CAPESTONE DESIGN DOCUMENT

Capestone is a JavaScript derived game that will rely on database data to operate different for every user and every use. The user will sign in and create a unique user account. From this point they will be able to sign in from anywhere and have the same game data available to them. After the user has logged in they will begin their Capestone adventure. Prior to page load the server will run an assortment of PHP functions to prepare the game data for use. With the use of PHP userData, heroData, itemData, dungeonData, and monsterData will all be gathered from the database in arrays then will fill JavaScript variables. Game.js will then compile the game environment and draw the game to a canvas on the page. As the user descends a staircase in game the state of the player will be saved back to the database with user stats and inventory list. This way the next time the user logs in the most recent game data for their game experience will be loaded and they may continue their adventure.

Arrays derived from database

heroData = armorClass, attackBonus, currentHP, damage, description, imagePath, maxHP, pass, userID,

username, x, y

(*)monsterData = monsterID, armorClass, attackBonus, currentHP, damage, description, dungeonLevel,

imagePath, maxHP, monsterName, pass

(*)dungeonData = dungeonID, code

(*)itemData = itemID, action, armorClass, attackBonus, damage, description, health, itemName, itemType

inventoryData = incriminator, userID, itemID

(*)Each of these is a multidimensional array. Each index is filled with the variables listed, which is an associative array.

Functions in game.js

Game.js is the main code for the Capestone game. Many functions are contained within which I will document to familiarize others with the structure of how it works up to this point. This is a living document.

attack() – A dynamic function that takes two arguments, assailant and defender, which is used to handle all combat within the game. If the user attempts to move into a square that is occupied and has an armorClass and a currentHP value, the assailant will attack that spot, whose values get passed as defender. This works with environment objects as well; very handy. Various messages are displayed to the user to explain what is happening during combat.

CAPESTONE DESIGN DOCUMENT

autoLoader() – Main function for the setup of the game. Automatically creates the canvas to draw on, draws the quadrants, creates a single instance of an enemy and an instance of the hero (the user's character), and runs the startGame() function.

being() – Main constructor for any creatures (enemies) or heroes within the game. The object constructor requires three arguments to be passed with an optional fourth. You must pass the image path of the creature you want to instantiate, as well as the x and y values. If you opt to use the options argument, you can pass it values as well, such as description and damage values.

canvasBackground() – Simply adds default values for the canvas background, including color and size.

checkDeath() – A function which checks for the death of the creature or user. Used in conjunction with the attack() function. A creature is deemed dead when it has zero or negative currentHP (current hit points).

clearConsole() – A function that clears the console. Somewhat unused at this point, just preparing for the future.

displayControls() – Displays to the user the control scheme of the game. If we add any controls to the game, we must update this function. displayControls() is also called in the startGame() function, so the user should know the controls when starting the game. This can also be called by pressing 'c'.

displayInventory() – Displays the inventory of the user. In the future this will be used in conjunction with equipItems() to show you the inventory you can equip.

enemyBehavior() – This function determines the behavior of every creature in the game. As of this point, the enemy will only move towards the user if it is within a square radius of ten spaces. Once it has 'seen' the user, it will move toward them no matter where they go, unless they fool them into getting behind an object they can't find a way around. We plan to improve this function using an 'a star' topology so the creatures can find their way around objects or other structures.

enemyLoader() – Adds the one instance of a creature on the page. Mainly used for testing purposes at this point. This may be used in the future to randomly place monsters on the playfield, with regeneration after a certain amount of turns.

environment() – This function is the default constructor for any environment objects contained within the game. If we wish to instantiate an object that has no movement, we can make it through this constructor. All environment objects are used to create destructible objects (trees, doors, etc.), impassible objects (rock wall, candelabra, etc.), and in the future we will use it to create objects the user can manipulate, such as doors or treasure chests.

equipItems() – Allows the user to choose which item they want to equip from their inventory. This function will change the user's attributes to reflect the item in which they are equipping.

CAPESTONE DESIGN DOCUMENT

fenceLoader() – Depricated. Originally used when the playfield was 10 x 10. It would draw a fence with an arrow inside of it which could be used to throw at an enemy.

heroLoader() – Creates the hero object which gets values passed from PHP. Default values are stored within our database upon signing up and can be saved by pressing 's' and running the saveData() function.

quadrantLoader() – A series of quadrant loaders numbering from one to four which draws the environment objects from a dynamically coded array. This allows us to place randomly the quadrants so that we can have semi-randomized environments.

randomBarrier() – Creates random, destructible barriers which are placed after the quadrants have been loaded. This adds another level on randomness to the levels. Between 2 and 30 barriers are created when this function is called.

randomInventory() – There is one argument for this function which gives the possessor a random weapon and a random armor. This will need to be changed to reflect the values given to me by the database.

redrawCoordinates() – When creating a turn based game using canvas, you need to clear out the canvas and re-draw everything whenever a creature or the user moves. This function does exactly that.

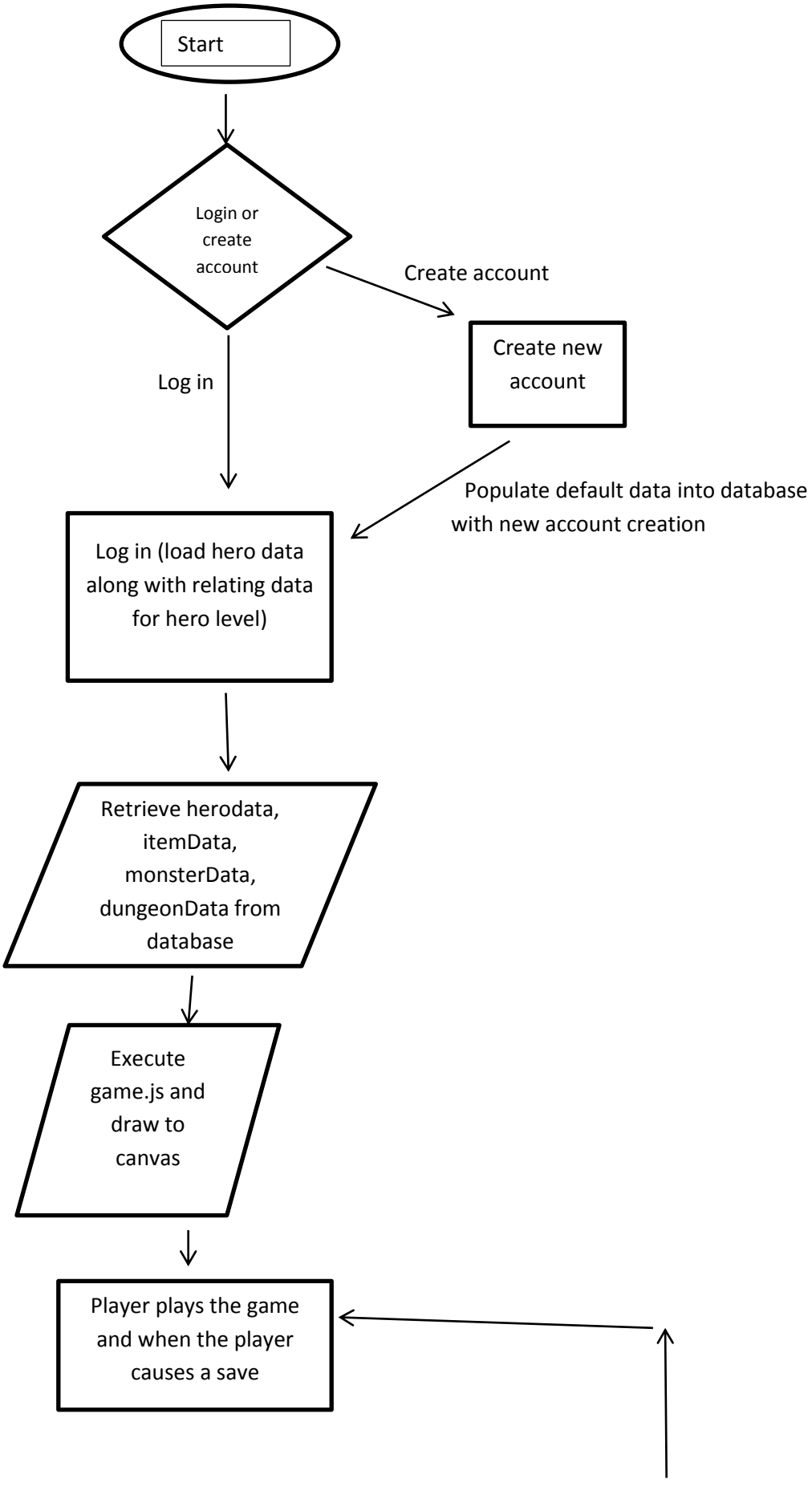
RNG() – Random number generator between one and the max number you decide when calling the function. Very simple but used often.

rollDice() – Does exactly the same thing as RNG() but the random number is between one and the max number.

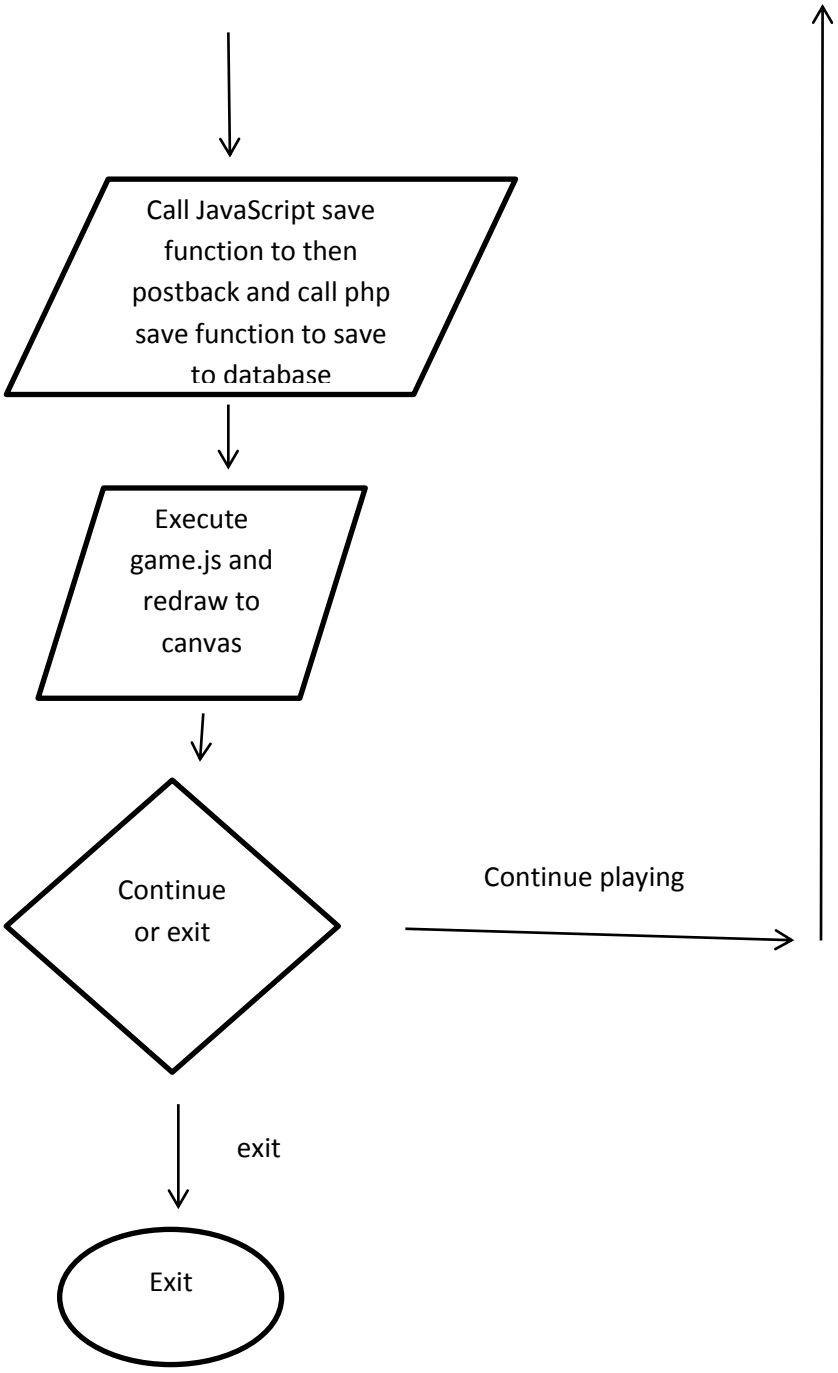
saveData() – Makes an AJAX call using jquery to send data back to PHP so it can be processed and placed back within the database. At this point, only saves the hero object.

document.onkeypress = function(e) – This is a function which gets called on the onkeypress event of the document. This allows the user to press any key on the keyboard and the game will be able to interpret it if logic is applied to that key press. Extremely useful. There is a console.log() contained within that shows the key code of the key that is pressed, so if we decide that we want to implement more functionality we can find the key code easily. This function also causes the enemies to move and the coordinates to be redrawn. Every time a key is pressed, everyone on the game field gets a turn, so therefore this determines the global counter of the game.

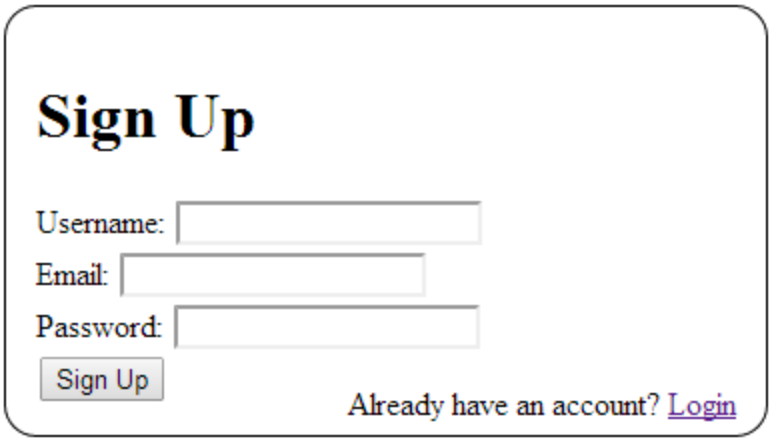
CAPESTONE DESIGN DOCUMENT

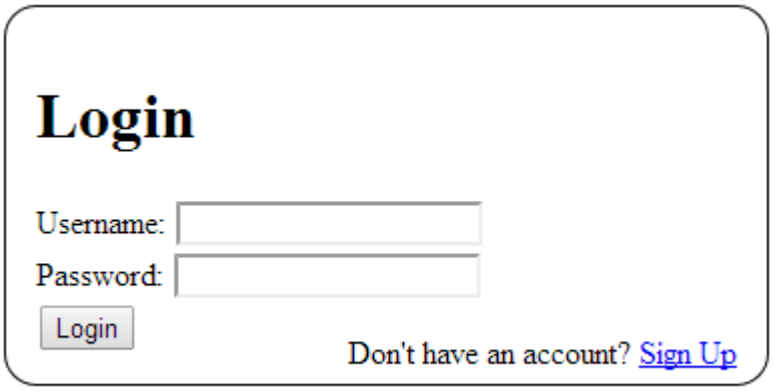


CAPESTONE DESIGN DOCUMENT

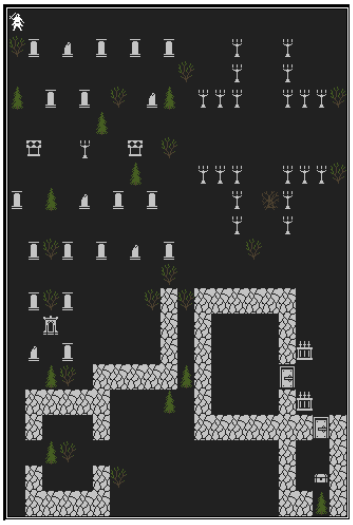


CAPESTONE DESIGN DOCUMENT





The user will first signup and must input a valid and unique username that will be displayed as their character name in game. After having an account they can sign in with that username and password.



CAPESTONE DESIGN DOCUMENT