the bulldoze button and then left clicking on the target building will bring up this screen. Choose yes to destroy.

Statistic/Budget: Might implement view selections as tabs depending on space and time...but buttons would be prettier.

Budget View: Current settings and empty input boxes for changing budget. Changes save when popup is closed or different view selected.

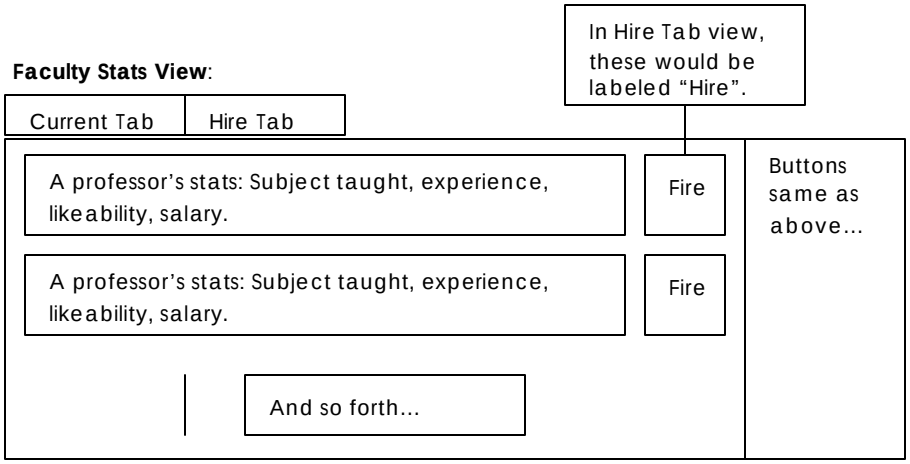


School Stats View: Includes ranking, number of incoming students, graduation rate, current class number, safety, food quality, total capacity, average student statistics. All stats are listed as numbers after titled box.

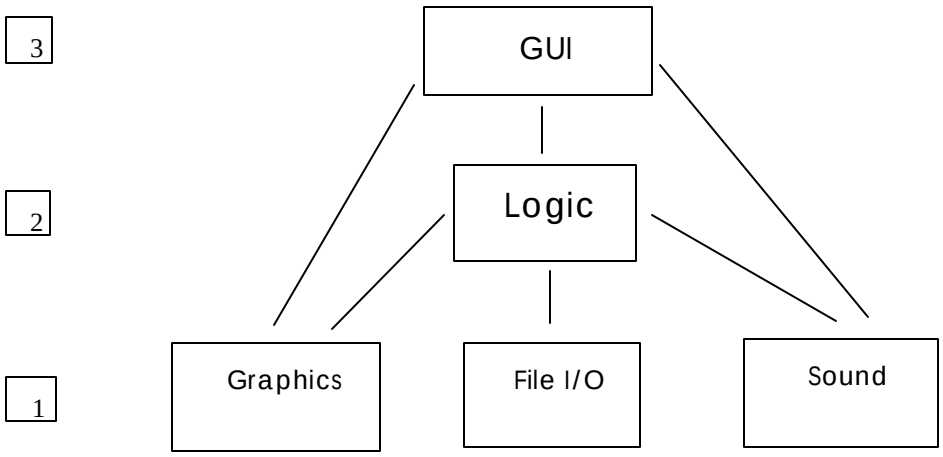
Building Stats View: List of buildings and their: capacities, maintenances, qualities, and types.

Student Stats View: Stats of General Happiness, GPA, Social Life, and Safety. Are all percentage numbers next to titles.

Faculty Stats View:



Component Diagram



Component Description and Interface

- **GUI**
  - Description (see GUI Layout)
    - Has a timer to keep the simulation moving
    - Displays panel for choosing what building to build
    - Displays information panel for selected object
      - Has buttons for upgrading that object
    - Has a menu for budget review / changes
    - Displays pop-up windows for warnings / events
    - Menu for changing options (music on/off, game speed)
    - Display for overall status (money, population)
- **Logic**
  - Description
    - Keeps track of student / building / professor / university statistics
    - Performs calculations to update the simulation
    - Keeps track of the school budget
    - Keeps track of the school map
      - Able to validate building placements
      - Update people positions
    - Determines if warnings should be issued to the player
  - Interface (for GUI to call)
    - saveState(string filename) – save the current GameState; called by the GUI.
    - nextState() - causes the LogicController to advance one time step, and update the GameState; called by the GUI.
    - addBuilding(BuildingType type, LogicPoint p) - used when the user is about to click to put down a building; it lets the graphics draw a square, so the user can see where the building would go.
    - getBuildingSize(BuildingType type) – returns the grid size of a building.
    - bulldoze (LogicPoint p) - destroys whatever is currently on the square.
    - fireProfessor(Professor *p)

Formatted: Bullets and Numbering

Deleted: <#>Has a timer to keep the simulation moving¶

- hireProfessor(Professor *p)
- std::vector<Professor*> getProfessors()
- Budget* getBudget() – pointer to Budget that GUI can modify and Logic can read.

- **File I/O**
  - Description
    - Saves simulation data to a file
    - Loads simulation data from a file
  - Interface (for Logic to call)
    - Load(filename)
    - Save(filename)
- **Sound**
  - Description
    - Plays background music (one song at a time)
    - Plays sound effects
    - Music and sound effects are enum'd (ie. MusicType, SoundType).
  - Interface (for Logic to call)
    - bool playMusic(int music_type) – plays background music (in a loop).
    - bool stopMusic() – stops background music.
  - Interface (for GUI to call)
    - bool openAudioDevice() - called before game starts.
    - bool closeAudioDevice() - called after game ends
    - bool mute() - mutes audio output.
    - bool resume() - resumes audio output.
    - bool setVolume()
    - int getVolume()
  - Interface (for Logic to call)
    - int playSound(int sound_type) - plays brief sound effects for game events
- **Graphics**
  - Description
    - Displays buildings / roads / trees / characters
  - Interface (for GUI to call)
    - int getWorldCoordinates(int pixelx, int pixely, float &x, float &y) – convert pixel coordinates to world coordinates.
    - int cameraPan(int x, int y)
    - int cameraZoom(int x, int y)
    - int cameraRotate(int x, int y
    - void init() – initialize the GL window (called by QGLWidget)
    - void draw() – draw the GL window (called by QGLWidget)
  - Interface (for Logic to call)
    - int addTerrain(int x, int y, int type);
    - int addProp(int x, int y, int type);
    - int addBuilding(Building *building);
    - int addRoad(Road *road);
    - int addCharacter(Character *character);
    - int deleteBuilding(Building *building);
    - int deleteRoad(Road *road);
    - int deleteCharacter(Character *character);
    - int prev iewBuilding(int x, int y, int type);

Deleted: <#>getGUICallback(displ ayFunction) – gets function to display pop-up warnings / display status in the GUI ¶
<#>setBudget(int) – updates the budget¶
<#>hireProfessor(professor) / fireProfessor(professor) – updates professor'sstatus ¶
<#>placeBuilding(x, y, type) / placeTree(x, y, type) / placePath(x, y, type) – places an object(if its possible) ¶
<#>upgradeBuilding(x, y, type)– upgrades a building¶
<#>setStudentPriorities(priorities) – updates student priorities¶

Deleted: <#>playMusic(file) ¶
<#>playSound(event) – plays the sound effect for a corresponding event. Can also play "click" sound when the GUI detects a button press. ¶

Deleted: <#>x, y getCoord(x, y)– converts pixel coordinates to world coordinates¶
<#>moveCamera(x, y)– moves the camera around¶

Deleted: <#>drawBuilding(x, y, type) – draws a certain type of building at a given location¶
<#>drawCharacter(x, y, type)– draws a certain character at a given location¶
<#>setCharacterAnimation(type) – sets what kind of animation to give a character (walk, stand, etc) ¶
<#>drawTree(x, y, type) / drawRoad(x, y, type)– draws various other objects¶

Schedule

Team Schedule

3/7 – Final Top Level Design
- Schedule is finalized
- Design is finalized and Architect's approval must be sought to make changes
- Requirements and specifications are frozen. Any changes must meet approval of Project Manager (PM).

3/12 – Interface Proposals
- Specify all methods that will be called by other components and describe their functionality.
- Create component-level testing strategies for each component and finalize testing deadlines.

3/14 – Final Interface Completed
- Interfaces frozen and changes must be approved by Architect.
- Update any changes to documentation.
- Directory structure / cvs setup

3/17 – Detailed Design Proposals
- Coders should finalize their class diagrams and implementation of their components.
- Any changes must be approved by Architect
- The layout and functionality of the UI should be finalized.
- Update any changes to documentation.
- Begin taking pictures for graphics

3/19 – 3/20 – Determine Design
- Coders should e-mail Architect to discuss design of individual components.

3/21 – Final Interface Handin / Detailed Design Completed

3/31 – Begin Coding
- Begin coding of individual components.

4/4 – Detailed Design Handin

4/7 – Testing Assignment Handin

4/11 – Individual Components compile with Minimum Functionality

4/13 – 4/15 – Testing of Individual Components
- Individual components should be continuously tested for bugs.

4/14 – Initial Integration
- Drivers completed. Components should integrate without full functionality.
- Coders continue testing their own components.

4/18 – Individual Components Completed
- All functionality is present.
- Individual components should be bug-free.
- Update any changes to design in documentation.

4/18 – 4/22 – Integration & Debugging
- Begin full system integration.
- *Coders & architect help us*

4/23 – Coding & Full System Integration Completed
- High priority requirements completed and frozen.
- Bugs should be minimal and small.
- First version of users manual should be completed.
- Update any changes in documentation.

4/23 – 4/27 – Testing and Debugging of Entire System
- Technical tester runs stress tests to break the code and figure out where bugs are. For example, game may seem to run fine, but may actually have an error that the user wouldn't notice.
- User tester plays game to check for proper functionality of GUI, user friendliness, lag, appearance, etc.

4/28 – Public Demo
- UI and Graphics should be bug free.
- Continue looking for Logic bugs.

4/28 – 5/1 – Testing & Debugging
- Find any last bugs.

5/2 – Public Demo in Lubrano
- Documentation should be mostly complete. Bugs should not effect the documentation.

5/2 – 5/8 – Testing & Debugging

5/9 – System Submission
- Coding and documentation completed.
- Final submission of project.

Component Schedule / Milestones

Logic

Schedule

3/21 - Do research on elements that could affect the operation of a school

Deleted: 31-4/4

- Finalize which information (budget, number of students, number of professor, etc) should be taken into account and come up with basic algorithms to calculate them.
- By the end of this week, we should have set up basic data structures for logical map.

3/31-4/11 - Work on basic features and map implementation

Deleted: 4/7

- Map should be done by the end of this week. We should be able to update the map, to add new buildings, delete buildings and check if a location is valid.
- Basic algorithm should be implemented. The SimsU simulation should be able to run and update school information according to time and data.
- Write a driver program to simulate input.

4/14 - Initial integration.

- The logic part should work smoothly with driver and produce correct information. Bugs may still exist at this time.

4/15-4/18 - Testing and debugging using driver program

- Logic part should be bug free using the driver program.

4/21-23 - Full integration with other components

- Together with tester, test the program and locate any bugs and fix them.
- First integrate with File I/O and sound. Make sure those work.
- Then integrate with graphics, make sure changes in the logical map is reflected in the visual map.
- Integrate with GUI. Make sure flow of control is correct.

Testing strategy

- Write a driver program which provides dummy input for the logical map. Add buildings, delete buildings, update map, and check for invalid updates.
- Write a driver program to test real time simulation of a school with dummy input for budget, buildings, student information, and professor information. The driver program should simulate potential user inputs from the GUI, such as hiring professors, adding new buildings, and reallocating budgets.
- Load an existing file and test the program using the data from the file and save it at the end.

File I/O

3/31 – File format specified

- The parameters describing the game's status at load/save time should be clearly defined and ready to parse from/to a (text) file.

4/4 – Loading function completed

- Start testing for load function; write driver that will load a dummy text file, and display parameters to see if they were loaded properly.

4/11 – Saving function completed

- Start testing for save function; write driver that will save dummy parameters into a text file, and check to see if they were saved properly.

4/14 – Integration with Logic

- At this point, Logic should be able to call File I/O's load and save functions.

4/18 – All functionality of File I/O completed.

Sound

3/20 – All sound files should be ready for implementation

- Research on SDL (http://www.libsdl.org) should also be completed by this date.

3/21 – Playback functions completed

- Start testing for Sound's playback
- Write driver that will call these functions; if the correct sound plays, then we're good.

4/14 – Integration with Logic and GUI

- At this point, Logic and GUI should be able to call Sound's playback functions to play appropriate files.

4/18 – All functionality of Sound completed.

Deleted: 4/4

Deleted: Skytech

Deleted: (http://www.skytech.org/~tuo/skysound/main.htm)

Deleted: 4/11

Graphics

3/20 – Rough Graphics engine with minimal graphics

- Should display a map with blocky graphics / colors representing different buildings
- Simple driver should be written to test the graphics display. This driver can be used throughout to test the graphics.

4/11 – Main buildings / characters modeled in 3D

- Buildings should be cubes with photo's texture mapped onto them
- Characters should be simple stick figures with walking animation

4/18 – All buildings completed

- More detailed models should be completed / texture mapped
- If time allows, particle effects may be implemented and fancy building / character animations created.

Deleted: 4/4

GUI

4/4 –Primitive GUI with plain Qt graphics

- Passing of MouseEvents to Graphics should be solidified .
- GUI component should be tested by itself. All internal functions should work.
- All pop up or changing screens should be created or transition properly, eg. The build mode button should bring up the build mode screen. GUI should have no "dead ends," that is, you must always be able to get back to the original selection screen.
- User assessment. Is the layout logical? Is it quick to learn? Does it interfere or distract from the graphical screen? Does processing a command take too many button clicks? Too many buttons? Is the GUI cluttered? And so forth... The final layout of the GUI will be finished shortly after this is through.

4/11 –The GUI has its buttons, but not fully polished.

- GUI should be correctly sending and receiving information to dummy components Graphics and Logic. Check some error handling. This is the largest part of testing, and is important to making integration smoother.
- Integration testing with Sound
- Final user assessment. Are the buttons obvious symbols? Are things easy to read? Are the colors easy on the eyes?

4/18 –Beautified GUI

- Integration testing
- Check error handling

Testing Plan

Coders should continuously test their individual components before integration using drivers and test code so that there are no bugs at final integration.

During the initial integration, drivers should be made to interface between components that must communicate with each other. This will make the full system integration go more smoothly.

During the integration process, each component will be tested with the components dependent on it, beginning with level one and working up.

Graphics, File I/O, and Sound are level one so they will be completed and tested individually first. Next they will be integrated with the Logic, which is level 2, and their interactions and interfaces will be tested thoroughly and all bugs found. Lastly, the level 3 component, the GUI, will be integrated on and testing will continue.

The user tester will mainly test the GUI, Graphics, and Sound, making sure that everything displays correctly. When bugs are found, they will be reported to the Architect who will locate them and have the corresponding coder fix them.

The technical tester will thoroughly test all the logic, by using test code, to make sure all the simulations are consistent and accurate, which are things that may not be noticed by the users.

External Dependencies

- Qt – used for the GUI
- OpenGL – used for 3D graphics
- SDL – used for sound playback (http://www.libsdl.org)

Deleted: SkySound

Deleted: http://www.skytech.org/~tuo/skysound

Group Organization

Manager/Administrator - Michelle Lin
Architect/Testing – Richard Hsieh
Documentation/User testing – Louisa Rosenheck
Graphics – Bernard Peng
UI - Heather Yazawa
Logic - Ross Fabricant and Ning Ge
Sound & File I/O – Dan Shue

Role Descriptions

Project Manager. The project manager is responsible for the overall progress of the project. His duties involve making sure that each team member is communicating with the rest of the group and accomplishing the milestones in the schedule. The project manager makes the final decision for all decisions.

Architect / Testing. The architect is responsible for knowing the design and working with team members to flesh out each person's role in the design. During the programming process, the tester will help the coders test each of their components. The tester is responsible for recording each and every bug he finds and must organize some method to report these bugs to the other team members. After integration, the tester will make sure the entire program works.

Documenter / User Testing. The main duty of the documenter is to write the user manual. The additional role that the documenter will have is to act as the librarian. The documenter can work with the tester to do this, since during the design stages, these roles do not have as much work as others. The user testing portion of this job involves recording what test users might think about the program, and reporting these opinions to the other team members.

Coder. The coder will implement and realize the design of the program. Each coder will be given a specific component to implement, given the detailed specifications worked on by the architect. The components are: Graphics, Game Logic, UI, and Sound & File I/O. Since graphics is such a large component, it would be helpful for the UI architect to work closely with the graphics coder. These two people together should find some way to get the art work for the game.