**Alliance: Cold War Conflict**

Bryce Wiedower, Andrew Mock, Charles Yan, George Cook, Mary Yukich, Elizabeth Shores, Jon Oden

4/25/21

Graphical user interface

Description automatically generated with medium confidence

**Contents**

Final Tool Selection [1]

Final Team Member Assignments [2]

Game Theme [3]

Game Core [4]

UI Design [5]

Product Architecture [7]

Graphics/Animations [8]

Sound [10]

AI [11]

Networking [13]

Help [15]

Installation [16]

**Final Tool Selection**

**Game Engine:** Unity

**IDE:** Visual Studio 2019

**Programming Languages:** C# & Python

**Version Control:** Github

**Networking:** Photon Unity Networking

**Install:** Inno Setup

**Communication:** Discord

**2D Graphics:** Paint 3D

[1]

**Final Team Member Assignments**

**Bryce Wiedower Elizabeth Shores**

Team Lead Graphics Consultant

Sound Lead **Jon Oden**

GUI Co-lead Sound Consultant

Help Co-lead **George Cook**

**Mary Yukich** AI Lead

GUI Lead **Charles Yan**

Install Lead Game Core Lead

Game Core Co-Lead GUI Co-Lead

**Andrew Mock**

Networking Lead

Game Core Co-Lead

GUI Co-Lead

[2]

**Game Theme**

When choosing the game theme, our team wanted to pick a theme that fit well with the game we were basing ours off of (Node), and to pick a theme that could actively engage the user in the full experience that a game like this offers.

We considered options from haunted houses to farming to cyberwarfare but finally landed on a Cold War theme. Our thinking is that the conflict that was rooted in that of the cold war provides the division and conflict needed between the 2 players in this game, along with providing a great reason to vie over parts of the world as these two countries face off to become the world’s greatest superpower. This theme allowed for us to take a semi-realism approach to our graphics so that the look would not be cartoonish and 3D, but a more refined 2D look that fit the importance of what this conflict represented.

*Bryce Wiedower*

[3]

**Game Core**

The game core manages the game logic, AI, UI interaction, and is the core of the entire project. The core of our game is divided into multiple scripts, the most important of which is BoardManager.cs. BoardManager.cs include most of the logic for our game, it creates the gameboard, manages the end turn process, the sound effect, trading and more.

The BoardManager.cs also make sure the player makes a legal move for both Network game and AI game. We created the BoardManager.cs for the AI game, and the Network game just rewrote most of the functionality.

We also have another interesting file called Relationships.cs. This file stored all the information of each element’s connection in the game board into different list, so it is basically a small database. This helped us figure out the legal move and calculate the score. Game Core sort of goes hand in hand with the architecture of the game. Depending on how you are going to structure your code affects how the Game Core is going to be written.

There are few things that I could improve, and I would recommend you think about it when you are doing the game core. First is write comments, both AI and Networking will need your code. Second is divide your code to more files. Third is communicate with your teammate more, you will likely finish your game core in first month, ask them help if you need and go and help them after you finished. This is a team project anyway. *Charles Yan*

[4]

**UI Design**

Developing the UI was an iterative process.  We began by storyboarding the basic elements and layouts of the major scenes and their menus.  I drew up a rough diagram to determine the flow of navigation between those scenes and menus.  We then spent much of the first half of the project implementing basic functionality, the most critical features of the game, and navigation within the application.  During the last couple of weeks, we focused more on adding the less important game features and fine-tuning the elements that were already in place.

As the project progressed, we changed our minds about several decisions we had made regarding how the user should be able to interact with the application, particularly in the game scene.  Many of these changes were prompted by discussions with other teams and by feedback from our beta testers.  I had to be patient with the process of trying several different things before the UI functioned satisfactorily.

I had never used Unity before, so getting started was somewhat intimidating.  I began by watching a few tutorials from the YouTube channel called Brackeys and working along with his examples.  After a little bit of practice, I started working on our project and gradually learned more and more as I went along. As the team’s UI lead, my work intersected a lot with the game core and AI.  This required careful communication with Charles and George to make sure, for example, that any shared functions would serve each of our purposes.  We sometimes encountered problems if one of us made changes and pushed them to the main branch without realizing that another team member’s feature would be affected.  I found that it was very important to test the project both before and after making any significant changes so that if something went wrong, [5]

It was easier to identify the source of the problem.  As I designed the UI, I tried to keep in mind the various principles regarding user-friendliness that we learned in the GUI class.  Perhaps the most important thing that helped me make refinements, however, was getting input from other people.  I asked for my teammates’ perspectives when I was not sure where to place something on the screen or how I should implement some interactable element.

Looking back, I believe the most valuable input came from our beta testers.  By watching people play our game for the first time, I could see that there were many things I had taken for granted because I was so familiar with the concept of the game and our implementation of it.  Some of the things I had failed to consider included the responsiveness of menu items, the visibility of buttons, and the timing of the AI’s turn.  Seeing what was unclear or confusing to our beta testers gave me a fresh perspective and helped me redesign the details of the UI.  *Mary Yukich*

[6]

**Product Architecture**

When it comes to the architecture of this project, there are some things we did correctly, but also some that we did not do. I believe that we started out the project strong, by playing many games, and then making a document that painstakingly went through each possible step of the game in a chronological order so we could visualize what needed to happen in the code. We decided that we needed a main file that would keep track of most of the game logic (we named this file BoardManager). However, one of the biggest mistakes we made was including way too much functionality of the game into BoardManager. Almost the entire game was in this file, which caused to be around 2000 lines long. This (paired with almost no documentation, also a bad idea) made this file extremely hard to read and navigate, hampering the speed at which we were able to debug and write code. Besides the BoardManager, we had a Player script that kept track of important information related to a player (things like score, pieces owned, etc). We also had a file named Game Info that kept track of some of the metadata of the game (such as game type, which side the player chose, and who is going first). We also had two files to keep track of the information of the two game piece types (Nodes and Branches). The AI had a script (called AI) that housed the functionality of the AI. For the networking, there was a file named NetworkController, which the BoardManager interacted with, and then the NetworkController interacted with the NetworkPlayer file, which took care of the actual passing of information over the network. All of the listed files so far are called from the BoardManager to make the game run. Besides these files, there were various files that connected the menus throughout the game, which house the logic to bring up any sort of popups or change scenes. *Andrew Mock* [7]

**Graphics/Animations**

One of the first challenges we encountered regarding graphics was finding a consultant.  We knew we wanted the graphics to have a realistic style that would fit the nature of our theme.  We asked Elizabeth Shores to work with us in part because her artistic style was what we were looking for.  When she agreed to help create our graphics, Elizabeth and Andrew and I met to discuss the basic scenes we had in mind.  Knowing that a war-related theme can be stereotypically masculine, I also asked her to work with coloring and lighting to create scenes that would help our game still appeal to a broad audience.  Elizabeth provided the majority of our graphics.  There were also a few images that we found online and adjusted to suit our needs.  I was selective about what I would use, however, because I wanted to make sure they matched the style of Elizabeth’s work as closely as possible.  We also found a package from the Unity Asset Store that included images we decided to use for most of the game’s buttons.  Other than simply attaching images to the correct game objects, there was not a great deal of work required to add graphics to our game.  But I did use several tools to make minor adjustments to some of the images.  In the earliest phases of the project, I used GIMP (<https://www.gimp.org/>) to create some mock-ups based on whiteboard drawings from a brainstorming session.  These mock-ups, though rough, helped document our vision and communicate it to Elizabeth.  I used Paint and Paint 3D throughout the project to make minor adjustments to images such as their size, coloring, and background.  A website called Kapwing (<https://www.kapwing.com/>) was also useful for removing unwanted image backgrounds.  Having a graphics consultant

[8]

was incredibly beneficial to our team because she was able to produce much

higher quality work than what any of us could have done.  One downside of not creating graphics in-house was that it sometimes made it harder to try new ideas and make adjustments on the fly.  If I had an idea for the GUI that would require additional graphics, for example, I had to rely on temporary images and my imagination while waiting for Elizabeth’s work.

Our game ended up having fewer animations than I think would have been ideal.  We used Unity’s animator only for the intro scene.  For the tutorial slides animation, Charles was able to create the desired effect with a script that used coroutines to adjust the position, rotation, and layer of the slides.  Once he got the functionality working, it was easy for me to edit his code to adjust the timing of the motion until I was satisfied with how it looked.  My biggest regret about the animations is that I did not spend more time earlier in the project learning how to use the animator.  With some more effort in that way, I think I could have added several more animations that would have enhanced the user’s experience. *Mary Yukich*

[9]

**Sound**

One of the features that seems menial but in reality, is a large part of how the user perceives the game are the sound effects & music. Sound effects raise the level of intractability for your game by making each mouse click or button toggle something that the user can hear audibly. Sound effects also can help set the stage better for your theme and can make for smoother transitions between scenes. Music similarly can raise the overall level of user experience, emphasis on “can” because even though you might have music it might be too fast, not work with your theme well, or just be really annoying and that can bring down the game play for the user.

For our game we had a music consultant make all of our sound effects and music tracks, and he made them from scratch (garage band/other music editing software). To work with Unity we had the music saved as mp3 files and loaded into a file labeled “audio”, and for using in a scene you load the mp3 file in a Game Object as an audiosource and unity has a bunch of built in functions in the editor that allow for the sound to be started and played very easily. My recommendation would be to get the sound tracks and a few if not all the sound effects by the UIP (mid-term) as it will allow for a better showing of the UI for the presentation and it gives you plenty of time to get used to working with the sound features in unity. *Bryce Wiedower*

[10]

**AI**

Early in the semester all of the AI leads found out that Node was not a solved game, and thus no perfect algorithm existed to guarantee a draw or win. With that in mind I considered three options for constructing the AI: Monty Carlo analysis, minimax (or one of its modified forms), or machine learning. When hearing the plans of the rest of the AI leads, I ultimately decided to go forward with using machine learning, and my team liked the idea as well.

When looking at machine learning I had a few options open to me. With a quick search you can find a lot of information about making and training neural networks from scratch or using some existing library/tool. I first investigated making my own, but it seemed that it may be too difficult to properly train it in integration with Unity, so I decided to use Unity’s neural network. This is used by adding the ml-agents package to Unity. Unity has a lot of information on how to use their neural network, but only one tutorial (hummingbirds). I watched, and to a small extent, followed along with the Unity tutorial to set up proper communication with the ml-agents class. The problem is that the tutorial is set up for continues input, not discreate input, which was solved by not adding the action requested and manually (in the AI class) calling request decision. I also needed to change the output from continues to discrete, and, to have double redundancy to try and prevent illegal moves, added a decision masker that prevents certain outputs from being possibilities. I had a problem with being able to test my AI due to a slow start on the game core and its testing. I was able to help on the testing side by creating a random AI to play the game. This random AI would go on to evolve into being the easy AI for the

[11]

final product. The early versions would select a random number of branches and

nodes to place and then randomly select where to place them. As I developed it, it went on to make as many moves as it could and to not do the worst possible moves. With the game core more developed and tested I was able to quickly interface the neural network coding with the game core. Once fully connected I could begin training the neural network. This was done with an associated Python3 package (ml-agents). Training helped me find many different errors within AI and my training configuration. I had one major problem that persisted to the end without being solved: I could only train for about 3 minutes at a time before having to stop and restart training. This problem caused training to take longer and get an ultimately worse AI for the final game.

Near the end of development, the AI was far from where I (and the rest of the team) wanted it to be. I first tried training it some more but could tell that it was not going to change much. Then I tried training my 10th network to see if I could get enough training in to make it good but saw that it would take too much time. Finally, I decided on overriding the neural network on the opening moves and making the best possible first move for it. This greatly increased the performance the neural network.

Looking at the AI, as a whole, it was one of the largest and most complex parts of the game. I would have like to have trained it more, but I was unable to solve that error and when I presented it to Unity, they were unable to help me solve it. I do not regret going with using a neural network but do question if using Unity’s neural network was the best decision.

*George Cook*

[12]

**Networking**

If I could go back to the beginning of the semester, there are many things that I would do differently regarding networking, and almost all of them have to do with time management. At the beginning of the semester, I volunteered to do that networking, since I had not had any experience in other aspects of the game and I had taken the networking class (a quick aside, do not be afraid to take on the networking if you have not had the networking class. There may be some very basic concepts that overlap, but from my experience working with networking in Unity was vastly different from what was done in the networking class). However, me and my team came to the consensus at the start that networking should take a backseat for a while until the Game Core was hammered out a bit. I do not recommend this approach, at least, not to the extent that our team did it where the networking was barely touched until UIP. I would recommend that to start, one should focus on learning their networking tool (we chose Photon Unity Networking, mostly because that is what most other teams decided to use). Do the included tutorials, watch tutorials on YouTube, and make smaller projects to test out small functions of the network (such as connecting to a lobby or sending moves). Also, while the Game Core is being written, advise the Game Core lead to try to write it with the networking in mind. Wherever you think there may be a need for networking code inside of the Game Core (such as sending a move) make a comment or document a general idea of what you think may need to happen. We ended up with a lot of practically duplicate code in our final product because we completely wrote the Game Core with only the AI in mind, and so we had to add several functions that did almost the exact same thing as other functions, but for

[13]

the networking, causing a lot of bloat. My next piece of advice would be that once

the networking code is being added to the main project, test as frequently as possible. Network code is a finicky thing, and the slightest change can cause many things not to work. Also, be sure not to only test in the Unity editor, but also test in an actual build of the project.

My team had a major scare, because I had the networking code fully functional, but had only tested it in the Unity editor, but lo and behold once it was put into a build it no longer would work (the cause was one client loading faster then the other, so be sure to look out for that and save yourself many hours of debugging!). Lastly, if you do not take any of this advice (and do what I did) and procrastinate way to long to star the networking, do not be afraid to ask for help, from your teammates, or even people on other teams. Capstone is a stressful class, and getting all the help you can will make your life much easier.

*Andrew Mock*

[14]

**Help**

For help we decided to have a few different ways for the user to have access to help or for help to be given to them automatically. Our main help feature was a tutorial slideshow that could be accessed from the main menu that was about 12 slides long. Each slide had a screenshot of a different aspect of the game we thought was important for the user to have a basic understanding of, and a 2 or 3 word description of that area of the game was provided in the slide as well. We felt that having a super long tutorial that had all the information about the game was too much to be covered in the tutorial slides, so we had a “help” popup that could be accessed from every scene in the game that had all the rules for the game in text and organized by category so that all the information the player needs can be easily found.

The last type of help feature we had was an indicator of who’s turn it is and if it was an opening move, instructions for how to go about preforming that move. The way our UI and AI were set up, it was a little difficult to tell when it was your turn, especially if the AI didn’t make a move so we decided to have a popup on the side of our game screen that showed which player’s turn it is. Help was the last aspect of our game that we completed because we wanted our screenshots in our tutorial to have the final graphics from our gameboard in it, along with the fact that we did not know exactly how many extra help features we needed till the end of development.

*Bryce Wiedower*

[15]

**Installation**

Based on the consistent recommendation of many teams from previous years who also used Unity, we quickly decided we would use Inno Setup to install our game.  It took me several tries to learn how to create an installer, but once I figured it out, the process was fairly simple. The first step to create an installer is to build your Unity project.  This option is found in the Unity Editor under File, then Build Settings.  You will need to specify a location in which to save the build files that are generated.  To that end, I created a new folder called Version[#] each time I made a new build of the project.  After building your project, open Inno Setup.  Inno has a script wizard that asks you to specify various settings, file names, etc., for your installer, then generates a script based on the information you provided.  When you compile that script, Inno creates an executable file which is the installer for your application.  (You may be asked if you want to save the script when closing Inno Setup.  I never did save it, because with some practice it became very easy to step through the wizard and create a new script, but that may be a helpful precaution to take.) Finally, when you run the executable produced by InnoSetup, a setup wizard will walk you through the process of installing your application.  To uninstall, find your application in the start menu, right click on it, and select “Uninstall,” just like you would for any other application.  Running the installer again would then reinstall your application.  I found that creating a new install worked best if I always uninstalled the previous version of the application first.  You do not have to delete the installer executable, however, because if you specify the same output folder each time, the new installer will simply replace the old one.   [16]

Once we had some basic functionality and navigation of our game working, I started installing it once or twice a week.

Originally, I would let the installer executable be saved to a folder called “output” under the Documents folder.  During the last two weeks of development, I installed the game much more frequently, and I began saving the installer executable in our team’s Install folder so that we could practice installing from the network share on other machines in the lab.

I think it was helpful to install the application often, for two particular reasons.  First, as we were testing more and more towards the end of development, we discovered a few bugs that surfaced only when playing the installed version, and not when we played the game in the Unity Editor.  Of course, the sooner you catch this kind of problem, the better.  Second, because I got lots of practice (almost 30 installs in all), I was able to work out some minor problems before too late in the semester.  By the time we were preparing to make our final install, I was very comfortable with the process.

*Mary Yukich*

[17]