Disclaimer: This code has not been tested on the actual Expertiza system, and therefore certain implementation-specific bugs could (and probably do) exist. Don't hesitate to contact me (Eric Lang) at ewlang@ncsu.edu if something's broken; I'll do my best to fix it as soon as possible.

Summary:

This is a game made for the Expertiza system (Ed Gehringer, NCSU). It transforms an assignment-based course into a world where the player levels up and completes areas. The player doesn't receive grades; rather, he gains experience, and the professor would convert the experience values of the players at the end of the course into grades. The map is split up into distinct "rooms" unlocked by completing assignments; these are arrayed around a "center" in the middle of the map which contains portals to each room (portals which unlock as the player progresses).

Implementation summary:

The game was created using the javascript game engine ImpactJS. As such, all code is in javascript. When the game is running on the Expertiza system, Expertiza should send the game the info it needs to generate the level and game state at load-time. No information should need to be sent at run-time (i.e. it's not critical for the user to see a grade exactly when the professor grades an assignment; they can refresh if they really want to). The professor must be explicit in his description of the course parameters, assigning XP values, values to open portals, assignments, and rooms to all be stored on the database and sent to the clients for level generation. The players should have XP data stored, as well as an indication of progress (which assignments are completed, how much XP the player got on each assignment, etc). Any information sent to and from the server was designed to be made in JSON.

ImpactJS notes:
All entities extend 'ig.Entity' and require 'impact.entity'. They have an 'init' function which is ran when the entity is created and an 'update' function which is ran every update loop. They keep their own texture, generally stored in 'animSheet'. All 'settings' passed to init functions are stored in the class's variables on the "this.parent" call of the init function; before then, the settings are only available through the actual settings object. For any other information, refer to the ImpactJS documentation: http://impactjs.com/documentation

The following is a brief summary of the javascript files being used. All of these are located in the 'lib/game' folder of this directory.

main.js: This is the file that starts the game off. It generates the level, spawns all relevant entities, and sets all properties the server gives the client. As such, this is the file which should parse the JSON sent by the server. You can find a placeholder level definition JSON string in 'levelDefinition' at the beginning of the 'init' function of this file.

entities/player.js: This is, as named, the player. The player holds his XP, level (calculated from XP), whether or not he's flying through a portal, how long it takes him to portal, and his movement physics. He handles all user input.

entities/building.js: This is, as not named, an assignment. The player walks up to the front of this and presses space to be redirected to the relevant assignment page. An assignment stores the maximum possible XP the player can earn on it, the XP the player actually earned on it, whether the assignment is uncompleted, completed but ungraded, or graded, and the redirect link for the assignment. Each assignment spawns its own trigger which it then hooks up onto itself, and this trigger is what responds to the player walking up and the spacebar press and calls the assignment's "triggeredBy" function, redirecting the player. While assignments can not be repeated in this implementation, multiple assignments can lead to the same page, so an assignment meant to be repeated twice should be included as two separate assignments.

entities/trigger.js: This is, as you would imagine, a trigger. It was stolen from some helpful sample code ImpactJS gave us for free; in particular, it is the same trigger.js that exists in the Biolab Disaster pack. It calls the 'triggeredBy' function in any object that exists in its target[] array.

entities/gate.js: This is, slightly misleadingly, actually a portal. Portals open when their hubs are fully powered (see entities/hub.js below). If the player walks up to an open portal, they will be rapidly transported to the portal exit. All portals lead from and to the center at the moment (see entities/center.js below). Portals store whether they're open and call the player's 'portal' function when triggered by the player.

entities/hub.js: This is the object the player must power in order to open the portal to the next room. It is powered by completing assignments - every time the player completes an assignment, they power its associated hub with an amount of power exactly equivalent to the maximum XP value of the assignment. This happens regardless of whether the student has been graded (and regardless of what their grade is); progress to each additional part of a course is only blocked until the player attempts assignments, not until they do well on them. Hubs store their power, the amount of power they take to open the portals associated with them, and an array of the portals associated with them. Each hub controls both the portal from the center to the next room and the portal from the next room back to the center.

entities/center.js: While this object doesn't do much itself, it should be referenced by many portals so they know where to send the player. Generally speaking, the center represents the room at the middle of the level that all portals lead from and to. The center has no assignments on it. The players will find themselves frequently traversing the center to get to the next room or to return to a previous room.

entities/pointer.js: This object, misleadingly named pointer, is actually every tooltip you see in the game. Tooltips are drawn for any object that has a title, xp, or power. It checks the respective title, xp, earnedXP, openPower, and power fields on the object the user is mousing over. The object is implemented as a collider that tracks the user's mouse cursor; when it hits something, it draws the information about what it is in contact with.

Gameplay summary:

So, roughly speaking, this is how the game should be played.
1. The player loads up the game for the first time.
2. The player chooses an assignment he wants to complete by walking up to it and pressing the spacebar.
3. The player is redirected to the relevant Expertiza assignment page.

4. The player completes the assignment.
5. The player comes back to the game, receiving power toward opening the next portal, and maybe restarting from step (2).
6. The player goes to the center and to the next room.
This continues.

After a while, the assignments get graded and the player earns XP and levels, effectively progressing through the real course. While simply attempting assignments can allow the player to go through the game rapidly, the player still won't receive a passing grade in the course without doing fairly well on his assignments.

What I know about implementing the game on Expertiza:

While I obviously didn't develop the backend to this game, the interface to the backend is as follows:
The game should be integrated in a view of some sort. From what I recall, the lib and media folders need to be in the public area of the site, and the view must include the same general information that is in the 'index.html' here, but index.html could easily be stripped down to allow the game to be played in some sort of layout. The important part is to include the two script definitions ('lib/impact/impact.js' and 'lib/game/main.js') and the canvas (<canvas></canvas>).
To interface with the game, load the level definition in the format seen in the example in 'lib/game/main.js' into that variable (levelDefinition), including any extra properties and updated values. Remember, the server should only need to talk to javascript at load-time, not at run-time.

How to use the game locally:

Using the game locally is tricky, as the game needs to reference files on the "server" (i.e. your local machine) to get assets and such. Basically what this means is you have to run a server on your computer to even test this game locally. As such, it's recommended you either test it remotely or you go ahead and set this system up early. It isn't too painful. See: http://impactjs.com/documentation/getting-started

Contact me:

I know this documentation could be a bit spotty (and parts of it may be slightly out of date; I've been updating as I develop the system and may have missed something), so don't hesitate to contact me at ewlang@ncsu.edu. I'm sure we could also exchange Skype information, if you prefer voice chat.