Project

Due: Friday, April 24, 2015

The term project will be to use HTML, CSS, and JavaScript to implement a web application. This page describes the project in terms of grid-based puzzles and games. If you would like to make a different kind of web application for your project, you will need to get approval from the instructor.

A point breakdown for the project is at the end of this page.

As part of your first assignment you should select a grid-based puzzle (e.g. Sudoku) or game (e.g., Battleship) that you will implement for your project. For simplicity I will use the term "game" to refer to games, puzzles, or even math problems or science questions. For each subsequent assignment you will improve the implementation of the game and add features. At the end of the term you will turn in the completed game. Keep in mind that although each assignment adds to the game, finishing all of the assignments is **not** enough to get a good grade on the project. The assignments focus mostly on the user interface, and you will also need to implement game logic that is not part of any assignment.

The main purpose of the assignments and project is to learn how to use HTML, CSS, and JavaScript to create web applications. You will learn how to generate HTML from JavaScript, work with user input in the form of mouse clicks and text input, dynamically change the style (CSS) of HTML elements, and load information from the server without reloading the entire HTML document (Ajax). Our projects will make use of HTML tables because there are many games that make use of grids of one kind or another, but we will use the same techniques that are used with other kinds of HTML elements and layouts.

There are many games from which you can choose, including classics like chess and checkers, and relatively new games like Sudoku. Here is a list of possibilities:

- chess
- checkers
- crossword puzzles
- word search
- Battleship
- Minesweeper
- Sudoku
- Concentration (matching game)
- Othello/Reversi
- adventure/dungeon game map
- sliding tile puzzle
- Black Box

Two other possibilities for projects that are not games are an event calendar and an expense tracker. Like the games listed above, these applications make use of a grid and therefore work well with the assignment specifications, which are written in terms of grids.

Since there are many possibilities for games, I have written the assignment specifications in general terms. It is very important that you do every assignment, even if it doesn't seem to fit your game very well. For the final version of your game you can leave out elements that don't fit, as long as you satisfy the project requirements listed at the end of this page.

I am asking you to choose a project early in the term, before you get a chance to see how JavaScript web applications work. If you decide later that your project is too ambitious, or you find something more interesting, you can change your project. Just send me email to get my approval of your new project.

Assignments will focus primarily on the interface. Over the course of the term you will need to write JavaScript code to implement the logic for your game. The JavaScript code for the game logic should be core JavaScript and *not* make use of client-side JavaScript functions or refer directly to web-page elements.

For the final project you will integrate code from the assignments with the game logic code to make a complete, functional game. Not every assignment will be relevant for every game, but you should do the assignments in such a way that they fit into the overall game as well as possible. If you are not sure how an assignment relates to your game, talk to me.

Game functionality

In order to earn full credit for the final project, you will need to implement a playable game that includes reasonable functionality. Here are some descriptions of game functionality that would be sufficient for full credit:

- **Battleship:** Ships take up multiple grid squares, with each type of ship having a different number of ships (aircraft carrier 5, battleship 4, etc.). Ships are randomly placed by the program, or the player is allowed to place ships. Whether or not the ships are placed by the program, the program ensures that all ships are within the bounds of the grid. There are two grids, one for the player and one for the computer player (game AI). Hits and misses are marked on each grid. The program notifies the player when a ship on either grid has been sunk, and the program indicates when the game is over and which player won. The computer player (game AI) makes follow-up shots when it hits, so that whenever it hits one of the player's ships it fires at adjacent squares until it has sunk the ship.

- **Sudoku:** Given numbers are in a different style than user-entered numbers, and preferably given numbers cannot be changed. The player can get feedback on incorrect numbers at any time. There are multiple difficulty levels and multiple puzzles to choose from in each difficulty level. Puzzles are loaded using Ajax. Your program does not have to generate new puzzles. There is a timer that tells how long the player has been working on the puzzle. Please put the solution to one puzzle on the game description page or on the game grid so that I can easily test your game.

- **Concentration/Memory** The program displays the backs of the cards in the grid and lets the player turn over two cards at a time. If the cards match, the cards are left face-up and the computer records the match (increasing the player's score). If the cards don't match, the computer changes them back to face down after giving player a chance to see what they both are. The card fronts and backs are images. There is more than one deck of cards (card fronts) and the player can choose which deck to use. Each deck represents a theme or category, like animals, cars, food, etc. There are different difficulty levels based on how many cards are in the grid and the amount of time allowed. The program only allows a fixed amount of time, and the score is the number of matches found within that time period.

- **Minesweeper** The program generates a different mine configuration for each game. The player can flag squares in the grid as mines, and can also change a square back to being unflagged. The program tells how many mines are in the grid and displays the number of mines that have been flagged at any given time. If a zero square is clicked, all adjacent zero squares (recursively) are also revealed. All non-zero squares adjacent to the revealed zero squares are also revealed. The player can choose the size of the puzzle and the number of mines. The program displays a message when the game is over (a mine is clicked, or all non-mine squares have been revealed and all mine squares arecorrectly flagged).

- **Word Search:** There are multiple pre-generated puzzles to choose from, loaded using Ajax. There is a list of words to find in the puzzle. The player can click on the start of a word and then the end of the word. The letters between the first click and the second click are highlighted. If the selected word is in the list, the program checks it off in the list. If the selected word is not in the list, the program displays a message with that information, and the highlighting is removed from the letters. There is a timer that tells how long the player has been working on the puzzle.

- **Chess or Checkers** The program sets up the pieces on the board. The program allows the user(s) to enter two player names and assign each player a color. The program alternates between players, allowing each to take a turn. Only legal moves are allowed. In checkers, kings should have a different appearance and be allowed to move backward. In chess, you don't have to handle castling, pawn promotion or *en passant*, but you do need to allow pawns to move two squares for their first move, and only allow pawns to capture diagonally. Your program does not have to detect check or checkmate. For checkers, the program indicates when the game is over and which player won.

- **Sliding tile puzzle** The player can choose from at least three different puzzles, and at least one of the puzzles is an image (with each tile showing one part of the image). When the player clicks on a tile adjacent to the hole, that tile is moved into the hole. Movement of tiles should be animated so that tiles move gradually into their new position. There is a timer that tells how long the player has been working on the puzzle. There is a "new game" button that resets the timer and randomizes the puzzle.

- **Event calendar** The calendar shows a grid representing the current month, with the days of the week listed across the top and the day of the month displayed in each square. Each square also shows a number that tells how many events have been created for that day. There is a button to move to the next month, and another button to move to the previous month. Clicking on a square changes the view to show a table of the events for the day. Each event has a start time, a duration, a name, a place, and notes. Above the table is the day of the year (like August 27, 2012) There are two buttons on the day view. One button returns to the month view and the other button allows the user to create a new event for that day. The event calendar should be initialized with sample data so that it's easy to determine whether it has all of the required functionality.

- **Expense tracker** The expense tracker allows the user to define any number of categories of expenses. Each category has a name and a budgeted amount. The display includes a button that the user can click to add a new category and a button to add a new expense item. Each expense item has a category, an amount, a date, a form of payment (check, credit card, cash, etc.), and a description. When the user creates a new expense item, the program displays the expense item in the proper category, with expenses in the same category sorted by date. The heading for each category tells the name of the category, the budgeted amount, the total of all current expenses in the category, and the difference between the total and the budgeted amount. The table row for each expense item includes a button that can be clicked to delete that item. The expense tracker should be initialized with sample data so that it's easy to determine whether it has all of the required functionality.

Since summer term is only seven weeks long, I recommend that you use one of the following games for your project: Battleship, Sudoku, Concentration, or Minesweeper.

It's especially important to keep your project simple if you don't have previous experience with HTML and JavaScript, or if you don't have much experience with programming in general. If you want to do a game for which I have not described the game functionality, please talk to me about what you plan to do for your project and the game functionality.

Model and view

It's important to decide how you will store the information needed for your game. For example, in the game of Battleship, you need to store information about where ships are in the grid, the player's shots, the computer's shots, and so on. That kind of game data is called the model. In this class, we will be using JavaScript arrays and objects for the model. You will need to decide what kind of arrays and objects you need, and how many.

The views for your game will be various HTML elements. It's important to make the view independent of the model, so that the view can be changed without changing the model.

The Model View Controller (MVC) design pattern is very commonly used in software development. One of the advantages of using MVC is that the same model can be used with many different kinds of views. For example, you could start out with a very simple view that uses lines of text to show the locations of ships and shots in the grid. Later, you could make a view to show images instead of characters. You could use that view without making any changes to the model. You could incorporate animation, video, audio, and renderings of 3D models into the view also, and still not have to change the model. We won't be studying MVC in this class, except for the basic idea of separating the model from the view, but you should be aware that it is an important design pattern.

Project checkpoints

One assignment will be a project checkpoint that is worth 25 points more than regular assignments. The additional 25 points will be for demonstrating some part of the game logic for your project.

HTML5

Choose one of the following features of HTML5 to use in your final project: audio tags, video tags, canvas (drawing graphics with JavaScript), or local storage. Audio and video should be controlled by your JavaScript code, not by the user.
NOTE: Using local storage to save login information (as in Assignment 5) does *not* satisfy this requirement.

XML or JSON

Your project should load XML or JSON game data using XMLHttpRequest. You can use your work for Assignment 6 to satisfy this requirement, but the log-in functionality for Assignment 5 does *not* satisfy this requirement.

Notes

- Please don't require a password to log in to your game page.
- Don't include the animation from the Assignment 4 unless it is relevant for your game.

Turning in the final project

See the top of this page or the course schedule for the due date of the final project. Submit a zip file with your project files in it by uploading it in Canvas.

Score Summary

Here is the score summary for the project:

10	Documentation explaining how you implemented your game and how it satisfies the remainder of the requirements. The documentation should be in a file named `documentation.html`, with a link to the documentation file on the project description page.
20	Game grid that uses CSS and is generated by JavaScript
20	User input in the form of text, select options, and/or buttons
20	User input in the form of mouse clicks
20	Dynamic modification of HTML using innerHTML or DOM functions
20	Use of XMLHttpRequest to load JSON or XML game data that is displayed on the grid page
20	Incorporates HTML 5 audio tags, video tags, canvas, or local storage
120	Game logic code
250	TOTAL