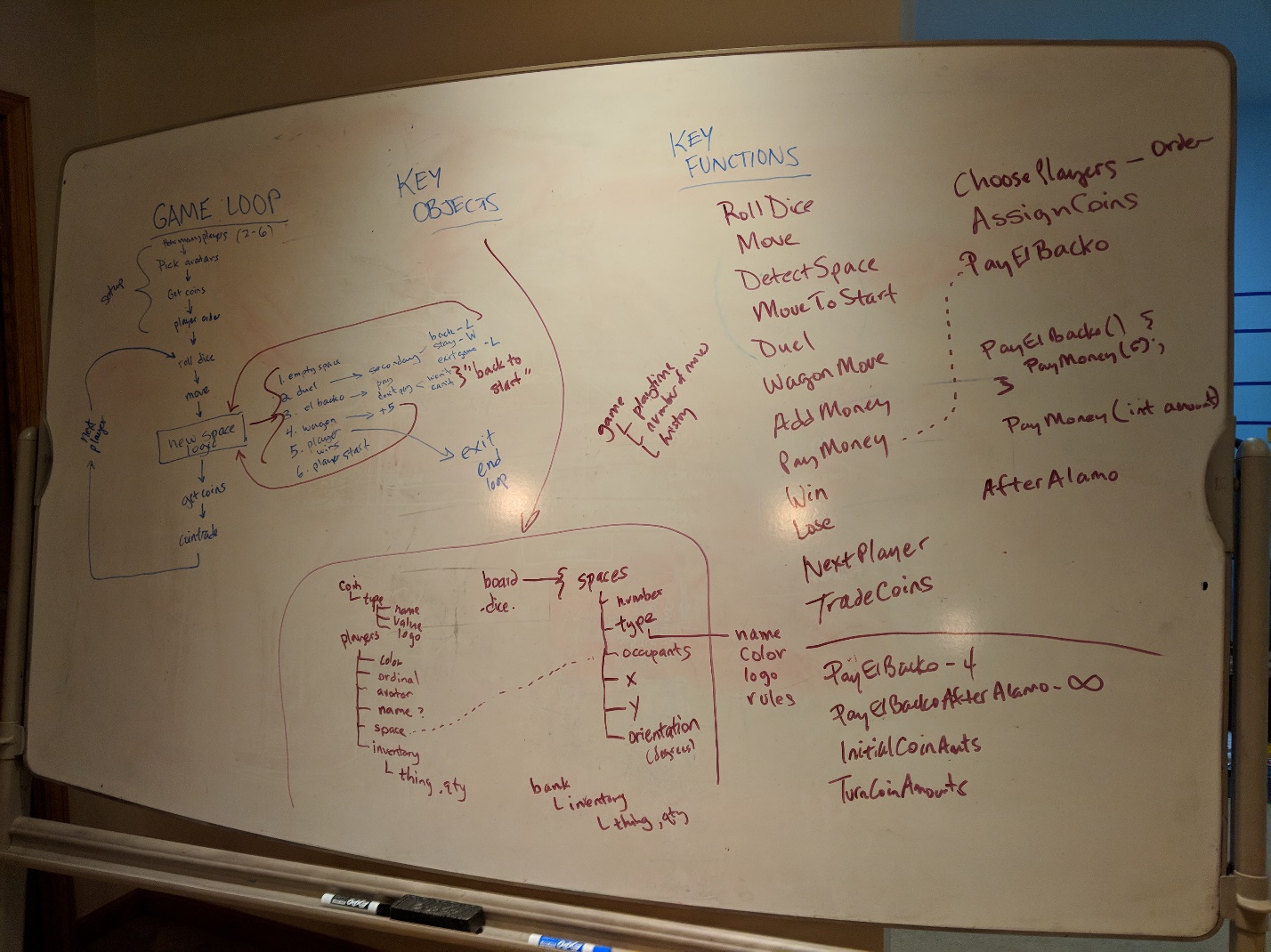
# Class 1

## Discuss the following

1. The Game Loop
2. Objects
3. Functions



# Homework 1.1

## Setting up NodeJS

Tasks

1. Install NodeJS
2. Install Sublime Text 2 (not Sublime Text 3)
3. Do a Pull in SourceTree
4. Install npm modules  
     
   \* Open Command Prompt  
   \* cd\Dev

\* cd elbacko  
\* npm install dict  
\* npm install readline-sync  
  
Notes: you will see some warnings when you do the two npm commands. You can ignore them.

1. Try some node programming  
     
   \* Open Command Prompt  
   \* cd\Dev

\* cd elbacko  
\* node  
  
> 5

5

> Math.max([5,4,3,1,9,10]);

NaN

> Math.max(5,4,3,1,9,10);

10

\* Press Ctrl+C  
\* Press Ctrl+C again

# Homework 1.2

## Node modules

Make a folder in c:\dev\elbacko – your name.

Create three files in Sublime Text 2, all within c:\dev\elbacko\[your name]:

**File 1.** spaceTypes.js

function SpaceType(name, size) {  
 this.name = name;  
 this.size = size || 1;  
}  
module.exports = SpaceType;

**File 2.** space.js

function Space(number, type) {  
 this.number = number;  
 this.type = type;  
}

module.exports = Space;

**File 3.** game.js

var spaceType = require('./spaceType.js');

var board = [];

// lay out the board

for (var i = 0; i < 10; i++) {  
 var newSpace = new space(i);  
 if (i == 0) {  
 newSpace.spaceType = spaceTypes.get("start");  
 } else if (i == 6) {  
 newSpace.spaceType = spaceTypes.get("elbacko");  
 } else {  
 newSpace.spaceType = spaceTypes.get("normal");  
 }  
 console.log("Space " + i + ": " + newSpace.spaceType.name);

board.push(newSpace);

}

**Try running your script.**

c:\Dev\elbacko\[your name]>node game.js

# Class 2

## Discuss the following

* GitHub Push, Pull
* The spaceType.js file and SpaceType class
* The “if” statement and the “for” loop for building spaces

Add the three types of space types to the game that are currently missing.

Stage, Commit, and Push your changes.

If a Push doesn’t work, do a Pull first.

# Homework 2.1

## Build the Board

Tasks

1. Update the for loop to have enough steps to build the whole board
2. Update the if statements to create the board in the right order and right types of spaces

# Homework 2.2

## Taking Names

Get player numbers

1. Add this to the declarations at the top of game.js

var readline = require("readline-sync");

1. At the end of game.js, add these lines:

var playerCount = readline.question("How many players?");  
console.log("");  
for (var p = 1; p <= playerCount; p++) {  
 console.log("Player #" + p);  
}

1. Try putting in different numbers to the question.
2. Try putting in non-numeric answers and see what it does!   
   (BONUS: how would we fix this? – see <https://www.bing.com/search?q=javascript+check+numeric> )

# Class 3

## Discuss the following

* Revisit GitHub stage, commit, push, pull
* for loops
* Creating a player class: player.js

var dict = require("dict");  
function Player(ordinal) {  
 this.ordinal = ordinal;  
}  
module.exports = Player;

* Load this into game.js
* Loop and list the players
* What other values could a Player have?
* Discuss the dictionary.

# Homework 3.1

## Make bank

The bank is a good use of a dictionary. Let’s make a bank. In game.js, let’s make a bank, just below the declarations part. By where we make the board (var board = []);

var bank = {  
 inventory : new dict({})  
};

What did we make? It’s a variable called bank, which is an object that has one property: “inventory” and the inventory has a dictionary.

Now go down to where you put in the number of players, and right above that, let’s create the bank’s inventory.

// create the bank  
bank.inventory.set("silver", 120);

Also add gold to it. Let’s have it start with 35 pieces of gold.

# Homework 3.2

## Tracking players

1. Create a variable at the top where we have the board[] and game{} called players. It will be an empty array.
2. Add each new player to the players array. Inside the loop at the end of your file, where you are creating each player, put this line before the } part of the loop.

players.push(newPlayer);

After this the players[] array will have a copy of each player.

1. Add gold and silver to each person’s inventory.

newPlayer.inventory.set("gold", 5);

Do this within the loop where you are creating the players.

# Class 4

* Look at homework
* Add silver to opening inventory
* Set values in the code for initial gold and silver
* Review syntax of our code

Build a game loop

* Add a player pointer
* Print current player
* Increment pointer
* Start over pointer

# Homework 4.1

## Rolling dice!

Start by going up near the top of the code – under all the variables setup, and make a new function called “rollDice”

function rollDice() {

}

Then inside your game loop, let’s call it:

while(true) {

rollDice();

… (this is where the other code already is)

}

Then inside rollDice, output something:

function rollDice() {

console.log("Dice rolled!");

}

# Homework 4.2

## Randomness

First, let’s make a new function and put it above rollDice. Paste this in.

function getRandomInt(min, max) {

return Math.floor(Math.random() \* (max - min + 1)) + min;

}

Let’s change what’s in rollDice to

function rollDice() {

PUT YOUR CODE HERE

}

On that line, call getRandomInt to roll dice. What two numbers should you pass in? Put those values into a console.log. Ask questions 😊

# Homework 4.3

## Loop Until Game Over

Right now, we have our game in an “infinite loop” because it’s in a **while (true)**

Let’s change it to use a function so that the game knows when it’s actually over.

First, make a new function in the functions area (by getRandomInt) called isGameOver. It will have no parameters.

Make it just return the value false.

Then change your game loop to check isGameOver() instead of true.

# Homework 4.4

## Game Over Check

How do we know if the game is over? There are two conditions.

1. **One of the players lands on the “end” space**
2. **?? what is this one?**

Let’s change isGameOver to check these conditions.

Start by declaring a variable called “isOver” in the function and start it out set to false.

Then we have to check if any players are on the “end” space:

players.forEach(p => { if (board[p.space].spaceType.name == "end") isOver = true; });

We’ll talk about what this line means in class.

At the end of the function, return isOver as the value of the function.

# Class 5

* Look at homework
* Add coins after each round

# Homework 5.1

## A general way to add and remove money

We add and remove money all the time while the game goes on.

Make a general function for this, called **addPlayerMoney**

The parameters would be

* player
* type
* amount

The type would end up being *gold* or *silver* when in use.

In the function, add the amount to that player’s inventory.

**player.inventory.set(type, player.inventory.get(type, 0) + amount);**

# Homework 5.2

## Use the general money function every round

Let’s change what happens in **addPlayerMoneyRound**

1. Inside this function, call our new function from Homework 5.1 instead of what we did before. Pass to it the appropriate three values, based on the three parameters that are in that function.
2. Change the amount of silver added each round to a variable. An ALL\_CAPS variable like the others that we have. This means that you need to create this variable, put it up with the other ALL\_CAPS ones and then use it in addPlayerMoneyRound.

# Homework 5.3

## Inventory Output

We’re usually going to want to know how a player’s inventory is, and the player also will want to know!

Let’s make a function called **outputCurrentInventory**

It should have two parameters

* player
* description

We’ll use description later.

Then inside the function, do a console.log that tells the player’s number and their current silver and gold quantities.

## Back to Start !

Something else that will happen quite often, is the player gets sent back to the start. Or they choose to go there.

Make a function that does two things

* does a console.log that outputs “BACK TO START!!!”
* moves the player to the beginning space

It will have one parameter. What is it?

# Homework 5.4

## Let’s Move

It’s time to start moving players around the board.

Create a method called **handlePlayerMove**

Two parameters:

* ???
* number

The number is how many spaces to move.

Inside the function, you should update the player’s current space. Try this to see what happens.