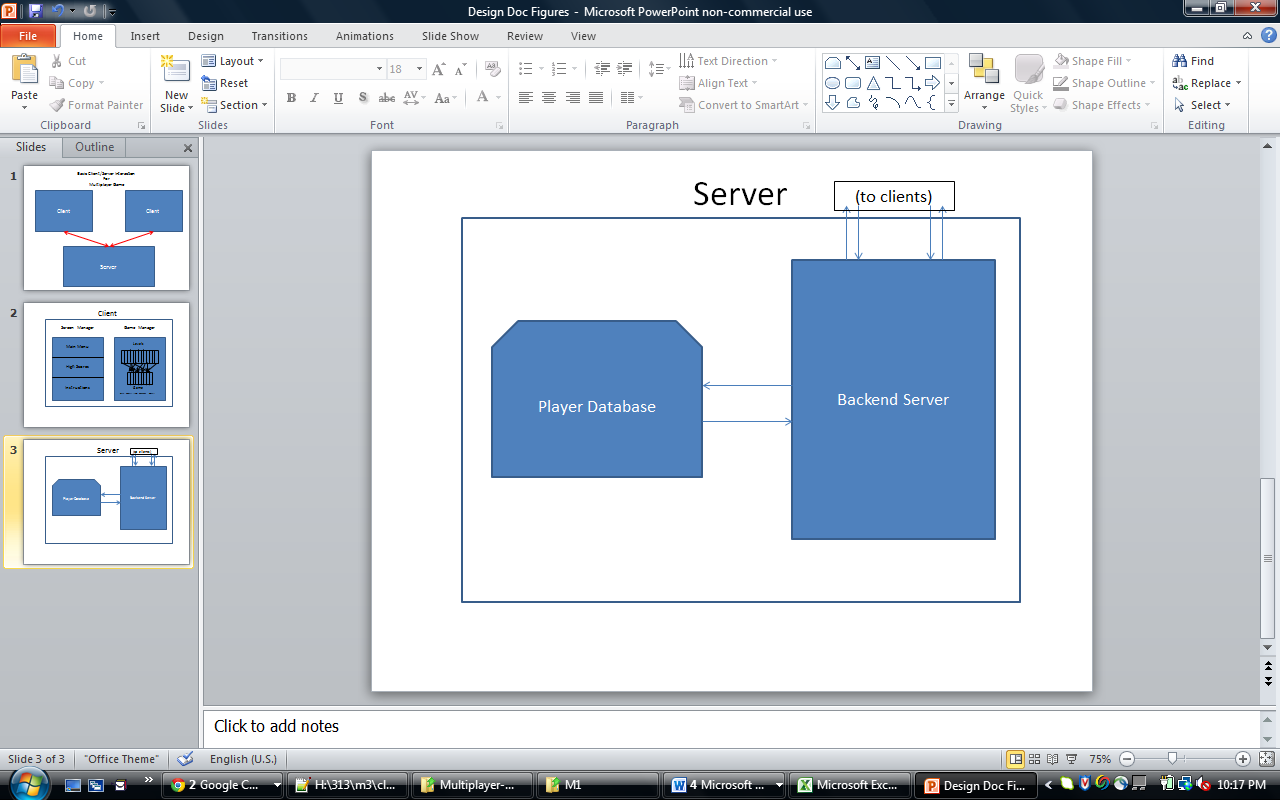
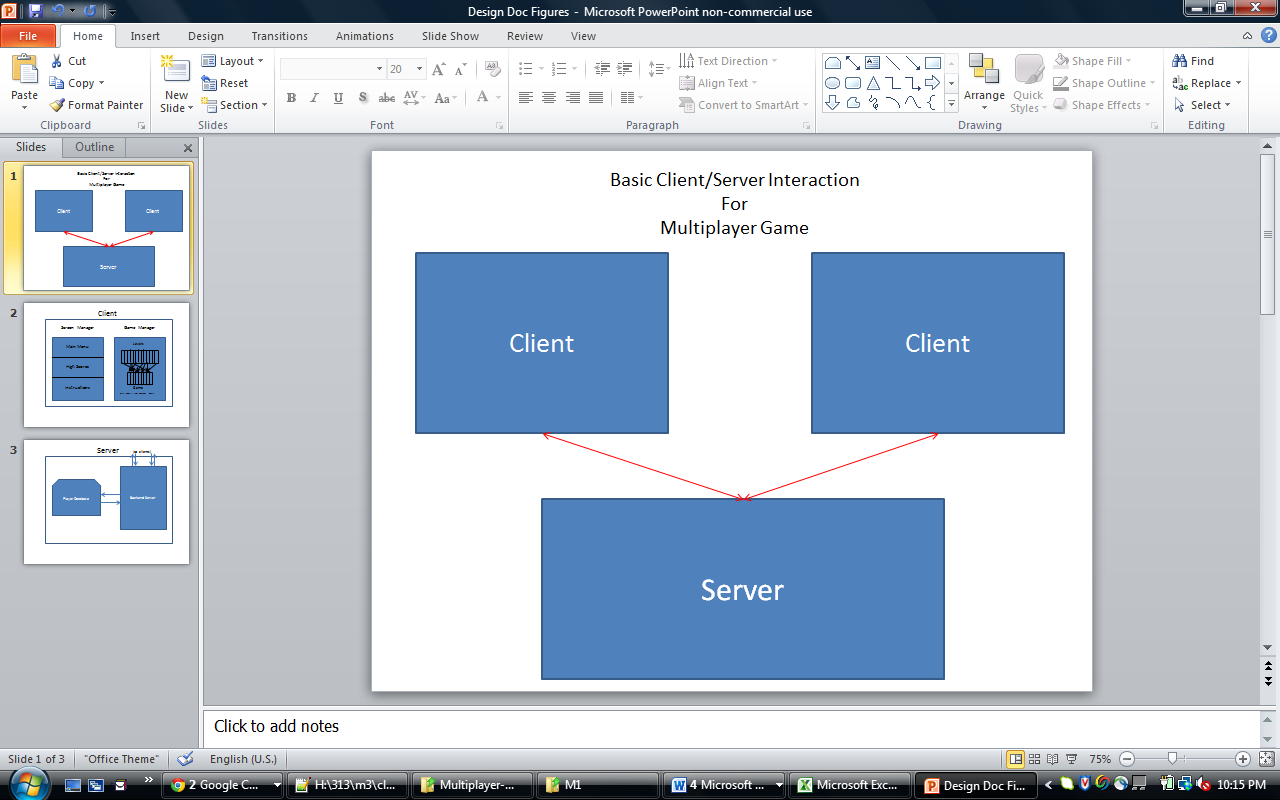
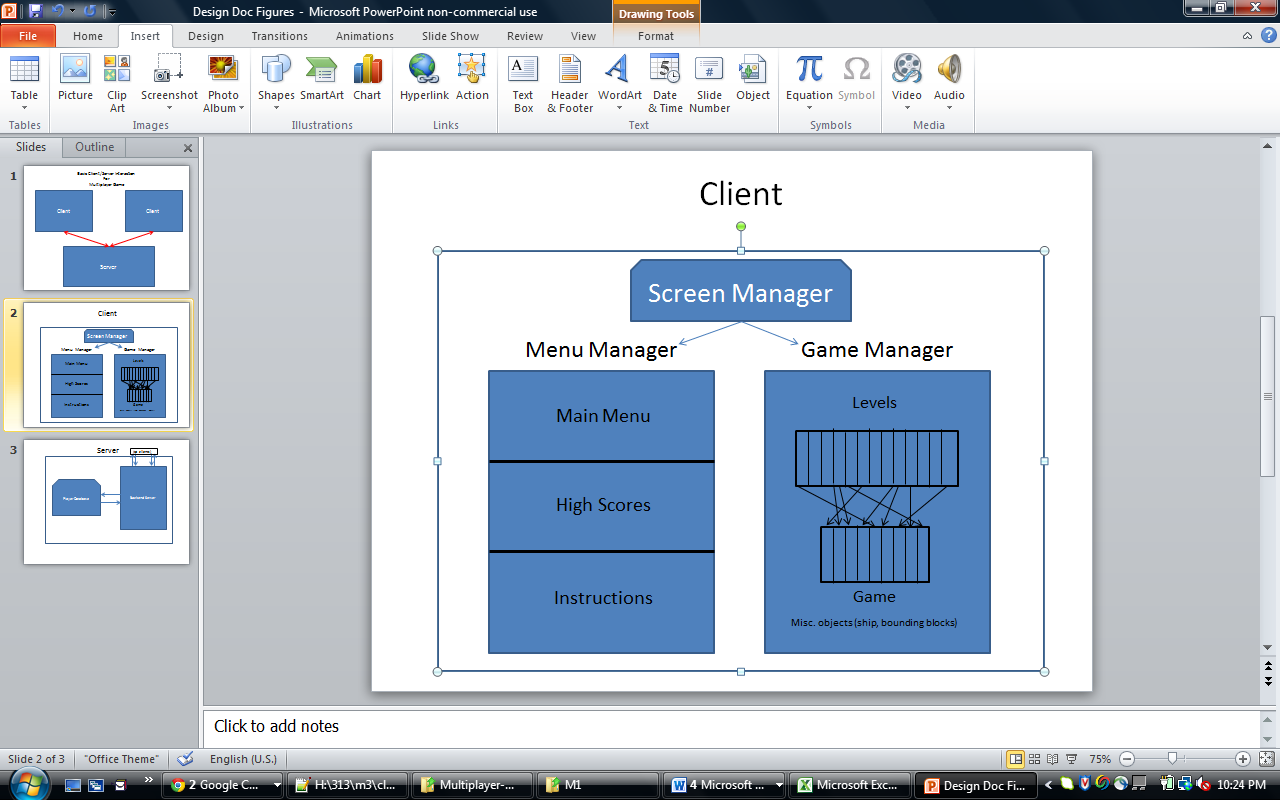
**Section 1 – State the purpose of your project/sub-system:**

This design describes on original game which attempts to provide all of the aspects of meaningful play. It is designed as an online, multiplayer racing game with a unique control system application. The style targets a retro aesthetic appeal, which is complimented by a simplistic control scheme.

**Section 2 – Define the high level entities in your design:**

High Level Entities:

In the game, we have:

The primary high level interaction in this design is between a client and a server, since the game is run in a multiplayer environment. Each player runs a local client on their own webpage, which communicates with a server, which in turn sends updates to another client, and vice versa. Furthermore, each of the high level entities can be further subdivide (especially the client side).

The client consists of a number of different aspects, which in turn create the game.

* Stage: hosts all of the local graphics. This is the easeljs underlying framework that we use to draw on the canvas.
* Screen Manager: passes control to the appropriate control manager with parameterized function calls. The two managers that it can pass of control to are the Menu Manager, and the Game Manager
  + Menu Manager: Contains all information about menus, and has access to server to retrieve high score information. Deals with any screen involving text only. Handles mouse inputs for these screens as well
  + Game Manager: Contains all information about the ongoing game. Included in the game manager is a list of all possible levels, as well as a list of the current game set up. The game manager contains a final tier of objects
    - Level: A numerical description of the various block-obstacle arrangements
    - Ship: The ship itself which the player controls, along with its various variables
    - CurrentGame: The randomly generated list of levels that make up a course, as well as the endpoints.
    - Collision Detector: An object to create simple collision detection. ***Notice that this is hosted by each client separately.***
    - Animator: Object to manipulate objects on the screen, mainly being the ship. Handles events such as a crash and respawn.

The objects on the screen itself are controlled by a local screen manager, which either falls into the classification of a menu or game manager. The menu manager handles text and clicks that transfer that change the menu. Again, to restate, all of the menus are run under the same manager.

Whenever the user clicks on the text to play the game itself, control is passed over to the game manager. A further high level entity contained in the gameplay is a level; a series of which make up a game screen.

Primarily, an interaction between at most two clients and a server

The Clients contain the following

stage, containing all of the objects that we draw

* Collection of levels that make up a race
  + Blocks for each level
* Ship
* Hazards

A collision detection mechanism (again, run on each client separately)

The server contains the following

JSON communication by receiving messages accessing each client

Database of usernames and multiplayer ratings

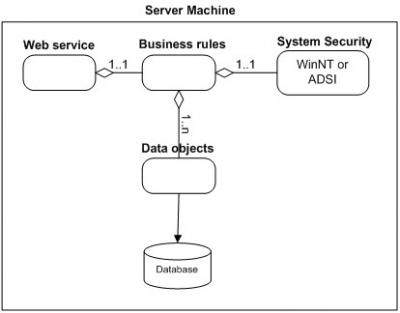
[](http://blog.slickedit.com/wp-content/uploads/2007/04/figure1.jpg)

Figure 1 (click to see full size)

In this section, explain in a few sentences what each entity does. The descriptions don’t have to be verbose, just enough to explain what each block’s purpose is. Be sure to describe your reasoning for defining the entities in your diagram and what their roles are.

**Section 3 – For each entity, define the low level design:**

Collection of level – describe scrolling

Internal design of what a level is

Simple description of a block

Ship – show how position and rotation are tracked and drawn

Hazards – how they move in relation to everything else

Collision detection – how it is called, and on what (only active blocks)

Database – organization and design

Communication

**Usage**

Describe in a paragraph how the object is used and what function it serves. If an object will interface with an external object or system, it is a good idea to show the interface for the object. Most importantly, you must again describe your thought process for defining the object as you did. List the benefits and risks. If an object provides an encapsulation, describe in a sentence why the encapsulation adds value. Use your descriptions to give meaning to the diagrams. They don’t have to be verbose, just enough to get the point across.

**Configuration**

If your object needs any special configuration or initialization, this is a good place to describe it. If not, this section can be left out.

**Model**

Figure 2 shows an example of a to supplement the System Security entity from figure 1. It is not perfect UML, but has some aspects of UML. Most importantly, it describes the design.

[](http://blog.slickedit.com/wp-content/uploads/2007/04/figure2.jpg)

Figure 2 (click to see full size)

Don’t worry about perfection in your models, but be sure to describe exactly what is going on in the diagram. Here, two concrete security objects derive from a base security object, and a security factory will create one or the other for a client depending on the security model of the system.

**Interaction**

This is also a good section for interaction diagrams. An interaction diagram shows how a set of objects or entities communicate with each other to perform a complex task. Figure 3 shows an example of an to show how a user might log in. It uses objects from the various entities shown in figure 1.

[](http://blog.slickedit.com/wp-content/uploads/2007/04/figure3.jpg)

Figure 3 (click to see full size)

Again, this diagram is not perfect UML, but it explains the communication sequence to accomplish a complex task. Interaction diagrams are most useful when you want to diagram how an object in your system will communicate with an object in another subsystem. This type of diagram will let the other developer verify that the interaction is correct.

**Section 4 – Benefits, assumptions, risks/issues:** In this section, make a list of 5-6 top benefits of the design, a list of **ALL** known risks/issues and a list of ALL assumptions. Some of this may simply be rehashing what you wrote in a previous section of the document. What’s important is getting all of these items into one section so that the reader doesn’t have to read the whole document to understand what the benefits, risks and assumptions are.

Never remove anything from this section! As risks become non-risks, document that they are now non-risks and why they became non-risks. Never erase them from the document. The same holds true for assumptions. You should be able to look at this section and know instantly what the current risks are to your design.