**Safe House**

**Personnel:**

Blanton, Zachary

Plummer, Kalen

Spivey, Kameron

**CS 499, University of Kentucky, May 4th 2016**

**Disclaimer:**

This project has been designed and implemented as part of the requirements for the CS-499 Senior Design Project for Spring 2016 semester. While the authors make every effort to deliver a high quality product, we do not guarantee that our products are free from defects. Our software is provided “as is”, and you use the software at your own risk.

We make no warranties as to performance, merchantability, fitness for a particular purpose, or any other warranties whether expressed or implied.

No oral or written communication from or information provided by the authors or the University of Kentucky shall create a warranty.

Under no circumstances shall the authors or the University of Kentucky be liable for direct, indirect, special, incidental, or consequential damages resulting from the use, misuse, or inability to use this software, even if the authors or the University of Kentucky have been advised of the possibility of such damages.

**Abstract:**

Safe House was originally a board game that has been translated into a game that will run on the Apple iPad. Built as a local, pass-and-play game, the games objective is to fulfill your character’s individual win condition, using the different game mechanics that are presented. Using different cards presented in the game (Investigation, Action, Equipment) as well as special abilities unique to your character, you are tasked with uncovering who other players are throughout the gameplay, and decide whether they are friend or foe. Using that information, you battle the other players throughout the game to fulfill your win condition.

**Introduction:**

Safe House is a very fun board game, however during the transition to the electronic age, not as many people sit down to play physical games with friends. This is why we have created the iOS version of this game, so that we can have people playing the game Safe House, without having the physical game present. This is needed so that we can get people playing the game, that may never have otherwise if it was not in digital format. Our customer, Mr. Tony Elam, is a board game enthusiast that has created many games in the past, and also owns and plays many other games, and is hopeful that Safe House can have future applications as a game, while also as possible use for other ventures as well. The hope is that people interested in quick games of strategy, skill, and deception will be interested in the game, and thus wish to play.

**Product Requirements:**

The Requirements for this process were laid out, and then agreed to by our customer, and are as follows:

* Playable board with 7 separate playable zones on said board
* Hit point board with up to 14 points to track the hit points of all players in the game, whether you know the identity of that player or not.
* Support for from 3 up to 8 different players
* Support for 3 different types, or affiliations, of characters (Terrorists, Counter-Terrorists, and Neutral)
* Ability to engage in combat between the players in the game, using their characters Hit Point values. Dice are used to determine the attack power of a character, and then combat is decided between the two dice rolls from the two characters involved in said battle. The damage done is determined by the difference in the attacking player and the defending players dice rolls. If the defender rolls higher than the attacker, then the defender takes no damage. In the case of a tied dice roll, no damage will be done
* Equipment cards will be created, and useable. Equipment cards can be equipped to characters to add extra boosts in attacks, or certain abilities to the character, depending on the card (Possibly adding additional damage at the end of a successful attack, or attacking players in zones otherwise not reachable)
* One-Time action cards implemented. These cards do various actions, and are played immediately following being drawn, and are then discarded. When all action cards are used, the discarded ones are shuffled and then made into a new stack of the cards.
* Investigation cards. These cards will each have an action that is taken by the player who drew the card. The cards are meant to help reveal which players are what affiliation in the game, all of these orders must be followed, except for a certain character who has the ability to subvert these requirements.
* Player deaths happen when a player loses all hit points, and that player is eliminated from the game.
* Each character has a special win condition, and when that win condition is met, that character will be declared a winner of the game.
* Each character has a special ability that will allow them to do special things during gameplay that is unique to that character, such as doubling the damage from a successful attack, or counter-attacking immediately after being attacked.
* User interfaces are created so that each player can see the board, the hit points of the other players, and will allow some interface for that player to see the equipment cards that he has.
* Local pass-and-play multiplayer will be implemented so that players can play a game together on one local iOS device.

**Product Planning:**

When first given this project, our sizing estimate was around 2500 lines of code. We assumed that between basic HTML files, some CSS, and then quite a bit of JavaScript, we could get the entire project done in that amount of code.

However, we drastically underestimated our code volume size. The following is a count of each file, and then the total number of lines for every file:

* index.html = 389 lines
* game.css = 908 lines
* classes.js = 368 lines
* game.js = 2640 lines
* Three\_JS\_Build.js = 1491 lines
* **Total lines of code:** 5796 lines of code

As you can see, we drastically underestimated the amount of code. We simply did not realize how much would go into making a full-fledged game, and also making it have all the features that we were required to have present in it. As we continued, we kept seeing the size of our files increase, and while there are probably some things that could have been done better, however we were already too invested into what we were doing to go back and change now. We ended up with nearly double the amount of lines we thought. Mainly, our game.js file was quite a bit larger than anticipated.

**Schedule and Milestones:**

Our schedule was as follows:

* Friday, January 29th: Project and team was assigned
* Monday, February 1st: Met with our customer to talk more about the project
* Wednesday, February 3rd: Project webpage was due with our project requirements
* Friday, February 5th: First team meeting occurred, project was planned out and duties were assigned among team members. Project requirements were written out, and then the research into how the game would be coded began to take place.
* Friday, February 12th: Decision was made to not use a game engine or Objective-C/Swift and to use HTML5, CSS3, JavaScript, Three.js, along with wrapping it using Apaches Cordova.
* Wednesday, February 24th: Design/Data flow was constructed of the game. This meeting was probably the longest of the semester, with countless hours spent trying to work through the logic of the game and determine how all of the key parts would work together.
* Wednesday, March 2nd: Design document was constructed, and Midterm presentation was practiced multiple times.
* Monday, March 7th: Design document was due on web page, Midterm presentation was presented
* Wednesday, March 16th: Classes for the objects in the game were completed. Initial Dice rolling animations were completed. Creation of Characters started.
* Thursday, March 31st: Characters were implemented. Attacks were created and attack testing began.
* Monday, April 4th: Testing plan was due on the product webpage, testing meeting with instructor was held.
* Thursday, April 7th: Animations for moving pieces on the board and the HP board were finished. Win conditions and special abilities were started.
* Monday, April 11th: Code review meeting with instructor was held
* Friday April 15th and Saturday April 16th: Win conditions were finished and tested. Pass and play features were tested. Special conditions were finished and tested. One Time Action cards were finished and tested.
* Monday, April 18th: Final presentation practice at presentation site was held
* Friday, April 22nd and Saturday April 23rd: Investigation cards were implemented and finished. Testing was done on all cards and the game was played all the way through multiple times. User Interface was fine tuned.
* Wednesday, April 27th: Final project presentation was given
* Thursday, May 5th: Final meeting with Customer and Instructor was held.

**Platforms, tools, and languages:**

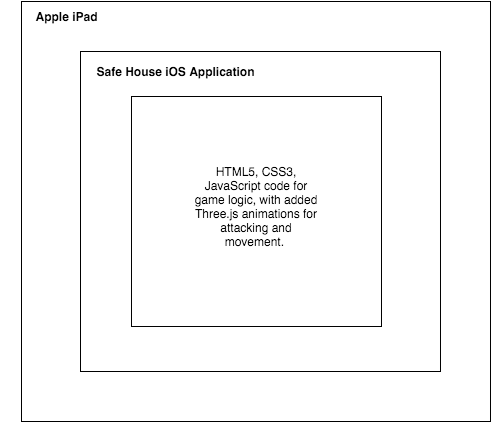
The code for this game was written entirely in JavaScript, CSS3, HTML5, and Three.js. The game logic was written in these languages to allow for effectiveness even on older hardware, the ability for all members of the team to test their code and work on developing the product, even without an Apple development environment, and for the ability for future use as a possible cross platform game. Three.js was used because it had pre-existing functions that allowed for certain animations, such as dice rolls, whereas other animation software like WebGL would have caused use to write our own functions, and would have taken longer amounts of time. Because of the way our game was, it is fully functional and playable on modern web browsers, and was tested with Google Chrome, Mozilla Firefox, and Apples Safari, as well as our built application for Apple iPad devices. To compile our code to be built for the iPad in Xcode, we used Apaches Cordova, which is an open-source mobile development framework that allows the use of standard web development technologies for cross-platform development. We used Cordova because not all of the team had Apple MacBook’s to test on, so we needed a way to make normal code that could be tested from anywhere into a mobile application. We used Xcode to build our project for deployment onto an iOS device. Xcode is the widely used Apple development environment, and this allowed us to simulate different iOS devices to deploy and test the current build of the game, while also allowing us to place restrictions on the application itself when it was deployed to the device.

**Design:**

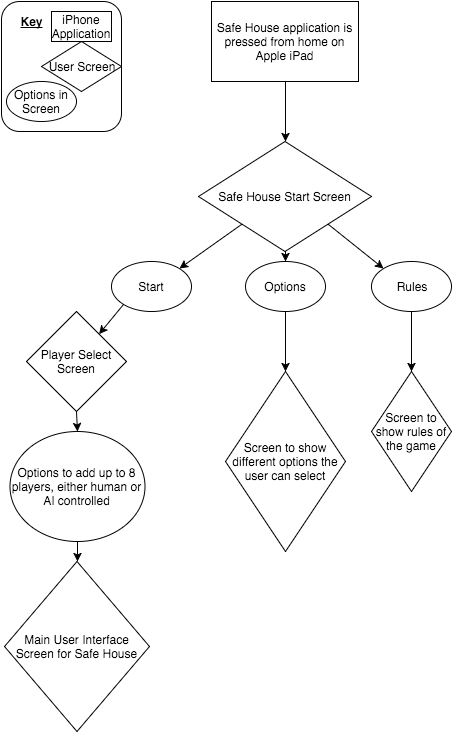
Our design started with a clear overview of the game and the process that went into understanding the game, as well as clearly stating what would happen when certain game conditions were met.   
  
During a typical Safe House game, depending on the number of players, preexisting character cards are randomly assigned, each having a special win condition and special ability. Once each player has a character and a color, the turn order is decided and a game will begin. The Safe House game board has 7 zones on it, marked with numbers from 2 to 12, that a player will move to when they roll the dice at the beginning of their turn. Depending on the zone that the player lands on, different actions can be taken. In some zones, players will draw cards, either Investigation, Action, or One-Time Action. These cards will perform different actions during the game, and then the players are given a choice of their next action. They may either attack players in the same area, unless they have equipment that gives them the ability to attack elsewhere, they can activate their special ability, or they can end their turn. The game ends when a single character’s win condition is satisfied.

Safe house was designed to be playable on Apple iPads running iOS 9.0 or later, and was tested on such physical devices. The game is also playable straight through a web browser. We developed in HTML5, CSS3, JavaScript, and Three.js. These technologies were wrote in text editors Atom and Notepad++, and then the code was compiled, using Apaches Cordova, into an Xcode project, where it could be deployed onto an iOS device.

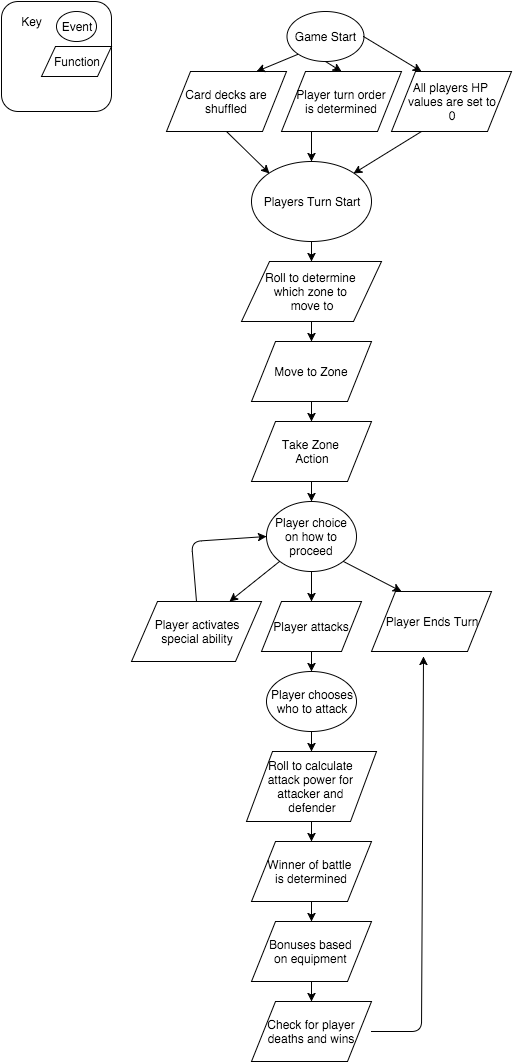
From the start, the game was designed as an application, which are designed as follows:



From the iPad running iOS 9.0 or later, the application will be launched, which will then run our game logic code to run the game itself. When a user presses the application on their iPad, they are presented with a welcome screen where they are given three options: To Start a New Game, a screen for game options, and also a choice to see the rules. If the player selects the rules option, they will be given a page where they may read the rules of the game. If they select the options choice, they will then be given the choice to select between a few different options. If they select the New Game choice, they will be taken to a “Player Select” screen, where they will be given the choice to choose how many players and what type (Human or Computer) players they will play with. They will then be taken to a screen that will be the main user interface for the game. This is portrayed as below in our UI diagram.



Once we have gotten to the main user interface screen, the main game will execute once the user has selected how many players that they want to play in the game and their colors. Once the number of players has been selected, the game will randomly assign characters to each player in the game. It will also shuffle all “decks” of cards in the game to randomize their order. Player turn order is randomly chosen. All players HP (Health Points) are set to zero. Now a players turn may commence. First a player will roll to see what zone they will be moving to. Once they move to the zone, they will take the appropriate action based on the zone that they moved to, whether that is taking one of the cards you may draw from the decks, or the appropriate actions the game tells you. Once that is over, the player then has three options how to proceed: They may either attack a player in their zone (or another zone depending on equipment attached), they can activate their special ability (which each character has and is unique), or they may end their turn. If they player chooses to attack, they will choose who to attack, roll to calculate their attack power, then the winner and loser of the battle is determined, once that happens, any bonuses based on equipment are applied or activated. It is then checked to see if the defending player died. Then it is checked to see if any player won. This can be seen from our gameplay diagram, which is shown below.



From here, our game logic refers to a state machine implementation to function. An example diagram of this state machine is show below. Each “state” became a case in JavaScript, and from there we could assume the next state or not.

****

**Implementation:**

We ran into some complications during development, some that were of our own cause, some that were based on time constraints, and some that were based on other limitations. Because of this, some features had to be altered to fully meet our customers’ expectations. One of the gameplay tweaks was that we only allowed one equipment card to be used at a time. There was no restriction at first, and when this was allowed, it led to unfair gameplay, and some players being about to control the entire game, and led to an experience that we did not feel was up to par with the rest of the experience that is Safe House. The next tweak we made was to force the play of cards as soon as they were drawn. If you allowed players to hold onto Action cards, it would cause one turn swarms where as soon as they discovered the identity of a character, they could completely eliminate him from the game, which had the same effect that multiple equipment cards that were equipped. With the consultation of our customer, who was much more advanced in gameplay of board games, he agreed these were acceptable things to do.

Because of the timeframe that we had to implement so much of a complete game, some things had to be restricted, and some had to be skipped completely. One of those was an AI. While we thoroughly wanted to implement this feature, the time constraints simply did not allow enough time to do everything, and our priority was getting a fully functional game first, then if time allowed going back and adding things at the end. We also wish we could have implemented network multiplayer, however pass-and-play was the first goal and that was achieved. Customization also would have been a nice feature to implement, however, it also was cut out of the final product due to time constraints.

Our **unit testing plan** was used Seeing as how our project is a fully functional game, the testing for such will be extensive. The game has many components that must be blended together to create gameplay (which will be tested later on during our integration testing). The aspects of this testing will range from objects, “action” functions, animation functions, win conditions, special abilities, and the different types of cards. These are as follows:

* Player objects, the objects that are created to initiate an instance of a player in the game, will need to be tested. These objects are, for the time being, created with 8 values that will determine the different aspects of that “Player”. These are: Character, Player Color, Equipped, HP, Attack Pts, Alive, Type, and current region. First we must check to make sure we can edit and call these methods of the class, and then we printed them to the console of a JavaScript debugger to make sure they are being stored correctly. Since we are using JavaScript, variable type does not matter as much, but we do test to make sure that what we want as Integers remains that, and what we want as Strings or Arrays remain those as well  
    
  *This test case went as planned, and the results were as expected for all characters.*
* Character objects, the objects that create the instances of the different characters available for players to play in the game, have 8 items that need to be tested. Char\_name, HP, img, affiliation, win\_condition, special, win\_condition\_text, and special\_text. As with player objects, we must make sure that all of the values are accessible within an instance of the object, as well as we must test that they contain the correct values/functions by outputting those values to the console in the JavaScript debugger as well  
    
  *This test case was done using the JavaScript debug console. By initiating the game, we were able to check the values of these throughout, and all test cases returned correct.*
* Investigation cards as objects must be tested for their values, card\_title, img, action\_function, and card\_text as well. They are tested that first, when calling a constructor with parameters that the new instance of an Investigation object is created, and that that instance has the appropriate values within it. This is still done with the JavaScript console in the debug window. Action cards and Equipment cards have the same 4 values that Investigation cards do, and are both tested in exactly the same way that Investigation cards are tested.   
    
  *The Investigation cards were the most difficult to test. Each card had to be drawn throughout gameplay, and when noted, we had to test to make sure it worked correctly for the right action it was supposed to take, to make sure it did nothing if the investigation was wrong, and also make sure that they worked and had the correct effect if the person they were investigating was the character that can lie. These were charted and cataloged. Action cards and Equipment cards are tested in the same way that these were, and recorded.*
* The Game object must also be tested for its implementation. Within this object, the game flow occurs, with the changing of player turns, attacks, special abilities, end\_game, roll\_dice, move\_to\_zone, zone\_action, and end\_turn. As with the previous objects, we test that these values are filled correctly by the JavaScript debugger and then we test that the class correctly appropriates characters when it is implemented by checking elements of our player’s array, again with the JavaScript debugger.   
    
  *These items were also tested using the JavaScript debug console.*

For **integration testing**, we will need to test the elements that integrate our different objects, as well as we will have to test the elements of our HTML5/CSS code that will integrate the different elements from our objects with the user interface of our game screen. This process consists of testing the following tests:

* Each instance of a character has a certain win condition, the actions that they must complete in the game for them to be declared. These are done within our game as functions, and each one must be tested to ensure that it does what it Is intended to do, so that that player may be declared a winner. These tests are done by first identifying what the are conditions for each character to win, and then manipulating the game in a way that should cause the win condition to get called and return true, that way we know that a character has indeed won the game and can be declared a winner. This must be done for all 10 characters.  
    
  *This was successfully done for all 10 characters. We manipulated code within the game, like increasing damage, to force a player to win and thus getting his win condition called.*
* As with win conditions, each character also has a special ability that they may activate at any point during the game, that allow that character to perform a special action within the game. These, also like the win conditions, must first be identified to see what should occur and if that character should be revealed to the rest of the players in the game. Then, we must test these functions to ensure that their effects are as specified by the character’s special ability text. For example, if one of them said that when a player attacks, and inflicts 3 or more damage, that the attacking player heals 2 points of damage, we must first make sure the attack was successful, then make sure the damage that the attacking player inflicted was 3 or more. Then, we must make sure that the attacking player then heals 2 points of damage.   
    
  *Each special was tested at the appropriate time, and all specials functioned as coded.*
* Each character also has an image and text within the object itself that will need to be shown to the user on the screen for them to read and understand their characters. This image and these text methods must be taken from the object and then displayed within the HTML5/CSS code. This is tested by first making sure that the variables themselves are accessible and that they contain the information that we need, and then we must actually access these and take the information in the and display it to the user.   
    
  *These were fairly simple to test. We just needed to make sure all the images and text appeared on the screen correctly formatted when called for each “card” in the game*
* Investigation cards have a few different aspects as well. They have images and also card text, that we must test and make sure contain the information that we want, and also must test to make sure that they will correctly mesh with the HTML5/CSS code necessary to display them to the users so that they may read the cards. We then must test the actual action function itself, to make sure that they act as the text lets us know that they should. For example, if a player uses an investigation card that says “I bet you are a Terrorist. If so, you receive 1 damage” on another player, we must check to see if the other player that the card is used on is a Terrorist, if so we give that player 1 damage, and if not we do nothing.   
    
  *The main function of this test case was to make sure the screens for investigation cards showed up, due to the pass and play nature of the game. This was tested for all cards and was determined to work.*
* Equipment and Action cards are very similar in how they are tested to Investigation cards. Their images and card text must be read from the object and then shown to the user with the HTML5/CSS code, and then they also have action functions with control the actions that both types of cards will make. For an example of Equipment cards, one might say “When you move you may roll twice and then choose which result you’d like to choose”. For something like this, we must test that the function knows when a movement for the player with that equipment card equipped occurs, and then we must make sure that it recalls the dice roll function, and stores both of those numbers, and then we must test that it gives the player the option to choose between both of the numbers of those rolls. Action cards, for example, might say "Pick a character (other than you) - Heal 1D6 points of damage." For this instance, we would need to test that the player who played the card is able to choose another character, and not able to select himself. Then we must check and make sure that the function allows the other player to roll a dice, and then their damage heals according to the number that is rolled.   
    
  *Similar to above, we had to test the Equipment and Action cards to make sure they displayed on the screen, and that the player was given appropriate screens to interact with if the card required them. For each Equipment and Action card, these were tested and proven to work correctly.*
* We must also test and make sure that the functions within the game object that interact with the other classes work correctly. For change color, we must make sure that the player who we pass to the functions color is changed appropriately, and that we can see that color change. This also goes for their type (Human, CPU) as well. We need to also test and make sure that the end\_game function correctly ends the instance of the game, and also that it displays the correct players that have won the game and shows the players that. We must make sure that the dice roll function correctly and randomly generates a number from a dice roll, and also appropriately shows that to the users. Our draw card functions should correctly pull information from a card that it has been passed as a parameter, and then correctly draws that information on screen for the user  
    
  *This test was to just make sure trivial aspects worked correctly. That the game officially ended when it was supposed to, the dice rolls returned the correct number and that number was a random number. These were all passed.*
* The move zone function should be tested as well. We must test that it takes the player it is passed as a parameter, and then moves that player into the correct zone, based on the dice roll that it is given. We also must test and make sure that it correctly delays for a few seconds, and then calls our zone action function. The zone action function should be tested to make sure that it correctly performs the zone action specified for that zone for the correct player given to it as a function. These elements must be tested by trial and error, and manipulating parts of the game to correct give us things that we want to see, like testing a certain zone, and then testing that specific zones action.  
    
  *The move function was tested to make sure that when called for each separate zone, that object was moved to that zone. Also that when players damaged, the move function moved the players marker on the HP board correctly. This test case was extensive because of the nature, but was tested thoroughly and passed.*

For **system testing**, we must test and make sure that the compiled application will work within Xcode, and then also that it may be compiled and placed on an Apple iPad and runs accordingly. All Apple devices that are running iOS version 9.0 or greater should be tested to make sure that they are running well on all platforms. Along with this in System testing, we must make sure that the game is only available in Landscape proportion, and not Portrait, for any of the devices. The user should be presented the game and then should play it to determine that all user interface portions are displayed and working correctly. The best way for this to be tested is for users to engage in playing the game.

*When playing the game, all of these things were “tested”. In reality, the only way to see if these things were working was to simply play the game multiple times, which we did, and everything worked great.*

**Future Enhancements and Maintenance:**

The future of Safe House could be a very bright one if played correctly. Not only is the game very fun, but with some future enhancements the game has a very great potential at possibly being profitable. One major future enhancement would be the implementation of network multiplayer. With some sort of interface to all people to play online, the game could reach an entirely different level of use. Another aspect that falls along with that would be the ability to play online, but cross-platform. Meaning a game with 5 players could have 2 players on different iPads, and then 3 more players each on a different laptop. Customization could be an interesting addition as well. If the users were given then ability to customize their character names, pictures, colors, among other things, it could provoke users to continue to play, and thus help promote the game even further. Another key thing, that was brought up among testing, would be the implementation of an AI or Computer controlled opponent. This way, if there was no one to play with, the user could still play a game with as many players as he wanted, just having them controlled by a computer player.

As far as maintenance is concerned, the only maintenance that would need to be performed, the main part of maintenance that would have to be done would be making sure all of the files and programs were kept up to date for new releases of iOS. For the files, the most recent versions of Three.js, fastclick.js, and jQuery are included, however those would need to be updated if the files ever needed to change for new versions of iOS being released, or if security holes were found in these files. As well as these files, the latest version of Cordova should be used if deploying to iOS. Cordova is installed onto local machine, and is a command line program that is simply to install, and step by step instructions are given at their website (Which will be documented). After updated versions were achieved, extensive testing would then have to occur to make sure everything continues to work.

**Conclusions:**

In conclusion, the entire team of team four has very much enjoyed our work on Safe House the game. The entire semester was a very large learning experience for all three members, and there are plenty of lessons to take away from this class. One of the main ones, which I am sure is a common one for a project like this, is learning to effective managing time on a large scale project, while also balancing it with other work. As students, we had to balance our time between this class and others, as well as to learn to dedicate time to exclusively work on Safe House and Safe House alone. Another key lesson learned was to understand the scope of what you are going to be doing before you start making promises. When we learned of the game, we immediately started talking about all of the things that we wanted to accomplish, and they were more “dream than reality”. When we finally dove into coding this game was when we realized the sheer magnitude of what we were asked to accomplish, and only then did we realize that our lofty goals may not be able to become reality with our time constraints.   
  
Looking back on development of Safe House, there are some things that we most likely would have done differently, had we a second chance to do them. One of those would have been to push for network multiplayer from the get go, versus the pass and play mechanism now. While the multiplayer works flawlessly, it certainly gave us trouble, and really caused a lot of aspects to be much harder in the game then they were intended to be. We also probably would have changed how we approached the project. As you can see from our schedule, we did not get much accomplished in the beginning because we underestimated the project, and then had to work extra hard to make sure we got everything accomplished that was required.   
  
When presented to Mr. Elam at the presentation, he seemed very happy with the product and very anxious for its future, because along with us, he does believe the game has an abundance of potential and a very bright future. Overall, we concluded that we would consider the project a great success, as not only did we accomplish the goals that we set out to accomplish, but we took away many invaluable lessons from the journey in developing this project, which we think was a very important thing.

**References:**

External code that was used:

Three.js - <http://threejs.org>

fastclick.js – <https://github.com/ftlabs/fastclick>

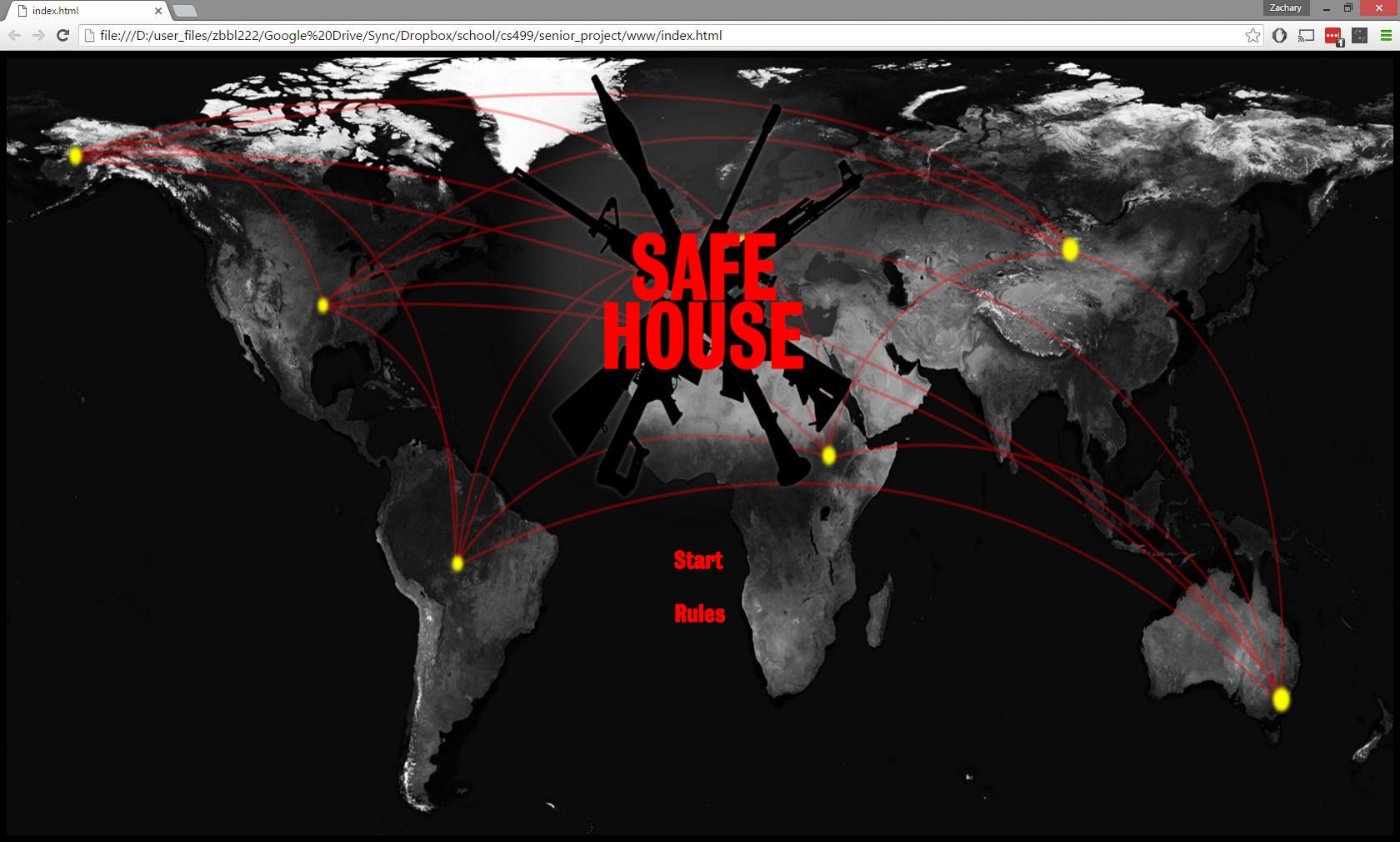
jQuery 1.12.0 – <https://jquery.com>  
Apache Cordova - <https://cordova.apache.org/>

**User’s manual:**

**Safe House Rules (Version 1.5)**

**Summary -** Safe House is a game of investigation, teamwork and survival.

Players take on the role of terrorists, counter-terrorist and neutral characters. Each character has different winning conditions and initially no one knows each other's identity. During the game you must uncover your teammates and enemies. Killing your enemies is always a top priority, as is survival!



**Preparation -**

1. Select number of players. Depending upon the number of players randomize the pool of characters as follows.

* 3 players, 1 Terrorist, I Counter-terrorist, I Neutral
* 4 players, 2 Terrorist, 2 Counter-terrorist
* 5 players, 2 Terrorist, 2 Counter-terrorist, 1 Neutral
* 6 players, 2 Terrorist, 2 Counter-terrorist, 2 Neutral
* 7 players (A), 2 Terrorist, 2 Counter-terrorist, 2 Neutral, (and one random additional either Terrorist, Counter-terrorist or Neutral)
* 7 players (B), 2 Terrorist, 2 Counter-terrorist, 3 Neutral (not George)
* 8 players (A), 3 Terrorist, 3 Counter-terrorist, 2 Neutral
* 8 players (B), 2 Terrorist, 2 Counter-terrorist, 2 Neutral, (and two random additional either Counter-terrorist or Ne:ut1ml)

1. Each players color marker is placed on the damage track position 0.

**Game Play –**

Each player takes a turn and then the next player clockwise takes their turn. This continues until someone achieves their victory condition.

A turn consist of the following actions.

1. Player press ROLL and moves to the region rolled (mandatory).

2. If the region can draw, the player must draw. Then follow the instructions on the card and the player log. If drawn follow the instructions on the CARD. (Place equipment or take action immediately).

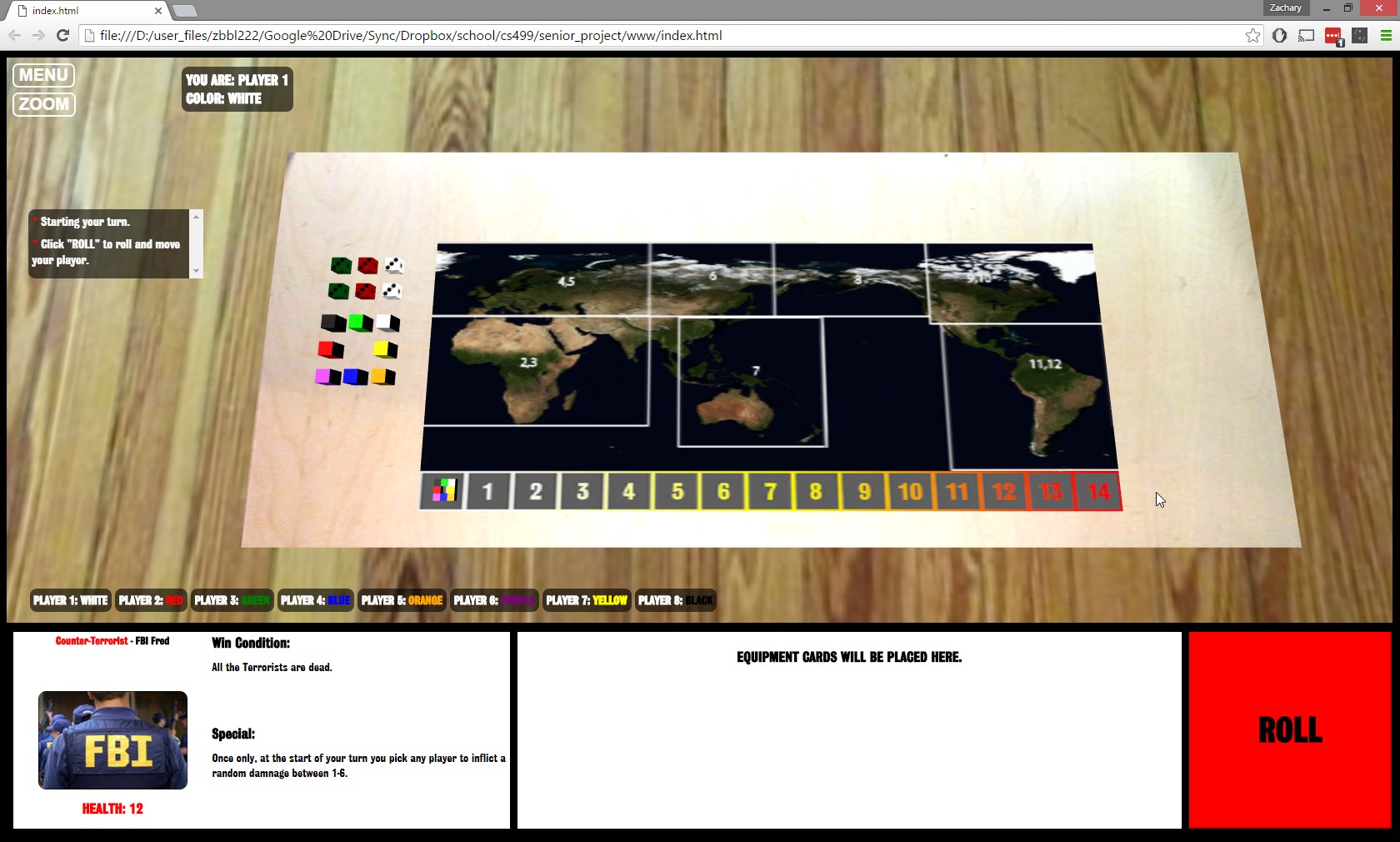
3. ATTACK other player(s). (Optional and requires other characters in "range").

**MOVE –**

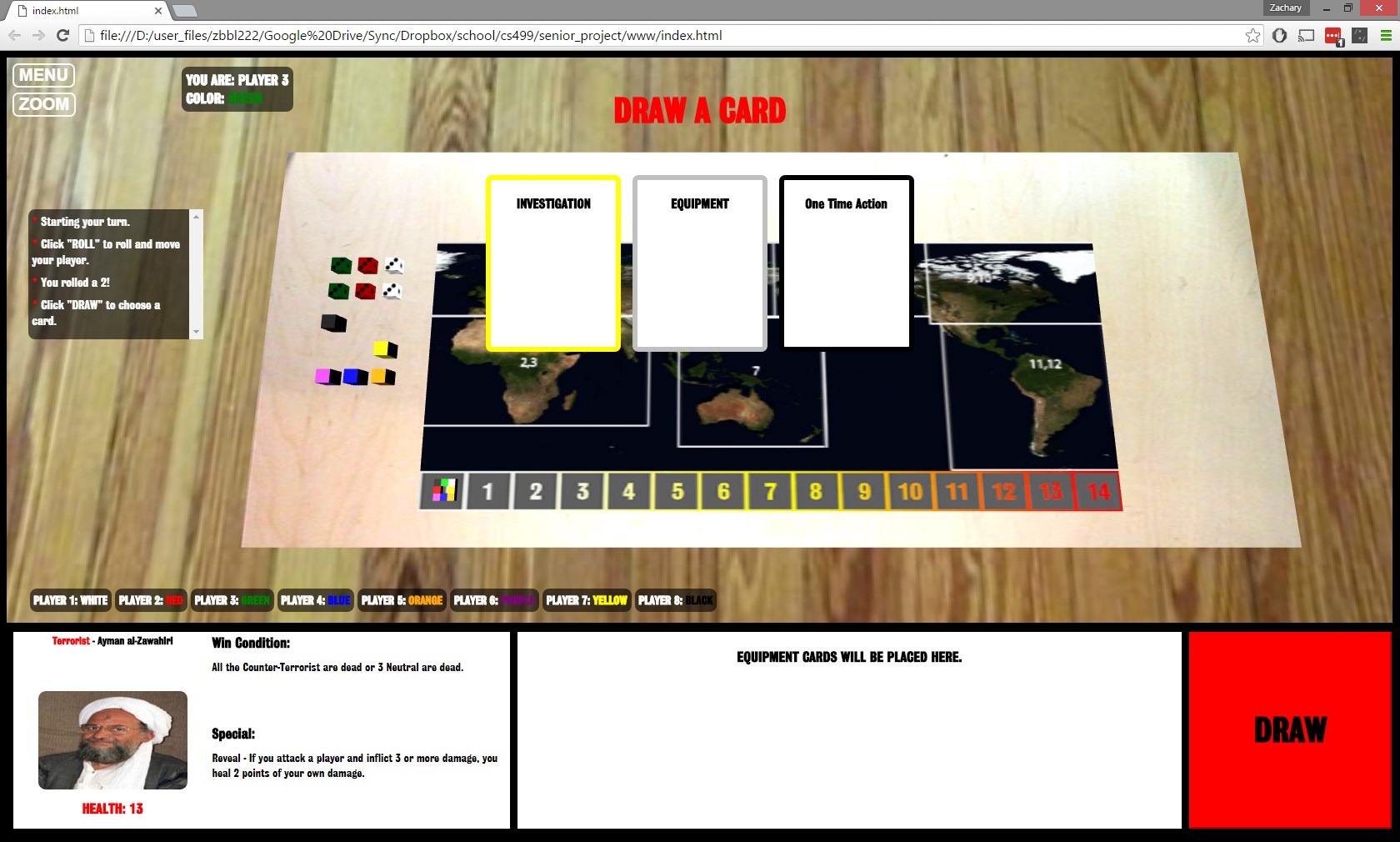
Rolls two green dice (2D6), move your marker to that area on the board.

Then take appropriate action:

* 2, 3-Take a card of your choice (yellow, silver or black).
* 4, 5 -Take a yellow card.
* 6 -Take a silver card.
* 7 -Safe House, either remain in the safe house area and "safe" or move to any area of your choice.
* 8 -Take a black card.
* 9, 10-Give two damage to any player or heal one point of damage to any player (you can choose yourself for either).
* 11, 12- You may steal an equipment card from any player (if no one has an equipment card then nothing happens).



**CARDS -**



Silver cards are Equipment- place them beside your color mini-sheet. They are now "equipped" and ready to use. There is no limit to the number of equipment cards you can have equipped.

Black cards are One Time Action- follow the instructions on the card then discard it (face up in a discard pile). When you run out of cards, shuffle discards and make a new stack of cards to draw.

Yellow Cards (Investigation):

Read card, then give the card to any player (do not let others see the card). The player that then receives the card must take the appropriate action based upon your affiliation- terrorist, counter terrorist or neutral. If there is no effect, then state "nothing happens" and return the card to the original player

**ATTACK -**

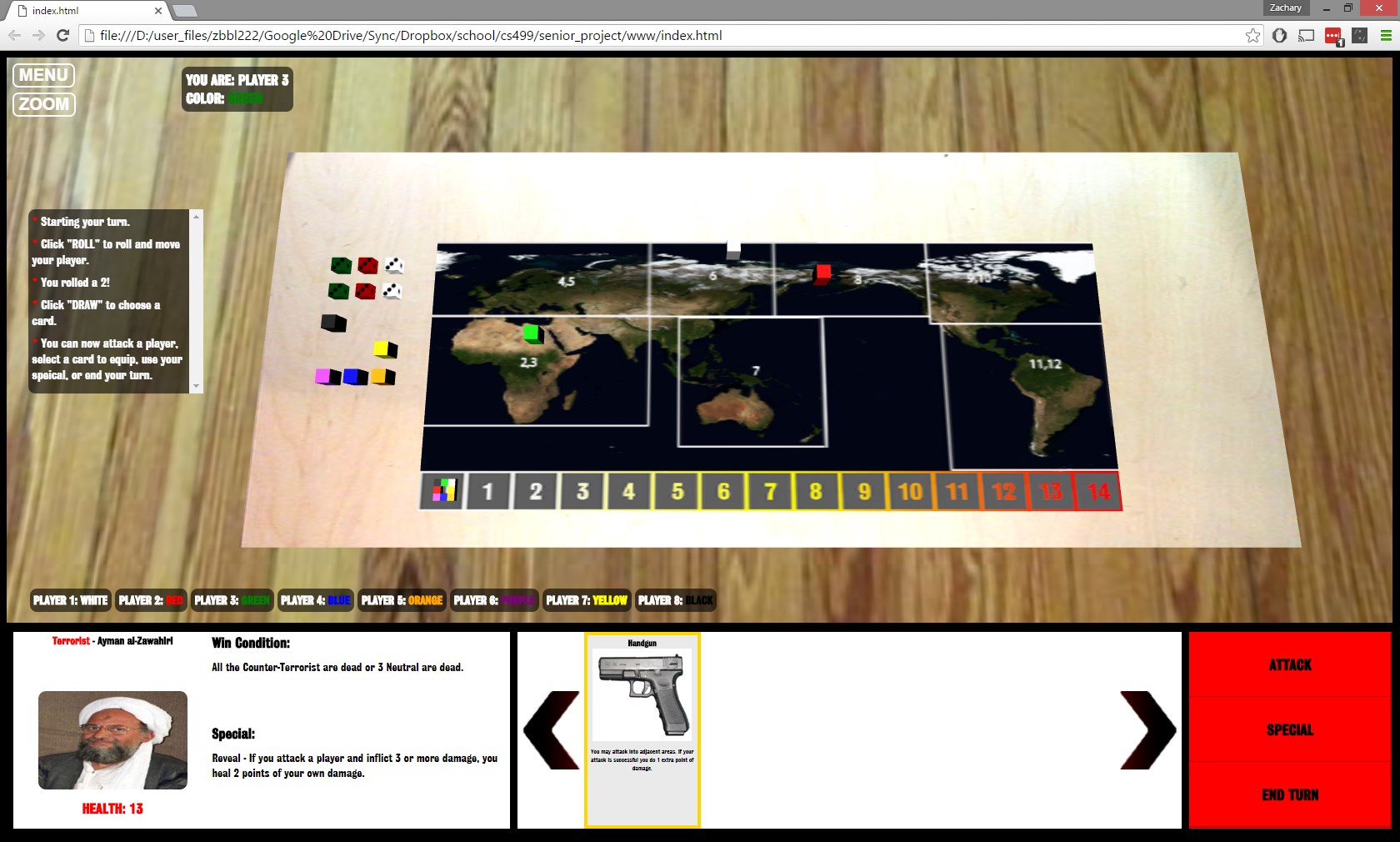
The last phase of a players turn is their optional attack phase. You may only attack another character/player that is in your same area or if an equipment allows for a ranged attack, you may do so.

If you choose to attack, pick your target and roll1D6 (red die), the defender (target) rolls 1D6 (white die). If the attacker's die is greater than the defender's die then the defender takes the difference between the two die as damage plus any additions or subtractions due to any relevant equipment the attacker or defender may have. If the defender's die is greater than or equal to the attacker's die- the defender takes no damage.

Example A. Attacker rolls a 5, defender rolls a 2 -no equipment involved- defender takes 3 damage!

Example B. Attacker rolls a 6, defender rolls a 4-attacker has an equipped handgun (1 extra damage) defender takes 3 damage!

Example C. Attacker rolls a 4, defender rolls a 4-attacker has an equipped Chainsaw (extra damage if attack is successful) defender takes no damage (attack was not successful).



**Player's Character Death -**

As you take damage your color marker moves along the damage track. If your maker lands on the space with your character's Hit Point (HP) value- you die and are out of the game. Your character is revealed and removed from the rotation.

**Winning the Game!**

If you fulfill your characters win condition, you immediately reveal your identity (if not already revealed) and declare that the game is over! All the players who fulfilled their character's win condition are winners!

Depending upon how the game ends, it is possible for terrorists or counter-terrorist who have been killed earlier to win due to other player's actions.

**Special Character Abilities -**

You can only utilize your character's special ability in return for revealing your identity (unless stated otherwise). Exception - Hassan (can lie during investigations).